# DEBRE TABOR UNIVERSITY


**Faculty of technology**
**Department of computer science**

***Course Module for* operating system (Bsc.)**

Compiled by

Huluager W. and Genet W.

DTU (Debre Tabot University)
Monday, 17 January 2022

## Contents page

## CHAPTER ONE

# Introduction

### 1.1. What is a computer operating system?

An operating system is simply a group of computer programs, sometimes called 'program files' or simply 'files', that are generally stored (saved) on a computer disk. Most computers need an operating system to be able to 'boot' (start-up), interact with devices such as printers, keyboards and joysticks, and to provide disk management tasks such as saving or retrieving files to/from your computer disks or to analyse problems with your computer.

There are many flavours of operating systems available in the marketplace today. The programs for the operating system are generally written specifically for the type of hardware they are installed on. For example, the Microsoft Windows operating system works primarily on an IBM-compatible personal computer (pc), whereas the Apple Macintosh operating system works on an Apple personal computer, but will not work on an IBM-compatible computer (without special software called an 'emulator'). UNIX is generally designed for larger mini or mainframe computers but there is now a version available for the desktop computer.

#### 1.1.1. Definition

- ✓ Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers. A program that acts as an intermediary between a user of a computer and the computer hardware.
- ✓ OS is a **resource allocator**
    - ♠ Manages all resources
    - ♠ Decides between conflicting requests for efficient and fair resource use

- ✓ OS is a **control program**: which Controls execution of programs to prevent errors and improper use of the computer
  - **Basic function of the OS are:**
    - ✓ Procès management
    - ✓ Memory management
    - ✓ Devise management
    - ✓ File management
    - ✓ Security management
    - ✓ User interfacing
    - ✓ Coordination of communication on the network

**Operating system goals:**

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

Examples of operating system

- ❖ **DOS: - Disk Operating System** is one of the first operating systems for the personal computer. When you turned the computer on all you saw was the command prompt which looked like c:\ >. You had to type all commands at the command prompt which might look like c:\>wp\wp.exe. This is called a command-line interface. It was not very "user friendly"
- ❖ **Windows**: - The Windows operating system, a product of Microsoft, is a GUI (graphical user interface) operating system. This type of "user friendly" operating system is said to have WIMP features: Windows, Icons, Menus and Pointing device (mouse)
- ❖ **MacOS** : Macintosh, a product of Apple, has its own operating system with a GUI and WIMP features.
- ❖ **UNIX:** - Linux(the PC version of UNIX) - UNIX and Linux were originally created with a command-line interface, but recently have added GUI enhancements.

## 1.2. Computer System Structure

Computer system can be divided into four components

- **Hardware** – provides basic computing resources
  - ✓ CPU, memory, I/O devices
- **Operating system**
  - ✓ Controls and coordinates use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
  - ✓ Word processors, compilers, web browsers, database systems, video games
- **Users**
  - People, machines, other computers

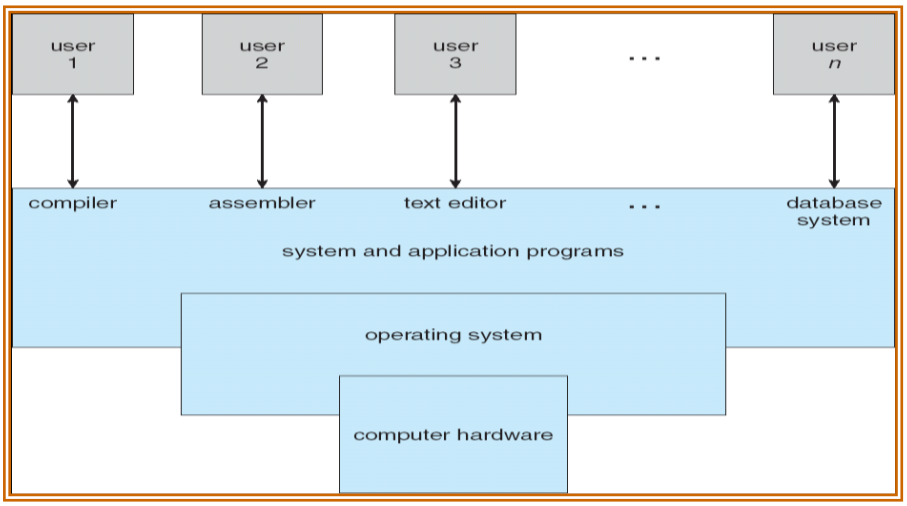



Fig: 1 Diagram of computer system structure

## 1.3. History of Operating System

Operating systems have been evolving through the years. Following table shows the history of OS.

| Generation | Year | Electronic devices used | Types of OS and devices |
|---|---|---|---|
| First | 1945 – 55 | Vacuum tubes | Plug boards |
| Second | 1955 – 1965 | Transistors | Batch system |
| Third | 1965 – 1980 | Integrated Circuit (IC) | Multiprogramming |
| Fourth | Since 1980 | Large scale integration | Pc |

## 1.4. Classes of operating systems

There are also various classes of operating systems, each with its own characteristics.

1. **Single user** — an operating system described as 'single user' means that only one user can use the services of the operating system at any one time. If somebody else wants to use the computer they have to wait until the person using it finishes. Older personal computer operating systems such as MS-DOS and up to Windows 3.0 were single user operating systems. Note that older versions of Windows actually used MS-DOS to operate. Windows simply provided the GUI interface.
2. **Multi user**: Multi user systems allow more than one person to use the operating system resources simultaneously. Obviously, two or more people would not want to physically operate the same computer at the same time, so the ability to do this is provided by network operating systems. A network operating system allows many personal computers to connect to other computers by means of communication media such as cable or wireless links.
3. **Single tasking:** These operating systems are more complex than single user operating systems because they have to handle many requests for devices, resources etc., by many different users at the same time. For example, if three users on a network all try to print a document on a single network printer at the same time, it is the Network operating system's responsibility to ensure that the documents are held on the hard disk (spooled/queued) until the printer is ready to receive them. Multi user systems also provide security functions such as who can access the system, what resources they can use when logged in, what environment areas they can change, etc.

These are operating systems in which only one task can be performed by the operating system at any one time. That single task must finish before the next task can be started. E.g in MS-DOS, if you wanted to format a floppy disk, the computer would need to finish that task before it gave control back to you to allow you perform the next task. Early single user operating systems were single tasking.

4 **Multi-tasking-single user** —This means that a user can sit in front of their computer (that is not attached to a network) and the computer appears to do many tasks at the same time. Eg while the operating system is printing a 100-page document on your printer, your database program is sorting hundreds of records for you, while you play your favourite card games, all at the same time. (Note that the computer does not run these tasks concurrently as explained later).

5 **Multi-tasking-multi user** — If you read the definition above for a multi user system, you would probably have realised that all multi user systems must be multi-tasking.

Some common operating systems

| Name | Computer type | Description |
|---|---|---|
| MS –DOS | IBM-compatible computers | Developed around 1980. A single user, single tasking OS with no GUI features. Not designed for running on a network. Other similar products were DR-DOS and PC-DOS. |
| Windows | IBM-compatible computers | First version appeared around 1985. Never really gained acceptability until the release of Windows 3.1 in 1992. Network capability was added to a new version called Windows for Workgroups later on in the same year. Used a GUI interface and supports Multi User/Multi-Tasking capabilities. Current standard version for the home computer is Windows XP. |
| UNIX | Mainframes and now IBM-compatible computers | Developed around the late 1960's. Has extensive Networking capabilities and handles Multi-Tasking/Multi User functions extremely well. Originally a command line operating system. |

| | | |
|---|---|---|
| LINUX | IBM-compatible computers | A popular freeware operating system that is very similar to UNIX. Have basically the same features as UNIX. Very popular for internet applications such as firewalls, gateways etc. Has a GUI but currently is not quite as user friendly as Windows or Apple Macs. |
| Macintosh OS | Apple Macintosh | Uses a GUI and used it before Microsoft Windows was written. It supports multi-tasking. It is very popular for use in businesses where graphical designing or video work is done. |

## 1.5. Types of operating system

**Batch operating system**

The users of batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers left their programs with the operator. The operator then sorts programs into batches with similar requirements.

The problems with Batch Systems are following.

- Lack of interaction between the user and job.
- CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.
- Difficult to provide the desired priority.

**Time-sharing operating systems**

Time sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing. The main difference between Multi programmed Batch Systems and Time-Sharing Systems is that in case of multi programmed batch systems, objective is to maximize processor use, whereas in Time-Sharing Systems objective is to minimize response time.Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are following

- Provide advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

**Distributed operating System**

Distributed systems use multiple central processors to serve multiple real time application and multiple users. Data processing jobs are distributed among the processors accordingly to which one can perform each job most efficiently.

**The advantages of distributed systems are following.**

- With resource sharing facility user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

**Network operating System**

Network Operating System runs on a server and provides server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), and a private network or to other networks. Examples of network operating systems are Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are following.

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

**Real Time operating System**

Real time system is defines as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. Real time

processing is always on line whereas on line system need not be real time. The time taken by the system to respond to an input and display of required updated information is termed as response time. So in this method response time is very less as compared to the online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail. For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-appliance controllers, Air traffic control system etc.

Best real time example to observe is when car driver tries to rotate the car the car will give flash light to the rotating direction. So such system is real time operating system.

# Chapter Two

## Processes and process management

### 2.1. Process concept

A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run time.

- ✓ Early systems: One program at a time was executed and a single program has a complete control.
- ✓ Modern OSs allows multiple programs to be loaded in to memory and to be executed concurrently. This requires firm control over execution of programs.
- ✓ The notion of process emerged to control the execution of programs.
- ✓ A process is
    - ❖ Unit of work
    - ❖ Program in execution
- ✓ Operating system consists of a collection of processes. Operating system processes execute system code. And user processes execute user code.
- ✓ By switching CPU between processes, the OS can make the computer more productive.
- ✓ A program is simply a text whereas Process (task or job) includes the current activity.– a program in execution; process execution must progress in sequential fashion.

**Components of a process**

- ❖ The program to be executed
- ❖ The data on which the program will execute
- ❖ The resources required by the program– such as memory and file (s)
- ❖ The status of execution

- ✓ program counter
- ✓ Stack

- ✓ A program is a passive entity, and a process is an active entity with the value of the PC. It is an execution sequence.
- ✓ Multiple processes can be associated with the same program (editor). They are separate execution sequences.
- ✓ For CPU, all processes are similar
  - ☞ Batch Jobs and user programs/tasks
  - ☞ Word file
  - ☞ Internet browser
  - ☞ System call
  - ☞ Scheduler

## 2.2. Process State

As a process executes, it changes *state .each process can be in one of the following state*

As a process: executes, it changes state. The state of a process is defiled in part by the current activity of that process. Each process may be in one of the following states:

1. **New**: The process is being created.
2. **Running**: Instructions are being executed.
3. **Waiting**: The process is waiting for some event to occur.
4. **Ready**: The process is waiting to be assigned to a processor.
5. **Terminated**: The process has finished execution.

The names may differ between OSs.

