Operating System
CE0418

Unit-1

Introduction to Operating System

Prof. Indr Jeet Rajput

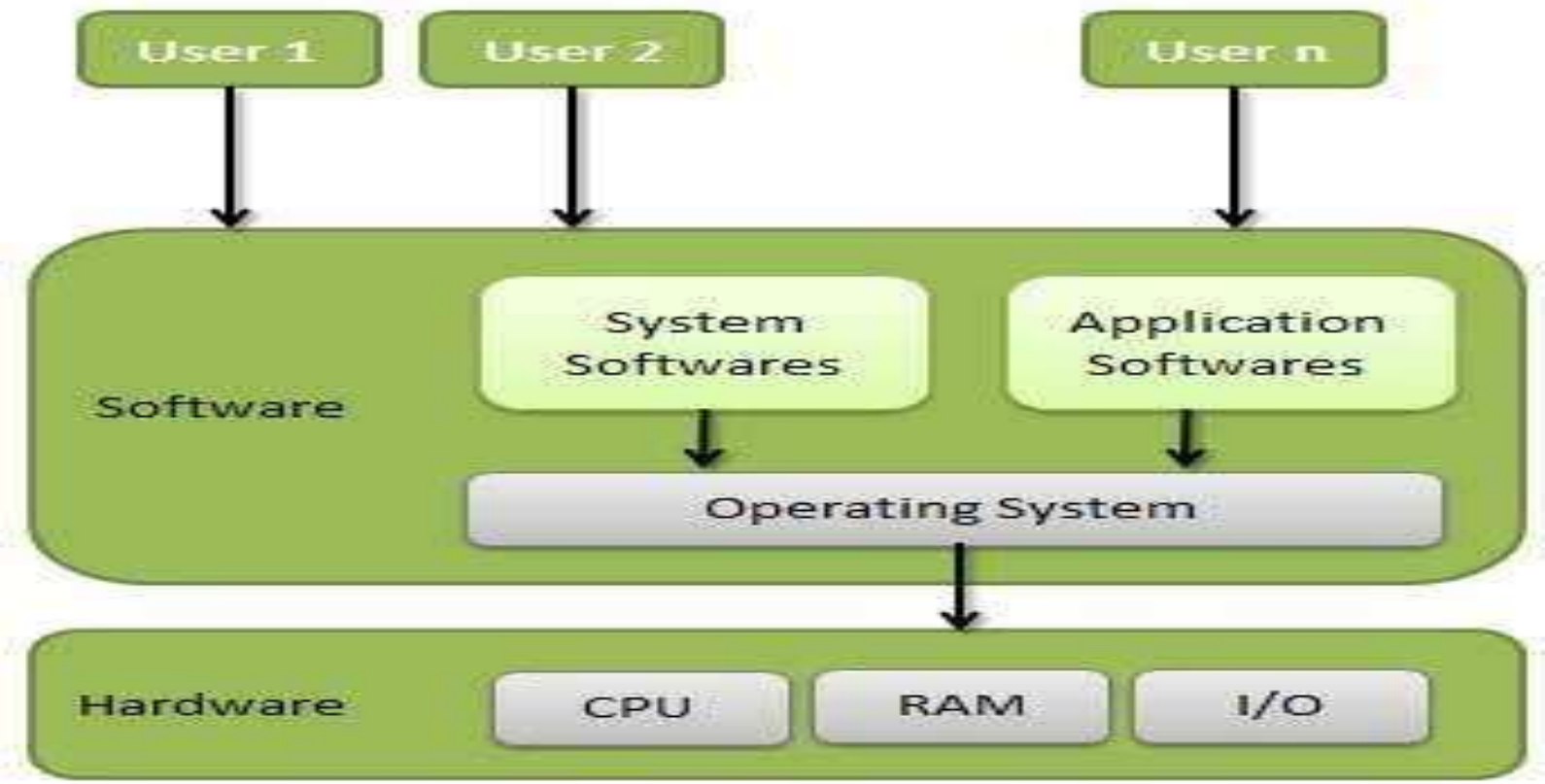
Assistant Professor CE & IT

Introduction to Operating System

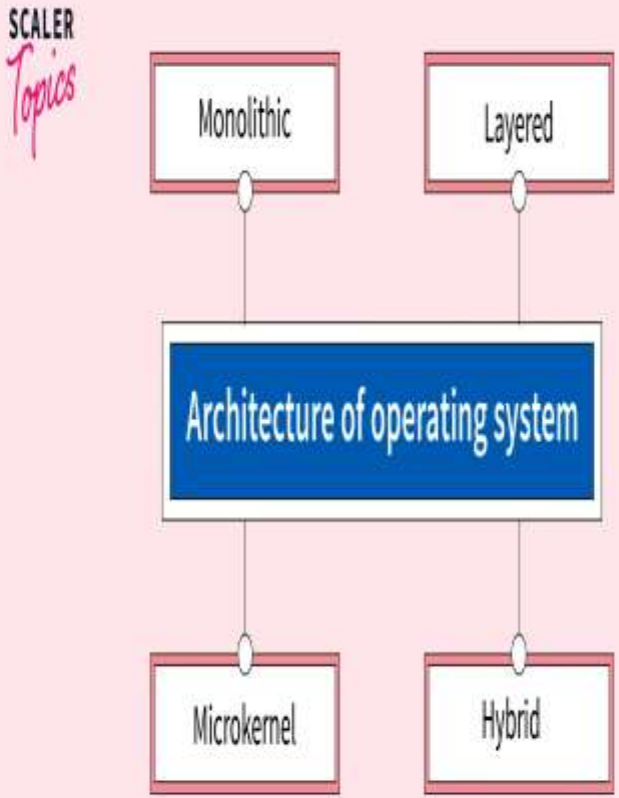
- An Operating System (OS) is an interface between a computer user and computer hardware.
- An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.
- An operating system is software that enables applications to interact with a computer's hardware.
- The software that contains the core components of the operating system is called the kernel.
- The primary purposes of an Operating System are to enable applications (softwares) to interact with a computer's hardware and to manage a system's hardware and software resources.

Architecture of OS

Architecture: We can draw a generic architecture diagram of an Operating System which is as follows:



Architecture of Operating System



Architecture of Operating System

- Before explaining various architectures of the operating systems, let's explore a few terms first which are part of the operating system.

1) Application: The application represents the software that a user is running on an operating system it can be either system or application software eg slack, sublime text editor, etc.

2) Shell: The shell represents software that provides an interface for the user where it serves to launch or start some program for which the user gives instructions.

- It can be of two types first is a command line and another is a graphical user interface for eg: MS-DOS Shell, PowerShell, csh, ksh, etc.

3) Kernel: Kernel represents the most central and crucial part of the operating system where it is used for resource management i.e. it provides necessary I/O, processor, and memory to the application processes through inter-process communication mechanisms and system calls.

Architecture of Operating System

- Types of Architectures of Operating System: Architectures of operating systems can be of four types monolithic, layered, microkernel, and hybrid.

1) Monolithic Architecture:

- In monolithic architecture, each component of the operating system is contained in the kernel i.e. it is working in kernel space, and the components of the operating system communicate with each other using function calls.
- Examples of this type of architecture are OS/360, VMX, and LINUX.

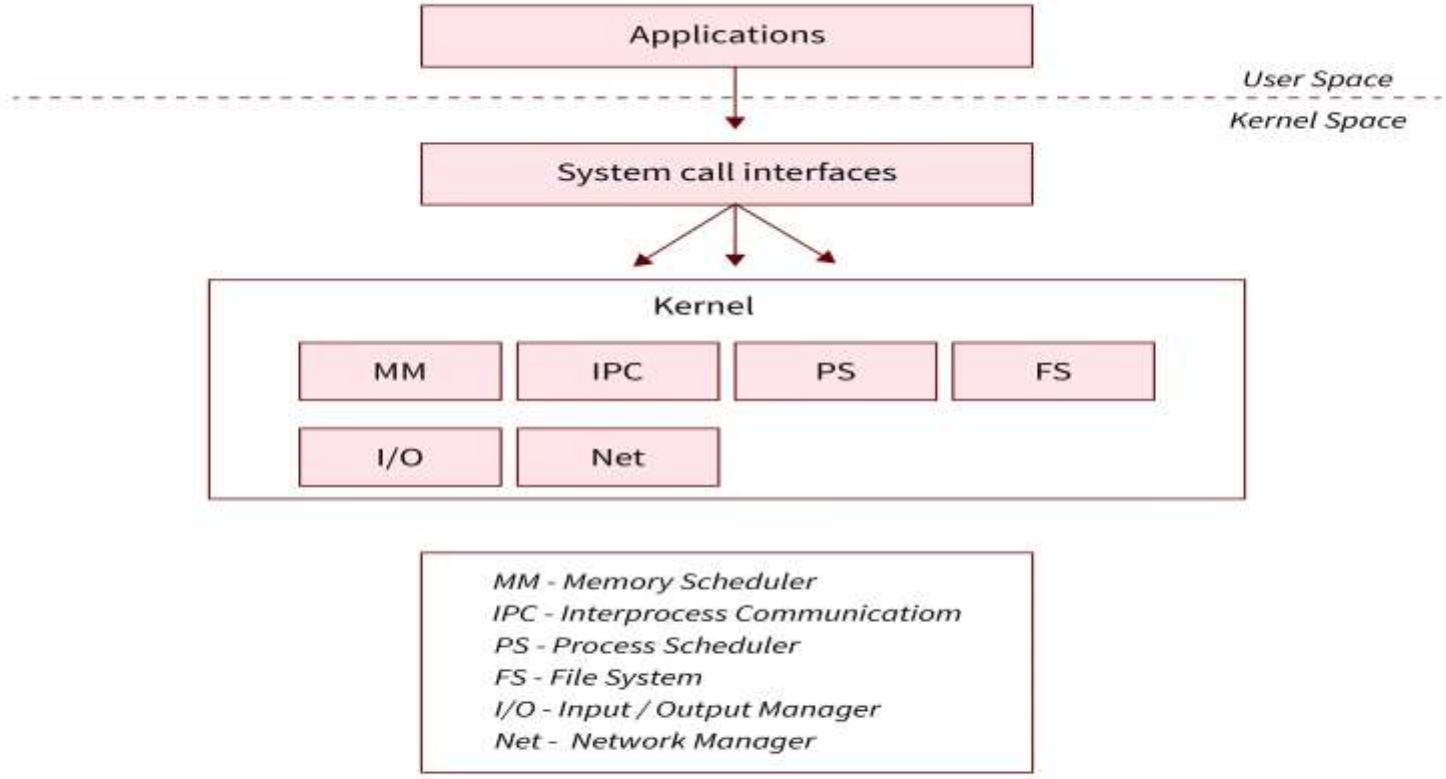
Advantages:

- The main advantage of having a monolithic architecture of the operating system is that it provides CPU scheduling, memory management, memory management, etc through system calls.
- In a single address space, the entire large process is running.
- It is a single static binary file.

