# Introductory Guide To Operating Systems

Jocelyn O. Padallan

AP ARCLER PRESS

# Introductory Guide to Operating Systems

# INTRODUCTORY GUIDE TO OPERATING SYSTEMS

Jocelyn O. Padallan

ARCLER
PRESS

www.arclerpress.com

**Introductory Guide to Operating Systems**

*Jocelyn O. Padallan*

**Arcler Press**
224 Shoreacres Road
Burlington, ON L7L 2H2
Canada
www.arclerpress.com
Email: orders@arclereducation.com

**e-book Edition 2023**

ISBN: 978-1-77469-629-3 (e-book)

ISBN: 978-1-77469-444-2 (Hardcover)

## ABOUT THE AUTHOR

**Ms. Jocelyn O. Padallan** is a graduate of the Bachelor of Science in Computer Science and finished her Certificate of Teaching Proficiency Program at Laguna State Polytechnic University - Los Banos Laguna. She also completed her Master in Educational Management.  Currently, Ms. Jocelyn O. Padallan is an IT Professor at the Laguna State Polytechnic University - Los Banos Campus under the College of Computer Studies (CCS) where she holds different professional courses such as Effective n Business Communication and Programming Languages. She is actively involved in the college extension services programs including Life Project 4 Youth and ITeach 4 Heroes.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| APIs | application programming interfacing |
| BPAM | basic partitioned access method |
| DDoS | distributed denial of service |
| DLL | dynamic link libraries |
| DMS | document management software |
| EDF | earliest deadline first |
| FCFS | first come, first served |
| FIFO | first in, first out |
| GB | gigabytes |
| I/O | input/output |
| IAR | instruction address register |
| Inode | information node |
| IPC | interprocess communication |
| KB | kilobytes |
| LAN | local area network |
| LDE | limited direct execution |
| LRU | least-recently used |
| ME | millennium edition |
| MFD | master file directory |
| MMU | memory management unit |
| NOS | network operating system |
| NT | new technology |
| OS | operating system |
| PC | personal computer |
| PC | program counter |
| PCB | process control block |
| PDS | partitioned data set |
| PDSE | partitioned data set extended |
| PID | process identification number |
| RAM | random access memory |

| | |
|---|---|
| RDBMS | relational database management system |
| RR | round robin |
| RTOSs | real-time operating systems |
| SJF | shortest job is completed first |
| SJN | shortest-job-next |
| SMP | symmetric multiprocessing |
| TLB | translation look-aside buffer |
| VTOC | volume table of contents |

# ABSTRACT

An operating system (OS) consists of programs that regulate the implementation of application programs and serve as a channel between the client and PC hardware. The operating system manages the computer hardware systems as well as gives a structure for applications to run. A few examples referenced in the volume are Windows, Windows/NT, OS/2, and MacOS. The book presents OS as advantageous and simple to use for the client and simplifies handling client issues.

For a PC to begin running, for example, when it is organized or rebooted, it must have a primary program. This core system, or bootstrap program, will, in general, be straightforward. It launches all parts of the framework, from CPU catalogs to device regulators to memory elements. Normally, it is put in read-only memory (ROM) or digitally erasable read-only memory (EEPROM), referred by the overall term firmware, inside the PC equipment.

In multiprogramming systems, the OS determines which cycle gets the processor and the duration. This capacity is known as process planning. The volume discusses an operating system for doing these activities:

- keeps check of processor and process status of interaction;
- allocates the processor (CPU) to a function; and
- de-assigns processors whenever a cycle is not generally needed.

# PREFACE

The operating system (OS) is a connection point between a PC client and PC equipment. The working framework is a product that plays out every one of the fundamental errands, including file management, memory administration, process administration, managing input and output, and handling peripheral instruments, for example, printers and disk drives. A few well-known Operating Systems incorporate Windows Operating System, Linux Operating System, VMS, OS/400, AIX, z/OS, and so forth. The operating system functions as a point of interaction between the client and the PC equipment and controls the execution of a wide range of programs. A few key subjects captured in this volume incorporate:

- memory management;
- processor management;
- security;
- control over structural output;
- device management;
- file management;
- fault detecting aids; and
- synchronization between other programs and clients.

The operating system (OS) is the presentation of a PC system equipment abstraction, whereby individuals manage the equipment, and to utilize the resources of the PC system.

To develop the students' basic understanding, pragmatic capacity, technology, and framework plan must synchronize through programming and hardware as a technique, the PC circuit system, PC design guideline, internal framework configuration, working framework, and structural organization. Accordingly, links between programs are reinforced, and the students' PC systems analysis, design capacity and innovation.

Lately, the working framework and its abstraction systems regarding the application programming becomes more complicated. The volume discusses the impact of different components of modern operating system planning and physical operation, clients, and planning, preparation of the structural framework and other systemic programs.

This volume examines a few significant forms of operating systems, which are most usually utilized, including multi-programmed batch system and batch operating system. In multiprogramming systems, the OS determines which interaction receives the processor when and overall period. This procedure is called process booking. An operating system performs different functions, such as:

- Evaluates processor and process status;
- Allocates the processor (CPU) to a function;
- De-allots processor when an interaction is not generally needed. An Operating System oversees device correspondence through their individual drivers. The digital system is moving towards cell phones. To find out more on operating systems, clients should acquire active bit programming experience in these conditions, which are very unique in relation to conventional computers and servers.

The functioning of the multi-systemic batched system includes roles being assembled empowering the CPU to execute each occupation in turn permitting the CPU to use its resources. Roles are sustained in the operating structure each in turn. For the role to be done, a program is completed that uses I/O activity. The client cannot become inactive while utilizing a multi-programming framework, as the working framework will change to another system.

The batched computer system keeps various tasks in the memory as it conducts a given assignment until it is completely executed. It follows a sequential process. The memory is freed once a given errand is finished and the work yield is moved to a result spoil, permitting it to be handled or printed later. The batch activity framework restricts client communication. The client is free once the framework takes assignments from the batch system.

CHAPTER 1

# AN OVERVIEW ON OPERATING SYSTEM

## CONTENTS

An operating system (OS) is a component of system software that controls computer hardware and software whilst also offering common functions to programs.

Activities are planned, almost scheduled, in time-sharing OS to increase the systems efficiency. There is software that are accountable for the distribution of processing time, mass storage, printing, and other services and could or could not be included (Yin et al., 2016).

Despite the fact that application code is normally executed directly by the hardware and frequently communicates with an OS function or is disrupted by it, the OS serves as a bridge between programs and computer hardware for hardware functions such as input and output and memory utilization. From cellphones and game consoles to web servers and supercomputers, OSs are available on a diverse variety of computer-based devices (Figure 1.1).

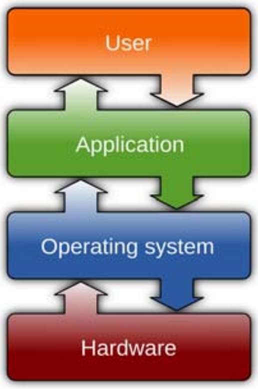


**Figure 1.1.** Overview of OP.

*Source: https://en.wikipedia.org/wiki/Operating_system.*

The following are some of important functions of an OS:

- Memory management;
- Device management;
- File management;
- Processor management;
- Security;
- Control over system performance;

- Job accounting;
- Error detecting aids;
- Coordination between other software and users.

## 1.1. MEMORY MANAGEMENT

Memory management is a sort of resource management that is used to keep track of how much memory a computer has. Memory management must contain techniques for efficiently allocating memory to programs on request and releasing it for use when it is no longer needed. This is critical in any computer system with numerous processes executing at the same time (Quigley et al., 2009).

There have been several ways explored to improve memory management efficacy. Virtual memory systems separate a process's memory addresses from its physical addresses, allowing it to separate processes and increase the virtual address space beyond the amount of RAM available by paging or shifting to auxiliary storage.

The performance of the virtual memory manager can have a big impact on overall system efficiency and productivity. In some OSs, such as OS/360 and successors, the OS manages memory. Other OSs, such as Unix-like OSs, handle memory at the application level.

There are two types of memory management in an address space: manual memory management and automatic memory management. Locating a sufficient-sized block of unoccupied memory is required to process an allocation request. Memory demands are fulfilled by allocating pieces of memory from the heap or free store, which is a large pool of memory (Whipple et al., 2009). At any one time, some parts of the heap are in use, while others are "free" (unoccupied) and hence available for subsequent allocations.

External fragmentation, which results from multiple small gaps between issued memory blocks, making their use for an allocation request invalid, is one of the obstacles to implementation. The information of the allocator can also exaggerate the size of (relatively) small allocations. Chunking is a popular method for accomplishing this. The memory management system must keep track of memory allocations to ensure that they never overlap and that no memory is ever "lost" (memory leaks) (Figure 1.2).

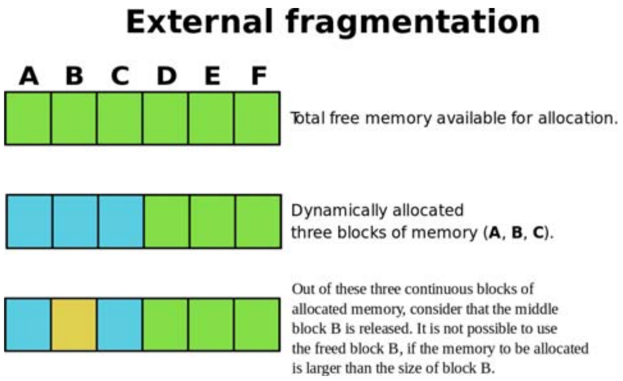

External fragmentation
A B C D E F
Total free memory available for allocation.
Dynamically allocated three blocks of memory (A, B, C).
Out of these three continuous blocks of allocated memory, consider that the middle block B is released. It is not possible to use the freed block B, if the memory to be allocated is larger than the size of block B.

**Figure 1.2.** An illustration of external fragmentation.

*Source: https://en.wikipedia.org/wiki/Memory_management#/media/File:External_Fragmentation.svg.*

## 1.2. DEVICE MANAGEMENT

Device management refers to the control of input/output (I/O) devices including microphones, keyboards, printers, magnetic tape, USB ports, camcorders, projectors, and other accessories, as well as accompanying units such support units control channels, in an OS. A process may necessitate a range of resources, including main memory, file access, and access to a disk drive, among others. Control can be transferred to the CPU if resources are available (Wentzlaff et al., 2010). Alternatively, the procedure would have to be postponed until enough resources were available. The OS needs a distinct application designated as an ad device controller to handle the numerous devices on the system, whether actual or virtual. It also determines whether or not the desired device is present (Figure 1.3).

The principles of I/O devices can be classified into three types:

- **Boot Device:** It organizes data into fixed-size blocks, with every single one of them having its own unique address. Disks, for example.
- **Character Device:** It sends or receives a continuous stream of characters, none of which may be addressed individually. For example, keyboards, printers, and so on.
- **Network Device:** It is used for data packet transmission.

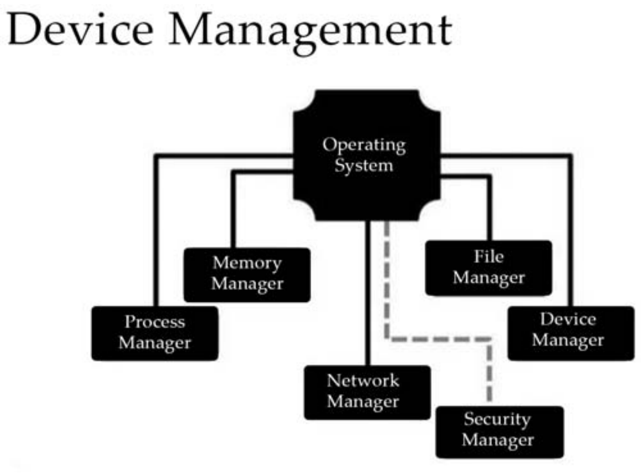


**Figure 1.3.** Device management.

*Source: https://zitoc.com/device-management/.*

The communication with the devices is handled by the OS via their drivers. The Operation Systems elements offers a harmonious interface for accessing devices with varying physical characteristics (Tsolakis et al., 2019). OS peripheral devices are classified into three types: dedicated, shared, and virtual.

The following are some device management features:

- The OS links up with device controllers through device drivers when directing the device to the various processes running on the system;
- Device drivers are also system software applications that connect processes and device controllers;
- Another critical purpose of the device management function is to implement the API;
- Device drivers are software programs which enable an OS to properly regulate the operation of multiple devices;
- The device controller, which is utilized in device management operations, consists primarily of three types of registers: command, status, and data.

## 1.3. FILE MANAGEMENT

To maintain files, a file management system is employed (or management). It's a type of computer application that manages data files in a system.

A file management system has limited functionality and is designed to manage individual or group files, such as specific office papers and records. It may show information about the report's owner, production date, completion status, and other features that are relevant in a workplace environment.

A file management system is often known as a file manager. Each computer's data is kept in a complex hierarchical file system with directories and subdirectories beneath them.

These folders are where files are saved, and they usually follow the hierarchical arrangements prescribed by a program's instructions.

However, many other data, such as images, videos, and documents, are organized by the user at his or her leisure. A file management system is essentially the software that is used to organize, move, and operate with these files (Tang et al., 2010). In reality, file management systems are concerned with how files are structured as opposed to just how they are saved.

The tracking component of a file management system is critical to the construction and maintenance of this system, in which documents in different phases of processing are distributed and interchanged on a continuous basis. It comprises of a simple interface that displays stored files.

It enables the user to explore, transfer, and sort them based on several criteria such as date of last modification, date of creation, file type/format, size, and so on (Figure 1.4).

The system may include elements such as:

- The procedure of assigning queued document numbers for processing;
- Ownership and process mapping are used to track the many phases of processing;
- Report generation;
- Notes;
- Status;
- Create, modify, move, copy, and delete files, as well as do other file operations;
- Basic metadata can be added or edited.

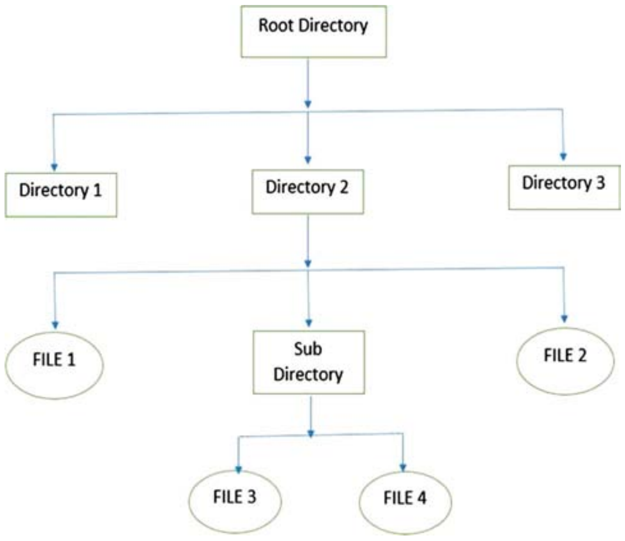


**Figure 1.4.** The storage structure in an operating system. The root directory is at the head of the hierarchical system in this illustration. It includes all of the sub-directories in which the files are kept. A subdirectory is a directory that resides within another directory in the file storage system. The directory-based storage program ensures better file organization in the memory of the computer system.

*Source: https://princeabhishek410.medium.com/understanding-file-management-system-in-operating-system-4c7fbfc306f2.*

Advanced file management systems, such as document management software (DMS), may be able to do additional tasks, such as arranging important documents. To establish a searchable database for faster retrieval, files are tagged or indexed depending on their properties (Shin et al., 2014).

A file management system is not the same as a file system, which stores all types of data and files in an OS, or a database management system, that includes relational database features and a scripting language for data processing.

## 1.4. PROCESSOR MANAGEMENT

### 1.4.1. Process

A process is a job that falls under the category of Execution. Process is often referred to as the Running Job. A system call must be made that calls the Processor or CPU to perform any operation for execution. The process comprises reading information from a file, writing data from a file, and printing a page, which means that any type of operation is known as a process. Every process has some characteristics, such as:

- **Process ID or Identification Number:** The CPU assigns a process id when we request an operation. When we request a service, we are given a process id, which is also referred to as a unique identifying number. As we all know, there are numerous types of operations that may be conducted on a computer. As a result, for identification, which Process will be executed (Singh, 2014).
- **Name of the Process:** Identify the process description. The name of the operation carried out by the Process. For instance, move the mouse cursor, click on My Documents, open a file, and so on.
- **Process Status (Ready, Active, Wait, or Suspend):** The process has some states, state specify the process state implies whether a process is operating or not, whether a process waits for CPU, and so on.

States are classified into three types.

- **Ready:** When we have completed all of the Inputs and Outputs, the process will enter the Ready State. We expect the Execution after providing the input. Following the end of user interaction, which includes all inputs and outputs.
- **Active:** A process that is active is one that is executing on the CPU.
- **Wait:** When a process is waiting for user input and output, it is said to be in the wait state.

Process state will provide you with information about the process's status.

### 1.4.2. Process Resources

Many resources are employed when running a process. To input information for instance, we have to use the keyboard, and to send data to the computer, we must use the CPU. So, what types of resources are required to carry out any operation? The term "system resources" refers to all of the devices that are connected to the computer (Sangorrin et al., 2010).

### *1.4.2.1. Scheduling Information*

Scheduling is utilized when multiple processes are executing at the same time in which the following processes will be executed by the CPU. So, we use scheduling to calculate CPU time, which means CPU time is divided among several processes (Figure 1.5).

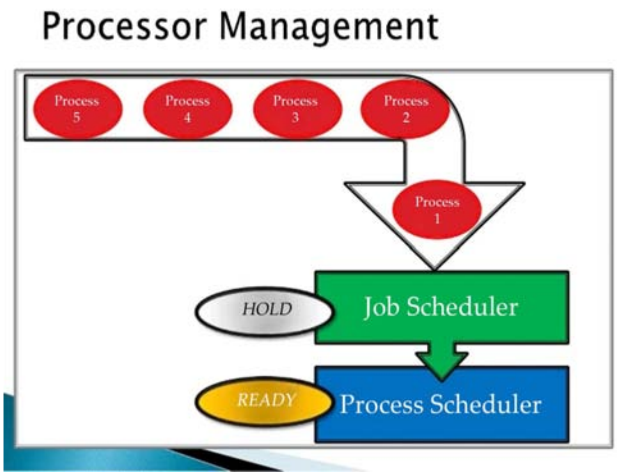


**Figure 1.5.** Processor management steps.

*Source: https://operatingsystemsam.wordpress.com/processor-management/.*

## 1.5. SECURITY

All computer system and software architecture have to address all security issues and implement the necessary security controls. At the same time, achieving a balance is critical because stringent security measures can increase costs while lowering the system's usability, utility, and efficiency. As a result, system designers must guarantee that performance is maximized while security is maintained.

OS security is the process of ensuring an OS's availability, confidentiality, and integrity. OS security refers to the strategies and tactics used to protect the OS against dangers like viruses, worms, malware, and remote hacker intrusions (Sharma et al., 2012). Any preventive-control mechanisms that secure any system assets that could be taken, modified, or erased if OS security is compromised are included in OS security (Figure 1.6).

**Figure 1.6.** OS security.

*Source: https://www.javatpoint.com/operating-system-security.*

Security alludes to the protection of computer system resources like software, CPU, memory, storage, and so on. It is equipped to safeguard against any risks, like viruses and unauthorized access. It can be enforced by maintaining the integrity, confidentiality, and reliability of the OS. If an unauthorized person starts a computer application, the system or data stored on it may be severely harmed.

Two transgressions can jeopardize system security, and they are as follows:

- **Danger:** A program that has the potential to gravely harm the system.
- **Assault:** A security breach that permits unauthorized access to a resource.

There are two kinds of security breaches that might cause problems for the system: purposeful and unintentional. Malicious threats are types of damaging computer code or web script that are designed to cause harm.

The following are some of an OS's other key functions:

- **Security:** It uses passwords and other security mechanisms to prevent unwanted access to programs and information (Arshad et al., 2018);
- **Monitoring System Performance:** This by keeping track of how long it takes to request a service and get a response from the system;

- **Job Accounting:** It is the practice of keeping track of how much time and resources certain occupations and users spend on them;
- **Debugging and Error-Detection Tools:** create dumps, traces, error messages, and other debugging and error-detection tools;
- **Integration with Other Software and Users:** Compilers, interpreters, assemblers, and other software are coordinated and assigned to computer system users.

## 1.6. A BRIEF HISTORY OF OPERATING SYSTEMS (OSS)

OSs were not available on the early computers. Single programs that operated off of these early computers had to contain every single code necessary to function on the computer, interface with the associated hardware, and do the computations that the program was assigned for. Even the simplest programs become extremely complicated due to this circumstance (Shaw et al., 2016).

As a result of this dilemma, the proprietors of the first computers began to design system software that helped in the authoring and execution of the computer's programs, giving birth to the first OSs. In 1956, GM created the first OS to run a single IBM central computer. The International Business Machines Corporation was the pioneering computer firm to take on the duty of designing OSs and to supply OSs with its machines in the 60s.

In the 1950s, when computers could only run a single program at a go, the first OSs were created. Later in the following decades, computers began to integrate an increasing number of software packages, what is now known as libraries, which combined to provide the foundation for today's OSs.

The original version of the Unix OS was developed in the 60s. Written in the C programming language and at first provided for free. Unix adapted swiftly to the new platforms and gained widespread appeal.

A number of contemporary OSs, like the Apple OS X and several Linux variants, are based on or date from the Unix OS. Microsoft Windows was developed to satisfy IBM's need for an OS to power its personal computer (PC) line. Microsoft's initial OS was called MS-DOS, and it was released in 1981 after the company purchased the 86-DOS OS from Seattle Computer Products and tweaked it to fit IBM's specifications (Silva et al., 2006).

A graphical user interface was fitted with MS-DOS in 1985, and created what is now known as Windows. Apple, OS X, Microsoft Windows, and different kinds of Linux (including Android) now control the great bulk of the current OS market. Most software packages are designed to run with a single company's OS, such as just Windows or only macOS (Figure 1.7).

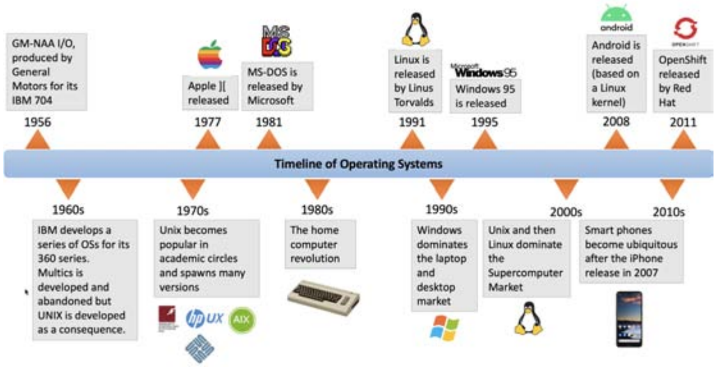

GM-NAA I/O, produced by General Motors for its IBM 704
1956
Apple ][ released
1977
MS-DOS is released by Microsoft
1981
Linux is released by Linus Torvalds
1991
Windows 95
Windows 95 is released
1995
android
Android is released (based on a Linux kernel)
2008
OpenShift released by Red Hat
2011
Timeline of Operating Systems
1960s
IBM develops a series of OSs for its 360 series. Multics is developed and abandoned but UNIX is developed as a consequence.
1970s
Unix becomes popular in academic circles and spawns many versions
hp UX
AIX
1980s
The home computer revolution
1990s
Windows dominates the laptop and desktop market
2000s
Unix and then Linux dominate the Supercomputer Market
2010s
Smart phones become ubiquitous after the iPhone release in 2007

**Figure 1.7.** Timeline of OS.

*Source: https://www.orb-data.com/the-history-of-the-operating-system-from-paper-tape-to-openshift/.*

A piece of software will clearly state which OSs it offers, and if necessary, will be quite detailed about which versions of those OSs it supports. For example, a video production software package may state that it is compatible with Windows 10, Windows 8, and Windows 7, but not with prior Windows versions such as Vista and XP.

Additional versions of software that run with different OSs or versions are routinely released by software developers.

It's also crucial to understand whether your OS is 32-bit or 64-bit. When you download software, companies frequently ask you this question.

## 1.7. OPERATING SYSTEM (OS) TYPES

OSs have existed since the dawn of computing, and they continue to evolve throughout time. In this chapter, we'll go through some of the most often utilized OSs.

### 1.7.1. Batch Operating System (OS)

A batch operating is designed for a specific purpose and does not have any of its processes controlled directly by the user. Each user presents a designated task to the computer operator through an off-line means like with punch cards (Sjöstrand et al., 2015). Jobs with similar needs are batched together and run as a group to hasten processing. The programmers give their programs to the operator, who organizes the programs into batches based on their requirements.

Below are some issues with batch systems:

- Interaction between the user and the work is lacking;
- Because mechanical I/O devices are slower than the CPU; the CPU is often idle;
- It's difficult to give the required priority.

## 1.7.2. Time-Sharing Operating Systems (OSs)

Time-sharing is a technique of letting multiple users to use a computer system at the same time from different stations or terminals. Multitasking or time-sharing is a natural variation of multiprogramming. Time-sharing is the name derived from the usage of a processor's time by more than one user at the same time. The fundamental difference between multiprogrammed batch systems and time-sharing systems is that the goal of multiprogrammed batch systems is to increase processor utilization, whereas the goal of time-sharing systems is to lower response time (Santos et al., 2013). The CPU changes between multiple jobs to complete them, but the shifts are common. As a consequence, the user can expect a fast response (Figure 1.8).

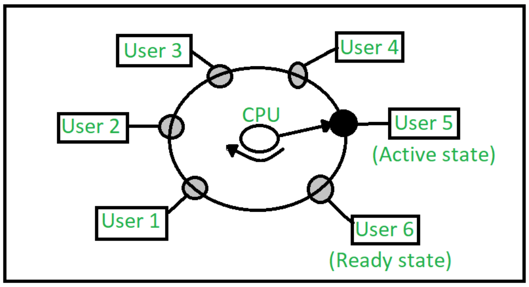


**Figure 1.8.** The User 5 is active state but User 1, User 2, User 3, and User 4 are in waiting state whereas User 6 is in ready state.

*Source: https://www.geeksforgeeks.org/time-sharing-operating-system/.*

In transaction processing, for instance, the processor executes every user application in fast and powerful quantum computations. That is, if a certain number of users are active, each of them can obtain a time quantum. When a user gives a command, the response time is only a few seconds (Peter et al., 2015). The OS makes advantage of CPU scheduling and multiprogramming to allocate a tiny amount of time to each user. Computer systems that were initially created as batch systems have now been converted to time-sharing systems.

## 1.7.3. Distributed Operating System (OS)

In distributed systems, numerous central processors are employed to take many real-time applications and consumers. As required, jobs for data processing are divided across the processors.

The processors connect with one another via numerous communication links (such as high-speed buses or telephone lines). These types of systems are referred to as loosely coupled systems or dispersed systems. In a distributed system, the size and function of processors may differ. These processors are referred as sites, nodes, computers, and other words.

## 1.7.4. Network Operating System (NOS)

A network operating system (NOS) operates on a server and allows it to deal with data, users, groups, security, applications, and other networking operations. The fundamental function of a NOS is to provide shared file and printer access across numerous computers in a network, which is often a local area network (LAN), a private network, or another network. Microsoft Windows, UNIX, Linux, and Mac OS X are examples of NOS (Figure 1.9).

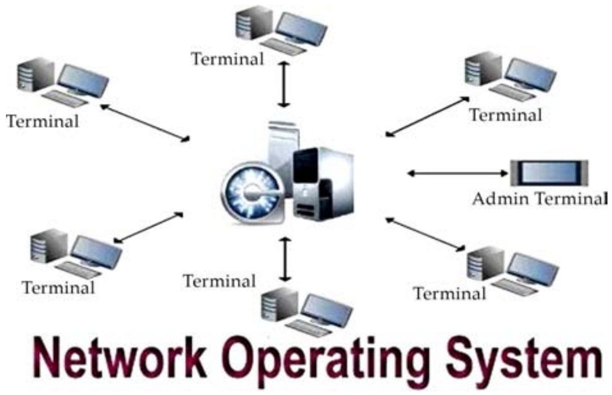


**Figure 1.9.** Diagram of network operating system.

*Source: https://digitalthinkerhelp.com/network-operating-system-nos-tutorial-examples-and-types/.*

CHAPTER 2

# TYPES OF OPERATING SYSTEMS

## CONTENTS

## 2.1. BATCH OPERATING SYSTEM (OS)

The batch operating system (OS) is very different from other kinds of OSs in that the computer and OS do not interact directly. The system makes use of an operator that takes similar jobs with the same requirement and groups them into batches. The operator, therefore, has the task of grouping jobs with similar needs. For users who utilize this OS, they do not directly interact with the computer (Ow, 2011). When using the software, the user prepares their work in an off-line device such as a punch card. They submit the work to the computer operator, which sorts the task. Jobs with similar needs are batched together and run as a group so as to speed up processing. When the programmer exits the programs with the operator then sorts the programs according to their similarities (Figure 2.1).

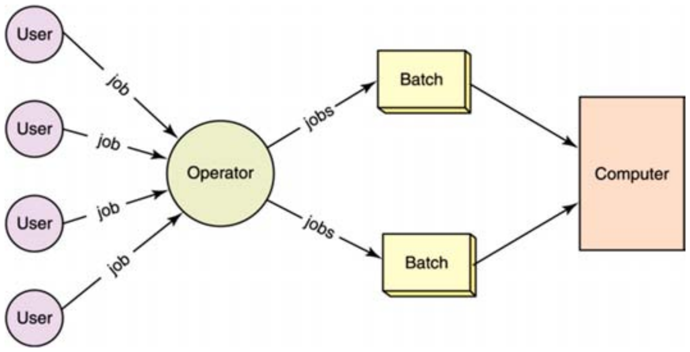


**Figure 2.1.** There are various kinds of batch operating system.

*Source: https://padakuu.com/batch-operating-systems-27-article.*

This kind of OS was very popular in the 1970s as most tasks were executed in batches. During that period, the commonly used computers were the mainframe. Jobs with similar functions were grouped together by the batch OS. After grouping, the jobs were treated as batches and were simultaneously executed. Some of the batch activities performed by the OS are as follows.

As a single unit, a job I'd composed of a preset sequence of programs, data, and commands. In the computer, processing is done in the order that they are received; it works on the first come first served basis. The jobs are then stored in the memory and are executed later on without the need for manual information. After the job is completed successfully, the OS releases its memory (Nimodia and Deshmukh, 2012).

## 2.2. TYPES OF BATCH OPERATING SYSTEM (OS)

Under batching OS, there are two main branches namely the simple batched system and the multi-programmed batched system.

### 2.2.1. Simple Batched System

When using the simple batched system, the use does not interact with the computer system directly when executing a task. When using this system, the user is required to prepare a job that includes the control information, program, and data on the nature of the job. The information is placed in the control cards. The job is then submitted to the computer operator. The computer operator is in the form of a punch card. From the processing, an output is generated which includes registers and results. When there was an error during the execution of the program, a memory dump is the output of the process. The output was produced after minutes, hours or days (Figure 2.2).

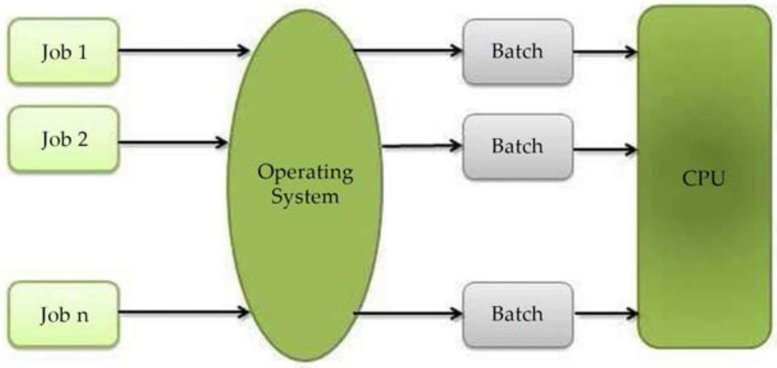


**Figure 2.2.** A simple batch operating system.

*Source: https://www.tutorialspoint.com/operating_system/os_properties.htm.*

The simple batch system works by transferring control from one job to another. The process mostly involved jobs with similar requirements being pooled together and the processor processing tasks. This makes processing speed highly important (Nollet et al., 2004). Operators out programs with similar needs into batches. When the batches are available, the computer runs them one at a time. The sequence of jobs is read by the system. Each job sequence usually Hass a control card and a predefined job which avails them.

## 2.2.2. Multi-Programmed Batched System

This system deals with jobs that have already been read and have not even run on a disk. A disk usually contains a pool of jobs and allows the OS to decide on what job it should run next to maximize COU utilization. Jobs that come on cards or magnetic tapes cannot be run in a different order. The jobs are executed simultaneously and are run on a first come first-served basis. By storing jobs in a direct access device, this makes job scheduling a in a disk very possible. An important feature of job scheduling is multi-programming. There are limitations in the overlapped I/O, offline, and spooling operations. A single user cannot maintain all the input and output devices making CPU busy at all times (Figure 2.3).

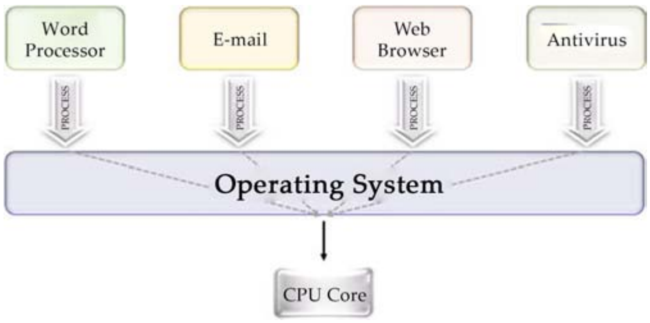


**Figure 2.3.** A multi-programmed batched operating system.

*Source: https://www.itrelease.com/2017/09/advantages-disadvantages-multi-programming-systems/.*

The working of the multi-programmed batched system involves jobs being grouped enabling the CPU to execute one job at a time allowing the CPU to utilize its resources. Jobs are maintained in the OS one at a time. For the job to be done, a task is completed that included mounting a tape on an I/O operation. The user cannot sit idle when using a multi-programming system as the OS will switch to another task. As another job is being completed, the other jobs remain in wait state and the CPU is returned (Moore and Stouch, 2016).

## 2.3. WORKING OF THE BATCHED OPERATING SYSTEM (OS)

The batched OS keeps a number of jobs in the memory as it performs a given task until it is fully executed. It follows the first come, first served (FCFS) manner. Jobs are grouped into sets known as batches. The memory is freed once a given task is completed and the work output is transferred to an output spoil allowing it to be processed or printed later. The batch operation system limits user interaction. The user is free once the system takes tasks from it. Batch system is of 10 used to update data related to records or transactions.

## 2.4. ADVANTAGES OF BATCH SYSTEM

Some of the advantages of batch processing system are as follows:

- The batch systems allow several users to make use of the same system making the system economical for several users in a given organization;
- In the batch system, the processors are able to identify how long it would take for a task to be completed;
- The batch system has a small idle time. Batch systems are very easy to manage and work repeatedly.

## 2.5. DISADVANTAGES OF BATCH SYSTEM

Some of the disadvantages of batch system include the fact that the computer operators work well when they are compatible with the batch system. The batch systems are known to be hard to debug. The systems are very expensive. If the task being run fails, other jobs will have to wait for an unknown duration before they are taken up (Monaco et al., 2013). Bank stats and payroll systems are examples of batch-based OSs.

## 2.6. TIME-SHARING OPERATING SYSTEM (OS)

The time-sharing OS is an extension of logical programming. It was developed to deal with issues noted in multi-programmed batched systems. Though the multi-programmed batched systems provided an environment where system resources could function properly, it presented a challenge with regards to user interaction. In the time-sharing OS, the CPU performs many tasks by switches and allows the user to interact with each program as it is being run (McClean et al., 2013). In this system, the CPU tends to operate at a faster compared to other peripheral devices such as printers and video display terminals. The high operation speed enables the CPU have adequate time to solve discrete problems during the input/output (I/O) process (Figure 2.4).

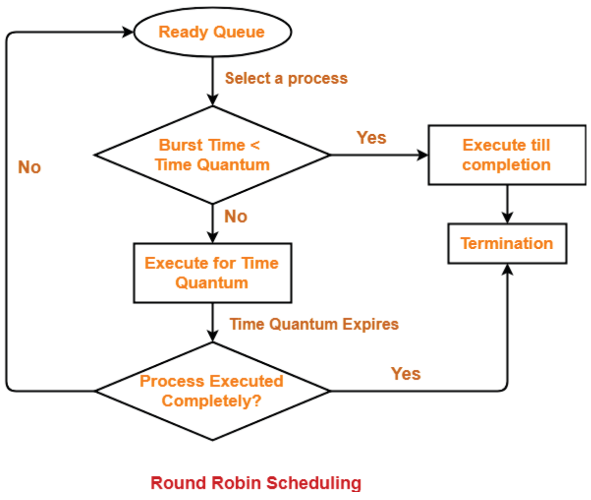


**Figure 2.4.** Time-sharing operating system is an extension of logical programming.

*Source: https://www.gatevidyalay.com/round-robin-round-robin-scheduling-examples/*

When using the time-sharing OS, the CPU addresses the problems one at a time. However, access and retrieval from time-sharing systems is considered instantaneous from the standpoint of remote terminals. This is because the solutions are availed to them after the problem has been completely. In the early 1960's and late 1950's, the time-sharing OS was developed to enable efficient use of processor time. There are various kinds of time-sharing OS techniques that include multi-programming, parallel operation, and multiprocessing. There are a number of computer networks organized with the goal of exchanging resources and data and are all centered on time-sharing systems. A good thing about time shared OS is that it enables several users to use the computers simultaneously (Manzalini and Crespi, 2016). Each action or order becomes smaller at a time in the time-shared OS. This gives the user little time when using the CPU. The system rapidly switches from one user to another. This makes the user feel as though the entire computer system is dedicated to its use though there are multiple users using it.

To ensure optimum functioning, the time-shared OS make use of multi-programming and CPU scheduling so that each is given a small portion of a shared computer at once. Each of the users gets a separate program in memory. The program is loaded on to a memory and is executed. The program performs a short period of time before or when it is about to complete I/O. The term time slice is used to refer to the short period of time during which users get attention of CPU. Other names include quantum or time slot. The time slice is usually of the order of 10 to 100 milliseconds. Compared to multi-programmed OSs, the time-shared OSs are highly complex. In the time-shared OS, multiple jobs are kept in the memory simultaneously. This enables proper memory management and security. Jobs are swapped in and out of the disk from the main memory to enable the system have a good response time. When the jobs are swapped, the disk serves as a backing store for the main memory. This goal can also be achieved when a virtual memory is used. This technique enables the execution of a job that may not be completed in memory (Figures 2.5 and 2.6).

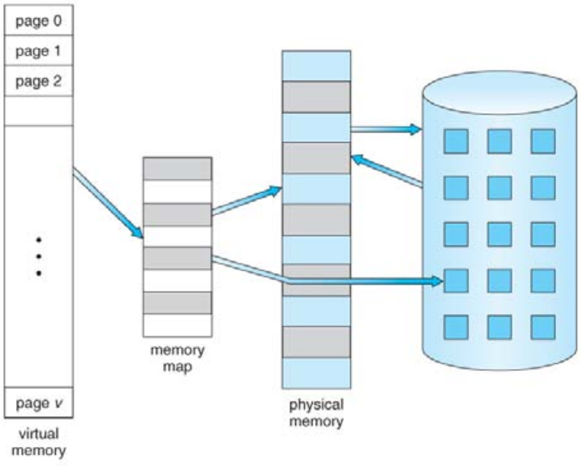


**Figure 2.5.** Time sharing operating systems may need virtual memory.

*Source: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html*

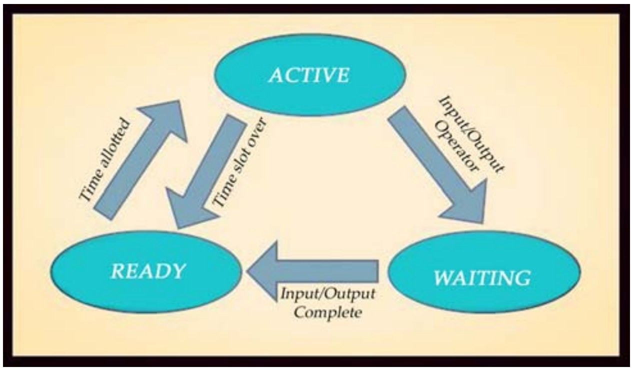


**Figure 2.6.** States of time-sharing operating system.

*Source: https://www.itrelease.com/2014/08/advantages-disadvantages-time-sharing-operating-system/.*

In the shown figure, there are different users all in different states. There's re three main states namely the active state, ready state and waiting state. In the active state, the CPU controls the user's program. In the active state, there is only one program that is available. In the ready state, the user's program is ready to execute but it has to wait its turn to get the CPU. Such a state is able to accommodate several users. The final state is the waiting state where the user's program waits for input or output operation. The waiting state can also accommodate more than one user at a time (Muehlstein et al., 2017).

## 2.7. REQUIREMENTS OF THE TIME-SHARING OPERATING SYSTEM (OS)

There are somethings that are essential in ensuring the time-sharing OS works well. They include an alarm clock mechanism. This mechanism sends an interrupt signal to the COU after every time slice. There is also the protection mechanism which prevents the instructions and data from one job from interfering with others.

Example of time-sharing OS is the transaction processing system. In the transaction processing system, there are different types of processors that have the ability to execute each user program in quantum or small burst computation. For instance, if there are n users then every user can get a time quantum. Examples of the time-sharing OS include the TOPS-20, TOOS-19, Windows NT server, Windows 2000 server, Linux, Multics, and UNIX (Mayoral et al., 2017). Among the features of time-sharing OS is that each user is able to grab a dedicated time for all operations, there are multiple online users that can use the same computer at the same time, better interaction between computers and users, the user's request can be made in small time response, it does not need longer time to wait for the task to end to the processor and it has the ability to make quick processing with lots of tasks.

## 2.8. ADVANTAGES OF TIME-SHARING OPERATING SYSTEM (OS)

Some of the advantages of time-sharing OS are as follows:

- Each of the tasks to be done is given equal opportunity to be performed;
- There are reduced cases of duplication of software;
- There is reduced idle time in the CPU.

## 2.9. DISADVANTAGES OF TIME-SHARING OPERATING SYSTEM (OS)

Some of the disadvantages of time-sharing OS are as follows:

- One is required to take security and integrity of the user's data and programs;
- There are problems in data communication;
- There is a problem with regards to reliability.

## 2.10. DISTRIBUTED OPERATING SYSTEM (OS)

The distributed OS is a system software made by the collection of physically separate, networked, independent, and communicating computational nodes. This kind of system handles jobs that have been services by multiple CPUs. It is considered to be the most important type of OS. One individual node holds a specific subset of the global aggregate OS. The given subset is a composite of two distinct service providers. Of the distinct service providers, the first one is a ubiquitous minimal kernel or microkernel. The kernel and microkernel directly control the node's hardware. The second encompasses the level collection of system management components (Lin and Ye, 2009). These components coordinate the node's collaborative and individual activities. These components therefore abstract microkernel functions and also support user applications. The management and microkernel components collection work together. The collaboration is useful in supporting the goal of the system of integrating multiple processing and resources functionality into a stable and efficient system. The integration of individual nodes seems to be seamless. The integration of individual nodes is done into a global system. For this reason, the integration process is called transparency. The single system image is useful in describing the illusion of a single computational entity to users of the global system's appearance (Figure 2.7).

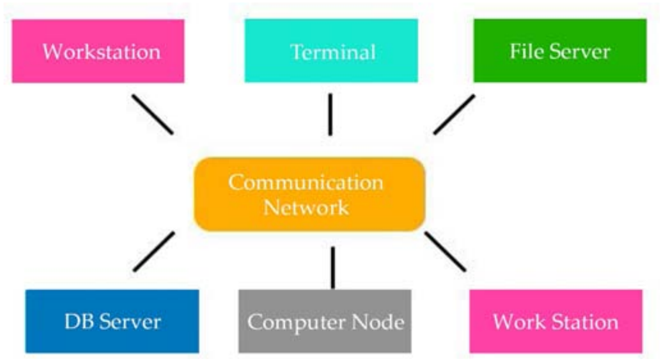


**Figure 2.7.** The distributed operating system handles multiple tasks.

*Source: https://teachcomputerscience.com/distributed-operating-system/.*

The distributed OS can be described as an OS with the ability to provide all the needed essential services and functionality needed by users. It also adds to the attributes and particular configurations that are needed to enable it support other requirements including availability and increased scale. The distributed OS and the single-node monolithic OS bear several similarities including the fact that even though it is made up of several nodes, the user views it as a single-node (Levis et al., 2005). The separation of policy and mechanism is achieved when there is a separation of minimal system-level functionality from additional user-level modular services. Mechanism can easily be interpreted as "What something is done" while policy is interpreted as "how something is done." There is a notable increase in scalability and flexibility when there is separation.

## 2.11. DISTRIBUTED COMPUTING MODELS

There are three basic distributions in the distribution OS. The three basic distributions can better be illustrated by examining three system architectures namely the centralized, decentralized, and distributed. In the examination, three structural aspects are considered. They include organization, connection, and control. The physical arrangement characteristics of a system is the organizational part. The communication pathways among the nodes are covered by connection. The operations of the earlier two considerations are managed by control.

### 2.11.1. Organization

In a centralized system, there is one level of structure and from it all constituent elements are directly depended on a single control element. A hierarchical system is illustrated in the decentralized system. The subsets of a system's entities are united in the bottom level. Ultimately, the entity subsets combine at higher levels and finally culminate at a central master element. The distributed OS is made of a collection of autonomous elements with no concept of levels.

### 2.11.2. Connection

The central master entity and centralized systems are directly connected in a hub and spoke fashion. This is quite different in a decentralized system as it makes use of both direct and indirect paths between the central entity and constituent elements. It is often at times configured as a hierarchy with only the shortest path between any two elements (Lass and Gronau, 2020). The distributed OS may function in the absence of a given pattern as there are both direct and indirect connections between any two given elements. This can be explained by the phenomena of the string art or Spirograph drawing as a fully connected system mostly used in the 1970s.

### 2.11.3. Control

Both centralized and decentralized systems have directed flows of connection that run to and from the central entity. In distributed systems, communication occurs along arbitrary paths. This forms the pivotal notion of the third consideration. Control mostly deals with the allocation of data and tasks to system elements thereby balancing complexity, responsiveness, and efficiency. Decentralized system offers users more control and also potentially easing administration by limiting options. There is a notable difficulty in controlling distributed OSs but have proven to scale better horizontally and offer fewer points of system wide failure. In this kind of system, the associations deal with the needs imposed by the designs. However, it does not deal with problems caused by organizational chaos.

## 2.12. DESIGN CONSIDERATION OF DISTRIBUTED OPERATING SYSTEM (OS)

### 2.12.1. Transparency

Among the elements taken into consideration in the design of the distributed OS is transparency. Transparency is also known as the single-system image. It refers to the ability of an application to handle an OS with little regards as to whether it is distributed or without regards to implementation or hardware details. Transparency benefits several areas of a system including migration, naming, performance, location, and access. Transparency affects decision making directly therefore, it generally affects the design of the distributed OS (Lange et al., 2011).

### 2.12.2. Inter-Process Communication

The inter-process communication involves the implementation of general dataflow, process interaction and communication between threads between or within nodes in a distributed OS. A low-level IPC design is achieved through intra-node and inter-node communication. This is a typical approach when implementing communication functions supporting transparency. For this reason, the inter-process communication is considered to be the greatest underling concept of the low-level design consideration of the distributed OS.

### 2.12.3. Process Management

When dealing with distributed processes, there is great need for proper distribution of resources. This is achieved through process management. This kind of management lays down mechanism and policies that could be used in effective and efficient sharing of resources. The mechanisms and policies support various operations that involve the allocation and de-allocation of processes ad ports to processors (Krohn and Tromer, 2009). The also give the system the mechanism to halt, migrate, suspend, run, and resume process execution. The operations and resources can either be remote or local with respect to each other. This enables the distributed OS to maintain its state and synchronize the overall process in the system.

### 2.12.4. Resource Management

Computer systems are made of a variety of resources including devices, files, and memory. These resources are distributed throughout the system. This makes the nodes light to idle workloads. Policy-oriented decisions influence load balancing and load sharing. Decisions can be made with the aid of several algorithms. However, there is need for a second level decision making policy that will decide on the most suitable algorithm for the given problem. In the distributed OS, the processors communicate with each other through the use of various communication lines. These communication lines resemble high-speed buses or telephone lines. Processors are known as distributed systems or loosely coupled systems. The processors of the system may vary in function and size.

## 2.13. ADVANTAGES OF DISTRIBUTIVE OPERATING SYSTEM (OS)

Some of the advantages of distributive OS includes the fact that a user at one site is able to use resources at another end as a result of the resource sharing facility. The other advantage is that the system speeds up data exchange with one another through electronic mail. Also, when there is a failure at one site in the distributed system, other sites will not be affected and can therefore remain functional during the operation process (Kushwaha and Kushwaha, 2011). The system also offers better services to the customers. The distributive OS has a notable reduction in the load of the host computer. There are minimal cases of delays in data processing when using these systems. There is more performance compared to single systems. In distributive OS, there is great ease in the addition of resources.

## 2.14. DISADVANTAGE OF DISTRIBUTIVE OPERATING SYSTEMS (OSS)

Some of the disadvantages of distributive OS include the problem of security due to sharing of systems. Another challenge is that there is a possibility that the messages could be lost in the network system. The databases in the network operating may present a challenge in administration compared to a single user system. The bandwidth may present a challenge when there are large amounts of data is being handled. In such a case, the network wires need to be replaced with those that are expansive. Another challenge of the distributive OS is overloading. This may be very common. There are cases of low performance when the database in connected to a local system that can be accessed by several users remotely or though the distributed way.

Some of the examples of the distributed OSs include the Linux, Ubuntu, Windows Server 2012, Windows Server 2008, and Windows Server 2003.

## 2.15. NETWORK OPERATING SYSTEM (NOS)

The network operating system (NOS) is defined as a specialized OS for network devices such as firewall, switch, or router. NOS are those that possess networking capabilities. Historically, these systems were fitted in personal computers (PCs) allowing them to participate in printer access, shared file, and computer networks in a local area network (LAN). Though this description of an OS in mostly historical, some of the common OSs include the network stack used in supporting a client-server model (Jeong et al., 2012). The NOS were mainly designed to support workstations and old terminals. The NOS is created using a software that allows multiple devices in a given network allowing them to communicate with each other and also share resources. The main role of NOS in devices is to provide basic network services and features which will support multiple input requests simultaneously in the environment. The NOSs were developed as a solution to the single-user computers (Figure 2.8).

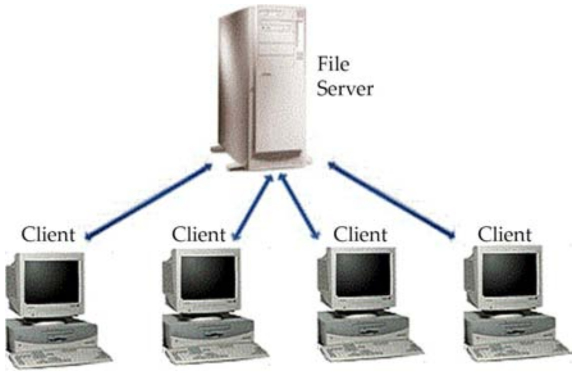


**Figure 2.8.** Network operating system is founding in several devices.

*Source: https://www.itrelease.com/2017/01/advantages-disadvantages-network-operating-system/.*

## 2.16. TYPES OF NETWORK OPERATING SYSTEMS (NOS)

There are two main types of NOS namely the client/server NOS and the peer-to-peer NOS. For the peer-to-peer NOS, the system enables the user to share network resources saved in an accessible network location. The architecture of the system involves devices being treated equally with regards to functionality. This system is known to work well in small to medium LANs. They are also relatively cheap to set up (Figure 2.9).

