Process

- A process is a set of sequential steps that are required to do a particular task.
- A process is an instance of a program in execution.
- For e.g.: in Windows, if we edit two text files, simultaneously, in notepad, then it means we are implementing two different instances of the same program.
- For an operating system, these two instances are separate processes of the same application.

Process

- A process needs certain resources such as:
  - CPU Time
  - Memory Files
  - I/O Devices

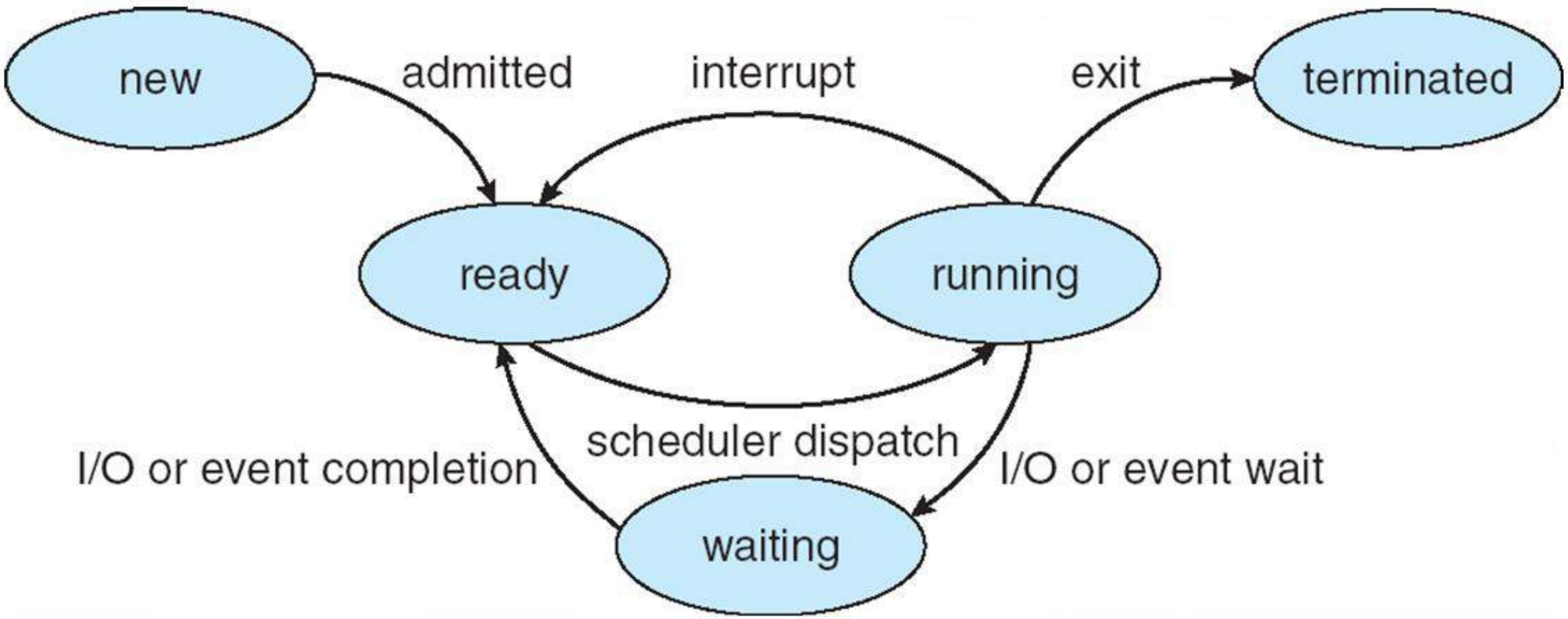
  to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.

Process States

- A process goes through a series of process states for performing its task.
- As a process executes, it changes state.
- Various events can cause a process to change state.

Process States

- The various states of a process are:



Process States

- **New:**
  - A process that has just been created.
- **Ready:**
  - The process is ready to be executed.
- **Running:**
  - The process whose instructions are being executed is called running process.

Process States

- **Waiting:**
  - The process is waiting for some event to occur such as completion of I/O operation.
- **Terminated:**
  - The process has finished its execution.
- **Note:** Only one process can be *running* on any processor at any instant. However, there can be many processes in *ready* and *waiting* states.