Disadvantages:

- The main disadvantage is that all components are interdependent and when one of them fails the entire system fails.
- In case the user has to add a new service or functionality the entire operating system needs to be changed.

Architecture of Operating System



SCALER Topics

Architecture of Operating System

2) Layered architecture:

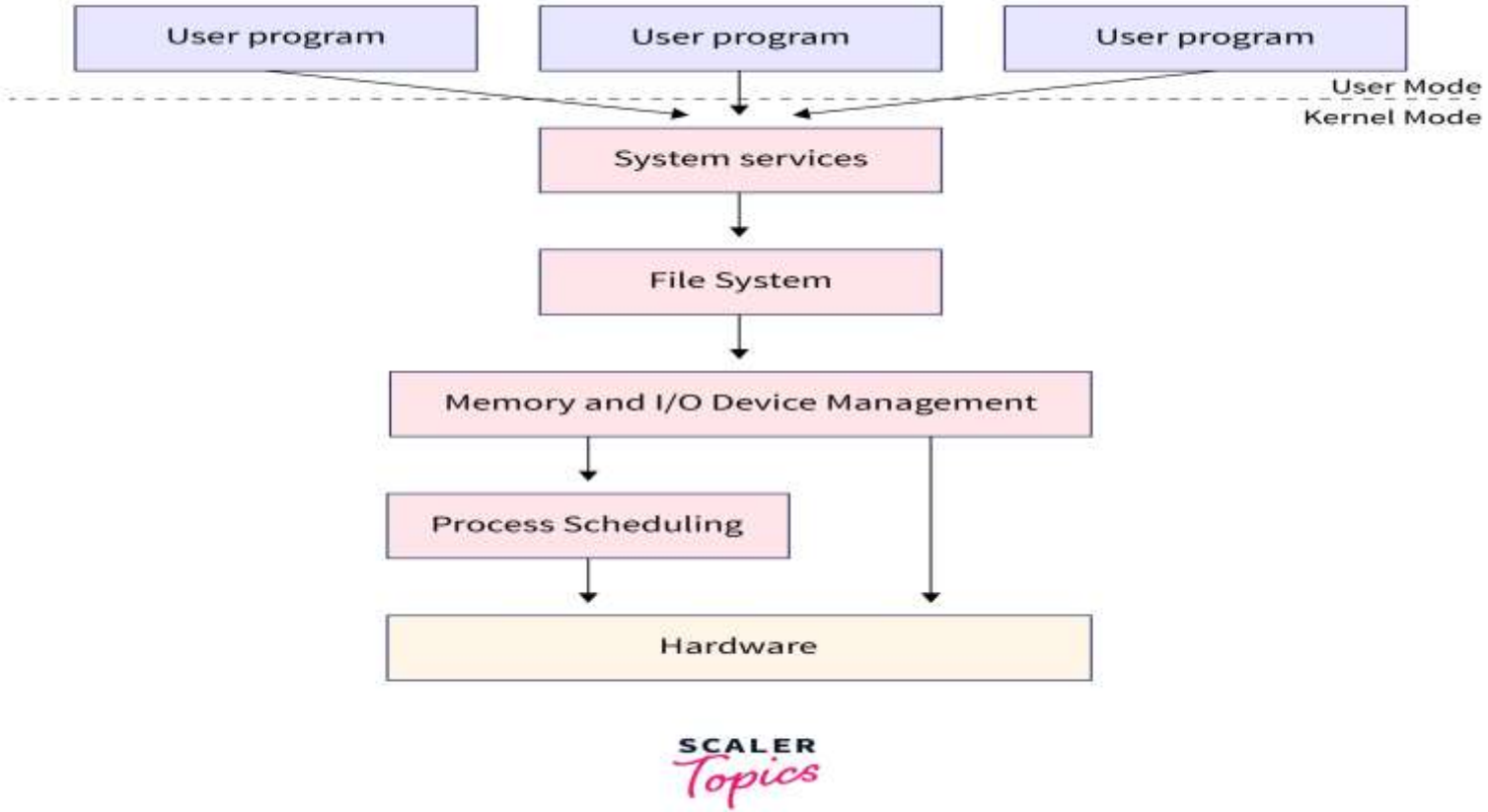
- In Layered architecture, components with similar functionalities are grouped to form a layer and in this way, total n+1 layers are constructed and counted from 0 to n where each layer has a different set of functionalities and services. Example: THE operating system, also windows XP, and LINUX implements some level of layering.

The layers are implemented according to the following rule:

- Each layer can communicate with all of its lower layers but not with its upper layer i.e. any ith layer can communicate with all layers from 0 to i-1 but not with the i+1th layer.
- Each layer is designed in such a way that it will only need the functionalities which are present in itself or the layers below it.

Architecture of Operating System

- There are 6 layers in layered architecture as shown below:
-



Architecture of Operating System

- There are 6 layers in layered architecture as shown below:

1) Hardware: This layer is the lowest layer in the layered operating system architecture, this layer is responsible for the coordination with peripheral devices such as keyboards, mice, scanners etc.

2) CPU scheduling: This layer is responsible for process scheduling, multiple queues are used for scheduling.

- Process entering the system are kept in the job queue while those which are ready to be executed are put into the ready queue.
- It manages the processes which are to be kept in the CPU and those which are to be kept out of the CPU.

3) Memory Management: This layer handles the aspect of memory management i.e. moving the processes from the secondary to primary memory for execution and vice-versa.

- There are memories like RAM, and ROM. RAM is the memory where our processes run they are moved to the RAM for execution and when they exit they are removed from RAM.

Architecture of Operating System

4) Process Management: This layer is responsible for managing the various processes i.e. assigning the CPU to those processes on a priority basis for their execution.

- Process management uses many scheduling algorithms for prioritizing the processes for execution such as the Round-Robin algorithm, FCFS(First Come First Serve), SJF(Shortest Job First), etc.

5) I/O Buffer: Buffering is the temporary storage of data and I/O Buffer means that the data input is first buffered before storing it in the secondary memory.

- All I/O devices have buffers attached to them for the temporary storage of the input data because it cannot be stored directly in the secondary storage as the speed of the I/O devices is slow as compared to the processor.

6) User Programs: This is the application layer of the layered architecture of the operating system, it deals with all the application programs running eg games, browsers, words, etc.

- It is the highest layer of layered architecture.

Architecture of Operating System

Advantages:

1) Layered architecture of the operating system provides modularity because each layer is programmed to perform its own tasks only.

2) Since the layered architecture has independent components changing or updating one of them will not affect the other component or the entire operating system will not stop working, hence it is easy to debug and update.

3) The user can access the services of the hardware layer but cannot access the hardware layer itself because it is the innermost layer.

4) Each layer has its own functionalities and it is concerned with itself only and other layers are abstracted from it.

Disadvantages:

1) Layered architecture is complex in implementation because one layer may use the services of the other layer and therefore, the layer using the services of another layer must be put below the other one.

2) In a layered architecture, if one layer wants to communicate with another it has to send a request which goes through all layers in between which increases response time causing inefficiency in the system.

Architecture of Operating System

3) Microkernel Architecture:

- In this architecture, the components like process management, networking, file system interaction, and device management are executed outside the kernel
- while memory management and synchronization are executed inside the kernel.
- The processes inside the kernel have relatively high priority, the components possess high modularity hence even if one or more components fail the operating system keeps on working.
- Example: Linux and Windows XP contain Modular components.

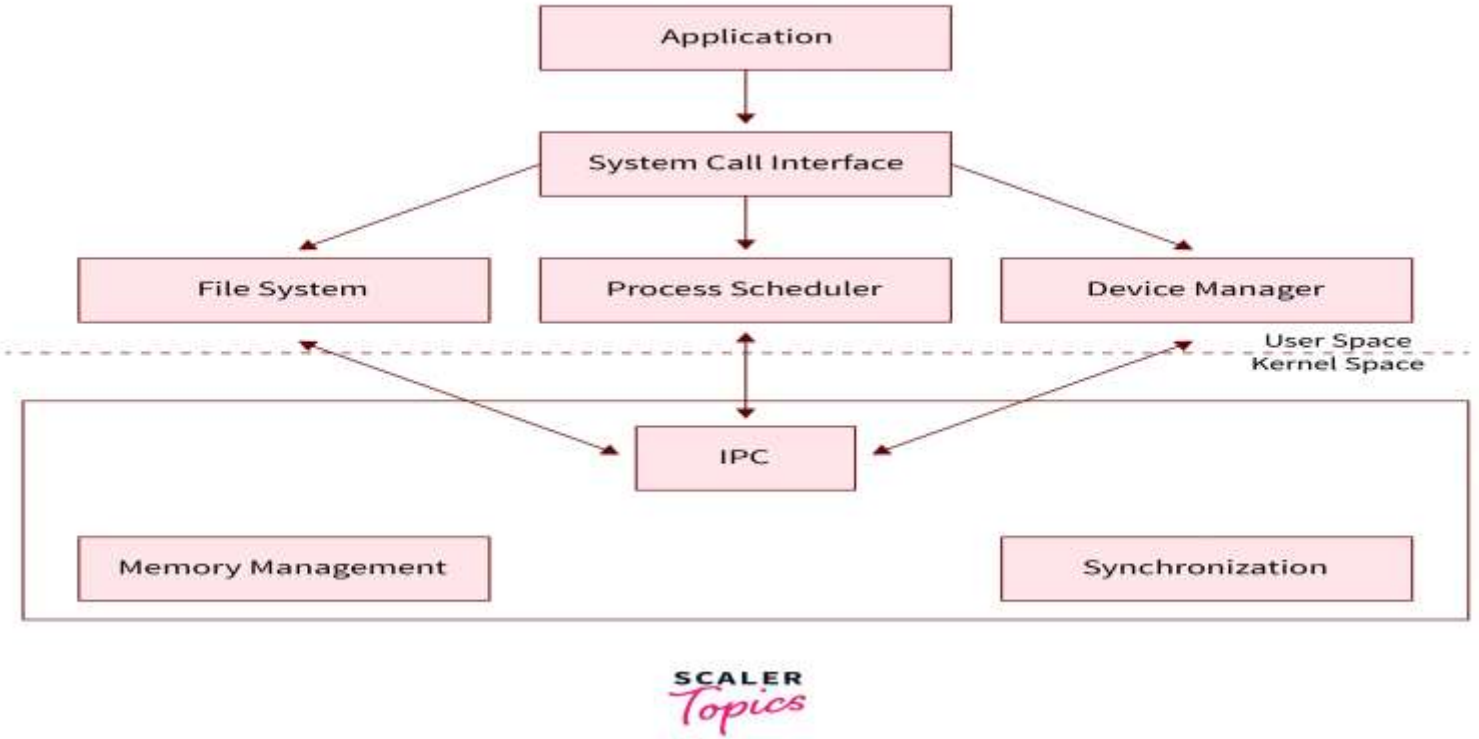
Advantages:

- Microkernel operating systems are modular and hence, disturbing one of the components will not affect the other component.
- The architecture is compact and isolated and hence relatively efficient.
- New features can be added without recompilation.