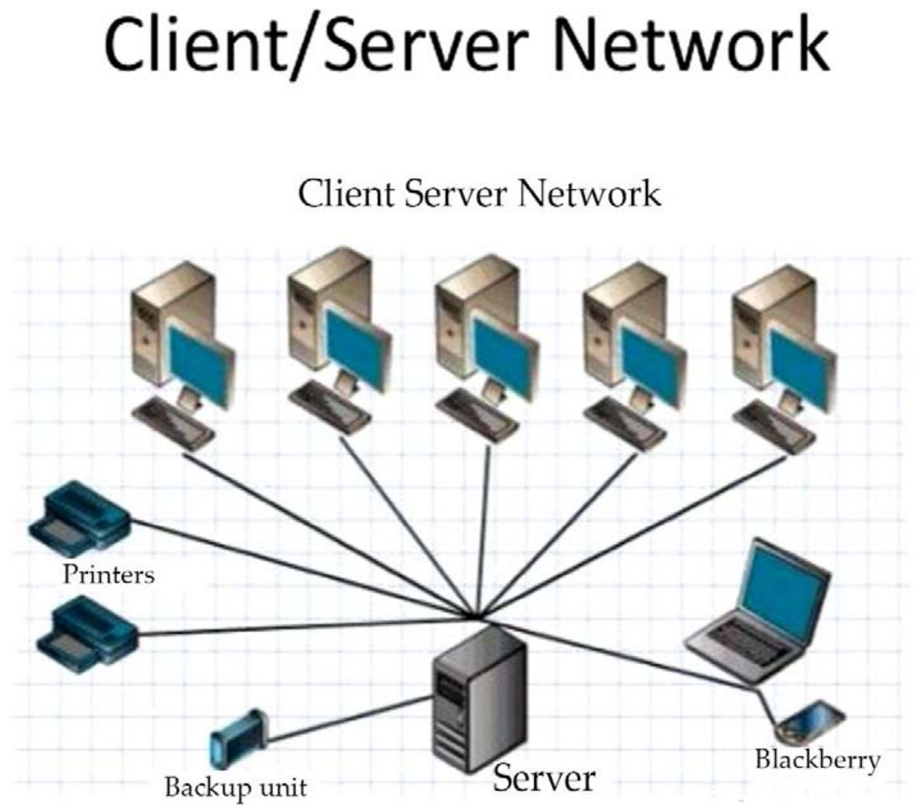


**Figure 2.9.** The client network operating system.

*Source: https://dcandcn.blogspot.com/2019/11/what-is-network-operating-system-nos.html.*

The client/server NOS grants users' access to resources by accessing a server. The architecture of this system is such that all applications and functions are unified under one file server that can be utilized in executing individual client actions despite their physical location (Jaeger, 2008). This system is quite expensive with regards to implementation and technical maintenance. However, they are very advantageous as the network is centrally controlled making it easy to make additions and changes to technology.

## 2.17. FEATURES OF NETWORK OPERATING SYSTEM (NOS)

There are a number of elements that influences the features of the NOS. These elements include the resource management functionality, system maintenance and user administration. Features of the system include internetworking, web services and backup, directory, network security capabilities such as access control and user authentication, database sharing and common file system, printer, and application sharing and basic support for OSs including multiprocessing, hardware detection and processor support.

### 2.17.1. Network Device Operating System (OS)

Some of the devices fitted with NOS include the hardware firewall or routers. Most of these devices operate the functions of the network layer.

Some of the NOS are as follows. Proprietary NOS common in most Cisco System routers and current Cisco network switches. There are various kinds of proprietary networks which include the LCOS that are mostly used in network devices made by LANCOM systems, the ZyNOS found in network devices made by ZyXEL, RouterOS made by MikroTik and the Cisco IOS which is a family of the NOS (Irwansyah et al., 2018).

In the Linux based OS, FreeBSD, and NetBSD there are various kinds of systems developed by different companies. There is the ONOS which is an open source SDN OS used by communication service provider. These systems are designed such that they ensure high availability, high performance, and scalability. The Vyatta routing package has an open-source fork called VyOS. The cumulus Linux distribution utilizes full IP/TCP stack of Linux. Other systems include SoNiC developed by Microsoft, OPNsense of pfSense, pfSense of MOnOwall and the Dell Networking OS.

### 2.17.2. Examples of Network Operating Systems (NOS)

NOS are categorized as software useful in enhancing the functionality of OSs by provision of added network features. Some examples of NOS include Microsoft's LAN manager which operates as a server application. This software was developed to run under Microsoft's OS.

## 2.18. REAL TIME OPERATING SYSTEM (OS)

The real time OS is a kind of operation known to have two main features namely determinism and predictability. The system is characterized by repeated tasks being performed within a tight time boundary. This is not the case for other general-purpose OS. The real time OS focuses more on determinism and predictability as they go hand in hand. By knowing how long a task is expected to take, we know that it will produce the same result. There are two main subdivisions of real time OSs which include the soft real time OS and the hard real time OS differentiated by the time taken to operate or react to a given situation (Hellmund, 2016). Real time OS is different from other OSs that consumers have interacted with. While other OS found on devices such as phones and PCs have multiple apps and features by ensuring that they are compatible with the user wants, the real time OS is very streamlined such that it is able to execute tasks quickly and effectively. Real time OSs are a fraction of the size of most OS by a few megabytes having a simple graphic interface but lacks some familiar features such as the web browser (Figure 2.10).

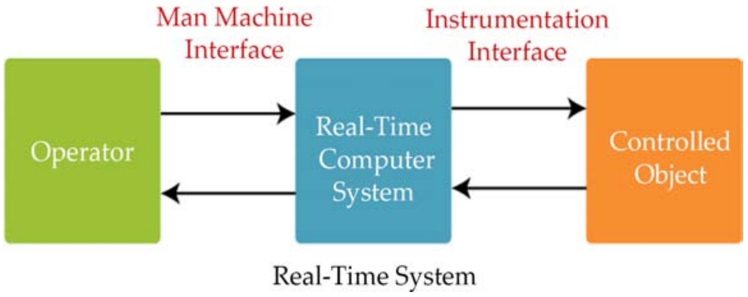


**Figure 2.10.** Real time operating systems have three main processes.

*Source: https://www.javatpoint.com/hard-and-soft-real-time-operating-system.*

## 2.19. CHARACTERISTICS OF REAL TIME OPERATING SYSTEM (OS)

There are five main types of real time OSs. They include determinism which involves the system repeating the input to generate the same output.

There is also high performance in that real time OS are very responsive and fast and are able to execute actions in a small fraction of time needed by a general OS. Security and safety are a factor where the OS is mostly utilized in critical systems where there are some catastrophic consequences such as flight controllers or robotics. These systems are very useful in protecting other components around them. Real time OSs protect other components by having high security standards and also having reliable safety features (Høiland-Jørgensen, 2018). Another characteristic is the priority-based scheduling which ensures that actions assigned as high priority are executed first followed by lower priority tasks. This means that the most important tasks will be executed first. Another characteristic is the small footprint where the real time OS weigh in a fraction of the size of other general OSs. For example, Windows 19 has post-install updates and takes up about 20 Gb while VxWorks on approximately 20,000 times smaller size when measured in the low single-digit megabytes.

Real time OS bear similarities and difference with embedded systems. An embedded system is usually embedded into a large machine such as a microcontroller on a robotic arm. The open-source general operating e such as the Linux are used in non-critical systems having some timeline flexibility. This makes the difference between real time OSs and embedded systems. Real time OSs are used in critical systems. Real time OSs have characteristics that are very vital for success. A good example is the robotic arm in a factory. It requires both reliable and predictable aspects of the systems enabling it to immediately stop when an employee enters its area of operation. In light of any variability, there may be injuries, issues with quality control and wasted resources.

## 2.20. REAL TIME OPERATING SYSTEM (OS) IN EMBEDDED SYSTEMS

Though real time OS may be different from embedded systems, its benefits enable it to be used in embedded systems. However, the real time OS will operate behind the scenes of a large operation. Real time OSs have no graphical interface. For this reason, several OSs are integrated simultaneously so as to provide usability of a general-purpose OS and operational capability. Real time OSs are considered to be very intelligent edge devices and are therefore known as electrochemical edge or cyber-physical systems. Devices having these systems are able to produce and operate on data (Hambarde et al., 2014). For instance, if the system is fitted into a car, it will be able to monitor the surroundings and act upon them instantaneously on its own. Most of the devices are often coupled with artificial intelligence or machine learning and in some cases, they are fitted with both. Presence of real time components in devices with real time OSs increase the underlying structure's capabilities.

## 2.21. ADVANTAGES AND FEATURES OF REAL TIME OPERATING SYSTEM (OS)

The advantages of real time operating are also its best features. Some of the advantages of the system include it being small, fast, deterministic, and responsive. As the features of the system, it makes them execute tasks efficiently and quickly. This enables the system to respond every time a task is given. The real time OSs tend to have a significance to the host device making its infrastructure more secure and minimizing chances of a fall or crash. Another advantage is that real time OSs are developer oriented. This means that there are roll outs of multiple updates enabling users to code more effectively.

There are different kinds of real time OSs. There are those developed by well-known companies and are ready to sell. There are companies that opt to develop their own real time OSs in house (Giorgetti et al., 2020). The system is usually tailor-made to suit the demands of the project. This is a better alternative compared to purchasing commercial off-the-shelf OSs. Tailor-made real time OSs have numerous advantages. One of the advantages is that the OS is designed to suit the need and the company has thorough knowledge of its mechanics and inner workings.

However, there are well known challenges of using tailor-made OSs. One disadvantage is that it is quite expensive as materials will have to be sourced. It also requires significant amounts of time for the system to be developed and used. If the developer is not familiar with working with OSs, they may require larger amounts of time to complete the project. This makes commercially available real time OSs easier, faster, and also highlights the experience of the technical team providing answers and support. Commercial OSs eliminate the need for guaranteeing capability and performance.

Several companies opt to use real time OSs as they are reliable and allow repeatable actions. This software is mostly used when both speed and reliability are greatly needed.

## 2.22. DISCIPLINES THAT IMPACT REAL TIME OPERATING SYSTEMS (OS)

There are various disciplines that influence the real time OS industry. Among them is by developments in the computer hardware industry. There are several developments in the technological world that affect hardware industry. They include adoption of new technologies including multi-core technologies. These technologies have become greatly common and their usage requires OSs to be updated so as to support them (Gunadi and Tiu, 2014). There are various emerging technologies that include 5G, machine learning and multi-core technologies that face scrutiny from developers of OSs. Changes in the industry means that real time OSs have to be updated and modified to support different cases. For these reasons, developers take into consideration trends in both software and hardware development. Changes in the field means that the developer's profile will pivot and updated allowing the support of new languages, new technologies, and new deployments.

## 2.23. REAL TIME OPERATING SYSTEM (OS) ARCHITECTURES

There are two main design philosophies that affect the design of such systems. They are the microkernel and monolithic kernel. Both systems are differentiated by their structures. Monolithic kernel system run in single space while microkernel systems make compartments in different components in the architecture.

### 2.23.1. Microkernel Systems

The microkernel system is made up of a microkernel architecture therefore its components are stored in separate rooms that are independent of each other though they share similar space. The architecture is such that a room can be renovated without necessarily impacting those around it. Moving from one room to another is disadvantageous as it consumes a lot of time. When a task is underway, the action has to return to the kernel before it can move to the component it reference. This causes some operations to take longer time than it should.

### 2.23.2. Monolithic Systems

Monolithic systems have a different architecture from the microkernel system as it does not have any walls between the rooms making an easy transition from one room to another. These kernels provide services of their own eliminating the need for the implementation of small kernels. These kernels also regulate those of other areas (Greenwald and Thomas, 2007). There are some exceptions made when using such systems. There are some operations that are executed in kernel spaces and remove the recurrent need to return to the kernel. This also improves the performance and speed of the system. Challenges are encountered when a change is made in one area. These changes could cause the ramifications of the entire system.

Operations and kernel are housed in separate spaces. The kernel is bare. Operation spaces are required to return to the kernel but are not given access to each other. Operation and kernel processes share the same space. Compared to the kernel, operations tend to move more quickly. The system boasts high performance. Extensive overhauls may be needed when making updates.

### 2.23.3. Examples of Real Time Operating Systems (OSs)

There are various products across the world fitted with real time OSs. About two billon devices across the world are powered by VxWorks. Some of the systems include car engines, deep-space telescopes, helicopter guidance systems and embedded systems. They all run all real time OSs. In A&D, some of the systems that make use of such systems include extraterrestrial rovers, drones, and engine turbine and flight display controller.

## 2.24. SOFT REAL TIME OPERATING SYSTEM (OS)

The soft real time OS form part of the two main kinds of real time OS. It is defined as a kind of system whose operation is degraded when produced results are not produced according to the specified timing requirement. In this system, it is not compulsory that each task meets their deadline as much emphasis is placed on processes being processes and generation the results. Though the system may miss deadline for some tasks, they cannot miss the deadline for every task or process according to task priority. The system can either miss or meet the deadline of the task (Figure 2.11).

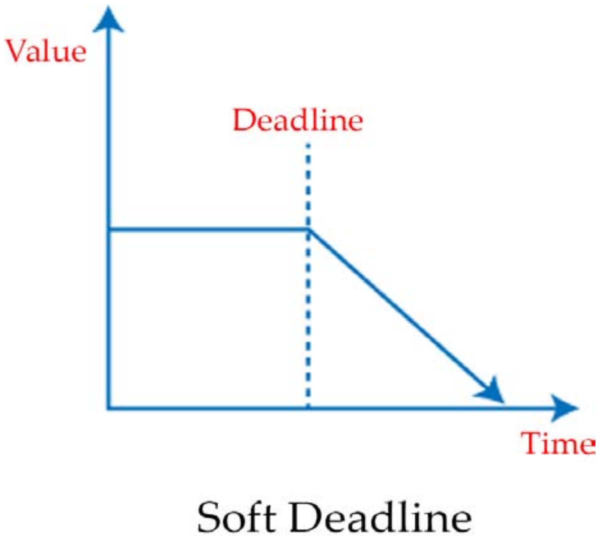


**Figure 2.11.** Soft real time operating systems have soft deadlines.

*Source: https://www.javatpoint.com/hard-and-soft-real-time-operating-system.*

If the system tends to have missing deadlines at a frequent basis, then it is likely that the system's performance will be worse rendering it un-usable by users. Some of the most common soft real time OSs used are video systems, audio systems and PCs. Soft real-time systems consider processes as main tasks and can handle the entire task.

### 2.24.1. Examples of Soft Real-Time Operating Systems (RTOSs)

Some of the most common soft real time OSs include mobile communication, virtual reality, online transaction systems, web browsing, multimedia systems, electronic games, weather monitoring systems, DVD players, set-tops boxes, audio, and video systems and PCs.

There are various characteristics used in identifying the soft real time OS. For instance, these systems are defined as systems in which one or more failures in meeting deadlines are not considered as much emphasis is placed on performance. With regards to file size, soft real time systems have fairly large data files. These systems also have fairly high response time and low utility. Soft real time OSs have enlarged databases (Fierro and Culler, 2015). These systems also tolerate peak loads and have long term data integrity. In soft real time OSs, safety is not critical and the system tends to be less restrictive. In these systems, there is rolling back of computation to a previously established checkpoint causing the initiation of a recovery system. These systems are also very flexible as there are greater laxities that can tolerate certain amounts of deadline misses. Users of the soft real time OSs do not get validation. Soft real time OSs are provided by Linux and telephone switches among other OSs.

## 2.25. HARD REAL TIME OPERATING SYSTEM (OS)

Unlike soft real time OSs, hard time real OSs consider timelines as deadlines. For such systems, timelines should not be omitted no matter the circumstance. These systems do not utilize any permanent memory making much importance on the need for processes to be completed properly in the first time itself (Figure 2.12).

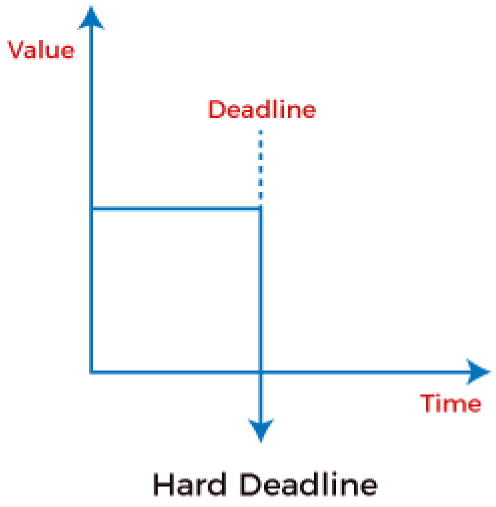


**Figure 2.12.** Hard real time operating system has hard deadlines.

*Source: https://www.javatpoint.com/hard-and-soft-real-time-operating-system.*

Hard real time OSs place much emphasis on generating accurate responses to events within a specified time. It can therefore be considered to be purely deterministic and a time constraint system. For example, if users expect output of a given input within 5 seconds then the system is expected to conduct all the processes and generate an output exactly by the 5th second. This means that output should not be generated in by the 4th second or 6th second. Therefore, 5 seconds is the deadline for the task to be completed. In the hard real time OS, meeting of deadlines is very important. When a deadline is not met then system performance fails (Fröhlich and Wanner, 2008).

### 2.25.1. Examples of Hard Real Time Operating Systems (OSs)

Some examples of hard real time OSs include pacemakers, autopilot system in plane, chemical plant control, anti-missile system, nuclear reactor control systems, air traffic control systems, railway signaling systems, inkjet printer system, medical system, weapons defense system, missile guidance system and flight control systems.

Some of the features used in identifying hard real time operation system include file size where the file size of such systems tends to be small or medium. The response time of such systems are predefined in milliseconds. With regards to utility, the utility of such systems is high. These systems have short databases and have predictable peak load performance. For hard real time OSs, safety is very critical and their data has short term integrity. The restrictive nature of such systems is very high. In such systems, occurrence of an error causes computation to be rolled back. These systems are not flexible and tend to have less laxity while deadlines are given full compliance. Users of such systems are able to get validation when needed.

CHAPTER 3

# POPULAR OPERATING SYSTEMS

## CONTENTS

It is important to understand the meaning of an operating system (OS) and an OS can't be defined as a software which act as a link between the end-user and computers hardware. For this reason, each computer needs to have at least one OS that is tasked with running other programs. That's our applications or programs that are found in our computer's hardware such as chrome Microsoft Word games and these applications require an environment where they can be able to perform the task (Estefo et al., 2019). The OS basically acts as a link between the password using the computer with the computer this is because it helps a person communicate with the computer using a computer's language. Therefore, it is not possible for one to use a computer without having an OS because he/she will not be able to understand the language of the computer (Figure 3.1).

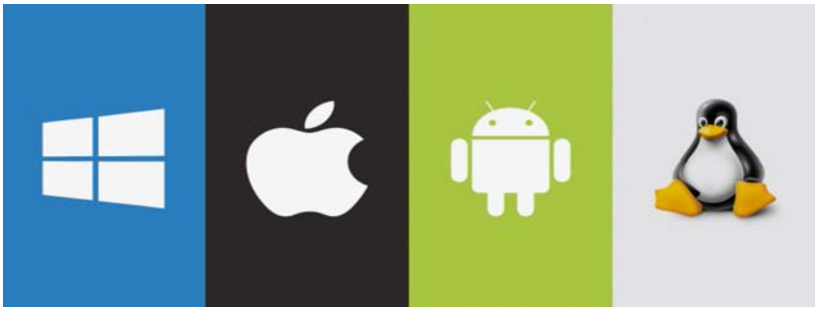


**Figure 3.1.** Operating systems.

*Source: https://www.howtogeek.com/361572/what-is-an-operating-system/*

## 3.1. HOW WINDOWS VERSIONS HAVE EVOLVED THROUGHOUT THE YEARS

### 3.1.1. 1985: Windows 1.0

This was a fast version to be released by Microsoft and its aim was to provide an interface that was user-friendly commonly known as a graphical user interface which was to allow, they use to navigate the system features more easily. However, the Windows 1.0 one did not drive well once it was released into the market because its release was shaky considering that Microsoft was considered to be a tech giant. Most of the people who used this version of Microsoft considered this software to be very unstable but one amazing feature about the Windows 1.1 is that it was able to fit into a single floppy disk. There was an interface known as the point and click interface which made it easy for new users to operate a computer that had these OS. Windows 1.0 has continued to offer various graphical user interface is such as the scroll bars as well as it OK buttons (Deseriis, 2017).

### 3.1.2. 1987: Windows 2.0 and 2.11

Compared to Windows 1.0, Windows 2.0 was considered to be more fast as well as reliable and had more graphical user interface features. The GUI had been slightly improved though it had a similar look to the Windows 1.01. This new version of the OS introduced the control panel and was able to run the fast version of Microsoft excel and Microsoft Word. The Windows 2.0 was able to support extended memory and updates for Microsoft which are compatible with the Intel's 80386 processor. After this revolutionary change Microsoft became the largest vendor for software in the world just as computers were becoming more renewal. Windows systems were considered to be user-friendly and also affordable which makes it easy for them to grow their PC market.

### 3.1.3. 1990: Windows 3.0

This version supported 16 colors and also included casual games that are Steeler used by the OS such as solitaire minesweeper and hearts. However, it is important to note that games that usually requires more processing power are still ran directly on MS-DOS. The Windows 3.0 and 3.1 had graphics and functionality that were more improved and they provided multimedia computer capabilities and also improved graphics and application support (Androulaki et al., 2018).

### 3.1.4. 1993: Windows New Technology (NT)

The Windows new technology (NT) pioneered the building of an advanced operational system because this fashion had a hardware abstraction layer. The DOS feature was only available through command prompt but it was not able to run the Windows operational system. The main idea behind Microsoft designing the NT as a workstation for the OS was for it to be used in businesses rather than at home this system was attributed with the introduction of the start button.

### 3.1.5. 1995: Windows 95

The introduction of this version is attributed with a major rhythm that Windows underwent in the year 1995 this revamp helped with improving multimedia and it became a more polished interface for its users. Does that menu that is usually a common feature in most computers was introduced by this version. Some built-in support systems included the Internet and networking support (De et al., 2007). This version was commonly used at home but it also proved to be very popular among various schools and businesses. It also facilitated the pioneering of hardware installation because of its plug and play feature.

### 3.1.6. 1998: Windows 98

This window version was almost similar to the Windows 95 because it's all filed the same functions but it is important to note that this version of bud and what idea display as well as enhancement of multimedia support. In this fashion the speed and the Play and Plug feature was improved. Another important feature that is associated with this version of Microsoft OS is the introduction of a USB support slot as well as a quick Launch bar. This version became very popular among people which made it a target for malware it is this fact that led to Microsoft integrating a Web technology into the OS and came up with the best web browser to be used on the desktops.

### 3.1.7. 2000: Windows Millennium Edition (ME)

This operating version was the last Windows version to use the Windows 95 code base. This version is usually denoted by a new feature which was known as a system restore. However, upon release all this version was found to be very unstable by many customers and some of the critics said that they ME abbreviation stood for Mistake Edition. During the same year Microsoft released its fast professional desktop which was initially called the NT 5.0 before it was renamed to the OS Windows 2000. The improvements that were made on the OS made it easy for each to be configured and installed in a private computer (DiLuoffo et al., 2018).

There was a more stable that was developed for the NT version. There was an increase in the number of home users that were using the Windows 2000 because it was greatly reliable. The plug and play support were updated which spurred many home users to use this OS. The most significant advantage of using the Windows 2000 fashion was that these system settings could easily be changed without necessarily having to restart the machine.

This version to prove to be a very stable OS because it offered enhance security as well as in the administration of the OS.

### 3.1.8. 2001: Windows XP

This was a fast NT-based system with a version that was specifically meant for a home user. People who use this version as well as critics rated the XP version highly. This system helped to improve their general appearance of the Windows system by providing colorful themes and also providing a stable platform. It also introduced a new gaming system known as the direct X enabled feature which was used in 3-D gaming.

### 3.1.9. 2006: Windows Vista

This window fashion was very hyped by Microsoft because they spent a lot of resources trying to develop an appearance of the Windows OS that was more polished. This distinguishing feature about the vista is that it had an interesting visual effect however the operational system was slow to start and run. This version had a major flow which is that it lacked the resources to run the systems which led to most businesses and home uses staying with the XP version (Dixon et al., 2012).

### 3.1.10. 2009: Windows 7

this version was built on the Vista Kernel. This version picked up a lot of vistas visual capabilities but the only advantage is that the capabilities were more stable. This version was complete advantages because it was faster to boot it had a new user interface and there was the addition of the Internet explorer 8.

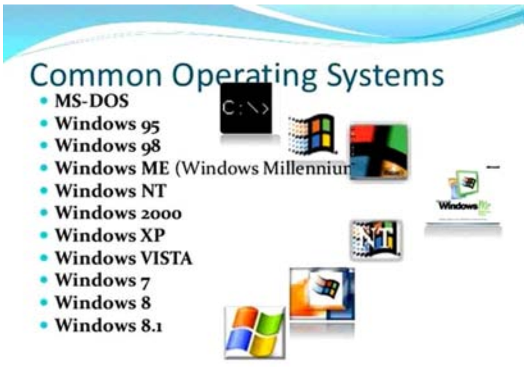
### 3.1.11. 2012: Windows 8

When Microsoft released this version of the OS it had a number of enhancement and it also had been diverted to use the metro user interface. Other enhanced features were the multi core processing, solid-state drives, alternate input methods such as touchscreen.

### 3.1.12. 2015: Windows 10

This Windows version was announced in September 2014 and Windows has skipped the launching of Windows 9 which was to be made on January 2015. What differentiates this version from other versions was the fact that

Windows 10 had the start menu which was absent in the Windows 8. The continuum is a responsive design feature found on the Windows 10 which was meant to adapt the interface depending on whether the password using the computer is using the touchscreen the keyboard or the mouse to input the commands (Dieber et al., 2017). There are Avenue features such as the on-screen back button which helped to simplify their touch input command. This operational system was also designed to have a consistent interface that was to be used across various devices including laptops, tablet as well as private computers (Figure 3.2).



Common Operating Systems

- MS-DOS
- Windows 95
- Windows 98
- Windows ME (Windows Millenniu
- Windows NT
- Windows 2000
- Windows XP
- Windows VISTA
- Windows 7
- Windows 8
- Windows 8.1

**Figure 3.2.** Common Windows OS.

*Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fskypip.com%2Ftypes-of-Windows-operating-systems.*

## 3.2. LINUX OPERATING SYSTEM (OS)

The Linux OS is among the most used as well as most used open-source OSs. This software is essential in every computer system because it usually sits behind most of the software in a computer and its work is to receive the requests from that software and those programs well drilling these requests or other commands to the computer's hardware (Figure 3.3) (Chinetha et al., 2015).

**Figure 3.3.** Introduction to LINUX operating system.

*Source: https://analyticsindiamag.com/5-reasons-why-linux-os-is-a-hot-favorite-among-coders/.*

The pioneers for this OS were Thompson Dennis Ritchie Douglas Mcllroy and Joe Osssna. The idea for this OS was initially conceived as well as implemented in AT&T's Bell labs based in the United States in 1969. It was initially called the UNIX OS. When it was first released in 1971, UNIX was written in a common language known as the assembly language that was considered to be a common practice at the time. In 1973, Dennis Richie became a Pioneer in writing this program using the C programming language. What made it easier to put these programming systems in different computer platforms was a high-level language implementation that was done on UNIX. There were many conflicts and wars that developed from how people used their OS between the year 1980 and the 1990 this led to Richard Stallman coming up with that project known as the GNU project which allowed people to access this OS for free (Cashmore et al., 2015). This project continued to grow and has today become the best operating tool used by the Linux OS. From the year 1990 till today it is safe to say that they Linux OS has undergone various transformations as well as development of the latest version of the OS and a good example is the Linux kernel 4.0 which was released in the year 2015.

## 3.2.1. Linux Distribution

The distribution of Linux OS is solely dependent on the opens with application or various software other than the Linux kennel. The distribution of the Linux system has various capabilities such as taking over the different system management tools, The Saba software, various applications from the desktop, and many other documentations that exist in a computer. The Linux distributions are also called distraught because their aim is to offer a common look and feel as well as an easy software management, various operational needs as well as the system security needs. Some of the Linux OS distributions are Ubuntu, the red heart, Debian, and Santos.' Other examples of the distribution of this OS include the oracle enterprise Linux (Figure 3.4) (Cao et al., 2008).

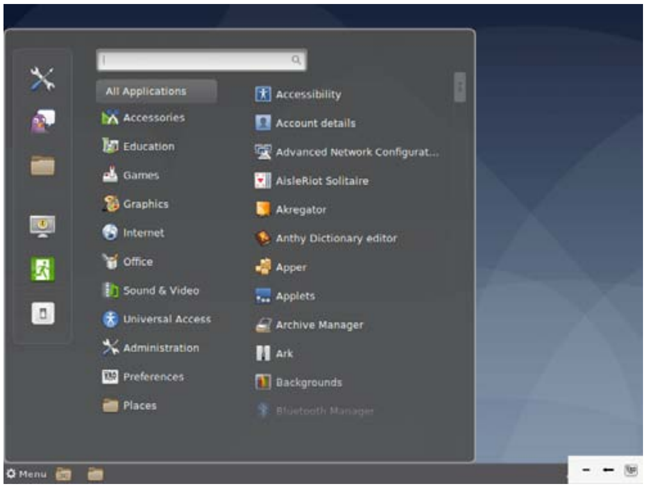


**Figure 3.4.** The different distributions of the Linux operating system.

*Sources: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.tecmint.com%2Flinux-distro-for.*

This distribution mostly consists of firms that come together and put effort into ensuring that they produce a better software and a system management tool making use of the Linux OS easy, secure as well as very effective. One example is the Ubuntu which is produced by a farm known as canonical which helped make the use of the Linux OS easier by allowing people to use this OS without using command lines. Putting this factor is into consideration many people who use their computers have opted to move from OS applications such as Microsoft and embracing the Linux OS such as Ubuntu programs.

### 3.2.2. How to Use Linux Operating System (OS) and Basic Linux Tools and Commands?

There are different versions of Linux that are usually produced by the programmers and their main work is to launch and handle applications, manage some of the hardware resources as well as provide user interface but in some specific forms. That is usually a very large number of developers as well as distributors when it comes to developing the Linux version which makes it easy for the Linux to perform any task as well as be able to penetrate many areas of the computing system. This OS can't be found in various settings which supports many different uses of this system (Bhattacharjee and Lustig, 2017).

Ubuntu requires that its users understand the basic commands found in the computer as well as tools that are used by this OS to allow them to have enough time as well as exciting experiences when enjoying the service is performed by the system and its distributions. The find command tool is usually common when using the Linux OS because this command makes it easy for one to locate files in the system as well as creating a more organized type of OS. Another important tool is there locating command which is important when you're using Linux the difference of this locate command from the find command is that it uses an index which helps it to identify and locate files within the system. This means that they look at command is usually faster as well as extra efficient as compared to the find command we're looking for files in the OS that uses their Linux program. There are other commands such as the data command which is used to display dates times and the location of the password using the OS. There is another operating command known as the Cal commander which is similar to the data command in that it is used to display dates current month and days of the user by highlighting the current data as well as month for the user to use easily.

Advantages of using Linux OS. As mentioned, Linux is usually free this means that you do not have to pay absolutely anything to access this OS one is not even required to pay the price of a CD. This OS is usually free because it can be downloaded because it is entirely free when access from the Internet this means that there are no registration fees, no cost per user, updates are usually free and the source code is usually available free if one wants to change the behavior of their system (Blackham et al., 2011). Linux is considered to be free speech this means that the license that is required for one to use Linux is their GNU public license this license stipulate that anyone who feels that like they want to change the Linux operation system are free to do so and actually have a right to change the OS and to distribute the version that they have created but there is usually one condition which is that the code should be made available after the redistribution.

The operation system is portable to any hardware platform this means that specific vendor who wishes to sell a new type of computer and do not know what kind of OS that is required in the new machine they can always up for the Linux kernel and install it to work on their hardware. The OS is secure and versatile. This means that the security model that is used in the Linux OS is usually based on their unique idea of security which has been known to be very robust and as well provide quality results.

One of the uses of Linux OS is to create a fort against enemy attacks such as cyber hacking from the Internet but it is usually used in other situations because it utilizes the same exceptional standards in the security system once this OS has been installed it is usually used as a firewall against attacks from various hackers on the Internet. Linux is scalable. This means that when using this OS what does not need a super computer because they're not OS usually provide services by using building blocks that are provided within the system. Linux is able to do little things such as making the OS for an embedded processor or recycling the old 486 OS (Ahmad et al., 2013). Most of the systems that use the Linux OS have a very short debug time. This means that areas are able to be identified very easily and taken care of because this OS is used by thousands of people and they are able to support these areas and report them which only takes a couple of hours for the bugs to be fixed.

The disadvantages of the Linux OS include. There are many different distributions that have risen from the Linux OS. This means that more people have access to this OS and therefore there are many feedback and opinions that these people have on the OS. Linux is also not user-friendly and can be confusing for beginners. This means that this type of OS is usually more complex than other OS such as the Microsoft Windows and Mac OS however the continued research of the OS has allowed it to be easy for new users.

## 3.3. ADVANTAGES AND DISADVANTAGES OF LINUX OPERATING SYSTEM (OS)

There are several advantages of using Linux OS and they include:

- Using this OS is very secure because they use a security and privacy data is considered a top priority in the OS. There are several locks designed for different layers which help in preventing the system from getting hijacked even when they are exposed to virus (Yin et al., 2016).
- This OS has a higher resistant ability to getting viruses because it has the ability to block out Malware is and software that might cause trouble for the OS. This OS has a restricted root access and like other OSs such as Windows. In this case every user needs to be an administrator so that they can be granted permission to control the system without causing trouble.
- This OS is stable because the likelihood of these OS to get bugs is very rare. It has regular updates which are interested in fixing the bags that might affect several Linux distributions. Before releasing every new version of the OS, the Linux distributions are tested to ensure that there are no bags that might interfere with their efficiency of this OS.
- Another advantage is that their OS is free and open source this means that one does not have to pay for them to be able to access the system. It is easily accessible by downloading it from the Internet and one can use and share in any way that they deem convenient for them. This means that everyone is entitled to reviewing the code and accessing the functionality of this OS by themselves.

There are several disadvantages of using Linux OS and they include:

- **Linux Operating System Blocks are Standard Edition:** Instead of having one single edition like in the case of Windows and Mac Linux OS has developed several additions known as distributions. This is often confusing to beginners because the different distributions have different ways of executing them which has proven to be tedious. Having this distribution also means that the community is divided which means that in cases where challenges are experienced getting support for a specific distribution of Linux is very challenging. The publishing of software is also a tedious task because every OS has it on manager package and it has different instructions on how to be packed (Bala et al., 2015).

- **Linux Operating System has a Hard Learning Curve:** For people who are just beginning to use Windows it is easier for them to understand how Windows operate. In the case of a pass on learning Linux for the past time it becomes hard because this OS is quite technical and requires a lot of experience. When dealing with Linux OS needs to know how to use the terminal which is similar to the command prompt which is found in Windows OS. This terminal is what makes their learning curve of Linux OS to be hard.
- **It has a Limited Market Share:** This is considered to be one of the biggest problems that this operation system undergoes. Linux has a limited market share which means that a lot of people are not able to access the Linux programs by developers. This makes it hard for many people to use Linux OS.
- **This Operating System is Difficult to Troubleshoot:** Troubleshooting done in a Linux OS can't be complex if you are not an expert in using this OS. It also becomes a problem finding tech expert for this OS. The problems that arise are specific to different users and answers on how to deal with such problems may not be necessarily available on the web (Aksoy et al., 2017). The issues that arise are from using different hardware and software which requires that the user has knowledge on the Linux operation system in order to provide a solution for his or her own problem. When purchasing this OS, the software has no warranty which means that in case it stops working or in case there is a damage to the software one cannot be able to be refunded or given another OS. It is also vital to note that finding the technical support to deal with software issues of Linux OS can be very time-consuming because one has to follow a process which is submitting a it is also vital to note that finding the technical support to deal with software issues of Linux OS can be very time-consuming because

one has to follow a process which is submitting a bad report and waiting for a publisher to solve the issue bug report and waiting for a publisher to solve the issue (Figure 3.5).

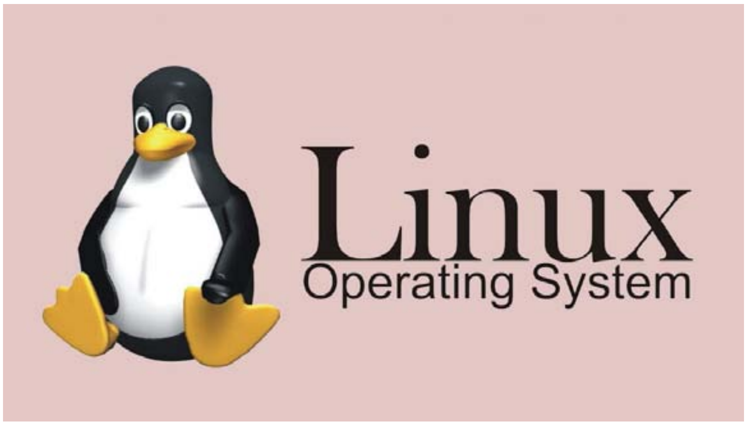


**Figure 3.5.** Advantages and disadvantages of Linux.

*Source: http://www.xlike.org/wp-content/uploads/Jenis-Linux-yang-Dapat-Digunakan-Pengguna-Dekstop-1024x576.jpg.*

## 3.4. THE VIRTUAL MACHINE SYSTEM

A virtual machine provides a complete system platform and also supports the execution of complete operational systems in a computer. This system virtual machine application has several uses such as providing a concrete platform where programs run and in situations where the real hardware is not available this means that the virtual machine system allows execution of several actions on platforms that are considered to be obsolete (Quigley et al., 2009). Another use of the system virtual machine is to create situations where there are multiple virtual machines which will help in leading to a more efficient way of computing resources. This means that this operational system is energy saving as well as cost effective. It is also vital to note that this system was originally defined by Popek and Goldberg as a duplication of an actual machine which is isolated and efficient (Figure 3.6).

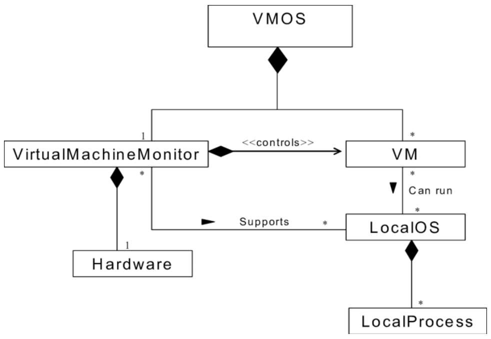


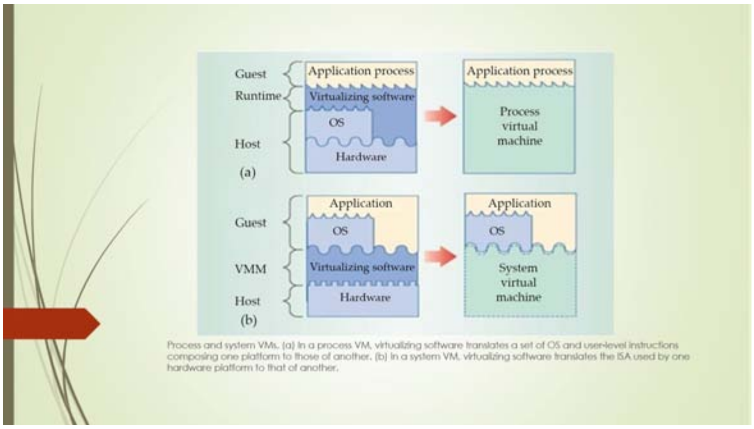
**Figure 3.6.** The virtual machine system.

*Sources: https://www.researchgate.net/profile/ for-the-Virtual-Machine-Operating-System-pattern.png.*

This OS has several advantages but the most important advantages include there are multiple operational system environments that have proven to have the core existent capability even when they are in the same primary hard drive. The operational system allows Visual petitioning of files shared that are either generated in the host operational system or the guest virtual environment. The most important aspect of these operational system is that all files are usually done and that one hard drive or the host operational system.

The operational system is able to conduct a disaster recovery process depending on the virtual machine software that is selected for that particular application. With the help of just-in-time compilation this operational system can provide hardware environments which are simulated as well as different from the Instruction architecture of the hosts (Whipple et al., 2009). The virtual machines are considered to be less is active compared to the real machine in situations where it is able to access the hard drive of its host indirectly. The virtual machines have proven to cause an unstable performance especially when it comes to execution of the malware protection when many virtual machines are running on the same hard drive.

It might require a supported software because most malware protections for VMs are usually not compatible with the hosts operational system, that is made for server consolidation when multiple versions are running via an operational system. This is done in order to reduce the interference from the separate virtual machines which are on the same machine platform. The invention of virtual machine was pioneered by the fact that there was a need for an operational system which allowed multiple operational system to run at once. These systems are considered to be vital because they allowed timesharing among operational systems which were tasked with the same task. This operational system requires that the user types a privileged instruction in there as a code and feed it to the computer as commands. The advantage of this operational system is that it added input/output (I/O) devices which were not allowed by the basic standard systems. The important advantages include improving the debugging process as well as enabling faster reboots of operational systems (Figure 3.7).



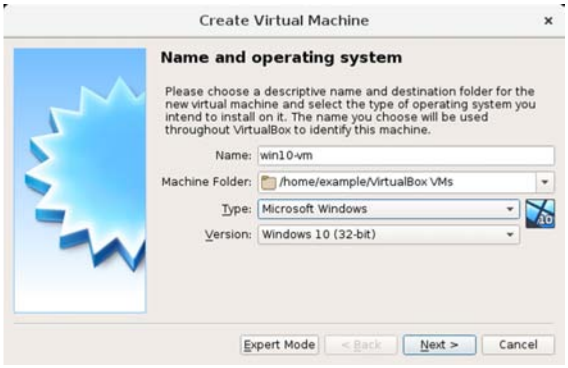
Process and system VMs. (a) In a process VM, virtualizing software translates a set of OS and user-level instructions composing one platform to those of another. (b) In a system VM, virtualizing software translates the ISA used by one hardware platform to that of another.

**Figure 3.7.** An example of a VM.

*Source: https://slidetodoc.com/presentation_image_h/bc70acfc21810a-f3a099993298f52f8c/image-19.jpg.*

## 3.5. TECHNIQUES USED IN THEIR VIRTUAL MANAGEMENT SYSTEMS

Based on their usage desired there are different virtualization techniques which are put into action. The first technique is known as the virtualization of the underlying rule hardware which is alternatively known as native execution. This is where the hardware fully virtualizes and is capable of implementation using to hypervisor that is the type one or type two hypervisor. The difference between these two hypervisors is that the type one hypervisor is executed directly on the hardware while the type two hypervisor is executed on another operational system which could either be a Linux or Windows (Figure 3.8).



Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: win10-vm
Machine Folder: /home/example/VirtualBox VMs
Type: Microsoft Windows
Version: Windows 10 (32-bit)

**Figure 3.8.** Creating a VM system.

*Source: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTzUIsbK2Ei_1ys9irJ2Y3EDSixN1ge2VpERQ&usqp=CAU.*

If the OS is supported by their underlying hardware, then it can be running buy a virtual machine OS (Wentzlaff et al., 2010). This characteristic allows it to run various guest operational systems simultaneously in different computers. Another technique is the emulation of a non-native system. This means that the virtual machine system can't perform the role of becoming an emulator. An emulator is tasked with allowing software applications as well as enabling operational systems for various computer processor to run. Another technique is the OS level virtualization. This technique is considered to be a server virtualization technology which uses the virtual images from different servers on an OS otherwise known as a Kernel Layer.

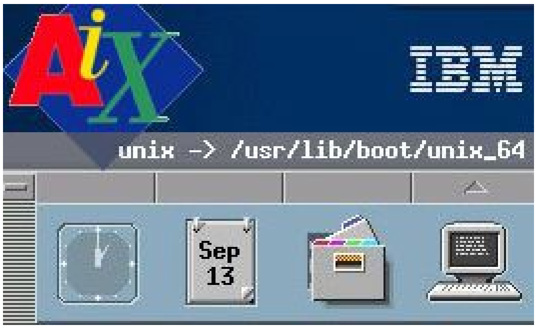
An example of the partitioning found on their virtual machine technique is known as the Solaris zones. This partitioning helps in allowing different operational systems to function under one hard drive as it is in the case for Solaris 10. One particular important points to note is that the host OSs used in the Solaris native zones are expected to be versions of Solaris because other manufacturers OSs are not supported by this particular system. Another non-partitioning is the system workload petition which is abbreviated as WPARs was the one that pioneered the introduction of bash on 6.1 of an IBM AIX OS. Use of sever resources our helps to maximize efficiency because they use low overhead in their systems. An example of a full virtualization system is the VMware. Another example is Xen or UML which is considered to be part of visualization. The existence of the system level virtualization does not allow more than one OS to run at the same time.

Components found in a virtual machine system:

- **The Dispatcher:** This component is in charge of controlling module that is called on by a trap set for the hardware. The dispatcher also helps in identifying the next expected module (Tsolakis et al., 2019).
- **Allocator:** This component is in charge of deciding the allocation of various resource systems. It is invoked by the dispatcher to allow the allocation of resources for the VMs.
- **Interpreter Routines:** This component resembles the job functionality of its users. However, it is important to note that the interpreter routines are not in charge of resource reassignment traps.

## 3.6. THE AIX OPERATING SYSTEM (OS)

AIX is an abbreviation which means advanced interactive executive. This OS is a proprietary four units with systems that are sold by the IBM. This OS is considered to be very secure and with robust infrastructure solutions for its market. The AIX OS is also considered to be scalable. According to the study done the AIX has a majority share of 58% in a server market where the estimation is over $5 billion. The power systems belonging to IBM Avenue to support several operational systems such as AIX and Linux. The AIX OS is very reliable and has a great support team which continues to be its added advantage with the market. This OS has recorded the lowest percentage when it came to serve a malfunction with an average of 7.2%. There is a live update functionality which makes the updates continue updating even without needing a system restart (Tang et al., 2010). Such updates include the kernel updates. This OS is also important for industries which rely on high-performance as well as dependability and such industries are financial institution such as banks, the government as well as healthcare providers. Because of the type of four months X has going to need to offer the most reliable infrastructure to its customers. The OS I love Customer to revise the data security change permission of the USA as well as configuring common criteria security evaluation (Figure 3.9).



unix -> /usr/lib/boot/unix_64

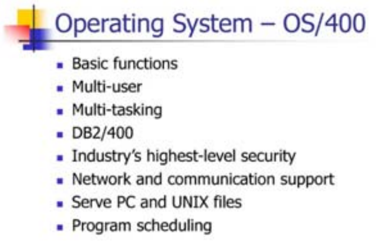
**Figure 3.9.** The AIX operating system.

*Sources: https://www.operating-system.org/betriebssystem/gfx/scr_preview_aix.jpg.*

This OS was fast designed to be used by IBM RT PC RISC. Upon further improve or an adjustment it was able to walk on various hardware platforms such as the Apple Network Server, the PowerPC-based systems and system 370 mainframes. It belongs to a group of five, known OSs which have been certified by UNIX 03 standard of the open group. The advanced interactive executive was launched in 1986 and the most recent stable version of this OS is the AIX 7.2. The target system type of this OS includes Server, NAS, and the workstation. The kernel type is considered to be monolithic with specific modules.

## 3.7. THE OS/400

The OS/400 is an OS that was initially designed to run on a hardware such as the AS/400. It was programmed and developed in the 1988 and it was introduced in June while releasing was done on August. The person who is credited with coming up with AS/400 computers is known as Frank Gerald Soltis who was a computer scientist best known in the United States. This OS has undergone a lot of changes and has been regularly updated and revised (Shin et al., 2014). However, this fact has not interfered with the fact that it is still a very competitive software because its features are compatible and they allow the users to access programs that were created on the specific operational system years ago. The main reason as to why the operational system was created is to help big companies in the enterprise resource planning process. This means that its specific functions include purchasing inventory managing the sales of the company marketing and also creating financial reports for the company (Figure 3.10).



Operating System – OS/400

- Basic functions
- Multi-user
- Multi-tasking
- DB2/400
- Industry's highest-level security
- Network and communication support
- Serve PC and UNIX files
- Program scheduling

**Figure 3.10.** The OS/400.

*Source: https://slideplayer.com/slide/12830081/78/images/3/Understanding+AS%2F400+System+Operations.jpg.*

There are two divisions of this operational systems that are controlled by the technology independent machine interface and these two divisions include the system licensed internal code and extended control program facility. The technology independent machine interface is a hardware that was split into two parts that they had were dependent and they had an independent system when it comes to these operational. These characteristics mean that they use are using the OS that specific OS that is separate without having to re-compile. The language used by this operational system was developed in 1959 and it is known as the report program generator (Figure 3.11).

What is IBM AS400 ERP Software?

IBM As400 is a legacy system that has been around since the 1980s. It has gone by many names throughout the years, including OS/400 and iSeries. The As400 operating system was the predecessor to the modern IBM i operating system, which is used today. The As400 platform has been continuously updated over the years and still remains in use in some industries today.

**Figure 3.11.** The AS 400 software.

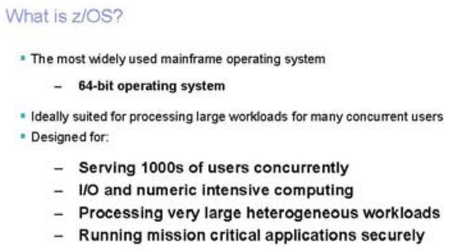
*Source: https://www.ximplesolution.com/wp-content/uploads/2022/01/IBM-AS400-ERP-Software.jpg.*

## 3.8. THE DIFFERENCE BETWEEN OS/400 VS. UNIX

- The OS 400 is object based and has an integrated database management system on the other hand UNIX is considered to be text based and has a relationship database system. It is vital to note that beginners might find the object the system had to use but the OS 400 is considered to be more reliable and secure and has the best data protection plan compared to UNIX.
- When it comes to storage the OS 400 uses the single level storage on the other hand UNIX uses a file system that is specific to Its operational system. The operational system 400 uses the storage as a two-dimensional plan for all of its addresses (Singh, 2014). The pages third in the operational system may be in the form of a primary storage also known as a ram or a secondary storage known as a disk. The UNIX operational system usually stores its large files a small unit so as to make it easy to manage all the files in the operational system. In this case the UNIX operational system is considered to be more efficient than the OS 400.
- The UNIX OS uses their Unix Kernel to control its hardware while the OS 400 uses this system licensed Internet code. These two hard drives are considered to be very good and very efficient but we're looking for a hardware layout that is more secure and has a complete file recovery system the system licensed internal code is considered to be the absolute best. This means that the OS/400 operational system is better than the UNIX OS. The system licensed internal code has the technology independent machine interface that allows the OSs as well as programs installed in a private computer to take advantages of advancing the hardware and the software without having to conduct a recompilation.
- As earlier stated, the iOS/400 uses a programming language known as the RPG while the UNIX OS uses a programming language known as the C programming language. In this case the UNIX OS is considered to be exceptional because of the C programming language which was able to run on many platforms. The RPG programming language was not portable which give the sea programming system an added advantage making it convenient to use the UNIX OS.

## 3.9. Z/OS OPERATING SYSTEM (OS)

This OS was introduced in 2000 by the IBM. It is considered to be a 64-bit OS with its mainframes belonging to the IBM. This operational system was developed from its predecessor OS/390 which in turn released the MVS versions. It continues to use its past functionalities that were first developed in the 1960s because it is designed for backward compatibility (Figure 3.12) (Sangorrin et al., 2010).


What is z/OS?

- The most widely used mainframe operating system
  - 64-bit operating system
- Ideally suited for processing large workloads for many concurrent users
- Designed for:
  - Serving 1000s of users concurrently
  - I/O and numeric intensive computing
  - Processing very large heterogeneous workloads
  - Running mission critical applications securely

**Figure 3.12.** Understanding the z/OS operating system.

*Source: https://slidetodoc.com/presentation_image_h/e7bcc65ae-a8911e729ef4441212a7207/image-4.jpg.*

This OS supports various stable mainframe facilities such as CICS, COBOL, SNA, and IBM MQ. This OS is also known to contain record-oriented data access methods such as REXX, CLIST, and ISPF. These compatibilities are essentially because they allow these Z/OS to run in a range of commercial as well as open-source software (Figure 3.13).