Process Control Block (PCB)

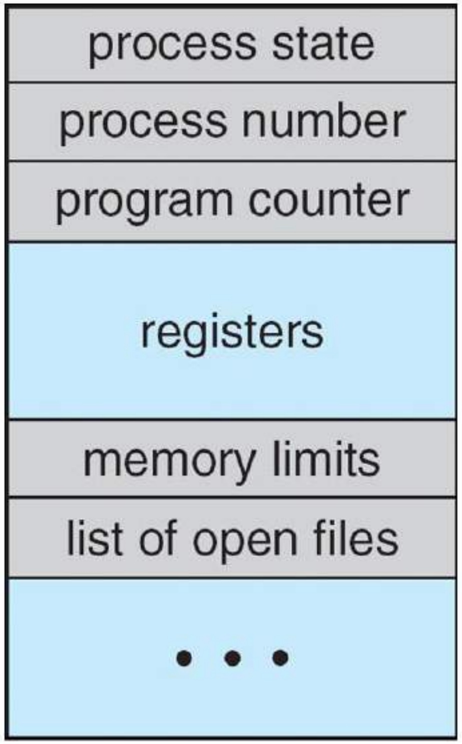
- Process Control Block (PCB) is a data structure used by operating system to store all the information about a process.
- It is also known as Process Descriptor.
- When a process is created, the operating system creates a corresponding PCB.

Process Control Block (PCB)

- Information in a PCB is updated during the transition of process states.
- When a process terminates, its PCB is released.
- Each process has a single PCB.

Process Control Block (PCB)

- The PCB of a process contains the following information:



Process Control Block (PCB)

- **Process Number:** Each process is allocated a unique number for the purpose of identification.
- **Process State:** It specifies the current state of a process.
- **Program Counter:** It indicates the address of next instruction to be executed.

Process Control Block (PCB)

- **Registers:** These hold the data or result of calculations. The content of these registers is saved so that a process can be resumed correctly later on.
- **Memory Limits:** It stores the amount of memory units allocated to a process.
- **List of Open Files:** It stores the list of open files and there access rights.

Process Scheduling

- In multiprogramming, several processes are kept in main memory so that when one process is busy in I/O operation, other processes are available to CPU.
- In this way, CPU is busy in executing processes at all times.
- This method of selecting a process to be allocated to CPU is called Process Scheduling.

Process Scheduling

- Process scheduling consists of the following sub-functions:
  - **Scheduling:** Selecting the process to be executed next on CPU is called scheduling.
    - In this function a process is taken out from a pool of ready processes and is assigned to CPU.
    - This task is done by a component of operating system called **Scheduler.**

Process Scheduling

- **Dispatching:** Setting up the execution of the selected process on the CPU is called dispatching.
  - It is done by a component of operating system called **Dispatcher**.
  - Thus, a dispatcher is a program responsible for assigning the CPU to the process, that has been selected by the Scheduler.
- **Context Save:** Saving the status of a running process when its execution is to be suspended is known as context save.

Scheduling Queues

- In multiprogramming, several processes are there in ready or waiting state.
- These processes form a queue.
- The various queues maintained by operating system are:
  - Job Queue
  - Ready Queue
  - Device Queue

Scheduling Queues

- **Job Queue:**
  - As the process enter the system, it is put into a job queue. This queue consists of all processes in the system.