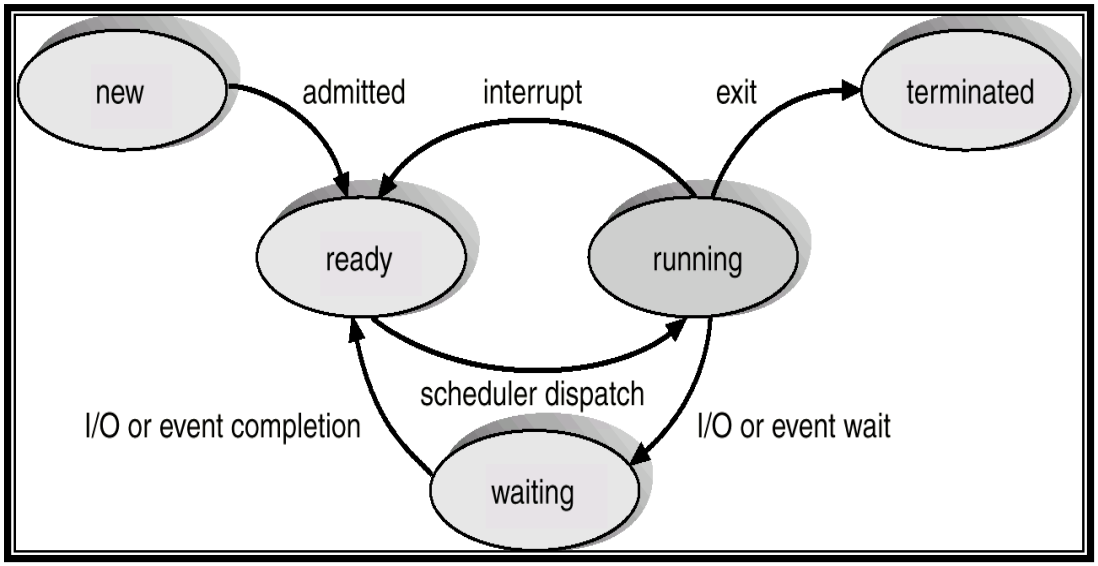



Fig 2: Diagram of Process State

## State change

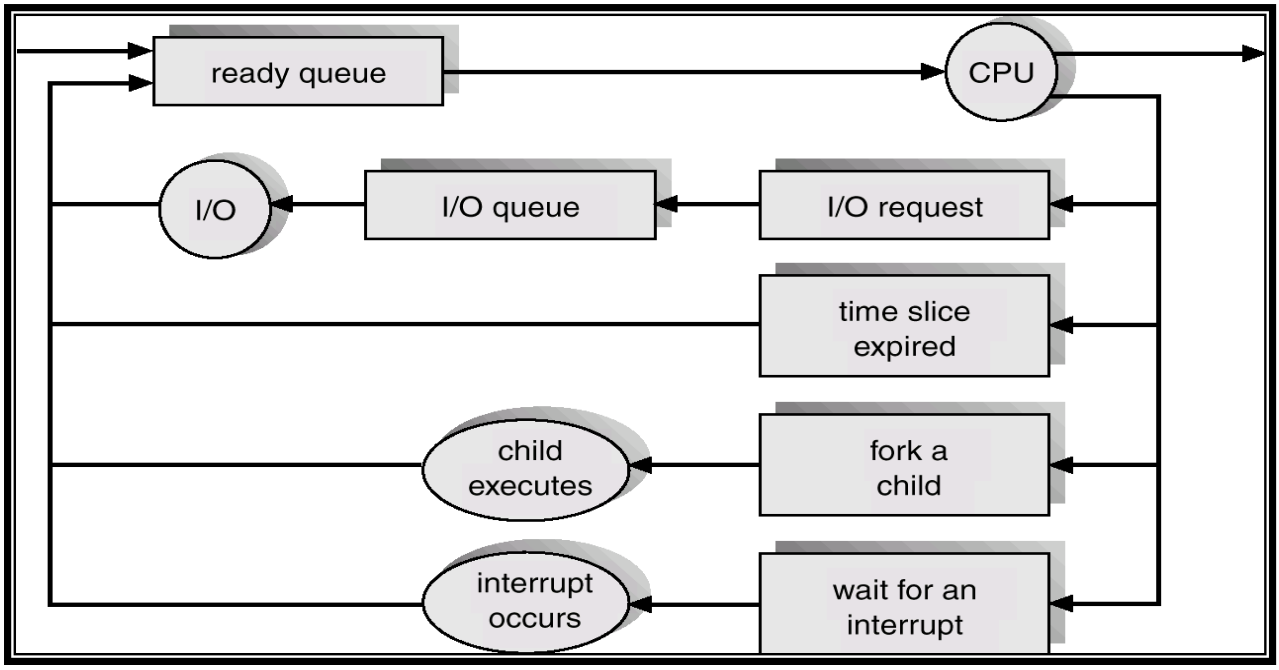
1. **Null → New :** a new process is created to execute the program
   - ☞ New batch job, log on
   - ☞ Created by OS to provide the service
2. **New → ready**: OS will move a process from prepared to ready state when it is prepared to take additional process.
3. **Ready → Running**: when it is a time to select a new process to run, the OS selects one of the processes in the ready state.
4. **Running → terminated:** The currently running process is terminated by the OS if the process indicates that it has completed, or if it aborts.
5. **Running → Ready**: The process has reached the maximum allowable time or interrupt.
6. **Running → Waiting:** A process is put in the waiting state, if it requests something for which it must wait.
   - ☝ Example: System call request.
7. **Waiting → Ready**: A process in the waiting state is moved to the ready state, when the event for which it has been waiting occurs.
8. **Ready → Terminated:** If a parent terminates, child process should be terminated
9. **Waiting → Terminated:** If a parent terminates, child process should be terminated

## 2.3. Process Control Block

Each process is represented in the operating system by control process block also called **Task control block.** It contains many pieces' of information associated with specific process including these:

1. ***Process state***: The state may be new, ready running, waiting, halt (terminate), and so on.
2. ***Program counter***: The counter indicates the address of the next instruction to be executed for this process.
3. ***CPU registers***: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
4. ***CPU-scheduling information***: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameter.
5. ***Memory-management information***: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depend on the memory system used by the operating system.
6. ***Accounting information***: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
7. ***I/O status information***: This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

**Representation of Process Scheduling**




**A new process is initially put in the ready queue**, Once a process is allocated CPU, the following events may occur

- A process could issue an I/O request
- A process could create a new process
- The process could be removed forcibly from CPU, as a result of an interrupt.

When process terminates, it is removed from all queues. Process control block and its other resources are de-allocated.

## 2.4. The threads concept

A thread is a flow of execution through the process code, with its own program counter, system register and stack. Threads are a popular way to improve application performance through parallelism. A thread is sometimes called a **light weight process**.

Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process. Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.

- Inter-process communication is simple and easy when used occasionally. If there are many processes sharing many resources, then the mechanism becomes bulky- difficult to handle. Threads are created to make this kind of resource sharing simple & efficient
- A thread is a basic unit of CPU utilization that consists of:

- ♠ Thread id
- ♠ Program counter
- ♠ Register set
- ♠ Stack

☞ Threads belonging to the same process share :its code, its data section and other OS resources

**Processes and Threads**

A **thread of execution** is the smallest unit of processing that can be scheduled by an OS. The implementation of threads and process differs from one OS to another, but in most cases, **a thread is contained inside a process.**

Multiple threads can exist within the same process and share resources such as **memory**, while different processes do not share these resources. Like process states, threads also have states:

- ✓ New
- ✓ Ready
- ✓ Running
- ✓ Waiting
- ✓ Terminated

Like processes, the OS will switch between threads (even though they belong to a single process) for CPU usage. Like process creation, thread creation is supported by APIs, Creating threads is inexpensive (cheaper) compared to processes.

- ☞ They do not need new address space, global data, program code or operating system resources
- ☞ Context switching is faster as the only things to save/restore **are program counters, registers and stack**

## *Similarity*

- ➢ Both share CPU and only one thread/process is active (running) at a time.
- ➢ Like processes, threads within a process execute sequentially.
- ➢ Like processes, thread can create children.
- ➢ Like process, if one thread is blocked, another thread can run

*Difference*

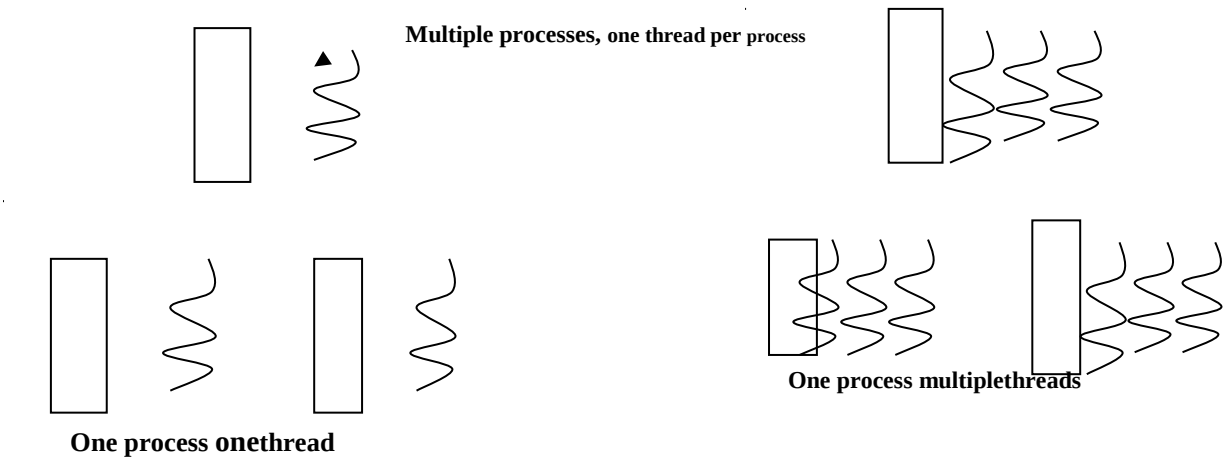
- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task.
- Unlike processes, threads are design to assist one other.

## 2.4.1. Multithreading

**Multithreading** refers to the ability on an operating system to support multiple threads of execution within a **single process.** A traditional (heavy weight) process has a single thread of control. There's one program counter and a set of instructions carried out at a time. If a process has **multiple thread of control**, it can perform more than one task at a time

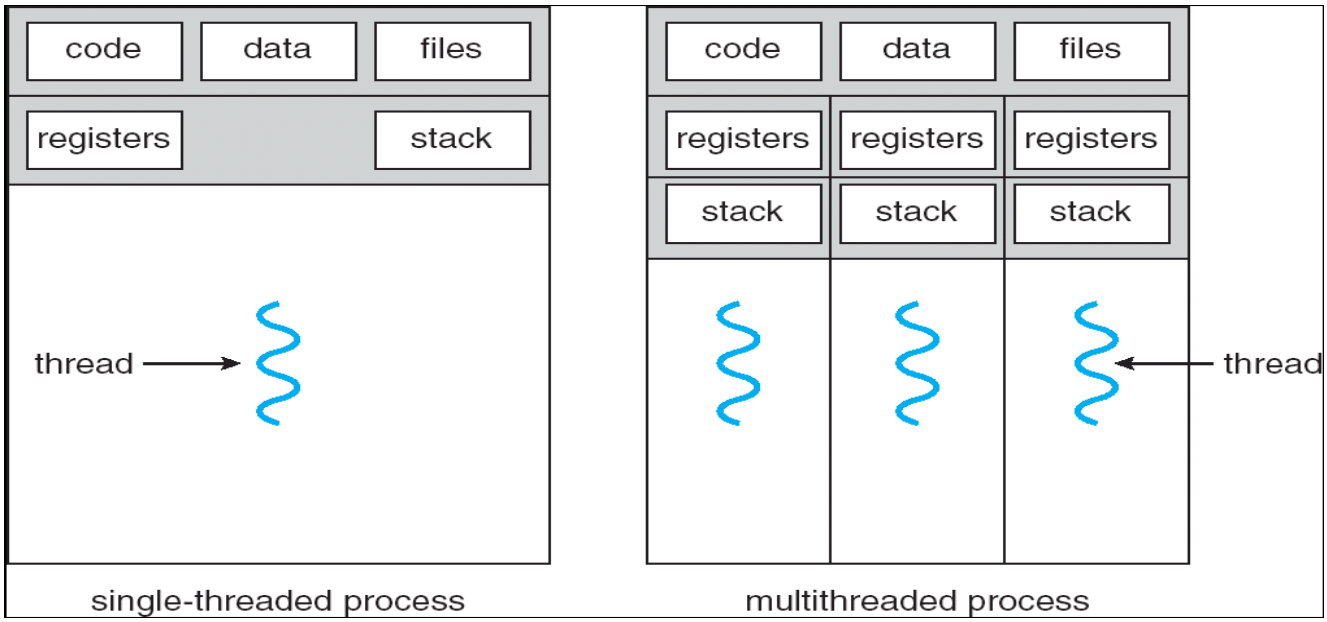
Each thread have their own program counter, stacks and registers But they share common code, data and some operating system data structures like files

- Traditionally there is a single thread of execution per process. Example: MSDOS supports a single user process and single thread.
- UNIX supports multiple user processes but only support one thread per process.
- Multithreading: Java run time environment is an example of one process with multiple threads.
- **Examples** of supporting multiple processes, with each process supporting multiple threads
- Windows 2000, Solaris, Linux, Mach, and OS/2



### *Single and Multithreaded Processes*

- In a single threaded process model, the representation of a process includes its **PCB, user address space, as well as user and kernel stacks.** When a process is running. The contents of these registers are controlled by that process, and the contents of these registers are saved when the process is not running.
- **In a multi-threaded environment,** there is a single PCB and address space, there are separate stacks for each thread as well as separate control blocks for each thread containing register values, priority, and other thread related state information.




## Benefits of Multithreading

There are four major benefits of multi-threading:

1. **Responsiveness:** one thread can give response while other threads are blocked or slowed down doing computations
2. **Resource Sharing:** Threads share common code, data and resources of the process to which they belong. This allows multiple tasks to performed within the same address space
3. **Economy:** Creating and allocating processes is expensive, while creating threads is cheaper as they share the resources of the process to which they belong. Hence, it's more economical to create and context-switch threads.

4. **Scalability/utilization of multi-processor architectures**: The benefits of multithreading is increased in a multi-processor architecture, where threads are executed in parallel. A single-threaded process can run on one CPU, no matter how many are available.

### 2.4.2. Multithreading Models

There are two types of multithreading models in modern operating systems:

1. kernel threads
2. User threads

1. **Kernel threads**: are supported by the OS kernel itself. All modern OS support kernel threads, Need user/kernel mode switch to change threads

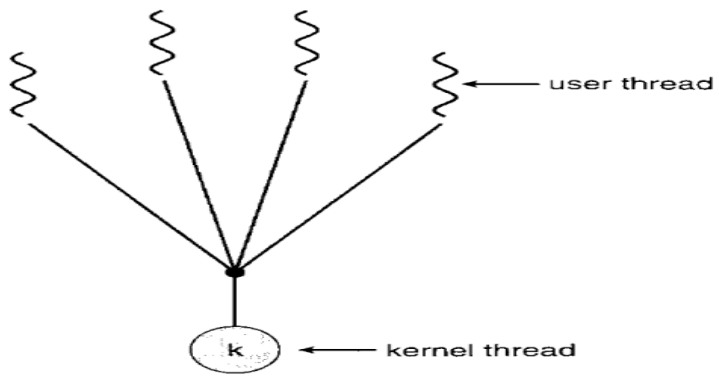
2. **User threads:** Is threads application programmers put in their programs. They are managed without the kernel support, does not require O.S. Support, Has problems with blocking system calls, Cannot support multiprocessing

There must be a relationship between the kernel threads and the user threads. There are 3 common ways to establish this relationship:

1. Many-to-One
2. One-to-One
3. Many-to-Many

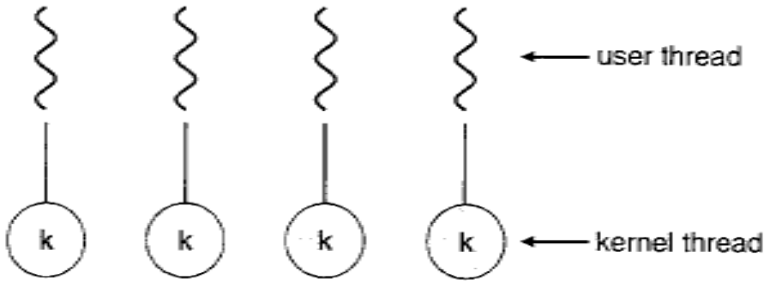
**1. Many-to-One:** it is characterized by the following

- ☞ It maps many user level threads in to one kernel thread
- ☞ Thread management is done by thread library in user space
- ☞ It's efficient but if a thread makes blocking system call, it blocks
- ☞ Only one thread access the kernel at a time, so multiple threads cannot run on multiprocessor systems
- ☞ Used on systems that does not support kernel threads.