**Figure 3.13.** Representation of z/OS.

*Source: https://upload.wikimedia.org/wikipedia/commons/1/19/IBM_System_z10.jpg.*

Some of the characteristics of the Z/OS include:

- This system is uniquely designed to preserve the integrity of data fed to servers regardless of how large the users of the servers might be. This is achieved because the OS prevents different users from accessing or changing the objects that are found within the system including the data of the user. The only way these changes can be implemented is if the system provided interfaces in charge of enforcing authority rules gives a go ahead (Sharma et al., 2012).
- This system is able to work under minimal supervision even considering the fact that it has to manage a large number of patch jobs which are concurrent. There is no external workforce that is needed to balance or evaluate the integrity of the program that arises from the simultaneous use of a given set of data.
- The security system of the OS is designed to extend its services to other files including simple files and it can be incorporated into applications as well as user's profiles.
- The desperate communication-oriented applications are able to run at the same time because the OS allows several sub systems of communication to run at the same time which creates an unusual flexibility. A good example is the IP addresses which are capable of serving different applications at the same time.
- The OS has an extensive software recovery system which is very efficient in a production environment as there is no need to make unplanned system restarts. The interface is found in the OS allow the application programs to provide their own layer of recovery system (Shaw et al., 2016). However, it is vital to note that these interfaces are not usually used by applications which are simple they are rather used by sophisticated applications that are quite extensive.

CHAPTER 4

# OPERATING SYSTEM PROPERTIES

## CONTENTS

## 4.1. BATCH PROCESSING

Batch processing is defined as the process by which computers complete batches of jobs in n a non-stop sequential order. The jobs are completed simultaneously. This process also works by ensuring that large jobs are computed in small parts for to ensure that the process is efficient during the debugging process. The process is a kind of command. These commands are given different kinds of names including job scheduling and workload automation (Silva et al., 2006). Batch processing has undergone a number of changes over the years. These changes are beneficial as they have made the process more efficient and sophisticated. Adopting of batch processing in business has made daily activities a success (Figure 4.1).

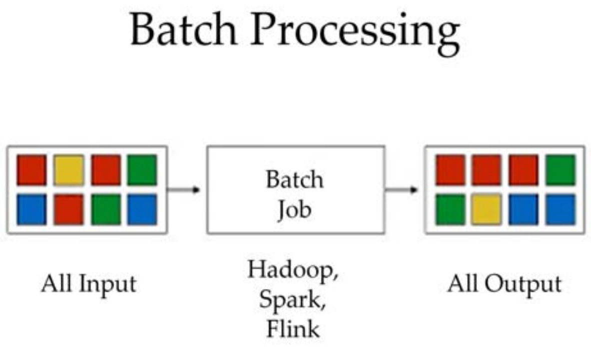


**Figure 4.1.** Batch processing is very effective in performing tasks.

*Source: https://medium.com/techco/ml-batch-processing-and-streaming-162d11973eab.*

## 4.2. EVOLUTION OF BATCH PROCESSING

Batch processing can easily be identified by one main characteristic which is the lack of user interaction. Batch processing need few, if any, manual processes to kick it off. The low amounts of processes needed to kick it of contributes to the efficiency and success of the process. High levels of success and efficiency is a recent development. Punch cards were initially used in batch processing. They were tabulated and would give computers instructions on what to do. Initially, batches or decks of cards would be processed one at a time. Punch cards were created in 1890 by Herman Hollerith. He created punch cards to process census data. This development came as he was working for the U.S. Census Bureau. He developed a system within which a punched cards could be read by an electromechanical device. The cards were punched manually. This led to the development of a company known as IBM (Sjöstrand et al., 2015). Development of punch cards revolutionized operation of businesses. Batch processing has continued it evolution over the past two decades. This has led to decline in the need for businesses to hire data entry professionals. These developments are such that batch processing functions can be enabled without interaction. Tasks are done and completed to meet specified timing needs. There are different ways of executing tasks. There are jobs that are done immediately while others are completed in real time with daily monitoring (Figure 4.2).

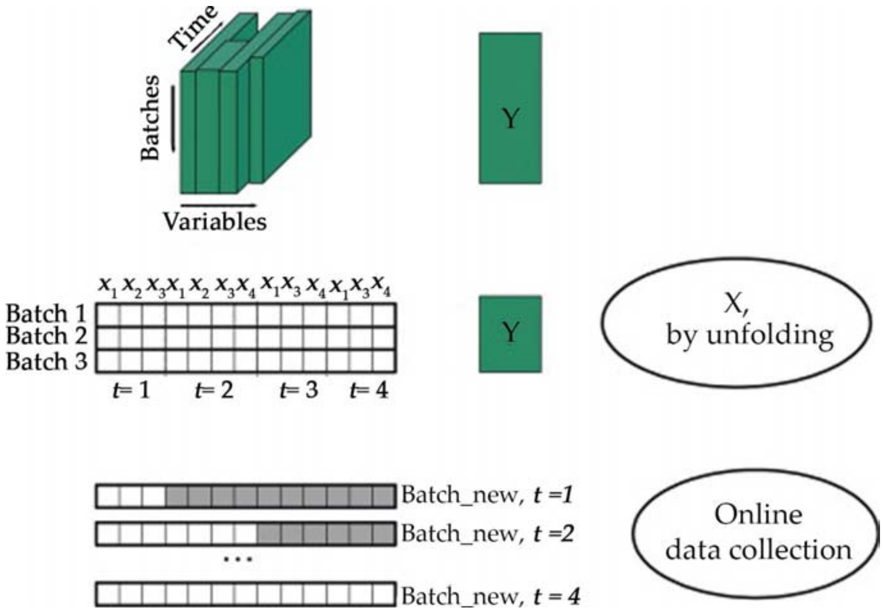


**Figure 4.2.** Evolution of batch processing has made the process dependable.

*Source: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/batch-process*

## 4.3. MONITORS AND DEPENDENCIES IN BATCH PROCESSING

In the current day and age, there are exception-based alerts used in batch processing that notify relevant stakeholders if there are any issues. This is very advantageous for managers are they are able to work freely without having to regularly check the progress of batches. The ideology behind exception-based management is that the manager does have to check on a batch unless they get an alert about a critical exception. A system of dependencies and monitors that are very vital to the software are useful in determining these exceptions. The dependencies are defined as the events that trigger the start of batch processing. A good example is when a customer places an order online or when they request for new supplies. This triggers the system to generate a request. Batch processing is therefore set in motion by dependency. Monitors are part of the system that identifies abnormalities in the batch. An abnormality includes a task taking a longer duration than the expected in completing a task. This causes a delay as other tasks cannot be completed before the other is completed. The unusual delay is caught by the monitor which will in turn generate and exception and send it to the manager (Santos et al., 2013).

When using batch processing, one may encounter latencies. Several cases of such delays are noted before the start of data transfer and is not a big issue as processes using this function are not mission critical during that exact moment. Some situation that may require the use of batch processing include times when large volumes of data are to be processes, work is repetitive, tables in relational databases have to be joined, complex algorithms must access the entire batch, real-rime transfers and results are no crucial and when data is accessed in batches rather than streams (Figure 4.3).

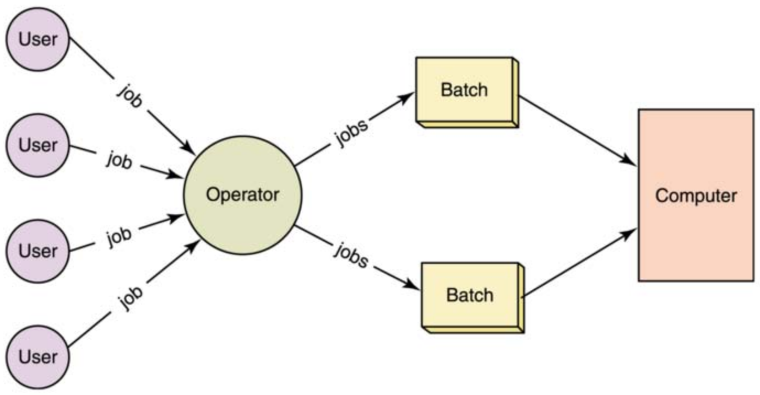


**Figure 4.3.** Process related to batch processing are conducted by the operating system.

*Source: https://padakuu.com/batch-operating-systems-27-article.*

Some features of batch processing are as follows: an operating system (OS) conducts activities related to batch processing, the OS defines a job having predefined sequence of data as a single unit, programs, and commands and the OS keeps several jobs in the memory that are executed without any manual information. Also, when the execution of a task is completed, the memory is released and the output of the job is copied into an output spool where it will be printed or process after sometime. Another feature is that tasks are processed in the order of submission that is the first come first served fashion.

## 4.4. ADVANTAGES OF BATCH PROCESSING

Batch processing systems have been developed by a number of companies due to a variety of reasons. These reasons have affected their outlook on software organization. Some of the advantages are as in subsections.

### 4.4.1. High Speed at Lower Costs

The evolution of batch processing is such that companies to do need to hire data entry clerks to support function of batch processing systems. This has been useful in reducing operational costs that could be spent on labor. The system does not need any additional hardware to function outside the computer. It is a known fact that adoption of batch processing can reduce a company's reliance on expensive hardware. This provides an inexpensive solution that helps the business to save on time and money. Batch processes can execute a task in the most efficient way in the absence of the possibility of user error. This makes the results fast, accurate, and time managed so that managers are able to spend their time on other day-to-day operations (Peter et al., 2015).

### 4.4.2. Offline Features

Another advantage of batch processing is that their systems can work offline. This means that batch systems are processing in the background when the workday ends. It is advantageous for managers are they get ultimate control over when to start processes. This also enables setting of software overnight allowing processing of batches. This is very convenient for businesses that may not want job like automatic downloads to disrupt daily activities.

### 4.4.3. Easy Hands-Off Management

Managers are able to benefit from the fact that they conduct other activities without having to log in to check on progress of batches. This is brought about by the use of exception-based notification systems. This system is a modern batch processing software that enables managers go about with their activities without having to worry about whether their software is properly functioning or whether the batches are being completed. An occurrence of an issue causes a notification to be sent to right person to solve it. This is the hands-off approach that enables managers' trust that batch processing software is doing their job (Ow, 2011).

### 4.4.4. Simplicity

When batch processing is compared to steam processing or real-time processing, it is found to be less complex. This is because it does not require constant system support for unique hardware or data input. When the system has already been installed or established, the system does not require heavy duty maintenance and is therefore a low-barrier-to-entry solution.

## 4.5. DISADVANTAGES OF BATCH PROCESSING

Even though batch processing has numerous advantages, it also has its own disadvantages. This may require companies to review its components and highlight some components that may be disadvantageous for them. Some of the disadvantages are as in subsections.

### 4.5.1. Deployment and Training

When dealing with technology, and update may require those using it undergo training for proper utilization of the equipment. Managing of batch systems may require owners undergo training. This is applicable for managers who are not familiar with what triggers a batch, how to schedule them and the meaning of exception notifications among others (Nimodia and Deshmukh, 2012).

### 4.5.2. Complex Debugging

Managers need to possess knowledge needed in fixing errors when they occur. Batch processing systems are quite complex to debug. For this reason, managers may require an in-house employee with an expertise in these systems. Absence of such an employee may mean that the company will have to incur extra costs on hiring an outside consultant to assist.

### 4.5.3. Cost

When switching to batching system, most businesses will save on money on hardware and labor as some do not have expensive hardware or data entry clerks. Such businesses may see the cost of batching systems as unfeasible.

In most cases, companies tend to deliberate on whether what system is beneficial to them. They try to compare the advantages between batching and steam processing. In as much as a clear-cut answer may be most ideal, there is no option that can be considered as the perfect solution for various instances. Depending on the needs of the company, there may be an optimal method.

## 4.6. MULTITASKING

Multitasking is a property of an OS where by the CPU of a computer is able to conduct multiple jobs simultaneously by switching between them. Switches are so frequent such that the user is able to interact with each of the programs while it is running. This kind of property therefore allows the user to perform multiple computer tasks at a time. Multitasking is very useful in that initially, when other tasks were being conducted the introduction of a new task would leave the process in jeopardy. This means that tasks share common processing resources such as the main memory while the computer executes segments of multiple tasks in an interleaved manner (Nollet et al., 2004). Multitasking in computer systems automatically interrupts running programs. When a new task is introduced, the system saves the already generated results and loads the saved state of the other program and transfers control to it. Switching can be initiated at fixed time intervals and is known as pre-emptive multitasking. Switching can also be initiated when a running program can be coded to signal the supervisory software when it can be interrupted. This is known as cooperative multitasking (Figure 4.4).

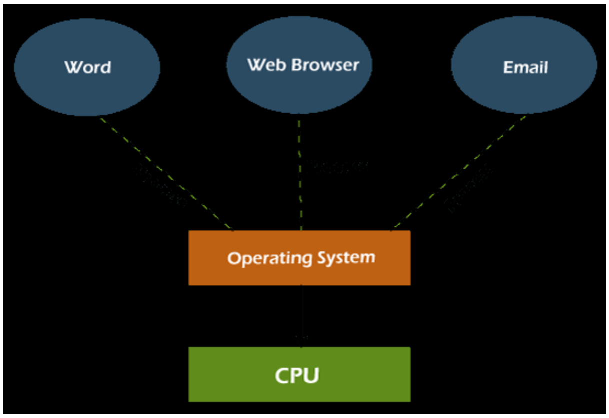


**Figure 4.4.** Multitasking enables the operating system perform several tasks in a given period of time.

*Source: https://www.javatpoint.com/multitasking-operating-system.*

Using multitasking in the OS does not require parallel execution of multiple tasks at a go. It however allows more than one task to be conducted for a given period of time. In multiprocessor computers, the multitasking feature enables the CPU run several tasks. In some cases, the tasks are more than the CPU. Multitasking enables full utilization of computer hardware. The computer can await for external event such as entry of data, the central processor can continue with other programs. There are various OSs that adopt multitasking including the real time OS. In real time OS, several users utilize the same processors as though they were dedicated to their use. In such a case, behind the scenes activities includes the computer serving the several users by multitasking individual programs. Multitasking OSs may include adoption of measures that enable the system to change the priority of individual tasks. This enables very important jobs to receive more processing time than those considered less significant (Moore and Stouch, 2016).

There are different kinds of tasks depending on the kind of OS. There are cases where tasks are larger as an entire application program. In other cases, there may be tasks made up of smaller threads that carry out portions of the entire program. Multitasking OSs are often used with certain kinds of processors that may include special hardware that securely supports multiple tasks such a protection rings and memory protection. Both the protection rings and memory protection are useful in ensuring that the supervisory software cannot be damaged or subverted by user-mode programs having errors.

## 4.6.1. Cooperative Multitasking

All multitasking systems are known to use applications that voluntarily cedes time to one another. The approach was supported by computer OSs and is currently known as cooperative multitasking. Cooperative multitasking is rarely used in larger systems in exception of specific applications including the JES2 and CICS subsystem. There was a time when cooperative multitasking was the only scheduling scheme employed by classic Mac OS and Microsoft Windows. In both cases, cooperative multitasking enabled running of multiple applications simultaneously. In the current day and age, cooperative multitasking is used in RISC OS systems (Monaco et al., 2013). The cooperative multitasking systems rely on each process regularly giving up time to other processes on the system. This means that if there is a poorly designed program, it will take up most of the CPU's time as it may conduct extensive calculations or it may be busy waiting. Both cases may cause the entire system to hang. When defined according to server environment, this is considered a hazard as the entire environment becomes unacceptably fragile (Figure 4.5).

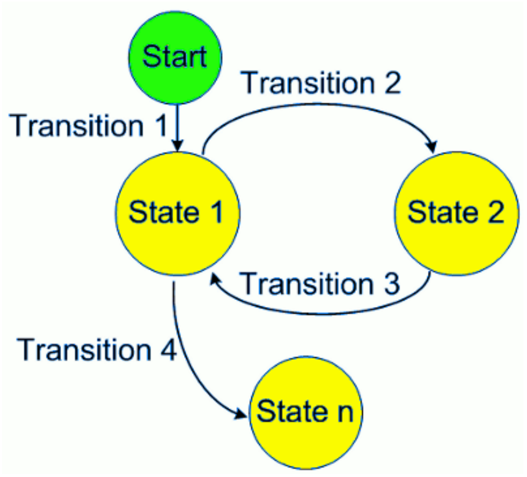


**Figure 4.5.** Cooperative multitasking enables several tasks to be performed simultaneously.

*Source: https://www.deeptronic.com/software-design/state-machine-and-co-operative-multitasking-model-simplify-complex-processes-programming-for-microcontroller/.*

### 4.6.2. Preemptive Multitasking

Preemptive multitasking is very different from cooperative multitasking in that it enables the computer system to reliably guarantee that each process gets a regular slice of operating time. This kind of multitasking also enables the system to deal rapidly with important external events such as new data that may require important attention from the processors. For this reason, OSs were developed in a manner that enabled them take advantage of these hardware capabilities as well as running of multiple processes preemptively. In 1964, the preemptive multitasking was implemented in Monitor and MULTICS, UNIX in 1969 and OS/360 MFT in 1967. Other computers were also able to access preemptive multitasking (Figure 4.6).

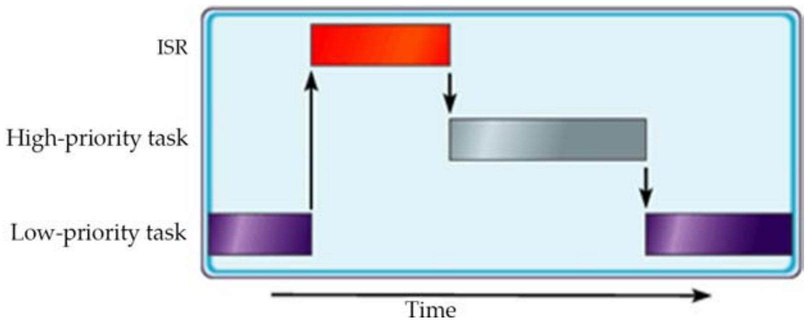


**Figure 4.6.** Preemptive multitasking ensures that all tasks are given an opportunity for operation.

*Source: https://www.embedded.com/introduction-to-preemptive-multitasking/.*

While using the preemptive multitasking, processes can be grouped in to two categories at any specific time. The two categories include those that are waiting for input or output hence the name I/O bound and those fully utilizing the CPU hence the name CPU bound. When used in primitive systems, the software could busy wait or poll while waiting for requested input such as network input or keyboard input. During this time the system is not performing useful work (McClean et al., 2013). In the advent of preemptive multitasking and interruptions, I/O bound processes could be put on hold or blocked as it waits the arrival of necessary data. This enables other processes to utilize the CPU. This is because the arrival of new data would cause the generation of an interruption. This also guarantees that blocked processes can be guaranteed a timely return to execution and completion.

In 1984, the Sinclair QDOS on the Sinclair QL was developed as the earliest form of preemptive multitasking OS easily available for home users. The challenge faced was that few people bought the machinery due to various reasons. This led to the release of the first commercially successful home computer that utilizes the technology. The multimedia capabilities of the machinery make it a clear ancestor of contemporary multitasking personal computers (PCs). In the early 1990s, Microsoft made preemptive multitasking a core feature of the flagship OS. This was done as they were developing Windows NT 3.1. The feature was later on adopted on the Apple Macintosh by Mac OX. The system utilized preemptive multitasking for every native application. Other similar models were used in the Windows NT family and Windows 9x. In both, native 2-bit applications are preemptively multitasked. For the 64-bit editions of Windows, preemptive multitasking supported all supported applications.

## 4.7. MULTIPROGRAMING

Multiprogramming is considered a rudimentary form of parallel processing where several programs are run at the same time on a uniprocessor. In multiprogramming there cannot be simultaneous execution of tasks as there is only one OS. In such as case, the computer conducts part of one program then parts of another. The user may insinuate that all programs are being run at the same time. In the case where the machine has the ability to cause an interruption after given period of time then the OS is able to execute each of the programs for a given length of time. It then takes time to regain control before it proceeds to execute another program for a given period of time. Absence of this mechanism means that the OS has no choice but to begin program execution with the expectation but no certainty that the program may eventually return control to the OS (Manzalini and Crespi, 2016). A program is less likely to interfere with the execution of other programs when it has the capability of protecting memory. If a system does not have memory protection, presence of a bug may make one program to change the contents of storage that has been assigned to other programs. It can also tamper with storage assigned to the OS. This leads to the occurrence of system crashes that are not disruptive but are very difficult to debug since there is no definitive knowledge on the program at fault (Figure 4.7).

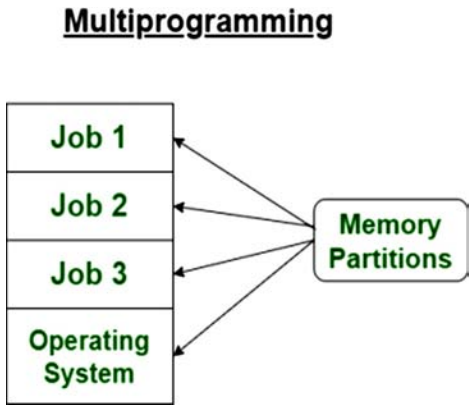


**Figure 4.7.** Multiprogramming is a rudimentary form of parallel processing.

*Source: https://www.geeksforgeeks.org/difference-between-multiprogramming-and-multitasking/ https://www.geeksforgeeks.org/difference-between-multiprogramming-and-multitasking/.*

The main goal of multiprogramming is to overcome issues of under-utilization of primary memory and CPU. Multiprogramming also ensures that all the resources are properly managed. There are certain components that enable multiprogramming on OSs. They include the transient area, I/O control system, file system and command processor. An OS designed to conduct multiprogramming activities are built on the basis of a principle stating that sub segmenting parts of transient area can be used to store individual programs. In most cases, resource management routines are attached with basic functions of the OS. While using a multiprogramming system, the system enables users to perform several tasks concurrently and can be stored into the main memory. This renders the OS able to deliver time to several programs while it is idle mode as one is engaging with other I/O systems (Muehlstein et al., 2017). While the system is working, one program is able to wait for I/O transfer while the other program is ready to use of processor. When all jobs are executed at the same time frame, it is not referred to as multiprogramming but is defined as multiple jobs present for processor. Part of the processor is executed followed by another segment and so on. When a program is in the execution process, it is referred to as a task, job, or process. Concurrent execution of programs is useful in improving performance of system throughput.

## 4.8. TYPES OF MULTIPROGRAMMING OPERATING SYSTEMS (OSS)

There are two main types of multiprogramming OSs. They include a multiuser OS and multitasking OS.

### 4.8.1. Multitasking Operating System (OS)

Multitasking OS allows the execution of two or more programs in the same duration. This is achieved by shifting each of the programs into and out of memory on at a time. In the case where the program is switched out of memory, it is temporarily saved on disk till the time is required by the system.

### 4.8.2. Multiuser Operating System (OS)

A multiuser OS is different from multitasking OS as it allows many users share processing time on a powerful central computer from different terminals. This is accomplished by the OS switching rapidly between terminals each of which receives a set amount of processor time on the central computer (Mayoral et al., 2017). The OS is able to change among terminals quickly such that each user is able to have continuous access to the central computer. In the case where there are several users on a system like this, making obvious time taken by the central computer to reply.

### 4.8.3. Activities Related to Multiprogramming

Some of the activities related to multiprogramming include the OS keeping several jobs in the memory at a time. The set of jobs is considered a subset of jobs kept in the job pool. The job is picked by the OS and its execution is started. The multiprogramming OSs monitor states of all active programs and system resources. This is done using memory management programs. These programs ensure that the CPU is never idle in exception of instances where there are no jobs to process. Examples of multiprogramming OS include Firefox browser, google chrome, MS-Excel, transfer data, Windows O/S, UNIX O/S, ESQ view, MP/M, XENIX, and downloaded apps.

## 4.9. ADVANTAGES OF MULTIPROGRAMING

There are numerous benefits of multiprogramming in OSs are as follows:

- Multiprogramming increases CPU utilization. This way, it is never idle.

- Multiprogramming also ensures that resources are smartly utilized.
- OSs using multiprogramming tend to have less time response time.
- Multiprogramming enables several users access the system all at once.
- Multiprogramming is useful in executing multiple tasks in a single application at the same time duration and is also useful in improving turnaround time for short jobs (Lin and Ye, 2009).
- Multiprogramming reduces total read time required in executing a job. It is also useful in optimizing total job throughput of a computer.
- Multiprogramming enables fast monitoring enables entire tasks to run in parallel.

## 4.10. DISADVANTAGES OF MULTIPROGRAMMING

Some of the disadvantages of multiprogramming is that the CPU requires scheduling from systems to run properly. Multiprogramming also requires memory management as all manner of jobs are stored in the main memory. Multiprogramming presents the challenge of managing all jobs and processes. Some users may find multiprogramming to be highly sophisticated and complex (Figure 4.8).

INTERACTIVITY

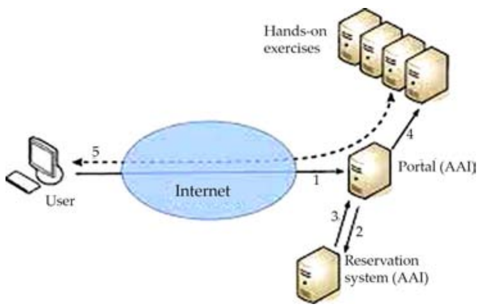


**Figure 4.8.** Operating systems have been developed to enable interactions between human beings and computers.

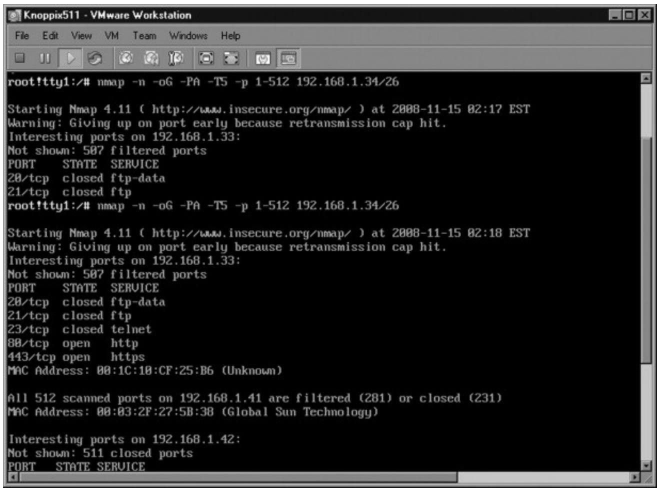
*Source: https://ercim-news.ercim.eu/en71/special/oslab-an-interactive-operating-system-laboratory.*

Interactivity in computer system refers to the dialog known to occur between a computer program and a human being which is the living creature. Interactivity occurs when there is an immediate user involved in the running of computer programs. Those that do not interact with immediate users are usually called background or batch programs. Great interactivity is mostly noted in games as the user interacts with the computer from long hours. There are other kinds of interactive software including order entry applications and other business applications. All though they are interactive, they are constrained in a certain way such that they offer the user limited interaction. When using the World Wide Web, the user not only interacts with the browser but also the pages brought by the browser. There are implicit invitations called hypertext useful in linking the user to pages. This is the most common form of interactivity when using the Web. The World Wide Web can be thought of as a giant made up of interconnected application programs (Levis et al., 2005). Other than hypertext there are other applications that offer possibilities for interactivity. They include the Web and other non-Web applications in a computer system. There are various kinds of input including any kind of user input such as clicking the mouse or typing commands. Output forms of interactivity include sounds, motion video sequences, printouts, text, and displayed images.

In the earliest development of interaction with computers contained an indirect interaction with computers and it consisted of submitting commands on punch cards once on the computer reads them and performs the commands. More advancement saw that computer system were designed such that even the average person other than programmers were able to interact immediately with computers. It tells them some of the programs to run and they can interact with those programs such as editors currently called processor, drawing programs and other interactive programs. Initially, the interactive human-computer interface tended to be input text sequences also known as commands and terse one-line responses from the system. The first graphical user interface was developed in the late 1970's and it emerged from Xerox PARC Lab. The interface found its way into Apple Macintosh PC and finally into Microsoft's Windows OS. This led to PCs adopting the use of graphical interface. There are estimates that suggest that about 90% of computer technology development effort is focused on enhancing and making innovations in interaction and interface. When developing such programs, designers need to possess knowledge of programming languages and a better understanding of human information processing capabilities (Lass and Gronau, 2020). All of these are useful in improving efficiency and effectiveness of programmer and computer software. It is important for them to know how individuals perceive screen color, some of the common patterns, how and why unambiguous icons should be constructed, errors that occur on the user's side and the role of user effectiveness is related to various mental model of systems possessed by people.

## 4.11. TYPES OF INTERACTIVE SYSTEMS

The start of interactive system was makes by command line systems. These systems facilitated controlled interactions between the computer and humans. For proper interactions, users were required know the commands that can be issued and how the arguments were to be ordered. Some of the classical examples include the Disk OS and UNIX OSs. While using the command line systems the users were to enter data in particular sequence. High control was also present with regards to data output. Output was generally limited. The command line system placed high demands on the need for the user to remember commands as well as the syntax used in issuing the commands (Figure 4.9).


Knoppix511 - VMware Workstation
File Edit View VM Team Windows Help

```
root!tty1:/# nmap -n -oG -PA -T5 -p 1-512 192.168.1.34/26

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-11-15 02:17 EST
Warning: Giving up on port early because retransmission cap hit.
Interesting ports on 192.168.1.33:
Not shown: 507 filtered ports
PORT    STATE  SERVICE
20/tcp  closed ftp-data
21/tcp  closed ftp
root!tty1:/# nmap -n -oG -PA -T5 -p 1-512 192.168.1.34/26

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-11-15 02:18 EST
Warning: Giving up on port early because retransmission cap hit.
Interesting ports on 192.168.1.33:
Not shown: 507 filtered ports
PORT    STATE  SERVICE
20/tcp  closed ftp-data
21/tcp  closed ftp
23/tcp  closed telnet
80/tcp  open   http
443/tcp open   https
MAC Address: 00:1C:10:CF:25:B6 (Unknown)

All 512 scanned ports on 192.168.1.41 are filtered (281) or closed (231)
MAC Address: 00:03:2F:27:5B:38 (Global Sun Technology)

Interesting ports on 192.168.1.42:
Not shown: 511 closed ports
PORT   STATE SERVICE
```

**Figure 4.9.** Command line systems were the first kind of interactive system developed.

*Source: https://www.sciencedirect.com/topics/computer-science/command-line-interface.*

Challenges in the command line system led to the development of the second generation of menu, form, and dialog-based systems. This system eased some of the demands on memory. A good example of the form-based program is the automatic teller machine. It gave users a tight controlled set of possible actions. Form-or dialog-oriented systems include data entry system which offered the user a limited set of choices while ensuring the memory is relieving the memory of certain demands) (Lange et al., 2011).

This was followed by the adoption of third generation interactive computing that was introduced in 1980 by Xerox Corporation. Half dozen years of research led to the development of the Xerox Star. During its development, bit-mapped displays, Windows, the desktop metaphor, icons, and the mouse were brought together and made to function. Xerox Star was later on replicated in Macintosh and the Lisa offered by Apple Computer Inc. This was during mid-1980s. Microsoft made Windows, icon, pointer, and menu universal. In the 1990s, the Windows family of OS was introduced. This led to the shift in graphical user interface also known as WIMP interfaces from command-based to direct manipulation. The command-based system placed much emphasis on users specifying an action as well as the object on which the action is to be performed. This is very different from the direct manipulation system when by an object is selected while the user specifies the action to be performed on the object. Currently, most developments in interactive systems have placed much focus in agents, visualization, and virtualization. The nature of the current generation of direct manipulation system as well as the coming generation of virtual and agent systems.

When developing interactive system, a major component is for developers to understand human capabilities. There is great importance for users to known how to properly utilize highly interactive computer system. There are different ways of describing human-computer interaction and Professor Donald A. Norman describes human-computer interaction in terms of gulfs of execution and evaluation. With this kind of understanding, the user has a goal in mind and is required to reformulate their goal in term of a plan ultimately involving the execution of series in actions on the system. The actions cause changes to the state of the system (Krohn and Tromer, 2009). The user then perceives, interprets, and evaluates them. Among the most important elements in computer system developers possessing the ability to understand how human beings perceive, interpret, evaluate, and respond to computer actions.

## 4.12. REAL TIME SYSTEMS

Real time systems can be defined as an information processing system having both software and hardware components able to perform real-time application functions. They also have the ability to respond to event within specific time and predictable constraints. Autonomous driving systems, process control systems and driving control systems are some examples of real time systems (Figure 4.10).

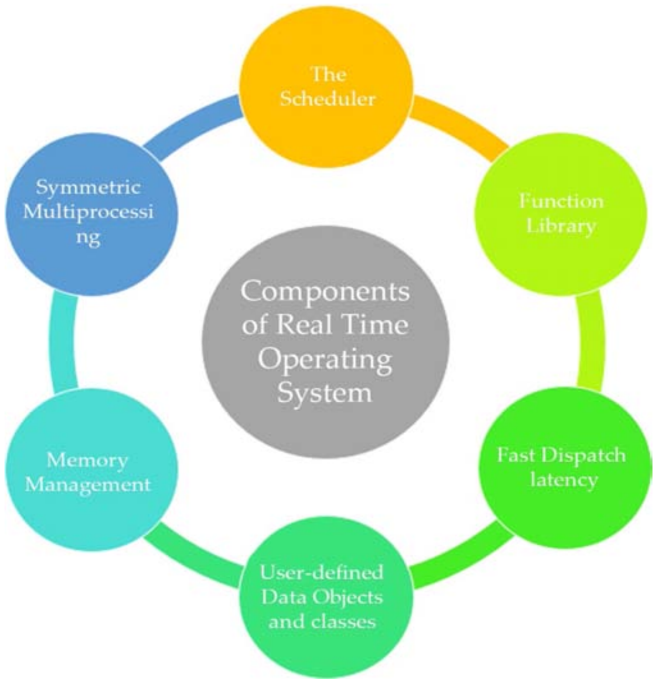

Components of Real Time Operating System
The Scheduler
Function Library
Fast Dispatch latency
User-defined Data Objects and classes
Memory Management
Symmetric Multiprocessi ng

**Figure 4.10.** Real time systems are information processing systems.

*Source: https://www.guru99.com/real-time-operating-system.html.*

### 4.12.1. Advantages of Real Time Systems

There are numerous benefits of real time in applications. They are as follows:

- **Precise Timing:** Real time systems have been adequately designers to perform tasks that should be executed within precise cycle deadlines. These deadlines can go down to microseconds.
- **High Reliability and Predictability:** Real time systems function within predictable time frames. This means that execution of

tasks is practically guaranteed making such systems very reliable in businesses. This is also achieved because such systems tend to use defined data (Kushwaha and Kushwaha, 2011).

- **Prioritization of Real Time Workloads:** Real time workloads are usually completed within a given deadline to ensure that critical systems do not fail. This is because real time systems are able to prioritize some workloads over others. There are some real time systems known to have task or workload prioritization.

## 4.12.2. Components of Real Time Systems

There are two main requirements to be met for real time systems to perform real time computing. They include timelessness and time synchronization. Timelessness defines the ability to produce expected result within a given time frame while time synchronization defines agent's ability to coordinate independent clocks and operating together as one. Real time system is evaluated by measuring the value of a system with regards to their predictability in completing tasks or events (Jeong et al., 2012). There are two main elements used when further evaluating real time system. They include compute jitter and latency. Compute jitter is the latency variation between iterations while latency is the measurement of time between two events. There are two main types of real time systems namely the hard real time software and the soft real time software.

Soft or hard real time system are used by companies depending of the tasks at hand. The goal is to help the company meet their growing needs for real-time data processing taking advantage of the predictability and reliability of the real-time system. Some of the companies include Intel. It provides solutions, technologies, and partners to achieve such systems. For this reason, a real time system is characterized by its ability in producing expected results within a given deadline as well as their ability to coordinate clocks and time synchronization. Hard real time systems make use of absolute deadlines and when the allotted time spans are not met, a system failure occurs. This is very different form soft real time systems as the system will continue to function even if there is a missing deadline. However, output quality of soft real time system is very low.

Intel has been useful as it offers hardware and reference system-level software needed in developing real time applications. For such applications, every element will be able to perform in a predictable and reliable manner within a specific time window and therefore able to meet hard real time requirements.

### 4.12.3. Need for Real Time Systems

There has been a growing needed for real time system brought about by the growing global connectivity, changing consumer demands for readily available data as well as on sensor-enabled enterprise environments that drive the creation, collection, and analysis of exponential data amounts. It is anticipated that by 2025, there will be about 79.41 zettabytes of data created of which 30% of it will need real time processing facilitated by real time systems. There is also a great need for real time processing in businesses such as robotics, high-precision industries, healthcare, and manufacturing (Jaeger, 2008). All these industries greatly rely on real time data in ensuring continuous improvement in reliability, efficiency, and safety. For a company, its ability to manage, prioritize, and execute real-time workloads over non-real time workloads is useful in ensuring that data is processed in real-time for business in various industries.

A good illustration is in the modern automotive manufactures known to be heavily reliant on the working together of robots in a production line when assembling a car. While in the production line, robots will pass each other weld, drill or parts. They are able to perform safety inspections. These kinds of inspections require a high level of meticulous timing and precision. In such a case, it is very important for real time system must have the ability to process data in a predictable and defined time frame and also ensure that critical systems are completed prior to less critical tasks. Real time systems also ensure that data driven industries are able to process data.

## 4.13. APPLICATIONS OF REAL TIME SYSTEMS

### 4.13.1. Machine Vision

Machines are able to perform rapid interpret data through the use of machine vision. It enables machines to see their surrounding and quickly make decisions according to visual input. Machines with vision are key to ensuring flow of production or the continuation of critical processes. Real time system has been helpful in ensuring that such machines are able to process data near real time.

### 4.13.2. Process Control Systems

Industrial applications greatly make use of process control systems. In such industries, production is continuous and there are minimal interruptions. Process control system help business improve performance and maintain quality by testing processes, collecting relevant data, and returning the data for monitoring and also troubleshooting (Irwansyah et al., 2018). Process control systems are mostly used by companies in the gas and oil sector. These companies are known to realize numerous benefits, increased efficiency to safer operation of facilities and also reduced losses.

## 4.14. SPOOLING

Spooling as a property of the OS is a process in which data is temporarily held to be used and executed by a system, program, or a device. In spooling, data is usually sent to a memory or another volatile storage where it is stored until the computer or program requests for its execution. An acronym of simultaneous peripheral operations online is SPOOL. The computer's physical memory maintains the spool. It is also maintained on I/O device specific interrupts or buffers. Processing of spool is done in the ascending order as it works on a first-in, first-out algorithm. Spooling may therefore be defined as the process of putting data of various I/O jobs in a buffer. The buffer is a special area in memory or hard disk that can be accessed by I/O devices. Some of the activities done by the OS related to the distributed environment include handling of I/O device data spooling as devices have different data access rates (Hellmund, 2016). The OS also maintains spooling buffer providing a waiting station where data rests as slower devices catch up. OSs also maintain parallel computation. This is because spooling process can be performed by a computer whose I/O is performed in parallel order. In such kind of an operation, the computer is able to read data from a tape, write out to a tape printer and write data to a disk while it performs a computing task (Figure 4.11).

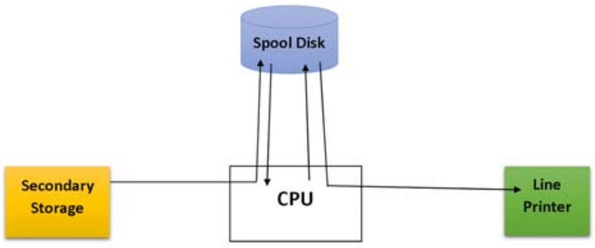


**Figure 4.11.** Spooling requires the creation of spools to manage data.

*Source: https://www.notesjam.com/2017/10/spooling-in-operating-system.html.*

## 4.15. WORKING OF SPOOLING IN OPERATING SYSTEM (OS)

The following steps are followed by the OS in spooling. A buffer called SPOOL is created and is used to hold off jobs and data till the device in which SPOOL is created is able to use and execute the job and also operate on the data. A secondary memory attached as a SPOOL buffer is used when a faster device sends data to a slower device to perform some operation. Data is retained in the SPOOL till the slower device is ready to operate on the data. Once the slower device is ready, data in the SPOOL is loaded on to the main memory for required operations. The next step involves spooling considering the entire secondary memory as a huge buffer with the ability to store data and several jobs for operations. Among the advantages of Spooling is that it is able to create a queue of jobs that execute in FIFO order executing jobs one by one. In the next step, the device is able to connect too many input devices that may require operation on their data (Høiland-Jørgensen, 2018). This means that the input devices can put their data onto the secondary memory and can also be executed one by one on the device. This ensures that the CPU is not left idle at any moment. One can therefore conclude that Spooling is a combination of queuing and buffering. The final step involves the CPU generating some output. The output is first saved in the main memory. The output is then transferred to the secondary memory from the main memory. The output finally finds its way to the respective put devices.

### 4.15.1. Examples of Spooling

Among the most common examples of spooling is printing. When a document is to be printed, it is stored in a spool and added to the queue for printing. At this time, there are numerous processes that can perform their operations. They also use the CPU without waiting. The printer executes the printing process on documents one-by-one. In OS there are over several features that can be added to the Spooling printing process. These features include notification or setting priorities when the printing process has been completed. It could also include the selection of different types of paper to print on.

### 4.15.2. Advantages of Spooling

There are numerous advantages of Spooling. They are as follows. With Spooling, the number of I/O devices or operations does not matter. In such a setup, there are sever I/O devices working together simultaneously without any interference to each other. There is so no interaction between the CPU and I/I devices. This e means that the CPU does not have to wait for the I/O operations to take place. Such kinds of operations take longer durations to finish executing and in such a case, the COY does not have to wait for them to finish. The other advantage of Spooling is that the CPU is kept busy for the most time (Hambarde et al., 2014). The only time it goes to the idle state when the queue is exhausted. This is a very important element of the CPU because and idle COU is not very efficient. Several protocols were created to effectively utilize the CPU in the least amount of time. In Spooling, all the tasks are added to the queue and the CPU completes the tasks before it goes into an idle state. Another advantage is that Spooling allows applications to run at the speed of the CPU as the I/O devices run at their respective full speeds.

### 4.15.3. Disadvantages of Spooling

Some disadvantages of spooling are as follows. While using spooling, large amounts of storage is required depending on the number of requests made by the input as well as the number of input devices connected. Spooling mostly involves the use of a secondary storage as the Spool is created in it. If there are several input devices working simultaneously may take up lots of space on the secondary storage. This causes an increase in disk traffic. The disk gets slower and traffic increases. Spooling is mostly used when copying and executing data from slower devices to faster devices. A spool is created in the slower device to store data to be operated upon in a queue and the CPU works on it. The process makes Spooling futile to use in real time environments where real time results are needed from the CPU. The delay causes output be produced at a later time than the real time.

CHAPTER 5

# OPERATING SYSTEM – PROCESSES

## CONTENTS

A program is useless until the instructions it contains are executed by a CPU. A process is a program that is currently running. Processes require computer resources to do their tasks.

There could be multiple processes in the system that need the same resource at the same time. As a result, the operating system (OS) must efficiently and effectively handle all processes and resources (Giorgetti et al., 2020).

To preserve consistency, some resources might have to be executed by one process at a time or else, the system may become incompatible and a deadlock may develop. In terms of Process Management, the OS is in charge of the following tasks.

## 5.1. PROCESS

A process is basically what one would think as a running software. The execution of a has to be carried out in a particular order.

A process is an entity that represents the very basic unit of work that must be implemented in the system. In other words, we write our computer programs in a text file, and when they are run, they turn into a process that completes all of the duties specified in the program.

A program can be separated into four pieces when it is put into memory turns into a process: stack, heap, text, and data. The diagram in Figure 5.1 depicts a basic representation of a process in main memory.

- **Stack:** This contains the temporary data such as method/function parameters, return address and local variables.
- **Heap:** This is dynamically allocated memory to a process during its run time.
- **Text:** This includes the current activity represented by the value of program counter and the contents of the processor's registers.
- **Data:** This section contains the global and static variables.

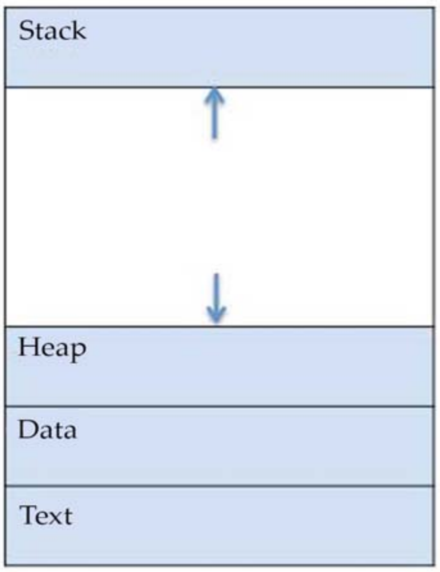


**Figure 5.1.** Components of a program.

*Source: https://www.tutorialspoint.com/operating_system/os_processes.htm.*

## 5.2. KERNELS

The kernel is a piece of software that operates at the core of a computer's OS and controls all of the system parts. It's a piece of OS code that's always in memory and aids in the interaction of hardware and software. A full kernel manages all hardware resources (like I/O and memory) through device drivers, resolves resource conflicts between processes, and optimizes the usage of shared resources including CPU and cache use, file systems, and network sockets. When most computers turn on, the kernel is among the earliest programs to run (Gunadi and Tiu, 2014).

It handles the remainder of the starting process, as well as requests for input/output (I/O) from memory, peripherals, and software, and converts them into data-processing instructions for the CPU.

Kernel code is usually loaded into a distinct area of memory that is isolated from application software and other less critical OS components. The kernel executes operations such as operating programs, controlling hardware devices such as the hard disk, and handling interrupts in this protected kernel area. Web applications, word processing, and interactive multimedia players, on the other end, employ user space, which is a separate memory area. This method avoids user data and kernel data from interacting and causing issues and delay, and also faulty programs from interfering with other applications or crashing the whole OS. Even on systems where the kernel is contained in application address spaces, memory protection is employed to stop unauthorized apps from modifying the kernel.

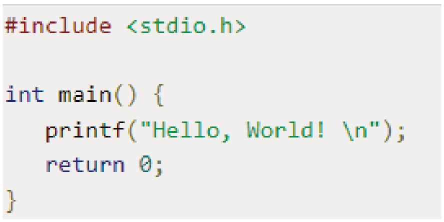
The kernel's interface is a low-level abstraction layer. Whenever a process requests a service from the kernel, it must issue a system call, which is typically done with the help of a wrapper function.

There are a variety of kernel structure designs to choose from. Monolithic kernels, which are intended for speed, run exclusively in a single address space with the CPU in supervisor mode. Microkernels, like user processes, run the majority of their services in user space, but even so not all of them, chiefly for resilience and modularity. The microkernel architecture of MINIX 3 is a good example. While the Linux kernel is monolithic, it is also modular in that it may install and unload loadable kernel modules at runtime (Greenwald and Thomas, 2007).

The execution of programs is the responsibility of this essential element of a computer system. The kernel is in charge of choosing which of the numerous running applications should be assigned to the processor or processors at any one time.

## 5.3. PROGRAM

A program is a collection of code that can be as simple as a single line or as complex as millions of lines. A computer program is usually developed in a programming language by a programmer. Here's a small program created in the C programming language as an illustration (Figure 5.2).



```
#include <stdio.h>

int main() {
   printf("Hello, World! \n");
   return 0;
}
```

**Figure 5.2.** Programming language.

*Source: https://www.tutorialspoint.com/operating_system/os_processes.htm.*

A computer program is a set of instructions that, when processed by a computer, perform a certain purpose. We can deduce that a process is a dynamic form of a computer program when we relate a program to a process (Arshad et al., 2018).

An algorithm is an element of a computer program that executes a certain task. A software package is a bundle of computer programs, libraries, and associated data.

## 5.4. LIFE CYCLE OF THE PROCESS

When a process runs, it goes through many states. Distinct OSs have different stages, and the terminology of these states are not uniform (Figure 5.3).

In principle, a process can be in one of the five states listed below at any given time.

| | |
|---|---|
| **1.** | **Start:** This is the first or primary state when a process is first started or created. |
| **2.** | **Ready:** The procedure is awaiting the assignment of a processor. Processes that are ready to run are waiting for the OS to allocate them a processor. The process may acquire this state after it has started or as it is operating, but the scheduler can suspend it so that the CPU resources can be allocated to other prioritized process. |
| **3.** | **Running:** Depending on the scheduling method, the OS will select one of the processes from the ready state. As a consequence, if our system only has a single CPU, there'll always be just the one process operating at any given time. We can have a specified number of processes running at the same time if the system has more processors (Fierro and Culler, 2015). |
| **4.** | **Waiting:** Based on the scheduling method or the underlying nature of the process, a process can migrate from the Running state to the Block or Wait states.<br>When a process waits for a resource to be allotted or for user input, the OS puts it in a block or wait state and gives the CPU resources to other processes. |
| **5.** | **Terminated or Exit:** The process is moved to the terminated state, where it waits to be removed from main memory, once it has carried out it its execution processes or been entirely stopped by the OS. |

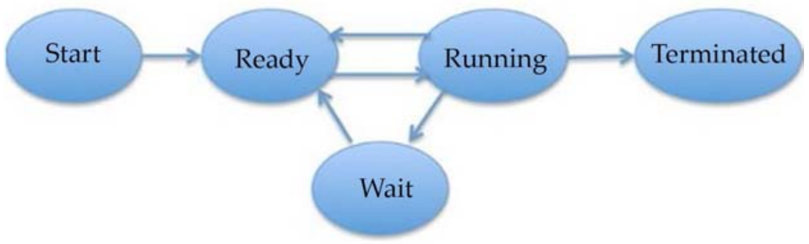


**Figure 5.3.** Process life cycle.

*Source: https://www.tutorialspoint.com/operating_system/os_processes.htm.*

## 5.5. PROCESS CONTROL BLOCK (PCB)

A process control block (PCB) is a data block that computer OSs use to store all process-related data. It's also known as a process description. When a process is created, the OS creates a PCB (initialized or installed). This tells you whether the process is started, ready, running, waiting, or finished.

In multitasking OSs, the PCB stores data that is essential for proper and successful process control. The major elements of these frameworks can be split into three groups; however, the specifics differ based on the system:

- Process identification;
- Process state; and
- Process control.

Status tables are there for every significant entity, like memory, I/O devices, files, and processes. Memory tables, for example, show how much main and secondary (virtual) memory every process has, as well as authorization settings for using memory areas used by multiple processes. I/O tables can include entries that describe a device's ability or assignment to a process, the state of I/O events, the placement of memory buffers used for them, etc. (Fröhlich and Wanner, 2008).

Process data for identification – a unique identifier for the process (mostly just an integer), as well as data such as the identifier of the main process, user identifier, user group identifier, and so forth, are contained in process identity information in a multiuser-multitasking system. The process ID is especially important because it is normally used to cross-reference the items or stages illustrated in tables above, e.g., displaying which process is occupying which I/O resources, or memory areas.

When a process is halted, process state data defines its current state, allowing the OS to restart it. Throughout a context switch, shift or transition, the currently executing process is terminated and replaced by a new one. The kernel must halt the current process's execution, duplicate the data in hardware registers to the fresh process's PCB, then refresh the hardware registers with the data from the new process's PCB (Androulaki et al., 2018).

Process control information is employed by the OS to handle the process as is. This uses components like:

- Process scheduling state – the status of a process, like "ready," "suspended," and some pertinent scheduling information, like the priority level and the time since the process used the services of the CPU or was suspended.
- Information about the process's children, or the ids of other processes that are functionally associated to the present one, which can be expressed as a queue or other data formats.
- Information about interprocess communication (IPC)–flags, signals, and messages linked with communication between independent processes.
- Process privileges – access to system resources allowed/ disallowed.
- Process state – new, ready, running, waiting, dead.
- Process identification number (PID) – each process has its own unique identification number (also known as Process ID).
- CPU registers – register organizes where process should be saved for execution for running state.
- Program counter (PC) – an indicator to the location of the next task to be executed for this process.
- Information on CPU scheduling – information on CPU scheduling.
- Information on memory management – page table, memory limitations, and segment table.
- Accounting data, such as the amount of CPU time spent on process execution, time constraints, and execution ID.
- I/O status information – a list of the process's I/O devices.

## 5.6. PROCESS SCHEDULING

### 5.6.1. Definition

Process scheduling is the process manager's job, and it entails removing a running process from the Central processing unit and picking another one based on established rules (Estefo et al., 2019).

Process scheduling is crucial in a Multiprogramming OS. In these kind of OSs, numerous processes can run in accessible memory at the same moment, and the current processes divide the CPU via temporal multiplexing (Figure 5.4).

