Table of Contents

Overall

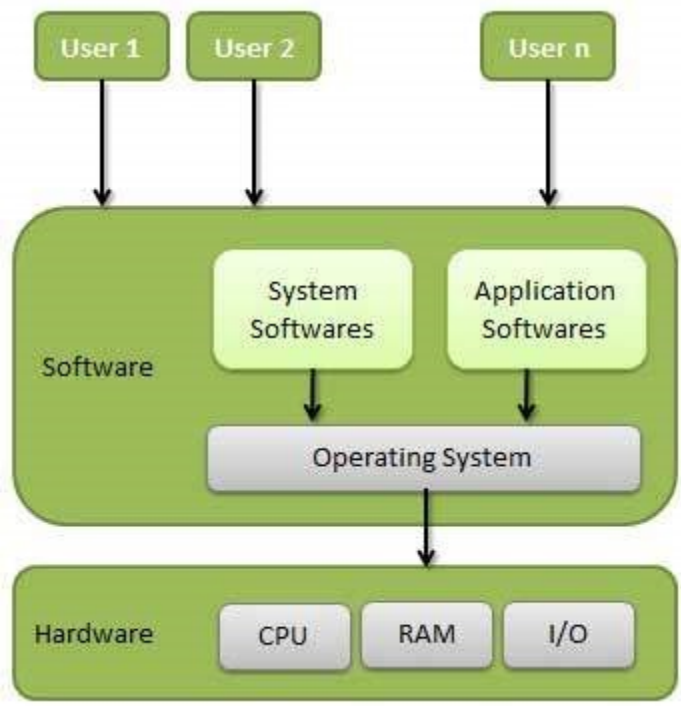
O1. What is OS?

Computer System Components

- Hardware – provides basic computing resources (CPU, memory, I/O devices).
- Operating system – controls and coordinates the use of the hardware among the
- various application programs for the various users.
- Applications programs – define the ways in which the system resources are used to
- solve the computing problems of the users (compilers, database systems, video games,
- business programs).
- Users (people, machines, other computers).

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- 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.



OS services

- Program Execution
- File system manipulation
- Input / Output Operations
- Communication
- Resource Allocation
- Error Detection
- Accounting
- Security and protection
- User Interface

O2. Booting

Bootstrap: is the program that initializes the operating system (OS) during startup
Boot ROM (contains BIOS)

- Check CMOS
- Interupt handlers and I/O controller
- Init registers and power management
- Check hardware (POST – Power on Self Test)
- Confirm computer can be started

Boot loader excecution

- Load remaning part of OS
- Login terminal

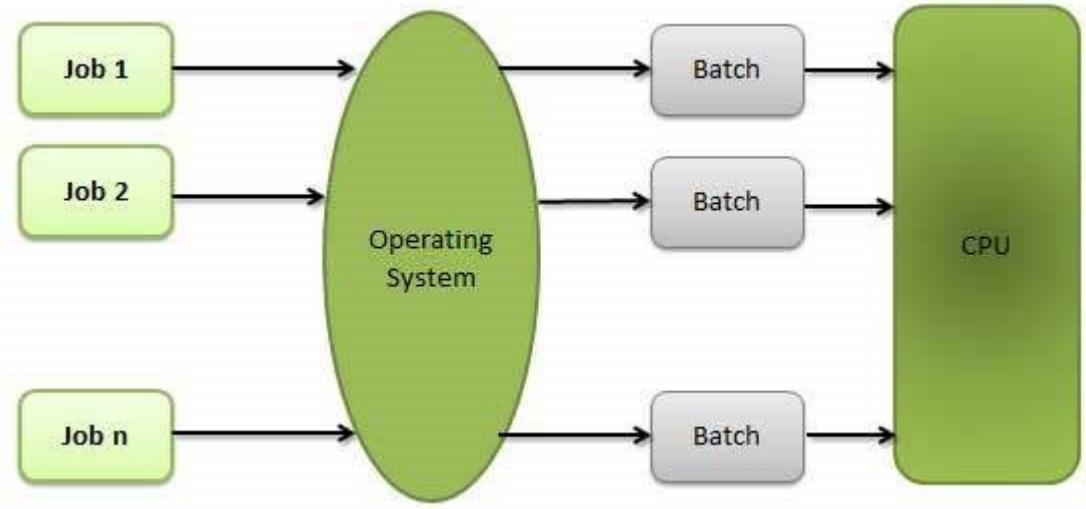
O3. OS zoo

Mainframe Systems

- Reduce setup time by batching similar jobs
- Automatic job sequencing – automatically transfers control from one job to another. First rudimentary operating system.
- Resident monitor
    - initial control in monitor
    - control transfers to job
    - when job completes control transfers pack to monitor

Multiprogrammed Batch Systems

Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



- Advantages
    - Batch processing takes much of the work of the operator to the computer.
    - Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention.
- Disadvantages
    - Difficult to debug program.
    - A job could enter an infinite loop.
    - Due to lack of protection scheme, one batch job can affect pending jobs.

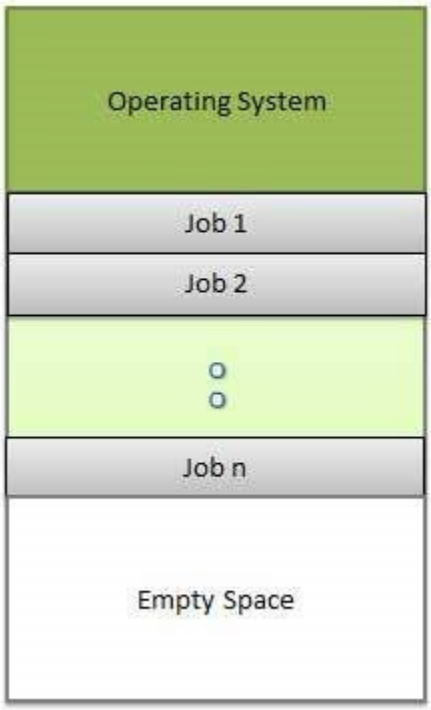
OS Features Needed for Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as multiprogramming. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

An OS does the following activities related to multiprogramming.

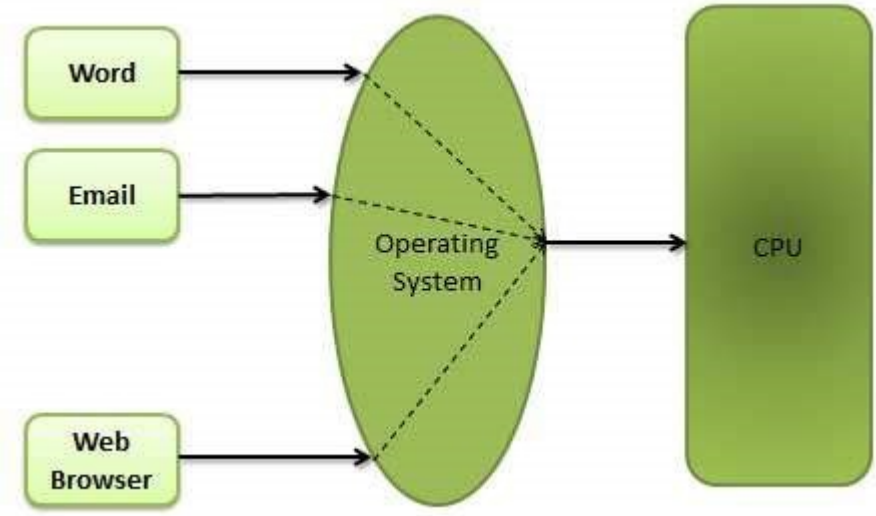
- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.

- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensures that the CPU is never idle, unless there are no jobs to process.



- Advantages
  - High and efficient CPU utilization.
  - User feels that many programs are allotted CPU almost simultaneously.
- Disadvantages
  - CPU scheduling is required.
  - To accommodate many jobs in memory, memory management is required.

Multitasking - Time-Sharing Systems



Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory
- A program that is loaded into memory and is executing is commonly referred to as a process.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Desktop Systems

- *Personal computers* – computer system dedicated to a single user.
- I/O devices – keyboards, mice, display screens, small printers.
- User convenience and responsiveness.
- Can adopt technology developed for larger operating system' often individuals have sole use of computer and do not need advanced CPU utilization of protection features.
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Parallel Systems

- Multiprocessor systems with more than on CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system:
    - Increased *throughput*
    - Economical
    - Increased reliability
        - graceful degradation
        - fail-soft systems
- *Symmetric multiprocessing (SMP)*
    - Each processor runs and identical copy of the operating system.
    - Many processes can run at once without performance deterioration.
    - Most modern operating systems support SMP
- *Asymmetric multiprocessing*
    - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
    - More common in extremely large systems

Distributed Systems

- Distribute the computation among several physical processors.
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
- Advantages of distributed systems.
    - Resources Sharing
    - Computation speed up – load sharing
    - Reliability
    - Communications
- Requires networking infrastructure.
- Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems.

Clustered Systems

- Clustering allows two or more systems to share storage.
- Provides high reliability.
- *Asymmetric clustering*: one server runs the application while other servers' standby.
- *Symmetric clustering*: all N hosts are running the application

Real-Time Systems

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.
- Real-Time systems may be either *hard* or *soft* real-time.
- Hard real-time:
    - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
    - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- Soft real-time
    - Limited utility in industrial control of robotics
    - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular telephones
- Issues:
    - Limited memory
    - Slow processors
    - Small display screens.

1. What are the advantages of multiprogramming?

- **Increased CPU Utilization** − Multiprogramming improves CPU utilization as it organizes a number of jobs where CPU always has one to execute.

- **Increased Throughput** − Throughput means total number of programs executed over a fixed period of time. In multiprogramming, CPU does not wait for I/O for the program it is executing, thus resulting in an increased throughput.
- **Shorter Turn around Time** − Turnaround time for short jobs is improved greatly in multiprogramming.
- **Improved Memory Utilization** − In multiprogramming, more than one program resides in main memory. Thus memory is optimally utilized.
- **Increased Resources Utilization** − In multiprogramming, multiple programs are actively competing for resources resulting in higher degree of resource utilization.
- **Multiple Users** − Multiprogramming supports multiple users.