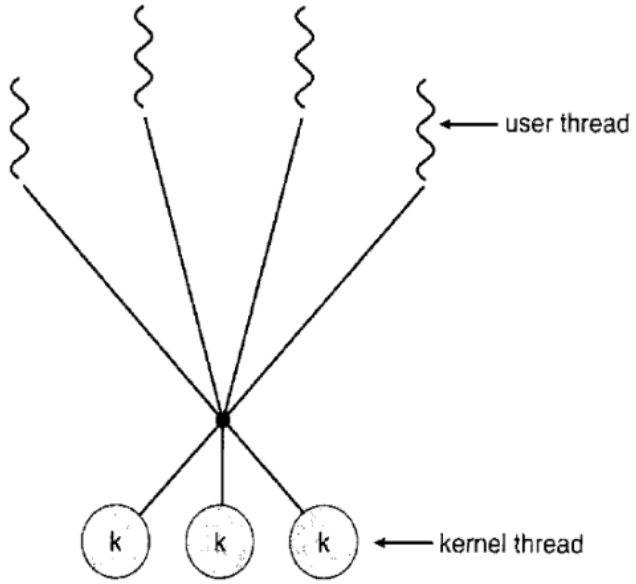
## 2. One-to-One

- Each user-level thread maps to kernel thread.
- A separate kernel thread is created to handle each user-level thread
- It provides more concurrency and solves the problems of blocking system calls
- Managing the one-to-one model involves more **overhead slowed down system**
- **Drawback:** creating user thread requires creating the corresponding kernel thread.
- Most implementations of this thread puts restrictions on the number of threads created



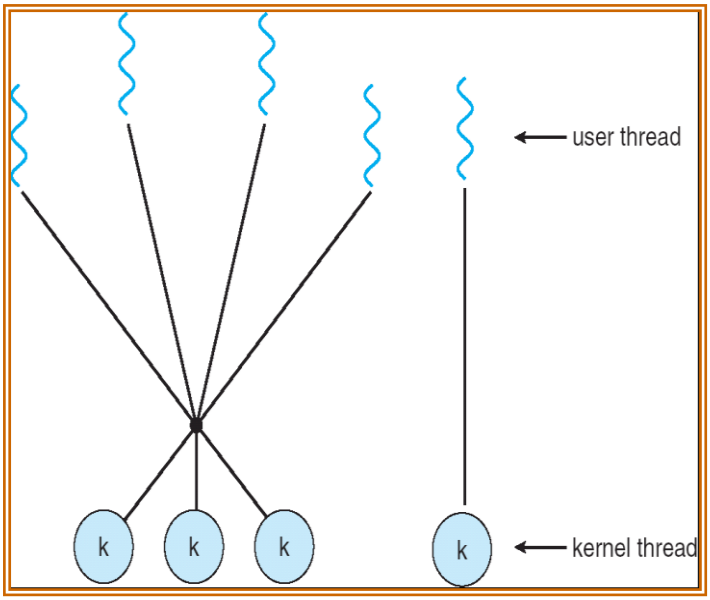

## 3. Many-to-Many

- allows the mapping of many user threads in to many kernel threads
- Allows the OS to create sufficient number of kernel threads
- It combines the best features of one-to-one and many-to-one model
- Users have no restrictions on the numbers of threads created
- Blocking kernel system calls do not block the entire process.




## 4. Two tire model

- Processes can be split across multiple processors
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors
- One popular variation of the many-to-many model is the two-tier model, which allows either many-to-many or one-to-one operation.




Two-tier model

**Threading Issues:** -Some of the issues to be considered for multi-threaded programs are

- Semantics of fork () and exec () system calls.
- Thread cancellation.
- Signal handling
- Thread pools
- Thread-specific data

1. **Semantics of fork () and exec () system calls**:- In multithreaded program the semantics of the fork and exec systems calls change, if one thread calls fork, **there are two options,** New process can duplicate all the threads or new process is a process with single thread and Some systems have chosen two versions of fork
2. **Thread cancellation**:-Task of terminating thread before its completion. Example: if multiple threads are searching database, if one gets the result others should be cancelled
   - **Asynchronous cancellation**: One thread immediately terminates the target thread
   - **Deferred cancellation**: The target thread can paradoxically check if it should terminate
3. **Thread signalling: -** Signals are used in UNIX systems to notify a process that a particular event has occurred

4. **Signal handler**: -is used to process signals.
    - ☞ Signal is generated by particular event
    - ☞ Signal is delivered to a process
    - ☞ Signal is handled

Options: -

- ☞ Deliver the signal to the thread to which the signal applies,
- ☞ Deliver the signal to every thread in the process
- ☞ Deliver the signal to certain threads in the process
- ☞ Assign a specific thread to receive all signals for the process

5. **Threading Pool: - Creating** new threads every time one is needed and then deleting it when it is done can be inefficient, and can also lead to a very large (unlimited) number of threads being created. An alternative solution is to create a number of threads when the process first starts, and put those threads into a ***thread pool***.

Threads are allocated from the pool as needed and returned to the pool when no longer needed. When no threads are available in the pool, the process may have to wait until one becomes available.

The (maximum) number of threads available in a thread pool may be determined by adjustable parameters, possibly dynamically in response to changing system loads.

- ➢ Win32 provides thread pools through the "Pool Function" function.
- ➢ Java also provides support for thread pools.

6. **Thread- specific data: -** Most data is shared among threads, and this is one of the major benefits of using threads in the first place. However sometimes threads need thread-specific data also in some circumstances. Such data is called thread-specific data. Most major thread libraries (P threads, Win32, Java) provide support for thread-specific data.

**Scheduler activation**

Both M: M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application. Scheduler activations provide **up calls** - a communication mechanism from the kernel to the thread library. This communication allows an application to maintain the correct number of kernel threads

### 2.4.3. Types of Thread

Threads are implemented in two ways:

1. User Level

2. Kernel Level

**User Level Thread**

In a user thread, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins running in that thread.

User level threads are generally fast to create and manage.

**Advantage of user level thread over Kernel level thread**:

1. Thread switching does not require Kernel mode privileges.

2. User level thread can run on any operating system.

3. Scheduling can be application specific.

4. User level threads are fast to create and manage.

**Disadvantages of user level thread:**

1. In a typical operating system, most system calls are blocking.

2. Multithreaded application cannot take advantage of multiprocessing.

**Kernel Level Threads**

In Kernel level thread, thread management done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process. The Kernel maintains context information for the process as a whole and for individual's threads within the process.

Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

**Advantages of Kernel level thread:**

a. Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

b. If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

c. Kernel routines themselves can multithreaded.

**Disadvantages:**

1. Kernel threads are generally slower to create and manage than the user threads.

2. Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

**Advantages of Thread**

1. Thread minimizes context switching time.
2. Use of threads provides concurrency within a process.
3. Efficient communication.
4. Economy- It is more economical to create and context switch threads.
5. Utilization of multiprocessor architectures –

The benefits of multithreading can be greatly increased in a multiprocessor architecture.

| Sr. No | User Level Threads | Kernel Level Thread |
|---|---|---|
| 1 | User level thread is faster to create and manage. | Kernel level thread is slower to create and manage. |
| 2 | Implemented by a thread library at the user level. | Operating system support directly to Kernel threads. |
| 3 | User level thread can run on any operating system. | Kernel level threads are specific to the operating system. |
| 4 | Support provided at the user level called user level thread. | Support may be provided by kernel is called Kernel level threads. |
| 5 | Multithread application cannot take advantage of multiprocessing. | Kernel routines themselves can be multithreaded. |

Difference between Process and Thread

| Sr. No | Process | Thread |
|---|---|---|
| 1 | Process is called heavy weight process. | Thread is called light weight process. |
| 2 | Process switching needs interface with operating system. | Thread switching does not need to call a operating system and cause an interrupt to the Kernel. |
| 3 | In multiple process implementations each process | All threads can share same set of |

| | | |
|---|---|---|
| | executes the same code but has its own memory and file resources. | open files, child processes. |
| 4 | If one server process is blocked no other server process can execute until the first process unblocked. | While one server thread is blocked and waiting, second thread in the same task could run. |
| 5 | Multiple redundant process uses more resources than multiple threaded. | Multiple threaded processes use fewer resources than multiple redundant processes. |
| 6 | In multiple processes each process operates independently of the others. | One thread can read, write or even completely wipe out another threads stack |

## 2.5. Inter-process communication

Processes frequently need to communicate with other processes. That communication between processes in the control of the OS is called as Inter process Communication or simply IPC. In some operating systems, processes that are working together often share some common storage area that each can read and write.

To see how IPC works in practice, let us consider a simple but common example, a print spooler. When a process wants to print a file, it enters the file name in a special spooler directory. Another process, the printer daemon, periodically checks to see if there are any files to be printed, and if there are, it sends them to printer and removes their names from the directory.

When two or more processes are reading or writing some shared data and the final result depends on

Who runs precisely when, are called race conditions.

## Race condition:

- The situation where several processes access and manipulate shared data concurrently.
- The final value of the shared data depends upon which process finishes last.
- The key to preventing trouble here and in many other situations involving shared memory, shared files, and shared everything else is to find some way to prohibit more than one process from reading and writing the shared data at the same time .
- To prevent race conditions, concurrent processes must coordinate or be synchronized.

## Critical Section Problem

- Processes ($P_1$, P2, P3….$P_n$) competing to use some shared data.
- Each process has a code segment, called Critical Section (CS), in which the shared data is accessed.
- A critical section is a piece of code in which a process or thread accesses a common shared resource.
- The important features of the system are that – ensure that when one process is executing in its CS, no other process is allowed to execute in its CS. i.e.no two processes are executed in their critical sections at the same time.
- When a process executes code that manipulates shared data (or resource), we say that the process is in its Critical Section (for that shared data).
- The execution of critical sections must be mutually exclusive: at any time, only one process is allowed to execute in its critical section (even with multiple processors).
- So each process must first request permission to enter its critical section.
- The section of code implementing this request is called the **Entry Section (ES).**
- The critical section (CS) might be followed by a **Leave/Exit Section (LS).**
- The remaining code is the **Remainder Section (RS).**
- The critical section problem is to design a protocol that the processes can use so that their action will not depend on the order in which their execution is interleaved (possibly on many processors).

**General structure of process $P_i$(other process $P_j$)**

**Do** {

*Entry section*

**Critical section**

Leave/exit *section*

**Reminder section**

} **while (1)**;

That part of the program where the shared memory is accessed is called the critical section (CS). If we could arrange matters such that no two processes were ever in their critical sections at the same time, we could avoid race conditions.

Although this requirement avoids race conditions, this is not sufficient for having parallel processes cooperate correctly and efficiently using shared data. We need four conditions to hold to have a good solution:

1. No two processes may be simultaneously inside their critical sections.
2. No assumptions may be made about speeds or the number of processors.
3. No process running outside its CS may block other processes.
4. No process should have to wait forever to enter its CS.

**Solution to Critical-Section Problem**

- ***A solution to a critical –section problem must satisfy the following three requirements.***

1. **Mutual Exclusion** - If process $P_i$ is executing in its **critical section,** then no other processes can be executing in their critical sections
2. **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then only those processes that are not executed in their **remainder sections** can participate in the decision on which will enter its critical section next , and this selection cannot be postponed indefinitely
3. **Bounded Waiting -** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted
    - Assume that each process executes at a nonzero speed
    - No assumption concerning relative speed of the N processes

**Solution to CS Problem – Mutual Exclusion**

**Advantages**

- Applicable to any number of processes on either a single processor or multiple processors sharing main memory
- It is simple and therefore easy to verify
- It can be used to support multiple critical sections

**Disadvantages**

- **Busy-waiting** consumes processor time
- **Starvation** is possible when a process leaves a critical section and more than one process is waiting.
- **Deadlock**
- If a low priority process has the critical region and a higher priority process needs, the higher priority process will obtain the **processor(CPU)** to wait for the critical region

## 2.6. Process Scheduling

Multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time multiplexing. The scheduling mechanism is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of particular strategy.

### *Scheduling Queues*

**Process Scheduling Queues**

Scheduling is to decide which process to execute and when

- The objective of **multi-programming** to have some process running at all times.
- **Timesharing**: Switch the CPU frequently that users can interact with the program while it is running.
- If there are many processes, the rest have to wait until **CPU is free**.
- Scheduling is to decide which process to execute and when.
- **Scheduling queues:**
  - ✓ **Job queue** – set of all processes in the system.
  - ✓ **Ready queue** – set of all processes residing in main memory, ready and waiting to execute.
  - ✓ **Device queues** – set of processes waiting for an I/O device.
  - ✓ Each device has its own queue.

Process migrates between the various queues during its life time. When the process enters into the system, they are put into a job queue. This queue consists of all processes in the system. The operating system also has other queues.

Device queue is a queue for which a list of processes waiting for a particular I/O device. Each device has its own device queue. Fig. 3.3 shows the queuing diagram of process scheduling. In the fig 3.3, queue is represented by rectangular box. The circles represent the resources that serve the queues. The arrows indicate the flow of processes in the system.