Architecture of Operating System

Disadvantages:

- Implementing drivers as procedures require a function call or context switch.
- In microkernel architecture, providing services is costlier than monolithic operating systems.



Architecture of Operating System

4) Hybrid Architecture:

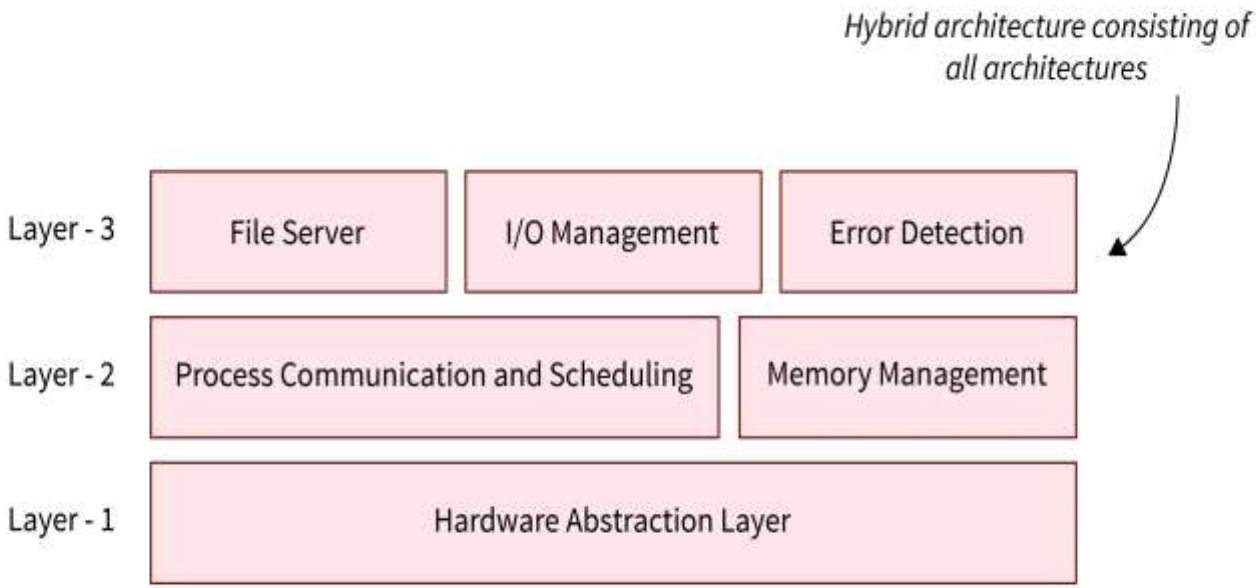
- Hybrid architecture as the name suggests consists of a hybrid of all the architectures explained so far and hence it has properties of all of those architectures which makes it highly useful in present-day operating systems.
- **The hybrid-architecture consists of three layers**

1) Hardware abstraction layer: It is the interface between the kernel and hardware and is present at the lowest level.

2) Microkernel Layer: This is the old microkernel that we know and it consists of CPU scheduling, memory management, and inter-process communication.

3) Application Layer: It acts as an interface between the user and the microkernel. It contains the functionalities like a file server, error detection, I/O device management, etc.

Architecture of Operating System



SCALER Topics

Architecture of Operating System

Advantages:

- Since it is a hybrid of other architectures it allows various architectures to provide their services respectively.
- It is easy to manage because it uses a layered approach.
- Number of layers is relatively lesser.
- Security and protection are relatively improved.

Disadvantage:

- Hybrid architecture of the operating system keeps certain services in the kernel space while moving less critical services to the user space.

Types of Operating System

- Operating systems are there from the very first computer generation and they keep evolving with time.

Batch operating system:

- The users of a 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.
- The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches

The problems with Batch Systems are as follows:

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

Types of Operating System

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.
- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently.
- Thus, the user can receive an immediate response.
- For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation.
- That is, if n users are present, then each user can get a time quantum.
- When the user submits the command, the response time is in few seconds at most.
- The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time.

Types of Operating System

Advantages of Timesharing operating systems are as follows –

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

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Types of Operating System

Multiprogramming Operating System:

- A multiprogramming operating system may run many programs on a single processor computer.
- If one program must wait for an input/output transfer in a multiprogramming operating system, the other programs are ready to use the CPU.
- As a result, various jobs may share CPU time.
- However, the execution of their jobs is not defined to be at the same time period.
- When a program is being performed, it is known as a "Task", "Process", and "Job".
- Concurrent program executions improve system resource consumption and throughput as compared to serial and batch processing systems.
- The primary goal of multiprogramming is to manage the entire system's resources.
- The key components of a multiprogramming system are the file system, command processor, transient area, and I/O control system.

Types of Operating System

Types of the Multiprogramming Operating System:

Multitasking Operating System:

- A multitasking operating system enables the execution of two or more programs at the same time.
- The operating system accomplishes this by shifting each program into and out of memory one at a time.
- When a program is switched out of memory, it is temporarily saved on disk until it is required again.

Multiuser Operating System:

- A multiuser operating system allows many users to share processing time on a powerful central computer from different terminals.
- The operating system accomplishes this by rapidly switching between terminals, each of which receives a limited amount of processor time on the central computer.
- The operating system changes among terminals so quickly that each user seems to have continuous access to the central computer.
- If there are many users on a system like this, the time it takes the central computer to reply can become more obvious.

Types of Operating System

Distributed operating System:

- Distributed systems use multiple central processors to serve multiple real-time applications and multiple users.
- Data processing jobs are distributed among the processors accordingly.
- The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines).
- These are referred as loosely coupled systems or distributed systems.
- Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

- With resource sharing facility, a 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.

Types of Operating System

Real Time operating System:

- A real-time system is defined 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.
- The time taken by the system to respond to an input and display of required updated information is termed as the response time.
- So in this method, the response time is very less as compared to 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.
- A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail.
- For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Types of Operating System

There are two types of real-time operating systems:

Hard real-time systems:

- Hard real-time systems guarantee that critical tasks complete on time.
- In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM.
- In these systems, virtual memory is almost never found.

Soft real-time systems:

- Soft real-time systems are less restrictive.
- A critical real-time task gets priority over other tasks and retains the priority until it completes.
- Soft real-time systems have limited utility than hard real-time systems.
- For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

System Calls in Operating System

- A system call is a way for a user program to interface with the operating system.
- The program requests several services, and the OS responds by invoking a series of system calls to satisfy the request.
- A system call can be written in assembly language or a high-level language like C or Pascal.
- System calls are predefined functions that the operating system may directly invoke if a high-level language is used.
- A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running.
- A system call is a method of interacting with the operating system via programs.
- A system call is a request from computer software to an operating system's kernel.

System Calls in Operating System

How are system calls made?

- When a computer software needs to access the operating system's kernel, it makes a system call.
- The system call uses an API to expose the operating system's services to user programs.
- It is the only method to access the kernel system.
- All programs or processes that require resources for execution must use system calls, as they serve as an interface between the operating system and user programs.

Below are some examples of how a system call varies from a user function :

- A system call function may create and use kernel processes to execute the asynchronous processing.
- A system call has greater authority than a standard subroutine. A system call with kernel-mode privilege executes in the kernel protection domain.
- System calls are not permitted to use shared libraries or any symbols that are not present in the kernel protection domain.
- The code and data for system calls are stored in global kernel memory.

System Calls in Operating System

Why do you need system calls in Operating System?

Following of the situations are as follows:

- It is must require when a file system wants to create or delete a file.
- Network connections require the system calls to sending and receiving data packets.
- If you want to read or write a file, you need to system calls.
- If you want to access hardware devices, including a printer, scanner, you need a system call.
- System calls are used to create and manage new processes.

System Calls in Operating System

Types of System Calls: There are commonly five types of system calls.

Process Control: Process control is the system call that is used to direct the processes.

- Some process control examples include creating, load, abort, end, execute, process, terminate the process, etc.

File Management: File management is a system call that is used to handle the files.

- Some file management examples include creating files, delete files, open, close, read, write, etc.

Device Management: Device management is a system call that is used to deal with devices.

- Some examples of device management include read, device, write, get device attributes, release device, etc.

Information Maintenance: Information maintenance is a system call that is used to maintain information.

- There are some examples of information maintenance, including getting system data, set time or date, get time or date, set system data, etc.

System Calls in Operating System

Communication: Communication is a system call that is used for communication.

- There are some examples of communication, including create, delete communication connections, send, receive messages, etc.

Examples of Windows and Unix system calls :

Process	Windows	Unix
Process Control	CreateProcess(), ExitProcess(), WaitForSingleObject()	Fork(), Exit(), Wait()
File Manipulation	CreateFile(), ReadFile(), WriteFile(), CloseHandle()	Open(), Read(), Write(), Close()
Device Management	SetConsoleMode(), ReadConsole(), WriteConsole()	Ioctl(), Read(), Write()
Information Maintenance	GetCurrentProcessID(), SetTimer(), Sleep()	Getpid(), Alarm(), Sleep()

System Calls in Operating System

Examples of Windows and Unix system calls :

Process	Windows	Unix
Communication	CreatePipe(), CreateFileMapping()MapViewOfFile()	Pipe(), Shmget(), Mmap()
Protection	SetFileSecurity(), InitializeSecurityDescriptor(), SetSecurityDescriptorgroup()	Chmod() Umask(), Chown()

Unix / Linux - What is Shells?

- A **Shell** provides you with an interface to the Unix system.
- It gathers input from you and executes programs based on that input.
- When a program finishes executing, it displays that program's output.
- Shell is an environment in which we can run our commands, programs, and shell scripts.
- There are different flavors of a shell, just as there are different flavors of operating systems.
- Each flavor of shell has its own set of recognized commands and functions.

Unix / Linux - What is Shells?

Shell Prompt:

- The prompt, $, which is called the command prompt, is issued by the shell.
- While the prompt is displayed, you can type a command.
- Shell reads your input after you press Enter.
- It determines the command you want executed by looking at the first word of your input.
- A word is an unbroken set of characters.
- Spaces and tabs separate words.
- Following is a simple example of the date command, which displays the current date and time −
- **$date**
- Thu Dec 29 08:30:19 MST 2022

Unix / Linux - What is Shells?