2. What are the advantages of Multiprocessing or Parallel System?

Answer: Multiprocessing operating system or the parallel system support the use of more than one processor in close communication.

The advantages of the multiprocessing system are:

- **Increased Throughput** − By increasing the number of processors, more work can be completed in a unit time.
- **Cost Saving** − Parallel system shares the memory, buses, peripherals etc. Multiprocessor system thus saves money as compared to multiple single systems. Also, if a number of programs are to operate on the same data, it is cheaper to store that data on one single disk and shared by all processors instead of using many copies of the same data.
- **Increased Reliability** − In this system, as the workload is distributed among several processors which results in increased reliability. If one processor fails then its failure may slightly slow down the speed of the system but system will work smoothly.

3. What are the differences between Batch processing system and Real Time Processing System?

Sr. No.	Batch Processing System	Realtime Processing System
1	Jobs with similar requirements are batched together and run through the computer as a group.	In this system, events mostly external to computer system are accepted and processed within certain deadlines.

2	This system is particularly suited for applications such as Payroll, Forecasting, Statistical analysis etc.	This processing system is particularly suited for applications such as scientific experiments, Flight control, few military applications, Industrial control etc.
3	It provides most economical and simplest processing method for business applications.	Complex and costly processing requires unique hardware and software to handle complex operating system programs.
4	In this system data is collected for defined period of time and is processed in batches.	Supports random data input at random time.
5	In this system sorting is performed before processing.	No sorting is required.
6	It is measurement oriented.	It is action or event oriented.
7	Transactions are batch processed and periodically.	Transactions are processed as and when they occur.
8	In this processing there is no time limit.	It has to handle a process within the specified time limit otherwise the system fails.

4. What are the differences between Real Time System and Timesharing System?

Sr. No.	**Real Time System**	**Timesharing System**
1	In this system, events mostly external to computer system are accepted and processed within certain deadlines.	In this system, many users are allowed to simultaneously share the computer resources.
2	Real time processing is mainly devoted to one application.	Time sharing processing deals with many different applications.

3	User can make inquiry only and cannot write or modify programs.	Users can write and modify programs.
4	User must get a response within the specified time limit; otherwise it may result in a disaster.	User should get a response within fractions of seconds but if not, the results are not disastrous.
5	No context switching takes place in this system.	The CPU switches from one process to another as a time slice expires or a process terminates.

5. What are the differences between multiprocessing and multiprogramming?

Sr. No.	**Multiprocessing**	**Multiprogramming**
1	Multiprocessing refers to processing of multiple processes at same time by multiple CPUs.	Multiprogramming keeps several programs in main memory at the same time and execute them concurrently utilizing single CPU.
2	It utilizes multiple CPUs.	It utilizes single CPU.
3	It permits parallel processing.	Context switching takes place.
4	Less time taken to process the jobs.	More Time taken to process the jobs.
5	It facilitates much efficient utilization of devices of the computer system.	Less efficient than multiprocessing.
6	Usually more expensive.	Such systems are less expensive.

Process

P1. Process General

1. What is the difference between Job and Process?

Answer: A process refers to a program under execution. This program may be an application or system program.

Job means an application program and it is not a system program

2. What is process?

- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

Process State

- As a process executes, it changes *state*
  - **running**: Instructions are being executed.
  - **blocking**: The process is waiting for some event to occur.
  - **ready**: The process is waiting to be assigned to a process.

Process Control Block (PCB)

Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution

    - Parent and children execute concurrently.
    - Parent waits until children terminate.
- Address space
    - Child duplicate of parent.
    - Child has a program loaded into it.
- UNIX examples
    - **fork** system call creates new process
    - **exec** system call used after a **fork** to replace the process' memory space with a new program.

Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
    - Output data from child to parent (via **wait**).
    - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (**abort**).
    - Child has exceeded allocated resources.
    - Task assigned to child is no longer required.
    - Parent is exiting.
        - Operating system does not allow child to continue if its parent terminates.
        - Cascading termination

Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
    - Information sharing
    - Computation speed-up
    - Modularity
    - Convenience

P2. Scheduler

Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.
- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming*.
- Processes can be described as either:
- I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
- *CPU-bound process* – spends more time doing computations; few very long CPU bursts.

Process scheduling

- Keeps tracks of processor and status of process.
- The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

1. What is a process scheduler? State the characteristics of a good process scheduler?

Answer: Scheduling can be defined as a set of policies and mechanisms which controls the order in which the work to be done is completed. The scheduling program which is a system software concerned with scheduling is called the scheduler and the algorithm it uses is called the scheduling algorithm.

Various criteria or characteristics that help in designing a good scheduling algorithm are:

- **CPU Utilization** − A scheduling algorithm should be designed so that CPU remains busy as possible. It should make efficient use of CPU.
- **Throughput** − Throughput is the amount of work completed in a unit of time. In other words throughput is the processes executed to number of jobs completed in a unit of time. The scheduling algorithm must look to maximize the number of jobs processed per time unit.
- **Response time** − Response time is the time taken to start responding to the request. A scheduler must aim to minimize response time for interactive users.
- **Turnaround time** − Turnaround time refers to the time between the moment of submission of a job/ process and the time of its completion. Thus how long it takes to execute a process is also an important factor.
- **Waiting time** − It is the time a job waits for resource allocation when several jobs are competing in multiprogramming system. The aim is to minimize the waiting time.
- **Fairness** − A good scheduler should make sure that each process gets its fair share of the CPU.

2. Explain time slicing? How its duration affects the overall working of the system?

Answer: Time slicing is a scheduling mechanism/way used in time sharing systems. It is also termed as Round Robin scheduling. The aim of Round Robin scheduling or time slicing scheduling is to give all processes an equal opportunity to use CPU. In this type of scheduling, CPU time is divided into slices that are to be allocated to ready processes. Short processes may be executed within a single time quantum. Long processes may require several quanta

3. Priority in Linux

In Linux we can set guidelines for the CPU to follow when it is looking at all the tasks it has to do. These guidelines are called ***niceness*** or ***nice value***. The Linux niceness scale goes from -20 to 19. The lower the number the more priority that task gets. If the niceness value is high number like 19 the task will be set to the lowest priority and the CPU will process it whenever it gets a chance. The default nice value is zero.

4. What are the different principles which must be considered while selection of a scheduling algorithm?

Answer: The objective/principle which should be kept in view while selecting a scheduling policy are the following −

- **Fairness** − All processes should be treated the same. No process should suffer indefinite postponement.
- **Maximize throughput** − Attain maximum throughput. The largest possible number of processes per unit time should be serviced.
- **Predictability** − A given job should run in about the same predictable amount of time and at about the same cost irrespective of the load on the system.
- **Maximum resource usage** − The system resources should be kept busy. Indefinite postponement should be avoided by enforcing priorities.
- **Controlled Time** − There should be control over the different times:
    - Response time
    - Turnaround time
    - Waiting time

  The objective should be to minimize above mentioned times.

P3. Process scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms

- First-Come, First-Served (FCFS)
- Scheduling Shortest-Job-Next (SJN)
- Scheduling Priority
- Scheduling Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

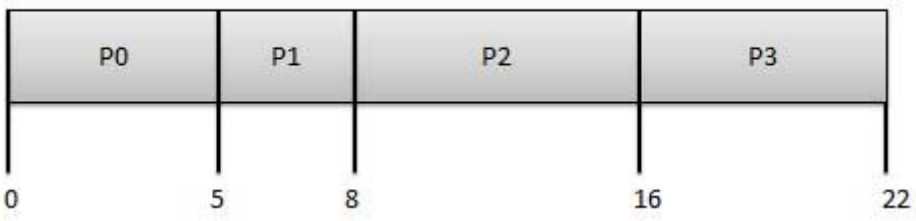
These algorithms are either non-preemptive or preemptive.

- Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time,
- Preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

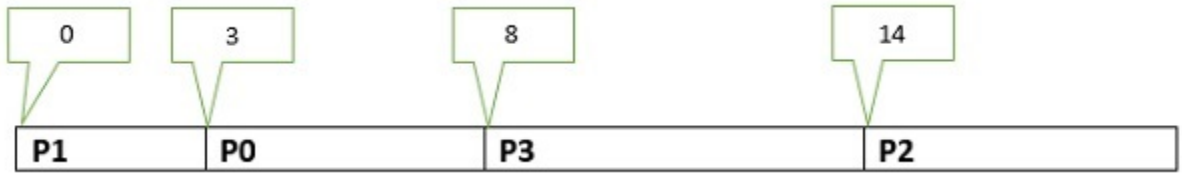
Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	14
P3	3	6	8

Gantt Chart



Wait time of each process is as follows −

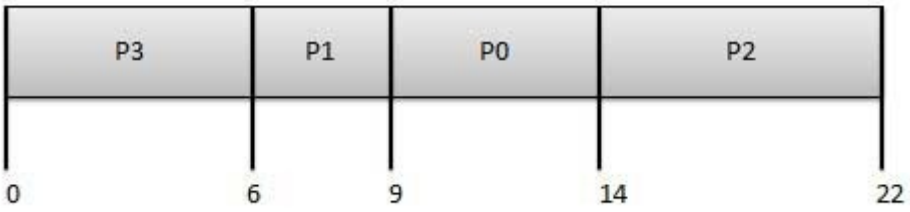
Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (3+12+5) / 4 = 5

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

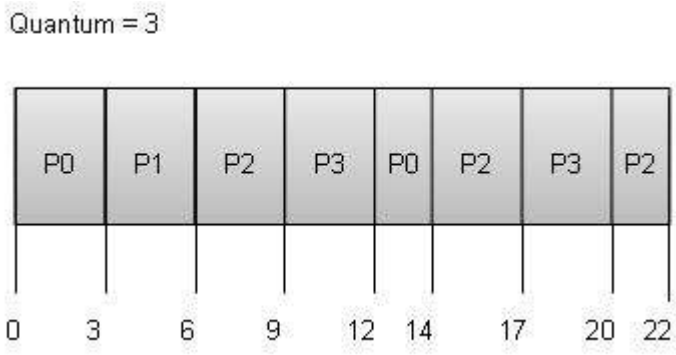
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.

- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows −

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

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.

- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

P3. Process Synchronization

Race Condition

- **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.
- To prevent race conditions, concurrent processes must be **synchronized**.

The Critical-Section Problem

- *n* processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Solution to Critical-Section Problem

- **Mutual Exclusion**. If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.
- **Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- **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

Semaphores

- Synchronization tool that does not require busy waiting.
- Semaphore *S* – integer variable
- can only be accessed via two indivisible (atomic) operations

Two Types of Semaphores

- *Counting* semaphore – integer value can range over an unrestricted domain.
- *Binary* semaphore – integer value can range only between 0 and 1; can be simpler to implement.
- Can implement a counting semaphore *S* as a binary semaphore

Classical Problems of Synchronization

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

1. Explain semaphores and write a short note on it?

Answer: Dijkestra proposed a significant technique for managing concurrent processes for complex mutual exclusion problems. He introduced a new synchronization tool called Semaphore.

Semaphores are of two types −

1. Binary semaphore
2. Counting semaphore

Binary semaphore can take the value 0 & 1 only. Counting semaphore can take nonnegative integer values.

Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The wait, signal operations are also called P and V operations. The manipulation of semaphore (S) takes place as following:

- The wait command P(S) decrements the semaphore value by 1. If the resulting value becomes negative then P command is delayed until the condition is satisfied.
- The V(S) i.e. signals operation increments the semaphore value by 1.

Mutual exclusion on the semaphore is enforced within P(S) and V(S). If a number of processes attempt P(S) simultaneously, only one process will be allowed to proceed & the other processes will be waiting.These operations are defined as under −

```
P(S) or wait(S):
If S > 0 then
  Set S to S-1
Else
  Block the calling process (i.e. Wait on S)
V(S) or signal(S):
If any processes are waiting on S
  Start one of these processes
Else
  Set S to S+1
```

The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature i.e. once started, execution of these operations cannot be interrupted.

Thus semaphore is a simple yet powerful mechanism to ensure mutual exclusion among concurrent processes.

Thread

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

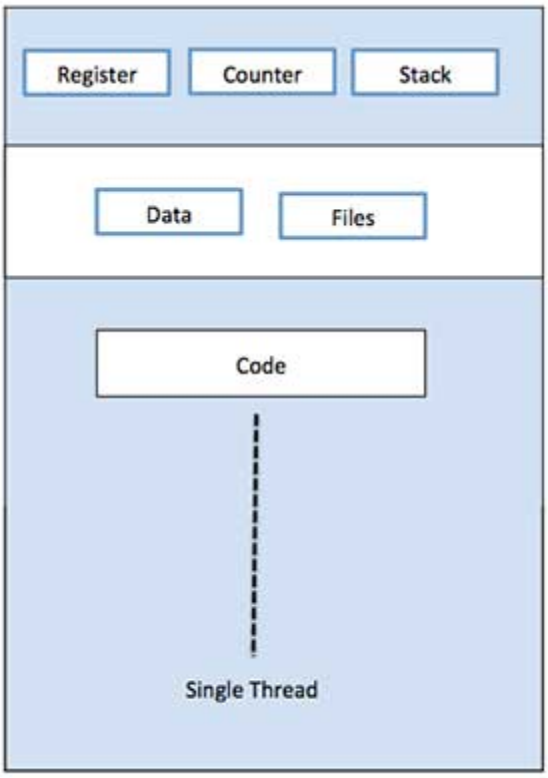
Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.

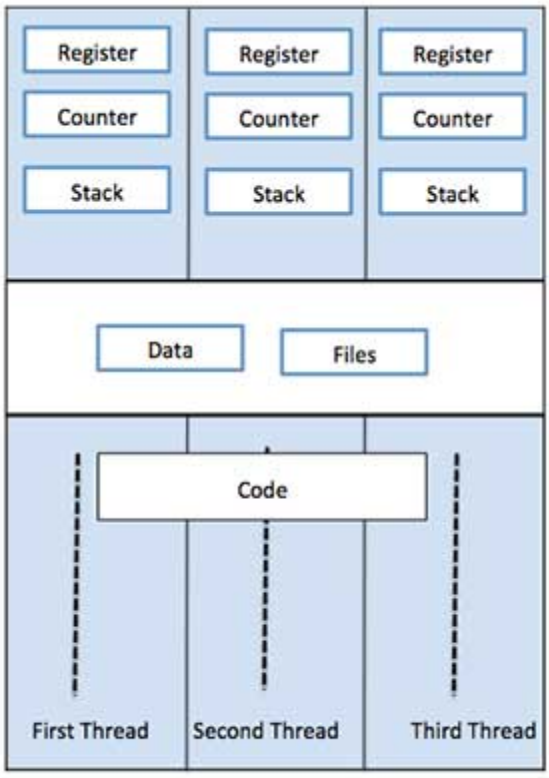
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread Threads are implemented in following two ways

- User Level Threads - User managed threads.
- Kernel Level Threads − Operating System managed threads acting on kernel, an operating system core.



Single Process P with single thread



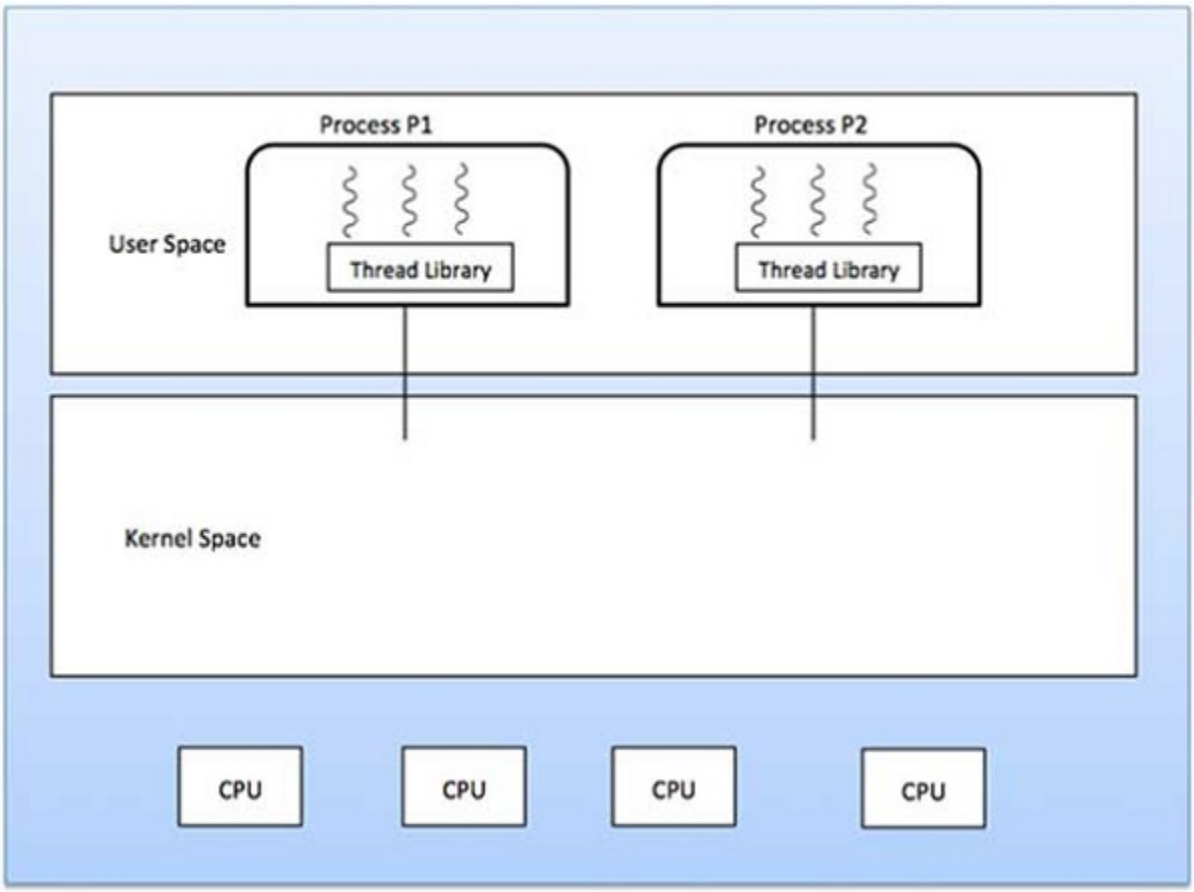
Single Process P with three threads

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.

4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

User Threads



- Thread management done by user-level threads library
- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*
- Advantages
  - Thread switching does not require
  - Kernel mode privileges.
  - User level thread can run on any operating system.
  - Scheduling can be application specific in the user level thread.
  - User level threads are fast to create and manage.
- Disadvantages
  - In a typical operating system, most system calls are blocking.
  - Multithreaded application cannot take advantage of multiprocessing.