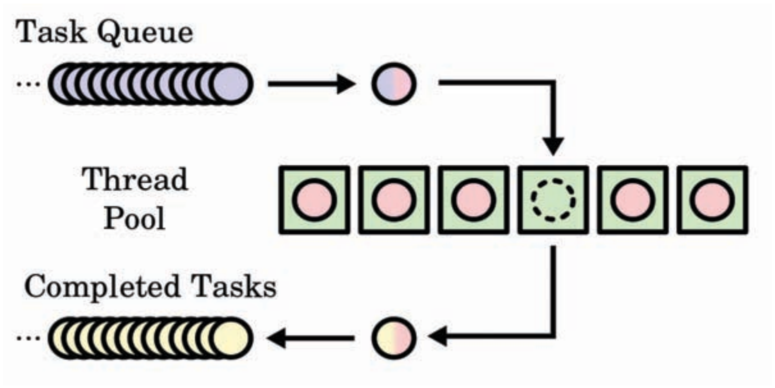


**Figure 5.4.** A sample thread pool (green boxes) with a queue (FIFO) of waiting tasks (blue) and a queue of completed tasks (yellow).

*Source: https://en.wikipedia.org/wiki/Scheduling_(computing)#/media/File:Thread_pool.svg.*

### 5.6.2. Process Scheduling Queues

The OS keeps all PCBs in Process Scheduling Queues. In the OS, every process state will have its own queue, and PCBs on all processes in the same execution stage are grouped together. When a process's status is altered, its PCB is withdrawn from its previous queue and moved to its new state queue.

The OS maintains the following key process scheduling queues:

- Every one of the system's processes are retained in the work queue (Ahmad et al., 2013).

- The ready queue keeps track of all the processes in main memory that are prepared to be run. New processes are constantly being added to this queue.
- Device queues are processes that are paused because an I/O device is unavailable (Figure 5.5).

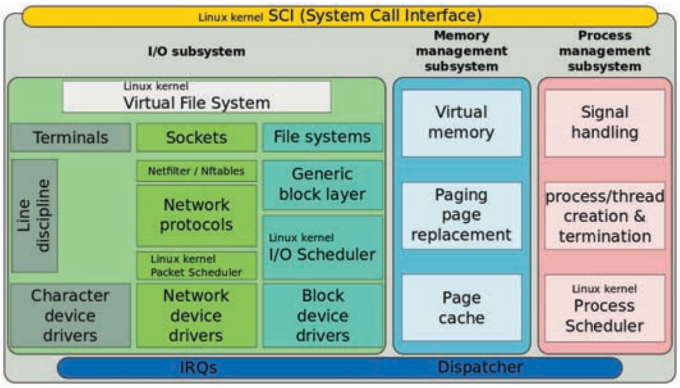


**Figure 5.5.** A basic organization of the Linux Kernel, that holds elements like process schedulers, I/O schedulers, and packet schedulers.

*Source: https://en.wikipedia.org/wiki/Scheduling_(computing)#/media/File:Simplified_Structure_of_the_Linux_Kernel.svg.*

## 5.7. TWO-STATE PROCESS MODEL

Two-state process model alludes to running and non-running states as shown below:

| SL. No. | State and Description |
|---|---|
| 1. | **Running:** When a new process is initiated, it is introduced into the system though the running state. |
| 2. | **Not Running:** Processes that aren't currently running are kept in a queue and will be executed later. Each queue entry corresponds to a specific process. A queue is implemented using a linked list. Here are some instances of how to make use of a dispatcher. A process is automatically transferred to the waiting list when it is stopped. Once a procedure has completed or failed, it is erased. In either case, the dispatcher selects a process from the queue to execute (Deseriis, 2017). |

## 5.8. SCHEDULERS

### 5.8.1. Process Scheduler

The process scheduler is an OS component that figures out which processes run at any given time. A preemptive scheduler can suspend an existing process, put it to the tail of the running queue, and initiate a new process; or else, it is a cooperative scheduler.

Depending on how frequently decisions must be performed, we discern between "long-term scheduling," "medium-term scheduling," and "short-term scheduling." Though others will be introduced below as well.

### 5.8.2. Long-Term Scheduling

The long-term scheduler or the admission scheduler, organize which jobs or processes are revealed to the ready queue; that is, when a program is attempted to be executed, the long-term scheduler can decide to authorize or delay its admission to the set of currently executing processes. As a result, this scheduler dictates which processes should run on a system, as well as the level of multitasking that should be supported at any given time – whether many or few processes should run at the same time, as well as how the divide between I/O-intensive and CPU-intensive processes should be handled. Controlling the level of multiprogramming is the job of the long-term scheduler.

Most processes can be classified as I/O-set or CPU-set in practice. An I/O-bound process is probably going to take longer time to execute I/O than performing computations. In contrary, a CPU-bound process creates fewer I/O demands and spends longer durations conducting computations. A proper process combination of I/O-bound and CPU-bound processes is critical for a long-term scheduler. The ready queue will nearly always be vacant if all processes are I/O-bound, and the short-term scheduler not have processes. If all processes are CPU-bound, on the other hand, the I/O waiting queue will nearly always be vacant, devices will go idle, and the system will be imbalanced once more (De et al., 2007). As a result, the system with the greatest performance will have a mix of CPU and I/O-bound tasks. This is utilized in current OSs to guarantee that real-time processes have enough CPU time to finish their duties (Figure 5.6).

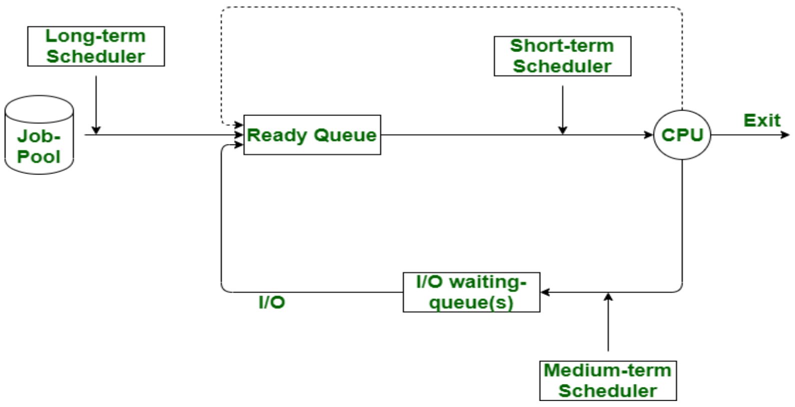


**Figure 5.6.** Difference between long-term, medium-term scheduler, and short-term scheduler.

*Source: https://www.geeksforgeeks.org/difference-between-long-term-and-medium-term-scheduler/.*

In large-scale systems like batch processing systems, computer clusters, supercomputers, and render farms, long-term scheduling is especially critical. Co-scheduling of interacting processes, for instance, is usually needed in concurrent systems to avoid them from blocking owing to waiting on one another. In these circumstances, on top of any basic admission scheduling functionality in the OS, special-purpose job scheduler software is generally utilized to help these functions.

Fresh tasks can only be introduced to some OSs if the present deadlines can still be fulfilled. The admission control mechanism is the specific heuristic technique employed by an OS to permit or reject new jobs.

## 5.9. MEDIUM-TERM SCHEDULING

The medium-term scheduler, like a hard disk drive, “swaps out” or “swaps in” applications from primary memory to secondary memory (also incorrectly as “paging out” or “paging in”)The medium-term scheduler may choose to swap out a process that has not been active for a long time, or a process with a lesser importance, or a process that is frequently page faulting, or a process that is using a large amount of memory in order to free up main memory for other processes, swapping the process back into play later when more memory is readily available, or when the process has been started and there are resources, in order to free up main memory for other processes (DiLuoffo et al., 2018).

By considering binaries as "swapped out processes" upon execution, the medium-term scheduler may now perform the functions of the long-term scheduler in several systems (those that permit mapping virtual address space to secondary storage other than the swap file). If a section of the binary is required, it can be swapped in on request, sometimes known as "lazy loading" or "demand paging."

## 5.10. SHORT-TERM SCHEDULING

After a clock interrupt, an I/O interrupt, an OS call, or a kind of signal, the short-term scheduler (commonly termed as the CPU scheduler) determines which of the ready, in-memory processes is to be run (assigned a CPU). As a result, the short-term scheduler must conduct scheduling decisions far more regularly than the long-term or mid-term schedulers – a decision must be made at the very least after each timeslot, and they are quite brief. This scheduler can be either preemptive (or "voluntary" or "co-operative"), at which point it can "force" processes off the CPU when it intends to assign that CPU to another process, and sometimes non-preemptive (also referred to as "voluntary" or "co-operative"), in which instance it cannot "force" processes off the CPU. A preemptive scheduler uses a programmable interval timer to trigger an interrupt handler in kernel mode, which performs the scheduling function (Figure 5.7).

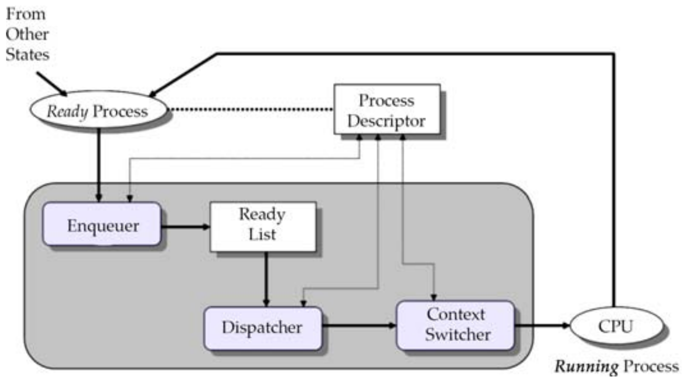


**Figure 5.7.** The main functional components of the scheduler.

*Source: http://faculty.salina.k-state.edu/tim/ossg/Process/scheduler/sched_parts.html.*

## 5.11. DISPATCHER

Another component involved in the CPU-scheduling function is the dispatcher, or module that transmits command of the CPU to the process picked by the short-term scheduler. It gets control in kernel mode as a response to an interrupt or system call. The following are the duties of a dispatcher:

- Context switches, where the dispatcher keeps the state (or a context) of the earlier operating process or thread; the dispatcher then imports the new process's starting or previously saved state (Dixon et al., 2012).
- Activating user mode.
- Navigating to the appropriate area within the user program in order to restart the program specified by its changed state.

Because it is called at each process changeover, the dispatcher should be as quick as possible. Because the processor is basically idle for a fraction of a second during context shifts, needless context switches should be minimized. The dispatch latency is the time taken for the dispatcher to halt one process and start another.

## 5.12. CONTEXT SWITCH

A context switch is a technique in the PCB that saves and recovers a CPU's state or context so that a process operation can be relaunched at a later time from the same point. This approach is used by a context switcher to allow multiple apps to share a single CPU. Context switching is a critical component of a multitasking OS.

When the scheduler switches the CPU from one process to another, the status of the presently executing process is stored in the PCB. The subsequent process's state is then imported from its appropriate PCB and used to setup the PC, registers, etc. The second procedure can then be started (Figure 5.8).

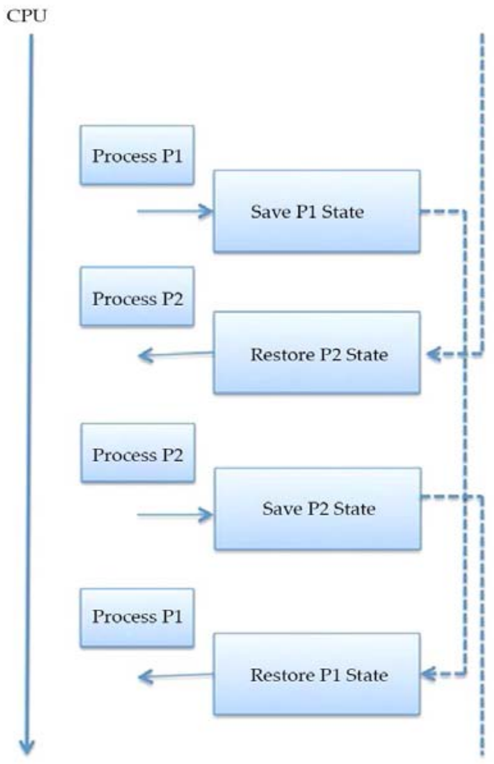


**Figure 5.8.** Context switch.

*Source: https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm.*

Because register and memory state must be preserved and recovered, context switches are computationally intensive. Some hardware systems use two or more types of processor registers to save context switching time. The data is saved for further use when the process is switched.

The term "context switch" has a span of meanings. It alludes to the process of saving the system state for one task so that it can be halted and another task restarted in a multitasking scenario. An interrupt could create a context switch, like when a process asks for disk storage, freeing up CPU time for other operations (Dieber et al., 2017).

To switch between user and kernel mode operations, several OS needs a context switch. Context switching could be a detriment to the performance of the system.

Context switches are typically computationally costly, therefore optimizing the use of context switches is a big part of OS architecture. Switching from one process to another necessitates some administrative time, such as saving and loading registers and memory mappings, updating different tables and lists, and so on. The complexity of a context switch is determined by the structures, OSs, and quantity of shared resources.

Switching registers, stack pointer, program counter, flushing the translation lookaside buffer, and loading the page table of the next process to start are all examples of context switching in the Linux kernel.

Additionally, equivalent context switching occurs across user threads, particularly green threads, and is normally a straightforward affair, preserving, and moving minimal context. A context switch is analogous to a coroutine yield in extreme instances, like switching between goroutines in Go, that is a little more costly than a subroutine call. There are three possible ways a context switch can be prompted which are discussed in subsections.

### 5.12.1. Multitasking

Typically, as part of a scheduling mechanism, one process must be turned off the CPU in order for another to execute. When a process is unable to run, this context switch is triggered. A pre-emptive multitasking system's scheduler could optionally swap processes that are still running (Chinetha et al., 2015). To prevent other processes from being deprived of CPU time, pre-emptive schedulers commonly configure a timer interrupt to take action when a job exceeds its time slice. This interrupt ensures that the scheduler can regulate the context switch.

### 5.12.2. Interrupt Handling

In today's architectures, interrupt-driven designs are widespread. This means that if the CPU wants data from a disk, it wouldn't have to wait until the read is complete before issuing the request; instead, it can submit the request and go on to another task. When the read is complete, the CPU can be interrupted (in this case, by sending an interrupt request from hardware) and the read delivered to it. For interruptions, an interrupt handler application is installed, and it is the interrupt handler that manages the disk interrupt.

When an interrupt occurs, the hardware automatically moves a piece of the context. The handler may save additional context depending on the unique hardware and software architectures. To limit the amount of time spent dealing with the interrupt, only a tiny section of the context is updated on a regular basis. The kernel does not begin or schedule a separate process to manage interrupts; rather, the handler executes in the context generated at the beginning of interrupt handling. When interrupt servicing is completed, the context that existed prior to the interrupt is restarted, enabling the interrupted process to continue in its proper state.

### 5.12.3. User and Kernel Mode Switching

A context switch is not required when the system switches from user mode to kernel mode; a mode transition is not a context switch in and of itself. However, a context switch may occur at this time, depending on the OS.

## 5.13. PROGRAM COUNTER (PC)

The program counter (PC) or instruction pointer in Intel x86 and Itanium microprocessors and sometimes referred to as the instruction address register (IAR), is a processor register that indicates where the computer is in its program sequence.

The PC is generally increased after receiving an instruction, and it now contains the memory address of the next instruction to be performed (Cashmore et al., 2015).

Control transfer instructions change the sequence by putting a new value in the PC, but processors typically retrieve instructions from memory continually. These include branches (also called jumps), subroutine calls, and returns. A transfer that is contingent on the truth of a statement allows the computer to follow a different sequence in different conditions (Figure 5.9).

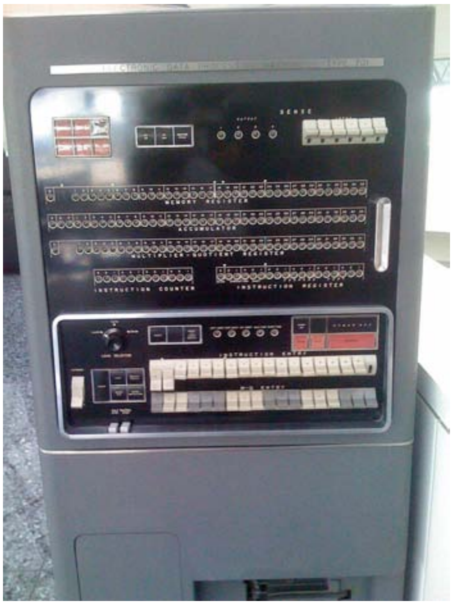


**Figure 5.9.** The front panel of an IBM 701 computer, which was debuted in 1952. The contents of numerous registers are displayed by lights in the center. The instruction counter is located in the lower left corner.

*Source: https://en.wikipedia.org/wiki/Program_counter#/media/File:IBM_701console.jpg.*

The following instruction is picked up from elsewhere in memory when a branch is used. A subroutine calls not only branches, but also preserves the PC's previous contents. The saved contents of the PC are retrieved and returned to the PC, and sequential execution resumes with the instruction following the subroutine call:

- Information regarding scheduling;
- Values for the base and limit registers.

The base registers specify where the page table begins in memory (physical or logical addresses), whereas the limit register specifies the table's side. Normally, the registers are not loaded directly. In most cases, these values are written to the hardware process context block (PCB).

- Currently used register;
- Changed state;
- I/O State information.

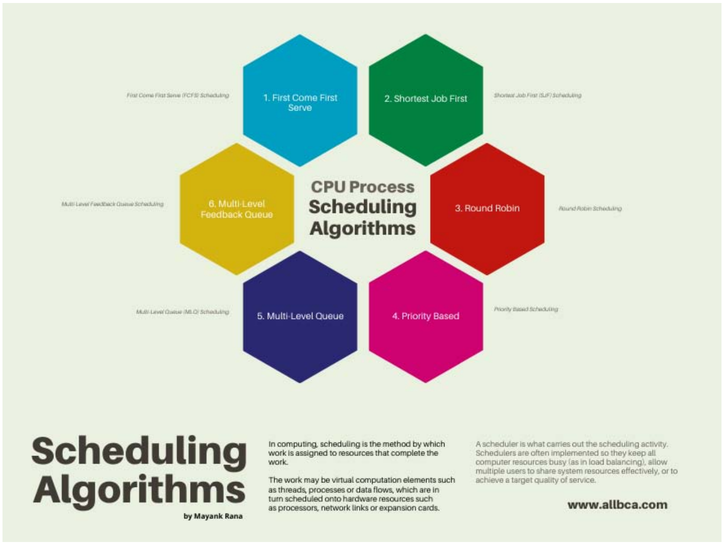
I/O status information tells you which I/O devices should be assigned to a specific process. It also shows which files are open, as well as the state of the I/O device. It describes the data input and output from the input device to the register. It describes the flow of data from the input device to the memory (Cao et al., 2008).

These bits indicate if the current instruction has been finished, whether a byte may be retrieved from the data-in register, and whether a device fault has occurred.

## 5.14. SCHEDULING ALGORITHMS

There are six popular process scheduling algorithms (Figure 5.10):

- First-come, first-served (FCFS) scheduling;
- Shortest-job-next (SJN) scheduling;
- Priority scheduling;
- Shortest remaining time;
- Round robin (RR) scheduling;
- Multiple-level queues scheduling.


First Come First Serve (FCFS) Scheduling
1. First Come First Serve
2. Shortest Job First
Shortest Job First (SJF) Scheduling
Multi Level Feedback Queue Scheduling
6. Multi-Level Feedback Queue
CPU Process Scheduling Algorithms
3. Round Robin
Round Robin Scheduling
Multi Level Queue (MLQ) Scheduling
5. Multi-Level Queue
4. Priority Based
Priority Based Scheduling
Scheduling Algorithms
by Mayank Rana
www.allbca.com

In computing, scheduling is the method by which work is assigned to resources that complete the work.

The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in load balancing), allow multiple users to share system resources effectively, or to achieve a target quality of service.

**Figure 5.10.** CPU process scheduling algorithms in OS.

*Source: https://www.allbca.com/2020/04/cpu-process-scheduling-algorithms-in-os.html.*

## 5.15. SCHEDULING ALGORITHMS/DISCIPLINES

A scheduling discipline or a scheduling algorithm is an algorithm for allocating resources to parties who request them in a synchronized and simultaneous manner. Routers (to manage packet traffic), OS (to split CPU resources among both threads and processes), disk drives, printers, and most embedded systems all use scheduling disciplines.

The main objective of scheduling algorithms is to lower resource use inefficiency and bring in equal among the programs who use the resources. The difficulty of selecting which one of the waiting requests should be allocated the resources is handles with or by scheduling. There are numerous scheduling algorithms to choose from. We'll go through a few of them in this segment (Bhattacharjee and Lustig, 2017).

The concept of a scheduling algorithm is employed as an alternative to first-come, first-served queuing of data packets in packet-switched computer networks and other statistical multiplexing.

First-come, first-served (FCFS) scheduling, shortest-job-next (SJN) scheduling, priority scheduling, shortest remaining time, and RR scheduling re some of the most common scheduling algorithms out there. Weighted fair queuing may be used if a distinct or assured level of service is provided rather than best-effort communication. Channel-dependent scheduling can be used to enjoy the benefits of channel state information in sophisticated packet radio wireless network. The capacity and system spectral efficiency could well be boosted if the channel conditions are favorable.

### 5.15.1. First Come, First Served (FCFS)

The simplest scheduling technique is first in, first out (FIFO), or first come, first served (FCFS). Processes are queued in the order they land in the ready queue with FIFO. FIFO is a method for arranging the manipulation of a data structure (typically, especially, a data buffer) where the oldest item, or "head" of the queue, is processed first in computing and systems theory. This approach is comparable to serving persons in a queuing area on a first-come, first-served basis.

The FIFO OS scheduling technique, which gives each CPU time in the sequence in which it is required, is also known as FCFS. LIFO, or last-in-first-out, is the inverse of FIFO, in which the newest entry, or "head of the stack," is processed first. Although a priority queue is neither FIFO nor LIFO, it may behave in a similar manner briefly or by default. These techniques for processing data structures, along with interactions between strict-FIFO queues, are covered by queueing theory (Blackham et al., 2011).

Context changes occur only when a process terminates, and there is no need to reorganize the process queue, therefore scheduling overhead is low.

Because long operations might hog the CPU, throughput can be low, requiring small processes to wait a long period. There is no starvation because since process is given a chance to run after a set amount of time.

Turnaround, waiting, and response times are all affected by the sequence in which they arrive and might be lengthy for the same reasons. As a result of the lack of prioritizing, this system struggles to achieve process deadlines.

Because there is no prioritizing, there is no famine as long as all processes are completed eventually. There can be famine in an environment when some operations aren't completed. It is based on the concept of queuing (Figure 5.11).

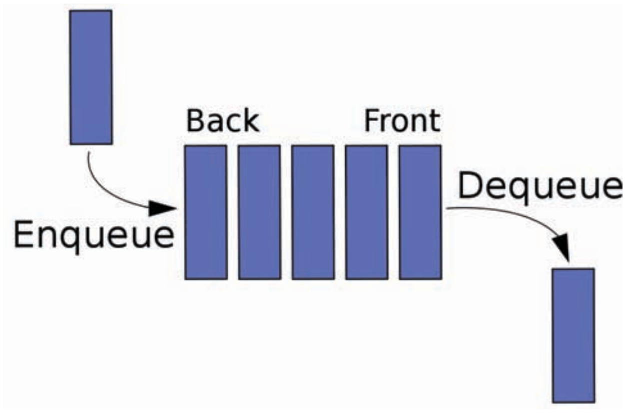


**Figure 5.11.** Representation of a FIFO queue.

*Source: https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)#/media/File:Data_Queue.svg.*

## 5.16. PRIORITY SCHEDULING

The earliest deadline first (EDF) or least time to go algorithm is a dynamic scheduling strategy used in real-time OSs to prioritize tasks. When a scheduling event occurs (e.g., a task is completed, a new job is released), the queue is searched for the process that is closest to its deadline, and that process is the next to be scheduled for execution.

The EDF method is a dynamic priority scheduling strategy used in real-time OSs to position processes in a priority queue. Every time a scheduling event occurs, the queue will be assessed for the process that is closest to its deadline (task completion, new task release, etc.). The next procedure to be scheduled for execution is this one (Bala et al., 2015).

EDF is a preemptive uniprocessor-optimized scheduling algorithm in the sense that if a catalog of independent jobs, each distinguished by an arrival time, an execution requirement, and a deadline, can be scheduled (by any algorithm) in such a way that all jobs complete by their deadline, the EDF will schedule this catalog of jobs to ensure that they all finish by their deadline.

When scheduling periodic processes with deadlines corresponding to their durations, EDF has a bound of 100 percent utilization. That example, as long as the total CPU use does not exceed 100 percent, EDF can guarantee that all deadlines are met. At greater loads, EDF can ensure all system deadlines more reliably than set priority scheduling methods such as rate-monotonic scheduling.

EDF is also an optimal scheduling method on non-preemptive uniprocessors, but only among scheduling strategies that do not allow inserted inactive time.

If the system is overburdened, however, the number of processes that miss deadlines is mostly unpredictable. For a real-time systems designer, this is a significant disadvantage. The algorithm is also tough to design in hardware, and the representation of deadlines in different ranges is problematic. If future relative deadlines are calculated using modular arithmetic, the field containing the future relative deadline must support at least the value of the "duration." As a result, EDF is rarely used in industrial real-time computer systems.

Most real-time computer systems, on the other hand, adopt set priority scheduling (usually rate-monotonic scheduling). With defined priorities, it's straightforward to foresee that under stress situations, low-priority processes will miss deadlines while the highest-priority processes will still make it.

In real-time computing, there is a substantial body of research on EDF scheduling; it is feasible to determine worst-case response times of processes in EDF, to handle processes other than periodic processes, and to employ servers to limit overloads.

## 5.17. SHORTEST REMAINING TIME FIRST

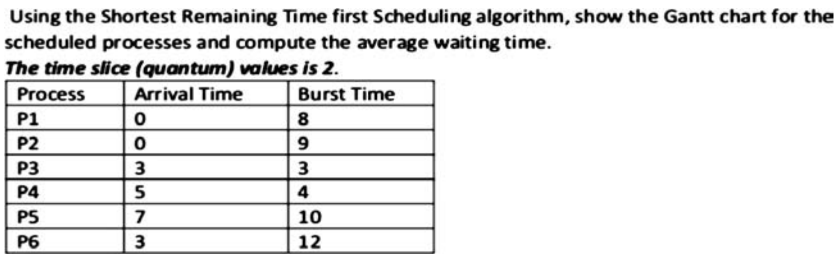
In the same way that the shortest job is completed first (SJF). The scheduler uses this method to place processes in the queue that have the shortest predicted processing time remaining. This necessitates in-depth knowledge or estimates of how long a procedure will take to complete.

When a shorter process comes during the execution of another, the current process is stopped breaking it into two independent computing blocks. This adds to the overhead by requiring more context shifts. In addition, the scheduler has to place every incoming task to a precise position in the queue, which adds to the overhead.

In most circumstances, this method is designed for optimal throughput. As the computational requirements of the operation increase, the waiting time and response time enhance as well. Because turnaround time is calculated as the sum of waiting time and processing time, it has a substantial impact on longer processes. However, because no process has to wait for the completion of the longest process, the overall waiting time is less than FIFO (Aksoy et al., 2017).

Deadlines aren't given much thought, and the programmer may only try to make processes with deadlines as brief as feasible.

In a busy system with numerous little processes running, starvation is a possibility. We should run at least two processes with different priorities in order to use this policy (Figure 5.12).



**Using the Shortest Remaining Time first Scheduling algorithm, show the Gantt chart for the scheduled processes and compute the average waiting time.**

***The time slice (quantum) values is 2.***

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 0 | 9 |
| P3 | 3 | 3 |
| P4 | 5 | 4 |
| P5 | 7 | 10 |
| P6 | 3 | 12 |

**Figure 5.12.** SRTS.

*Source: https://www.chegg.com/homework-help/questions-and-answers/using-shortest-remaining-time-first-scheduling-algorithm-show-gantt-chart-scheduled-proces-q25910377.*

## 5.18. FIXED PRIORITY PRE-EMPTIVE SCHEDULING

Fixed-priority preemptive scheduling is a scheduling strategy that is frequently employed in real-time systems. With fixed priority preemptive scheduling, the scheduler ensures that the processor executes the highest priority job of all those tasks that are currently ready to execute at any given time.

The preemptive scheduler has a clock interrupt job that can provide the scheduler choices to switch once the task has been running for a certain amount of time (the time slice). This scheduling approach has the benefit of ensuring that no task consumes the CPU for longer than the time slice. This scheduling approach, however, is vulnerable to process or thread lockout: because higher-priority tasks are prioritized, lower-priority processes may have to wait an unlimited length of time (Aksoy et al., 2017). Aging is a typical approach of resolving this scenario, as it gradually raises the priority of waiting processes and threads, guaranteeing that they all execute at some point. Preemptive schedulers are present in most real-time operating systems (RTOSs). You can also get the non-preemptive RTOS by turning off time slicing.

Preemptive scheduling is frequently distinguished from cooperative scheduling that allows a task to execute completely from start to finish without being interrupted by other processes. The job must specifically call the scheduler to have a task swap. A few RTOS, such as Salvo or TinyOS, use cooperative scheduling. Every process has a defined priority rank assigned by the OS, and the scheduler organizes the processes in the ready queue in order of priority. Incoming higher-priority processes disrupt lower-priority processes:

- Overhead is neither minor nor significant;
- In terms of throughput, FPPS has no advantage over FIFO scheduling;
- It can be described as a group of FIFO queues, one per priority ranking, if the range of rankings is restricted. Only when all of the higher-priority queues are empty are processes in lower-priority queues picked (Aksoy et al., 2017);
- The amount of time spent waiting and responding is determined by the process's priority. Waiting and response times are shorter for higher-priority tasks;

- Deadlines can be met by prioritizing procedures that have deadlines;
- With a large number of high-priority processes queuing for CPU time, lower-priority processes may be starved.

## 5.19. ROUND-ROBIN SCHEDULING

Round-robin (RR) is yet another type of algorithms employed by process and network schedulers in computing. Time slices are distributed in equal amounts to each process in a circular fashion, managing all processes without regard to priority, as the term is typically used (also known as cyclic executive). Round-robin scheduling is simple, uncomplicated, and devoid of famine. Round-robin scheduling can be used to handle other scheduling challenges, such as data packet scheduling in computer networks. It's a concept for a computer OS (Yin et al., 2016).

The algorithm's name is derived from the round-robin principle, which is used in various disciplines to ensure that everyone gets an equal share of something (Figure 5.13).

- The preemptive process scheduling algorithm is known as Round Robin (RR);
- Each process is given a fixed amount of time to complete, which is referred to as a quantum;
- After a process has run for a set amount of time, it is preempted and another process runs for the same amount of time;
- Preempted processes' states are saved using context switching.

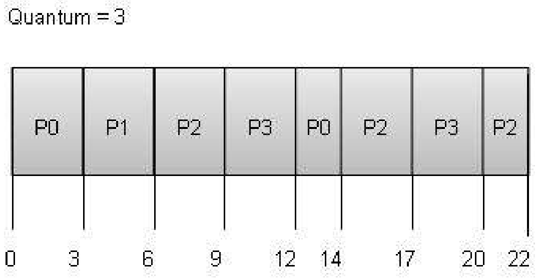


**Figure 5.13.** Round robin scheduling.

*Source: https://www.tutorialspoint.com/operating_system/os_processes.htm.*

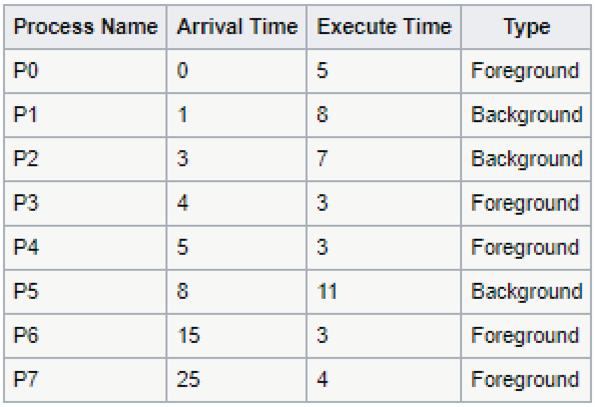
Wait time of each process is as follows:

| **Process** | **Wait Time: Service Time – Arrival Time** |
|---|---|
| P0 | (0 – 0) + (12 – 3) = 9 |
| P1 | (3 – 1) = 2 |
| P2 | (6 – 2) + (14 – 9) + (20 – 17) = 12 |
| P3 | (9 – 3) + (17 – 12) = 11 |

Multi-level queueing, which has been in use since the early 1960s, is a queue having a specified number of levels. Items are allocated to a given level (accordance with a predetermined algorithm) at the time of entry, and they cannot be altered to another level. Items are removed from the queue by removing all entries from one level and then moving on to the next. When an object is moved to a higher level, the "fetching" process is restarted. Each level of the queue is able to use its own scheduling, providing it more flexibility than a multi-level queue. In situations where processes may be classified into groups based on characteristics such as process type, CPU time, and so on (Quigley et al., 2009).

The multi-level queue: 196 scheduling approach is used for IO access, memory capacity, etc. In essence, there are two sorts of processes: foreground processes and background processes. A multi-level queue scheduling technique will have 'n' queues, where 'n' is the number of groups into which the processes are classified. Each queue gets its own priority and scheduling method, like 194 or FCFS round-robin scheduling. To finish the whole process in a queue, all queues with a higher priority than it must be cleared, signaling that the process in those higher priority queues has finished. In this scheduling strategy, once a process is allocated to a queue, it will not move to any other queues.

To comprehend non-preemptive and pre-emptive multilevel scheduling with FCFS algorithm for both queues, evaluate Figure 5.14 with the arrival time, execute time, and type of process (foreground or background – where foreground processes are given high priority).



| Process Name | Arrival Time | Execute Time | Type |
|---|---|---|---|
| P0 | 0 | 5 | Foreground |
| P1 | 1 | 8 | Background |
| P2 | 3 | 7 | Background |
| P3 | 4 | 3 | Foreground |
| P4 | 5 | 3 | Foreground |
| P5 | 8 | 11 | Background |
| P6 | 15 | 3 | Foreground |
| P7 | 25 | 4 | Foreground |

**Figure 5.14.** Non-preemptive and pre-emptive multilevel scheduling.

*Source: https://en.wikipedia.org/wiki/Multilevel_queue.*

## 5.20. INTER-PROCESS COMMUNICATION

In computer science, inter-process communication, often known as IPC, refers to the mechanisms that an OS provides to allow processes to manage shared data. Clients and servers are two types of IPC that applications can use. The client asks information, and the server responds. Many apps, as is usual in distributed computing, act as both clients and servers.

IPC is significantly used in the design of microkernels and nanokernels, which limit the number of functionalities supplied by the kernel. Certain capabilities are supplied by connecting with servers via IPC, culminating in a large increase in communication as compared to a traditional monolithic kernel. IPC interfaces frequently incorporate variable analytic framework structures. These processes maintain the compatibility of the multi-vector protocols on which IPC models rely (Whipple et al., 2009).

There are both asynchronous and synchronous IPC techniques. Synchronization primitives can be used to achieve synchronous behavior in an asynchronous IPC system. This is how processes talk to each other (Figure 5.15).

There are basically two methods:

- **Shared Memory (with a Process "Kick"):** Fast/no data transfer.
- **Message Passing:** distributed/better isolation.

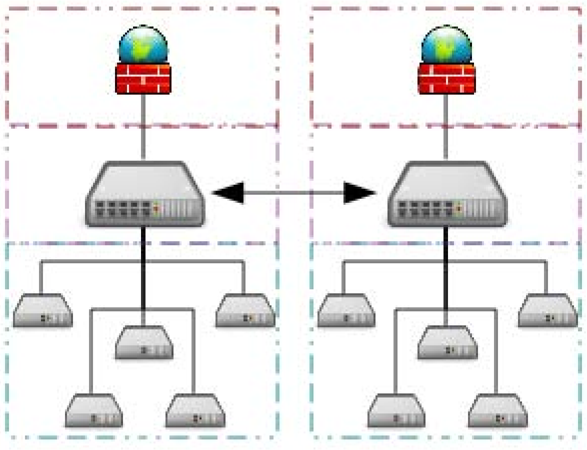


**Figure 5.15.** A grid computing system that connects many personal computers over the internet via inter-process network communication.

*Source: https://en.wikipedia.org/wiki/Inter-process_communication.*

CHAPTER 6

# OPERATING SYSTEM MULTI-THREADING

## CONTENTS

## 6.1. INTRODUCTION

An operating system (OS) is a system program that directs computer hardware, computer program resources, and gives common organizations for computer programs. Time-sharing working systems arrange assignments for capable utilize of the system and may in addition consolidate bookkeeping program forgotten designation of processor time, mass capacity, printing, and other resources (Wentzlaff et al., 2010). For gear capacities such as input and abdicate and memory assignment, the working system acts as an arbiter between programs and the computer gear, in show disdain toward the reality that the application code is more frequently than not executed directly by the gear and as frequently as conceivable makes system calls to an OS work or is ruined by it. Working systems are found on various contraptions that contain a computer, from cellular phones and video entertainment underpins to web servers and supercomputers.

## 6.2. OPERATING SYSTEM (OS) MULTI-THREADING

### 6.2.1. A Thread

A thread of execution is the humblest course of action of adjusted enlightening that can be directed independently by a scheduler, which is customarily a parcel of the working system. The execution of strings and shapes shifts between working systems, but in most cases a string may well be a component of a handle. The various strings of a given handle may be executed concurrently through multithreading capabilities, sharing assets such as memory, though assorted shapes do not share these resources. In particular, the strings of a handle share its executable code and the values of its effectively assigned variables and non-thread-local around the world variables at any given time (Figure 6.1).

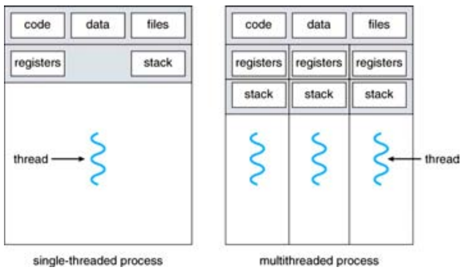


**Figure 6.1.** An image showing a thread.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.cs.uic.edu%2F~jbell%2FCourseNotes%2FOperatingSystems%2Fimages%2F*

*Chapter4%2F4_01_ThreadDiagram.jpg&imgrefurl=https%3A%2F%2Fwww.cs.uic.edu%2F~jbell%2FCourseNotes%2FOperatingSystems%2F4_Threads.html&tbnid=UWyRFcn55ynvqM&vet=12ahUKEwij-qu9n4n2AhXJ0KQKHZb4BlwQMygAegUIARC2AQ.i&docid=obeJ6owZO6We4M&w=704&h=406&q=a%20thread%20in%20os&ved=2ahUKEwij-qu9n4n2AhXJ0KQKHZb-4BlwQMygAegUIARC2AQ.*

Computers were built to perform an arrangement of single errands, like a calculator. Essential working framework highlights were created within the 1950s, such as inhabitant screen capacities that seem consequently run diverse programs in progression to speed up handling. Working frameworks did not exist in their cutting-edge and more complex shapes until the early 1960s. Equipment highlights were included, that empowered utilize of runtime libraries, hinders, and parallel handling. When personal computers (PCs) got to be well known within the 1980s, working frameworks were made for them comparative in concept to those utilized on bigger computers. In the 1940s, the most punctual electronic advanced frameworks had no working frameworks (Lass and Gronau, 2020). Electronic frameworks of this time were modified on columns of mechanical switches or by jumper wires on plugboards. These were special-purpose frameworks that, for a case, produced ballistics tables for the military or controlled the printing of finance checks from information on punched paper cards.

Strings made an early appearance underneath the title of "errands" in OS/360 Multiprogramming with a Variable Number of Errands in 1967. Saltzer credits Victor A. Vyssotsky with the term "thread." The notoriety of threading has extended around 2003, as the advancement of the CPU repeat was supplanted with the improvement of the number of centers, in turn requiring concurrency to utilize diverse centers.

Multithreading is the ability of the OS handle to supervise its utilities by more than one client at a time and to in fact direct distinctive requests by the same client without having to have various copies of the programming running inside the computer. Each client inquires for a program or system advantage, and here a client can as well be another program, is kept track of as a string with an apportioned character. As programs work on the purpose of the starting inquiry for that string and are ruined by other requests, the status of work on the purpose of that string is kept off-track until the work is completed (Tsolakis et al., 2019).

### 6.2.2. Process

At the kernel level, a process contains one or more-bit strings, which share the process's assets, such as memory and record handles. A process is a unit of assets, whereas a string may be a unit of planning and execution. Bit planning is ordinarily consistently done preemptively or, less commonly, agreeably. At the client level, preparing such as a runtime framework can itself plan numerous strings of execution. In case these don't share information, as in Erlang, they are ordinarily similarly called forms, whereas in case they share information they are as a rule called client strings, especially if preemptively planned. Agreeably planned client strings are known as strands; diverse forms may unexpectedly plan client strings. Client strings may be executed by part strings in different ways, one-to-one, many-to-one, many-to-many. The term light-weight handle differently alludes to client strings or part components for planning client strings onto bit strings (Figure 6.2).

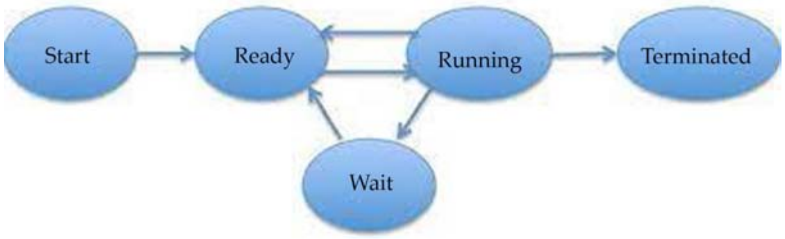


**Figure 6.2.** An illustration of process cycle in operating system.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.tutorialspoint.com%2Foperating_system%2Fimages%2Fprocess_state.jpg&imgrefurl=https%3A%2F%2Fwww.tutorialspoint.com%2Foperating_system%2Fos_processes.htm&tbnid=xXkImcvB5GXjRM&vet=12ahUKEwj0gJTjn4n2AhUMxuAKHU8SAyUQMygBegUIARCvAQ.i&docid=RovHrGZeLI3K9M&w=600&h=201&q=a%20process%20in%20os&ved=2ahUKEwj0gJTjn4n2AhUMxuAKHU8SAyUQMygBegUIARCvAQ.*

A process could be a heavyweight unit of bit planning, as making, crushing, and exchanging forms is moderately costly. Forms possess assets apportioned by the working framework. Assets incorporate memory for both code and information, record handles, attachments, gadget handles, Windows, and a handle control square. Processes are confined by handle separation and don't share address spaces or record assets but through unequivocal strategies such as acquiring record handles or shared memory portions, or mapping the same record in a shared way (Tang et al., 2010). Making or wrecking a process is moderately costly, as assets must be obtained or discharged. Forms are ordinarily preemptively multitasked, and prepare to exchange is moderately costly, past essential taken a toll of setting exchanging, due to issues such as cache flushing; in specific, handle exchanging changes virtual memory tending to, causing negation, and hence flushing of an untagged interpretation lookaside buffer, strikingly on x86.

## 6.3. DIFFERENCE BETWEEN A PROCESS AND A THREAD

Process implies any program is in execution. Thread implies a portion of a process. A process takes more time to terminate. The thread takes less time to terminate. It takes more time for creation. It takes less time for creation. It too takes more time for setting switching. It takes less time for setting switching. A process is less proficient in terms of communication. Thread is more effective in terms of communication. Multiprogramming holds the concepts of multi-process. We don't require multi-program inactivity for different strings since a single handle comprises numerous strings. A process is isolated. Threads share memory. A process is called the overwhelming weight process. A String is lightweight as each string in a prepare offers code, information, and resources.

Process exchanging employments interface in a working system. Thread exchanging does not require calling a working framework and causes a hinder to the kernel. If one handle is blocked at that point it'll not impact the execution of other processes Second string within the same assignment may not run, whereas one server string is blocked (Shin et al., 2014). A process has it possess Handle Control Square, Stack, and Address Space. Thread has Parents' PCB; it possesses String Control Piece and Stack and common Address space. If one handle is blocked, then no other handle can execute until the primary prepare is unblocked. While one string is blocked and holding up, a moment string within the same errand can run.

Changes to the parent process do not impact child forms. Since all strings of the same handle share address space and other resources so any changes to the foremost string may impact the behavior of the other strings of the strategy.

## 6.4. BUILDING BLOCKS FOR THE FUNCTIONING OF A THREAD

The building blocks for the functioning of a thread of an OS all exist in an organized way in order to make the particular parts of a computer work together. All client computer programs must go through the OS in orchestrating to utilize any of the gear, whether it is as basic as a mouse or as complex as a Web component.

### 6.4.1. The Bit

With the assistance of the firmware, the kernel, commonly referred to as the kernel gives the first basic level of control over all of the computer's hardware. The kernel also manages memory get to the programs inside the Pummel, it decides which programs get to be processed in which hardware resources, it sets up or resets the CPU's working states for perfect operation at all times, and it organizes the data for long-term non-volatile capacity with record systems on such media as disks, tapes, and flash memory (Figure 6.3).

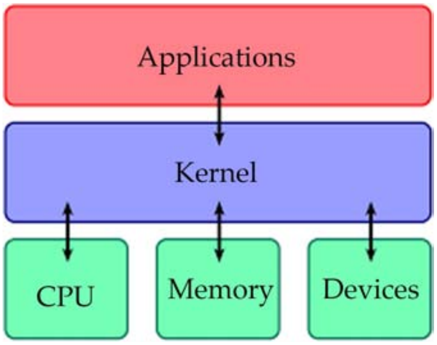


**Figure 6.3.** An illustration of the kernel category.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fupload.wikimedia.org%2Fwikipedia%2Fcommons%2Fthumb%2F8%2F8f%2FKernel_Layout.svg%2F1200px-Kernel_Layout.svg.png&imgrefurl=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FKernel_(operating_system)&tbnid=t7ri_jKHTjYo9M&vet=12ahUKEwiDzamGoIn2AhXuCBAIHZ-2BJwQMygAegUIARDUAQ.i&docid=nXMR5bbjPdkGrM&w=1200&h=947&itg=1&q=kernel&ved=2ahUKEwiDzamGoIn2AhXuCBAIHZ--2BJwQMygAegUIARDUAQ.*

## 6.5. THE CENTRAL PROCESSING UNIT

A multiprogramming OS must be able to manage all system memory that is frequently utilized by programs. This ensures that a program does not intrude with memory as it is being utilized by another program. Since programs share time, each program must have independent get to memory. Agreeable memory organization, utilized by various early working systems, acknowledges that all programs make proper use of the kernel's memory chief, and do not outperform their assigned memory (Singh, 2014). This system of memory organization is better than previous ones since programs frequently contain bugs that can cause them to outperform their assigned memory. On the off chance that a program comes up brief, it may cause memory utilized by one or more other programs to be affected or overwritten (Figure 6.4).

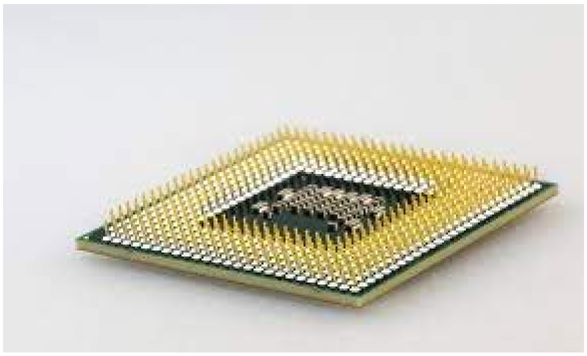


**Figure 6.4.** An illustration of a CPU.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.redhat.com%2Fsysadmin%2Fsites%2Fdefault%2Ffiles%2Fstyles%2Ffull%2Fpublic%2F2020–07%2Fcpu-564771_1920%2520Cropped.jpg%3Fitok%3DmLy0ZtKX&imgrefurl=https%3A%2F%2Fwww.redhat.com%2Fsysadmin%2Fcpu-components-functionality&tbnid=BVjupNthyV_4wM&vet=12ahUKEwj-jq2hoIn2AhXK16QKHVMxAHcQMygAegUIARDWAQ.i&docid=wqmdJ_hdWXj_qM&w=1000&h=600&q=the%20cpu&ved=2ahUKEwj-jq2hoIn2AhXK16QKHVMxAHcQMygAegUIARDWAQ.*

Malicious programs may purposely adjust another program's memory or may impact the operation of the working system itself. With a cooperative memory organization, it takes only one malicious program to crash the framework. Memory confirmation enables the bit to compel a process' get to the computer's memory. Distinctive techniques of memory confirmation exist, checking memory division and paging. All strategies require a number of levels of hardware reinforcement, which doesn't exist in all computers.

In both division and paging, certain guaranteed mode registers show to the CPU what memory address it has to be permit a running program to induce to. Endeavors to induce to other addresses trigger an interrupt which cause the CPU to re-enter boss mode, putting the portion in charge (Sangorrin et al., 2010). As a rule, called a segmentation, and since it is both troublesome to consign a significant result to such an operation, as a result it is commonly a sign of a getting into insidiousness program, the portion for the foremost portion resorts to finishing the chafing program, and reports the botch.

## 6.6. INFORMATION STORAGE SYSTEM

Getting to information put away on disks may be a core process of all working frameworks. Computers store information on disks utilizing records, which are organized in particular ways in arrange to allow for quicker get to, higher unwavering quality, and to create superior utilize of the drive's accessible space. The particular way in which records are put away on a disk is called a file system, and empowers records to have names and properties. It too permits them to be put away in a pecking order of catalogs or organizers organized in a catalog tree (Figure 6.5).

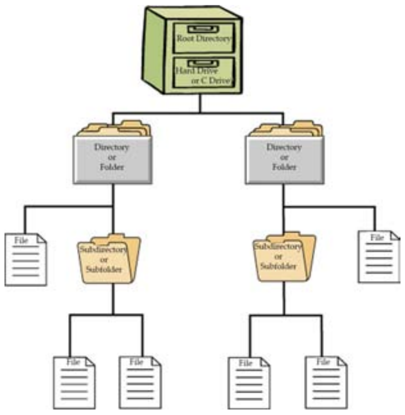


**Figure 6.5.** Illustration of the file system structure.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.cs.colostate.edu%2F~cs155%2Ffilecabinetsystem.gif&imgrefurl=https%3A%2F%2Fwww.cs.colostate.edu%2F~cs155%2FSprin*

*g18%2FLecture%2FFileSystem&tbnid=yP8zpmCwCAGS6M&vet=12ahUKEwj277rFoIn2AhWWM8AKHd_mAQAQMygGegUIARDCAQ.i&docid=0KF7iuCk17ydMM&w=436&h=447&q=file%20system%20in%20a%20computer&ved=2ahUKEwj277rFoIn2AhWWM8AKHd_mAQAQMygGegUIARDCAQ.*

An associated capacity gadget, such as a hard drive, is accessed through a gadget driver. The gadget driver gets the particular dialect of the drive and is able to decipher that dialect into a standard dialect utilized by the working framework to get to all disk drives. Different contrasts between record frameworks make supporting all record frameworks troublesome (Sharma et al., 2012). Permitted characters in record names, case affectability, and the nearness of different sorts of record qualities make the usage of a single interface for each record framework an overwhelming errand. Working frameworks tend to suggest utilizing record frameworks particularly planned for them. In any case, in hone, third party drivers are as a rule accessible to donate back for the foremost broadly utilized record frameworks in most general-purpose working frameworks.

## 6.7. SIMILARITIES BETWEEN A PROCESS AND A THREAD

Both processes and threads share CPU and as it were one string dynamic at a time. Like processes, strings inside a process, strings inside a form execute sequentially. Like processes, a string can make children. And like processes, in case one string is blocked, another string can run. Both processes and strings have a parent handle. Process features a process ID the same way string contains a string id. Numerous threads can exist inside the same handle and share assets such as memory, whereas diverse forms don't share these assets.

### 6.7.1. The Advantages of a Thread

Strings minimize the setting exchanging time. The use of threads gives concurrency inside a process. Efficient communication. It is more prudent to make and set switch threads. Threads permit utilization of multiprocessor designs to a more noteworthy scale and effectiveness. Responsiveness, in case the method is separated into numerous strings, if one string completes its execution, at that point its yield can be promptly returned. Faster setting switch time between strings is lower compared to prepare setting switch. Prepare setting exchanging requires more overhead from the CPU. Effective utilization of multiprocessor framework, in the case of different strings in a single handle, at that point it is appropriate to plan different strings on numerous processors. This will make handling execution quicker (Levis et al., 2005).