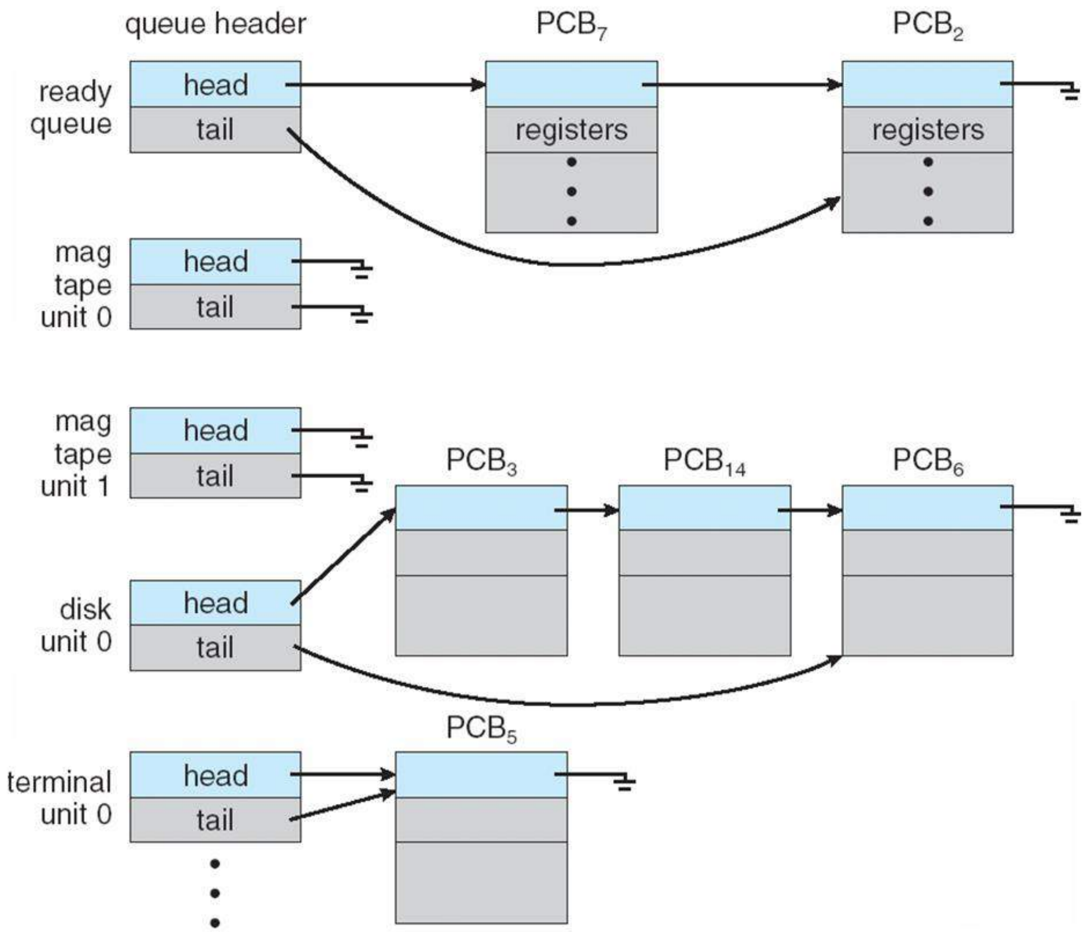
- **Ready Queue:**
  - It is a doubly linked list of processes that are residing in the main memory and are ready to run.

Scheduling Queues

- **Device Queue:**
  - It contains all those processes that are waiting for a particular I/O device.
  - Each device has its own device queue.

- Diagram on the next slide shows the queues.