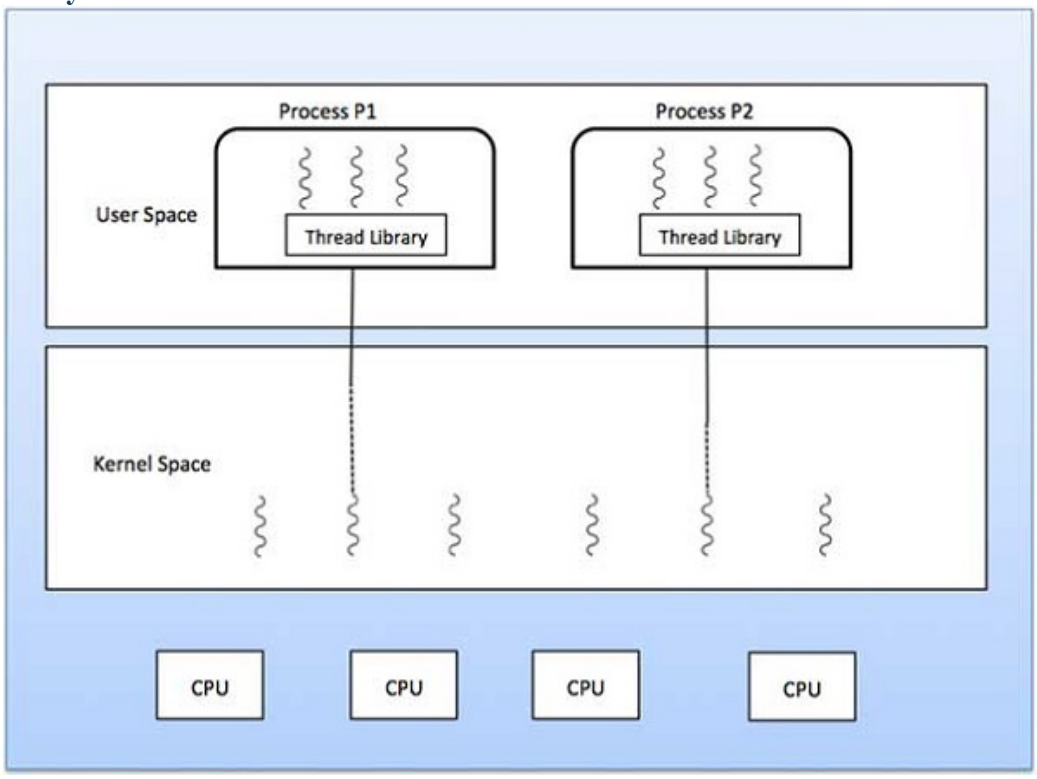
Kernel Threads

- Supported by the Kernel
- Examples
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux
- Advantages
  - Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
  - If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
  - Kernel routines themselves can be multithreaded.
- Disadvantages
  - Kernel threads are generally slower to create and manage than the user threads.
  - Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

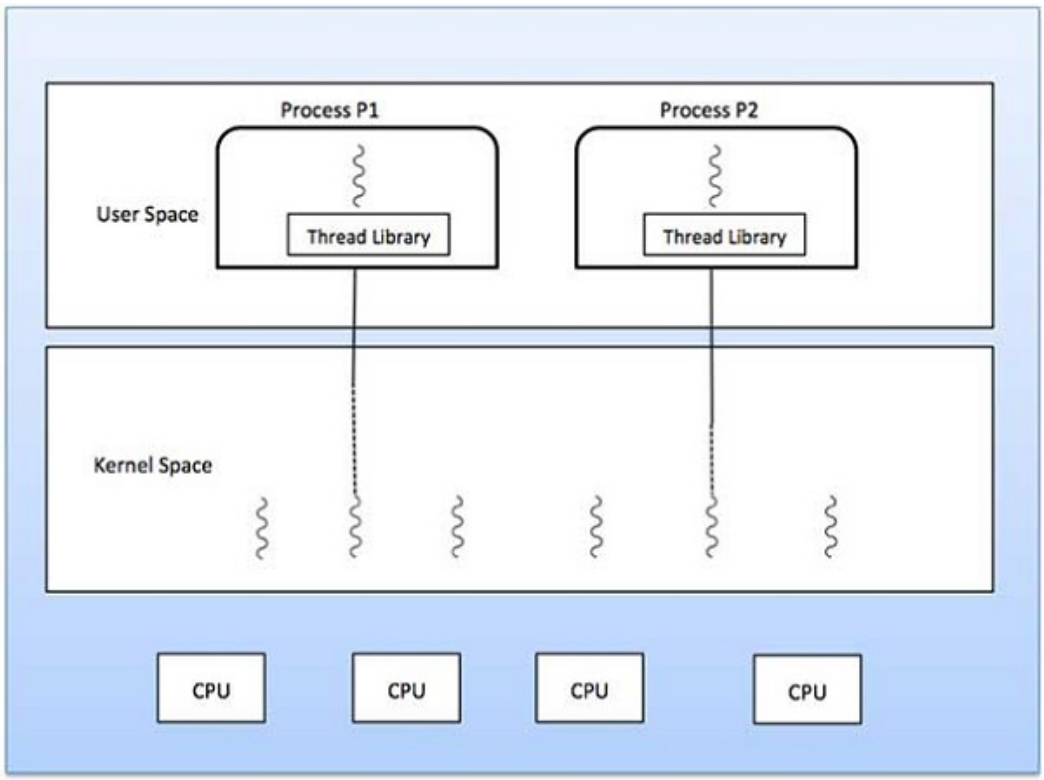
- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One



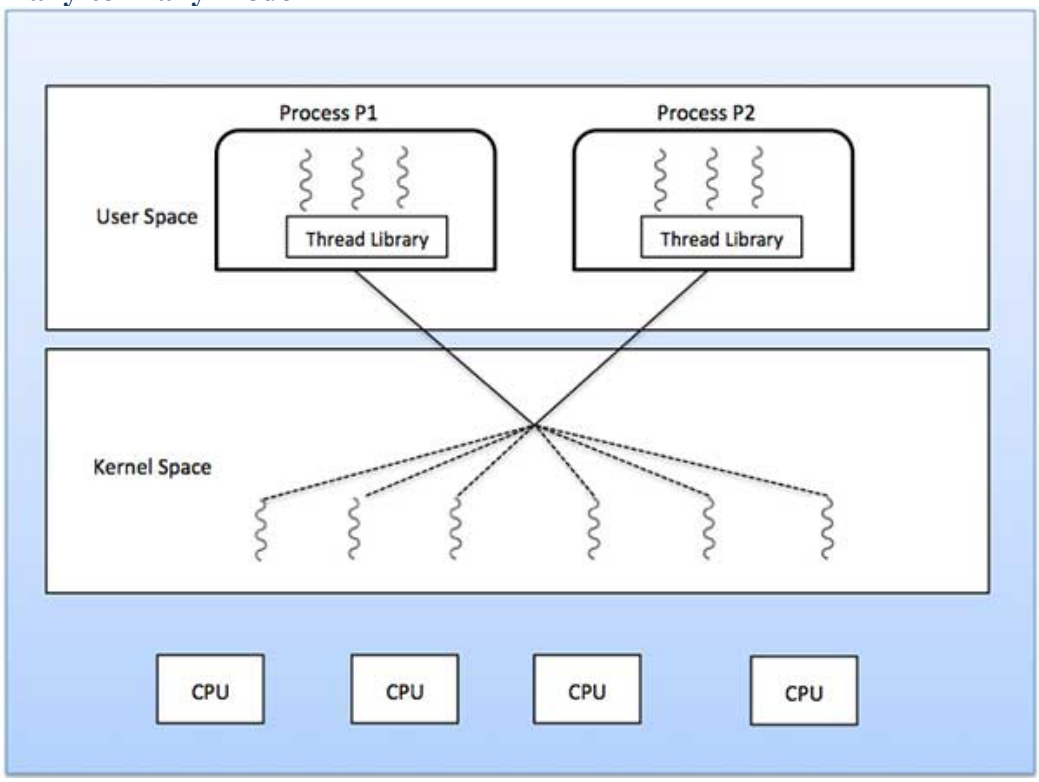
- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.

One-to-One



- Each user-level thread maps to kernel thread.
- Examples: Windows 95/98/NT/2000; OS/2

Many-to-Many Model



- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- Solaris 2

- Windows NT/2000 with the *ThreadFiber* package

Threading Issues

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

Pthreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library; implementation is up to development of the library.
- Common in UNIX operating systems.

Windows 2000 Threads

- Implements the one-to-one mapping.
- Each thread contains
  - a thread id
  - register set
  - separate user and kernel stacks
  - private data storage area

Linux Threads

- Linux refers to them as *tasks* rather than *threads*.
- Thread creation is done through clone () system call.
- Clone() allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface
- Java threads are managed by the JVM.

Deadlock

The Deadlock Problem: A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set {$P0$, $P1$, …, $P0$} 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 $P0$ is waiting for a resource that is held by $P0$.

Resource-Allocation Graph

A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P1, P2, \ldots, Pn\}$, the set consisting of all the processes in the system.
  - $R = \{R1, R2, \ldots, Rm\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P1 \rightarrow Rj$
- assignment edge – directed edge $Rj \rightarrow Pi$

Basic Facts

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

PL1. Suppose n processes, P1, …. Pn share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process Pi is Si, where Si > 0. Which one of the following is a sufficient condition for ensuring that deadlock does not occur?

(a) $\forall i, s_i < m$ (b) $\forall i, s_i < n$

(c) $\sum_{i=1}^{n} s_i < (m+n)$ (d) $\sum_{i=1}^{n} s_i < (m*n)$

Answer (c)

In the extreme condition, all processes acquire Si-1 resources and need 1 more resource. So following condition must be true to make sure that deadlock never occurs.

$\sum_{i=1}^{n}(Si-1)$ < m The above expression can be written as following. $\sum_{i=1}^{n} Si$ < (m + n)

Banker

Total resources in system:

A B C D

6 5 7 6

Available system resources are:

A B C D

3 1 1 2

Processes (currently allocated resources):

A B C D

P1 1 2 2 1

P2 1 0 3 3

P3 1 2 1 0

Processes (maximum resources):

A B C D

P1 3 3 2 2

P2 1 2 3 4

P3 1 3 5 0

Need = maximum resources - currently allocated resources.