Shell Types: In Unix, there are two major types of shells –

- **Bourne shell –** If you are using a Bourne-type shell, the $ character is the default prompt.
  - The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.
  - Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".
  - Bourne shell is usually installed as /bin/sh on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.
- **C shell –** If you are using a C-type shell, the % character is the default prompt.

Unix / Linux - What is Shells?

Types of Bourne shell :

Bourne Again shell (bash): In the bash shell, bash means Bourne Again Shell.

- It is a default shell over several distributions of Linux today.
- It is a sh-compatible shell.
- It could be installed over Windows OS.
- It facilitates practical improvements on sh for interactive and programming use which contains:
  - Job Control
  - Command-line editing
  - Shell Aliases and Functions
  - Unlimited size command history
  - Integer arithmetic in a base from 2-64

Unix / Linux - What is Shells?

Korn shell (ksh):

- Ksh means for Korn shell.
- It was developed and designed by David G. Korn.
- Ksh shell is a high-level, powerful, and complete programming language and it is a reciprocal command language as well just like various other GNU/Unix Linux shells.
- The usage and syntax of the K shell are very same as the C programming language.

Zsh Shell:

- Zsh shell is developed to be reciprocal and it combines various aspects of other GNU/Unix Linux shells like ksh, tcsh, and bash.
- Also, the POSIX shell standard specifications were based on the Korn shell.
- Also, it is a strong scripting language like other available shells.

Unix / Linux - What is Shells?

Some of its unique features are listed as follows:

- Startup files
- Filename generation
- Login/Logout watching
- Concept index
- Closing comments
- Variable index
- Key index

Fish: Fish stands for "friendly interactive shell".

  - It was produced in 2005.
  - Fish shell was developed to be fully user-friendly and interactive just like other shells.
- It contains some good features which are mentioned below:
  - Web-based configuration
  - Man page completions
  - Auto-suggestions
  - Support for term256 terminal automation
  - Completely scripted with clean scripts

Basic Shell Commands in Linux

- A shell is a special user program that provides an interface to the user to use operating system services.
- Shell accepts human-readable commands from the user and converts them into something which the kernel can understand.

Displaying the file contents on the terminal:

cat: It is generally used to concatenate the files. It gives the output on the standard output.

more: It is a filter for paging through text one screenful at a time.

less: It is used to viewing the files instead of opening the file.Similar to more command but it allows backward as well as forward movement.

head : Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

tail : Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

Basic Shell Commands in Linux

2). File and Directory Manipulation Commands:

mkdir : Used to create a directory if not already exist. It accepts the directory name as an input parameter.

cp : This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.

mv : Used to move the files or directories. This command's working is almost similar to cp command but it deletes a copy of the file or directory from the source path.

rm : Used to remove files or directories.

touch : Used to create or update a file.

Basic Shell Commands in Linux

3). Extract, sort, and filter data Commands:

grep : This command is used to search for the specified text in a file.

grep with Regular Expressions: Used to search for text using specific regular expressions in file.

sort : This command is used to sort the contents of files.

wc : Used to count the number of characters, words in a file.

cut : Used to cut a specified part of a file.

4). Basic Terminal Navigation Commands:

ls : To get the list of all the files or folders.

ls -l: Optional flags are added to ls to modify default behavior, listing contents in extended form -l is used for “long” output

ls -a: Lists of all files including the hidden files, add -a flag

cd: Used to change the directory.

du: Show disk usage.

pwd: Show the present working directory.

Basic Shell Commands in Linux

4). Basic Terminal Navigation Commands: Continue

man: Used to show the manual of any command present in Linux.

rmdir: It is used to delete a directory if it is empty.

ln file1 file2: Creates a physical link.

ln -s file1 file2: Creates a symbolic link.

locate: It is used to locate a file in Linux System

echo: This command helps us move some data, usually text into a file.

df: It is used to see the available disk space in each of the partitions in your system.

tar: Used to work with tarballs (or files compressed in a tarball archive)

5). File Permissions Commands: The chmod and chown commands are used to control access to files in UNIX and Linux systems.

chown : Used to change the owner of the file.

chgrp : Used to change the group owner of the file.

chmod : Used to modify the access/permission of a user.

Shell programming in Linux

4).

Shell programming in Linux

4).

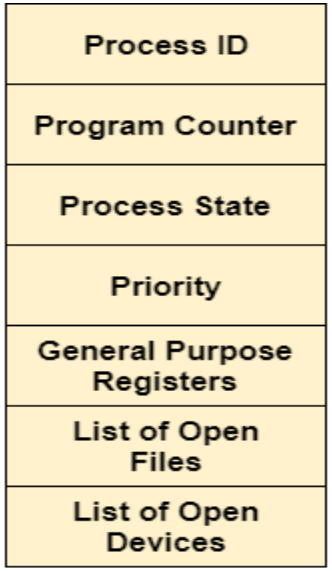
Process Management

- A Program does nothing unless its instructions are executed by a CPU.
- A program in execution is called a process.
- In order to accomplish its task, process needs the computer resources.
- A process is basically a program in execution.
- The execution of a process must progress in a sequential fashion.
- There may exist more than one process in the system which may require the same resource at the same time.
- Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.
- Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.
- The operating system is responsible for the following activities in connection with Process Management
  - Scheduling processes and threads on the CPUs.
  - Creating and deleting both user and system processes.
  - Suspending and resuming processes.
  - Providing mechanisms for process synchronization.
  - Providing mechanisms for process communication.

Process Management

Attributes of a process:

- The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them.
- This is also called context of the process.
- Attributes which are stored in the PCB are described below.



Process Attributes

Process Management

1. Process ID: When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter: A program counter stores the address of the last instruction of the process on which the process was suspended.

- The CPU uses this address when the execution of this process is resumed.

3. Process State: The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting.

- We will discuss about them later in detail.

4. Priority: Every process has its own priority.

•The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block**.**

5. General Purpose Registers: Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

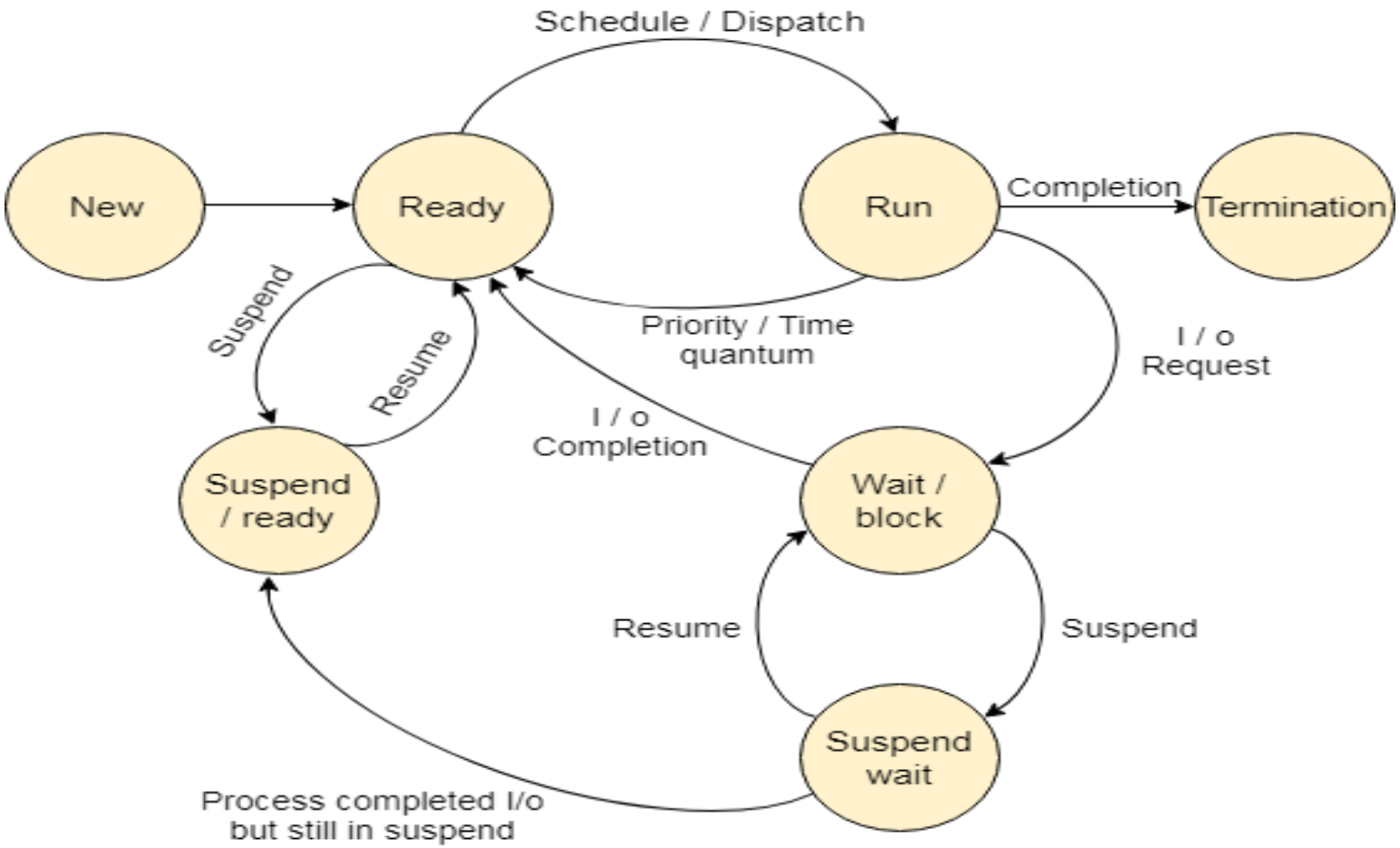
6. List of open files: During the Execution, Every process uses some files which need to be present in the main memory.

- OS also maintains a list of open files in the PCB.

7. List of open devices: OS also maintain the list of all open devices which are used during the execution of the process.

Process Management

Process States: The process, from its creation to completion, passes through various states. The minimum number of states is five.



Process Management

1. New: A program which is going to be picked up by the OS into the main memory is called a new process.

2. Ready: Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned.

- The OS picks the new processes from the secondary memory and put all of them in the main memory.
- The processes which are ready for the execution and reside in the main memory are called ready state processes.
- There can be many processes present in the ready state.

3. Running: One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm.

- Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one.
- If we have n processors in the system then we can have n processes running simultaneously.

Process Management

4. Block or wait: From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

- When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination: When a process finishes its execution, it comes in the termination state.

- All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready: A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

- If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory.
- The suspend ready processes remain in the secondary memory until the main memory gets available.