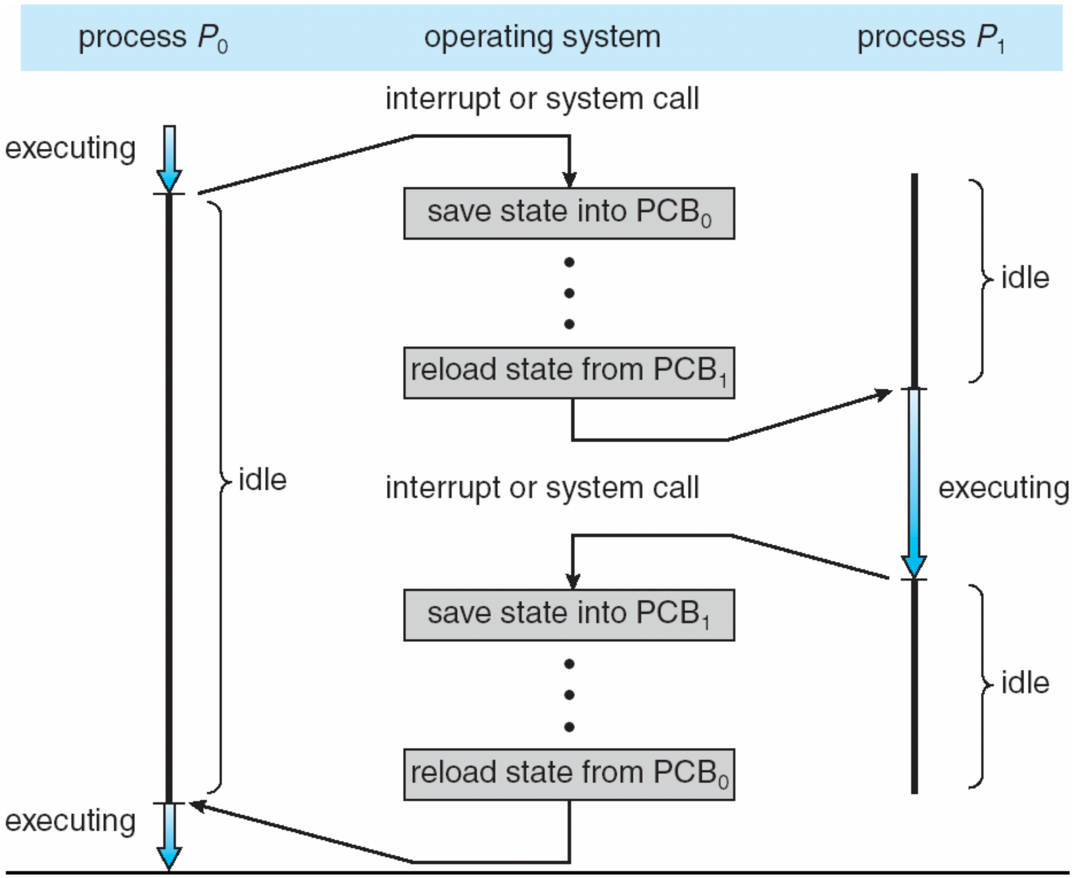
Scheduling Queues



Context Switch

- Switching the CPU from one process to another process requires saving the state of old process and loading the saved state of new process.
- This task is known as **Context Switch**.
- When context switch occurs, operating system saves the context of old process in its PCB and loads the saved context of the new process.

Context Switch



Threads

Thread

- A thread is a single sequential flow of execution of the tasks of a process.
- A thread is a lightweight process and the smallest unit of CPU utilization. Thus, a thread is like a miniprocess.
- Each thread has a thread id, program counter, register set and a stack.
- A thread undergoes different states such as new, ready, running, waiting and terminated similar to that of a process.
- However, a thread is not a program as it