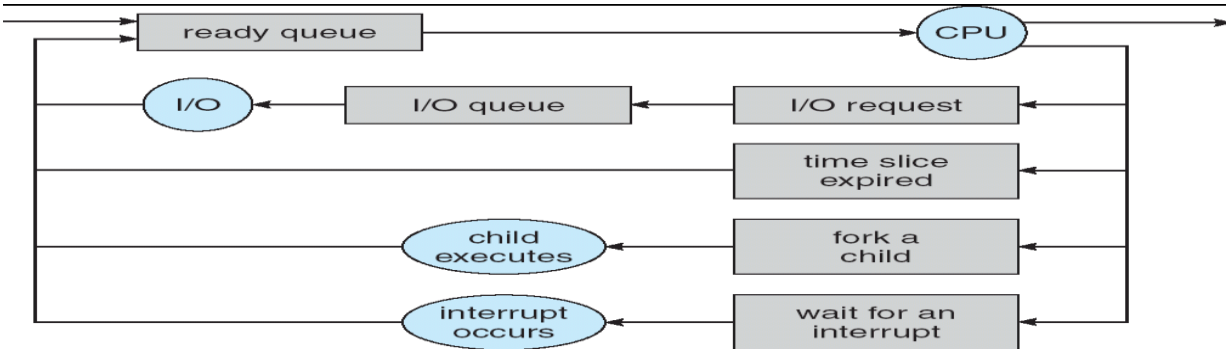



Fig: Queuing Diagram

Queues are of two types: ready queue and set of device queues. A newly arrived process is put in the ready queue. Processes are waiting in ready queue for allocating the CPU. Once the CPU is assigned to the process, then process will execute. While executing the process, one of the several events could occur.

1. The process could issue an I/O request and then place in an I/O queue.

2. The process could create new sub process and waits for its termination.

3. The process could be removed forcibly from the CPU, as a result of interrupt and put back in the ready queue.

Schedules

Schedulers are of three types.

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

**1. Long Term Scheduler**

It is also called job scheduler. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduler. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On same systems, the long term scheduler may be absent or minimal. Time-sharing operating systems have no long term scheduler. When process changes the state from new to ready, then there is a long term scheduler.

**2. Short Term Scheduler**

It is also called CPU scheduler. Main objective is increasing system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects from among the processes that are ready to execute and allocates the CPU to one of them. Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long tern scheduler.

**3. Medium Term Scheduler**

Medium term scheduling is part of the swapping function. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium term scheduler is in charge of handling the swapped out-processes.

Comparison between Scheduler

| Sr. No. | Long Term | Short Term | Medium Term |
|---|---|---|---|
| 1 | It is job scheduler | It is CPU Scheduler | It is swapping |
| 2 | Speed is less than short term scheduler | Speed is very fast | Speed is in between both |
| 3 | It controls degree of multiprogramming | Less control over degree of multiprogramming | Reduce the degree of multiprogramming. |
| 4 | Absent or minimal in time sharing system. | Minimal in time sharing system. | Time sharing system use medium term scheduler. |
| 5 | It select processes from pool and load them into memory for execution. | It select from among the processes that are ready to execute. | Process can be reintroduced into memory and its execution can be continued. |
| 6 | Process state is (New to Ready) | Process state is (Ready to Running) | - |

| | | | |
|---|---|---|---|
| 7 | Select a good process, mix of I/O bound and CPU bound. | Select a new process for a CPU quite frequently. | - |

### 2.6.1 Preemptive vs. non-preemptive scheduling

**Non-preemptive scheduling**

- ☞ Once in running state, process will continue
- ☞ Potential to monopolize the CPU
- ☞ May voluntarily yield the CPU

## •Preemptive scheduling

- ☞ Currently running process may be interrupted by OS andput into ready state
- ☞ Timer interrupts required (for IRP)
- ☞ Incurs context switches
- ☞ Should kernel code be preemptive or non-preemptive?

**Scheduling Criteria 1**

A. User-oriented

- ✓ Response time: Elapsed time between submission of a request and until there is an output
- ✓ Waiting time: Total time process is spending in ready queue
- ✓ Turnaround time : Amount of time to execute a process, from creation to exit

**Scheduling Criteria 2**

- ❖ System-oriented
  - ✓ Effective and efficient utilization of CPU(s)
  - ✓ Throughput: Number of jobs executed per unit of time, Often, conflicting goals

**Scheduling Criteria 3**

- ❖ Performance related:- Quantitative and Measurable, such as response time & throughput
- ❖ Non-performance related :- Qualitative, Predictability and Proportionality

Scheduling Policies

We will concentrate on scheduling at the level of selecting among a set of ready processes. Scheduler is invoked whenever the operating system must select a user-level process to execute:

- after process creation/termination
- a process blocks on I/O
- I/O interrupt occurs
- clock interrupt occurs (if preemptive)

Criteria for a good scheduling

- fairness: all processes get fair share of the CPU
- efficiency: keep CPU busy 100% of time
- response time: minimize response time
- turnaround: minimize the time batch users must wait for output
- throughput: maximize number of jobs per hour
- They are competing. Fairness/efficiency, interactive/batch

## 2.7. Deadlock

Deadlock can be defined as a permanent blocking of processes that either compete for system resources or communicate with each other .The set of blocked processes each hold a resource and wait to acquire a resource held by another process in the set. All deadlocks involve conflicting needs for resources by two or more processes

A set of processes or threads is deadlocked when each process or thread is waiting for a resource to be freed which is controlled by another process

**Example 1**: Suppose a system has 2 disk drives, If $P_1$ is holding disk 2 and $P_2$ is holding disk 1 and if *P1* requests for disk 1 and *P2* requests for disk 2, then deadlock occurs. In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a wait state. It may happen that waiting processes will never again change state, because the resources they have requested are held by other waiting processes. This situation is called deadlock.

A process must request a resource before using it, and must release the resource after using it. A process may request as many resources as it requires carrying out its designated task.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

**1. Request**: If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.

**2. Use:** The process can operate on the resource.

**3. Release**: The process releases the resource

### 2.7.1. Deadlock Characterization

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **Mutual exclusion**: At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. **Hold and wait**: There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.

3. **No preemption**: Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process, has completed its task.

4. **Circular wait**: There must exist a set {P0, P1, ..., Pn } of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …., Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

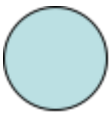
**Resource Allocation Graph**

Deadlock can be better described by using a directed graph called **resource allocation graph.** The graph consists of a set of vertices *V* and a set of edges *E*V is partitioned into two types:

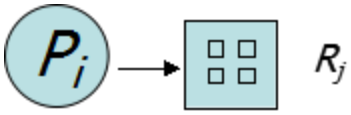
- $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
- $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.

- **request edge** – directed edge $P_1 \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$
- If a Resource Allocation Graph contains a cycle, then a deadlock may exist

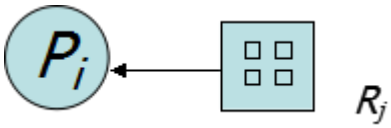
**Diagrammatically, processes and resources in RAG are represented as follow:**

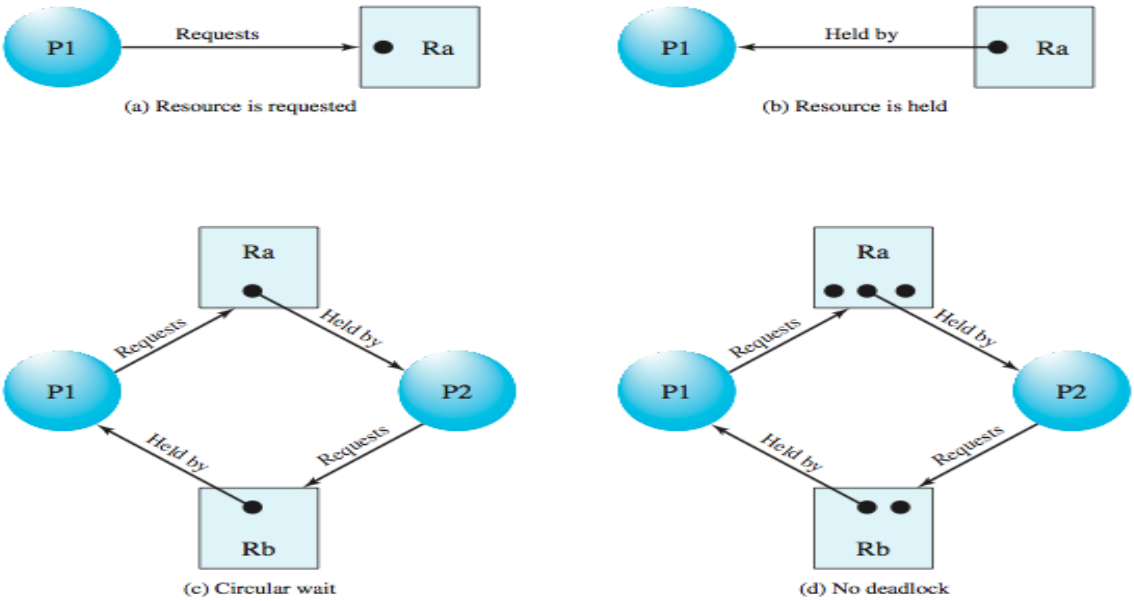
1. **Process:** 
2. **Resource Type with 4 instances:** 

3. **Pi requests instance of Rj:**

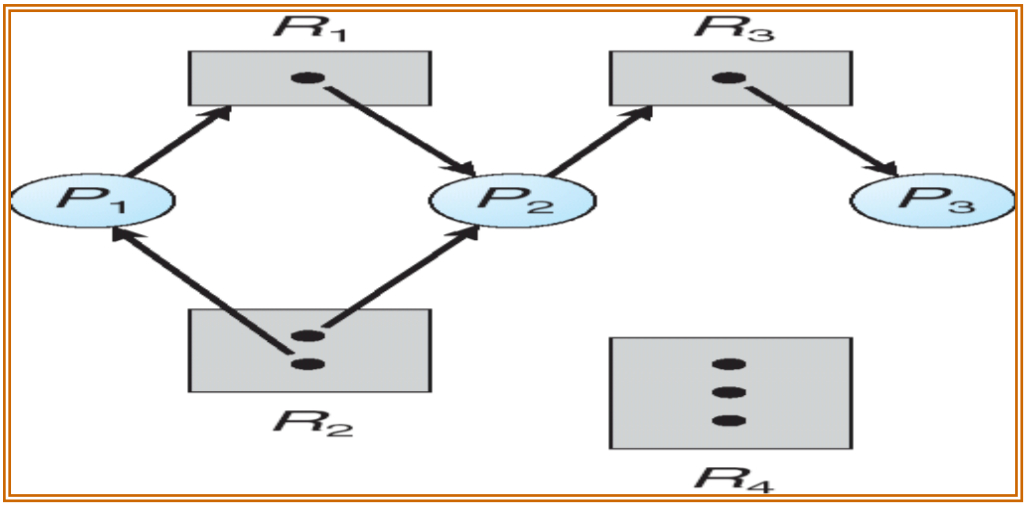



4. **Pi is holding an instance of Rj:**




**Example of Resource Allocation Graph**

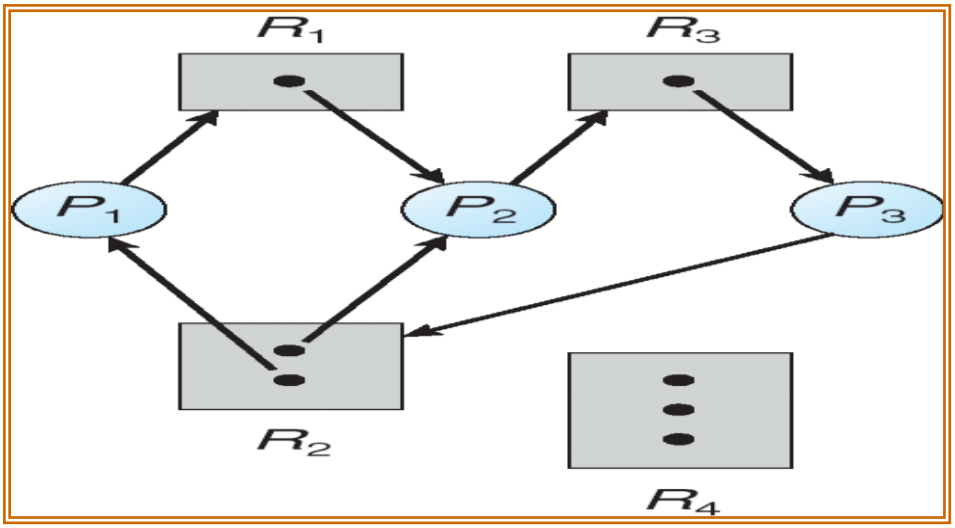



- The RAG shown here tells us about the following situation in a system:
  - P= {*P1, P2, P3}*
  - R= {*R1,R2, R3, R4}*
  - *E ={P1→R1,P2→R3, R1→P2, R2→P1,R3→P3}*

**The process states**

- *P1* is holding an instance of R2 and is waiting for an instance of R1
- *P2* is holding an instance of R1 and instance of R2, and is waiting for an instance of R3
- *P3* is holding an instance of R3
-

**Resource Allocation Graph With a Deadlock**




- There are two cycles in this graph
  - *P1→R1→P2→R3→P3→R2→P1*
  - *P2→R3→P3→R2→P2*
- Processes *P1, P2* and *P3* are deadlocked:

- *P1* is waiting for P2 to release *R1*
- *P2 is waiting for R3 held by P3 and*
- *P3 is waiting for either P1 or P2 to release R2*

### 2.7.2. Methods for handling Deadlocks

Deadlock problems can be handled in one of the following 3 ways:

1. Using a protocol that prevents or avoids deadlock by ensuring that a system will *never* enter a deadlock state **deadlock prevention** and **deadlock avoidance** scheme are used

2. Allow the system to enter a deadlock state and then recover

3. Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

**Deadlock Prevention:** By ensuring at least one of the necessary conditions for deadlock will not hold, deadlock can be prevented. This is mainly done by restraining how requests for resources can be made

- **Deadlock prevention** methods fall into two classes:
    1. An ***indirect*** method of deadlock prevention prevents the occurrence of one of the three necessary conditions listed previously (items 1 through 3)
    2. A ***direct*** method of deadlock prevention prevents the occurrence of a circular wait (item 4)

1. **Mutual Exclusion** – This is not required for sharable resources; however to prevent a system from deadlock, the mutual exclusion condition must hold for non-sharable resources
2. **Hold and Wait** – in order to prevent the occurrence of this condition in a system, we must guarantee that whenever a process requests a resource, **it does not hold any other resources**. **Two protocols are used to implement this:**
    i. Require a process to request and be allocated all its resources before it begins execution or
    ii. Allow a process to request resources only when the **process has none**
    iii. Both protocols have **two main disadvantages**:

Since resources may be allocated but not used for a long period, ***resource utilization will be low***

A process that needs several popular resources has to wait indefinitely because one of the resources it needs is allocated to another process. Hence ***starvation is*** possible.