Process Management

7. Suspend wait: Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory.

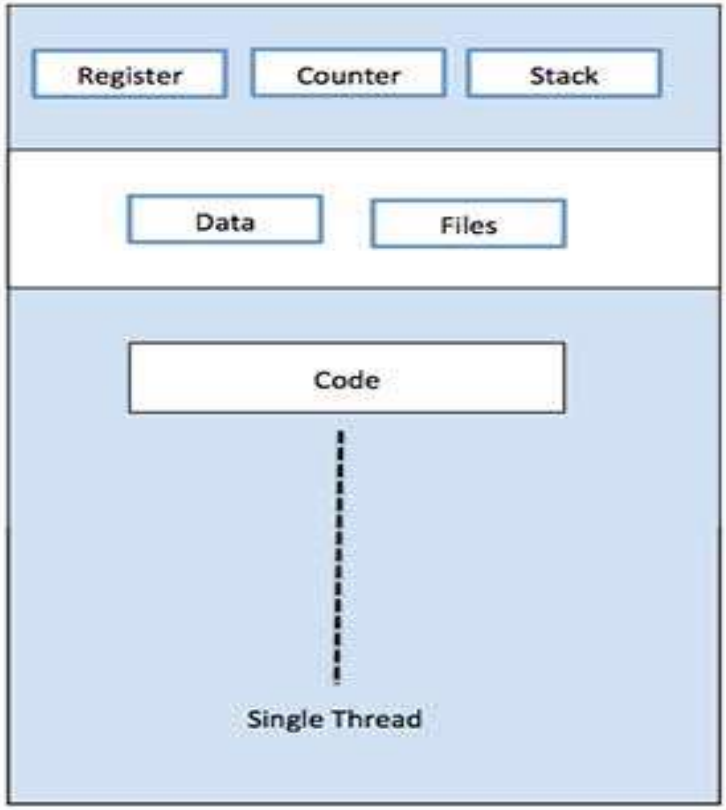
- Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process.
- These processes complete their execution once the main memory gets available and their wait is finished.

Threads

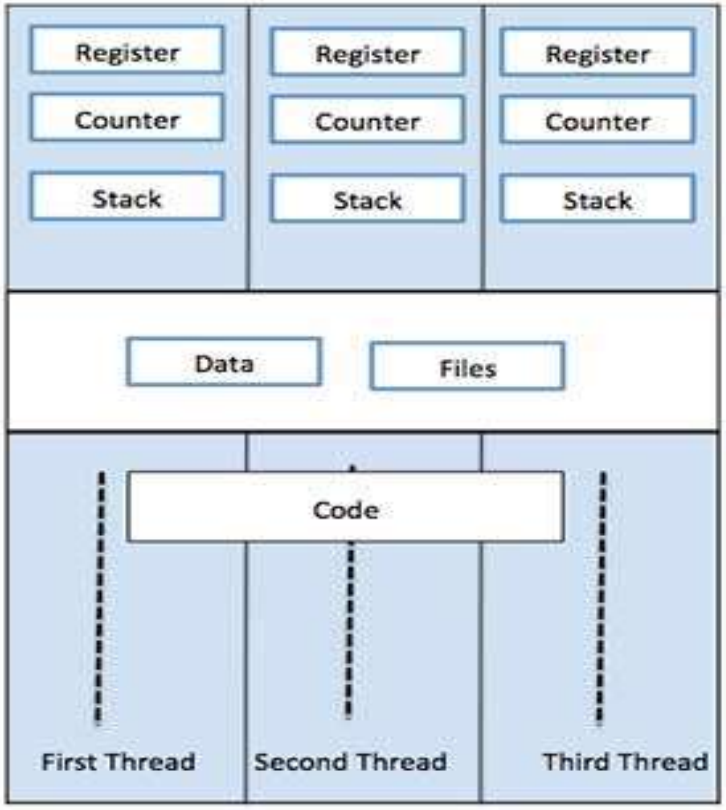
- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files.
- When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a lightweight process.
- Threads provide a way to improve application performance through parallelism.
- 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.
- Threads have been successfully used in implementing network servers and web server.

Threads

- They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.
- The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Threads

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.

Threads

Types of Thread:

- **Threads are implemented in following two ways**

User Level Threads:

- In this case, the thread management 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 starts with a single thread.

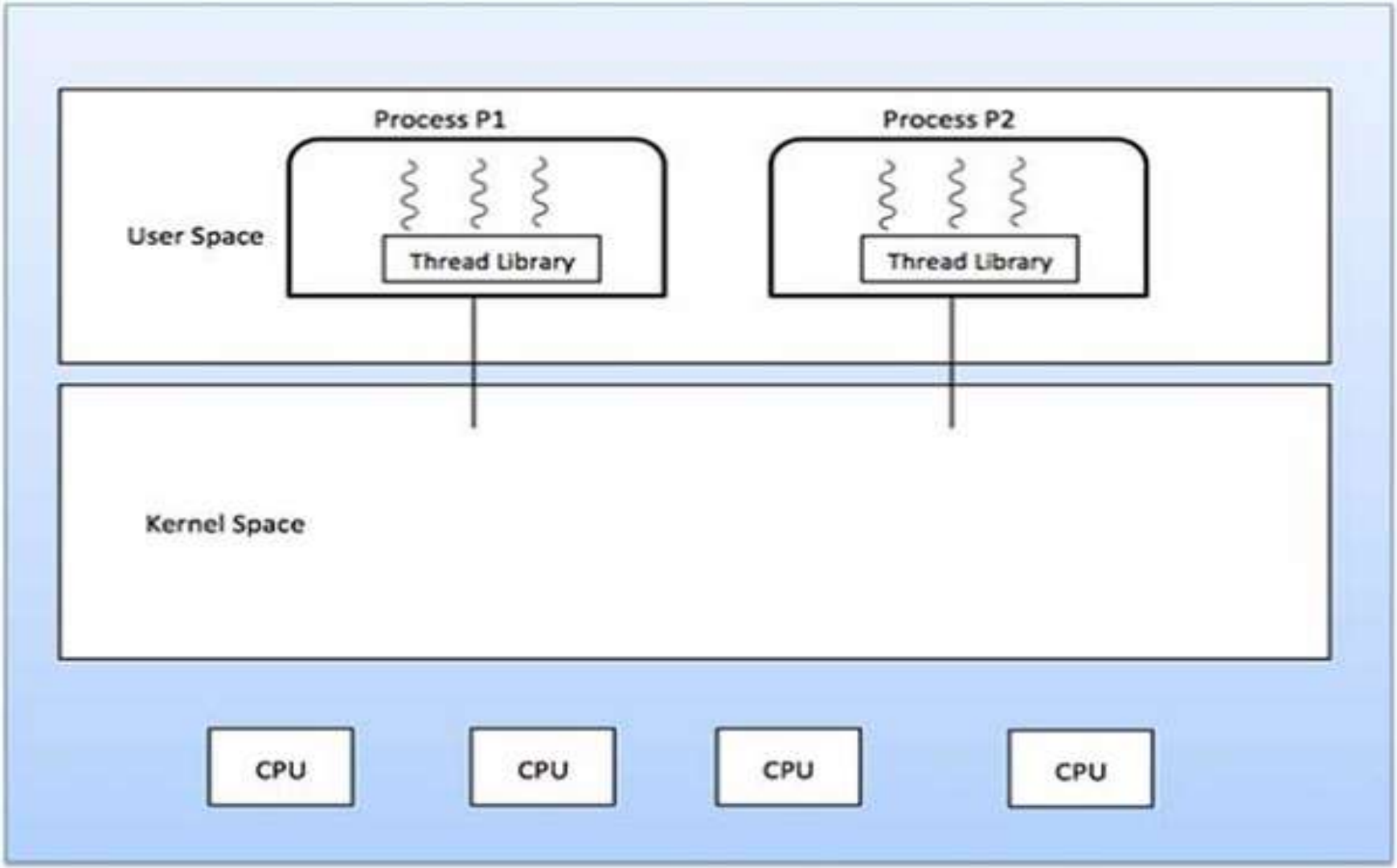
Advantages:

- 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 multipro

Threads


Process P1
Process P2
User Space
Thread Library
Thread Library
Kernel Space
CPU
CPU
CPU
CPU

Threads

Kernel Level Threads:

- In this case, thread management is 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 individuals 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.

Threads

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.

Threads

Multithreading Models:

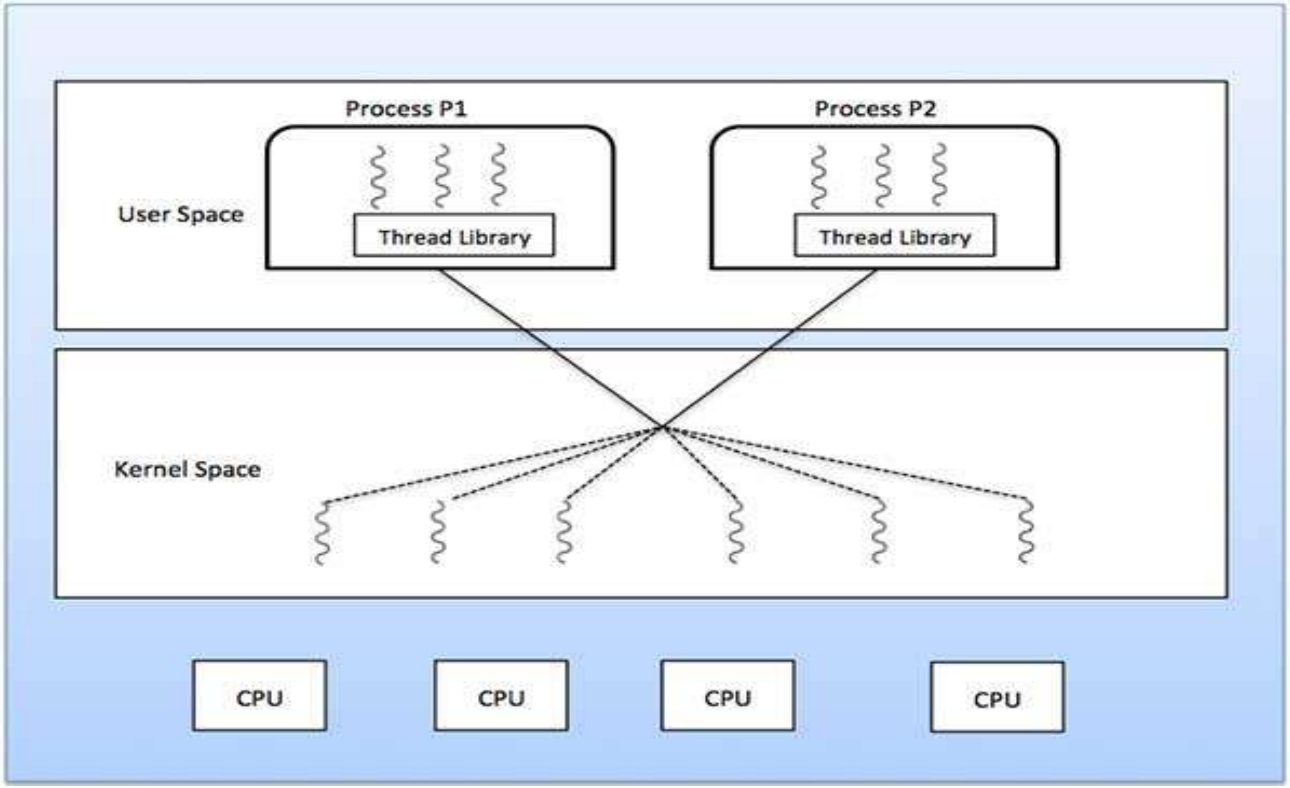
- Some operating system provide a combined user level thread and Kernel level thread facility.
- Solaris is a good example of this combined approach.
- In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.
- Multithreading models are three types:
  - Many to many relationship.
  - Many to one relationship.
  - One to one relationship.