Processes (need resources):

A B C D

P1 2 1 0 1

P2 0 2 0 1

P3 0 1 4 0

Memory

M1. Memory General

To view swap file in Linux:

```
swapon -s
Filename                Type            Size    Used    Priority
/dev/sda5               partition       31249404        0       -1
```

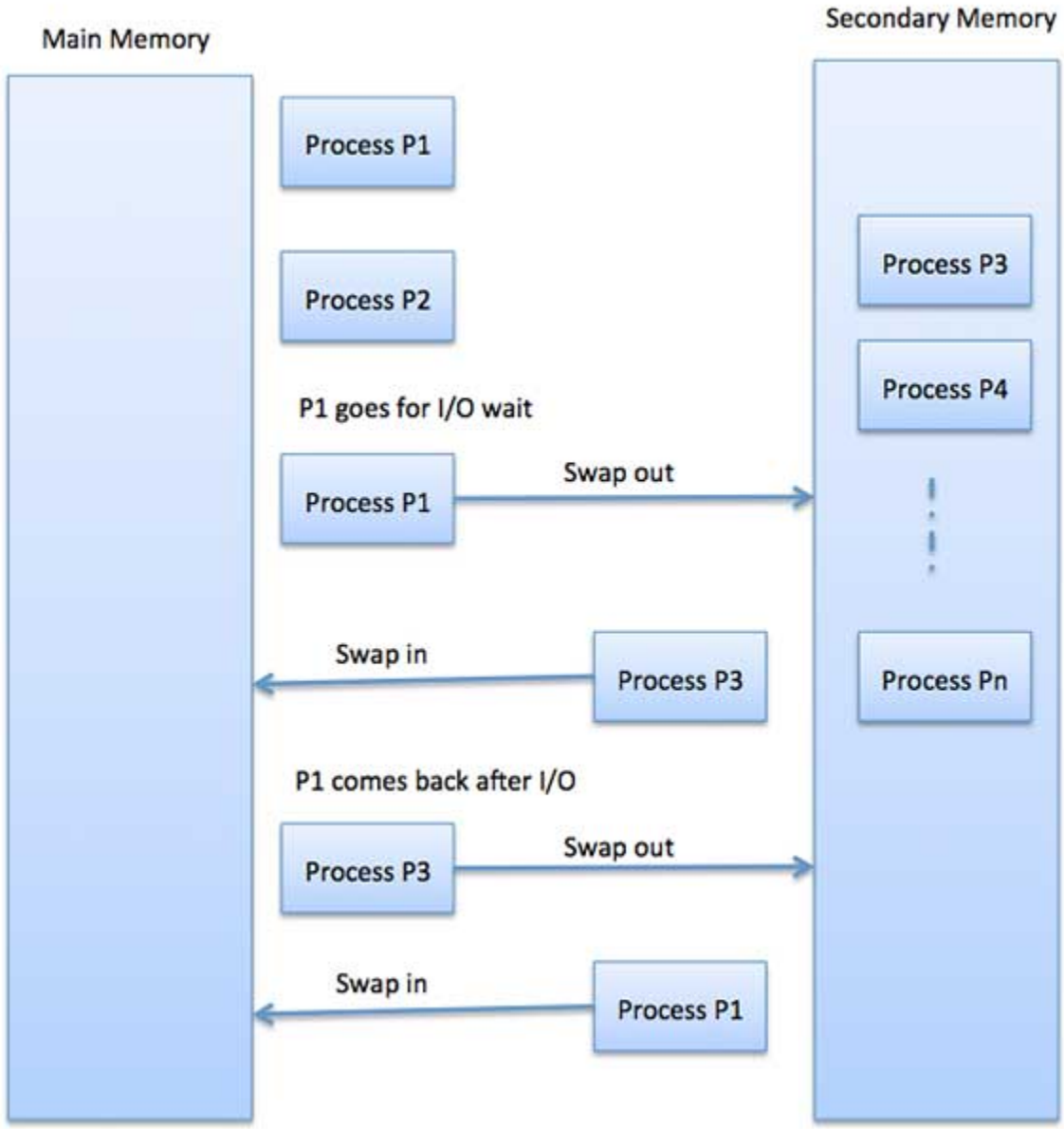
Type: partation
Size: in kB
macOS

```
ls -lh /private/var/vm/swapfile*
```

An Operating System does the following activities for memory management

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Swapping is also known as a technique for memory compaction.



PM1. The main problem with having two programs in memory without memory abstraction is

a. it is difficult to determine the name of each program
b. each program will have different ideas about what the CPU should be doing
c. each program can overwrite the memory of the other
d. none of the above

PM2. Without memory abstraction we can still have multiprogramming

a. by only running one program at a time and completely swapping each successive program in and out of memory
b. by using VT
c. by taking advantage of an MMU
d. with paging

PM3. Even without swapping or memory abstraction we can multiprogram
a. if we write our code very carefully
b. if we have an MMU
c. if we have special hardware to divide memory between different programs
d. all of the above

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program.

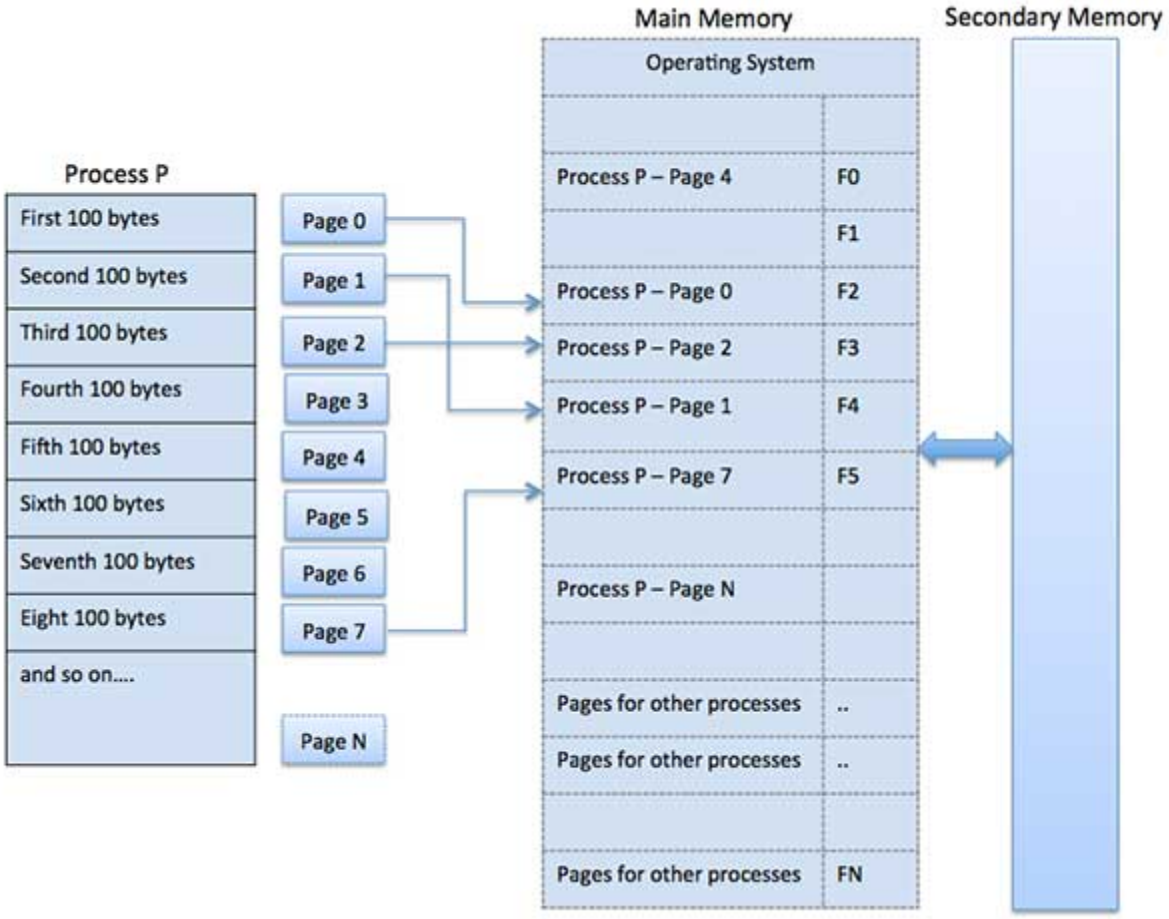
The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

M2. Paging & Segmentation

1. What are the differences between paging and segmentation?

Sr. No.	Paging	Segmentation
1	A page is a physical unit of information.	A segment is a logical unit of information.
2	A page is invisible to the user's program.	A segment is visible to the user's program.
3	A page is of fixed size e.g. 4Kbytes.	A segment is of varying size.
4	The page size is determined by the machine architecture.	A segment size is determined by the user.
5	Fragmentation may occur.	Segmentation eliminates fragmentation.
6	Page frames on main memory are required.	No frames are required.

Paging



Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Address Translation

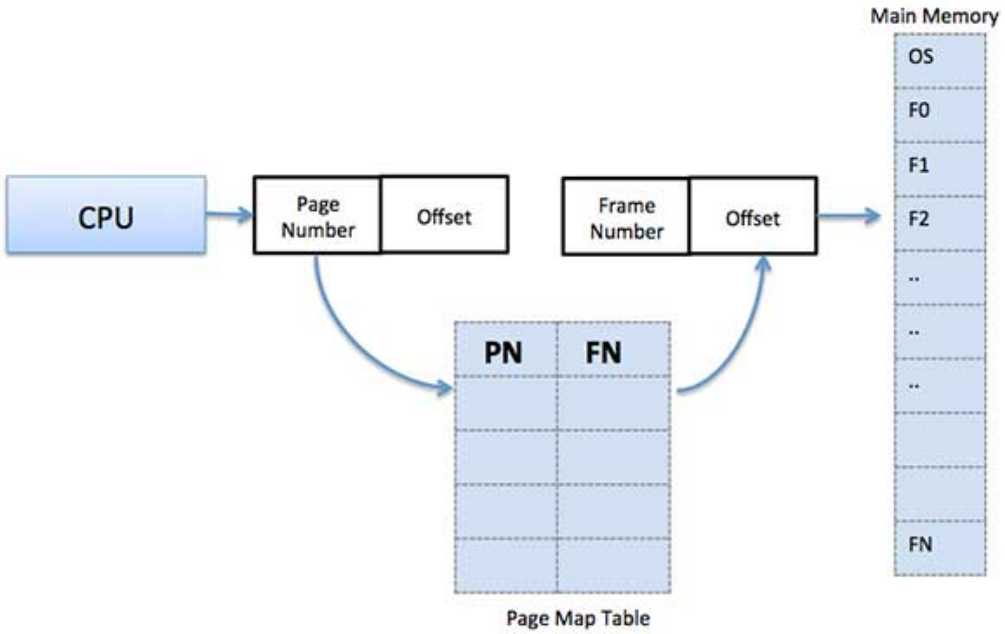
Page address is called **logical address** and represented by **page number**and the **offset**.

```
Logical Address = Page number + page offset
```