**3. No Preemption**

If a process holding certain resources is denied further request, that process must release its original resources allocated to it .If a process requests a resource allocated to another process waiting for some additional resources, and the requested resource is not being used, then the resource will be preempted from the waiting process and allocated to the requesting process. Preempted resources are added to the list of resources for which the process is waiting .Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting. This approach is practical to resources whose state can easily save and retrieved easily

**4. Circular Wait**

A linear ordering of all resource types is defined and each process requests resources in an increasing order of enumeration So, if a process initially is allocated instances of resource type R, then it can subsequently request instances of resources types following R in the ordering

### 2.7.3. Deadlock Avoidance

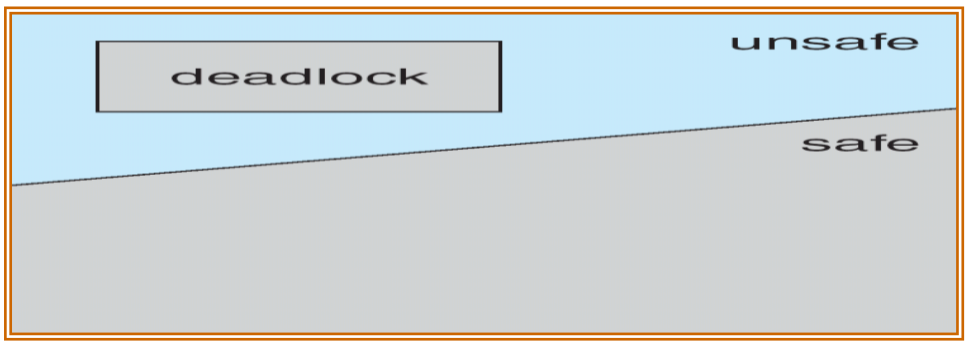
Deadlock avoidance scheme requires each process to declare the ***maximum number*** **of resources of each type that it may need in advance**

- Having this full information about the sequence of requests and release of resources, we can know whether or not the system is entering unsafe state
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a **circular-wait condition**
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes
- **A state is safe** if the system can allocate resources to each process in some order avoiding a deadlock.
- A deadlock state is an **unsafe state**.

Deadlock avoidance: Safe State

Basic Facts

- If a system is in a safe state, then there are no deadlocks.
- If a system is in unsafe state, then there is a possibility of deadlock
- Deadlock avoidance method ensures that a system will never enter an unsafe state




**Deadlock Detection**

If a system does not implement either deadlock prevention or avoidance, deadlock may occur. Hence the system must provide. A deadlock detection algorithm that examines the state of the system if there is an occurrence of deadlock.

- An algorithm to recover from the deadlock

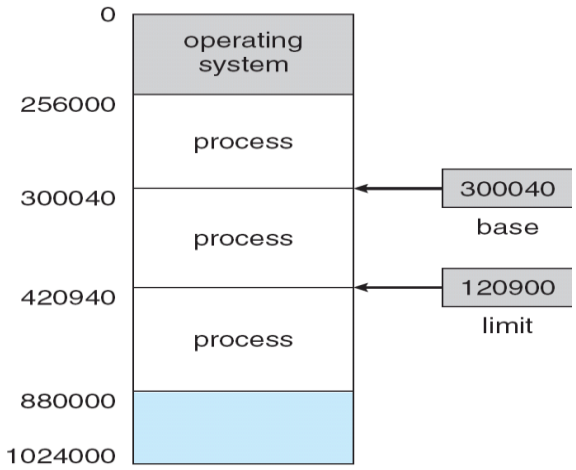
# CHAPTER THREE

## 3.0. Memory management

Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each with its own address. A program resides on a disk as a binary executable file. The program must be brought into memory and placed within a process for it to be executed. Depending on the memory management in use the process may be moved between disk and memory during its execution. The collection of processes on the disk that are waiting to be brought into memory for execution forms the input queue. i.e. selected one of the process in the input queue and to load that process into memory.

- Program must be brought into memory and placed within a process for it to be executed.
- **Input Queue** - collection of processes on the disk that are waiting to be brought into memory for execution.
- Main memory and registers are only storage CPU can access directly.
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Memory needs to be allocated efficiently to pack as many processes into memory as possible.

We can provide protection by using two registers, usually a base and a limit, as shown in fig. the base register holds the smallest legal physical memory address; the limit register specifies the size of the range.



Figs a base and limit register define a logical address space.

The binding of instructions and data to memory addresses can be done at any step along the way:

1. **Compile time**: If it is known at compile time where the process will reside nine memories, then absolute code can be generated.
2. **Load time**: If it is not known at compile time where the process will reside in memory, then the compiler must generate **re-locatable** code.
3. **Execution time**: If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time.

## 3.1. Logical versus Physical Address Space

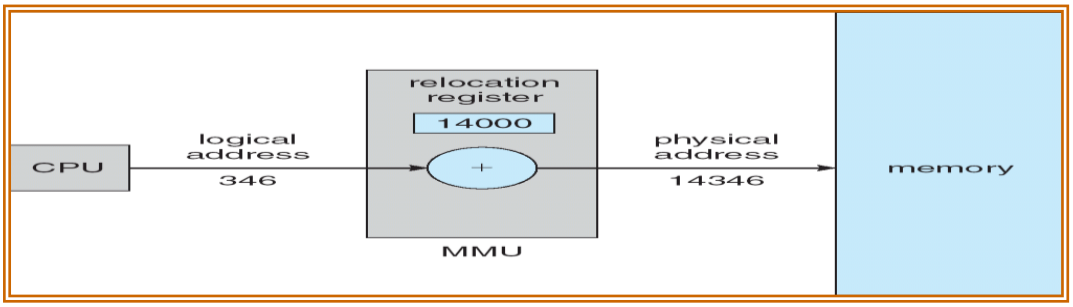
An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit is commonly referred to as a **physical address**.

- ☞ The compile-time and load-time address-binding schemes result in an environment where the logical and physical addresses are the same.
- ☞ The execution-time address-binding scheme results in an environment where the logical and physical addresses differ. In this case, we usually refer to the logical address as a virtual address.
- ☞ The set of all logical addresses generated by a program is referred to as a logical address space
- ☞ The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

The run-time mapping from virtual to physical addresses is done by the memory management unit (MMU), which is a hardware device.

- The base register is called a relocation register. The value in the relocation register is added to every address generated by a user process at the time it is sent to memory. For example, if the base is at 13000, then an attempt by the user to address location 0 dynamically relocated to location 14,000; an access to location 347 is mapped to location 13347. The MS-DOS operating system running on the Intel 80x 86 families of processors uses four relocation registers when loading and running processes.
- The user program never sees the real physical addresses. The program can create a pointer to location 347 store it memory, manipulate it, compare it to other addresses all as the number 347.
- The user program deals with logical addresses. The memory-mapping hardware converts logical addresses into physical addressed Logical addresses (in the range 0 to max) and physical addresses (in the range R + 0 to R + max for a base value R). The user generates only logical addresses.

The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.




## 3.2.Swapping

A process, can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.

- **Backing Store** - fast disk large enough to accommodate copies of all memory images for all users, must provide direct access to these memory images.
- **Roll out, roll in** - swapping variation used for priority based scheduling algorithms, lower priority process is swapped out, and so higher priority process can be loaded and executed.

- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped (**might be?).**
- In swapping
    - Context time must be taken into consideration.
    - Swapping a pending process for an IO needs care.
- Modified versions of swapping are found on many systems, i.e. UNIX and Microsoft Windows.
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk.

Give possible solutionAssume a multiprogramming environment with a round robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed ( Fig).

When each process finishes its quantum, it will be swapped with another process.

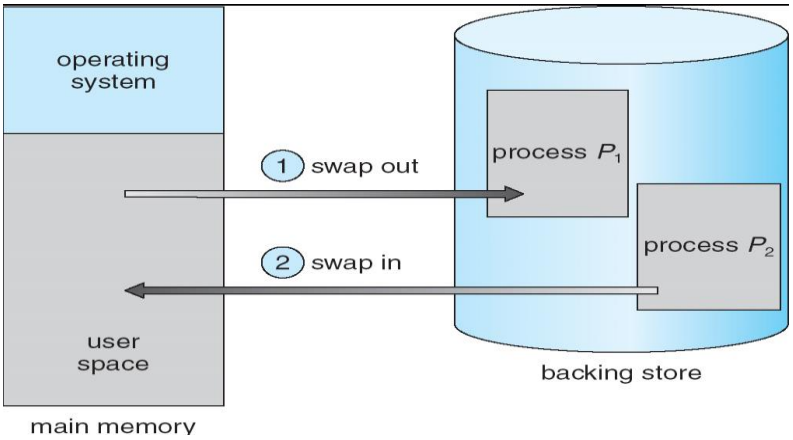



Fig Swapping of two processes using a disk as a blocking store

A variant of this swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process so that it can load and execute the higher priority process. When the higher priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is sometimes called rollout, roll in.

A process is swapped out will be swapped back into the same memory space that it occupies previously. If binding is done at assembly or load time, then the process cannot be moved to different location. If execution-time binding is being used, then it is possible to swap a process into a different memory space.

Swapping requires a backing store. The backing store is commonly a fast disk. It is large enough to accommodate copies of all memory images for all users. The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.

The context-switch time in such a swapping system is fairly high. Let us assume that the user process is of size 100K and the backing store is a standard hard disk with transfer rate of 1 megabyte per second. The actual transfer of the 100K process to or from memory takes

100K / 1000K per second = 1/10 second = 100 milliseconds

### 3.2.1.Overlay

Replacement of a block of stored instructions or data with another. Keep in memory only those instructions and data that are needed at any given time.

- ✓ Needed when process is larger than amount of memory allocated to it.
- ✓ Implemented by user, no special support from OS;
- ✓ Programming design of overlay structure is complex.

## 3.3.Contiguous Allocation

The main memory must accommodate both the **operating system** and the various **user processes**. The memory is usually divided into two partitions, one for the ***resident*** operating system, and one for the ***user processes.***

- Resident Operating System, usually held in low memory with interrupt vector.
- User processes then held in high memory.

### 3.3.1.Single-Partition Allocation

If the operating system is residing in low memory, and the user processes are executing in high memory. And operating-system code and data are protected from changes by the user processes. We also need protect the user processes from one another. We can provide this 2 protection by using **relocation registers**.

The **relocatio**n register contains the value of the smallest physical address; the limit register contains the range of logical addresses (for example, relocation = 100,040 and limit = 74,600). With relocation and limit registers, each logical address must be less than the limit

register; the MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory.

The relocation-register scheme provides an effective way to allow the operating system size to change dynamically.

### 3.3.2. Multiple-Partition Allocation

One of the simplest schemes for memory allocation is to divide memory into a number of fixed-sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. When a partition is free, a process is selected from the **input queue** and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

The operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes, and is considered as one large block, of available memory, a hole. When a process arrives and needs memory, we search for hole large enough for this process.

For example, assume that we have 2560K of memory available and a resident operating system of 400K. This situation leaves 2160K for user processes. FCFS job scheduling, we can immediately allocate memory to processes P1, P2, P3. Holes size 260K that cannot be used by any of the remaining processes in the input queue. Using a round-robin CPU-scheduling with a quantum of 1 time unit, process will terminate at time 14, releasing its memory.

Memory allocation is done using Round-Robin Sequence as shown in fig. When a process arrives and needs memory, we search this set for a hole that is large enough for this process. If the hole is too large, it is split into two: One part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hole is adjacent to other holes, we merge these adjacent holes to form one larger hole.

This procedure is a particular instance of the general dynamic storage-allocation problem, which is how to satisfy a request of size n from a list of free holes. There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate, first-fit, best-fit, and worst-fit are the most common strategies used to **select a free hole** from the set of available holes.

**First-fit**: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

**Best-fit:** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is kept ordered by size. This strategy-produces the smallest leftover hole.

**Worst-fit**: Allocate the largest hole. Again, we must search the entire list unless it is sorted by size. This strategy produces the largest leftover hole which may be more useful than the smaller leftover hole from a best-t approach

### 3.3.3.Fixed Partitioning

- **Equal-size partitions**

Any process whose size is less than or equal to the partition size can be loaded into an available partition. If all partitions are full, the operating system can swap a process out of a partition. a program may not fit in a partition. The programmer must design the program with overlays

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called **internal fragmentation**. Because all partitions are of equal size, it does not matter which partition is used.

**Unequal-size partitions**

- can assign each process to the smallest partition within which it will fit
- queue for each partition
- Processes are assigned in such a way as to minimize wasted memory within a partition.