Many to Many Model: The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.

Threads

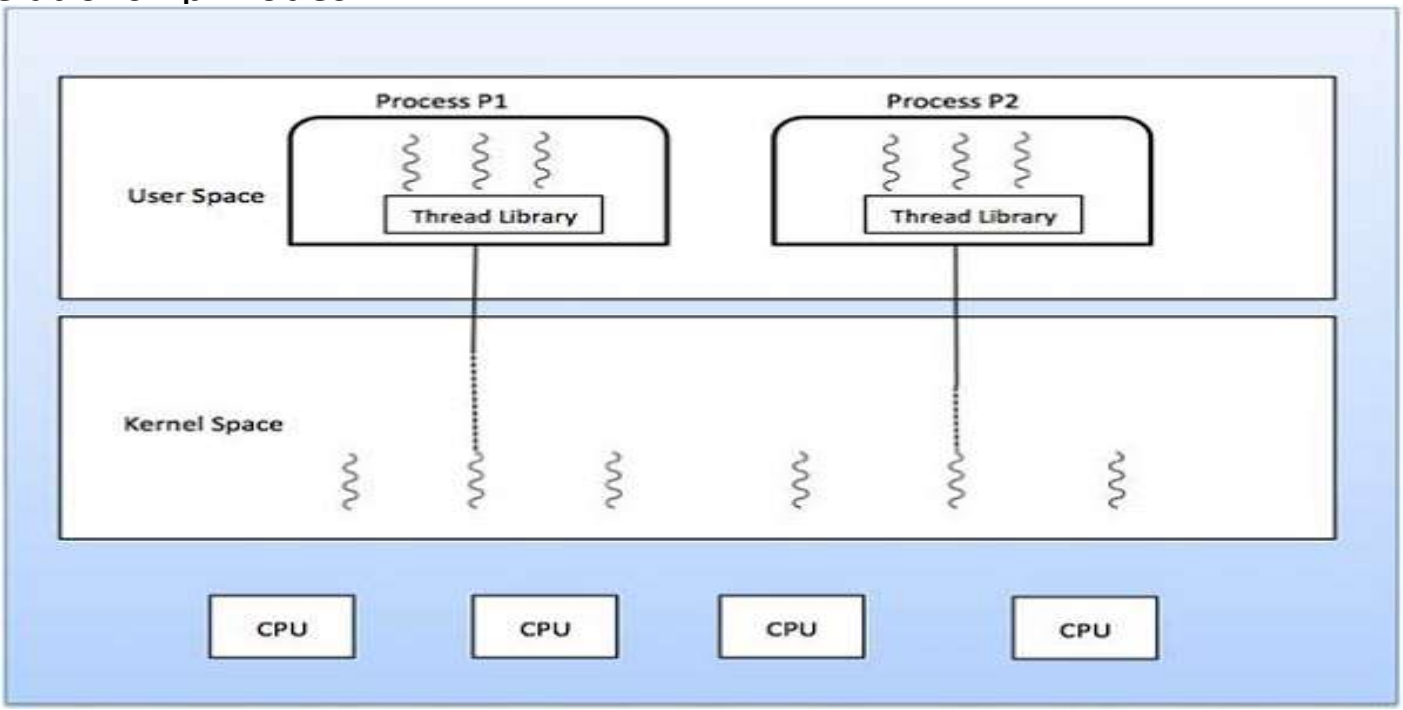
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Threads

Many to One Model:

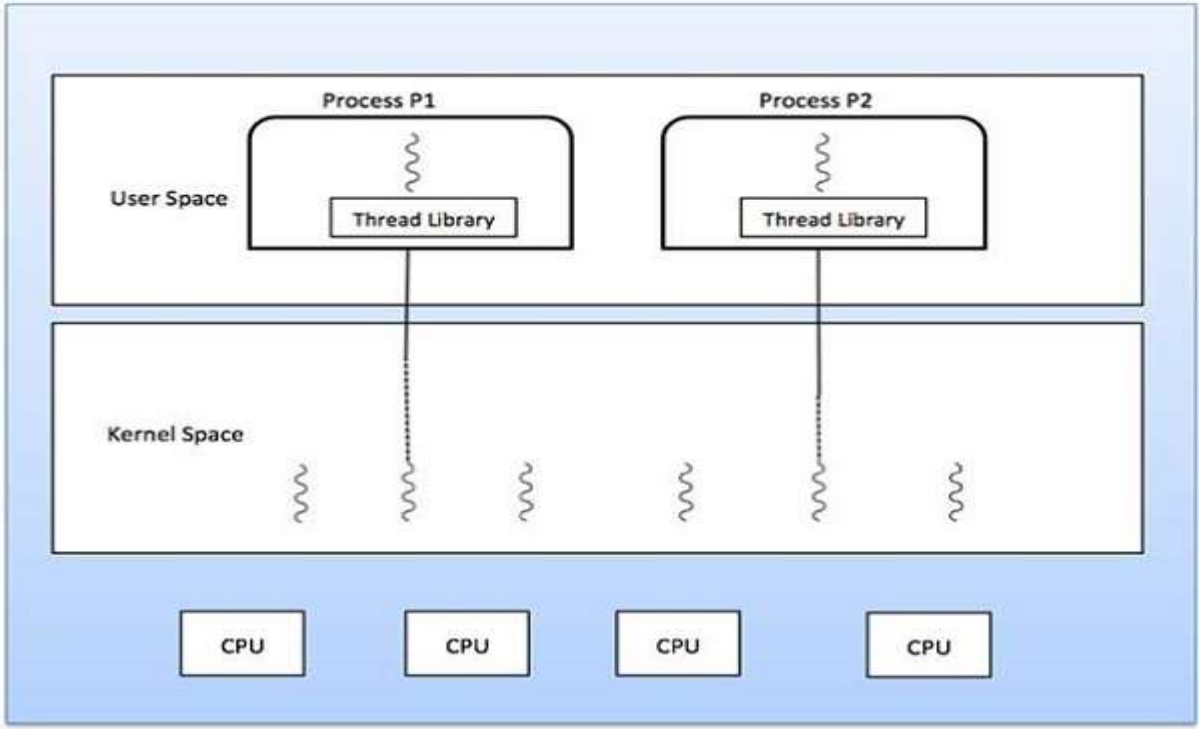
- Many-to-one model maps many user level threads to one Kernel-level thread.
- Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked.
- Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



Threads

One to One Model: There is one-to-one relationship of user-level thread to the kernel-level thread.

- This model provides more concurrency than the many-to-one model.
- It also allows another thread to run when a thread makes a blocking system call.
- It supports multiple threads to execute in parallel on microprocessors.
- Disadvantage of this model is that creating user thread requires the corresponding Kernel thread.
- OS/2, windows NT and windows 2000 use one to one relationship model.



Threads

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.

Process Scheduling in Operating System

Definition: The process scheduling is the activity 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 a particular strategy.

- Process scheduling is an essential part of a Multiprogramming operating systems.
- Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.
- Categories of Scheduling: There are two categories of scheduling

Non-preemptive: Here the resource can't be taken from a process until the process completes execution.

- The switching of resources occurs when the running process terminates and moves to a waiting state.

Preemptive: Here the OS allocates the resources to a process for a fixed amount of time.

- During resource allocation, the process switches from running state to ready state or from waiting state to ready state.
- This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

Process Scheduling in Operating System

Process Scheduling Queues:

•The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues.

•The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.

•When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

•**The Operating System maintains the following important process scheduling queues –**

•**Job queue –** This queue keeps all the processes in the system.

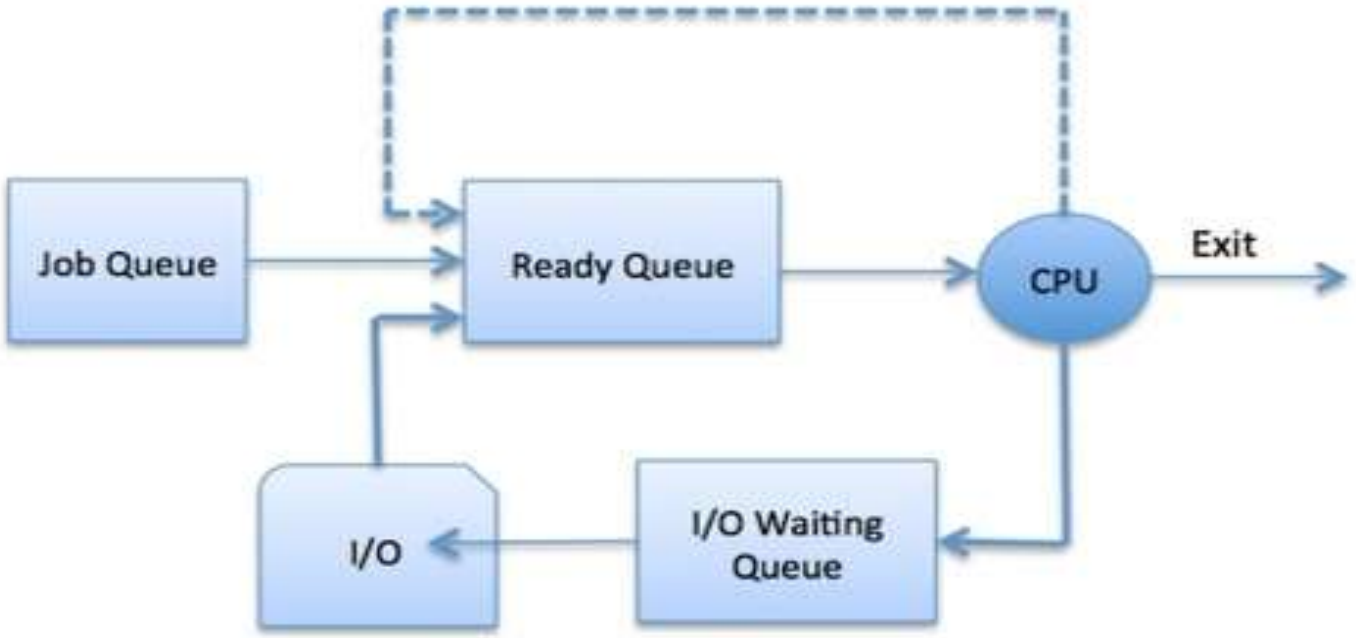
•**Ready queue –** This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

•**Device queues –** The processes which are blocked due to unavailability of an I/O device constitute this queue.