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

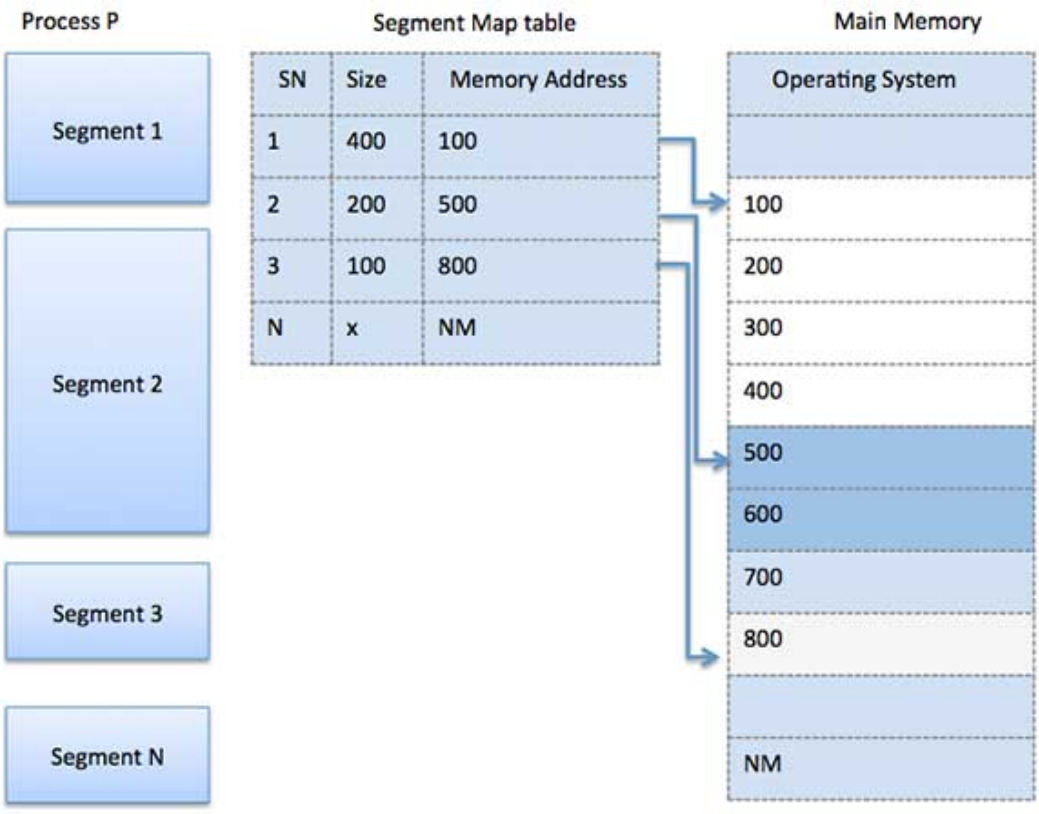
```
Physical Address = Frame number + page offset
```

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



Segmentation

The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



M3. Allocation algorithms

Memory allocation

Main memory usually has two partitions

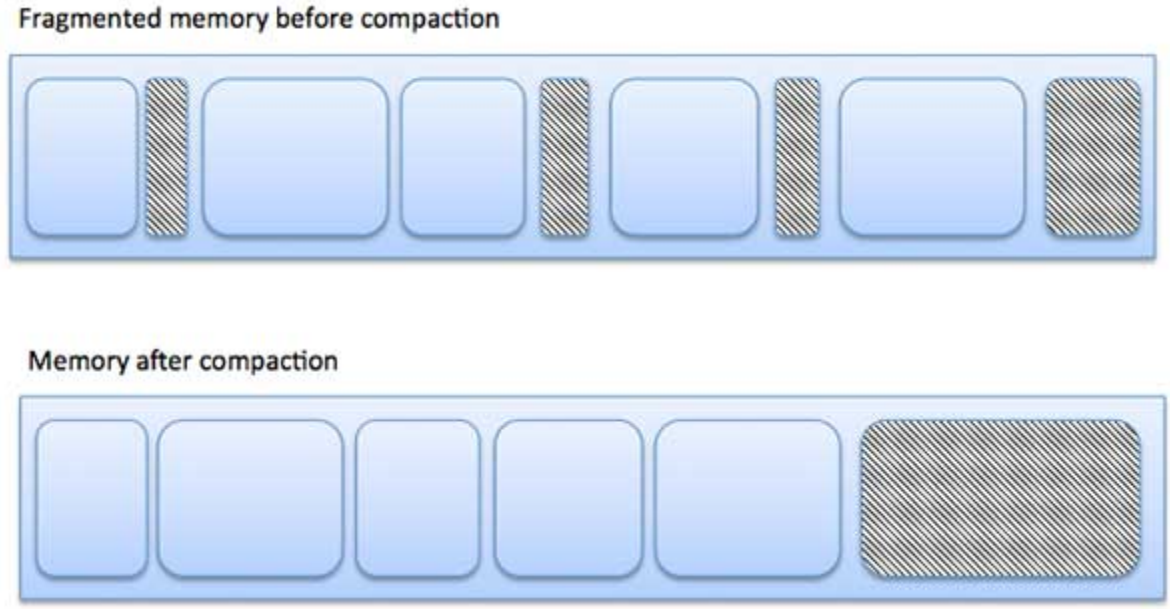
- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. 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.