**Dynamic Partitioning**

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required. Eventually get holes in the memory. This is called **external fragmentation**
- Must use compaction to shift processes so they are contiguous and all free memory is in one block
  - ✓ Example: 2560K of memory available and a resident OS of 400K. Allocate memory to processes P1…P4 following FCFS.
  - ✓ **Shaded regions are holes**

✓ Initially P1, P2, P3 create first memory map.

| Process | Memory | Time |
|---|---|---|
| P1 | 600K | 10 |
| P2 | 1000K | 5 |
| P3 | 300K | 20 |
| P4 | 700K | 8 |
| P5 | 500K | 15 |

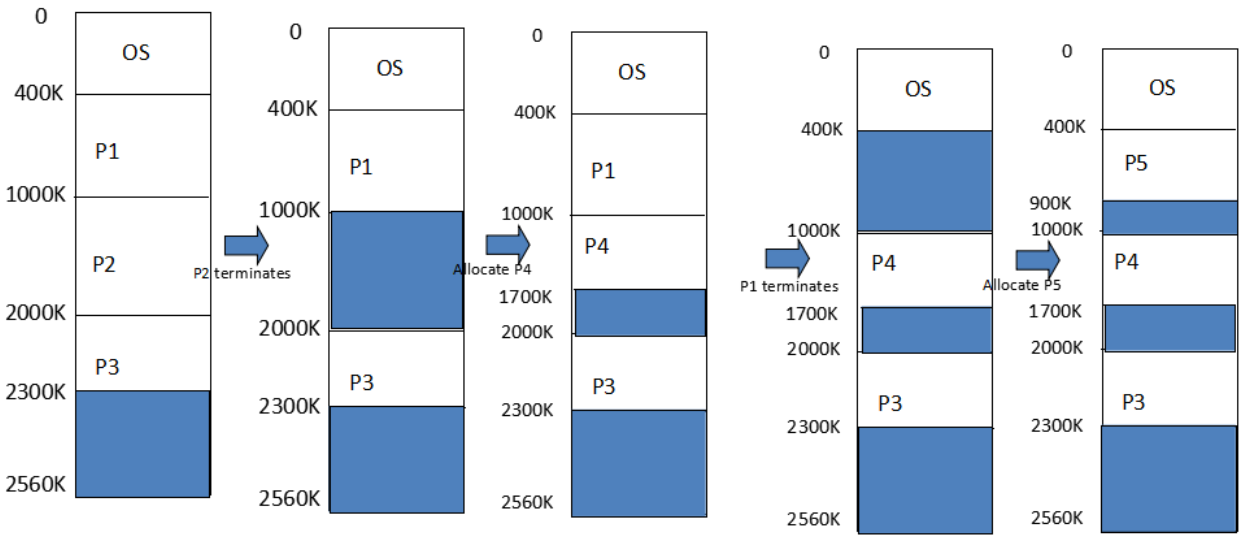



Fig: Contiguous Allocation

## 3.4.Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
  - 50 % rule: Given N allocated blocks 0.5 blocks will be lost due to fragmentation.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

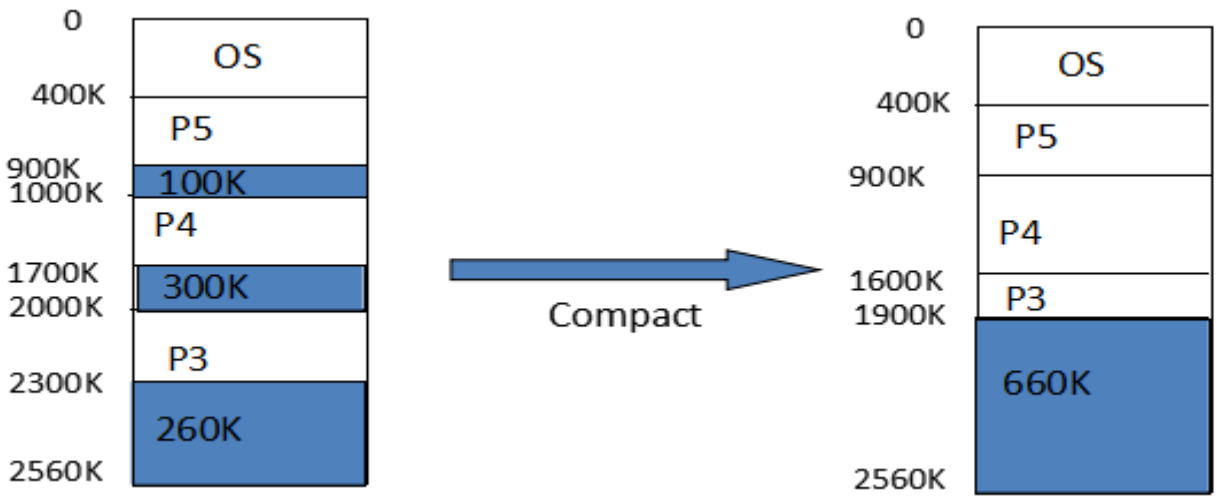
- Consider the hole of 18,464 bytes and process requires 18462 bytes.
- If we allocate exactly the required block, **we are left with a hole of 2 bytes.**
- The overhead to keep track of this free partition will be substantially larger than the whole itself.
- **Solution**: allocate very small free partition as a part of the larger request.

**Solution to fragmentation**

1. **Compaction**
2. **Paging**
3. **Segmentation**

**1. Compaction:** Reduce external fragmentation by compaction

- Shuffle memory contents to place all free memory together in one large block.
- Compaction is possible *only* if relocation is dynamic, and is done at execution time.
- **I/O problem**
  - Latch job in memory while it is involved in I/O.
  - Do I/O only into OS buffers.
- Compaction depends on cost.




## 3.5. Paging

☞ **Basic Idea:** logical address space of a process can be made noncontiguous; process is allocated physical memory wherever it is available.

- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses

**Address Translation Scheme**

- **Address generated by CPU is divided into:**
  - ***Page number (p)*** – used as an index into a *page table* which contains base address of each page in physical memory.
  - ***Page offset (d)*** – combined with base address to define the physical memory address that is sent to the memory unit.

- Page number is an index to the page table.
- The page table contains base address of each page in physical memory.
- The base address is combined with the page offset to define the physical address that is sent to the memory unit.
- The size of a page is typically a power of 2.
  - 512 –8192 bytes per page.
- The size of logical address space is $2^m$ and page size is $2^n$ address units.
- Higher m-n bits designate the page number
- lower order bits indicate the page offset.

***Address Translation Architecture***

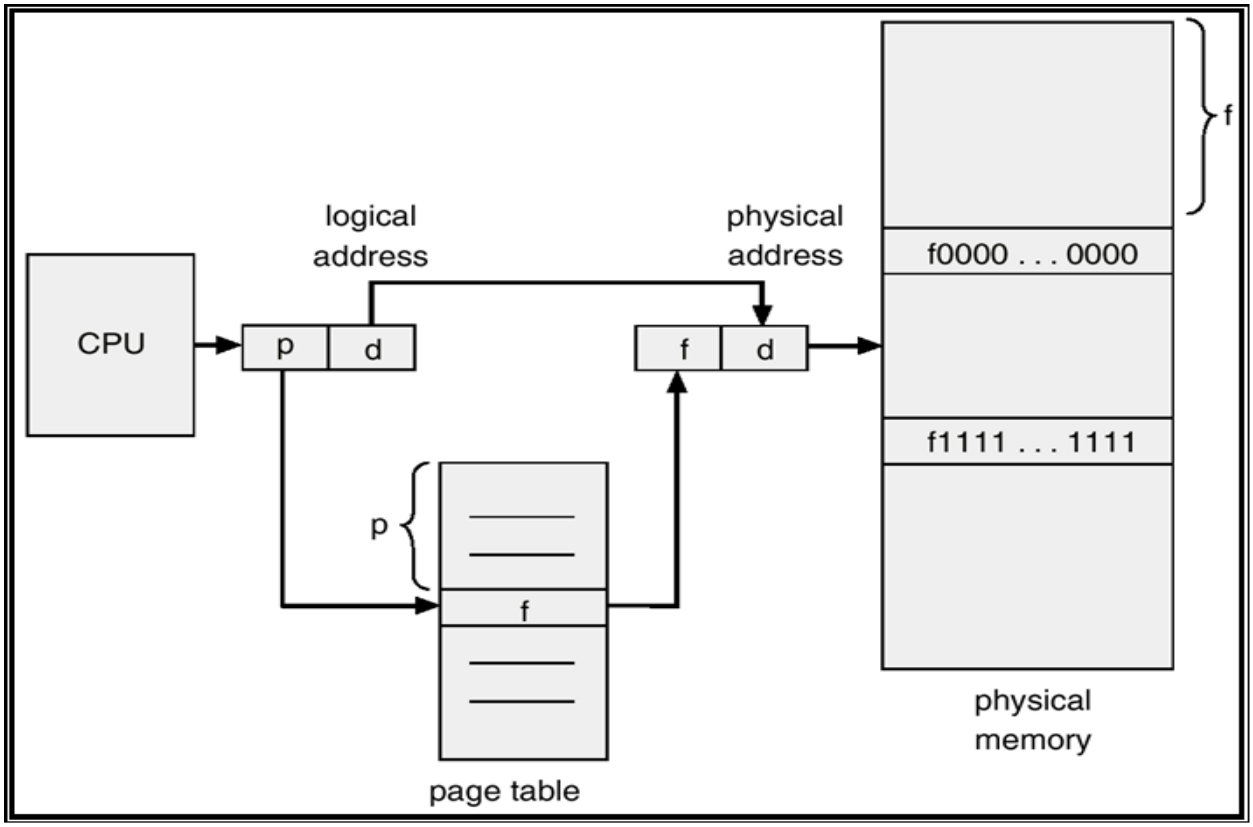
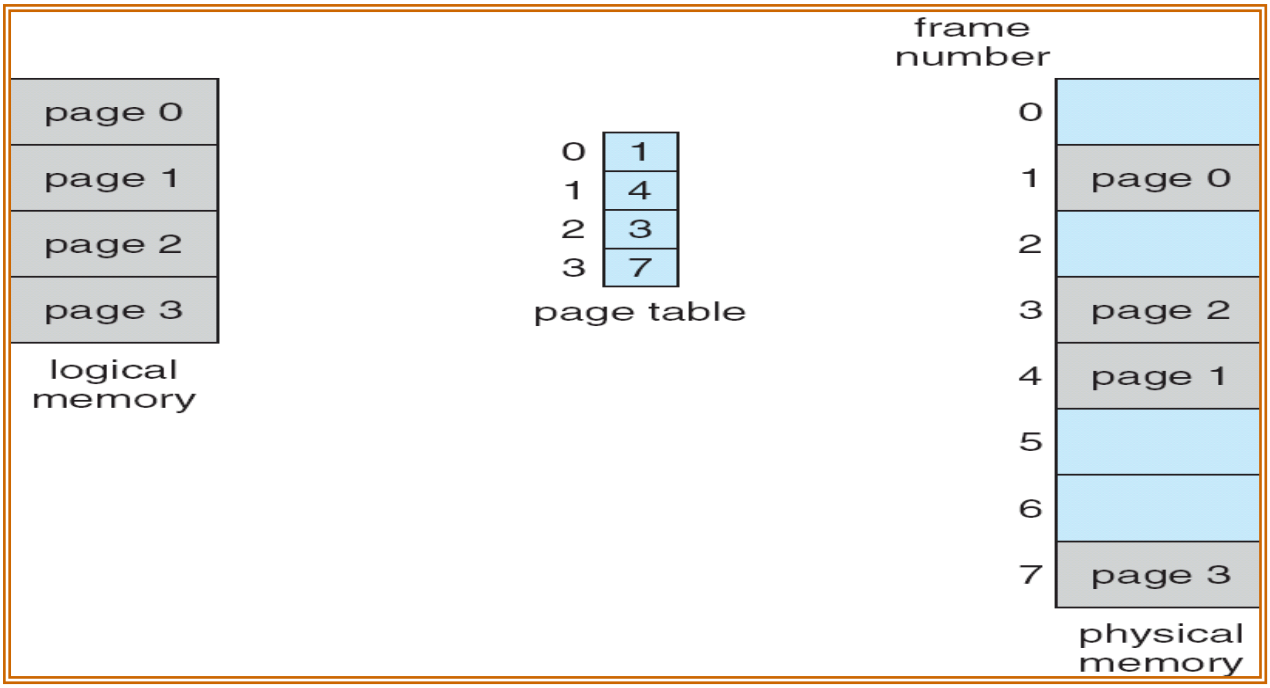



Fig Paging Hardware

The page size like is defined by the hardware. The size of a page is typically a power of 2 varying between 512 bytes and 8192 bytes per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset. lf the size of logical address space is 2m, and a page size is 2n addressing units (bytes or words), then the high-order m - n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:

**Example of Paging**




- ☞ Page size= 4 bytes; Physical memory=32 bytes (8 pages)
- ☞ Logical address 0 maps 1*4+0=4
- ☞ Logical address 3 maps to= 1*4+3=7
- ☞ Logical address 4 maps to =4*4+0=16
- ☞ Logical address 13 maps to= 7*4+1=29.

**Assume:-**

- page size=4 bytes
- Physical memory = 32 bytes (8 pages).

***How a logical memory address can be mapped into physical memory?***

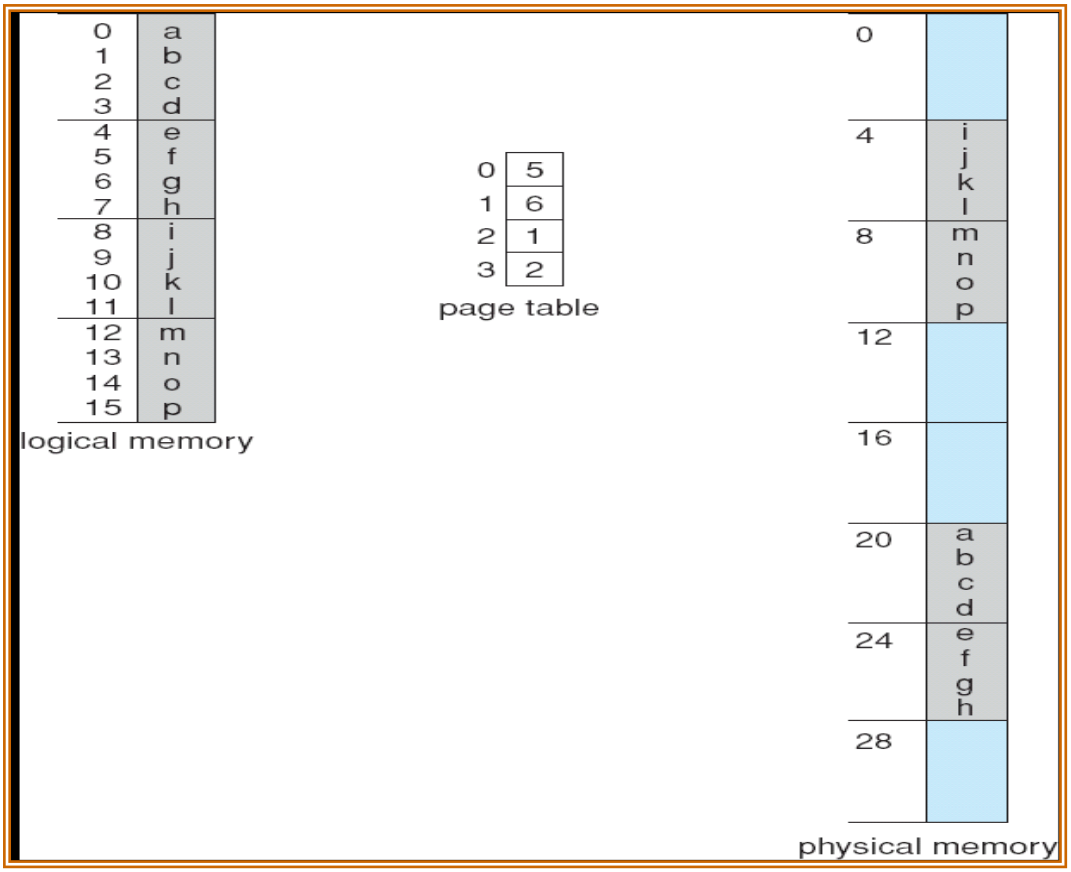
- ✓ Logical address 0(containing '**a**')
  - i. is on page 0.
  - ii. Is at offset 0.
- ✓ Indexing into the page table, you can that page 0 is in frame 5.