Process Scheduling in Operating System

Process Scheduling Queues:

- The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.).
- The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the below diagram, it has been merged with the CPU.



Process Scheduling in Operating System

Schedulers:

- Schedulers are special system software which handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types −
  - Long-Term Scheduler
  - Short-Term Scheduler
  - Medium-Term Scheduler

Long Term Scheduler:

- It is also called a job scheduler.
- A long-term scheduler determines which programs are admitted to the system for processing.
- It selects processes from the queue and loads them into memory for execution.
- Process loads into the memory for CPU scheduling.
- 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

Process Scheduling in Operating System

- 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.
- Time-sharing operating systems have no long term scheduler.
- When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler:

- It is also called as CPU scheduler.
- Its main objective is to increase 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 a process among the processes that are ready to execute and allocates CPU to one of them.
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next.
- Short-term schedulers are faster than long-term schedulers.

Process Scheduling in Operating System

Medium Term Scheduler:

- Medium-term scheduling is a part of swapping.
- 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.
- A running process may become suspended if it makes an I/O request.
- A suspended processes cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- This process is called swapping, and the process is said to be swapped out or rolled out.
- Swapping may be necessary to improve the process mix.

Process Scheduling in Operating System

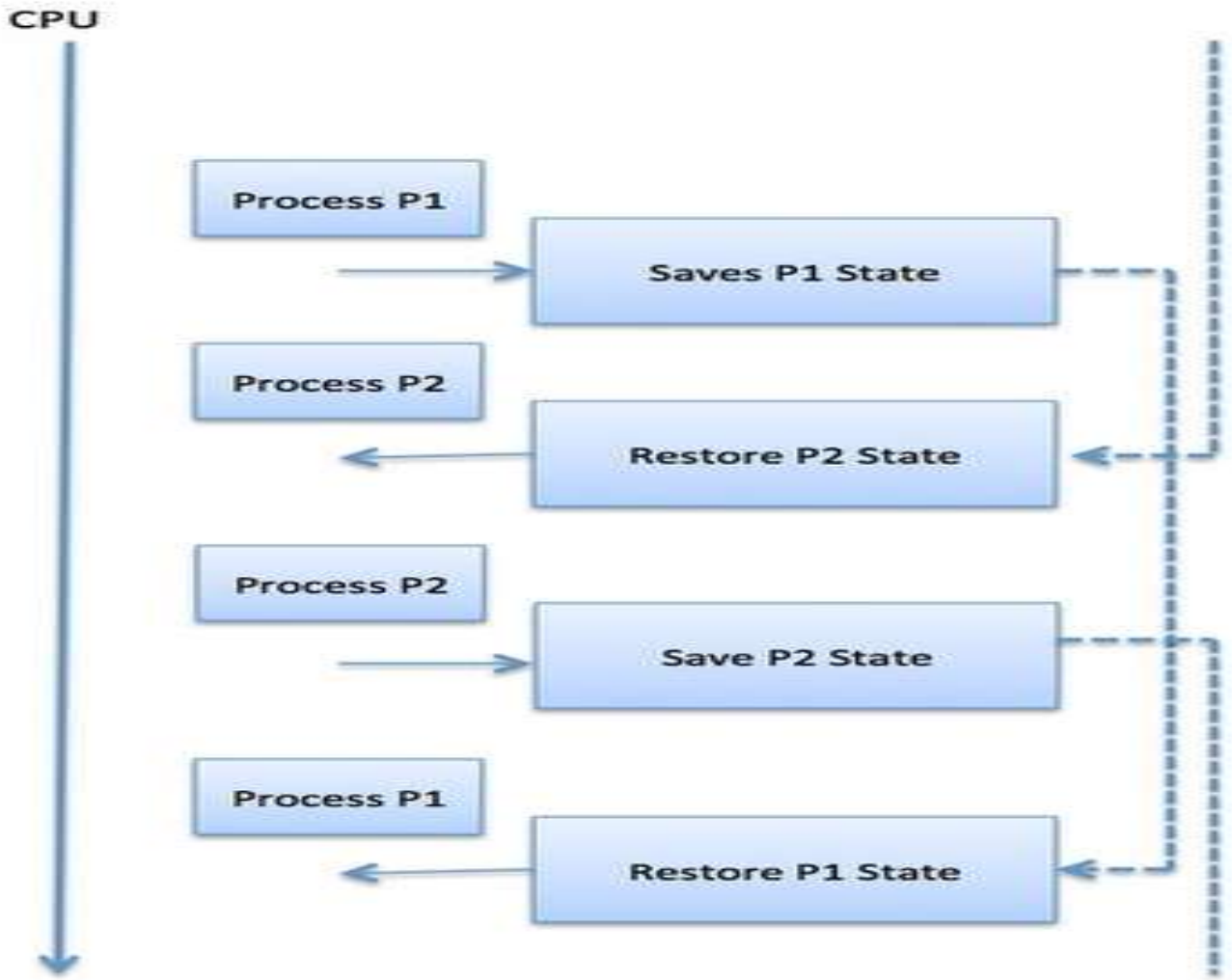
- Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switching

- **Context Switching:**
- A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block.
- After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc.
- At that point, the second process can start executing.

Process Scheduling in Operating System



Process Scheduling in Operating System

- Context switches are computationally intensive since register and memory state must be saved and restored.
- To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers.
- When the process is switched, the following information is stored for later use.
  - Program Counter
  - Scheduling information
  - Base and limit register value
  - Currently used register
  - Changed State
  - I/O State information
  - Accounting information

Scheduling algorithms

- A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.
- **The Purpose of a Scheduling algorithm:**
  - Maximum CPU utilization
  - Fare allocation of CPU
  - Maximum throughput
  - Minimum turnaround time
  - Minimum waiting time
  - Minimum response time

Scheduling Criteria

- Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another.
- In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

CPU utilization:

- We want to keep the CPU as busy as possible.
- Conceptually, CPU utilization can range from 0 to 100 percent.
- In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).

Throughput:

- If the CPU is busy executing processes, then work is being done.
- One measure of work is the number of processes that are completed per time unit, called throughput.
- For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

Scheduling Criteria

Turnaround time:

- From the point of view of a particular process, the important criterion is how long it takes to execute that process.
- The interval from the time of submission of a process to the time of completion is the turnaround time.
- Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Waiting time:

- The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O.
- It affects only the amount of time that a process spends waiting in the ready queue.
- Waiting time is the sum of the periods spent waiting in the ready queue.

Response time:

- In an interactive system, turnaround time may not be the best criterion.
- Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user.

Scheduling Criteria

Response time: Continue..

- Thus, another measure is the time from the submission of a request until the first response is produced.
- This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.

FCFS Scheduling Algorithm

- First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time.
- The job which comes first in the ready queue will get the CPU first.
- The lesser the arrival time of the job, the sooner will the job get the CPU.
- FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS:

- Simple
- Easy
- First come, First serv

Disadvantages of FCFS:

- The scheduling method is non preemptive, the process will run to the completion.
- Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
- Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

FCFS Scheduling Algorithm

- Q. 1 for Given data find Completion Time, Turn Around Time and Waiting Time for each process?

Process ID	Arrival Time	Burst Time
P0	0	2
P1	1	6
P2	2	4
P3	3	9
P4	6	12

Solution: The Turnaround time and the waiting time are calculated by using the following formula.

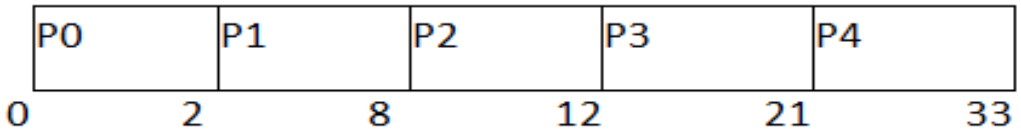
Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turnaround time - Burst Time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P0	0	2	2	2	0
P1	1	6	8	7	1
P2	2	4	12	10	6
P3	3	9	21	18	9
P4	6	12	33	29	17

FCFS Scheduling Algorithm

Avg Waiting Time=31/5



Gantt chart

Convoy Effect in FCFS:

- FCFS may suffer from the convoy effect if the burst time of the first job is the highest among all.
- If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time.
- This is called convoy effect or starvation.

Shortest Job First (SJF) Scheduling Algorithm

- SJF scheduling algorithm, schedules the processes according to their burst time.
- In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.
- However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF:

- Maximum throughput
- Minimum average waiting and turnaround time

Disadvantages of SJF:

- May suffer with the problem of starvation
- It is not implementable because the exact Burst time for a process can't be known in advance.

Shortest Job First (SJF) Scheduling Algorithm

- In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

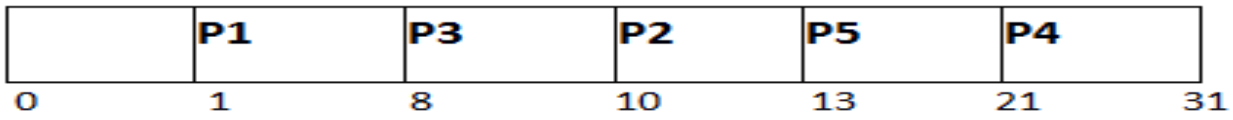
PID	Arrival Time	Burst Time
1	1	7
2	3	3
3	6	2
4	7	10
5 `	9	8

- Since, No Process arrives at time 0 hence; there will be an empty slot in the Gantt chart from time 0 to 1 (the time at which the first process arrives).

- According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Shortest Job First (SJF) Scheduling Algorithm

PID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5 `	9	8	21	12	4



Gantt chart

Round Robin Scheduling Algorithm

- Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems.
- This is the preemptive version of first come first serve scheduling.
- The Algorithm focuses on Time Sharing.
- In this algorithm, every process gets executed in a cyclic way.
- A certain time slice is defined in the system which is called time quantum.
- Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Advantages:

- It can be actually implementable in the system because it is not depending on the burst time.
- It doesn't suffer from the problem of starvation or convoy effect.
- All the jobs get a fare allocation of CPU.