Assets like code, information, and records can be shared among all strings inside a process. Furthermore, stack and registers can't be shared among the strings. Each string has possessed its own stack and registers. Communication between different strings is less demanding, as the strings offer common address space. Whereas in handle we got to take after a few particular communication procedures for communication between two processes. Enhanced throughput of the framework. On the off chance that a handle is isolated into different strings, and each string work is considered as one work, at that point the number of occupations completed per unit of time is expanded, hence expanding the throughput of the framework.

### 6.7.2. Disadvantages of a Thread

With more strings, the code gets to be troublesome to investigate and maintain. Thread creation puts a stack on the framework in terms of memory and CPU resources. We ought to do exemption taking care of interior the worker strategy as any unhandled special cases can result within the program slamming. Strings will moderate the execution of programs when a successive calculation is refactored into an arrangement of strings. In case the strings execute on the same center at that point the moderate down is essentially caused by setting exchanging between the strings. In the event that the strings are on diverse centers at that point, the moderate down is fundamentally caused by the information synchronization required for the arrangement of strings to pass information from one to the other (Shaw et al., 2016).

String synchronization is an additional overhead to the developers. Sharing the common information over the strings might cause information irregularity or string adjustment issues. Threads blocking for assets is a more common problem. Difficult in overseeing the code in terms of investigating or composing the code.

## 6.8. FIBERS IN MULTITHREADING

Fibers are an indeed lighter unit of planning which are agreeably planned: a running fiber must unequivocally surrender to permit another fiber to run, which makes their execution much less demanding than the part or client strings. Fiber can be planned to run in any string within the same handle. This grants applications to pick up execution changes by managing planning themselves, rather than depending on the part scheduler which may not be tuned for the application.

Parallel programming situations such as Open MP ordinarily execute their assignments through strands. Closely related to filaments are coroutines, with the qualification being that coroutines are a language-level development, whereas strands are a system-level development.

### 6.8.1. Thread Scheduling

There exist different scheduling techniques in the world of computing threads. The most common scheduling techniques include preemptive cooperative scheduling and single multiprocessor systems scheduling.

## 6.9. PREEMPTIVE AND COOPERATIVE SCHEDULING

OS plan strings either preemptively or agreeably. Multi-user working frameworks for the most part favor preemptive multithreading for its finer-grained control over execution time by means of setting exchanging. In any case, preemptive planning may context-switch strings at minutes unexpected by software engineers, in this way causing bolt escort, need reversal, or other side-effects (Silva et al., 2006).

In differentiate, agreeable multithreading depends on strings to give up control of execution, hence guaranteeing that strings run to completion. This may cause issues in case an agreeably multitasked string squares by holding up on an asset or in the event that it starves other strings by not yielding control of execution amid seriously computation.

## 6.10. SINGLE AND MULTIPROCESSOR SYSTEM SCHEDULING

Until the early 2000s, most desktop computers had since it was one single-core CPU, with no back for preparing strings, in show up loathe toward of the reality that strings were still utilized on such computers since trading between strings was for the preeminent allocate still quicker than full-process setting switches. In 2002, Intel included back for concurrent multithreading to the Pentium 4 processor, underneath the title hyper-threading; in 2005, they appeared the dual-core Pentium D processor and AMD appeared the dual-core Athlon 64 X2 processor.

Frameworks with a single processor by and large execute multithreading by time cutting: the central handling unit switches between diverse program strings. This setting exchanging, as a rule, happens habitually sufficient that clients see the strings or errands as running in parallel for prevalent server working frameworks, most extreme time cut of a string when other strings are holding up, is frequently constrained to 100–200 ms (Sjöstrand et al., 2015). On a multiprocessor or multi-core framework, different strings can execute in parallel, with each processor or center executing a partitioned string at the same time; on a processor or center with equipment strings, isolated program strings can moreover be executed concurrently by isolated equipment strings.

## 6.11. THREAD POOLS

A well-known programming design including strings is that of string pools where a set number of strings are made at startup that at that point hold up for an errand to be relegated. When a modern assignment arrives, it wakes up, completes the assignment, and goes back to holding up (Santos et al., 2013). This maintains a strategic distance from the generally costly string creation and annihilation capacities for each errand performed and takes string administration out of the application developer's hand and takes off it to a library or the working framework that's superior suited to optimize string administration (Figure 6.6).

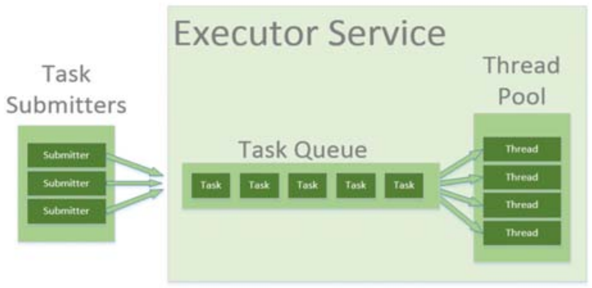


**Figure 6.6.** A thread pool.

*Source: https://www.baeldung.com/thread-pool-java-and-guava*

## 6.12. PROGRAMMING LANGUAGE SUPPORT FOR THREADS

IBM PL/I (F) included support for multithreading called multitasking and this was continued inside the Optimizing Compiler and a though a while later shapes. The IBM Undertaking PL/I compiler appeared an unused appear up "string" API. Not one or the other change was distributed of the PL/I standard. A different utilization of C and C++ back threading, and grant gets to the neighborhood threading APIs of the working system. A standardized interface for string utilization is POSIX Strings, which may be a set of C-function library calls. OS shippers are free to actualize the interface as required, but the application arrange ought to be able to utilize the same interface over unmistakable stages. Most Unix stages checking Linux back Pthreads. Microsoft Windows has its claim set of string capacities inside the strategy (Cashmore et al., 2015).

A few higher levels and as a rule cross-platform programming dialects, such as Java, Python, and .NET system dialects, uncover threading to engineers whereas abstracting the stage particular contrasts in threading usage within the runtime. A few other programming dialects and dialect expansions too attempt to theoretical the concept of concurrency and threading from the engineer completely. A few dialects are planned for consecutive parallelism instep, without requiring concurrency or strings.

A few deciphered programming dialects have execution. The GIL may be a shared avoidance bolt held by the translator that can avoid the mediator from at the same time deciphering the application's code on two or more strings at once, which viably limits the parallelism on numerous center frameworks. The execution of this limit is generally for processor-bound strings, which require the processor, and not much for I/O-bound or network-bound ones. Other usages of deciphered programming dialects, such as Tcl utilizing the String expansion, dodge the GIL constrain by utilizing a Flat show where information and code must be expressly shared between strings. In Tcl, each string has one or more translators.

In programming models such as CUDA planned for information parallel computation, a cluster of strings run the same code in parallel utilizing as it were its ID to discover its information in memory. In pith, the application must be outlined so that each string performs the same operation on distinctive fragments of memory so that they can work in parallel and utilize the GPU architecture. Hardware portrayal dialects such as Verilog have diverse threading demonstrate that underpins greatly expansive numbers of strings.

## 6.13. THREADS DATA SYNCHRONIZATION

Strings within the same prepare share the same address space. This permits concurrently running code to couple firmly and helpfully trade information without the overhead or complexity of an IPC. When shared between strings, in any case, indeed basic information structures got to be inclined to race conditions on the off chance that they require more than one CPU instruction to upgrade: two strings may conclusion up endeavoring to overhaul the information structure at the same time and discover it suddenly changing underneath. Bugs caused by race conditions can be exceptionally troublesome to duplicate and confine (Chinetha et al., 2015).

To avoid this, threading application programming interfacing (APIs) offers synchronization primitives such as mutexes to bolt information structures against concurrent get to. On uniprocessor frameworks, a string running into a bolted mutex must rest and subsequently trigger a setting switch. On multi-processor frameworks, the string may instep survey the mutex in a spinlock. Both of these may sap execution and drive processors in symmetric multiprocessing (SMP) frameworks to fight for the memory transport, particularly on the off chance that the granularity of the locking is as well fine.

### 6.13.1. Components of Threads

Threads are made up of three main components. They include the program counter, the register set, and the stack space.

#### *6.13.1.1. Program Counter (PC)*

The program counter commonly called the instruction pointer in Intel x86 and Itanium chip, and a few of the time called the instruction address to enlist, the instruction counter, or reasonable parcel of the instruction sequencer, maybe a processor enlist that appears where a computer is in its program course of action. Ordinarily, the PC is increased after bringing instruction and holds the memory address of another instruction that would be executed (Dieber et al., 2017). Processors more often than not get information consecutively from memory, but control exchange enlightening alters the arrangement by putting modern esteem within the PC. These incorporate branches, subroutine calls, and returns. An exchange that's conditional on the truth of a few statements lets the computer take after a diverse arrangement beneath diverse conditions.

A branch gives that the taking after instruction is gotten from someplace else in memory. A subroutine calls not because it branched but saves the going sometime recently substance of the PC a few put. A return recuperates the saved substance of the PC and places it back inside the PC, proceeding progressive execution with the instruction taken after the subroutine call.

### *6.13.1.2. Register Set*

A register set is one of a little set of information holding places that are a portion of the computer processor. A enlist may hold an instruction, a capacity address, or any kind of information. A few informational indicate registers as the portion of the instruction. For a case, an instruction may indicate that the substance of two characterized registers is included together and after that put in an indicated enlist. A register must be gigantic enough to hold an instruction. In a number of computer plans, there are smaller registers for shorter instructions. Depending on the processor arrangement and language rules, registers may be numbered or have self-assertive names. A processor frequently contains distinctive record registers, additionally known as address registers or registers of change. The effective address of any substance in a computer consolidates the base, list, and relative addresses, all of which are put absent inside the record enlist. A move select is another sort. Bits enter the move select at one conclusion and rise from the other conclusion. Flip flops, additionally known as bistable entryways, store, and plan the data.

### *6.13.1.3. Stack Space*

A stack space could be a piece of memory utilized to store brief information required for legitimate program execution. The Propeller naturally handles the stack for the application cog, but propelling extra cogs may require additional stack space apportioned by the engineer. In most present-day computer frameworks, each string features a saved locale of memory alluded to as its stack. When a work executes, it may include a few of its neighborhood state information to the beat of the stack; when the work exits it is dependable for evacuating that information from the stack. At least, a thread's stack is utilized to store the area of a return address given by the caller in order to permit return explanations to return to the right area (Dixon et al., 2012).

Since the information is included and evacuated in a last-in-first-out way, stack-based memory allotment is exceptionally basic and regularly much speedier than heap-based memory assignment ordinarily distributed through malloc. Another highlight is that memory on the stack is consequently, and exceptionally effectively, recovered when the work exits, which can be helpful for the software engineer in the event that the information is not required.

On the off chance that in any case, the data ought to be kept in some shape, at that point it must be imitated from the stack to the stack a few times as of late the work exits. Hence, a stack-based task is sensible for data that is not required after the current work exits. A thread's assigned stack estimate can be as little as only a few bytes on some small CPUs. Assigning more memory on the stack than is available can result in a crash due to stack surge.

Stack-based assignment can moreover cause minor execution issues: it leads to variable-size stack outlines so that both stack and outline pointers got to be overseen. This is often ordinarily much less exorbitant than calling malloc and free besides. In specific, in case the current work contains both calls to alloca and pieces containing variable-length nearby information at that point a struggle happens between alloca's attempts to extend the current stack outline until the current work exits versus the compiler's got to put neighborhood factors of variable length within the same area within the stack outline. This strife is regularly settled by making a partitioned chain of pile capacity for each call to allocate (Cao et al., 2008). The chain records the stack profundity at which each allotment happens, ensuing calls to alloca in any work trim this chain down to the current stack profundity to in the long run free any capacity on this chain. A call to alloca with the contention of zero can moreover be utilized to trigger the liberating of memory without designating any more such memory. As a result of this strife between alloca and neighborhood variable capacity, utilizing alloca could be no more productive than utilizing malloc.

Numerous Unix-like frameworks as well as Microsoft Windows, actualize a work called alloca for powerfully designating stack memory in a way comparative to the heap-based malloc. A compiler ordinarily interprets it as inlined enlightening controlling the stack pointer, comparative to how variable-length clusters is handled. Although there is no need to unequivocally free the memory, there is a hazard of vague behavior due to stack flood.

## 6.14. TYPES OF THREADS

In the world of OSs, threads are divided into two major categories. The first category is referred to as the Kernel-level thread, whereas the second category is referred to as the User-level thread.

### 6.14.1. Kernel-Level Threads

A kernel thread may be a lightweight unit of bit planning. At slightest one part string exists inside each handle. In case different part strings exist inside a handle, at that point they share the same memory and record assets. Bit strings are preemptively multitasked in case the working system's process scheduler is preemptive. Bit strings don't claim assets but for a stack, a duplicate of the registers counting the program counter, and thread-local capacity in the event that any, and are thus generally cheap to make and annihilate. String exchanging is additionally generally cheap: it requires a setting switch, but does not alter virtual memory and is hence cache-friendly (DiLuoffo et al., 2018).

The bit can relegate one string to each consistent center in a framework, since each processor parts itself up into numerous coherent centers in case it underpins multithreading, or as it were underpins one consistent center per physical center on the off chance that it does not, and can swap out strings that get blocked. Be that as it may, part strings take much longer than client strings to be swapped.

The kernel-level threads have their own advantages over the user-level threads. Part can at the same time plan different strings from the same handle on different processes. If one string in a handle is blocked, the Part can plan another string of the same process. Kernel schedules themselves can be multithreaded. Just like a coin has two sides, so do kernel-level threads, they also have their drawbacks. One, they take time to create and manage as compared to user threads, making them slow. The other disadvantage is that the transfer of one thread to another within the same process requires a mode switch to the kernel, for the transfer to be a success (Figure 6.7).

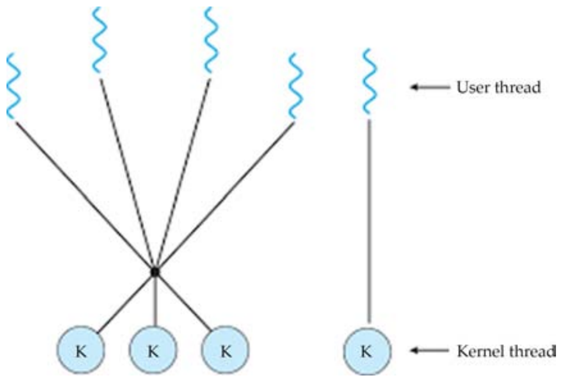


**Figure 6.7.** Illustration of both user-level and kernel-level threads.

*Source: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.cs.uic.edu%2F~jbell%2FCourseNotes%2FOperatingSystems%2Fimages%2FChapter4%2F4_08_TwoLevel.jpg&imgrefurl=https%3A%2F%2Fwww.cs.uic.edu%2F~jbell%2FCourseNotes%2FOperatingSystems%2F4_Threads.html&tbnid=grhu6IeE8_WZIM&vet=12ahUKEwi889SboYn2AhXZAGMBHZ6lA4AQMygPegUIARDbAQ.i&docid=obeJ6owZO6We4M&w=597&h=401&q=kernel%20level%20thread&ved=2ahUKEwi889SboYn2AhXZAGMBHZ-6lA4AQMygPegUIARDbAQ.*

### 6.14.2. User-Level Threads

Strings are some of the times executed in userspace libraries, hence called user threads. The bit is unconscious of them, so they are overseen and planned in userspace. A few executions base their client strings on the best of a few bit strings, to advantage from multi-processor machines. Client strings as executed by virtual machines are moreover called green strings (De et al., 2007). As client string executions are ordinarily completely in userspace, setting exchanging between client strings inside the same handle is greatly productive since it does not require any interaction with the part at all: a setting switch can be performed by locally sparing the CPU registers utilized by the as of now executing client string or fiber and after that stacking the registers required by the client string or fiber to be executed. Since planning happens in user space, the planning arrangement can be more effortlessly custom-made to the prerequisites of the program's workload.

In any case, the utilization of blocking framework calls in client strings as contradicted to part strings can be risky. On the off chance that a client string or a fiber performs a framework call that pieces, the other client strings, and strands within the handle are incapable to run until the framework call returns. A normal illustration of this issue is when performing I/O: most programs are composed to perform I/O synchronously. When an I/O operation is started, a framework call is made and does not return until the I/O operation has been completed. Within the mediating period, the complete preparation is blocked by the part and cannot run, which starves other client strings and strands within the same preparation from executing.

A common arrangement to this issue utilized, in specific, by numerous of green strings executions is giving an I/O API that actualizes an interface that squares the calling string, instead of the complete handle, by utilizing non-blocking I/O inside, and planning another client string or fiber whereas the I/O operation is in advance. Comparative arrangements can be given for other blocking framework calls. Then again, the program can be composed to maintain a strategic distance from the utilization of synchronous I/O or other blocking framework calls in specific, utilizing non-blocking I/O, counting lambda continuations, and awaiting primitives (Bhattacharjee and Lustig, 2017).

The advantages of User-Level threads are numerous. String exchanging does not require part mode privileges. The user-level string can run on any OS so long as it is certified. Scheduling can be application particular within the client level thread. User-level threads are quick to form and oversee. This type of thread does also have its own disadvantages. They have multiple system calls which tend to be blocked in most cases. User-level threads do not utilize the advantage of multiprocessing, which slows down the threading process.

## 6.15. DIFFERENCES BETWEEN KERNEL-LEVEL THREADS AND USER-LEVEL THREADS

User-level strings are speedier to form and manage. Kernel-level strings are slower to make and manage. Implementation is by a string library at the client level. Operating framework bolsters creation of Bit threads. User-level string is nonexclusive and can run on any working system. Kernel-level string is particular to the working system. Multi-threaded applications cannot take advantage of multiprocessing. Kernel schedules themselves can be multithreaded.

Client string are executed by users. Kernel strings are actualized by OS. OS doesn't recognize client level threads. Kernel strings are recognized by OS. Implementation of Client strings is easy. Implementation of Bit string is complicated. Context switch time is less. Context switch time is more. Context switch requires no equipment support. Hardware bolster is needed. If one client level string performs blocking operation at that point whole prepare will be blocked. If one part string performs blocking operation at that point another string can proceed execution. User level strings are planned as subordinate threads. Kernel level strings are planned as autonomous strings.

## 6.16. SIMILARITIES BETWEEN USER-LEVEL THREADS AND KERNEL-LEVEL THREADS

In Windows, there's basically zero contrast between bit and client strings, as strings. A bit string runs with a substantial bit address space and an obscure client address space; a client string runs with a substantial client address space and no bit address space at all. Synchronization primitives in a bit string utilize pointers to the synchronization objects rather than client strings which require handles, but something else they work a bit like their user-level reciprocals (Deseriis, 2017). These are minor contrasts. It is commonplace, but not required, that part strings run at needs 16–31 whereas client strings run at needs 1–15. But these are at best rules, not prerequisites and there are numerous exemptions to them. The minor points of interest of the contrasts have small effect on how strings are utilized. We cannot talk to other working frameworks. The scheduler fair sees a bunch of strings. A high-priority user-level string (16–31) is fair a string as distant as the scheduler is concerned. A low-priority part string is fair another string. There are marginally diverse dispatches to run these strings, but from a philosophical perspective these are unessential. A string could be a string; the subtle elements of client vs. bit have as it were minor impacts, such as how the page maps are set up to run them.

### 6.16.1. Multithreading Models

As stated earlier, multithreading licenses the application to restrict its errand into person strings. In multi-threads, the same get prepared or errand can be done by the number of strings, or arranged to say that there's more than one string to perform the errand in multithreading. With the use of multithreading, multitasking can be wrapped up. The first disadvantage of single threading systems is that since it was one task can be performed at a time, so to overcome the disadvantage of this single threading, there's multithreading that awards unmistakable assignments to be performed (Estefo et al., 2019).

In a working framework, strings are isolated into the user-level string and the Kernel-level string. User-level strings dealt with autonomous shape over the bit and subsequently overseen without any part bolster. On the inverse hand, the working framework straightforwardly oversees the kernel-level strings. All things considered, there must be a frame of the relationship between user-level and kernel-level strings.

There exist three main categories of multithreading models. They include many to one multithreading models; one to one multithreading models; and many to many multithreading models.

### 6.16.2. Many to One Multithreading Model

The many to one model maps numerous client levels strings to one part string. This sort of relationship encourages a viable context-switching environment, effortlessly executed indeed on the basic part with no string support. The drawback of this demonstration is that since there's as it were one kernel-level string plan at any given time, this demonstration cannot take advantage of the equipment increasing speed advertised by multithreaded forms or multi-processor frameworks. In this, all the thread management is tired of the userspace. On the off chance that blocking comes, this demonstrates squares the complete framework.

There are a few points of interest to this model: Cheap synchronization. When a client string wishes to perform synchronization, the user-level string library checks to see in the event that the string ought to square. On the off chance that it does, at that point, the library enqueues the string on the synchronization primitive, dequeues a client string from the library's run line, and switches the dynamic thread (Fröhlich and Wanner, 2008). If it does not have to be square, at that point the dynamic thread continues to run. No framework calls are required in either case. Cheap string creation. To form an unused string, the strings library requires as it made a setting for the modern string and enqueue it within the user-level run queue. Resource proficiency, in that; Bit memory isn't squandered on a stack for each client string. This permits as numerous strings as virtual memory permits. Portability; cause user-level strings bundles are executed completely with standard UNIX and POSIX library calls; they are regularly very versatile (Figure 6.8).

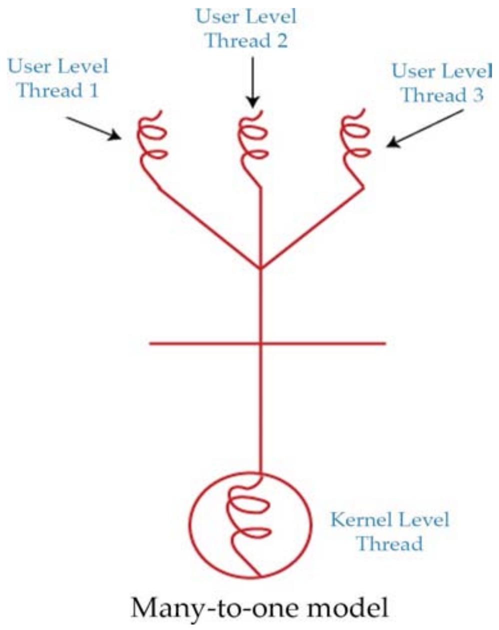


**Figure 6.8.** An image of the many to one model.

*Source: https://static.javatpoint.com/operating-system/images/multithreading-models-in-operating-system3.png.*

Be that as it may, the many-to-one show does not come without a cost. Specifically: Single-threaded OS interface. Since there's as it were one bit string, on the off chance that a client string executes a blocking system call, the complete prepare squares, since no other client string can execute until the part string (which is blocked within the framework call) gets to be accessible (Fierro and Culler, 2015). Whereas it includes essentially to usage complexity, the library can delude this issue where nonblocking variations of framework calls exist. No parallelism. Multithreaded programs beneath the many-to-one demonstrate will run no speedier on multiprocessors than they run on uniprocessors. The single part string acts as a bottleneck, anticipating ideal utilize of the multiprocessor.

In spite of considerable impediments, the relative ease of usage of many-to-one strings bundles has made it the foremost prevalent demonstrate to date. For case, the current usage of Netscape and Java accomplish their multithreading entirely through user-level strings bundles.

### 6.16.3. One to One Multithreading Model

The one-to-one show maps a single user-level string to a single kernel-level string. This sort of relationship encourages the running of numerous strings in parallel. Be that as it may, this advantage comes with its disadvantage. The era of each unused client string must incorporate making a comparing bit string causing an overhead, which can ruin the execution of the parent preparation. Windows arrangement and Linux working frameworks attempt to handle this issue by constraining the development of the string check (Figure 6.9).

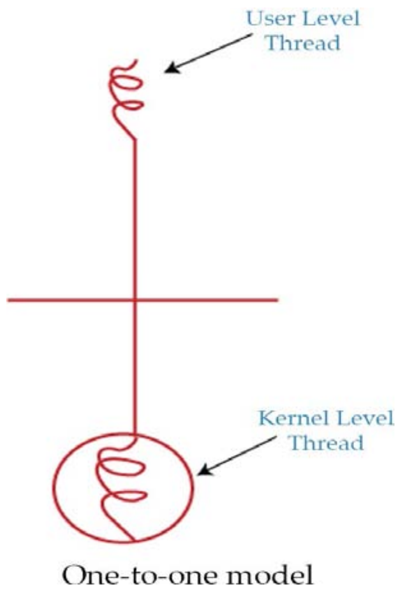


**Figure 6.9.** An image of the one-to-one model.

*Source: https://static.javatpoint.com/operating-system/images/multithreading-models-in-operating-system4.png.*

### 6.16.4. Many to Many Multithreading Model

In this sort of model, there are a few user-level strings and a few kernel-level strings. The number of bit strings made depends upon a specific application. The engineer can make numerous strings at both levels but may not be the same. The numerous to numerous demonstrate could be a compromise between the other two models. In this demonstration, in case any string makes a blocking framework call, the bit can plan another string for execution (Greenwald and Thomas, 2007). Moreover, with the presentation of multiple threads, complexity isn't shown as within the past models. In spite of the fact that this demonstration permits the creation of different part strings, genuine concurrency cannot be accomplished by this demonstration. This is often since the bit can plan as it were one handle at a time (Figure 6.10).

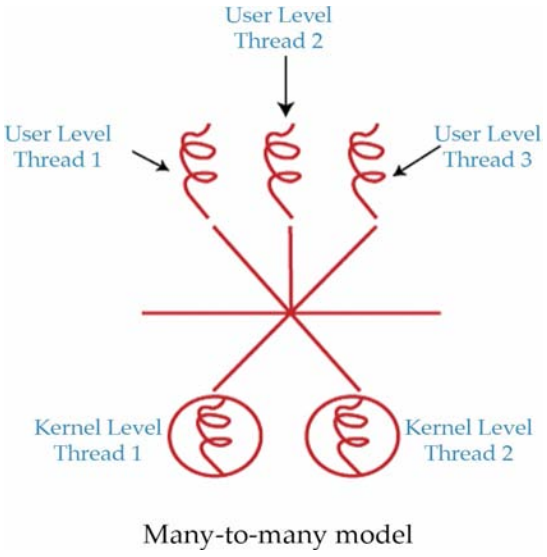


**Figure 6.10.** An image of the many to many models.

*Source: https://static.javatpoint.com/operating-system/images/multithreading-models-in-operating-system5.png.*

CHAPTER 7

# MEMORY MANAGEMENT

## CONTENTS

## 7.1. PROCESS ADDRESS SPACE

Process address space is a category of logical addresses that synchronize process references within the code. For instance, when 32-bit addressing is applied, addresses can vary from 0 to 0x7fffffff; meaning, 2^31 potential figures, for an average hypothetical scale of 2 gigabytes (GB) (Gunadi and Tiu, 2014). The internal framework handles encoding of the logical to physical addresses at the period of memory designation to the program (Figure 7.1). Three forms of addresses are applicable in the program when memory is dispensed:

- Physical addresses. The loader produces these addresses when a program is stacked into primary memory.
- Symbolic addresses. They tend to be utilized in the source code. The variable titles, constants, and guidance tags are the essential components of a symbolic address space.
- Relative addresses. During compilation, a compiler changes over symbolic addresses into relative ones.

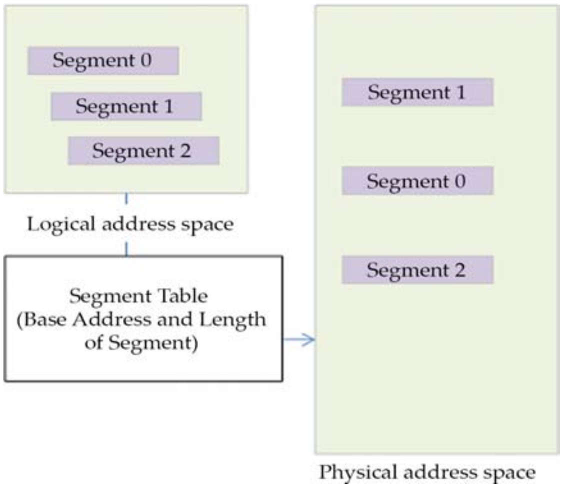


**Figure 7.1.** Operating system address space.

*Source: https://study.com/academy/lesson/how-operating-systems-manages-address-space.html.*

## 7.2. ADDRESS SPACES

Address space refers to the PC memory space and each process covers it. There are two types which include; the physical address space and virtual address space. Physical and virtual addresses are similar in compile duration and load duration address-binding plans. They both vary in execution-period address-binding plan.

### 7.2.1. Physical Address Space

Physical address refers to the actual spot in memory that's material. The system accesses information in the primary memory, with the assistance of physical address. Every component in the PC has a one-of-a-kind physical address. There are two memory models in which the memory address maps within the physical address space: The flat model whereby the physical address and linear address spaces are similar (Giorgetti et al., 2020). The segmented pattern where the physical address space is parted into segments making a logical address space per each section CPU Intel 64 architecture, and supporting physical address space more than 64 GB; the original physical address scale of IA-32 processors is execution categorical. In 64-bit configuration, there is compositional support for 64-bit lineal address space. Nevertheless, processors having Intel 64 structure might implement under 64-bits. The linear address expanse is mapped within the processor physical address space using the PAE paging instrument. The greatest scale of the physical memory is restricted by: the address bus width within the address storage, i.e., assuming an address store: a byte (most popular) or a phrase.

#### *How to get to the physical memory area by CPU?*

Logical Address is utilized as a reference point to get to the physical memory area through CPU. The Memory-Management Unit utilizes address-binding techniques for structuring the logical address to its relating physical address. The group of all logical addresses produced by a program is known as the logical address space, whereas the set of addresses comparing to these logical addresses is called physical address space. The runtime mapping from digital to physical address is performed through the memory management unit (MMU), or a hardware instrument. MMU applies various mechanisms to change virtual to physical address.

- The base register value is included to each address produced through user interaction, which is considered an offset during the period it is sent to memory. For instance, in the event that the base register integer is 10,000, an endeavor by the client to utilize address location 100 will be redistributed to location 10,100.
- The user program handles virtual addresses; it's not associated with the physical addresses.

### 7.2.2. Virtual Address Space

The Virtual Address Space is processed externally from the primary memory within the virtual memory, and it is developed within the hard disk. Whenever our primary memory is less and we need to get more advantage from this less memory, then, at that point, we make virtual memory. Virtual address doesn't exist physically, thus it cannot be said that physical address translates to logical address. The process virtual address space is the collection of virtual memory locations applicable. The address space for every process is private and can't be availed to different processes except if it is shared (Blackham et al., 2011).

A virtual address doesn't exemplify the real physical address of a memory object; rather, the system keeps a page table for each interaction, which is a private data system used to make an interpretation of virtual addresses into their comparing physical addresses. Every time the thread references a location, the system makes an interpretation of the virtual location to a physical location. The virtual location space for 32-bit Windows is 4 GB in scale and separated into two allotments: one for application by the process while the other held for use by the internal structure (Figure 7.2).

Figure 7.2 shows the default memory category for each segment.

- Default Virtual Address Scale (32-bit Windows):

**Memory range:** Application:

Low 2 GB (0x00000000 across 0x7FFFFFFF) process based.

High 2 GB (0x80000000 across 0xFFFFFFFF) system based.

- Virtual Address Space (32-bit Windows) with 4GT

If 4-gigabyte tuning (4GT) is allowed, the memory scale for each segment is as per the following.

Virtual Address Scale (32-Bit Windows) With 4GT

**Memory range:** Application:

Low 3GB (0x00000000 across 0xBFFFFFFF) process based.

High 1 GB (0xC0000000 across 0xFFFFFFFF) system based.

When 4GT is allowed, a process with the IMAGE_FILE_LARGE_ADDRESS_AWARE banner set in its photo header will have 1 extra GB of memory over the low 2 GB (Hambarde et al., 2014).

- Synchronizing the Virtual Address Space (32-bit) Windows:

You can apply the below command to establish a boot entry system that organizes the partition size which is accessible for use within the process, up to a value of between 2,048 (2 GB) and 3,072 (3 GB):

BCDEdit/set increase USERVA *Megabytes*

Following the setting of the boot entry set, the memory scale for each segment is per the below.

- Changing the Virtual Address Space (32-BIT) Windows

**Memory range**: Application

Low (0x00000000 across Megabytes) process based.

High (Megabytes+1 through 0xFFFFFFFF) system based.

Windows Server 2003: Adjust the/USERVA switch in boot.ini to a value of the range 2,048 and 3,072.

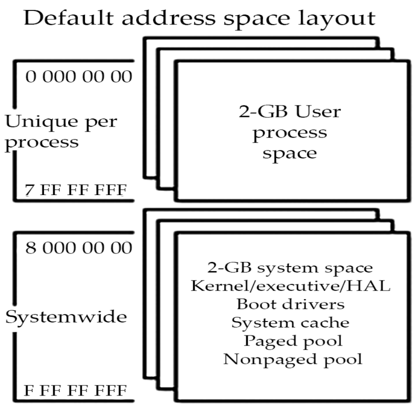


**Figure 7.2.** Default address space layout.

*Source: https://askaresh.com/2008/11/20/32-bit-memory-management-explained/.*

## 7.3. STATIC VERSUS DYNAMIC LOADING

The decision between static or dynamic loading is to be made at the hour of PC program development. In case you need to stack the program statically, the overall programs must be accumulated and connected without leaving any surface program or module reliance. The linker consolidates the item program with other fundamental object modules into a comprehensive program, which additionally incorporates logical addresses.

In case you are composing a dynamically loaded program, your compiler shall accumulate the program and for every one of the modules which you need to integrate robustly, just references will be given and remainder of the work shall be performed at the hour of performance (Høiland-Jørgensen, 2018).

At the hour of loading, consisting of static loading, the comprehensive program (and data) is included into memory for implementation to begin. Assuming that you are utilizing dynamic loading, dynamic library schedules are put away on a circle in relocatable structure and are added into memory just when they are required by the program.

(A) Static Loading and (B) Dynamic Loading:

A. The standard program is connected and complied without reliance on external programs.

B. All the modules are loaded progressively. The operator gives a reference to every one of them and the remainder of the work is done at execution time.

A. Direct information and program are loaded into the memory to begin execution.

B. Loading of information and data takes little by little in run time.

A. The linker joins the article program with other item modules to make a solitary program.

B. The connecting process happens progressively in a flexible structure. Information is loaded into the memory just when it is required in the program.

A. The handling speed is quicker as no documents are refreshed during the handling time.

B. The handling speed is slower as records are transferred at the hour of handling.

A. The code might be executed whenever it is loaded into the memory.

B. Execution happens just when it is required.

A. Static loading is done uniquely on account of organized programming language, for example, C.

B. Dynamic loading happens on account of product group programming language like C++, Java, and so on.

A. The main disadvantage is the wastage of memory since, when the code is loaded, it may or probably won't be executed.

B. The main advantage of dynamic loading is proficient memory use.

The choice of picking between static and dynamic loading technique likewise relies upon the program size. For example, in case the program is too huge, its modules ought to be chosen and loaded into the primary memory according to the prerequisites applying the dynamic loading technique. Alternatively, static loading is the most ideal choice when the programs are more modest in size.

## 7.4. STATIC VERSUS DYNAMIC LINKING

As clarified above, whenever static linking is applied, the linker consolidates any remaining modules required by a program into a solitary executable program to prevent any runtime reliance.

In dynamic linking, it isn't expected to interface the real module or library with the program, instead a reference to the unique module is given at the hour of assemblage and connecting. Windows Dynamic Link Libraries (DLL) is a genuine illustration of dynamic libraries.

The primary distinction of static and dynamic linkage is that static connection duplicates all library modules utilized in the program into the last executable record at the last phase of the computation while, in dynamic linkage, the connecting happens at run time when both implementable documents and libraries are added in the memory.

For the most part, a PC program is an arrangement of steps in a programming language that directs the PC or the CPU to play out a specific assignment. Despite the fact that the software engineer comprehends this program, the PC doesn't. Consequently, it is important to change over the source code to machine code (Hellmund, 2016). Additionally, this program could require different programs or libraries. In such cases, it is important to carry those programs and libraries along with the program to execute it. Subsequently, linking is the most common way of joining external programs with the developer's program to execute it effectively. By and large, there are two connecting systems, the static and dynamic linkage.

### 7.4.1. What Is Static Linking?

Static linking duplicates every one of the libraries expected for the program into the last executable document. The linker plays out this errand, and it is the last advance of aggregation. The linker joins the important libraries with the program code to determine outside references. Ultimately, the linker creates an executable document appropriate for stacking into memory. The last genuinely connected document contains the calling program and called programs. For the most part, these records are huge on the grounds that they are associated with different documents.

Accepting that there are a few changes in external programs. All things considered, it is important to recompile and once again interface. Any other way, the current executable document doesn't mirror these changes. Besides, in factual connecting, every one of the modules and libraries are accessible as a solitary executable module. Consequently, this connecting is quicker and doesn't cause correspondence issues.

### 7.4.2. What Is Dynamic Linking?

In powerful linking, the names of the outer libraries/shared libraries are duplicated into the last executable; in this way, the genuine connection happens at run time when the executable document and libraries burden to the memory. The working framework performs dynamic connecting. By and large, there is just one duplicate of a common library in the memory (Irwansyah et al., 2018). Subsequently, the size of the executable record is lower. It is feasible to refresh and recompile the outer libraries. Also, assuming the common library code is presently accessible in memory, there will be minimal loading time.

As a rule, in dynamic liking, it is smarter to have a viable library. Assuming that there is a change in the library, the application needs to figure out how to make it viable with the new form of the library. Also, eliminating the library can cause the program not to operate further.

## 7.5. CONTRASTS BETWEEN STATIC AND DYNAMIC LINKING

### 7.5.1. Description

Static linking is the method involved with duplicating all library modules applied in the program into the last executable picture. Conversely, dynamic linking is the method involved with stacking the outer common libraries into the program, and afterward ties those common libraries progressively to the program. Hence, this is the primary contrast between static and dynamic linking.

### 7.5.2. Occurrence

Additionally, static linking is the final phase of compilation, whereas dynamic linking happens at run time.

### 7.5.3. Document Size

Whereas statistically linked documents are bigger in scope, dynamically linked records are more modest in size.

### 7.5.4. Load Time

Additionally, static linking takes steady loading time while dynamic linking requires less loading time. Thus, this is one more contrast between static and dynamic linking (Jaeger, 2008).

### 7.5.5. Compatibility

Besides, there can be no compatibility issues on static linking. Simultaneously, there will be similarity issues with dynamic linking.

In a nutshell, static and dynamic linking are pair of linking systems. The fundamental distinction among static and dynamic linking is the former duplicates all library modules utilized in the program into the last executable record at the last advance of the assemblage whereas in dynamic linking, the connecting happens at run time when both executable documents and libraries are put in the memory.

## 7.6. SWAPPING

Swapping is a system where an interaction can be exchanged briefly out of primary memory (or move) to elective storage (disk) and ensure the memory is accessible to different cycles. Later, the framework exchanges back the interaction from the auxiliary storing to principal memory. However, execution is normally impacted by swapping process yet it helps in running different and huge cycles in equal and that is the explanation. Swapping is otherwise called a method for memory compaction (Figure 7.3) (Jeong et al., 2012).

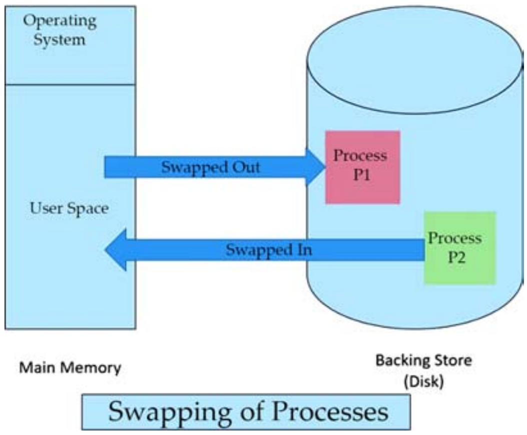


**Figure 7.3.** Swapping between main memory and backing store.

*Source: https://binaryterms.com/swapping-in-operating-system.html.*

Swapping is a memory administration system in which any interaction can be briefly traded from fundamental memory to optional memory so the principal memory can be made accessible for different processes. It is utilized to further develop fundamental memory usage. In optional memory, where the exchanged process is put away is called trade space.

The reason for the trading in working framework is to get to the information present in the hard plate and carry it to RAM with the goal that the application programs can utilize it. What to recollect is that swapping is utilized just when information is absent in RAM. Albeit the method involved with swapping influences the exhibition of the framework, it assists with running bigger and more than one interaction. This is the motivation behind why swapping is additionally known as memory compaction.

The idea of swapping can be subdivided into two additional ideas: swap-in and swap-out:

- **Swap-In:** It is a technique for eliminating a program from a hard disk and returning it to the primary memory or RAM (Kushwaha and Kushwaha, 2011).
- **Swap-Out:** It is a technique for eliminating a cycle from RAM and adding it to the hard disk.

*Example:* Considering the process's size is 2,048 Kb and is a common hard disk, whereby swapping covers a data transfer ratio of 1 Mbps. To determine how long it shall take to move from primary memory to auxiliary memory, the following equation can be made:

- User process size = 2,048 Kb.
- Data exchange rate (1 Mbps = 1,024 kbps).
- Time = process scope/exchange rate.
- = 2,048/1,024.
- = 2 seconds.
- = 2,000 milliseconds.
- Factoring swap in and swap out duration, the cycle will take 4,000 milliseconds.

Benefits of swapping:

- It assists the CPU with managing numerous cycles inside a solitary primary memory;
- It assists with making and utilizing virtual memory;
- Swapping permits the CPU to play out numerous assignments simultaneously. Accordingly, processes don't need to wait too long well before implementation (Krohn and Tromer, 2009);
- It enhances primary memory usage.

Weaknesses of swapping:

- If the PC system loses power, the individual might lose all data connected with the program if there should arise an occurrence of significant swapping activity;
- In case the swapping calculation isn't ideal, the composite technique can build the amount of Page Fault and reduce the general processing output.

***Note:***

- In single tasking operating framework, just one cycle involves the client program memory area and remains in memory until the procedure is finished.
- In multiple tasking framework, a condition emerges when every one of the dynamic cycles can't occur in the primary memory, then, a process is switched from the primary memory so different cycles can access it.

The absolute time taken by swapping process incorporates the time it takes to move the whole interaction to an auxiliary plate and afterward to duplicate the cycle back to memory, as well as the time the interaction takes to recover primary memory (Lange et al., 2011).

Expecting that the client interaction is of size 2,048 KB and on a standard hard plate where swapping will occur with move rate of around 1 MB each second. The primary swapping of the 1,000 K interaction to or from memory will take:

2,048 KB/1,024 KB each second

= 2 seconds

= 2,000 milliseconds

Considering in and out time, it will take total 4,000 milliseconds in addition to another upward where the process can recover primary memory.

### 7.6.1. Memory Allocation

Primary memory ordinarily has two segments:

- **Low Memory:** Operating framework exists within this memory.
- **High Memory:** User processes are saved in high memory.

Operating framework utilizes the below memory allocation system.

### 7.6.2. Memory Allocation and Definition

- **Single-Partition Distribution:** In this kind of allotment, migration register plot is utilized to shield user processes from one another, and from exchanging operating framework code and information. Relocation register covers value of slightest physical location while limit register comprises scope of logical addresses. Every logical address should be not less compared to the limit register.

- **Multiple-Segment Portion:** In this kind of designation, primary memory is subdivided into various fixed-sized segments where each segment ought to contain just one cycle. When the partition is released, a cycle is chosen from the information line and is stacked into the free segment. Whenever the cycle ends, the parcel opens up for another interaction.

## 7.7. CONTIGUOUS MEMORY ALLOTMENT IN OPERATING SYSTEM (OS)

In contiguous memory allotment, each interaction is contained in one adjoining part of memory. In this memory portion, all the accessible memory space stays together in one spot which suggests that the unreservedly accessible memory parcels are not spread to a great extent across the entire memory space.

In contiguous memory portion which is a memory administration strategy, at whatever point there is a request by the client interaction for the memory then a single segment of the bordering memory block is given to that cycle as per the necessity. Contiguous memory allotment is accomplished only by separating the memory into the fixed-sized segment (Lass and Gronau, 2020).

The memory can be isolated either in the fixed-sized segment or in the variable-sized segment to apportion contiguous space to client processes.

### 7.7.1. Fixed-Size Partition Scheme

This strategy is otherwise called Static partitioning. In this plan, the framework isolates the memory into fixed-size parts. The allotments might possibly be a similar size. The scale of each segment is fixed as demonstrated by the title of the method and it can't be transformed. In this partition strategy, each segment might contain precisely one interaction. There is an issue that this strategy will restrict the level of multiprogramming on the grounds that the amount of parcels will fundamentally determine the amount of cycles. Whenever any cycle ends then the fragment opens up for another interaction.

#### *7.7.1.1. Case Study*

Consider an illustration of fixed size parceling plan, we will separate a memory size of 15 KB into fixed-size allotments: It is vital to take note of that these allotments are designated to the cycles as they show up and the parcel that is apportioned to the revealed process fundamentally relies upon the calculation followed. In case there is some wastage within the parcel, it is called internal fragmentation (Levis et al., 2005).

Benefits of fixed-size partition scheme:

- This scheme is basic and not difficult to execute;
- It upholds multiprogramming as numerous cycles can be stored within the primary memory;
- The management is simple utilizing this plan.

#### *7.7.1.2. Detriments of Fixed-Size Partition Scheme*

A few drawbacks of utilizing this plan are as per the following:

- **Inner Fragmentation:** Assume the size of the cycle is lesser compared to the size of the parcel, all things considered some size of the segment gets squandered and stays unused. This wastage inside the memory is for the most part called internal fragmentation.
- **Restriction on the Process Size:** In case the process scale size of an interaction is more than that of a larger measured parcel then that cycle can't be stacked into the memory. Because of this, a condition is forced on the size of the interaction and it is: the size of the cycle can't be bigger than the size of the biggest parcel (Lin and Ye, 2009).
- **External Fragmentation:** It is one more downside of the fixed-size parcel as absolute unused space by different allotments can't be utilized to stack the cycles, despite the fact that there is enough room yet it isn't in the contiguous design.
- **Level of Multiprogramming is Less:** In this partitioning plot, as the scale of the segment can't change as indicated by the size of the cycle. In this manner the level of multiprogramming is exceptionally less and is permanent.

### 7.7.2. Variable-Size Partition Scheme

This program is otherwise called dynamic partitioning and appeared to defeat the disadvantage, i.e., internal fragmentation that is brought about by Static dividing. In this apportioning, scheme allotment is done progressively.

The scale of the segment isn't proclaimed at first. Whenever any cycle shows up, a segment of size equivalent to the size of the interaction is made and afterward dispensed to the interaction. Along these lines the size of each parcel is equivalent to the scale of the cycle.

As segment size fluctuates as indicated by the need of the interaction so in this parcel design there is no internal separation.

### *7.7.2.1. Benefits of Variable-Size Partition Scheme*

A few advantages of the segment scheme are as per the following:

- **No Internal Fragmentation:** Since the partition scheme space in the primary memory is distributed uniquely as indicated by the necessity of the cycle, along these lines there is no way of internal fragmentation. Likewise, there will be no unused space left in the parcel (Mayoral et al., 2017).
- **Level of Multiprogramming is Dynamic:** Since there is no internal fragmentation within this partition plot because of which there is no vacant space in the memory. Consequently, more cycles can be stacked into the memory simultaneously.
- **No Restraint on the Size of Process:** In this parcel as the segment is assigned to the interaction progressively subsequently the scale of the cycle can't be confined on the grounds that the segment size is determined by the interaction scope.

### *7.7.2.2. Drawbacks of Variable-Size Partition Scheme*

A few disadvantages of utilizing this segment conspire are as per the following:

- **External Fragmentation:** Since there is no internal fragmentation which is a benefit of utilizing this partition design, it doesn't mean there will be no external partitioning. The vacant space in memory can't be distributed as no spreading over is permitted in coterminous allotment. Since the standard says that interaction should be persistently present in the primary memory to get executed. In this manner it brings about External Fragmentation (Muehlstein et al., 2017).
- **Tough Implementation:** The execution of this partition design is hard when contrasted with the Fixed Partitioning plan as it includes the allotment of memory during run-time, rather than

during the framework design. The OS monitors every one of the allotments however here portion and deallocation are performed consistently, and parcel size will be changed at each time so it will be hard for the working framework to oversee every process.

## 7.8. FRAGMENTATION

As processes are stacked and taken out from memory, the freed memory space is partitioned into small sections. It occurs after in some cases that cycles can't be distributed to memory blocks considering their little size and memory blocks stay unused. This issue is known as Fragmentation.

It is an undesirable issue in the operating framework where the cycles are stacked and dumped from memory, and free memory space is divided. Processes can't be allocated to memory blocks because of their small size, and the memory blocks stay unused (Manzalini and Crespi, 2016).

Contiguous memory assignment dispenses space to processes at whatever point the cycles enter RAM. The RAM spaces are separated either by fixed partitioning or through dynamic apportioning. Since the process is stacked and dumped from memory, these regions are divided into small bits of memory that can't be distributed to the next cycles.

Fragmentation is of two kinds:

- **External Break:** Absolute memory space is to the point of fulfilling a request or to live an interaction in it, yet it isn't contiguous, thus it can't be utilized.
- **Internal Break:** Memory block allocated to process is larger. Some aspect of memory is left unused, as it can't be utilized by another cycle.

External fragmentation can be decreased by compaction or mix memory substance to put all free memory together in one huge square. To make compaction attainable, migration should be dynamic.

The inner fragment can be decreased by successfully allocating the tiniest segment yet huge enough for the interaction.

- **Reasons for Fragmentation:** Client processes are stacked and dumped from the primary memory, and cycles are kept in memory blocks in the primary memory. Many spaces remain after process stacking and trading that another cycle can't stack because of their scale. Primary memory is accessible, yet its space is inadequate to stack another process as a result of the dynamical distribution of primary memory processes.

### 7.8.1. Internal Fragmentation

When a process is allotted to the memory block, and assuming the process is more modest than how much memory is required, a free space is made in the particular memory block. Because of this, the free space within the memory block is unused creates internal discontinuity. For Instance: Considering that memory allocation in RAM is performed utilizing fixed partitioning (that is., memory squares of fixed sizes). 2 MB, 4 MB, 4 MB, and 8 MB are the accessible sizes. The OS utilizes a section of this RAM (Bala et al., 2015).

Considering a P1 with a scale of 3 MB shows up and is provided a memory square of 4 MB. Thus, the 1 MB of free space in this square is unused and can't be utilized to assign memory to another platform. It is called internal fragmentation.

#### *7.8.1.1. Preventing Internal Fragmentation*

The issue of internal fragmentation might arise because of the fixed scales of memory blocks. It very well might be solved by relegating space to the cycle through dynamic apportioning. Dynamic partitioning allots just the amount of space mentioned by the process. Accordingly, there isn't any internal fragmentation.

### 7.8.2. External Fragmentation

It occurs when a unique dynamic memory strategy distributes some memory yet leaves a modest quantity of memory unusable. The ratio of accessible memory is considerably diminished assuming that there is an excess of external fragmentation. There is sufficient memory space to finish a request, yet it isn't contiguous. It's called external fragmentation.

#### *7.8.2.1. Preventing External Fragmentation*

This issue happens when you distribute RAM to processes continually. It is present in paging and segmentation, whereby memory is dispensed to processes non-contiguously. Subsequently, in the event that you eliminate this condition, external fragmentation would be diminished (McClean et al., 2013).

Compaction is one more technique for eliminating external fragmentation. External fragmentation might be diminished when dynamic fragmentation is utilized for memory designation by joining all free memory into a single massive square. The bigger memory block is utilized to apportion space in light of the necessities of the new processes. This strategy is otherwise called defragmentation.

## 7.8.3. Benefits and Drawbacks of Fragmentation

There are different benefits and drawbacks of fragmentation which are as per the following:

- **Quick Data Writes:** Data write in a framework that upholds data fragmentation might be quicker than rearranging data storage to allow contiguous data writing.
- **Less Failures:** Assuming there is inadequate sequential space in a framework that doesn't support fragmentation, the write will come up short.
- **Capacity Optimization:** A fragmented structure could conceivably make a better utility of storage appliance by making use of the storage block.
- **Detriments:** There are different detriments of fragmentation. Some of them are as per the following:

### *I. Need for Constant Defragmentation*

A highly fragmented storage product's performance will debase with time, requiring the necessity for tedious defragmentation tasks (Monaco et al., 2013).

### *II. Reduced Read Times*

The duration it takes to go through a non-sequential document could increase as a storage device turns out to be more divided.

To put it plainly, internal, and external fragmentation are normal cycles that cause either memory loss or void memory space. Be that as it may, the issues in the two cases can't be totally overcome, in spite of the fact that they can be diminished somewhat utilizing the measures given above.