Fragmentation is of two types

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

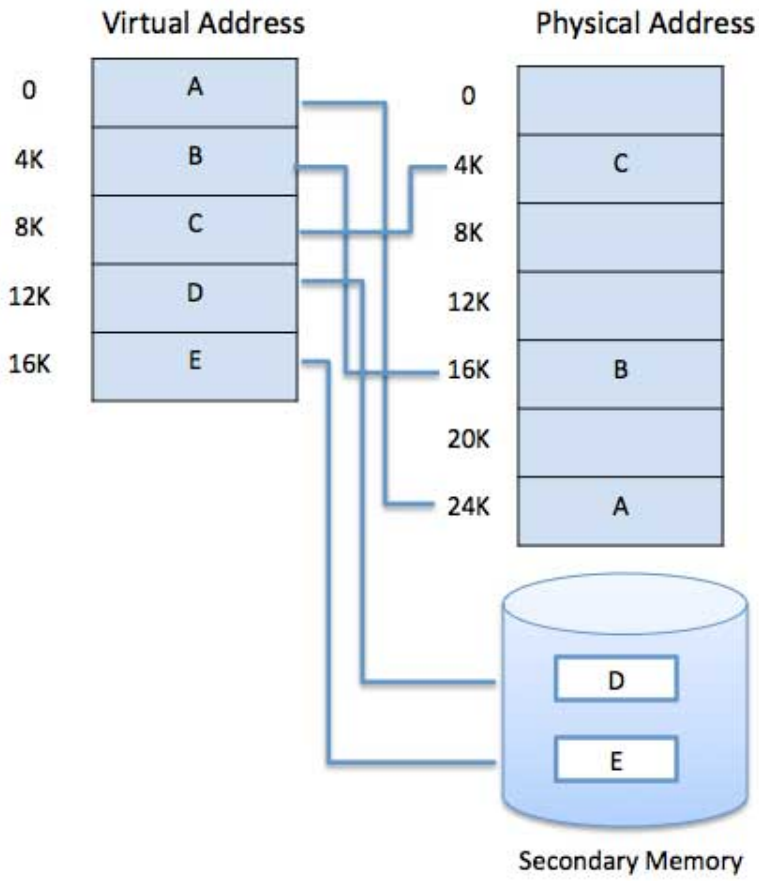


Algorithms

- First Fit
- Best fit
- Worst fit
- Quick fit
- Next fit

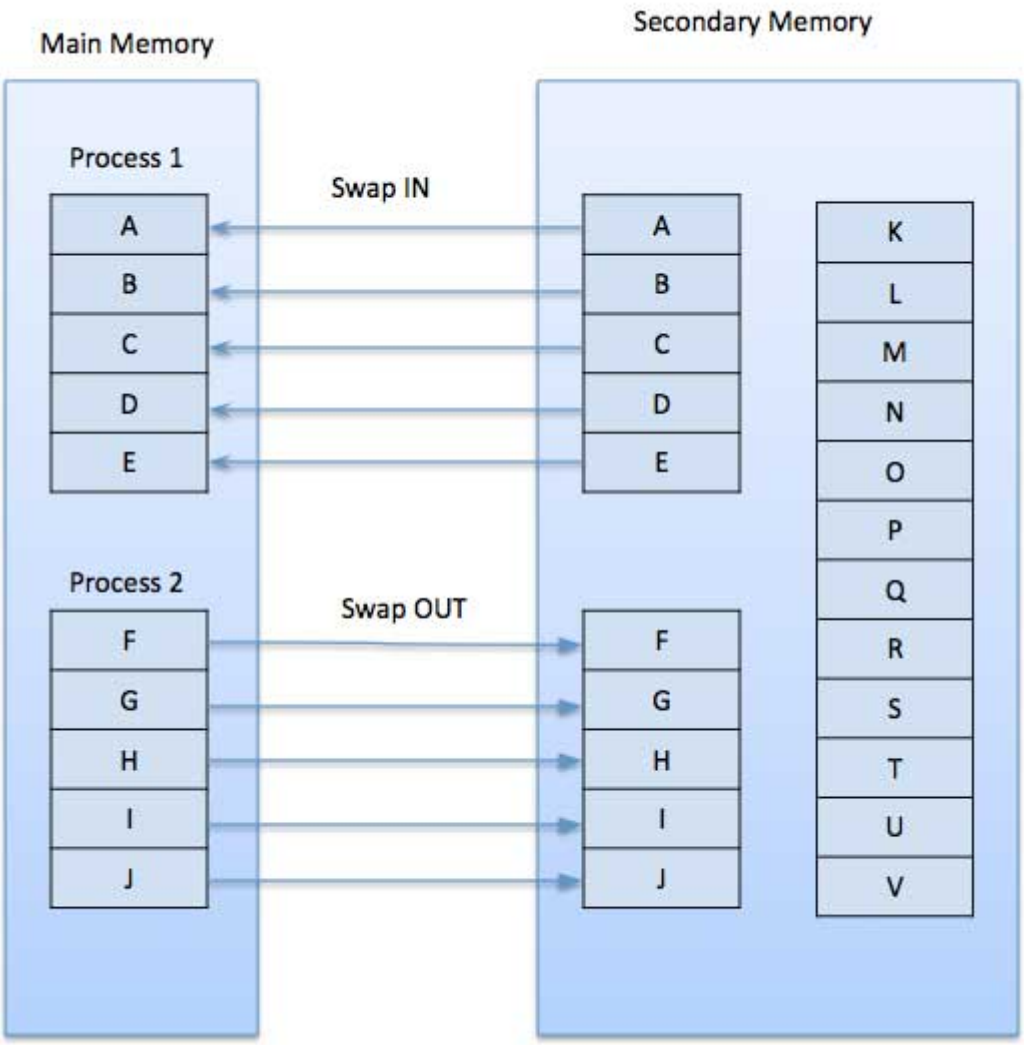
M4. Replacement algorithms

Virtual memory



Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



Advantages of demand page

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages of demand page

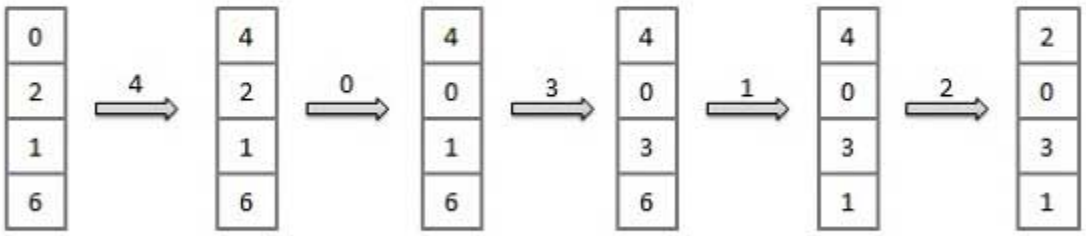
- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Replacement Algorithms
First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



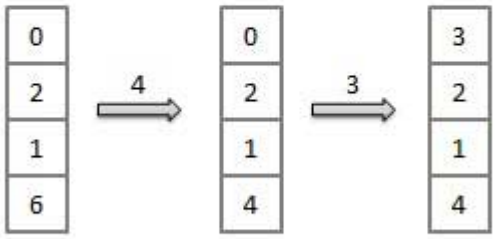
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



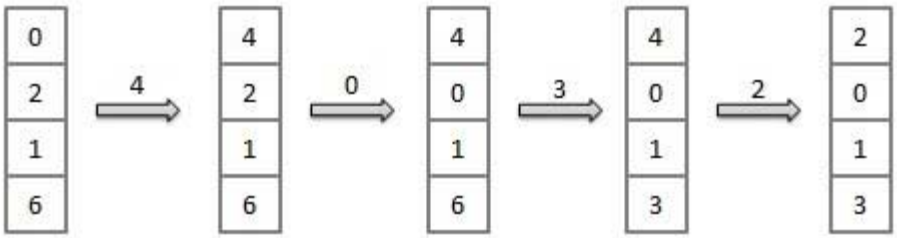
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Disk

D1. Disk Concepts

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
    - Sector 0 is the first sector of the first track on the outermost cylinder.

- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time ≈ seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).
  98, 183, 37, 122, 14, 124, 65, 67
  Head pointer 53

FCFS

SSTF

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.

C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

Disk Management

- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
    - *Partition* the disk into one or more groups of cylinders.
    - *Logical formatting* or "making a file system".
- Boot block initializes system.
    - The bootstrap is stored in ROM.
    - *Bootstrap loader* program.

Methods such as *sector sparing* used to handle bad blocks.

D2. Disk latancy - Practice

- **Seek time –** The time taken by the R-W head to reach the desired track from it's current position.
- **Rotational latency –** Time taken by the sector to come under the R-W head.
- **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
- **Controller time –** The processing time taken by the controller.
- **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

PD1. Consider a hard disk with:

- 4 surfaces
- 64 tracks/surface
- 128 sectors/track
- 256 bytes/sector

What is the capacity of the hard disk?
Disk capacity = surfaces * tracks/surface * sectors/track * bytes/sector
Disk capacity = 4 * 64 * 128 * 256
Disk capacity = 8 MB
The disk is rotating at 3600 RPM, what is the data transfer rate?
60 sec -> 3600 rotations
1 sec -> 60 rotations
Data transfer rate = number of rotations per second * track capacity * number of surfaces (since 1 R-W head is used for each surface)
Data transfer rate = 60 * 128 * 256 * 4
Data transfer rate = 7.5 MB/sec
The disk is rotating at 3600 RPM, what is the average access time?
Since, seek time, controller time and the amount of data to be transferred is not given, we consider all the three terms as 0.
Therefore, Average Access time = Average rotational delay
Rotational latency => 60 sec -> 3600 rotations
1 sec -> 60 rotations
Rotational latency = (1/60) sec = 16.67 msec.
Average Rotational latency = (16.67)/2
= 8.33 msec.
Average Access time = 8.33 msec.

PD2. A hard disk system has the following parameters

- Number of tracks = 500
- Number of sectors/track = 100
- Number of bytes /sector = 500
- Time taken by the head to move from one track to adjacent track = 1 ms
- Rotation speed = 600 rpm.

What is the average time taken for transferring 250 bytes from the disk ?
(A) 300.5 ms
(B) 255.5 ms
(C) 255.0 ms
(D) 300.0 ms

Explanation: Avg. time to transfer = Avg. seek time + Avg. rotational delay + Data transfer time

- **Avg Seek Time** – time taken to move from 1st track to 1sr track : 0ms, 1st to 2nd : 1ms, 2ms, 3ms,….499ms
  Avg Seek time =(∑0+1+2+3+…+499)/500 = 249.5 ms
- **Avg Rotational Delay –** RMP : 600 , 600 rotations in 60 sec (one Rotation = 60/600 sec = 0.1 sec) So, Avg Rotational Delay = 0.1/2= 50 ms
- **Data Transfer Time:** In One 1 Rotation we can read data on one track = 100 * 500 = 50,000 B data is read in one rotation. 250 bytes -> 0.1 * 250 / 50,000 = 0.5 ms

Therfore ATT = 249.5+50+0.5 = 300 ms

PD3. Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing _____ number of requests.
(A) 1
(B) 2
(C) 3
(D) 4

Explanation: In Shortest-Seek-First algorithm, request closest to the current position of the disk arm and head is handled first.
In this question, the arm is currently at cylinder number 100. Now the requests come in the queue order for cylinder numbers 30, 85, 90, 100, 105, 110, 135 and 145.

The disk will service that request first whose cylinder number is closest to its arm. Hence 1st serviced request is for cylinder no 100 (as the arm is itself pointing to it), then 105, then 110, and then the arm comes to service request for cylinder 90. Hence before servicing request for cylinder 90, the disk would had serviced 3 requests.

PD4. Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance

of user programs?
(A) 50%
(B) 40%
(C) 25%
(D) 0%

Explanation: Since Operating System can execute a single sequential user process at a time, the disk is accessed in FCFS manner always. The OS never has a choice to pick an IO from multiple IOs as there is always one IO at a time

PD5. Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks
(A) 8
(B) 9
(C) 10
(D) 11

Explanation: In SSTF closest request to the current position of the head, and then services that request next.
In SCAN, requests are serviced only in the current direction of arm movement until the arm reaches the edge of the disk. When this happens, the direction of the arm reverses, and the requests that were remaining in the opposite direction are serviced, and so on.

Given a disk with 100 tracks

And Sequence 45, 20, 90, 10, 50, 60, 80, 25, 70.

Initial position of the R/W head is on track 50.

In SSTF, requests are served as following

Next Served	Distance Traveled
50	0
45	5
60	15
70	10
80	10
90	10
25	65
20	5
10	10

Total Dist = 130

If Simple SCAN is used, requests are served as following

Next Served	Distance Traveled
50	0
60	10
70	10
80	10
90	10
45	65 [disk arm goes to 100, then to 45]
25	20
20	5
10	10

Total Dist = 140

Less Distance traveled in SSTF = 130 - 140 = 10

Therefore, it is **not additional** but it is **less distance** traversed by SSTF than SCAN.

PD6. Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50 × 10^6 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is _____**?**

Explanation:

Disk latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead

Seek Time? Depends no. tracks the arm moves and seek speed of disk

Rotation Time? depends on rotational speed and how far the sector is from the head

Transfer Time? depends on data rate (bandwidth) of disk (bit density) and the size of request

Disk latency = Seek Time + Rotation Time +
Transfer Time + Controller Overhead

Average Rotational Time = (0.5)/(15000 / 60) = 2 miliseconds

[On average half rotation is made]

It is given that the average seek time is twice the average rotational delay

So Avg. Seek Time = 2 * 2 = 4 miliseconds.

Transfer Time = 512 / (50 × 10^6 bytes/sec)
= 10.24 microseconds

Given that controller time is 10 times the average transfer time
Controller Overhead = 10 * 10.24 microseconds
= 0.1 miliseconds

Disk latency = Seek Time + Rotation Time +
Transfer Time + Controller Overhead
= 4 + 2 + 10.24 * 10^{-3} + 0.1 miliseconds
= 6.1 miliseconds

File system

An Operating System does the following activities for file management

- Keeps track of information, location, uses, status etc.
- The collective facilities are often known as file system.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Access Methods

- **Sequential Access**
  read next
  write next
  reset
  no *read after last write*
  (rewrite)
- **Direct Access**
  read n
  write n
  position to n
  read next
  write next
  rewrite n

  n = relative block number

File types

- Ordinary files
  - These are the files that contain user information.
  - These may have text, databases or executable program.

    - The user can apply various operations on such files like add, modify, delete or even remove the entire file.
- Directory files
    - These files contain list of file names and other information related to these files.
- Special files
    - These files are also known as device files.
    - These files represent physical device like disks, terminals, printers, networks, tape drive etc.

Special files types

- Character special files − data is handled character by character as in case of terminals or printers.
- Block special files − data is handled in blocks as in the case of disks and tapes.

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.