➔Logical address 0 is mapped to ***physical 20***, i.e.

- ✓ 20=[(5x4)+0]

Similarly,

a. Logical address 13 (page 3,offset 1) mapped ➔physical address 9=((2x4)+1).




- ✓ Logical address 0 maps 5*4+0=**20**
- ✓ Logical address 3 maps to= 5*4+3=**23**
- ✓ Logical address 4 maps to =6*4+0=**24**

Logical address 13 maps to= 2*4+1=**9**

## 3.6. Segmentation

A user program can be subdivided using segmentation, in which the program and its associated data are divided into a number of segments. It is not required that all segments of all programs be of the same length, although there is a maximum segment length. As with paging, a logical address using segmentation consists of two parts, in this case a **segment number** and an **offset.**

Because of the use of unequal-size segments, segmentation is similar to dynamic partitioning. In segmentation, a program may occupy more than one partition, and these partitions need not be contiguous. Segmentation eliminates internal fragmentation but, like dynamic partitioning, it suffers from external fragmentation. However, because a process is broken up into a number of smaller pieces, the external fragmentation should be less. Whereas paging is

invisible to the programmer, segmentation usually visible and is provided as a convenience for organizing programs and data.

Another consequence of unequal-size segments is that there is no simple relationship between logical addresses and physical addresses. Segmentation scheme would make use of a segment table for each process and a list of free blocks of main memory. Each segment table entry would have to as in paging give the starting address in main memory of the corresponding segment. The entry should also provide the length of the segment, to assure that invalid addresses are not used. When a process enters the Running state, the address of its segment table is loaded into a special register used by the memory management hardware.

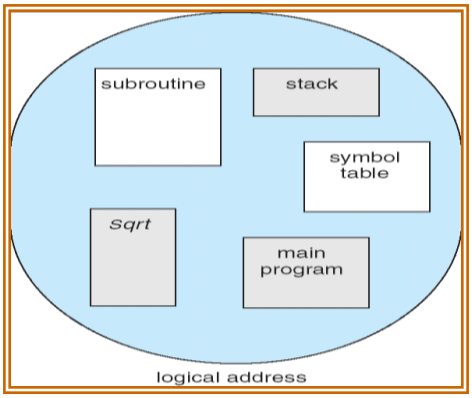
Consider an address of n + m bits, where the leftmost n bits are the segment number and the rightmost m bits are the offset. The following steps are needed for address translation: Extract the segment number as the leftmost n bits of the logical address.

Use the segment number as an index into the process segment table to find the starting physical address of the segment. Compare the offset, expressed in the rightmost m bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid.

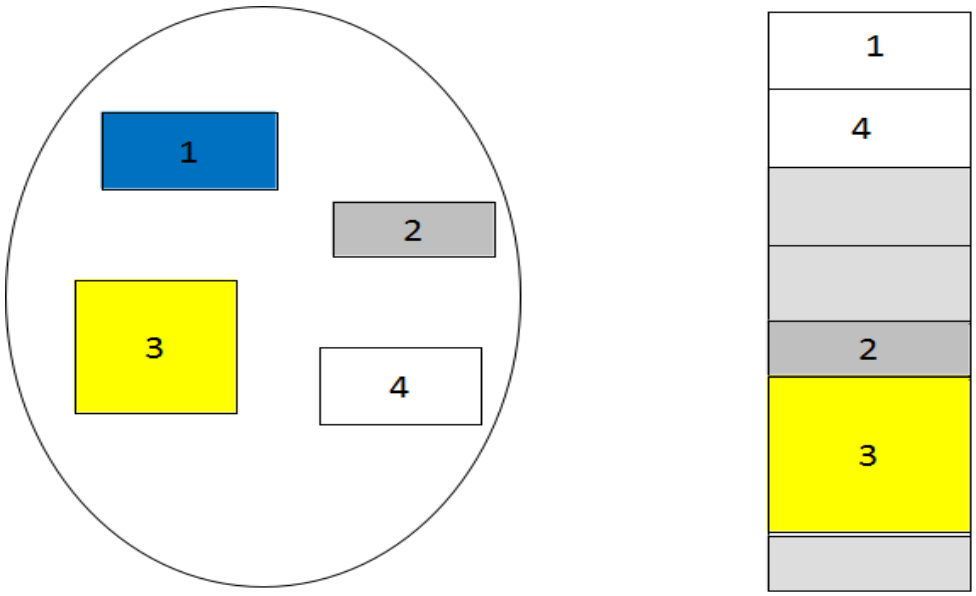
The desired physical address is the sum of the starting physical address of the segment plus the offset. Segmentation and paging can be combined to have a good result.

- Memory Management Scheme that supports **user view of memory.**
- A program is a collection of segments.
- A segment is a logical unit such as
  - main program, procedure, function
  - local variables, global variables, common block
  - stack, symbol table, arrays and so on.
  - Protect each entity independently
  - Allow each segment to grow independently
  - Share each segment independently

**User's View of a Program**




**Logical View of Segmentation**




### 3.6.1. Segmentation Architecture

- **Logical address consists of a two tuple**
  - ✓ **<segment-number, offset>**
- **Segment Table**
  - ✓ **Maps two-dimensional user-defined addresses into one-dimensional physical addresses.**
  - ✓ **Each table entry has**
    - **Base - contains the starting physical address where the segments reside in memory.**
    - **Limit - specifies the length of the segment.**

- ✓ ***Segment-table base register*** **(STBR) points to the segment table's location in memory.**
- ✓ ***Segment-table length register*** **(STLR) indicates the number of segments used by a program;.**
- ✓ **Note:** ***segment number s is legal if s < STLR***.

- ☞ **Relocation is dynamic** - by segment table
- ☞ **Sharing**
  - ― Code sharing occurs at the segment level.
  - ― Shared segments must have same segment number.
- ☞ **Allocation** - dynamic storage allocation problem
  - ― Use best fit/first fit, may cause external fragmentation.
- ☞ **Protection**
  - ― protection bits associated with segments
    - – read/write/execute privileges
    - – Array in a separate segment - hardware can check for illegal array indexes.

# CHAPTER FOUR

# Device management

## 4.1. Characteristics of parallel and serial devices

### Buffering

A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons.

One reason is to cope with a speed mismatch between the producer and consumer of a data stream.

Second buffer while the first buffer is written to disk. A second use of buffering is to adapt between devices that have different data transfer sizes.

A third use of buffering is to support copy semantics for application I/O.

### Caching

A cache is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.

Caching and buffering are two distinct functions, but sometimes a region of memory can be used for both purposes.

## 4.2. Direct memory access

### Spooling and Device Reservation

A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time.

In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.

Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors.

**Device drivers**

In computing, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device.

A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device.

Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific.

They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

## 4.3. Recovery from failure

**Backup and Restore**

Magnetic disks sometimes fail, and care must be taken to ensure that the data lost in such a failure are not lost forever. To this end, system programs can be used to backup data from disk to another storage device, such as a floppy disk, magnetic tape, optical disk, or other hard disk. Recovery from the loss of an individual file, or of an entire disk, may then be a matter of the data from backup.

To minimize the copying needed, we can use information from each file's directory entry. For instance, if the backup program knows when the last backup of a file was done, and the file's last write date in the directory indicates

that the file has not changed since that date, then the file does not need to be copied again. A typical backup schedule may then be as follows:

1. Copy to a backup medium all files from the disk. This is called full backup.
2. Copy to another medium all files changed since day 1. This is called incremental backup.
3. Copy to another medium all files changed since day 2.
4. Day *N*. Copy to another medium all files changed since day N-1. Then go back to Day 1.

# Chapter 5

# File Systems

## 5.1. Fundamental concepts on file

Computer store information in storage media such as disk, tape drives, and optical disks. The operating system provides a logical view of the information stored in the disk. This logical storage unit is a file. The information stored in files are non-volatile, means they are not lost during power failures. A file is named collection of related information that is stored on physical storage.Data cannot be written to a computer unless it is written to a file. A file, in general, is a sequence of bits, bytes, lines, or records defined by its owner or creator. The file has a structure defined by its owner or creator and depends on the file type.

- ✓ Text file – It has a sequence of characters.
- ✓ Image file – It has visual information such as photographs, vectors art and so on.
- ✓ Source file – It has subroutines and function that are compiled later.
- ✓ Object file – It has a sequence of bytes, organized into bytes and used by the linker.
- ✓ Executable file – The binary code that the loader brings to memory for execution is stored in an exe file.

### 5.1.1. File Attributes

A file has a single editable name given by its creator. The name never changes unless a user has necessary permission to change the file name. The names are given because it is humanly understandable.

The properties of a file can be different on many systems, however, some of them are common:

1. Name – unique name for a human to recognize the file.
2. Identifier – A unique number tag that identifies the file in the file system, and non-human readable.
3. Type – Information about the file to get support from the system.
4. Size – The current size of the file in bytes, kilobytes, or words.
5. Access Control – Defines who could read, write, execute, change, or delete the file in the system.
6. Time, Date, and User identification – This information kept for date created, last modified, or accessed and so on.

## 5.2. Operations, organization and buffering in file

**File Operations**

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. Let's examine what the operating system must do to perform each of these six basic file operations. It should then be easy to see how other, similar operations, such as renaming a file, can be implemented

**Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 11. Second, an entry for the new file must be made in the directory.

Writing a file. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs

**Reading a file.** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from writing to a file, the current operation location can. be kept as a per-process current file-position pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity

**Repositioning within a file.** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value, Repositioning writing a file need not involve any actual I/O. This file operation is also known as a file seek.

**Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

**Truncating a file.** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function a11o"vs all attributes to remain unchanged -except for file length-but lets the file be reset to length zero and its file space released.

A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referred by name. Files have unique file names and may be created and deleted. Restrictions on access control usually apply at the file level.

A file is a container for a collection of information. The file manager provides a protection mechanism to allow user's administrator how processes executing on behalf of different users can access the information in a file. File protection is a fundamental property of files because it allows different people to store their information on a shared computer.

File represents programs and data. Data files may be numeric, alphabetic, binary or alpha numeric. Files may be free form, such as text files. In general, file is sequence of bits, bytes, lines or records.

A file has a certain defined structure according to its type.

1. Text File
2. Source File
3. Executable File
4. Object File

## File Structure

Four terms are used for files

- Field
- Record
- Database

A field is the basic element of data. An individual field contains a single value. A record is a collection of related fields that can be treated as a unit by some application program. A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referenced by name. Files have file names and maybe created and deleted. Access control restrictions usually apply at the file level.

A database is a collection of related data. Database is designed for use by a number of different applications. A database may contain all of the information related to an organization or project, such as a business or a scientific study. The database itself consists of one or more types of files. Usually, there is a separate database management system that is independent of the operating system.

**File Types – Name, Extension**

A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts: a name and an extension. Following table gives the file type with usual extension and function.

| File Type | Usual Extension | Function |
|---|---|---|
| Executable | exe, com, bin | Read to run machine language program. |
| Object | obj, o | Compiled, machine language, not linked |
| Source Code | c, cc, java, pas asm, a | Source code in various language |
| Text | txt, doc | Textual data, documents |

## 5.3. File Management Systems

A file management system is that set of system software that provides services to users and applications in the use of files. Following objectives for a file management system:

• To meet the data management needs and requirements of the user which include storage of data and the ability to perform the aforementioned operations?

- ☞ To guarantee, to the extent possible, that the data in the file are valid.
- ☞ To optimize performance, both from the system point of view in terms of overall throughput.
- ☞ To provide I/O support for a variety of storage device types.

- To minimize or eliminate the potential for lost or destroyed data.
- To provide a standardized set of I/O interface routines to use processes.

TO provide I/O support for multiple users, in the case of multiple-user systems File System Architecture. At the lowest level, device drivers communicate directly with peripheral devices or their controllers or channels. A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request. For file operations, the typical devices controlled are disk and tape drives. Device drivers are usually considered to be part of the operating system.

The I/O control consists of device drivers and interrupt handlers to transfer information between the memory and the disk system. A device driver can be thought of as a translator. The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk.

The file-organization module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer. Each file's logical blocks are numbered from 0 (or 1) through N, whereas the physical blocks containing the data usually do not match the logical numbers, so a translation is needed to locate each block. The file-organization module also includes the free-space manager, which tracks unallocated and provides these blocks to the file organization module when requested.

The logical file system uses the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. The logical file system is also responsible for protection and security.

To create a new file, an application program calls the logical file system. The logical file system knows the format of the directory structures. To create a new file, it reads the appropriate directory into memory, updates it with the new entry, and writes it back to the disk.