## 7.9. PAGING

Paging is a memory administration strategy where process address space is divided into squares of a similar size known as pages (scale is power of 2, between 512 bytes and 8,192 bytes). The scale of the process is estimated in the quantity of pages (Figure 7.4).

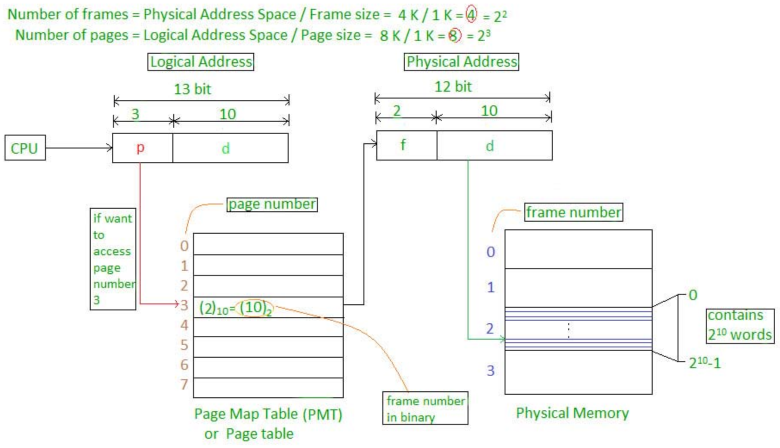

Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2
Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3
Logical Address
13 bit
3
10
Physical Address
12 bit
2
10
CPU
p
d
f
d
page number
if want to access page number 3
(2)10 = (10)2
frame number
contains 2^10 words
0
2^10-1
frame number in binary
Page Map Table (PMT) or Page table
Physical Memory

**Figure 7.4.** Flowchart diagram of the paging process.

*Source: https://tutorialspoint.dev/computer-science/operating-systems/operating-system-paging.*

Also, primary memory is divided into tiny fixed-sized squares of (physical) memory known as frames, and the frame size is kept equivalent to that of a page to have ideal usage of the principle memory and to keep away from external fragmentation. This plan allows the actual location space of interaction to be non-coterminous.

- Logical Address or Virtual Address (addressed in bits): A location created by the CPU.
- Logical Address Space or Virtual Address Space (addressed in words or bytes): The arrangement of all logical addresses created by a program.
- Physical Address (addressed in bits): A location really accessible on memory unit.
- Physical Address Space (addressed in words or bytes): The arrangement of all actual addresses relating to the logical addresses.

***Example:***

- In the event that Logical Address = 31-piece, Logical Address Space = 231 words = 2 G words (1 G = 230).
- In the event that Logical Address Space = 128 M words = 27 * 220 words, then, at that point, Logical Address = log2 227 = 27 pieces.
- If Physical Address = 22-piece, Physical Address Space = 222 words = 4 M words (1 M = 220).
- If Physical Address Space = 16 M words = 24 * 220 words, then, at that point, Physical Address = log2 224 = 24 pieces.

The planning from virtual to physical location is done by the MMU which is known as paging strategy.

- The Physical Address Space is adeptly separated into various fixed-size blocks, known as frames.
- The Logical Address Space is additionally subdivided into fixed-size blocks, known as frames.
- Page Size = Frame Size.

Consider this example:

- Physical Address = 12 pieces, then, at that point, Physical Address Space = 4 K words.
- Logical Address = 13 pieces, then, at that point, Logical Address Space = 8 K words.
- Page size = frame scale= 1 K words (assumption) (Moore and Stouch, 2016).

Address produced by CPU is segmented into:

- **Page number(p):** Amount of bits needed to address the pages in Logical Address Space or Page number.
- **Page offset(d):** Amount of bits needed to address specific word in a page or page size of Logical Address Space or word integer of the page or page offset.

Physical address is separated into:

- **Frame number(f):** Amount of bits expected to address the Physical Address Space frame or Frame number.

- **Frame offset(d):** Amount of bits needed to address specific word in a frame or casing size of Physical Address Space or word integer of a casing or frame offset.

The hardware execution of page table should be possible by utilizing specific registers. However, the use of register for the page table is good provided that page table is minor. In the event that page table contains a large number of sections, it is possible to utilize a TLB (translation Look-aside buffer), a unique, small, quick look into equipment cache (Nollet et al., 2004).

- The TLB is acquainted, fast memory;
- Every passage in TLB comprises of two sections: a tag and value;
- At the point when this memory is utilized, then, at that point, something is contrasted and all labels simultaneously. If the item is found, then, at that point, relating value is returned.

Primary memory access time = m

Assuming page table are kept in primary memory:

Viable access time = m(for page table) + m(for specific page in page table)

A PC can address more memory than the sum physically introduced on the framework. This additional memory is really called virtual memory and it is a part of a hard that is set up to imitate the PC's RAM. Paging method assumes a significant part in applying virtual memory.

### 7.9.1. Benefits and Disadvantages of Paging

Here is a rundown of benefits and drawbacks of paging:

- Paging decreases external fragmentation, yet at the same time experience the negative effects of internal fragmentation;
- Paging is easy to execute and expected as a proficient memory management method;
- Because of equivalent size of pages and frames, swapping is exceptionally simple;
- Page table needs additional memory space, so might not be great for a framework having minimal RAM.

## 7.10. ADDRESS TRANSLATION

In fostering the virtualization of the CPU, researchers zeroed in on an instrument known as limited direct execution (LDE). The thought behind LDE is basic: generally, let the program run directly on the equipment; notwithstanding, at specific central issues on schedule, (for example, when an interaction gives a structural call, or a clock hinderance happens), orchestrate so the OS reaches out and ensures the "right" thing occurs. Hence, the OS, with a little equipment support, makes an honest effort to move of the running project, to convey a productive virtualization; in any case, by intervening at those basic moments, the OS guarantees that it keeps up with command over the equipment. Effectiveness and control together are two of the principle objectives of any advanced working structure (Nimodia and Deshmukh, 2012). In virtualizing memory, researchers seek after a comparable technique, accomplishing both proficiency and control while giving the ideal virtualization. Productivity directs that they utilize equipment support, which at first will be very simple (e.g., only a couple of registers) yet will develop to be genuine implementation (e.g., TLBs, page-table help, etc.). Control suggests that the OS guarantees that no application is permitted to get to any memory however its own; consequently, to shield applications from each other, and the OS from applications.

Page address is known as logical address and provided by page integer and the offset:

Logical Address = Page number + page offset

Frame address is known as physical address and numbered by a frame integer and the offset:

Physical Address = Frame number + page offset

A data system known as page map table is utilized to monitor the connection between a frame page and interaction to a physical memory frame.

When the framework assigns a frame to some page, it makes an interpretation of this consistent location into a physical address and make entry into the page table to be utilized all through implementation of the program. When a cycle is to be implemented, its comparing pages are stacked into any suitable memory frames. Assuming you have an 8 Kb program yet your memory can oblige just 5 Kb at a given moment, then, at that point, the paging idea will come into picture. At the point when a PC runs out of RAM, the OS will move inactive or undesirable pages of memory to optional memory to let loose RAM for different cycles and brings them back when required by the program. This process continues during the entire execution of the program where the OS continues to eliminate inactive pages from the primary memory, and consider them onto the secondary memory and bring them back when expected by the program.

The generic method which can be considered apart from the LDE, is something alluded to as hardware-style address translation, or simply address translation. With address translation, the equipment changes every memory access (e.g., guide fetch, loading, or store), changing the virtual location given by the guidance to an actual location where the actual data is really found. Subsequently, on every single memory reference, a location interpretation is performed by the equipment to divert application memory references to their real areas in memory (Ow, 2011). Obviously, the equipment can't virtualize memory, as it simply gives the low-level instrument to doing so productively. The OS should get involved at central issues to set up the equipment so the right interpretations occur; it should in this manner oversee memory, monitoring which areas are free and which are being used, and sensibly interceding to keep up with command over how memory is utilized.

The objective of all of this work is to make a lovely deception: that the program has its own private memory, where its own code and information dwell. Behind that augmented simulation lies the revolting actual truth: that many programs are really sharing memory simultaneously, as the CPU (or CPUs) switches between running one program and the following. Through virtualization, the OS (with the equipment's assistance) transforms the revolting machine reality into a valuable, strong, and simple to use platform.

## 7.11. SEGMENTATION

In OSs, Segmentation is a memory management procedure where the memory is partitioned into variable size fragments. Each part is called a section which can be designated to a sequence. The insights concerning each fragment are put in a table known as segment table. It is stored in one or multiple segments.

Segment table contains primarily two details on segment:

- **Base:** It is the base segment address; and
- **Limit:** It is the segment length.

Segmentation is a memory administration procedure wherein each occupation is separated into a few portions of various sizes, one for every module that contains bits that fill related roles. Each fragment is really an alternate consistent location space of the program. At the point when a cycle is to be executed, its relating division are stacked into non-contagious memory however every section is stacked into an adjacent block of accessible memory. Segmentation memory is basically the same as paging however here fragments are of variable-dimensions whereas in paging pages are of permanent size (Peter et al., 2015).

A program section contains the program's primary capacity, utility capacities, information structures, etc. The working framework keeps a section map table for each cycle and a rundown of free memory blocks alongside fragment numbers, their size and comparing memory areas in primary memory. For each portion, the table stores the beginning location of the section and the length of the fragment. A reference to a memory area incorporates a value that distinguishes a section and an offset.

## 7.11.1. Why Segmentation Is Necessary?

While paging is a common memory management procedure. It is nearer the OS compared to the User. It separates every one of the processes into the type of pages, no matter what the interaction can have a few relative capacity sets which should be stacked in a similar page.

The operating framework couldn't care less with regards to the User's perspective on the cycle. It might separate similar capacity into various pages and those pages could possibly be stacked simultaneously into the memory. It diminishes the proficiency of the framework. It is smarter to have division what isolates the interaction into the sections. Each fragment contains similar forms of capacities, for example, the principle capacity can be remembered for one portion and the library capacities can be remembered for the other section.

CHAPTER 8

# VIRTUAL MEMORY

CONTENTS

A virtual memory in the OS is a software-managed page replacement algorithm; this type of algorithm assumes that the entire working set of the process is to be kept in main memory. For this feature in an OS, memory available for a program to use can expand beyond the four gigabytes (GB) installed in the machine by using the hard disk drive as extra RAM. The virtual memory can also be termed as an artificial memory function provided by the OS using the space of your hard disk. It helps one when dealing with software and large memory file (Bala et al., 2015). Virtual memory is a mechanism that allows each process/user to have its own private virtual address space. A virtual address can be defined as a program generated integer that an application program uses to access its work space in main memory. A virtual address therefore is a pointer to physical memory" (Figure 8.1).

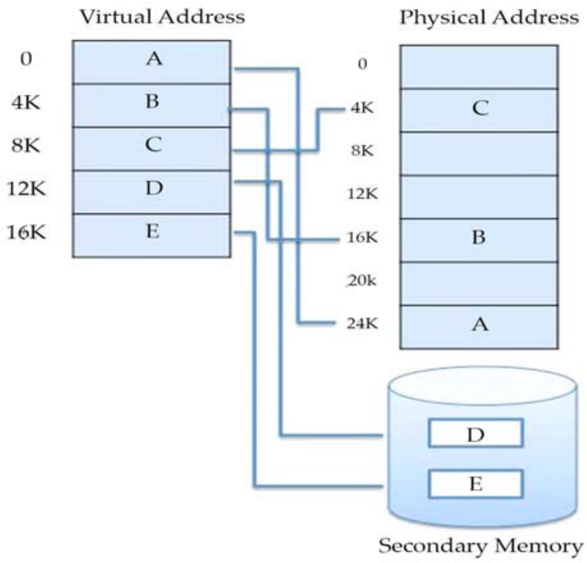


**Figure 8.1.** A visualization of virtual memory in the operating system.

*Source: Image by Tutorialspoint.*

If one's computer frequently runs out of memory while running multiple applications, virtual memory allows your system to use hard disk space as additional RAM. The OS can move pages rarely used to the hard drive, freeing up precious, physical RAM for more active pages. Increasing the amount of your paging file may help speed up applications that require a lot of memory like video-editing software or web browsers with multiple tabs. Virtual memory solves the problem of overextending the physical RAM on your device. Virtual memory is a storage area that holds frequently used programs or data in the system's RAM so they can be accessed quicker than if they had to be accessed on the hard drive. When a computer runs a program, it creates an area in memory for that program's use. This area is called virtual memory, and it is essentially a section of the hard drive that's been allocated to function as RAM (Figure 8.2).

**Figure 8.2.** A sample random access memory as used in virtual memory.

*Source: Image by Wikipedia.*

It can also be termed as page file or swap file. The swap file maps the virtual addresses used by a process to physical addresses in computer memory. The OS manages virtual address spaces and the assignment of real memory to virtual memory. Address translation hardware in the CPU, often referred to as a memory management unit or MMU, automatically translates virtual addresses to physical addresses when virtual memory is implemented (Santos et al., 2013). Virtual memory can be implemented in software using a page table. Virtual memory assigns computer pages to random access memory (RAM) and enables the OS to run with larger memory on any machine. It is important to note that memory (RAM) is too expensive for each application to have as much as it might need at any one moment. Virtual memory is a mechanism that allows the OS to extend physical memory by temporarily transferring data to disk storage. Virtual memory thus expands the size of the address space, which can be larger than the physical memory. This forms the basis of its existence.

As a memory management technique, it is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory; main storage as seen by a process or task appears as a contiguous address space or collection of contiguous segments. Software within the OS may extend these capabilities to provide a virtual address space that can exceed the capacity of real memory and thus reference more memory than is physically present in the computer. A paging system moves data between RAM and disk. Other post-virtualization techniques such as shared libraries, interpreted programs, and disk caching (not used for DASD Paging) occur only after this time. In principal main storage (real memory) is much faster than disk storage but too small in capacity; disk storage is larger but less fast. By segmenting out blocks from main storage, replacing it with disk storage, combining them with demand paging techniques, the OS can create a large, slower virtual storage from smaller, faster main storage (Figure 8.3).

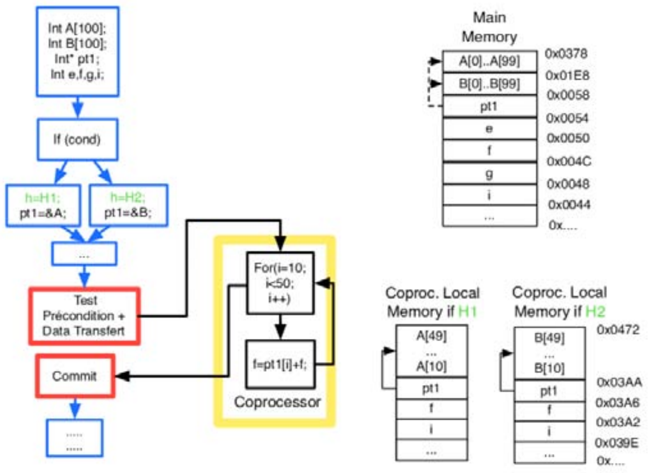


**Figure 8.3.** Memory mapping, the basic functionality of virtual memory.

*Source: Image by ResearchGate.*

A virtual memory system uses a combination of RAM and a portion of a hard drive, called a swap file or paging file, to logically expand the available memory beyond what physically exists. The virtual memory allows one to maintain consistent memory despite the physical components changing. It handles this by creating a Page file on your hard drive and uses it as the swap file, where information that is currently not in use is stored temporarily A (Ksoy et al., 2017). This method has its drawbacks, primarily the performance hit due to hard drive access speeds being slower than RAM, but it does allow for uses such as being able to use more virtual RAM than you have installed. This technique virtualizes a computer architecture's various hardware memory devices, allowing a program to be designed as though there is only one type of memory (RAM) (Figure 8.4).

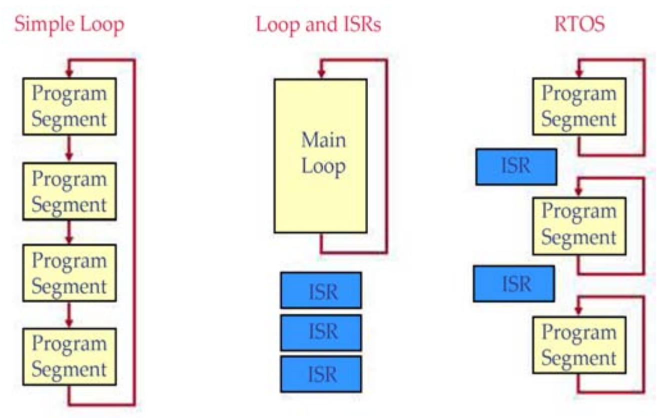


**Figure 8.4.** One line code multitasking kernel. It was initially the purpose of virtual memory to work on this.

*Source: Image by Embedded.*

The virtual memory was initially developed for multitasking kernels. This technique virtualizes a computer architecture's various hardware memory devices, allowing a program to be designed as though the computer has (for example) only one kind of memory, "random access memory" (RAM), when it might in fact have several different kinds of memory and longer latency for some than for others. Virtual address space design protects the actual hardware from an errant program. This new virtual memory in the OS will definitely change the way people use their PCs. It is amazing that today's hard disk drive, compared with the pre-war computer, almost reached the level of thousands of times. Virtual memory organizes the usage of both RAM and hard disk storage to ensure that programs work efficiently. For this reason, virtual memory often uses multiple partitions on your computer's hard disk drive.

Virtual memory is a huge part of what allows modern computers to function. Without it, your code would have to have access to all the memory it needed at all times (it would be said to be "resident" in memory), or you'd have to manage swapping the code in and out of memory yourself (Ahmad et al., 2013). Virtual memory allows for your applications and OS to have more memory available than is physically installed on the machine, which means that you can run many programs concurrently, or at least give the illusion of doing so. Because virtual memory utilizes drive space, it inherently slows down program execution and data access. However, virtual memory can be allocated to provide a swap file for temporary storage and retrieval of program data. Windows 9x uses by default a swap file of 2.5 times the amount of RAM, while Windows NT uses a default swap file of 1.5 times the amount of RAM. The OS manages the information stored in RAM to ensure that a program is placed into a region of memory in which it can run without being interfered with by another program. The OS accomplishes this virtual memory by dividing memory into regions called pages, which are assigned at execution time to executing programs (page mapping).

## 8.1. IMPLEMENTATION OF VIRTUAL MEMORY

Implementation of virtual memory in computing is the process by which memory addressing is supported transparently to the application by the computer hardware and OS. It is based on the principle that main memory is usually too small to accommodate all data and programs with reasonable response time. The idea behind virtual memory is to allow a program to address more memory than actually available in the computer. Virtual memory is a memory management technique that provides an "idealized abstraction of the storage resources that are actually available on a given machine" which "create the illusion to users of a very large (main) memory" (Figure 8.5).

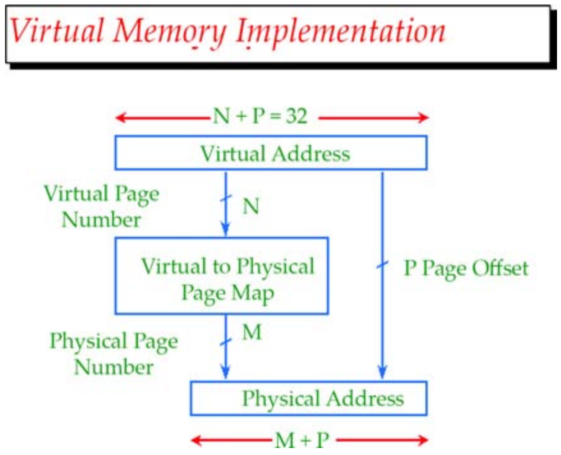


**Figure 8.5.** An illustration of the implementation of virtual memory.

*Source: Image by People.*

Implementation of virtual memory in computing further refers to handling the memory management aspects of a system such as demand paging, page replacement, segmentation, and swapping. The implementation of virtual memory in computing is a method that allows for the OS to separate its processes' memory into a disk in order to generate an address space. Virtual memory can be implemented into a computer system using a number of methods, but the implementation of virtual memory in computing is generally done through a process called 'paging.' This involves mapping virtual addresses to physical addresses through the use of 'memory pages' that are sections of physical memory allocated to each process (Figure 8.6).

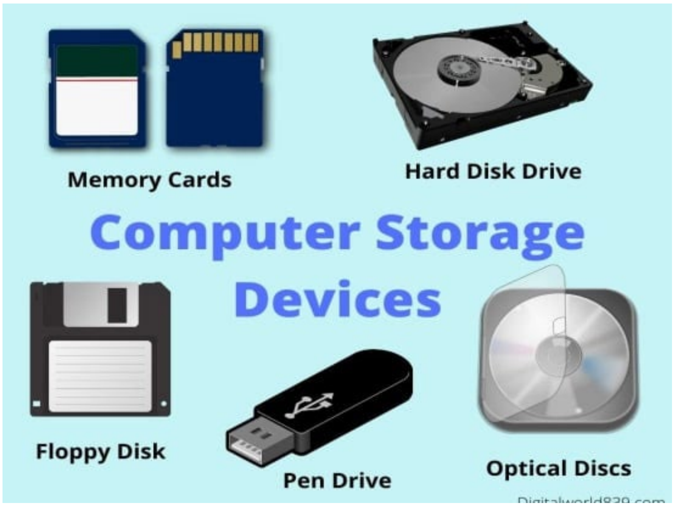


**Figure 8.6.** Computer hardware memory devices.

*Source: Image by Digitalworld839.*

Virtual memory is a memory management technique developed for multitasking kernels. This technique virtualizes a computer architecture's various hardware memory devices (e.g., RAM, hard disk, CD-ROM) so that multiple processes (i.e., programs) can be executed independently in the same address space (i.e., with their own view of the virtual address space). Each separate process can be allocated its own quota of virtual memory by associating "pages" from the main system's physical memory; thus, each process is given the impression that it is running on a machine with as much physical memory as it "sees" (which may actually be far less than what the entire physical machine has). Virtual memory also allows for greater multitasking capacity since a program which would otherwise use up too much physical memory can be assigned some other pages of virtual memory which are swapped in and out when need (Sjöstrand et al., 2015). During this swapping or "virtual memory," some pages at the end of the program are saved to space on the hard drive temporarily while others are brought into those empty slots. This allows several programs to share resources and space in physical memory simultaneously; if they do not need access to those resources at exactly the same time, this swapping or "virtual memory" (Figure 8.7).

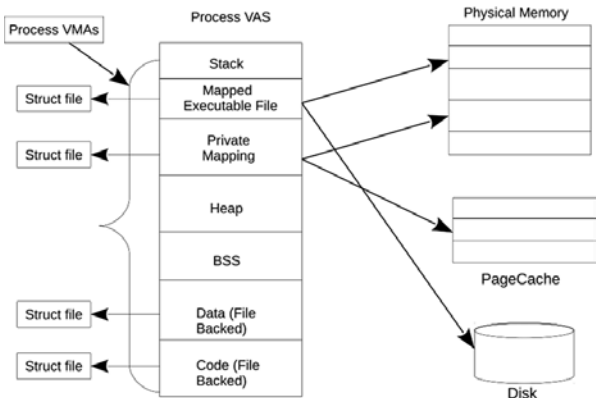


**Figure 8.7.** The concept of address space.

*Source: https://www.researchgate.net/figure/An-example-of-a-process-virtual-address-space_fig1_344039145.*

This technique virtualizes the concept of the address space, allowing processes to use memory that they do not physically own, or requires no physical memory at all the address space is divided into pages and frames (page frame). When a program is launched, it may not be available in the main memory. It must be brought from external storage to an address space. A special graphic window is used to display all text being entered into the system and to show all text output by the system (Androulaki et al., 2018). In addition, the majority of windowing systems provide at least one window, which is used as a small display surface and a keyboard input mechanism, called a cursor. A computing platform for developing and running applications (also called a runtime environment), where the OS, database management system and development tools can be installed.

As we shall see, virtual memory is actually a form of very modular caching. It separates the set of logical addresses that the program sees from actual main memory addresses. This separation ensures that programs access memory in an appropriate way and are largely isolated from locations of data in main memory. A virtual memory system is an abstraction layer on top of the real hardware memory which allows processes (software) to use more memory than they physically own. Virtual memory is implemented by creating page tables that contain references to blocks of physical memory.

## 8.2. DEMAND PAGING

Demand paging is a method of handling memory management while implementing virtual memory. Use it to facilitate multitasking and improve system efficiency by reducing the use of RAM. The process occurs as needed, rather than being executed as a matter of course. This is a technique that calls for pages to be brought into main memory only as needed by the running processes. Also known as lazy paging or on-demand, it is a method of handling memory management. Demand paging is used by some modern OSs (specifically virtual memory operations) to solve "out of memory" error when allocating pages for task. In such a case, the OS finds a page in physical memory, using demand paging, instead of allocating pages only once, the OS takes care of the allocation while the process is being executed (Figure 8.8).

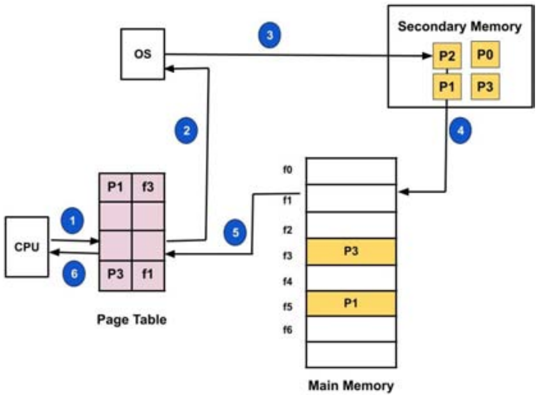
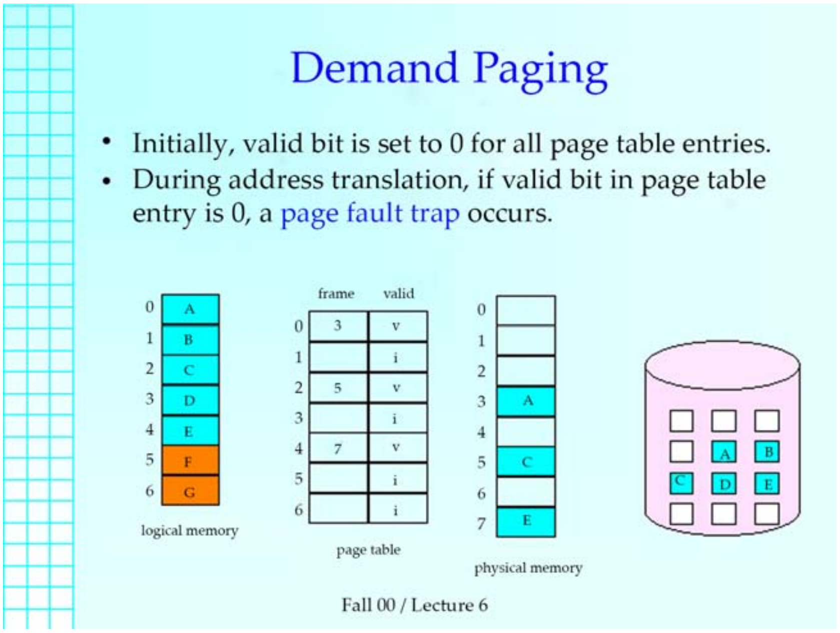


**Figure 8.8.** Demand paging and pre-paging.

*Source: https://afteracademy.com/blog/what-are-demand-paging-and-pre-paging#:~:text=In%20demand%20paging%2C%20that%20page,loads%20them%20into%20the%20memory.*

This process allows hardware to load data only when it needs to be used and thus allows the use of larger virtual memory than the physical memory. The basic principle is that instead of having all the items in working set, we keep only those pages that are constantly needed in RAM. Demand paging is a method of implementing virtual memory. True virtual memory demand-paged systems were never very common, primarily because the increase in programming complexity was not considered worthwhile for most systems (Silva et al., 2006).

Virtual memory is a hugely important computational resource. It allows the entire memory of a computer to be used by programs rather than size being limited by the amount of physical RAM. In this chapter, we explore how demand paging works as a method of implementing virtual memory through mapping of virtual address to physical addresses. We show that demand paging must consider all applications running on the machine as well as their memory requirements, and this may be problematic when implementing a method for saving, swapping out and swapping in processes in need of access to main memory (Figure 8.9).



Demand Paging

- Initially, valid bit is set to 0 for all page table entries.
- During address translation, if valid bit in page table entry is 0, a page fault trap occurs.

**Figure 8.9.** The initial stages of demand paging.

*Source: Image by Jhu.*

Nonetheless, it is a page-loading scheme that is used in virtual memory systems. In this method, pages of a process are loaded into memory only on the demand of instruction fetch by the CPU. When an address translation results in a page fault, whether due to the page being in a swapped form or not being present at all, the OS loads the required page into RAM, bringing it into memory. It is a swap-out process that essentially turns virtual memory into RAM when it is needed (as opposed to pre-loading all possible memory options into RAM and then swapping as needed).

It allows parts of each process to be loaded into memory only when they are needed, rather than at the time each process is created. This reduces the amount of RAM needed to run multiple processes at once, but increases the potential for CPU work and system I/O. Demand paging is an important method of handling memory management while implementing virtual memory (Shaw et al., 2016). Demand paging is used in conjunction with page tables, page faults, and page frames to implement a working virtual memory system. These systems allow for much larger memory pools than would normally be possible on any given machine.

Paging occurs when main memory cannot hold the entire address space of a process and places parts of the process into secondary storage. When the program needs to use that particular page from the secondary storage, it is brought back into main memory. Demand paging takes this a step further and brings pages only when needed instead of bringing them all at once as part of swapping processes. This is useful for CPU intensive programs where certain parts are not needed immediately. Demand paging as a method of implementing virtual memory is one of the most efficient memory management methods existing today. The method has been in use for several decades, and is universally accepted by OS designers.

Demand paging is a method of handling memory management at the physical level of computer operation. The page table must keep track of which physical pages may contain valid data for a given process. It does so by paging only those pages from disk to main memory that are actually needed, rather than paging in all pages from disk. The terms "demand paging" and "lazy paging" are used interchangeably. Virtual memory systems separate the memory space into fixed-size units called pages. A typical page size might be 4 kilobytes (KB), although many OSs allow the use of different page sizes and a mixture of pages within an address space. The programs in the virtual memory are divided into pages, and the real memory is divided into equal-sized partitions that are the same size as a page; each partition is opened to hold one page from the program being executed at any time. The process gets brought in from disk, the OS allocates enough space for it to execute, then it writes that memory back out onto the disk when complete. This is a great method for handling limited amounts of physical memory and running multiple processes at the same time. Demand paging is the opposite of demand feeding and a more advanced mode than the preemptive paging. It reduces the access time to data and increases throughput performance by overlaying additional pages onto underlying ones. A crossed-out page symbol (∅) in a circle with the letter P on some OSs is a warning that not enough space is left on the specified drive (Sharma et al., 2012).

The process involved starts when memory space is allocated only when an application requests for more memory. Pages are loaded into main memory from disk as needed by the process. The program in execution accesses them whenever it refers to any page not already in main memory. Pages of the current process, which have been loaded in main memory but are not being used currently, can be swapped out to make space for other processes or I/O buffers, thus freeing some main storage. This is done using a dynamic relocation algorithm and a logical-to-physical address translation mechanism (MMU) (Figure 8.10).

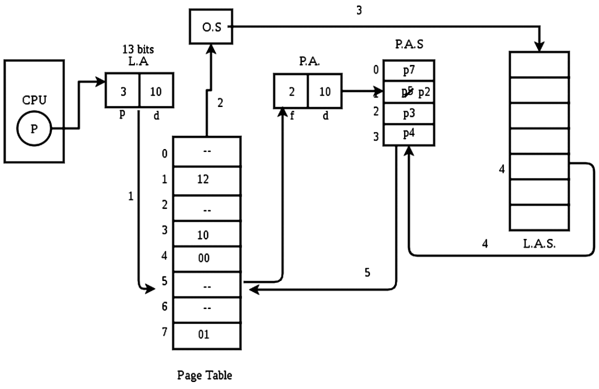


**Figure 8.10.** The process of demand paging.

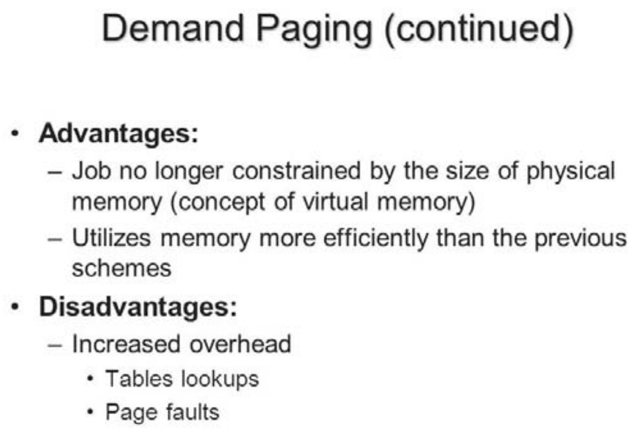
*Source: Image by GeekforGeeks.*

- It starts when the CPU tries to indicate or rather refer a particular page not available at the moment.
- The computer will hence generate an interrupt that indicates a memory access fault.

- The OS, e.g., Windows, Linux or Ubuntu will put the then interrupted process in a blocking state.
- This implies that for the execution to proceed, the OS must bring the required page into the memory.
- The OS will hence go searching for a logical address space.
- From the logical address space, it will be brought into the physical address space with the use of page replacement algorithms.
- The page table is hereby updated.
- The CPU will be signaled to hereby continue with the execution.
- Without any fault, demand paging will be successful.

## 8.2.1. Advantages of Demand Paging

Rather than trying to fit everything in RAM, the MMU stores and retrieves only what you need when you need it. Demand paging also reduces the risk of exceeding your processor's memory capacity by reducing your workload during these operations. Its major advantage is that it allows a process to be allocated a range of virtual addresses that are not physically resident on the machine. A program may be developed assuming it will have complete memory available for its use, when in fact only part of the address range is resident at any given time (Figure 8.11).



Demand Paging (continued)

- **Advantages:**
  - Job no longer constrained by the size of physical memory (concept of virtual memory)
  - Utilizes memory more efficiently than the previous schemes
- **Disadvantages:**
  - Increased overhead
    - Tables lookups
    - Page faults

**Figure 8.11.** Summarized advantages and disadvantages of demand paging.

*Source: Image by Slide Player.*

Demand paging is a mechanism in which we don't load the pages in RAM until they are needed, page faults are expensive and quite slow. But all the pages aren't needed at the same time and can be loaded when required. This requires more memory to keep track of existence of pages, RAM isn't left fragmented rather it is used effectively if a new process is allocated memory, if it needs a particular page, it can be loaded from disk to RAM instead of demanding all the processes to use first fit or best fit algorithm (Sangorrin et al., 2010).

The primary benefit of demand paging is that it allows the physical memory of a computer system to be used much more efficiently than demand paging. With demand paging, if a process references a page that is not resident in physical memory, the processor issues a page fault. The OS then allocates space in physical memory and loads the required page into this newly-allocated space. However, if the page was previously swapped out and exists on the swap device, then the OS can either read the entire page back into physical memory (and thus occupy a full frame), or it can read just part of each page back into memory (rather than reading all 4 KB for each page). In this manner, multiple pages may be loaded into memory and share a single physical frame.

The advantage of demand paging over swapping is that it allows a process only to pay for those pages it needs (or the OS thinks the process needs). In contrast, the process must be continuously kept in memory with swapping even if those are rarely or never used. When demand paging is used, memory use can be distributed more efficiently and the system can run with a full working set of pages as many programs experience a locality-of-reference effect.

### 8.2.2. Disadvantages of Demand Paging

It might be an unfair question. The more pertinent question may be: when not to use demand paging? The advantages are mostly obvious, and for a typical user with a typical machine, the disadvantages may not be obvious at all. One of the problems with demand paging is locality of reference. Locality refers to frequently accessed data. If a page you need is not in memory, then it takes time to bring it in from disk. That is known as a page fault (Singh, 2014). While it is bringing the page into memory, your program has to wait (contrast this with paging systems that use pre-paging). Locality suggests that you will be accessing pages that have been recently accessed. Contrast this with the situation on some old time-share computers where everyone's jobs were lined up together and ran round-robin. They had no locality since they essentially picked random pages each time.

Furthermore, not all programs benefit from paging, Program layout affects the performance of demand paging, Memory may become rapidly fragmented and Program size tends to increase due to overlaying. If the user has selected too small a size for the unit of paging, then there is excessive internal fragmentation, which results in wastage of memory and decreased efficiency. Physical memory storage is expensive and limited compared to virtual memory. When the OS loads a process into memory, the loading takes much longer than normal execution, which creates processor inefficiency. The OS has times of idleness when it could be performing useful work and instead spends time waiting for input/output (I/O) operations needed to load a page from disk into memory.

## 8.3. DEMAND SEGMENTATION

Demand segmentation was a way of implementing virtual memory in many early microprocessors. As a feature of most modern computer systems, it enables programs to address more memory than actually exists in real storage. It is usually achieved by using disk storage instead of main memory as part of the system's virtual address space.

Demand segmentation is a technique for implementing virtual memory in which main memory is organized into fixed-sized blocks, called "segments." The registers are also organized as a segment, this one called "the stack" or simply "stack." (The subject of this chapter is the traditional organization of the stack and its relationship to the rest of the machine's virtual memory system.) At each execution point, some information is kept on the stack, while other information is on disk. The disk information can be brought into the main memory stack area when needed (Shin et al., 2014). When no parts of that disk data are needed any more, it is freed, thus allowing another data section to be placed there. This system makes better use of the limited and expensive desk space than do other techniques, such as paged virtual memory. Segmented virtual memory systems afford two distinct but closely related benefits. First, they allow easy access to large data structures by providing short absolute addresses within each segment. Second, they make effective use of main storage since only those portions of an active program that are currently required need be stored in main storage. From computer user's standpoint, both attributes improve performance (Figure 8.12).

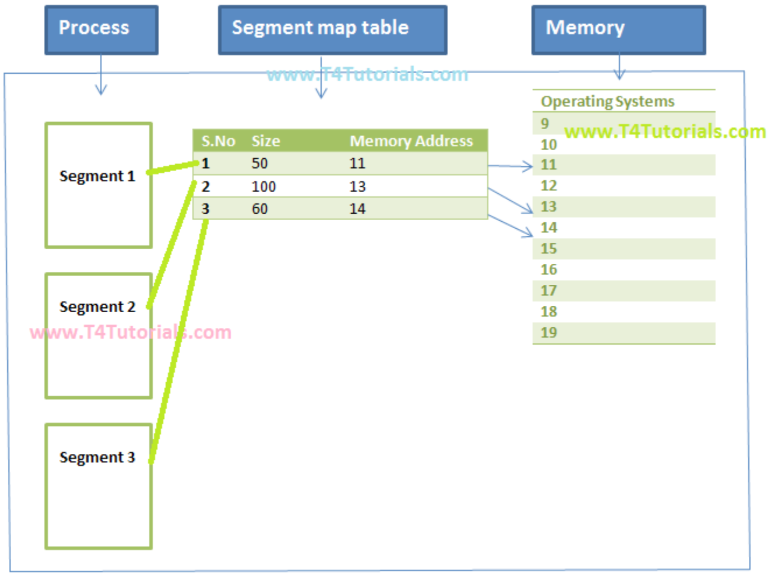


**Figure 8.12.** Demand segmentation.

*Source: Image by T4Tutorials.*

It is a scheme of implementing virtual memory by dividing the larger logical address space into fixed-size blocks called segments. Virtual addresses consist of two parts; a segment number and an offset. The segment number represents the segment table which contains the base address (starting memory address) of each segment in main memory. The program counter contains the address of the next instruction to be fetched from memory. Usually, the program counter contains only the 12 most significant bits since instructions are typically aligned on word boundaries. If a program exceeds its allotted memory space, it is said to be thrashing and can run very slowly as the OS tries to keep up with its requests for additional memory storage. It can be termed as the idea of only loading into main memory the pieces of a process or program that are currently needed. The rest of the program or process is kept on disk (Figure 8.13).

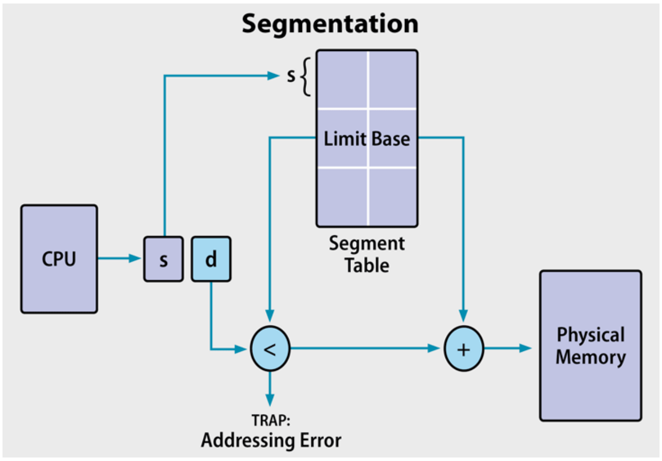


**Figure 8.13.** Illustration of the segmentation of virtual addresses.

*Source: Image by Enterprise Storage Forum.*

In converting physical memory addresses to virtual addresses, each address whose most significant (leftmost) bits correspond to a page number is translated (segmented) into a virtual address with these bits translated into an equivalent value that identifies a corresponding virtual-memory page. The less significant bits of the physical address, which correspond to an offset within the physical boundary of the real memory chip in which they reside, are simply left unaltered to serve as offsets in the corresponding virtual-address space (Tang et al., 2010). The earlier conversion of the contents of any real memory page (whose references now lie outside the currently active region) into a virtual-memory page was called sectioning. Sectioning was one way of physically implementing virtual memory.

Modern computers may support both a virtual mode of operation and an unprivileged mode of operation. In the virtual mode, instructions and accesses to memory are supplied by a monitor program which permits any user of the computer to behave as if he were in complete control of the machine; all accesses are privileged and also checked against two tables in memory to see if they are permitted. In unprivileged mode, the normal OS is in control of the computer and checks accesses on a more limited basis. Virtual memory systems permit more processes (jobs, programs) to be active than can be contained directly within physical main memory (that is, RAM). The extra flexibility this provides is usually worthwhile, given that most programs do not use all their allocated memory all the time and simply dividing up main memory between active programs would leave many parts unused or under-used.

## 8.3.1. Advantages of Demand Segmentation

An advantage of demand segmentation as a way of implementing virtual memory is that it allows more than one process to be in main memory at the same time. This increases CPU utilization and, therefore, overall system throughput. Demand segmentation as a way of implementing virtual memory can help reduce the number of logical address spaces, manage the system 'memory et. Furthermore, (i) demand segmentation is fully automatic, since OS software manages it; (ii) there is no need to select virtual memory size, since there is no preallocation of physical memory and no limit to virtual memory size; (iii) in some systems you can swap processes in and out of main memory by using a disk storage area for each process; (iv) it is easily implemented on most general-purpose computers; (v) it is more efficient in terms of space and time because when a page fault occurs the frame allocated to the faulting logical page contains only the data needed by that process (Tsolakis et al., 2019).

It is also related to demand paging. Demand paging organizes virtual memory by segments and stores each segment on a separate location on a hard drive. This approach gains many advantages because it allows the OS to access virtual memory that is not stored in physical memory, increase the program size beyond what fits in physical memory, allow multiple programs to run at the same time (multitasking), allow for the sharing of files and resources by multiple users and processes, create an effective swapping mechanism for moving data between memory locations and help bring more complex programs into memory from a larger virtual memory space.

Every process may not require the use of all its allocated memory during execution. Memory required to execute a process is called "demand set" of the process. The demand set of a process will be kept in the main memory and another part is as usual on disk. This can be achieved by dividing a logical address space into two parts; one which is held in main memory and other on disk in segments. This helps to reduce thrashing which was the main cause for decrease in OS performance for pure demand paging systems, since a process has all its pages with it, the pages are likely to be re-referenced again and again thus decreasing thrashing.

### 8.3.2. Disadvantages of Demand Segmentation

Demand segmentation and demand paging are two common methods for implementing virtual memory. While demand segmentation has the advantage of loading less pages at first, it imposes more overhead on the swapping out of memory. Disadvantages of demand segmentation as a way of implementing virtual memory include: slow running time (as processes are swapped in and out of physical memory), complex hardware, and not cost effective on large jobs.

Segmentation has disadvantages as a memory management technique. The segmentation hardware must translate each segmented address. This means that the performance of the system is impacted by the execution of memory reference instructions (Wentzlaff et al., 2010). Improper segmentation can result in a situation where enough segments are used that the overhead associated with accessing them causes significant system slowdown or errors due to lack of available room in the segment table. It is also based on the principle that a program cannot use all its allocated store at any one time, so should be split into parts that can be brought into memory as needed.

Other disadvantages include; (i) A single process may need to be loaded into RAM prior to execution, and then it will be swapped out of RAM to make room for one or more other processes. Since several processes are using physical memory at the same time, the system experiences more overhead from swapping in and swapping out processes than if only one process was loaded into memory at a time. (ii) The division of memory of each application cannot be adjusted if necessary, so there might be a waste some sectors of the system's memory.

## 8.4. SWAPPING/SWAP FILE

Swapping can be used as a method of implementing a virtual memory abstraction, although doing so is now less common than it once was. When swapping is used for implementing virtual memory, pages of memory are moved between a RAM and a swap disk. The swap disk is usually a hard disk. A swap disk may, however, be any sort of block storage device. Typically, the OS will write RAM pages to a predetermined area on a hard drive when they are no longer needed in physical memory. The free areas in RAM are then filled with data from other programs/pages as needed. Swapping is usually faster than an alternative method of implementing virtual memory, known as paging (Figure 8.14).

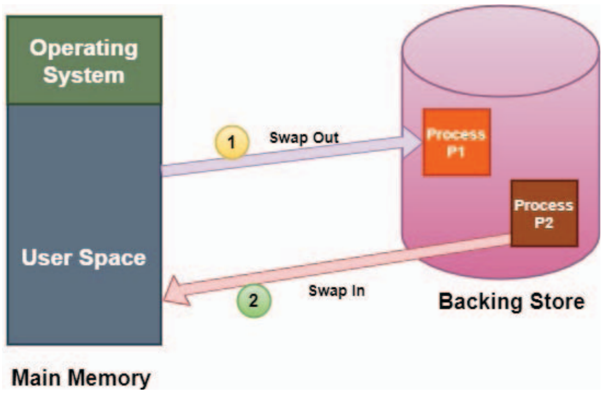


**Figure 8.14.** Swapping in the operating system.

*Source: Image by Study tonight.*

Swapping is a method for giving processes as many of the advantages of a large address space as possible. Virtual memory is implemented through paging and segmentation. The use of real memory is managed by the MMU, and portions of a process are loaded from disk into real memory when needed. The paging mechanism prevents processes from interfering with each other. Processes are moved in and out of main memory to secondary storage. A process can be swapped temporarily out of memory to a disk, and then brought back into memory for continued execution. Computer users have been familiar with virtual memory since the mid-1980s. But most users probably aren't aware that virtual memory is implemented using a technique known as swapping.

Swapping can also be termed as a method of moving processes and data from main memory to disk when they are not being used. This allows additional processes to be loaded into memory and can help the system to be more efficient. Swapping can be manual or automatic, but is typically transparent to the user. This is a process whereby a page of memory is copied to the pre-allocated space on the hard disk called swap space, then that page of memory is cleared. If the system needs more memory resources and the RAM is full, inactive pages in RAM are moved to the swap space. While swapping is less efficient than paging because of seeks times and transfer times of disks, it can be simple to implement because it never requires relocation of data (Whipple et al., 2009).

Physical memory and virtual memory are two different things. As a quick reminder from CS 170, virtual memory is a layer that lies over physical memory and consists of pages instead of frames. In other words, for each process, there is a translation table that maps virtual addresses to physical addresses in this way: (process id, virtual page number) → (physical frame number). A process can access only one set of physical frames at any given time; the rest will be swapped out by the OS kernel until needed again.

### 8.4.1. Advantages of Swapping

They include: (i) address-bounds registers can be omitted from the CPU; (ii) a program can be larger than real memory; (iii) it is easier to implement than paging; (iv) swapping creates smaller page tables than paging. This process also helps in free memory organization and producing a process; all the inactive pages are being removed from memory and held on a paging device. This is a very common way of implementing virtual memory. The overhead of keeping track of all the pages that are allocated to a given process can be reduced by using Swapping. It decreases the number of page frames required for a job, by keeping only those page frames in the physical memory that are actively being used. Swapping is a kind of scheduling, so it is done only as needed. The swap area is not allocated from the main memory unlike fixed partition method. So, user can use it more efficiently. It makes effective memory usage. So, it enhances performance.

### 8.4.2. Disadvantages of Swapping

It can suffer from external fragmentation. If many CPU bursts are small and slow processes require large contiguous memory, there may not be enough contiguous memory to swap them in. It is expensive. Data needs to be moved between disk and RAM. This data movement takes time and adds considerable overhead. Also, data has to be moved twice (once from main memory to swap area and then from swap area back to main memory).

Other disadvantages include: (i) Low-throughput. The entire process is slow, because each time a process goes to the secondary storage, there will be context switch time and then low I/O transfer rate; (ii) complex management algorithm (Quigley et al., 2009). With swapping, it is hard to determine if there should be another process swapped in or which one should be swapped out; (iii) memory with no access reference will not get swapped out. All above problems also exist for Demand Paging, but with Demand Paging these problems become much less serious compared to swapping.

## 8.5. PAGE REPLACEMENT

Page replacement is a way of implementing virtual memory. Virtual memory is a concept that allows an application to address more physical (and logical) memory than is installed on the system. It works by having the processor create an address space larger than the physical RAM and mapping addresses within that space to physical memory as an application accesses memory (Yin et al., 2016). The portion of the address space that has no corresponding physical memory is called virtual memory, and needs to be mapped to some location in real, physical memory (RAM). Virtual memory allows programs that address bigger than physical RAM. When the system needs additional physical memory for its work, it can take some of the pages that are not busy and store them temporarily on a disk, so that this space becomes available for the future memory needs (Figure 8.15).

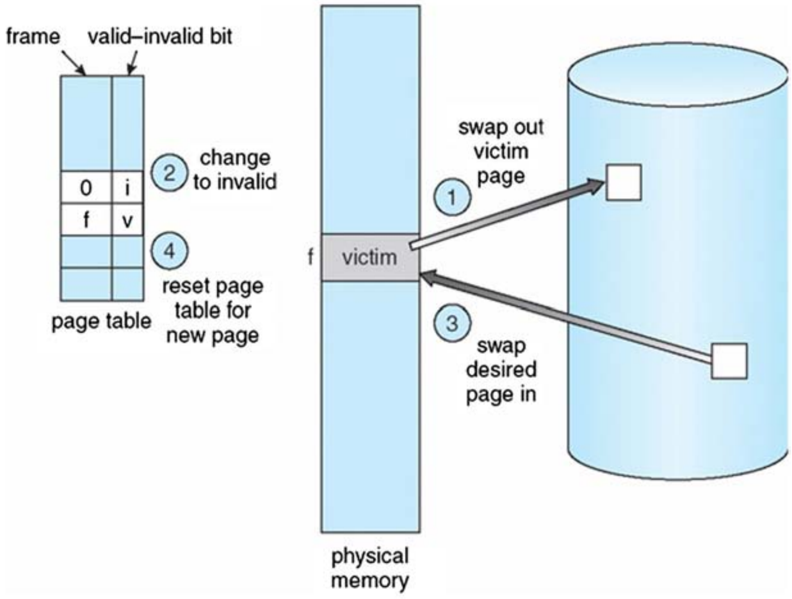


**Figure 8.15.** Page replacement algorithm.

*Source: Image by GitHub.*

Page replacement is a property of virtual memory utilized by OSs, hardware, and software side, in order to implement virtual memory. Page replacement policies are used in virtual memory implementations to select the page frame pages into the working set of a process. It is a mechanism in which a computer stores and retrieves data from secondary storage such as a hard disk drive. When there is a lack of space in memory, the page replacement algorithm moves pages from memory to disk. The page replacement algorithm decides which page needs to be replaced when new page comes in. Most OSs use paging for virtual memory management, in addition to another method of virtual memory management like segmentation. In an OS using virtual memory, there are many benefits to using an implementation like demand paging and page replacement. Some of these benefits include the capacity for multiprogramming and efficient memory utilization. A system that implements page table's takes advantage of this mechanism by allowing page faults to occur when loading programs into main memory.