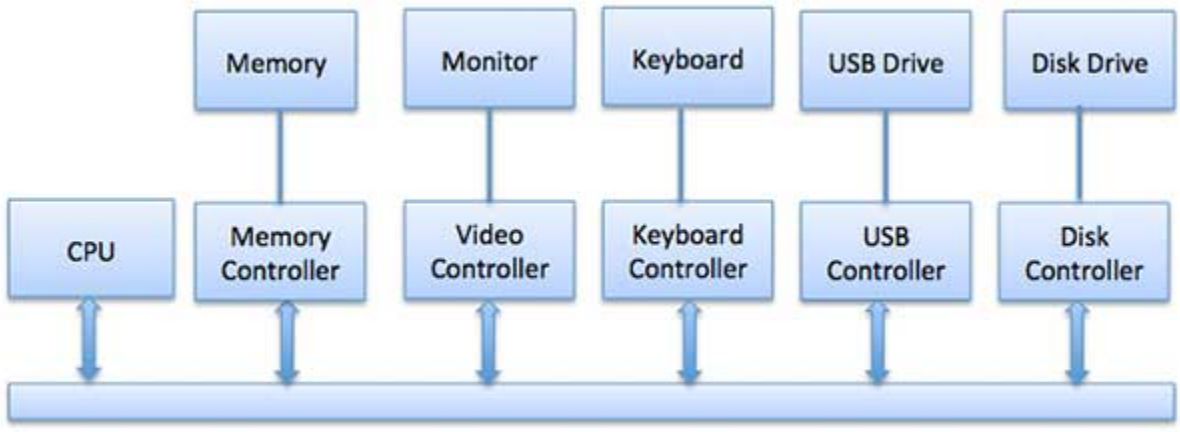
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

I/O

IO1. I/O concepts

I/O devices can be divided into two categories

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc



Synchronous vs asynchronous I/O

- Synchronous I/O − In this scheme CPU execution waits while I/O proceeds
- Asynchronous I/O − I/O proceeds concurrently with CPU execution

Communication to I/O Devices

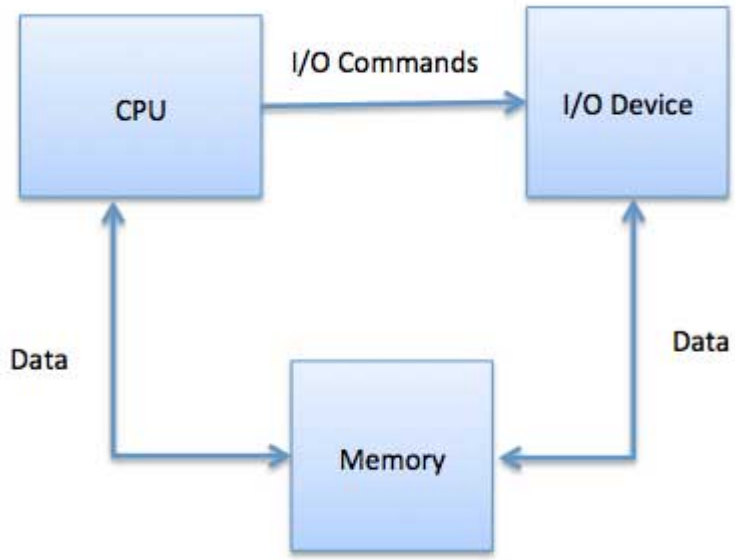
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

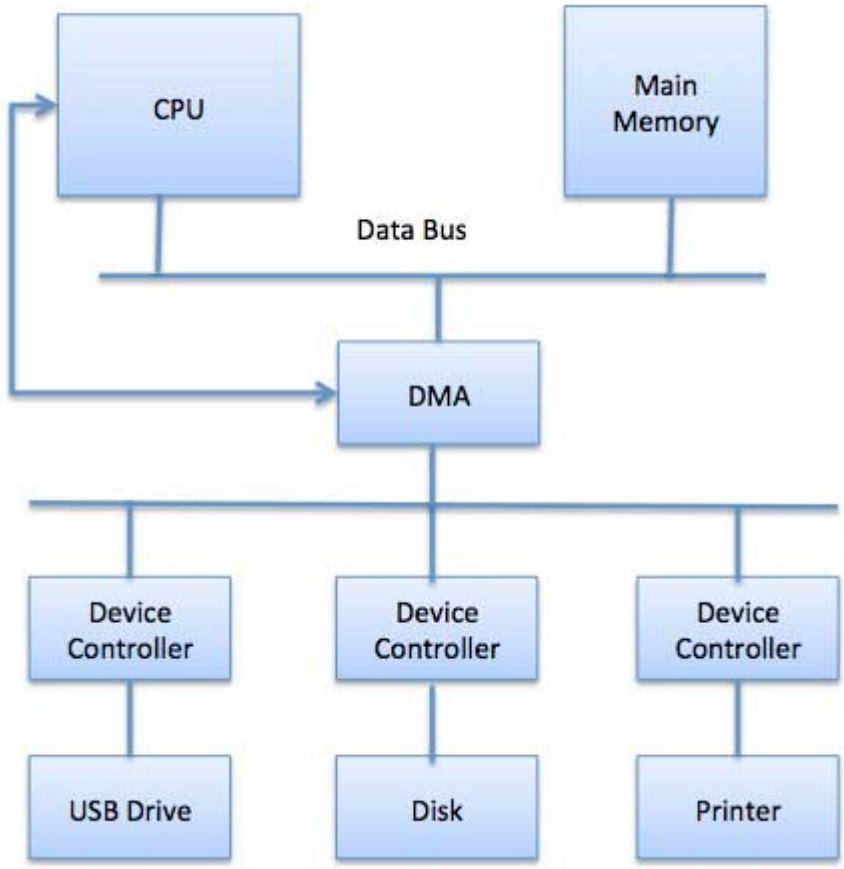
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as polling and interrupts. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.
Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

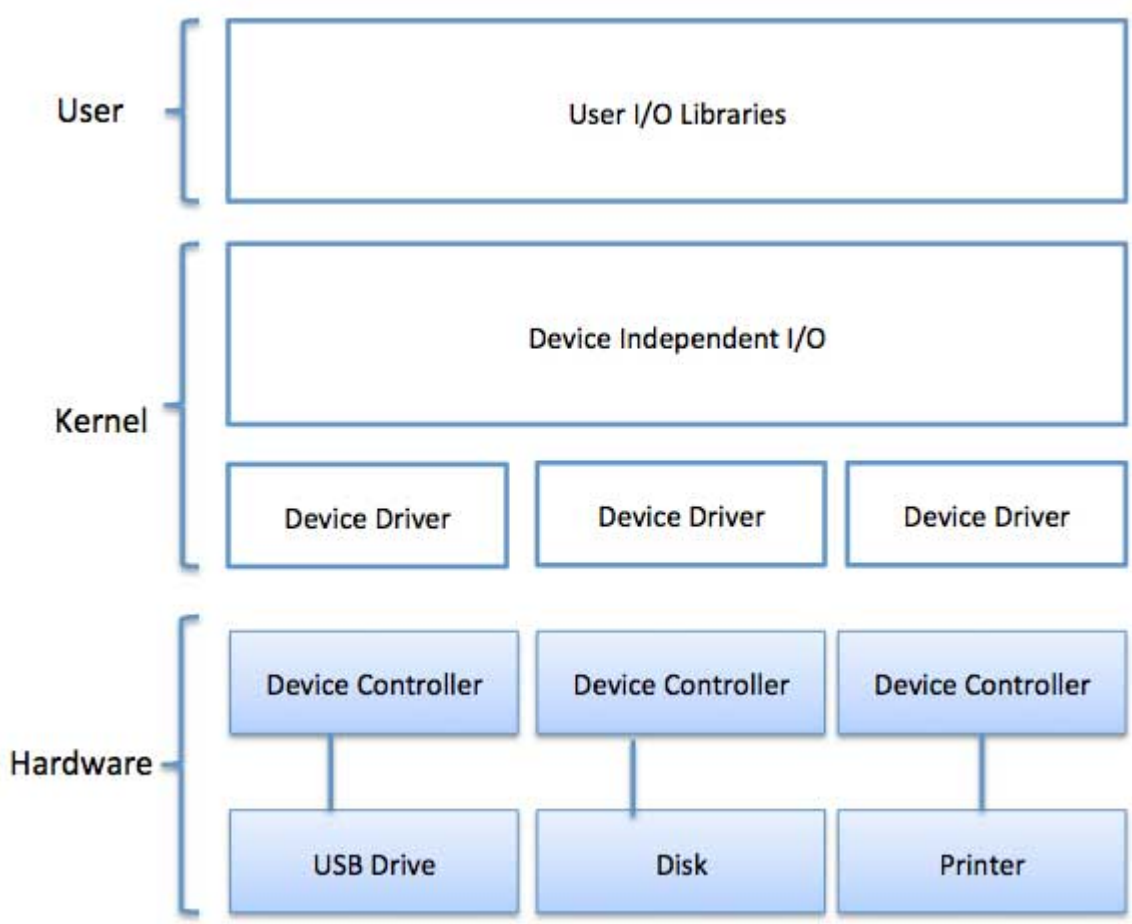
Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.
A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

IO2. I/O layer

An Operating System manages device communication via their respective drivers. It does the following activities for device management

- Keeps tracks of all devices.
- Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.



Device Drivers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software

- Uniform interfacing for device drivers
- Device naming - Mnemonic names mapped to Major and Minor device numbers
- Device protection
- Providing a device-independent block size
- Buffering because data coming off a device cannot be stored in final destination.
- Storage allocation on block devices
- Allocation and releasing dedicated devices
- Error Reporting

User-Space I/O Software

These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

Kernel I/O Subsystem

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

- **Scheduling** − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering** − Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
- **Caching** − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **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.

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.

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username / Password** − User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.
- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

System Threats

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

- **Worm** − Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
- **Port Scanning** − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.

- **Denial of Service** − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

Computer Security Classifications

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications in computer systems: A, B, C, and D. This is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security.
2	**Type B** Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types. **B1** − Maintains the security label of each object in the system. Label is used for making decisions to access control. **B2** − Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. **B3** − Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.
3	**Type C** Provides protection and user accountability using audit capabilities. It is of two types. **C1** − Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. **C2** − Adds an individual-level access control to the capabilities of a Cl level system.
4	**Type D** Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category.