Disadvantages :

- The higher the time quantum, the higher the response time in the system.
- The lower the time quantum, the higher the context switching overhead in the system.
- Deciding a perfect time quantum is really a very difficult task in the system

Round Robin Scheduling Algorithm

- In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

Process ID	Arrival Time	Burst Time
1	0	5
2	1	6
3	2	3
4	3	1
5	4	5
6	6	4

Round Robin Scheduling Algorithm

- he completion time, Turnaround time and waiting time will be calculated as shown in the table below. As, we know,

Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	5	17	17	12
2	1	6	23	22	16
3	2	3	11	9	6
4	3	1	12	9	8
5	4	5	24	20	15
6	6	4	21	15	11

Avg Waiting Time = (12+16+6+8+15+11)/6 = 76/6 units

Priority Scheduling Algorithm

- In Priority scheduling, there is a priority number assigned to each process.
- In some systems, the lower the number, the higher the priority.
- While, in the others, the higher the number, the higher will be the priority.
- The Process with the higher priority among the available processes is given the CPU.
- There are two types of priority scheduling algorithm exists.
- **One is Preemptive priority scheduling** while the other is **Non Preemptive Priority scheduling.**
- The priority number assigned to each of the process may or may not vary.
- If the priority number doesn't change itself throughout the process, it is called static priority, while if it keeps changing itself at the regular intervals, it is called dynamic priority.

Priority Scheduling Algorithm

Non Preemptive Priority Scheduling:

- In the Non Preemptive Priority scheduling, The Processes are scheduled according to the priority number assigned to them.
- Once the process gets scheduled, it will run till the completion.
- Generally, the lower the priority number, the higher is the priority of the process.
- In this **Example**, there are 7 processes P1, P2, P3, P4, P5, P6 and P7. Their priorities, Arrival Time and burst time are given in the table.

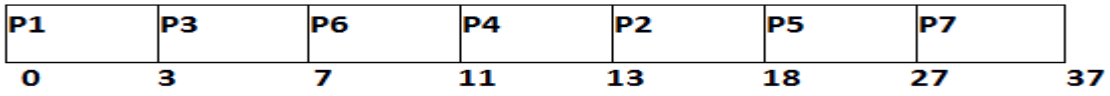
Process ID	Priority	Arrival Time	Burst Time
1	2	0	3
2	6	2	5
3	3	1	4
4	5	4	2
5	7	6	9
6	4	5	4
7	10	7	10

Priority Scheduling Algorithm

Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

Process Id	Priority	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
1	2	0	3	3	3	0	0
2	6	2	5	18	16	11	13
3	3	1	4	7	6	2	3
4	5	4	2	13	9	7	11
5	7	6	9	27	21	12	18
6	4	5	4	11	6	2	7
7	10	7	10	37	30	18	27



Gantt Chart

Priority Scheduling Algorithm

Preemptive Priority Scheduling:

- In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time.
- The One with the highest priority among all the available processes will be given the CPU next.
- The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.
- Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

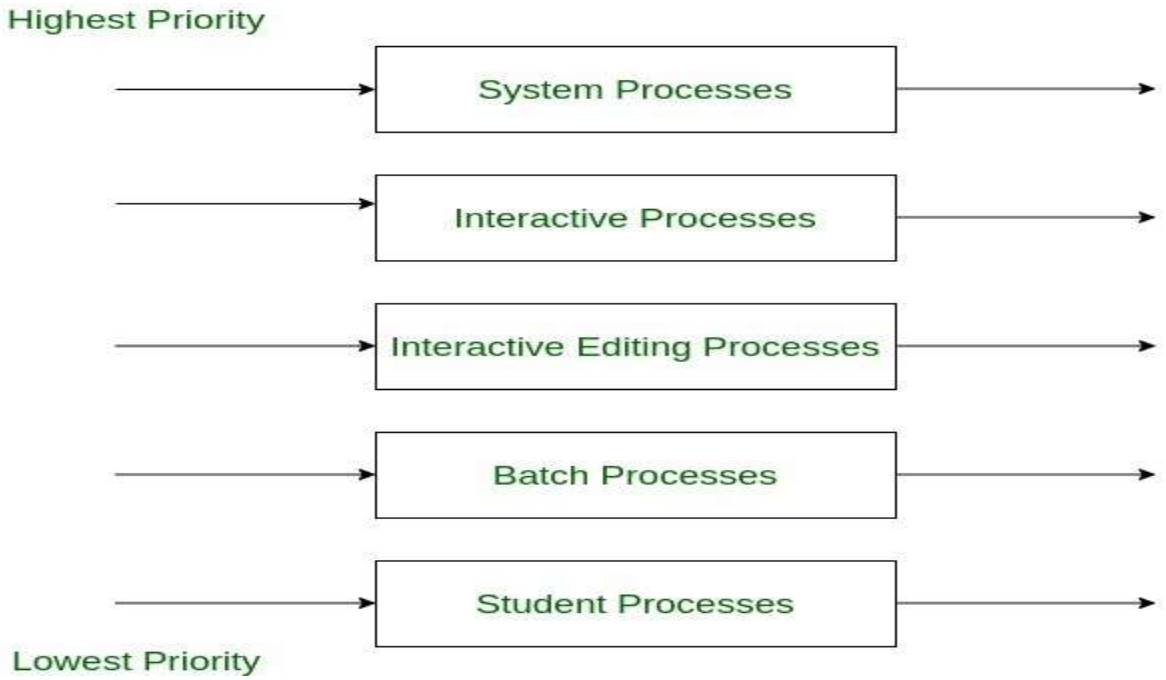
Multilevel Queue Scheduling

- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues .
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm.
- For example, separate queues might be used for foreground and background processes.
- The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.
- For example, the foreground queue may have absolute priority over the background queue.

Multilevel Queue Scheduling

- Let's look at an example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes



Multilevel Queue Scheduling

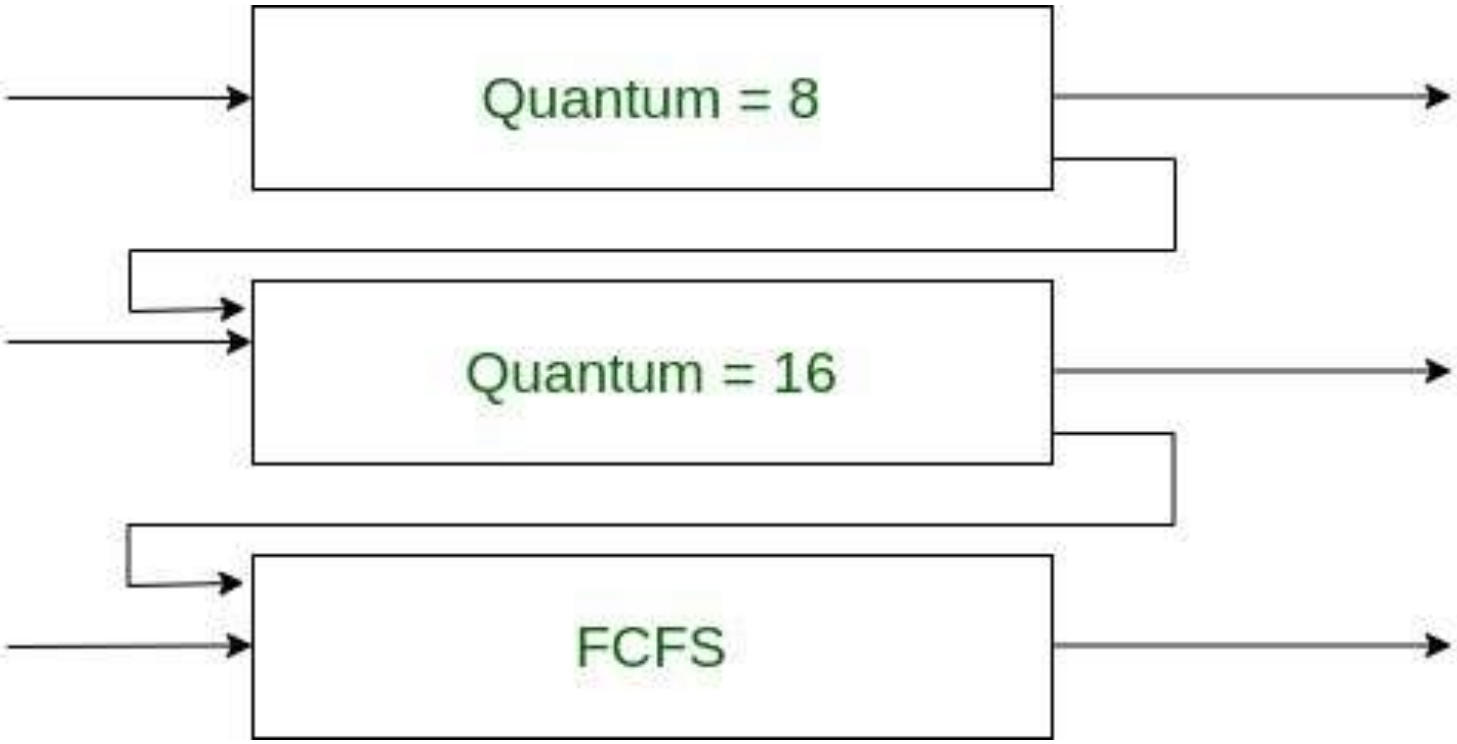
- Each queue has absolute priority over lower-priority queues.
- No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty.
- If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.
- Another possibility is to time-slice among the queues.
- Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes.
- For instance, in the foreground–background queue example, the foreground queue can be given 80 percent of the CPU time for RR scheduling among its processes, while the background queue receives 20 percent of the CPU to give to its processes on an FCFS basis.

Multilevel Feedback Queue Scheduling

- The multilevel feedback queue scheduling algorithm, allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.
- This form of aging prevents starvation.
- For example, consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2.
- The scheduler first executes all processes in queue 0.
- Only when queue 0 is empty will it execute processes in queue 1.
- Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty.

Multilevel Feedback Queue Scheduling

- A process that arrives for queue 1 will preempt a process in queue 2.
- A process in queue 1 will in turn be preempted by a process arriving for queue 0



Multi-level feedback queue scheduling

- A process entering the ready queue is put in queue 0.
- A process in queue 0 is given a time quantum of 8 milliseconds.
- If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds.
- If it does not complete, it is preempted and is put into queue 2.
- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- This scheduling algorithm gives highest priority to any process with a CPU burst of 8 milliseconds or less.
- Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/O burst.
- Processes that need more than 8 but less than 24 milliseconds are also served quickly, although with lower priority than shorter processes.
- Long processes automatically sink to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.

Multi-level feedback queue scheduling

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher-priority queue
- The method used to determine when to demote a process to a lower-priority queue
- The method used to determine which queue a process will enter when that process needs service

Thread Scheduling

- F

Thread Scheduling

- F

Thread Scheduling

- F

Question and Answer

1.