To compensate for a relatively small amount of physical memory, LMOS uses a technique known as page replacing. This feature enables the system to address and access a much larger virtual memory than it has physical page-frames for storing physical pages. In other words, whatever is in virtual memory may not be in physical memory at any given time. The OS will move things from virtual to real storage just in time for their use (Arshad et al., 2018).

This technique is used to decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Whenever a new page has to be brought in, one which is already in the working set must be evicted and marked as free. The least-recently used (LRU) algorithm will discard the page that was not used for the longest time. This is an extremely efficient algorithm as it reflects real life behavior in handling pages. The FIFO policy discards the oldest (i.e., first-in) page when a new one needs to be loaded.

## 8.5.1. Advantages of Page Replacement

The major advantage of page replacement as a way of implementing virtual memory is that it can be done fairly simply and easily. The main disadvantage is that finding the appropriate page to replace at any given time can be processor-intensive.

Many of the advantages of page replacement as a way of implementing virtual memory are based on its being an effective alternative to address translation hardware. The TLB is a limited resource, and it may not support all of the pages that are simultaneously in use. Elements cannot be moved from one cache to another as easily as pages can be moved from RAM to disk. Pages can be grouped into processes or even sub processes, thereby avoiding process contention for shared elements. Pages may execute from disk without being first loaded into RAM or cache. Protection can be enforced by removing pages, rather than by trapping references to the removed pages.

CHAPTER 9

# FILE SYSTEM

## CONTENTS

## 9.1. FILE

A file is a labeled collection of related data that is stored on secondary storage devices like magnetic disks, magnetic tapes, and optical disks. A file, in principle, is a series of bits, bytes, lines, or entries whose value is established by the creator and user of the file.

A file system or filesystem (usually shortened to fs) is a technique and data structure used by the OS to regulate how data is saved and accessed. Without a file system, data stored on a storage medium would just be one huge body of data with no way of knowing where one set of data ended and the next started, or where a particular piece of data was stored when it came time to access it (Estefo et al., 2019).

The data is conveniently extracted and distinguished by dividing it into pieces and giving every component a name. Every group of data is referred to as a "file," after the manner a paper-based information management system is referred to. A "file system" is the design and logic rules employed to organize sets of data and their names.

There are numerous types of file systems. Each one has a unique structure and logic, as well as speed, adaptability, security, size, and other characteristics. Some file systems are intended to be used just for specific applications. The ISO 9660 file system, for instance, is developed exclusively for optical discs.

File systems can be employed on a variety of storage devices that utilize various types of media. Hard disk drives are still important storage devices in 2018 and are expected to be so for the foreseeable future. SSDs, magnetic tapes, and optical discs are some of the other types of media that are used. In other circumstances, such as with tmpfs, the main memory (RAM) of the computer is used to establish a temporary file system for relatively brief use.

Some file systems are utilized on local data storage devices, while others enable file access through a network protocol. Other file systems are "virtual," which means that the given "files" (known as virtual files) are computed on demand (like procfs and sysfs) or are just a mapping into another file system exploited as a backing store. The file system controls access to both the information of files and their metadata. It is in charge of organizing storage space; dependability, efficiency, and tuning in relation to the actual store medium are all significant design factors (Fröhlich and Wanner, 2008).

Prior to the invention of computers, the phrase file system referred to a method of storing and accessing paper records. By the 60s, the phrase had been extended to include automated filing in addition to its original meaning. It was widely used by 1964.

A file system is made up of two or three layers. The levels are sometimes expressly segregated, and other times the functions are mixed.

Interface with the user application is handled via the logical file system. It offers the API for file operations such as OPEN, CLOSE, READ, etc., and forwards the input to the layer under it for execution. The logical file system "controls open file table entries and per-process file descriptors" This layer is in charge of "file access, directory operations, [as well as] security and protection."

The virtual file system is the next alternative layer. "This interface supports several simultaneous versions of physical file systems, which are referred to as a file system implementation."

The physical file system is the third tier. This layer is associated with the storage device's physical functioning. It interprets physical blocks that are being read or written (Fierro and Culler, 2015). It is in charge of buffering and memory administration, as well as the physical arrangement of units on the storage media. To operate the storage device, the physical file system communicates with the device drivers or the channel (Figure 9.1).

**Figure 9.1.** File management is one of the basic and important features of operating system. Operating system is used to manage files of computer system. All the files with different extensions are managed by operating system.

*Source: https://princeabhishek410.medium.com/understanding-file-management-system-in-operating-system-4c7fbfc306f2.*

## 9.2. FILE STRUCTURE

A file structure should be in a format that the OS can interpret:

- Depending on its category, a file has a specific defined structure;
- A text file is a line-by-line succession of characters;
- A source file consists of a series of processes and routines;
- An object file is a series of bytes structured into machine-understandable blocks;
- When an OS defines a new file structure, it also includes the code to support that structure. Unix and MS-DOS support a limited number of file structures.

### 9.2.1. File Format

The capacity of the OS to discriminate between distinct types of files, such as text files, source files, and binary files, is referred to as file type. Many OS support a wide range of file formats. The following file types are found in OSs such as MS-DOS and UNIX:

1. **Ordinary Documents:**

i. These are the files that carry user information and may hold text, databases, or executable programs;

ii. The user can perform numerous actions on such files, such as adding, modifying, deleting, or removing the entire file (Deseriis, 2017).

2. **Files in a Directory:**

i. These files display a collection of file names as well as other information about these files.

3. **Specimen Files:**

i. These files are also referred to as device files;

ii. These files contain physical devices such as disks, terminals, printers, networks, and tape drives, among others.

There are two kinds of files here (Figure 9.2):

- As with terminals or printers, data is processed character by character in character special files; and
- Block special files: data is processed in blocks, similar to disks and tapes.

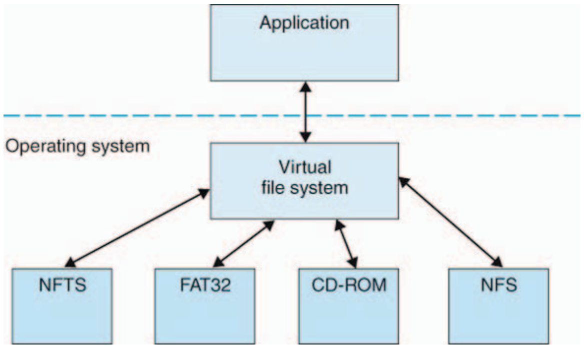


**Figure 9.2.** OS and file system.

*Source:* *http://faculty.salina.k-state.edu/tim/ossg/File_sys/file_system_stds.html.*

## 9.3. TYPES OF FILE SYSTEMS

There are many sorts of file systems, most of which are listed in subsections.

### 9.3.1. File Systems on Disk

A disk file system can randomly address data on a disk storage media within a few milliseconds. It also contains the expectation that resulted in the speed with which data was accessed. With the use of a disk file system, numerous users can access various data on the drive regardless of the data's sequential location.

A disk file system normally includes a master file directory (MFD) as well as a map of occupied and free data areas. Each and every file addition, updates, or removals necessitate the updating of the directory as well as the used/free maps. Because random access to data areas is recorded in milliseconds, this technique is suitable for drives.

### 9.3.2. Flash Storage Devices

A flash file system is in charge of flash memory's constraints, performance, and special abilities. It is preferable to use a file system developed for a flash device; nonetheless, a disk file system is the fundamental storage media that can be used with a flash memory device.

## 9.3.3. Tape File Systems

As a tape format and file system, a tape file system is used to store files on the tape. Magnetic tapes are more robust than disks for accessing data for extended periods of time, which presents issues for a broad sense file system in regards to generation and efficient administration. A tape file system is a good file system and tape format generally used for storing files on tape. Magnetic tapes are sequential storage media that have much longer random data access periods than disks, making the development and management of a general-purpose file system difficult later (Greenwald and Thomas, 2007).

To wind and unwind possibly exceedingly long reels of media, tape demands linear motion. Moving the read/write units from one side of the tape to the other could take a few seconds to many minutes.

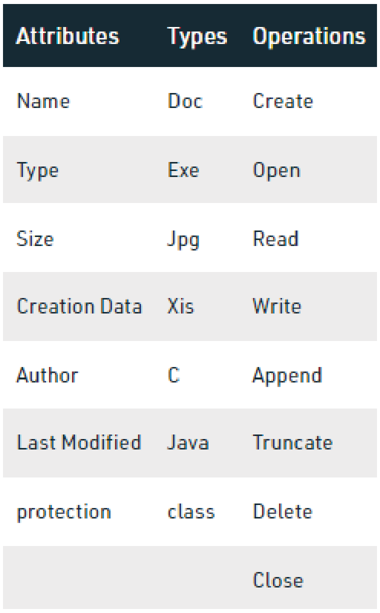
As a result, a MFD and usage map on tape might be highly slow and ineffective. Reading the block usage map to discover empty blocks for writing, updating the usage map and directory to include the data, and finally forwarding the tape to store the data in the correct position are common steps in writing.

Each subsequent file write necessitates updating the map and directory as well as recording the data, which can take numerous seconds per file.

Tape file systems, on the other hand, often allow the file directory to be scattered over the tape intermixed with the data, a process known as streaming, which eliminates the need for time-consuming and recurring tape movements to write new data.

Nevertheless, as a result of this layout, reading a tape's file directory frequently necessitates scanning the whole tape to read together all dispersed directory entries. Many data archiving software to run with tape storage will keep a local duplicate of the tape catalog on a disk file system, allowing you to rapidly add files to a tape. If a local tape catalog replica is not utilized for a set amount of time, it is normally deleted, at which juncture the tape has to be re-scanned were it to be accessed again later (Gunadi and Tiu, 2014).

The Linear Tape File System is a tape file system created by IBM. The IBM Linear Tape File System—Single Drive Edition product is an open-source implementation of this file system. The Linear Tape File System records index meta-data on a distinct partition on the tape, eliminating the issues associated with distributing directory entries throughout the tape (Figure 9.3).



| Attributes | Types | Operations |
|---|---|---|
| Name | Doc | Create |
| Type | Exe | Open |
| Size | Jpg | Read |
| Creation Data | Xis | Write |
| Author | C | Append |
| Last Modified | Java | Truncate |
| protection | class | Delete |
|  |  | Close |

**Figure 9.3.** Attributes, types, and operations of file systems.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/?ref=lbp.*

## 9.3.4. Database File Systems

Another way for managing files is to use a database-based file system. Rather than hierarchical organized administration, files are identified by their properties (such as file type, author, topic, and so on). A database-based file system is another option for file management. Files are distinguished by their attributes, such as kind of file, topic, author, or equivalent rich metadata, instead of or in addition to hierarchical organized management (Giorgetti et al., 2020).

IBM DB2 for I (previously identified as DB2/400 and DB2 for i5/OS) is a database file system that runs on IBM Power Systems and is part of the object-based IBM OS. It was designed by Soltis, IBM's former chief scientist. In the 1970s and 1980s, IBM Rochester successfully invented and implemented technology such as the database file system, which others such as Microsoft later struggled to do. These technologies are colloquially known as 'Fortress Rochester' and were, in several respects, more sophisticated technologically than early Mainframe systems in a few basic features. Other projects that aren't "pure" database file systems but make use of some database file system features.

Many web content management systems store and retrieve files using a relational database management system (RDBMS). XHTML files, for instance, are saved as XML or text fields, while picture files are stored as blob fields; SQL SELECT queries fetch the files and enabling the application of more advanced logic and richer information associations than "ordinary file systems." Many CMSs also allow you to store merely metadata in the database, with the conventional filesystem employed to store file content.

Some database file system features are used in very large file systems, as typified by applications such as Apache Hadoop and Google File System.

### 9.3.5. Transactional File Systems

Some programs, for whatever reason, require one or more changes to fail, while others require numerous file system changes yet do not make any changes. For example, when installing or updating software, a program may generate configuration files, libraries, and executables. If the software is interrupted during upgrading or installing, it may become unusable or malfunctioning. Furthermore, if the process of installing or updating the software is not completed, the entire system may become inoperable (De Hambarde et al., 2014). Some programs must either make several file system modifications or, if one or more of the changes fails for whatever reason, make no changes at all. A program that installs or updates software, for example, may generate executables, libraries, and/or configuration files. If portion of the writing misses and the software is only partly installed or updated, it may become faulty or unusable. An inadequate update of a critical system utility, like the command shell, could render the entire system inoperable. Transaction processing provides the atomicity promise, which ensures that activities within a transaction are either entirely resolved or the transaction can be canceled and all of its preliminary data are discarded. This implies that if there is a failure or a power outage, the stored state will be constant following recovery. The software will be fully installed or the unsuccessful installation will be totally reversed, but a useless partial installation will not be stored on the machine. Transactions also give the isolation guarantee, which means that operations within a transaction are concealed from other processes on the system until the transaction is confirmed, and that interfering activities on the system will be integrated properly with the transaction. Starting with Vista, Windows introduced transaction capabilities to NTFS in the form of Transactional NTFS, however its adoption is currently discouraged. There are several research prototypes of transactional file systems for UNIX systems, like the Valor file system, Amino, LFS, and a transactional ext3 file system on the TxOS kernel, and also transactional file systems for embedded systems, such as TFFS.

Without file system transactions, ensuring consistency across many file system operations is challenging, if not impossible. File locking can be used to manage concurrency for individual files, although it usually does not secure the directories or file metadata. File locking, for example, cannot stop TOCTTOU race situations on symbolic links. File locking cannot also immediately roll back an unsuccessful operation, like a software upgrade, because atomicity is required (Høiland-Jørgensen, 2018).

One strategy for introducing transaction-level integrity to file system structures is logging file systems. Journal transactions are not available to programs as elements of the OS API; instead, they are utilized internally to assure consistency at the resolution of a single system call. Data backup systems often do not offer direct backup of data stored in a transactional way, making it hard to retrieve accurate and reliable data sets. Irrespective of the transactional state distributed across numerous files in the entire dataset, conventional backup software merely records what files have modified since a particular time. As a solution, some database systems easily produce an archived state file holding all data up to that moment, and the backup software solely backs that up, not interacting remotely with the active transactional databases. After the backup software has recovered the file, the database must be recreated separately from the state file (Figure 9.4).

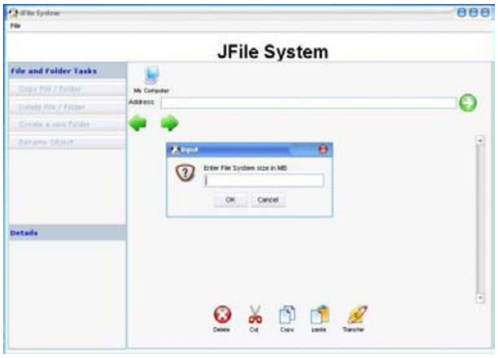


**Figure 9.4.** The file system enables you to view a file in the current directory as files are often managed in a hierarchy.

*Source: https://sourceforge.net/projects/javafilesystem/.*

### 9.3.6. Network File Systems

A network file system allows you to access files on a server. Programs on remote network-connected machines can use local interfaces to generate, manage, and retrieve hierarchical files and directories. Network file systems include file-system-like clients for FTP and WebDAV, as well as AFS, SMB protocols, and NFS.

### 9.3.7. File Systems on Shared Disks

A shared-disk file system lets several machines to access the very same external disk subsystem; however, when multiple computers access the same external disk subsystem, conflicts may arise; therefore, to avoid collisions, the file system determines which subsystem to access (DiLuoffo et al., 2018).

### 9.3.8. Minimal File System

Disk and digital tape technologies were prohibitively expensive for some early microcomputer users in the 1970s. A few low-cost basic data storage devices based on conventional audio cassette tape were created. When the system required data to be written, the user was notified to push “RECORD” on the cassette recorder. To alert the system, press the “RETURN” key on the keyboard. In addition, when the system wanted to read data from the cassette recorder, the user had to hit the “PLAY” button.

### 9.3.9. Flat File Systems

The flat system does not support subdirectories. It has only one directory, and all files are stored in that directory. Because of the limited volume of data space accessible, this form of file system sufficed when floppy disk media became finally available.

There are no subdirectories in a flat file system; directory entries for all files are stored in a single directory. Due to the limited amount of data capacity available on floppy disk medium at the time, this form of file system was suitable. CP/M machines had a flat file system in which files could be allocated to one of 16 user regions and generic file operations could be limited to work on one rather than all of them. These user areas were just special properties connected with the files; that is, no explicit quotas were required for each of these areas, and files may be introduced to groups as soon as there was still spare storage space on the disk. The Macintosh File

System, which was included with early Apple Macintosh computers, was similarly a flat file system. The file management tool (Macintosh Finder) provided the illusion of a somewhat hierarchical filing system on top of EMFS, which was uncommon. This layout required that every file, even if it looked to be in a separate folder, have a distinct name. IBM DOS/360 and OS/360 keep entries for all files on a disk pack (volume) in a directory called the volume table of contents (VTOC).

While simple flat file systems are easy to use, they become cumbersome when the number of files increases, making it hard to structure data into affiliated groups of files (Dixon et al., 2012).

Amazon's S3, a remote storage service that is purposely simple to let users to modify how their data is stored, is a new addition to the flat file system class. The only variables are buckets (picture an infinitely large hard drive) and objects (similar, but not identical to the standard concept of a file). The flexibility to use practically any character (even '/') in the object's name allows for advanced file management, as does the ability to choose portions of the bucket's content based on the same prefixes.

## 9.4. STRUCTURES OF DIRECTORY IN OPERATING SYSTEM (OS)

A directory is a framework that holds folders and files. It uses a hierarchical structure to organize files and directories (Figure 9.5).

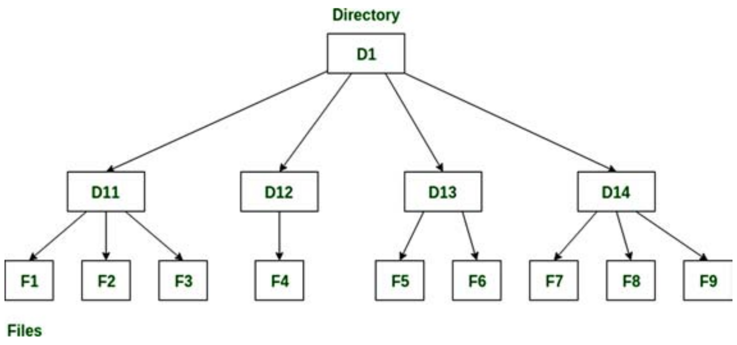


**Figure 9.5.** Structures of directory in operating system.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

A directory has multiple logical structures, which are listed below:

- **Single-Level Directory:** The single-level directory structure is the most basic. All files are stored in the same location, making it simple to support and comprehend.

When the quantity of files grows or the system has more than one user, a single level directory has a severe constraint. Because all of the files are all in the same directory, each one must have a distinct name. If two users refer to their dataset as test, the distinctive name rule has been broken (Figure 9.6).

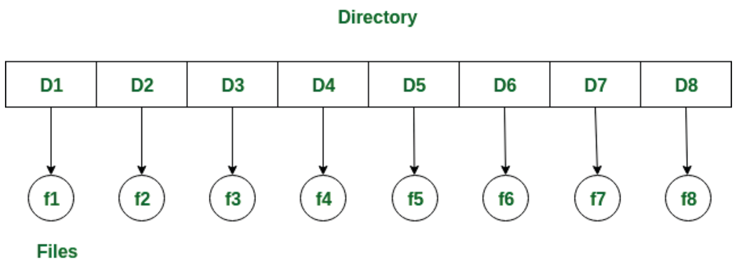


**Figure 9.6.** Single-level directory.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- Because it is a single directory, it is relatively simple to implement;
- Searching will be faster if the files are less in size (Blackham et al., 2011);
- In such a directory structure, actions such as file creation, searching, deletion, and updating are relatively simple.
- **Disadvantages:**
- There is a possibility of name conflict because two files can have the same name;
- Searching will take longer if the directory is vast;
- This cannot group files of the same type together.

1. **Two-Level Directory:** As we have seen, a single level directory frequently leads to file name confusion among different users. This problem can be solved by creating a different directory for each user.

Each user all have their own user files directory in the two-level directory structure (UFD). The structures of the UFDs are comparable, but each only displays the files of a single user. When a different user ID is logged in, the system's MFD is searched. The MFD is categorized by username or account number, and each entry leads to the user's UFD (Figure 9.7).

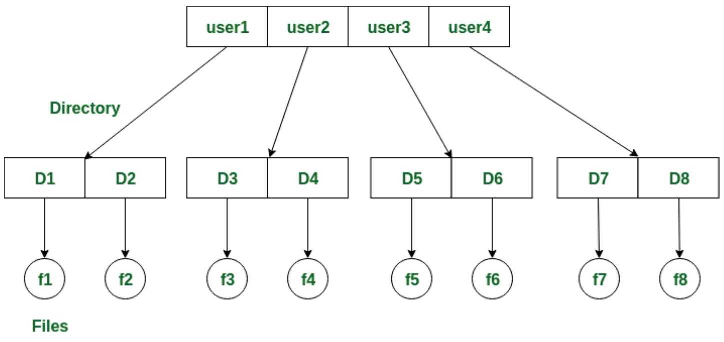


**Figure 9.7.** Two-level directory.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- We can provide the whole path, such as /User-name/directory-name;
- Multiple users can share the same directory and file name (Høiland-Jørgensen, 2018);
- Path-name and user-grouping make it simpler to search for files.
- **Disadvantages:**
- A user cannot exchange files with other users; nevertheless, it is not particularly scalable, as two files of the same type cannot be pooled together in the same user.

2. **Tree-Structured Directory:** Having seen a two-level directory as a tree of height 2, the simple generalization is to stretch the directory structure to any height tree.

This generalization enables the user to construct their own subdirectories and organize their files as desired (Figure 9.8).

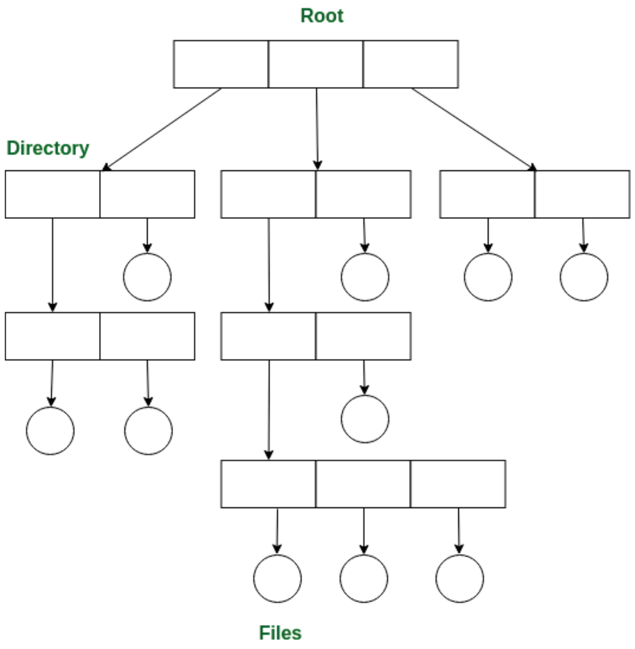


**Figure 9.8.** Tree-structured directory.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

The most frequent directory structure is a tree structure. There is a root directory in the tree, and each file in the system has a distinct path.

- **Advantages:**
- Very general, because the complete pathname can be specified;
- Because it is very scalable, the likelihood of name collision is low;
- Searching becomes much easier because we can utilize both absolute and relative paths.
- **Disadvantages:**
- Because not every file fits into the hierarchical model, files may well be saved in many directories;
- We are unable to share files;
- It is inefficient since obtaining a file may require browsing across numerous folders.

1. **Acyclic Graph Directory:** An acyclic graph is a non-cyclic graph that enables us to exchange subdirectories and files. The same file or subdirectory may exist in two distinct directories. It is a logical extension of the tree-structured directory.

It is used when two programmers are working on a collaborative project and need to access files. The related files are placed in a subfolder, which separates them from other programmers' projects and files because they are collaborating on a shared project and want the subdirectories to be in their own directories) (Hellmund, 2016). The subdirectories that are shared should be shared. As a result, we employ Acyclic directories in this case. It is important to understand that the shared file and the copy file are not the same thing. Any modifications made by a programmer in the subfolder will be reflected in both subdirectories (Figure 9.9).

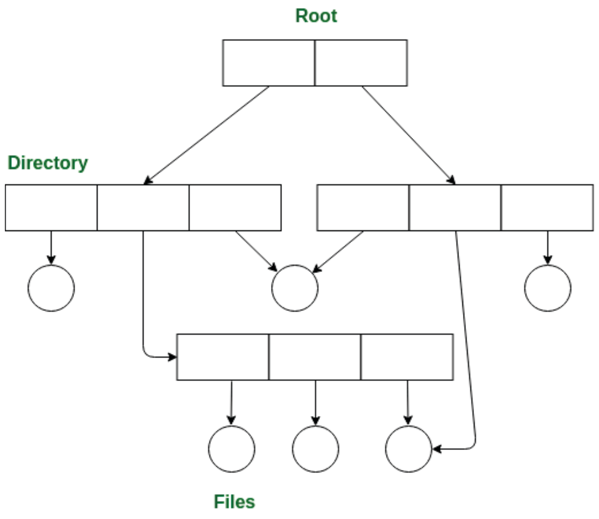


**Figure 9.9.** Acyclic graph directory.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- We can exchange files;
- Because of the numerous paths, searching is simple.
- **Disadvantages:**
- We share files by linking them in case deleting them causes an issue;
- If the link is a soft link, we are left with a hanging reference after deleting the file;
- In the case of a hard link, deleting a file necessitates deleting all references to it.

2. **General Graph Directory Structure:** Cycles are permitted within a directory structure in which multiple directories can be

generated from more than one root directory in the general graph directory structure. The biggest issue with this type of directory layout is calculating the total amount or space occupied by the files and directories (Figure 9.10).

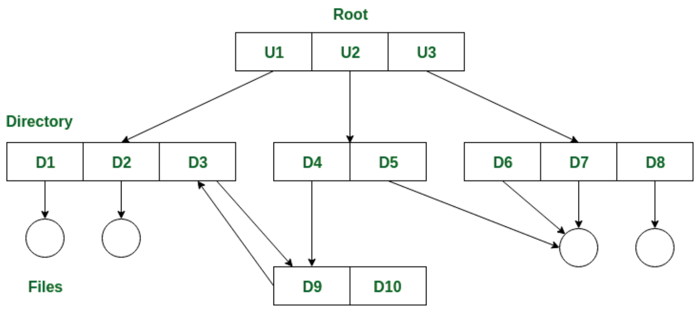


**Figure 9.10.** General graph directory structure.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- It permits cycles;
- It is more adaptive than other directories structure.
- **Disadvantages:**
- It is more expensive;
- It needs recycling bins.

## 9.5. FILE ACCESS MECHANISMS

The technique by which the data in a file can be retrieved is referred to as the file access mechanism. There are various methods for gaining access to files.

### 9.5.1. Sequential Access Method

It is the most basic mode of access. The data in the file is handled sequentially, one record after the other. This is the most popular way of access; for instance, editors and compilers typically access the file in this manner (Irwansyah et al., 2018).

The majority of file operations are read and write. A read action -read next- reads the file's next place and advances a file pointer, which takes account of I/O location. Likewise, for the -write next- command, add to the end of the file and move forward to the newly written data.

- **Key Points:**
- In an order, data is accessed one record after another;
- If we use the read command, it advances the pointer by one;
- If we use the write command, memory is allocated and the pointer is moved to the end of the file;
- This procedure is appropriate for tape.

### 9.5.2. Direct or Random Access Methods

Another method is direct access method also known as relative access method. A filed-length logical record that allows the program to read and write record rapidly in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block (Jaeger, 2008). For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59, and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

A block number given by the user to the OS is often a relative block number, with 0 being the first relative block in the file, followed by 1, and so on. In other terms:

- Random access file organization allows for direct access to records;
- Each entry has its own address on the file, which can be used to access it directly for reading or writing;
- The records do not have to be in any particular order within the file or at multiple places on the storage medium.

### 9.5.3. Index Access Method

An indexed file is a computer file that has an index that enables quick random access to every record providing its file key. The key is a feature that distinguishes a record in a unique way. If many indexes are present, the others are referred to as alternate indexes. The indexes are created with the file but preserved by the system.

### *9.5.3.1. Index Sequential Access Method*

The direct access method has been modified by the index sequential access method. Essentially, it is a hybrid of both sequential and direct access. The basic idea behind this method is that it first directly accesses the file and then accesses it sequentially (Jeong et al., 2012). It is required to keep an index in this access mechanism. The index is simply a pointer to a block. The index is used directly to access an item in a file. The information collected from this access is utilized to get access to the file. The indices might be quite large at times. To manage all of these index hierarchies, one direct index access connects to information from another index access. The primary advantage of this form of access is that it allows for both direct and sequential file access.

Important points:

- This technique is based on sequential access;
- For each file, an index is produced that contains pointers to various blocks;
- The index is examined consecutively, and its pointer is used to directly access the file.

A sequential access is one in which the records are accessed in a specific order, i.e., the information in the file is processed one record at a time. This is the most basic mode of access. Compilers, for example, typically access files in this manner.

### *9.5.3.2. Basic Partitioned Access Method (BPAM)*

The basic partitioned access method (BPAM) organizes entries on DASD as members of a partitioned data set (PDS) or a partitioned data set extended (PDSE). BPAM may be used to see a UNIX directory and its files as if they were PDS files. With BSAM or QSAM, you can see each PDS, PDSE, or UNIX member sequentially (Kushwaha and Kushwaha, 2011). A PDS or PDSE has a directory that maps member names to data set locations. Individual members can be found in the PDS, PDSE, or UNIX directories. The directory for program libraries (register modules and program objects) provides program properties required to load and rebind the member. Despite the fact that UNIX files can include program objects, program management does not reach UNIX files via BPAM.

## 9.6. SPACE ALLOCATION

The OS allots disk space to files. OSs use one of three methods to allocate disk space to files.

- Contiguous allocation;
- Linked allocation;
- Indexed allocation.

### 9.6.1. Contiguous Allocation

Each file in this approach takes up a contiguous set of blocks on the disk. For instance, if a file needs n blocks and is given an initial location of block b, the blocks allocated to the file will be: b, b+1, b+2, …, b+n–1 (Krohn and Tromer, 2009). This implies that we can calculate the blocks filled by the file from the originating block address and the length of the file (in units of blocks required).

The directory entry for a file with contiguous allocation includes the following information (Figure 9.11):

- Starting block address;
- The length of the allotted section.

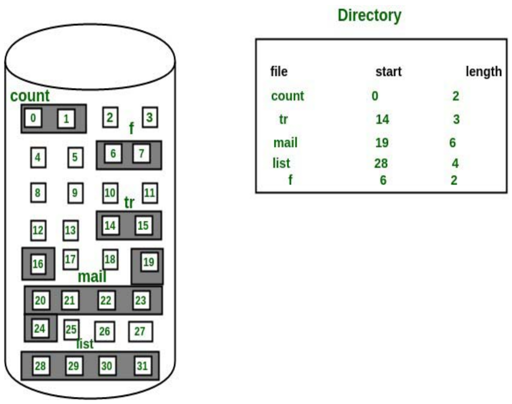


**Figure 9.11.** Contiguous allocation.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- The address of the kth block of the file, which begins at block b, can be easily determined for direct access as (b+k);

- This is exceptionally quick because the number of seeks is kept to a minimum due to the contiguous allocation of file blocks.
- **Disadvantages:**
- As a result, it is inefficient in terms of memory consumption;
- It is difficult to increase file size since it is dependent on the availability of contiguous memory at a given time.

So, to summarize:

- On disk, each file takes up a contiguous address space;
- Assigned disk address is in linear order;
- Easy to implement;
- External fragmentation is a key issue with this kind of allocation method.

### 9.6.2. Linked Allocation

Each file in this approach is a connected array of disk blocks that do not have to be contiguous. Disk blocks can be found wherever on the disk (Blackham et al., 2011).

The directory entry includes a pointer to the beginning and end of the file block. Each block provides a link to the next block that the file will occupy.

The file 'jeep' in the image below shows how the blocks are spread at random. The final block (25) includes the value–1, indicating a null pointer that does not reference to any other block (Figure 9.12).

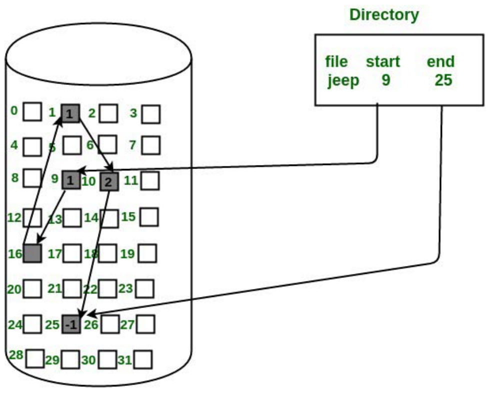


**Figure 9.12.** Linked allocation.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- In regards to file size, this is quite adaptable. Because the system does not have to hunt for a contiguous portion of memory, file size can be readily increased;
- This approach is free of external fragmentation. As a result, it performs relatively well in regards to memory use.
- **Disadvantages:**
- Because file blocks are placed arbitrarily on the disk, it takes a huge number of seeks to retrieve each block independently. This slows down linked allocation;
- It does not allow for either random or direct access. We cannot directly access a file's blocks. A file's block k can be retrieved by traveling k blocks sequentially (sequential access) from the file's starting block using block pointers) (Lange et al., 2011).

The use of pointers in the linked allocation incurs some additional overhead:

- Each file contains a list of disk block links;
- The directory includes a link or a pointer to the first block of a file.

There is no external fragmentation:

- Used well in a sequential access file;
- Inefficient when using a direct access file.

## 9.7. ALLOCATION BASED ON INDEXES

A unique block defined as the Index block in this architecture holds references to all the blocks consumed by a file. Each file has a unique index block. The disk address of the ith file block is contained in the index block's ith entry. As illustrated in Figure 9.13, the directory entry includes the index block's address.

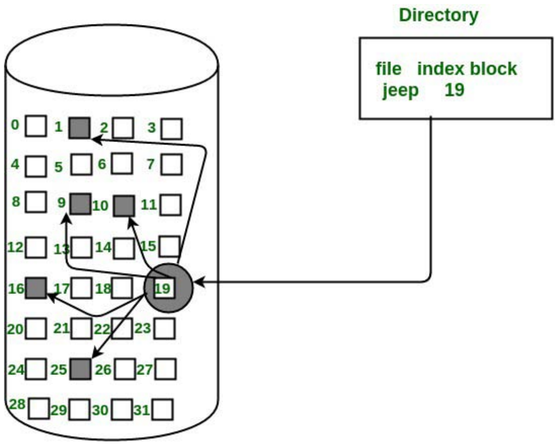


**Figure 9.13.** Index blocks.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

- **Advantages:**
- It allows for direct access to the file's occupied blocks, allowing for quick access to the file's blocks.
- It solves the issue of external fragmentation.
- **Disadvantages:**
- Indexed allocation has a higher pointer overhead than linked allocation.
- In the case of very small files, like those that extend only 2–3 blocks, the indexed allocation would retain one complete block (index block) for the pointers, which is limited in terms of memory consumption. Nevertheless, in linked allocation, we lose only one pointer per block (Bala et al., 2015).
- For particularly big files, a single index block may not be able to retain all of the pointers.

To resolve this, the following mechanisms can be used:

- **Schemes That are Linked:** This approach connects two or more index blocks to retain the pointers. Each index block will also have a pointer or address to the next index block.
- **Multilevel Index:** In this strategy, a first level index block is utilized to point to second level index blocks, which refer to the

disk blocks filled by the file. Based on the peak file size, this can be expanded to three or more levels.

- **Combined Scheme:** In this system, a special block known as the Inode (information node) includes all of the file's information like the name, size, authority, and so on, and the residual space of the Inode is used to store the Disk Block addresses that contain the actual file, as shown in the image below (Lass and Gronau, 2020). The first several pointers in Inode point to direct blocks, i.e., the pointers hold the addresses of the disk blocks containing the file's data. The following points go to indirect blocks. There are three types of indirect blocks: single indirect, double indirect, and triple indirect. A single indirect block is a disk block which does not carry file data but does include the disk addresses of the blocks that do.

Similarly, double indirect blocks need not include file data but rather the disk address of the blocks holding the file data (Figure 9.14).

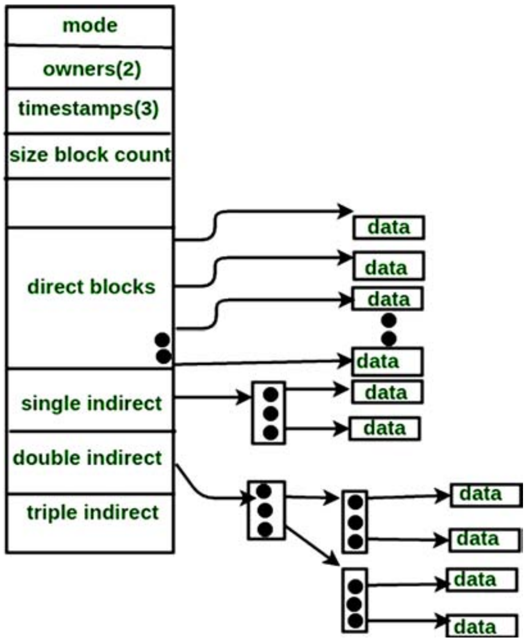


**Figure 9.14.** Index blocks.

*Source: https://www.geeksforgeeks.org/file-systems-in-operating-system/.*

In general, indexed allocation:

- Provides answers to contiguous and linked allocation difficulties;
- An index block is built with all file pointers;
- Each file has its own index block, that stores the file's disk space addresses;
- The directory holds the addresses of file index blocks.

Within a single system, multiple file systems can exist. Retail systems are often designed with a single file system that takes up the entire storage drive. Another option is to split the disk and use multiple file systems with varying properties. One file system may be established with a minimal allocation size for usage as a browser cache or email storage. This maintains the regular browser behavior of generating and deleting files (Levis et al., 2005).

This maintains the regular browser activity of generating and deleting files in a restricted area of the disk where it wouldn't interfere with other file allocations. Another partition could be formed to store music or video files with reasonably high block sizes. Another may generally be set to read-only and only be made writable on a periodic basis.

A third option, which is typically used in cloud systems, is to employ "disk images" to store other file systems, with or without the same properties, as files within another file system. Virtualization is a frequent example: one person can operate an experimental Linux distribution in a virtual machine within his or her production Windows environment (with NTFS).

The ext4 file system is included within a disk image, which is viewed as a file (or multiple files, based on the hypervisor and settings) in the NTFS host file system.

Having many file systems on just one system has the added benefit of ensuring that if a single partition becomes corrupted, the other file systems are generally still intact. This includes virus-induced system partition damage or a system that would not boot. File system utilities that need dedicated access can be executed effectively piecemeal (Lin and Ye, 2009). Furthermore, defragmentation may be more effective. Virus scans and backups are two examples of system maintenance utilities that can be run in segments. For example, if no new files have been recorded since the last backup, it is not essential to backup the file system holding videos together with all other files. In terms of image files, one may easily "spin off" differential images that include solely "new" data added to the master image. Differential images could be used for both safety issues (as a "disposable" system – can be quickly revived if destroyed or tainted by a virus, as the old image can be deleted and a new image can be generated in a matter of seconds, with or without automated procedures and quick virtual machine deployment.

## 9.8. DESIGN LIMITATIONS

Every file system has a functional limit that defines the maximum amount of data that can be stored within that system. These functional limits are the designer's best estimation based on how huge the storage systems are now and how large they are projected to become in the future. Disk storage has continued to grow at near exponential rates (see Moore's law), therefore after a few years, file systems have reached design limits that force computer users to upgrade to a newer system with ever-greater capacity (Figure 9.15).

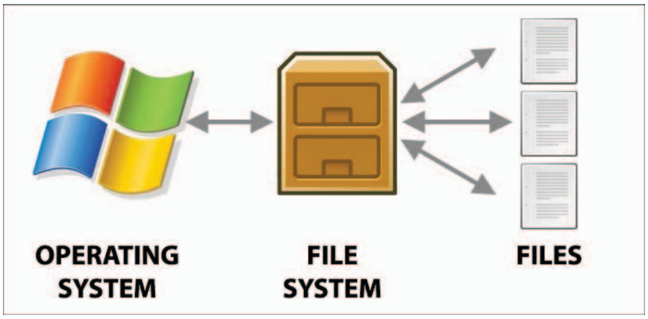


**Figure 9.15.** File systems and operating systems.

*Source: https://www.sitesbay.com/os/os-file-system-in-operating-system-os.*

The complexity of a file system is often proportionate to the available store capacity. Early 1980s home computer file systems with 50 KB to 512 KB of storage would be unsuitable for current storage systems with hundreds of gigabytes (GB) of memory. Similarly, current file systems may not be a good fit for these early systems since the intricacy of modern file system architecture would soon deplete or even surpass the early storage systems' very low capabilities.

### 9.8.1. File Systems and Operating Systems (OSs)

#### *9.8.1.1. Limitations*

- **Converting the Type of a File System:** It may be advantageous or necessary to have files in a different file system than they currently exist. Reasons include the need for an increase in the space requirements beyond the limits of the current file system. The depth of path may need to be increased beyond the restrictions of the file system. There may be performance or reliability considerations. Providing access to another OS which does not support the existing file system is another reason.
- **In-Place Conversion/Migrating to a Different File System:** It may be desirable or required to store files in a separate file system than the one in which they are now stored. The need for increased space requirements beyond the constraints of the current file system is one of the reasons. The depth of the route may need to be raised beyond the file system's limitations. There may be efficiency or reliability issues to consider. Another motive is to provide access to another OS that does not match the existing file system (Aksoy et al., 2017).
- **Long File Paths and File Names are Both Acceptable:** Files in hierarchical file systems are accessible via a path, which is a branching list of folders holding the file. The depth of the path is limited differently in different file systems. Individual filenames are likewise limited in length in file systems.

Copying files with large names or those placed in deep paths from one file system to another can have unfavorable consequences. This is determined by how the tool performing the copy resolves the disparity.

CHAPTER 10

# I/O SOFTWARE AND I/O HARDWARE

## CONTENTS

## 10.1. I/O HARDWARE

An OS is designed to take care of the most important jobs as well as manage some of the input/output (I/O) devices. Some of these input output devices include various hardware of a computer such as the keyboard and mouse attached pad display adapter USB device and a disk drive. An OS is also known to have the analog to digital converter and other important parts such as the printers and stable network connections. An input output system is designed to take the request of any application to add that includes or input data, then it sends it to a physical device. The input output system then relays the information obtained from the physical device and relays the feedback to the application. Basically, the input output system hardware acts as a medium between the application and an external physical device. This hardware device can be divided into two categories and that is the block devices and character devices (Figure 10.1).

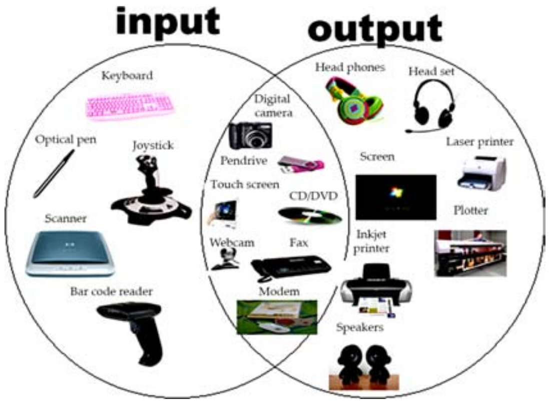


**Figure 10.1.** Output and input devices.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_Intro.png.*

A block device can be defined as the part that is responsible for communication by sending the designated blocks of data. In other times the block device is also known as a driver communicator. Some examples of the block devices include a hard disk, USB cameras, as well as disks on key devices. This device size usually ranges from 512 to 4,096 bytes. On the other hand, a character device can be defined as the part that drives communication by using and sending as well as receiving single characters. By single characters this means bytes and octets (Muehlstein et al., 2017). Examples of these devices include zero pots, parallel pots, and sound cards.

The character device does not pay any attention to any block structure. This device is also not addressable and is not mandated to seek operation. Other examples of this device generally include devices that do not take the form of a disk but instead I reviewed as character devices. Another category of the input output devices includes that 'doesn't really fit' category. In this case this means that these devices do not fall in either block devices or character devices. A good example is the clocks which cannot be blocked and they also do not accept any character streams. Another example of these devices includes the memory map screens.

## 10.2. DEVICE CONTROLLERS

This is the electronic component of the input-output units. It is also known as an adapter because the OSs use these adapters to handle all input output hardware devices. Each device contains a device controller and a device driver for the purpose of communicating with the overall OS. The device controller is capable of handling several devices at the same time. It is considered to be an interface and it is mainly tasked with processing serial bit streams and converting them to block all bytes as well as ensuring that errors that might arise are corrected as required. Examples of device controllers include the cathode ray tube controller and LCD controller.

The cathode ray tube controller refers to the oldest versions of existing monitors which were very bulky, consumed a lot of power and were very fragile. This controller is tasked with firing a beam of electrons onto a fluorescent screen creating the effect that we see in our private computers. This system also uses a magnetic field which allows it to bend this beam of electrons and the end result is a drawing of pixels that are seen on the screen. The first laptop to be under this controller weighed about 12 kg which is quite heavy as compared to the newest forms of computers (Manzalini and Crespi, 2016).

The LCD controller is considered to be a bit of a serial device but it belongs to the lower level. This device is tasked with reading the bytes that have the characters which are meant to be displayed from the original memory. The signals generated from this process I meant to modify the polarization of the backlight that belong to other corresponding pixels in an attempt to write them on the screen. Devices that have the LCD controller do not require an OS programmer to use their expertise in programming the electrical field that is found on the screen.

## 10.3. THE MEMORY MAPPED INPUT OUTPUT SYSTEM

The controller contains registers and the operational system is able to ride these registers in a way of suggesting giving order to the devices or reading the state of the device. These registers are similar to the central processing unit registers but the registers in these devices are specifically designed for this device. Some of the orders written by these registers include shutting down of the computer as well as accepting data by the computer. These are some of the orders that the registers convey to the device and the device obeys these registers by executing what the register states. There are two ways that the central processing unit can interact with the control registers and the device data buffers. It can interact with the two devices through either dedicated pod which are allocated for this purpose or using the device memory to map all the control registers as well as device data buffers. The central processing unit is able to communicate to the control registers and the device that is above us in three ways and they include:

- The separate input output and memory space. In this case every control register is usually assigned an input and output port number that is meant to be a differentiation factor for every controlled register. The special input output instructions include IN REG, PORT, and OUT PORT REG and IN and MOV. These instructions are meant to be different and unique to every control register.
- The memory mapped input output hardware system. In this case the memory and input output devices share the same address space (McClean et al., 2013). In order for their input-output devices to transfer blocks of data to and from the memory there are certain main memory locations that are connected to the devices so as to ensure that the data does not go through the central processing unit.
- The last way is the hybrid way. The pods for control registers are memory-mapped above us as well as separate input-output ports. This way is also known as the Pentium way (Figure 10.2).

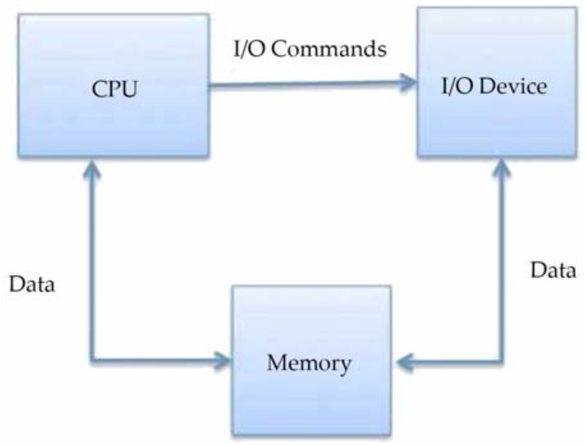


**Figure 10.2.** The memory mapped I/O systems.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_Memory-Mapped.png.*

The strength or advantages of the memory mapped input–output systems:

- The OS usually results in using assembly codes in cases where special input output instructions are fed to the computer. The special instructions include in and out and they cannot be executed using programming languages such as C or C++.
- The memory mapped input output system hardware usually allows the programming language C to have a direct connection with a memory. The programming language C simply writes to the memory without using other mediums.
- The control registers are usually mapped to the overall memory.

Weaknesses or disadvantages of using memory mapped input–output hardware systems:

- The use of memory-caching in devices that use the control registers is usually very disastrous. In this case it is very difficult to detect when a device has changed its state. To fix such a problem requires that several selected caching disabled (Monaco et al., 2013).
- Using the memory mapped input output hardware device system required that all memory modules as well as devices examine the differences of each memory to see if it best suits what

they represent. These days the memory bus operates at a very high speed. This becomes a problem because the I/O hardware devices won't be able to see the allocated memory addresses on the Memory bus. One way to fix this issue is by sending all the requests to the main memory of the device as a test to see if these requests will fail. The requests are then sent to the I/O device. Another way is snooping on the requests from the memory and then relaying the important ones to the Input-Output controls. The problem with this specific method is that the snooping process is usually very slow (Figure 10.3).

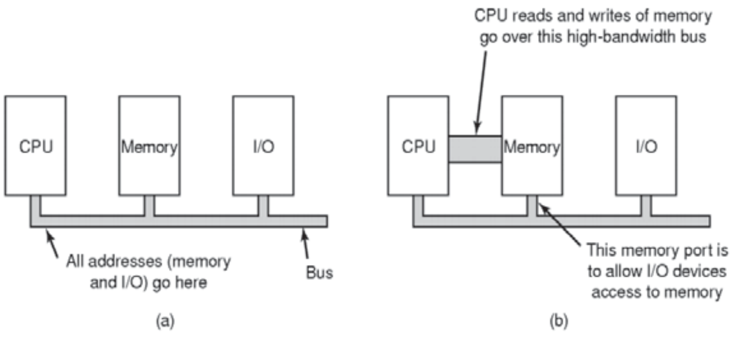

CPU reads and writes of memory go over this high-bandwidth bus
CPU
Memory
I/O
CPU
Memory
I/O
All addresses (memory and I/O) go here
Bus
This memory port is to allow I/O devices access to memory
(a)
(b)

**Figure 10.3.** Input output hardware.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_BusArch.png.*

## 10.4. DIRECT MEMORY ACCESS

Direct memory access has a task of transferring data between the input output device and the memory on a private computer. It does this by bypassing the central processing unit and its aim is to reduce the overhead of interruptions (Moore and Stouch, 2016). Direct memory access he's also in charge of controlling the exchange of data between the input output device and a memory that is considered to be the main memory of a private computer. During this process the central processing unit is only involved in the process at the beginning and towards the end. Interruption is only possible after the entire block has been completely transferred (Figure 10.4).

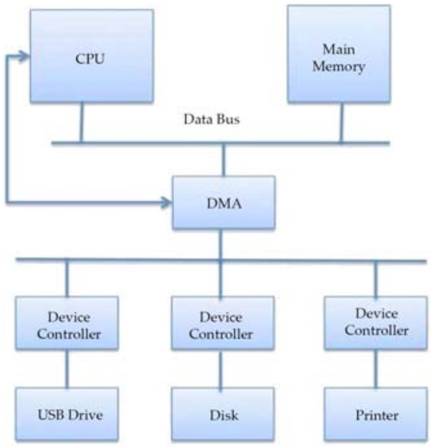


**Figure 10.4.** Direct memory access.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_DMA.png.*

## 10.5. DIRECT MEMORY ACCESS CONTROLLER

This principle is in charge of transferring data and it also gets to arbitrate between the system buses. The direct memory access controller is written as android by the central processing unit because of several registers. Some of these registers include the memory address register, byte count register as well as more than one control registers.

### 10.5.1. How Direct Memory Access Works?

- The first step is for the central processing unit to program the day or tomorrow access controller and this is achieved by setting registers in a way that makes it easy for the controller to know what it's meant to transfer and where it is supposed to transfer.
- The Direct memory access controller usually transfers commands to certain parts of the OS such as the disk controller. The command is meant to instruct the disk controller to internalize data that comes from other disks. Internalization is meant to occur in the internal buffer as a way to verify the checksum. The direct memory access can only begin when valid data is entered into their disk controller buffers.

- The transfer is initiated by the direct memory access controller issuing an additional request over the control buses to the disk controller (Ahmad et al., 2013).
- The memory written is considered to be another type of bus cycle.
- The bus acts as a median between the controller and the direct memory access controller. When the writing process is complete their direct memory access controller usually receives a signal of acknowledgment from the disk controller.
- The controller usually increases the memory addresses that are to be used in the byte count. In a hypothetical situation where the byte count turns out to be greater than value zero there is the repetition of steps two, three, and four until a count of value zero is reached. During the repetition process the controller usually interrupts the program that is run by the central processing unit and informs the CPU that the transfer has been completed (Figure 10.5).