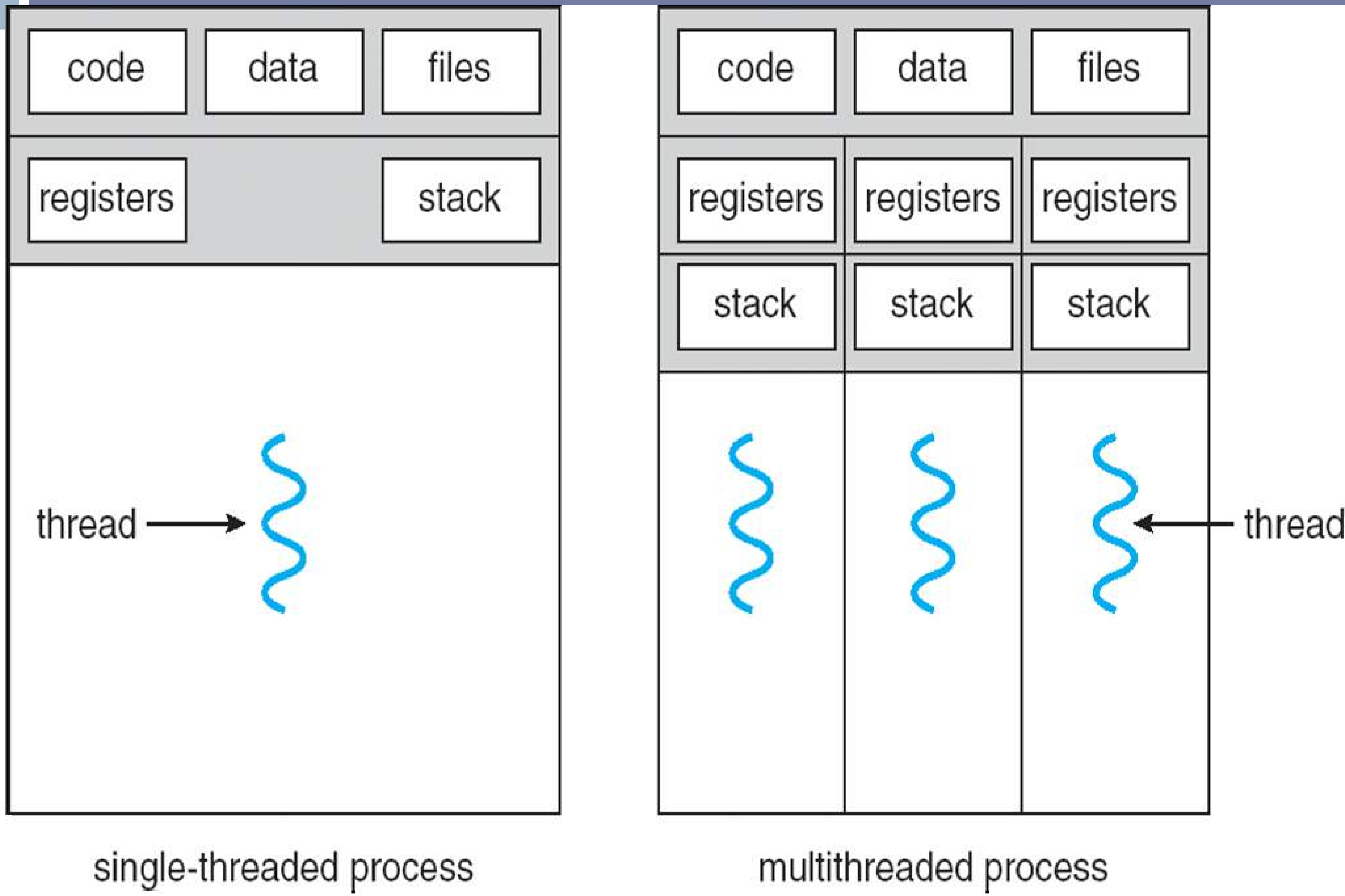
Multi-Threading

- A process can have single thread of control or multiple threads of control.
- If a process has single thread of control, it can perform only one task at a time.
- Many modern operating systems have extended the process concept to allow a process to have multiple threads.
- Thus, allowing the process to perform multiple tasks at the same time.
- This concept is known as **Multi-Threading**.