Once the file is found the associated information such as size, owner, access permissions and data block locations are generally copied into a table in memory, referred to as the open-file fable, consisting of information about all the currently opened files.

The first reference to a file (normally an open) causes the directory structure to be searched and the directory entry for this file to be copied into the table of opened files. The index into this table is returned to the user program, and all further references are made through the index rather than with a symbolic name. The name given to the index varies. UNIX systems

refer to it as a file descriptor, Windows/NT as a file handle, and other systems as a file control block.

Consequently, as long as the file is not closed, all file operations are done on the open-file table. When the file is closed by all users that have opened it, the updated file information is copied back to the disk-based directory structure.

# Chapter 6

# Security

## 6.1. Overview of system security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

Operating system security (OS security) is the process of ensuring OS integrity, confidentiality and availability. OS security refers to specified steps or measures used to protect the OS from threats, viruses, worms, malware or remote hacker intrusions. OS security encompasses all preventive-control techniques, which safeguard any computer assets capable of being stolen, edited or deleted if OS security is compromised.

**The Security Problem**

- Security must consider external environment of the system, and protect it from:
  - ✓ Unauthorized access.
  - ✓ malicious modification or destruction
  - ✓ Accidental introduction of inconsistency.
  - ✓ These are management, rather than system, problems.
- Easier to protect against accidental than malicious misuse.
- We say that the system is secure if its resources are used and accessed as intended under all circumstances.
- Unfortunately total security cannot be achieved.
- It is easier to protect against accidental misuse than malicious misuse.
- Intruder or cracker: attempt to breach the security
- Threat: potential of security violation such as discovery of vulnerability

- ♠ Attack: an attempt to break security
- ♠ Form of malicious access
  - ✓ Breach of confidentiality: Unauthorized reading of data (theft of information)
  - ✓ Breach of integrity: Unauthorized modification of data
  - ✓ Breach of availability: Unauthorized destruction of data
  - ✓ Theft of service: Unauthorized use of resources.
  - ✓ Denial of service: Preventing legitimate use of the system (denial of service)
- ✓ Absolute protection against malicious use is not possible; however cost of perpetrator can be sufficiently high to deter unauthorized attempts.

### 6.1.1. Policies and mechanism of system security

A security policy is a statement of what is, and what is not, allowed which reflects an organization's strategy to authorize access to the computer's resources

- ➢ Managers have access to personnel files
- ➢ OS processes have access to the page table

A security mechanism is a method, tool, or procedure for enforcing a security policy.

Authentication mechanisms are the basis of most protection mechanisms which are

1. External Authentication
   - ➢ „ User/process authentication
   - ➢ Authentication in networks
2. Internal Authentication
   - ➢ Sharing parameters
   - ➢ Confinement
   - ➢ Allocating rights
   - ➢ Trojan horse

## 6.2. System protection, authentication

System Protection is especially important in a multiuser environment when multiple users use computer resources such as CPU, memory, etc. It is the operating system's responsibility to offer a mechanism that protects each process from other processes. In a multiuser environment, all assets that require protection are classified as objects, and those that wish to access these objects are referred to as subjects. The operating system grants different access rights to different subjects.

Protection in operating system is mechanism that controls the access of programs, processes, or users to the resources defined by a computer system are referred to as protection. You may utilize protection as a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files. It needs the protection of computer resources like the software, memory, processor, etc. Users should take protective measures as a helper to multiprogramming OS so that multiple users may safely use a common logical namespace like a directory or data. Protection may be achieved by maintaining confidentiality, honesty and availability in the OS. It is critical to secure the device from unauthorized access, viruses, worms, and other malware.

### 6.2.1. Models of protection

Active parts (e.g., processes or threads) are called subjects and act on behalf of users. „
Passive parts (i.e., resources) are called objects. „
The particular set of rights a process has at any given time is referred to as its protection domain. „ A subject is a process executing in a specific protection domain. „
A protection system is composed of a set of objects, a set of subjects, and a set of rules specifying the protection policy. „
What mechanism to implement different security policies for subjects to access objects

- ✓ Many different policies must be possible
- ✓ Policy may change over time

### 6.2.2. Memory protection

One of the important aspects of Operating system security is Memory Protection. Memory provides powerful indirect way for an attacker to circumvent security mechanism, since every piece of information accessed by any program will need to reside in memory at some point in time, and hence may potentially be accessed in the absence of memory protection mechanisms. Memory protection is a way for controlling memory usage on a computer, and is core to virtually every operating system. The main purpose of memory protection is to prevent a process running on an operating system from accessing the memory of other processes, or is used by the OS kernel. This prevents a bug within the process from affecting other processes, and also prevents malicious software from gaining unauthorized access to the system, e.g., suppose that process A is permitted access to a file F, while process B is not. Process B can bypass this policy by attempting to read F's content that will be stored in A's

memory immediately after A reads F. Alternatively, B may attempt to modify the access control policy that is stored in the OS memory so that the OS thinks that B is permitted access to this file.

### 6.2.3. Encryption

**Encryption**: Encrypt clear text into cipher text. Properties of good encryption technique:

- ☞ Relatively simple for authorized users to encrypt and decrypt data.
- ☞ Encryption scheme depends not on the secrecy of the algorithm but on a parameter of the algorithm called the encryption key.
- ☞ Extremely difficult for an intruder to determine the encryption key.
  - ☞ *Data Encryption Standard* substitutes characters and rearranges their order on the basis of an encryption key provided to authorize users via a secure mechanism. Scheme only as secure as the mechanism.

**RSA: is** one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret.

### 6.2.3. Recovery management

**Four levels of security measures must be taken**.

- ✓ Physical
  - ❖ Against armed or surreptitious entry by intruders.
- ✓ Human
  - ❖ Careful screening of users to reduce the chance of unauthorized access.
- ✓ Network
  - ❖ No one should intercept the data on the network.
- ✓ Operating system
  - ❖ The system must protect itself from accidental or purposeful security beaches.

- ❖ A weakness at a high level of security allows circumvention of low-level measures.
- ❖ In the remainder of this chapter we discuss security at OS level

**Security measures at OS level**

- **User authentication**: Verifying the user's authentication
- **Program threats**: Misuse of programs unexpected misuse of programs.
- **System threats**: Worms and viruses
- **Intrusion detection**: Detect attempted intrusions or successful intrusions and initiate appropriate responses to the intrusions.
- **Cryptography**: Ensuring protection of data over network
- **Authentication:**User identity most often established through *passwords* can be considered a special case of either keys or capabilities. Easy to understand and use
  - they can be easily guessed, illegally transferred
  - Visual or electronic monitoring, network sniffing
  - Passwords must be kept secret.
  - Frequent change of passwords.
  - Use of "non-guessable" passwords.
  - Log all invalid access attempts.
  - Passwords may also either be encrypted or allowed to be used only once.
  - For given a value of x, f(x) is calculated and stored. It is difficult to guess x by seeing f(x).
- **One time passwords:** Password is different for each session.
  Example: My mother's name is K…: Password "MmnisK"
- **Biometrics:** Palm and hand readers: temperature map, finger length, finger width and line patterns.
  Example: Fingerprint readers.

***Two factor authentication scheme***

- Password plus fingerprint scan

## Program Threats

1. **Trojan horse:** Code segment that misuses its environment. Exploits mechanisms for allowing programs written by users to be executed by other users. Consider the use of "." character in a search path. The "." Tells the shell to include the current directory. If the user sets current directory to a friend's directory, the program runs in users domain and effects friend's directory.
2. **Trap Door**
   - The designer of the code might leave a hole in the software that only she is capable of using.

- ☞ Specific user identifier or password that circumvents normal security procedures.
- ☞ Could be included in a compiler.
- ☞ Stack and Buffer Overflow
- ☞ Exploits a bug in a program (overflow either the stack or memory buffers.)

3. **Stack and Buffer Overflow:** Exploits a bug in a program (overflow either the stack or memory buffers.)The attacker determines the vulnerability and writes a program to do the following. Overflow an input-field, command-line argument, or input buffer until it writes into the stack. Overwrite the current return address on the stack with the address of the exploit code in the next step. Write a simple set of code for the next space in the stack that includes commands that the attacker wishes to execute, for example, spwan a shell.

**System Threats**

1. **Worms** – use spawn mechanism; standalone program. The worm spawns copies of itself, using up systems resources and perhaps locking out system use by all other processes.
2. **Internet worm:** Exploited UNIX networking features (remote access) and bugs in *finger* and *send mail* programs. Grappling hook program uploaded main worm program.
3. **Viruses** – fragment of code embedded in legitimate program.Mainly effect microcomputer systems. Downloading viral programs from public bulletin boards or exchanging floppy disks containing an infection. MSWORD (micros) ,RTF format is safe and Safe computing.
4. **Denial of Service**
   - ☞ Overload the targeted computer preventing it from doing any useful work.
   - ☞ Downloading of a page.
   - ☞ Partially started TCP/IP sessions could eat up all resources.
   - ☞ Difficult to prevent denial of service attacks.

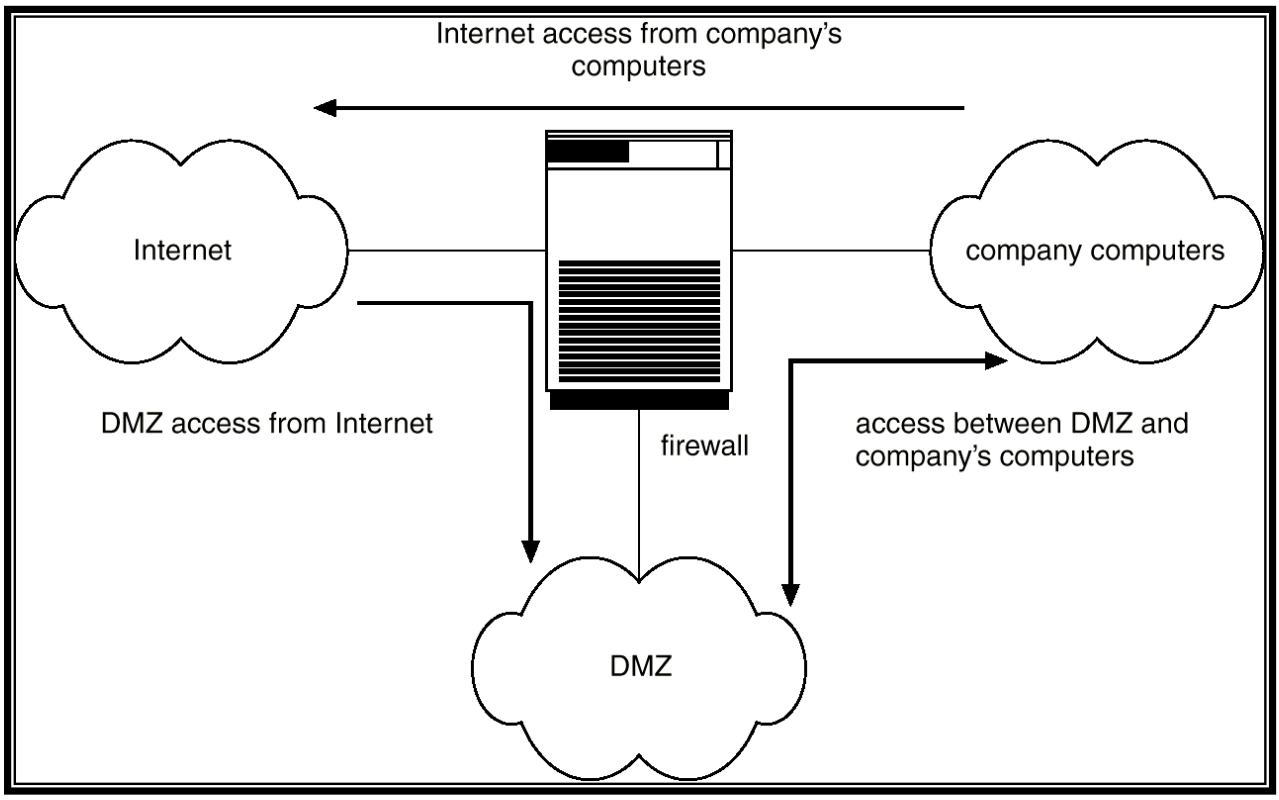
## Threat Monitoring

- ➢ Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.

- Audit log – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- Scan the system periodically for security holes; done when the computer is relatively unused.
- Check for:
  - Short or easy-to-guess passwords
  - Unauthorized set-uid programs
  - Unauthorized programs in system directories
  - Unexpected long-running processes
  - Improper directory protections
  - Improper protections on system data files
  - Dangerous entries in the program search path (Trojan horse)
  - Changes to system programs: monitor checksum values

**Firewall:** is placed between trusted and untrusted hosts. A firewall is a computer or router that sits between trusted and untrusted systems. It monitors and logs all connections. **It** limits network access between these two security domains.

**Spoofing:** An unauthorized host pretends to be an authorized host by meeting some authorization criterion.




**Intrusion Detection:**

- Detect attempts to intrude into computer systems.
- Wide variety of techniques
  - The time of detection
  - The type of inputs examined to detect intrusion activity
  - The range of response capabilities.

Alerting the administrator, killing the intrusion process, false resource is exposed to the attacker (but the resource appears to be real to the attacker) to gain more information about the attacker.The solutions are known as intrusion detection systems.

**Detection methods:**

- Auditing and logging.
- Install logging tool and analyze the external accesses.
- Tripwire (UNIX software that checks if certain files and directories have been altered – I.e. password files)
- Integrity checking tool for UNIX.
- It operates on the premise that a large class of intrusions results in anomalous modification of system directories and files.
- It first enumerates the directories and files to be monitored for changes and deletions or additions. Later it checks for modifications by comparing signatures.

**System call monitoring:** Detects when a process is deviating from expected system call behavior.

**Cryptography**: Eliminate the need to trust the network. Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key. Keys are designed to be computationally infeasible to derive from the messages.