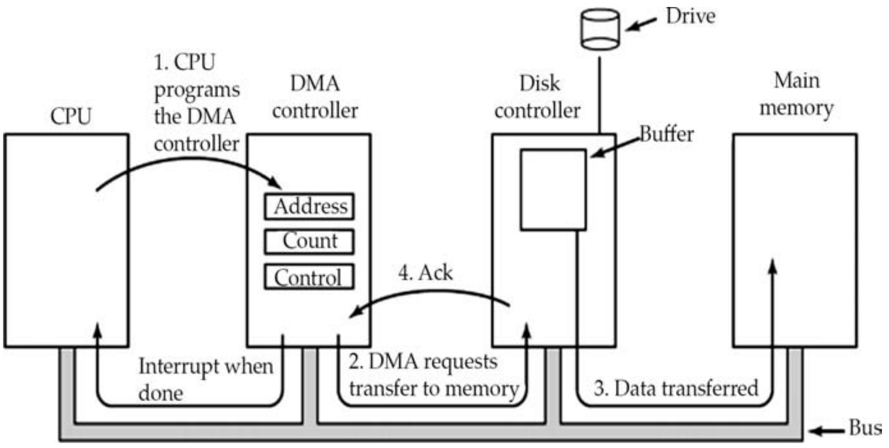


**Figure 10.5.** Operation of a DMA transfer.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_DMA-Transfer.png.*

The controllers are different based on their sophistication. The controllers can either be simple or sophisticated. The simple controllers are in charge of handling one transfer at a time while they're sophisticated controllers are made up of multiple sets belonging to registers that are found internally. The transferring of various words can use the round-robin algorithm which is designed to enable transfer. Another way for a transfer to become successful is if their OS has a priority skin design which is meant to favor some input-output devices over other devices (Nollet et al., 2004). There are only two mods that can operate on many bases and include one word at a time mode as well as a block mode. There are some controllers which are capable of operating in either of the two modes. In the word at a time would the controller ask to transfer just one word and the request is approved.

This means that in cases where the central processing unit is in need of the bus it has to wait. This equal telling mechanism can be defined as controllers trying to sneak in and steal some of their bicycles from the central processing unit. This is only done once in a while and the result is that it results in slight delays. On the other hand, the block mod allows the controller to inform the input output devices that they need to acquire the buses, ensure that a series of transfers occurs, and only then can it release the buses. The best mood which is considered to be a form of operation in the block mode is more efficient than the cycles stealing. The only disadvantage of using the burst mode is that it is capable of blocking the central processing unit as well as other devices for a considerable amount of time if there is a long burst that is meant to be transferred. The advantage is that the process of acquiring buses is cut short because the mode is capable of transferring several words by acquiring only one bus and not many as compared to the cycle stealing.

## 10.6. INTERRUPTS REVISITED

After the input-output device has completely finished its walk, it usually causes an interruption by sending an attaching signal to the bus line indicating that it has already been assigned. The interrupt controller chip is in charge of detecting the signals. If there are no other interests that are pending, the interrupt controller ensures that the interrupt process is completed almost immediately (Ow, 2011). If there is an interrupt in progress or there is a request that is considered to be simultaneous, there is an interrupt signal that is released continuously until the Central Process Unit notices and acts on servicing the interrupt. The only way for the controller to interrupt the central processing unit is by putting numbers on the address lines and coming up with signals that are meant to interrupt the CPU. The number is usually used in the interrupt with that as an index and its main task is to start a corresponding interrupt service procedure (Figure 10.6).

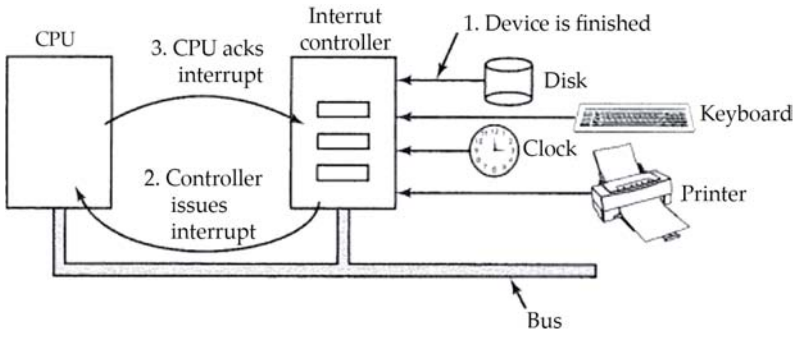


**Figure 10.6.** The interrupt revisited system.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOHw_Interrupt.png.*

## 10.7. PRECISE AND IMPRECISE INTERRUPTS

A precise interrupt is defined as an interrupt that leaves a machine in a well-defined state. The process interrupt has four properties and they include:

- The program counter which is usually saved in a place that is known;
- Completion of all instructions before the one pointed out by the PC;
- Once the pointed instruction has been completed there are no other instructions beyond this point; and
- This date of execution by the instructions pointed to the PC is usually known.

In cases where an Internet does not have this property it is known as an imprecise interrupt. The imprecise interrupt is difficult and unpleasant for items to program the OS because they are tasked with finding out exactly what happened and what is meant to happen.

## 10.8. I/O SOFTWARE

The software is often organized in various layers and they include:

- **The User Level Libraries:** The main function of this layer is to provide an interface that allows programs to perform input and output functions. A good example is the studio which is a library that is brought about by the existence of the C and the C$^{++}$ programming languages.

- **The Kernel Level Modules:** This layer usually provides a device driver that is meant to interact with other device controllers and independent modules which are meant to be used by the device drivers (Peter et al., 2015).
- **Hardware:** In this layer they had a controller and the actual hardware usually interacted with several device drivers as a way of ensuring that the hardware stays alive.

An important factor to keep in mind when designing the input output software is that the software has to be device independent which means that it can write programs that can be able to access any input output device without having to specifically list that device in its system in advance.

## 10.8.1. Principles of I/O Software

That input output software has goals that they are meant to achieve and they include:

- **The Device Independence:** It is a key concept that is contained in the input of the software design. This concept allows their input output devices accessible to various programs without having to specify the device that is supposed to be done in the accessibility process in advance.
- **Another Goal is the Uniform Naming:** The name is meant to be a thing or an integer that is not dependent on any device. This is common in the UNIX OS.
- **Error Handling:** Institutions where the controller comes across a read error is supposed to ensure that it corrects the error by itself if it is possible. In other situations where their problem is to integrate the device driver should be used to handle the situation and this is done by trying to read the block again. There are many situations where the correction of errors is done transparently in lieu of labels and attention is that the high levels are not aware of that error (Santos et al., 2013).
- **Synchronous and Asynchronous Transfers:** The synchronous transfer is also known as blocking and then a synchronous transfer is known as the Interrupt driven transfer. Most of the physical input of what software devices are considered to be

interrupt driven however, this situation leads to the demand of high-performance applications that are supposed to control the details of their input output software devices.

- **Buffering:** This process is important because they usually come up with a device that is not capable of being stored directly in their set final destination.
- **Shareable and Dedicated Devices:** In this case several Input out of your devices are capable of being used by many users at the same time and an example is the disks. There is no problem that is encountered by multiple users having to open data files that belong to the same disks at the same time. On the other hand, devices such as printers can only be used by a single user until they have finished their tasks. It is only after one user has finished with the printer does another person get to use the printer. The disadvantage of one user using a single device at a time, is the existence of deadlock (Figure 10.7).

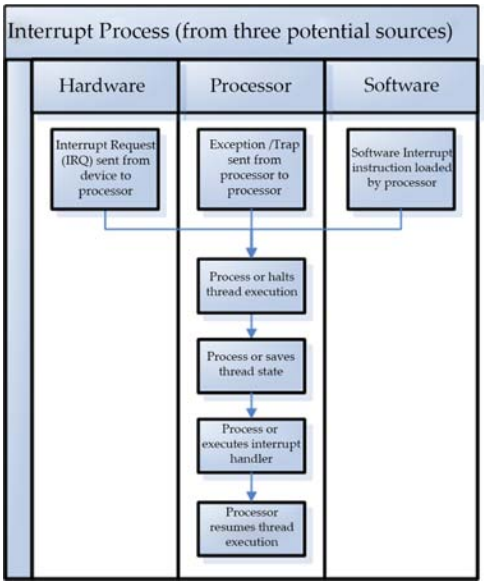

Interrupt Process (from three potential sources)
Hardware
Processor
Software
Interrupt Request (IRQ) sent from device to processor
Exception /Trap sent from processor to processor
Software Interrupt instruction loaded by processor
Process or halts thread execution
Process or saves thread state
Process or executes interrupt handler
Processor resumes thread execution

**Figure 10.7.** Goals of I/O software.

*Source: https://upload.wikimedia.org/wikipedia/commons/c/cf/Interrupt_Process.PNG.*

## 10.9. PROGRAMMED I/O

This is considered to be one of the three principles of input output software. It is considered to be the simplest type of input output technique that is responsible for extending data as well as facilitating any type of communication that is done between the processes and other external devices. When it comes to programmed input output software, Exchange of data is made to be precisely between the processor and the I/O module. The program that is executed with the processor allows it to have total control over the I/O operation (Sjöstrand et al., 2015). This control allows it to sense the status of devices, write commands and interpret comments that they have received, and ensure that data is transferred. Once a comment has been issued to the input output model by the processor, the processor has to wait until the input output operation comes to a standstill. In cases where the processor is faster than the model, this leads to a waste of the processor time. The following is the summary of an operation of the programmed input output software device:

- The processor is performing its function of executing problems when it encounters instructions that are related to the input output operation;
- Once the processor encounters the instructions it executes his instructions by ensuring that it issues a command but only to the appropriate module;
- The model is in charge of performing the requested action based on the commands that are issued with its input output device, especially the processor;
- The last operation is for the processor to periodically check the status of that module and this is supposed to go on until the operation has been completed.

### 10.9.1. Programmed I/O Mode: Input Data Transfer

- During data transfer the input is read only after the devices have been tested to ascertain whether they are ready for the input.
- The program is usually tasked with waiting for the ready status from the input devices. The status is achieved by testing the status bit repeatedly and ensuring all bytes are read (Silva et al., 2006).

- The program is said to an unwitting state which indicates that it is busy but this is done only after the device is in the right state (Figure 10.8).

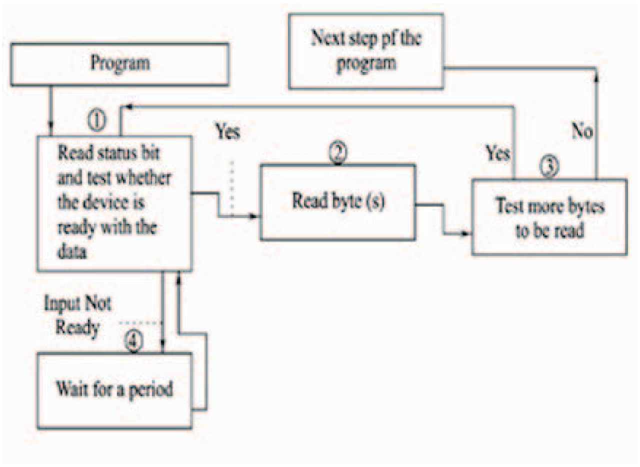

Program
Next step pf the program
①
Read status bit and test whether the device is ready with the data
Yes
②
Read byte (s)
Yes
③
Test more bytes to be read
No
Input Not Ready
④
Wait for a period

**Figure 10.8.** Input data transfer process.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOSw_Programmed_Input.png.*

### 10.9.2. Programmed I/O Mode: Output Data Transfer

- After the output has been written which is done after testing the device to a certain that the device is in a position to accept bytes.
- The program then waits for their release status by testing their status bits repeatedly.
- The program automatically goes to a busy State another time soon as a non-waiting state.

Using the programmed I/O has several advantages such as this system is simple to implement and requires almost none hardware support. The disadvantages of the programmed I/O Software is they have a long busy waiting period. And these periods are usually the central processing unit from contacting and the functions or other useful works (Figure 10.9).

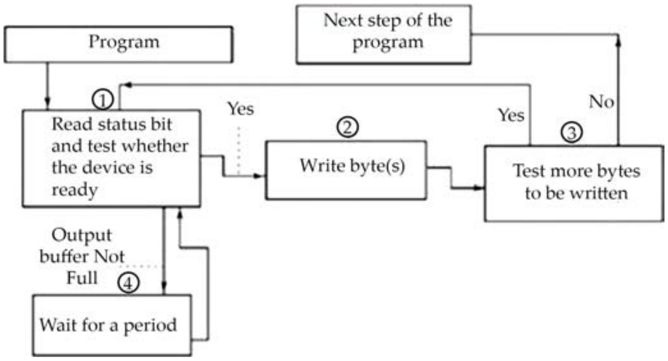


**Figure 10.9.** Output data transfer system.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOSw_Programmed_Output.png.*

## 10.9.3. Interrupt-Driven I/O

They interrupt driven input of the best software that is used to deal with I/O. It is a way of controlling I/O activities. It is in charge of ensuring that a tell Mina that is in need of making and receiving data has a capability of sending signals to initiate this process. This process allows a program interrupt to come to light. In this case the processor usually enters the interrupt service routine.

- **Inputs:** In this case the device is in charge of interrupting the central processing unit when there is a rival of fresh data which can be easily retrieved by the system processor (Shaw et al., 2016). The actions to be performed usually depend on certain conditions and one of them is whether the device uses I/O ports or Memory Mapping.
- **Outputs:** In this case the device usually delivers an interrupt either when it gets a chance to acquire a new set of data or when it is ready to acknowledge that the data transfer was successful. The interrupts are generated by Memory-Mapped and direct memory access devices. They usually inform the system once they are done with the buffer.

### *10.9.3.1. Operations Found in the Interrupt I/O*

- The central processing unit which is the backbone of the whole computer system is tasked with reading the commands that are generated and interpreting them.
- The I/O module usually receives a set of data from the peripheral. The Central Processing Unit is meant to continue doing other tasks.
- The module is in charge of interrupting the CPU once a signal has been received.
- The central processing unit is in charge of requesting fresh data for analysis.
- The I/O modules are tasked with ensuring that the transfer of data is made possible.

The interrupt driven I/O has several advantages but the most important include, the software is very fast as compared to the software that is found in the Programmed I/O devices. The disadvantages of the Interrupt-Driven I/O devices include, the commands are usually hard to write especially in cases where only low-level languages are accepted. The other disadvantage is that it becomes such a tedious task to get various pieces of the OS to perform the same function.

## 10.10. I/O USING DIRECT ACCESS MEMORY

As stated, direct memory access is considered to be a technique that is responsible for transferring data that is contained within the main memory of a computer to and from an external device without having to use the central processing unit. The dirt access memory is responsible for improving processor activity as well as the input output transfer rate. This is done by the technique taking over the functions of transferring data that is meant to be stored in the processor and allowing the processor to perform other tasks which are of importance (Sharma et al., 2012). This technique is better than the other two input output techniques because it is not time consuming when it comes to issuing comments that are meant to facilitate the data transfer. It is also safe and very efficient to use the direct access memory when dealing with large volumes of data that needs to be transferred. For this technique to be implemented, the processor is supposed to share its system bus with the direct memory access module. The main functions of the direct memory access include:

- Reading and writing commands;
- controlling the lines that are meant for communication purposes;
- Data lines that are also meant for communication purposes.

The only advantage of the direct memory access is that it performs in functions at a high speed which means that there are no waiting time which results in a shorter execution path (Figure 10.10).

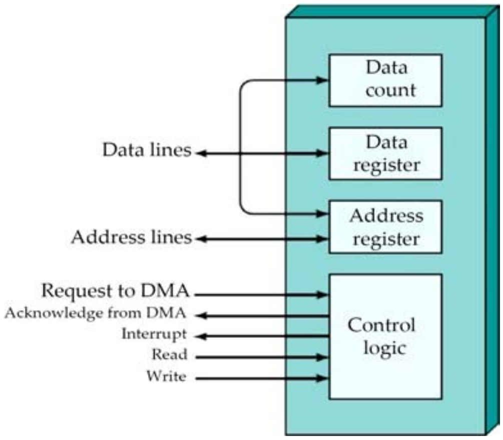


**Figure 10.10.** Using direct access memory.

*Source: https://gcallah.github.io/OperatingSystems/graphics/IOSw_DMA_Read.png.*

CHAPTER 11

# OPERATING SYSTEM – SECURITY

## CONTENTS

Operating system security (OS security) is the process of assuring the integrity, confidentiality, and reliability of an OS.

OS security refers to the process or methods implemented to protect the OS against threats like as viruses, worms, malware, or distant hacker intrusions. Such preventive-control methods that secure any computerized assets that could be seized, manipulated, or destroyed if OS security is broken are included in OS security (Sangorrin et al., 2010).

The term OS security consists of techniques and practices that can ensure the confidentiality, integrity, and resilience of an OS (CIA).

The goal of OS security is to protect the OS against numerous threats like malicious software like worms, trojans, as well as other viruses, incorrect setups, and remote intrusions.

Generally, OS security entails the adoption of control measures that can safeguard your assets against unauthorized alteration and destruction, as well as theft.

OS security encompasses all preventive and control mechanisms implemented on a computer to protect it and other connected devices (e.g., printers) that contain secret information that hackers will likely take, modify, or destroy if the system is infiltrated (Figure 11.1).

**Figure 11.1.** Operating system security.

*Source: https://www.techslang.com/definition/what-is-operating-system-security/.*

Consider OS to be all of the processes (e.g., heading through a security check at an entrance of the building, and so on.) and safeguards (e.g., locking out all unauthorized access from internal staff-only areas, etc.), used by building management and employees to keep crooks and other undesirable people out of company premises (Singh, 2014).

The most typical methods for protecting OSs include using antivirus software as well as other terminal safety protocols, many OS patch updates, a firewall for controlling network traffic, and the implementation of secure access through minimum privileges and user controls.

OS security includes a wide range of approaches and methods for protecting against threats and attacks. OS security enables multiple applications and programs to accomplish needed activities while preventing illegal intervention (Figure 11.2).

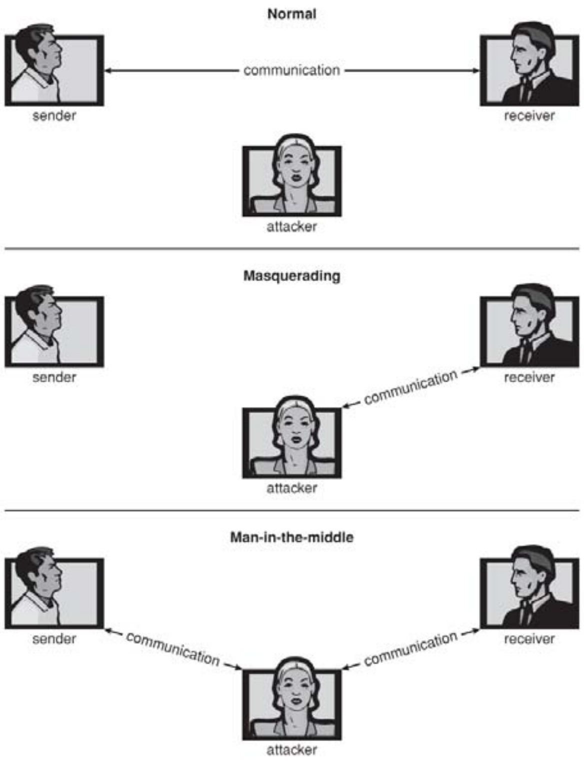


**Figure 11.2.** Standard security attacks.

*Source: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/15_Security.html.*

OS security can be tackled in a variety of ways, such as the following (Figure 11.3):

- Updating the OS's patches on a regular basis;
- Updating antivirus engines and software;
- Using a firewall to inspect all incoming and outgoing network connections;
- Establishing secure accounts with just the required privileges (i.e., user management).

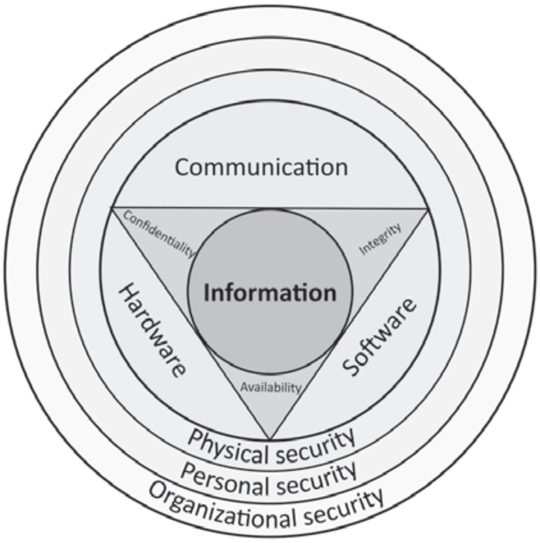


**Figure 11.3.** Information security.

*Source: https://en.wikipedia.org/wiki/Security-focused_operating_system#/media/File:CIAJMK1209-en.svg.*

## 11.1. COMMON OS SECURITY THREATS

Some of the most frequent threat vectors that can harm an OS are discussed in subsections.

### 11.1.1. Authentication

Authentication is the procedure of detecting each system user and associating the programs that are operated by those users. It is the responsibility of the OS to create a security mechanism that ensures the validity of a user who is running a certain program. OSs often employ the three methods listed below to recognize and authenticate users:

- To access or log into the system, the user needs to submit an existing username and password with the OS;
- To log in, the user must insert a card into a card slot or insert a key generated by a key generator into an area given by the OS; and
- User attribute – to login into the system, the user needs to pass his or her physical attribute (could be a retina scan or a fingerprint identifier) through the OS's designated input device.

### 11.1.2. Malware

Malware is short for malicious software, that comprises of a wide range of attack vectors such as viruses, trojans, worms, and rootkits. Malware is software that is introduced into a system without the consent of the owner, or that masquerades as legitimate software, with the intent of stealing, damaging, or distorting data, or corrupting the device (Shin et al., 2014).

Malware can also replicate itself, causing it to spread further inside and outside of a business network. Malware attacks are typically undiscovered by the target victim, allowing for the stealthy extraction of critical data. In other cases, attackers' "herd" hijacked devices into botnets, where they are exploited for illegal operations such as distributed denial of service (DDoS) attacks (Figure 11.4).

**Figure 11.4.** Malicious software.

*Sources: https://www.metacompliance.com/blog/what-is-malware-and-how-to-prevent-against-it/.*

## 11.2. PROGRAM THREATS

The processes and kernel of the OS carry out the prescribed task as ordered. When a user application induces these processes to do harmful tasks, this is referred to as a program threat. A common scenario of a software threat is a program loaded on a system that can save and send user credentials to a hacker through a network. The list includes well-known program dangers.

### 11.2.1. Trojan Horse

A Trojan Horse is a program that, in complement to its visible actions, surreptitiously executes destructive behavior.

Some Trojan horses are purposefully designed to be that way, while others are the consequence of genuine programs that got infected with viruses.

Long search routes, particularly those that have the current directory (.””) as part of the path, are a potential entry point for Trojan horses. If a harmful application with the same name as a genuine program (or a common misspelling, like “sl” instead of “ls”) is inserted wherever on the path, a user could be tricked into launching the erroneous program.

A login emulator is another common Trojan Horse that collects a user’s account ID and password, displays a “password wrong” message, and then logs the user out of the system. The user then attempts afresh (with a legitimate login attempt), correctly gets in, and is unaware that their data has been stolen (Figure 11.5).

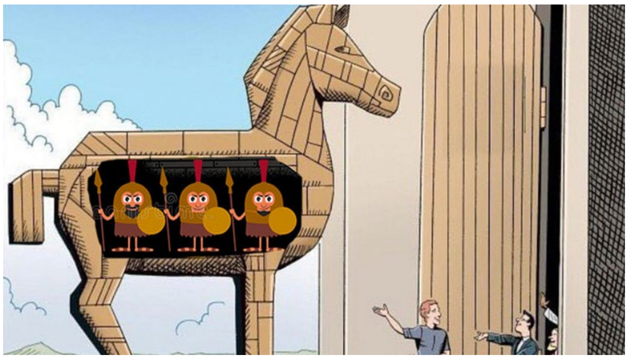


**Figure 11.5.** Trojan horse.

*Source: https://knowyourmeme.com/memes/trojan-horse-object-labels.*

To combat Trojan Horses, two methods are to have the system display the usage data on logouts and to necessitate the use of non-trappable key combinations like Control-Alt-Delete to log in. This is the reason this specific series is required to begin logging in, that can't be imitated or captured by standard programs. That is, that key sequence always gives control to the OS (Androulaki et al., 2018). Spyware is a type of Trojan Horse that is frequently found in software acquired from the Internet. Spywares display pop-ups that possibly may collect data on the user and send it to a centralized server. This is an illustration of a covert channel where secret communications take place. Some other typical tasks of spyware are to transmit spam e-mails that appear to be sent by the afflicted user.

### 11.2.2. Trap Door

A trap door occurs when a developer or programmer purposefully adds a security flaw which would later be exploited to gain access to the system.

Due to the possibilities of trap doors, if a system becomes unreliable, it could never think of as reliable again. Backups may also hold a duplicate of a cleverly concealed back door.

A smart trap door could be placed into a compiler, resulting in a security weakness in just about any programs created using that compiler. This is particularly problematic because a visual review of the code being generated would disclose no errors (Figure 11.6).

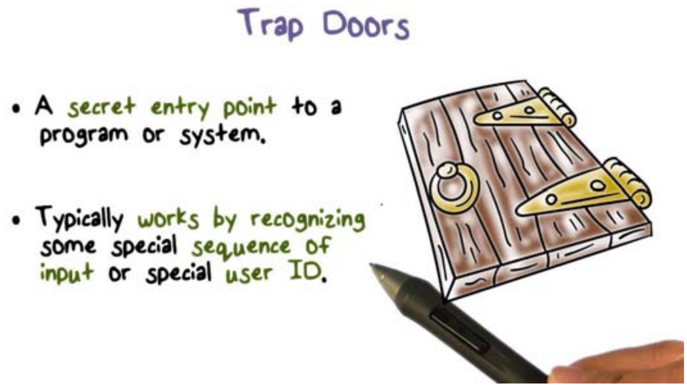

Trap Doors
• A secret entry point to a program or system.
• Typically works by recognizing some special sequence of input or special user ID.

**Figure 11.6.** Trap doors.

*Source: https://www.youtube.com/watch?v=iDZjyotmkRw.*

### 11.2.3. Logic Bomb

A logic bomb is software that is designed to cause havoc only when a specified set of circumstances is met, such as when a specific date or time is met or some other significant event occurs.

A common example is the Dead-Man Switch, which is designed to check if a specific person (for example, the writer) logs in frequently, and if they don't log in for a longer length of time (ostensibly as they've been ejected), then the logic bomb goes off, potentially exposing security vulnerabilities or resulting in other problems (Figure 11.7).

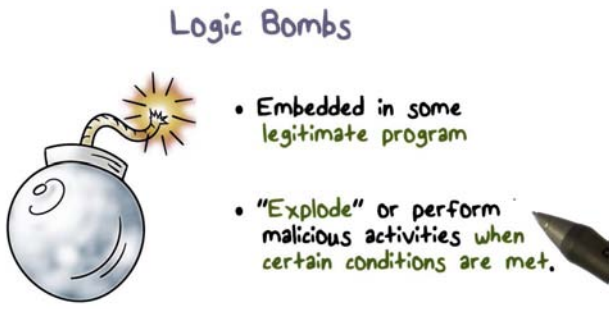

Logic Bombs
• Embedded in some legitimate program
• "Explode" or perform malicious activities when certain conditions are met.

**Figure 11.7.** Logic bombs.

*Source: https://www.youtube.com/watch?v=yZswqdrb88s.*

### 11.2.4. Virus

Viruses, as the term suggests, have the power to move throughout computer systems. They are incredibly harmful, with the capacity to alter or even delete user files and crash machines. A virus is usually a small piece of code that is embedded in software. The virus starts to attach itself in other files or programs as the user interacts with the program, potentially turning the user's machine useless (Tang et al., 2010).

A virus is a bit of code that is placed in a legitimate program and is designed to multiply (by infecting other programs) and (in the end) wreak havoc.

Viruses are more effective in attracting PCs than UNIX or other multi-user systems, because applications in the latter have limited authorization to update other programs or access critical system structures (like the boot system).

Viruses are often spread to systems by a viral payload, which is generally a Trojan Horse, as well as via e-mail or harmful downloads. Viruses can manifest themselves in a variety of ways (Figure 11.8).

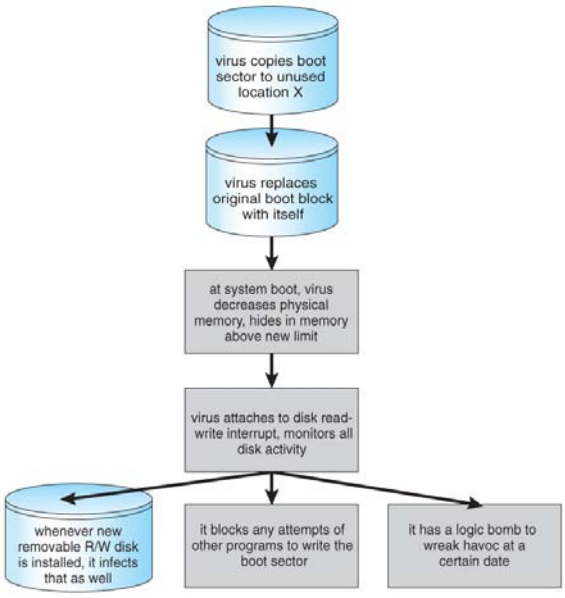

virus copies boot sector to unused location X
virus replaces original boot block with itself
at system boot, virus decreases physical memory, hides in memory above new limit
virus attaches to disk read-write interrupt, monitors all disk activity
whenever new removable R/W disk is installed, it infects that as well
it blocks any attempts of other programs to write the boot sector
it has a logic bomb to wreak havoc at a certain date

**Figure 11.8.** A boot-sector computer virus.

*Source: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/15_Security.html.*

Below is a list of some computer viruses:

- **File:** A file virus infects itself onto executable file, forcing it to run the viral code first and then return to the original program's beginning. These viruses are classified as parasitic since they leave no new files behind and the initial application remains fully functional.
- **Boot:** A boot virus infects the boot system and operates before the OS loads. These viruses are sometimes referred to as memory viruses since they operate in memory which do not show in the file system.
- **Macro Viruses:** These viruses reside in the form of a macro (script) which is executed by a specific set of programs that run them like Microsoft Word or Excel. Macroviruses can be found in word processing or spreadsheets.

- **Source Code:** Viruses seek out and infect source code to be able to spread.
- **Polymorphic:** These viruses modify when they spread – not their fundamental functioning, but only the way they register and are detected by virus checkers.
- **Encrypted:** These viruses move in encryption in order to avoid being found out. In reality, decrypt themselves, allowing them to infect more files.
- **Stealth:** These viruses attempt to stay undetected by changing system components that can be used to spot them. For instance, the read system call could be updated such that when an infected file is read, the infected portion is ignored and the reader is presented with the original, unaltered file (Tsolakis et al., 2019).
- **Tunneling:** These viruses try to evade detection by embedding themselves through interrupt handler chains or device drivers.
- **Multipartite:** These viruses target several sections of the system, including files, the boot sector, and memory.
- **Armored:** These viruses are coded in such a way that anti-virus experts find it difficult to decode and analyze them. Furthermore, many virus-related files are buried, secured, or given innocuous-sounding names like "…"

A virus attacked three weaknesses in Microsoft products to infect scores of Windows servers (along with many trustworthy sites) using the Internet Information Server, infecting any web browser that accessed any of the infected server sites. A keystroke logger, which captures users' keystrokes, revealing passwords and other confidential material, was one of the backdoor apps it installed.

In the computing field, there is some dispute about whether a monoculture, in which almost every system run the very same hardware, OS, and applications, enhances the threat of viruses and their potential harm.

## 11.3. SYSTEM THREATS

System threats are described as the exploitation of system services and network communications to produce user difficulties. A system threat could be employed to launch program threats throughout a whole network, resulting in a program attack. System vulnerabilities provide an environment in which

OS resources and user files are misused. A collection of well-known system threats is provided in subsections.

### 11.3.1. Worm

Worm is a mechanism that can choke the performance of a system by using all system resources. A Worm process generates several clones, each of which uses system resources and prevents all other processes from acquiring them (Arshad et al., 2018). Worm processes have the potential to put a network to a standstill.

A worm is a process that duplicates itself using the fork or spawn process in order to wreak havoc on a system. Worms use system resources, usually interfering with legitimate processes. Worms that propagate across networks can be especially disruptive since they can waste vast amounts of network resources and bring down large-scale systems. In 88, Robert Morris, a graduate student at Cornell, developed one of the most well-known worms. The worm, which targeted Sun and VAX computers that run BSD UNIX version 4, traversed the Internet in a couple of hours and devoured enough resources to drive many systems down (Figure 11.9).

This worm was made up of two elements:

- A small program known as a grappling hook, was implanted on the target system through one of three vulnerabilities; and
- The main worm program, that was moved onto the target system and transmitted by the grappling hook program.

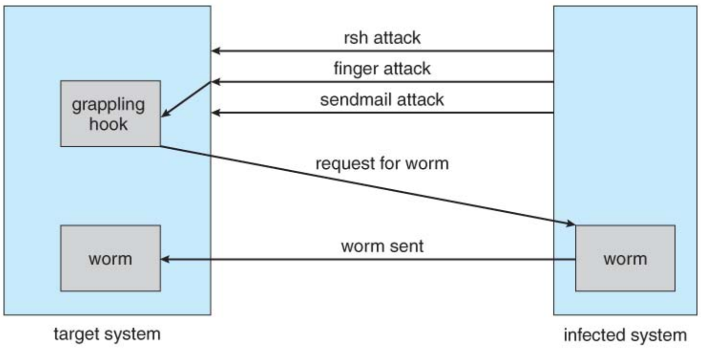


**Figure 11.9.** The Morris internet worm.

*Source: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/15_Security.html.*

### 11.3.2. Port Scanning

Port scanning is a strategy or methodology used by hackers to identify system flaws in order to conduct a network attacks.

Port scanning is not technically an attack, but rather a search for weaknesses to exploit. The basic aim is to integrate to every open (or common or likely) network port on some distant machine and make contact. When it has been established that a specific computer is watching a specific port, the next objective is to establish what daemon is listening and whether or not it is an edition with a known security flaw that could be exploited (Wentzlaff et al., 2010).

Port scanning is frequently launched by zombie systems since it is easily detected and tracked. Specifically, previously compromised systems are being exploited without the knowledge or agreement of their true owner. As a result, it is vital to protect both "innocent" systems and accounts as well as those carrying sensitive data or special privileges.

Other port scanners are available for administrators to utilize to test their own systems, which reveal any faults identified but do not attack or create any problems. Such systems include Nmap (http://www.insecure.org/nmap) and Nessus (http://www.nessus.org). The former determines the OS, firewalls, and which services are listening on which ports. The latter also maintains a database of known security flaws and detects those that are uncovered.

### 11.3.3. Denial of Service

Denial of service attacks often prevent users from lawfully utilizing the system. For instance, if a denial-of-service attack hits the browser's information preferences, the user might be incapable to access the internet.

DOS attacks do not intend to gain access to or harm computers, but rather to clog them so excessively that they can't be employed any practical purposes. This attack consists of short loops that constantly request system services (Whipple et al., 2009).

DOS attacks can also involve social engineering, such as Internet chain letters that say "send this promptly to 10 of your friends, then go to a certain URL," which clogs not only the Internet mail system but also the web server to which everyone is sent.

Security systems that block accounts after a certain number of failed attempts are subject to DOS assaults, which repeatedly attempt to login to all accounts with wrong passwords in order to block all accounts.

DOS is not always the consequence of evil intent. Consider the following:

- A website that receives a large number of visitors as a consequence of a successful marketing effort;
- CNN.com periodically becomes overburdened on major news days, like September 11, 2001;
- When given their first programming project utilizing fork(), CS students frequently quickly fill up process tables or otherwise occupy all system resources. :-)
- When functioning on the inter-process communications assignment, please use ipcs and ipcrm!

## 11.4. ONE TIME PASSWORDS

One-time passwords, in addition to regular authentication, provide an additional degree of protection. A unique password is required each time a user attempts to connect into the One-Time Password system. A one-time password cannot be reused after it has been used once. One-time passwords can be used in a variety of situations.

Users are randomly assigned cards with numbers and corresponding alphabets printed on them. The system asks for integers that correspond to a few alphabets chosen at random (Quigley et al., 2009). Users are provided with a hardware device capable of generating a secret ID that is linked to their user ID. The system prompts you for a secret id, which you must enter every time you log in.

Some commercial apps generate one-time passwords and deliver them to registered cellphone or email addresses, that must be entered before logging in (Figure 11.10).

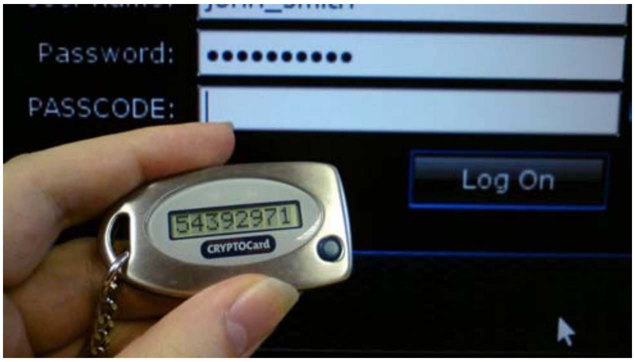


**Figure 11.10.** One-time passwords (OTP).

*Source: https://www.10duke.com/blog/one-time-passwords-a-beginners-guide/.*

## 11.5. COMPUTER SECURITY CLASSIFICATIONS

There are four security classes in computer systems, according to the US Department of Defense Trusted Computer System Evaluation Criteria: A, B, C, and D. This specification is commonly used to determine and simulate the security of systems and security solutions. Each classification is described briefly below.

| SL. No. | Classification Type and Description |
|---|---|
| 1. | **Type A:** The highest level. Formal design specifications and verification methodologies are used. Provides a high level of process security assurance. |
| 2. | **Type B:** Provides a system of required protection. Have all of the characteristics of a class C2 system. Each object is given a sensitivity label. There are three kinds of it.<br>· B1 keeps track of each object's security label in the system. Labels are used to make access control choices.<br>· B2 Extends sensitivity labels to all system resources, like storage objects, and provides covert channels and event auditing (Yin et al., 2016).<br>· B3 Enables the creation of lists or user groups for access control in order to grant or revoke access to a designated object. |

| | |
|---|---|
| 3. | **Type C:** Using audit capabilities, it offers protection and user accountability. There are two kinds of it.<br>· C1 includes settings that allow users to protect their personal information and prevent other users from mistakenly reading or deleting their data. The majority of UNIX versions are Cl class.<br>· C2 extends the possibilities of a Cl level system by adding individual-level access control. |
| 4. | **Type D:** Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category. |

# BIBLIOGRAPHY

Ahmad, M. S., Musa, N. E., Nadarajah, R., Hassan, R., & Othman, N. E., (2013). Comparison between android and iOS operating system in terms of security. In: *2013 8th International Conference on Information Technology in Asia (CITA)* (pp. 1–4). IEEE.

Aksoy, A., Louis, S., & Gunes, M. H., (2017). Operating system fingerprinting via automated network traffic analysis. In: *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2502–2509). IEEE.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., & Yellick, J., (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. In: *Proceedings of the 13th EuroSys Conference* (pp. 1–15).

Arshad, S., Shah, M. A., Wahid, A., Mehmood, A., Song, H., & Yu, H., (2018). SAMADroid: A novel 3-level hybrid malware detection model for android operating system. *IEEE Access, 6*, 4321–4339.

Bala, K., Sharma, S., & Kaur, G., (2015). A study on smartphone-based operating system. *International Journal of Computer Applications, 121*(1).

Bhattacharjee, A., & Lustig, D., (2017). Architectural and operating system support for virtual memory. *Synthesis Lectures on Computer Architecture, 12*(5), 1–175.

Blackham, B., Shi, Y., Chattopadhyay, S., Roychoudhury, A., & Heiser, G., (2011). Timing analysis of a protected operating system kernel. In: *2011 IEEE 32nd Real-Time Systems Symposium* (pp. 339–348). IEEE.

Cao, Q., Abdelzaher, T., Stankovic, J., & He, T., (2008). The liteOS operating system: Towards Unix-like abstractions for wireless sensor networks. In: *2008 International Conference on Information Processing in Sensor Networks (IPSN 2008)* (pp. 233–244). IEEE.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., & Carreras, M., (2015). ROSPlan: Planning in the robot operating system. In: *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 25, pp. 333–341).

Chinetha, K., Joann, J. D., & Shalini, A., (2015). An evolution of android operating system and its version. *International Journal of Engineering and Applied Sciences, 2*(2), 257997.

De, P., Kothari, R., & Mann, V., (2007). Identifying sources of operating system jitter through fine-grained kernel instrumentation. In: *2007 IEEE International Conference on Cluster Computing* (pp. 331–340). IEEE.

Deseriis, M., (2017). Direct parliamentarianism: An analysis of the political values embedded in Rousseau, the "operating system" of the five-star movement. In: *2017 Conference for E-Democracy and Open Government (CeDEM)* (pp. 15–25). IEEE.

Dieber, B., Breiling, B., Taurer, S., Kacianka, S., Rass, S., & Schartner, P., (2017). Security for the robot operating system. *Robotics and Autonomous Systems, 98*, 192–203.

DiLuoffo, V., Michalson, W. R., & Sunar, B., (2018). Robot operating system 2: The need for a holistic security approach to robotic architectures. *International Journal of Advanced Robotic Systems, 15*(3), 1729881418770011.

Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., & Bahl, P., (2012). An operating system for the home. In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (pp. 337–352).

Estefo, P., Simmonds, J., Robbes, R., & Fabry, J., (2019). The robot operating system: Package reuse and community dynamics. *Journal of Systems and Software, 151*, 226–242.

Fierro, G., & Culler, D. E., (2015). XBOS: An extensible building operating system. In: *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments* (pp. 119–120).

Fröhlich, A. A., & Wanner, L. F., (2008). Operating system support for wireless sensor networks. *Journal of Computer Science, 4*(4), 272.

Giorgetti, A., Sgambelluri, A., Casellas, R., Morro, R., Campanella, A., & Castoldi, P., (2020). Control of open and disaggregated transport networks using the open network operating system (ONOS). *Journal of Optical Communications and Networking, 12*(2), A171–A181.

Greenwald, L. G., & Thomas, T. J., (2007). *Toward Undetected Operating System Fingerprinting* (Vol. 7, pp. 1–10). Woot.

Gunadi, H., & Tiu, A., (2014). Efficient runtime monitoring with metric temporal logic: A case study in the android operating system. In: *International Symposium on Formal Methods* (pp. 296–311). Springer, Cham.

Hambarde, P., Varma, R., & Jha, S., (2014). The survey of real time operating system: RTOS. In: *2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies* (pp. 34–39). IEEE.

Hellmund, A. M., Wirges, S., Taş, Ö. Ş., Bandera, C., & Salscheider, N. O., (2016). Robot operating system: A modular software framework for automated driving. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (pp. 1564–1570). IEEE.

Høiland-Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T., Ahern, D., & Miller, D., (2018). The express data path: Fast programmable packet processing in the operating system kernel. In: *Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies* (pp. 54–66).

Irwansyah, F. S., Yusuf, Y. M., Farida, I., & Ramdhani, M. A., (2018). Augmented reality (AR) technology on the android operating system in chemistry learning. In: *IOP Conference Series: Materials Science and Engineering* (Vol. 288, No. 1, p. 012068). IOP Publishing.

Jaeger, T., (2008). Operating system security. *Synthesis Lectures on Information Security, Privacy, and Trust, 1*(1), 1–218.

Jeong, K., Kim, J., & Kim, Y. T., (2012). QoS-aware network operating system for software defined networking with generalized OpenFlows. In: *2012 IEEE Network Operations and Management Symposium* (pp. 1167–1174). IEEE.

Krohn, M., & Tromer, E., (2009). Noninterference for a practical DIFC-based operating system. In: *2009 30th IEEE Symposium on Security and Privacy* (pp. 61–76). IEEE.

Kushwaha, A., & Kushwaha, V., (2011). Location based services using android mobile operating system. *International Journal of Advances in Engineering & Technology, 1*(1), 14.

Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., & Peter, M., (2011). L4Android: A generic operating system framework for secure smartphones. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 39–50).

Lass, S., & Gronau, N., (2020). A factory operating system for extending existing factories to industry 4.0. *Computers in Industry, 115*, 103128.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., & Culler, D., (2005). TinyOS: An operating system for sensor networks. In: *Ambient Intelligence* (pp. 115–148). Springer, Berlin, Heidelberg.

Lin, F., & Ye, W., (2009). Operating system battle in the ecosystem of smartphone industry. In: *2009 International Symposium on Information Engineering and Electronic Commerce* (pp. 617–621). IEEE.

Manzalini, A., & Crespi, N., (2016). An edge operating system enabling anything-as-a-service. *IEEE Communications Magazine, 54*(3), 62–67.

Mayoral, V., Hernández, A., Kojcev, R., Muguruza, I., Zamalloa, I., Bilbao, A., & Usategi, L., (2017). The shift in the robotics paradigm—The hardware robot operating system (H-ROS); an infrastructure to create interoperable robot components. In: *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (pp. 229–236). IEEE.

McClean, J., Stull, C., Farrar, C., & Mascarenas, D., (2013). A preliminary cyber-physical security assessment of the robot operating system (ROS). In: *Unmanned Systems Technology XV* (Vol. 8741, p. 874110). International Society for Optics and Photonics.

Monaco, M., Michel, O., & Keller, E., (2013). Applying operating system principles to SDN controller design. In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks* (pp. 1–7).

Moore, T., & Stouch, D., (2016). A generalized extended Kalman filter implementation for the robot operating system. In: *Intelligent Autonomous Systems 13* (pp. 335–348). Springer, Cham.

Muehlstein, J., Zion, Y., Bahumi, M., Kirshenboim, I., Dubin, R., Dvir, A., & Pele, O., (2017). Analyzing HTTPS encrypted traffic to identify user's operating system, browser, and application. In: *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (pp. 1–6). IEEE.

Nimodia, C., & Deshmukh, H. R., (2012). Android operating system. *Software Engineering, 3*(1), 10.

Nollet, V., Marescaux, T., Verkest, D., Mignolet, J. Y., & Vernalde, S., (2004). Operating-system controlled network on chip. In: *Proceedings of the 41st Annual Design Automation Conference* (pp. 256–259).

Ow, D. W., (2011). Recombinase‐mediated gene stacking as a transformation operating system F. *Journal of Integrative Plant Biology, 53*(7), 512–519.

Peter, S., Li, J., Zhang, I., Ports, D. R., Woos, D., Krishnamurthy, A., & Roscoe, T., (2015). Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems (TOCS)*, *33*(4), 1–30.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., & Ng, A. Y., (2009). ROS: An open-source robot operating system. In: *ICRA Workshop on Open-Source Software* (Vol. 3, No. 3.2, p. 5).

Sangorrin, D., Honda, S., & Takada, H., (2010). *Dual Operating System Architecture for Real-Time Embedded Systems, 6*, 1–24.

Santos, J. M., Portugal, D., & Rocha, R. P., (2013). An evaluation of 2D SLAM techniques available in robot operating system. In: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (pp. 1–6). IEEE.

Sharma, H. K., Shastri, A., & Biswas, R., (2012). A framework for automated database tuning using dynamic SGA parameters and basic operating system utilities. *Database Systems Journal, 3*(4), 25–32.

Shaw, H., Ellis, D. A., Kendrick, L. R., Ziegler, F., & Wiseman, R., (2016). Predicting smartphone operating system from personality and individual differences. *Cyberpsychology, Behavior, and Social Networking, 19*(12), 727–732.

Shin, S., Song, Y., Lee, T., Lee, S., Chung, J., Porras, P., & Kang, B. B., (2014). Rosemary: A robust, secure, and high-performance network operating system. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (pp. 78–89).

Silva, D. D., Krieger, O., Wisniewski, R. W., Waterland, A., Tam, D., & Baumann, A., (2006). K42: An infrastructure for operating system research. *ACM SIGOPS Operating Systems Review, 40*(2), 34–42.

Singh, R., (2014). An overview of android operating system and its security. *Int. Journal of Engineering Research and Applications, 4*(2), 519–521.

Sjöstrand, T., Ask, S., Christiansen, J. R., Corke, R., Desai, N., Ilten, P., & Skands, P. Z., (2015). An introduction to PYTHIA 8.2. *Computer Physics Communications, 191*, 159–177.

Tang, S., Mai, H., & King, S. T., (2010). Trust and protection in the Illinois browser operating system. In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*.

Tsolakis, N., Bechtsis, D., & Bochtis, D., (2019). AgROS: A robot operating system-based emulation tool for agricultural robotics. *Agronomy, 9*(7), 403.

Wentzlaff, D., Gruenwald, III. C., Beckmann, N., Modzelewski, K., Belay, A., Youseff, L., & Agarwal, A., (2010). An operating system for multicore and clouds: Mechanisms and implementation. In: *Proceedings of the 1st ACM Symposium on Cloud Computing* (pp. 3–14).

Whipple, J., Arensman, W., & Boler, M. S., (2009). A public safety application of GPS-enabled smartphones and the android operating system. In: *2009 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 2059–2061). IEEE.

Yin, X., Guo, S., Hirata, H., & Ishihara, H., (2016). Design and experimental evaluation of a teleoperated haptic robot-assisted catheter operating system. *Journal of Intelligent Material Systems and Structures, 27*(1), 3–16.

# INDEX

# Introductory Guide to Operating Systems

The operating system (OS) is a connection point between a PC client and PC equipment. The working framework is a product that plays out every one of the fundamental errands, including file management, memory administration, process administration, managing input and output, and handling peripheral instruments, for example, printers and disk drives. A few well-known Operating Systems incorporate Windows Operating System, Linux Operating System, VMS, OS/400, AIX, z/OS, and so forth. The operating system functions as a point of interaction between the client and the PC equipment and controls the execution of a wide range of programs. A few key subjects captured in this volume incorporate:

- memory management;
- processor management;
- security;
- control over structural output;
- device management;
- file management;
- fault detecting aids; and
- synchronization between other programs and clients.

The operating system (OS) is the presentation of a PC system equipment abstraction, whereby individuals manage the equipment, and to utilize the resources of the PC system. To develop the students' basic understanding, pragmatic capacity, technology, and framework plan must synchronize through programming and hardware as a technique, the PC circuit system, PC design guideline, internal framework configuration, working framework, and structural organization. Accordingly, links between programs are reinforced, and the students' PC systems analysis, design capacity and innovation. Lately, the working framework and its abstraction systems regarding the application programming becomes more complicated. The volume discusses the impact of different components of modern operating system planning and physical operation, clients, and planning, preparation of the structural framework and other systemic programs. This volume examines a few significant forms of operating systems, which are most usually utilized, including multi-programmed batch system and batch operating system. In multiprogramming systems, the OS determines which interaction receives the processor when and overall period. This procedure is called process booking. An operating system performs different functions, such as:

- Evaluates processor and process status;
- Allocates the processor (CPU) to a function;
- De-allots processor when an interaction is not generally needed. An Operating System oversees device correspondence through their individual drivers. The digital system is moving towards cell phones. To find out more on operating systems, clients should acquire active bit programming experience in these conditions, which are very unique in relation to conventional computers and servers.

The functioning of the multi-systemic batched system includes roles being assembled empowering the CPU to execute each occupation in turn permitting the CPU to use its resources. Roles are sustained in the operating structure each in turn. For the role to be done, a program is completed that uses I/O activity. The client cannot become inactive while utilizing a multi-programming framework, as the working framework will change to another system. The batched computer system keeps various tasks in the memory as it conducts a given assignment until it is completely executed. It follows a sequential process. The memory is freed once a given errand is finished and the work yield is moved to a result spoil, permitting it to be handled or printed later. The batch activity framework restricts client communication. The client is free once the framework takes assignments from the batch system.

**Ms. Jocelyn O. Padallan** is a graduate of the Bachelor of Science in Computer Science and finished her Certificate of Teaching Proficiency Program at Laguna State Polytechnic University - Los Banos Laguna. She also completed her Master in Educational Management. Currently, Ms. Jocelyn O. Padallan is an IT Professor at the Laguna State Polytechnic University - Los Banos Campus under the College of Computer Studies (CCS) where she holds different professional courses such as Effective and Business Communication and Programming Languages. She is actively involved in the college extension services programs including Life Project 4 Youth and ITeach 4 Heroes.

ARCLER PRESS

ISBN 978-1-77469-629-3