Multi-Threading

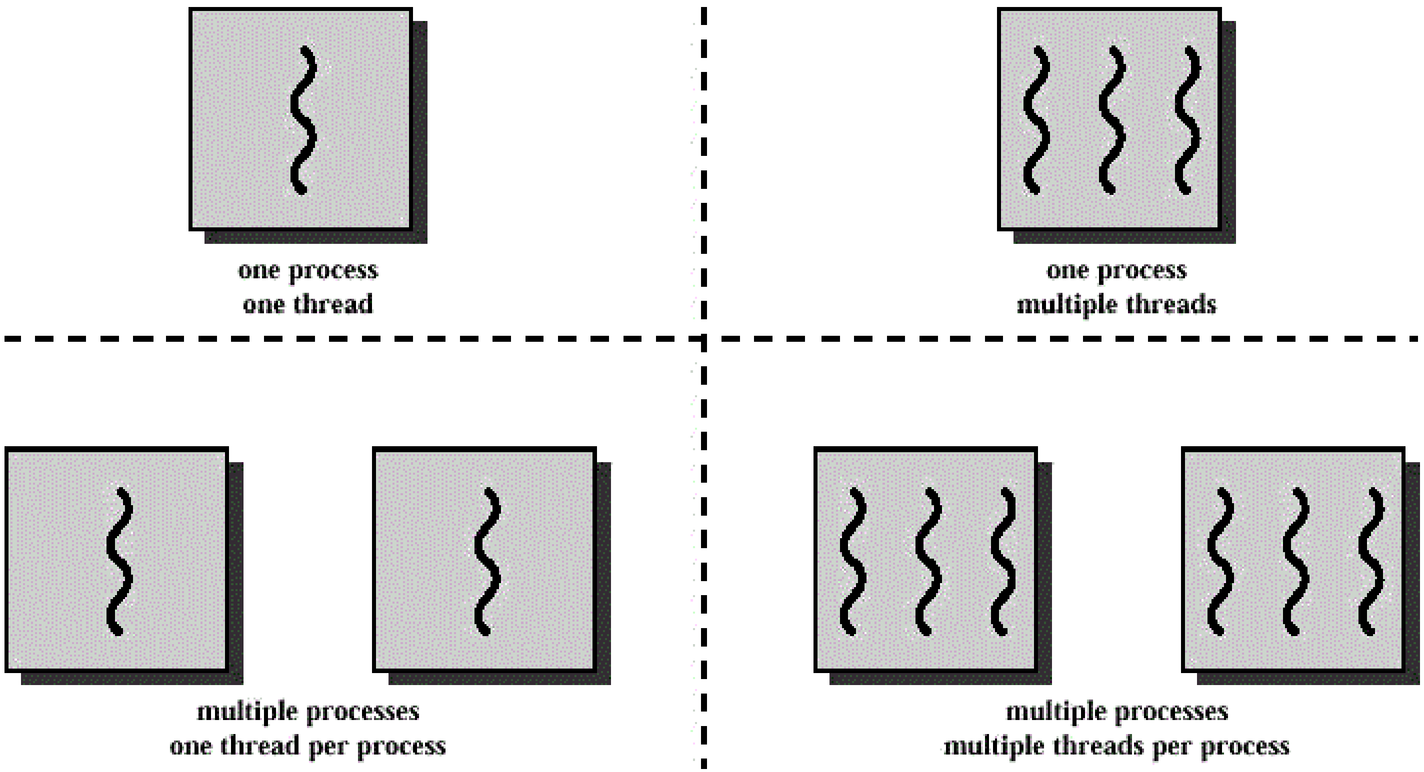
- For e.g.:
  - The tasks in a web browser are divided into multiple threads.
  - Downloading the images, downloading the text and displaying images and text.
  - While one thread is busy in downloading the images, another thread displays it.
- The various operating systems the implement multithreading are Windows XP, Vista, 7, Server 2000 onwards, Linux etc.
- In multithreading, a thread can share its code, data and resources with other threads of same process.

Single Thread & Multi-Thread



Threads & Processes

- An idea of how threads & processes can be related to each other is depicted in the fig.:



Threads & Processes

- There are several similarities and differences between a thread and a process:
  - **Similarities:**
    - Like process, each thread has its own program counter and stack.
    - Threads share CPU just as a process.
    - Threads also run sequentially, like a process.
    - Threads can create child threads.
    - Threads have the same states as process: new, ready, running, waiting and terminated.

Threads & Processes

Differences:

- Each process has its own distinct address space in the main memory. On the other hand, all threads of a same process share same address space.
- Threads require less system resources than a process.
- Threads are not independent of each other, unlike processes.
- Threads take less time for creation and termination than a process.
- It takes less time to switch between two threads than to switch between two processes.

Types of Threads

- Threads are of three types:
  - Kernel Level Threads
  - User Level Threads
  - Hybrid Threads

Kernel Level Threads

- Threads of processes defined by operating system itself are called **Kernel Level Threads**.
- In these types of threads, kernel performs thread creation, scheduling and management.
- Kernel threads are used for internal workings of operating system.
- Kernel threads are slower to create and manage.
- The various operating systems that support kernel level threads are: Windows 2000, XP, Solaris 2.

User Level Threads

- The threads of user application process are called **User Level Threads.**
- They are implemented in the user space of main memory.
- User level library (functions to manipulate user threads) is used for thread creation, scheduling and management without any support from the kernel.
- User level threads are fast to create and manage.

Hybrid Threads

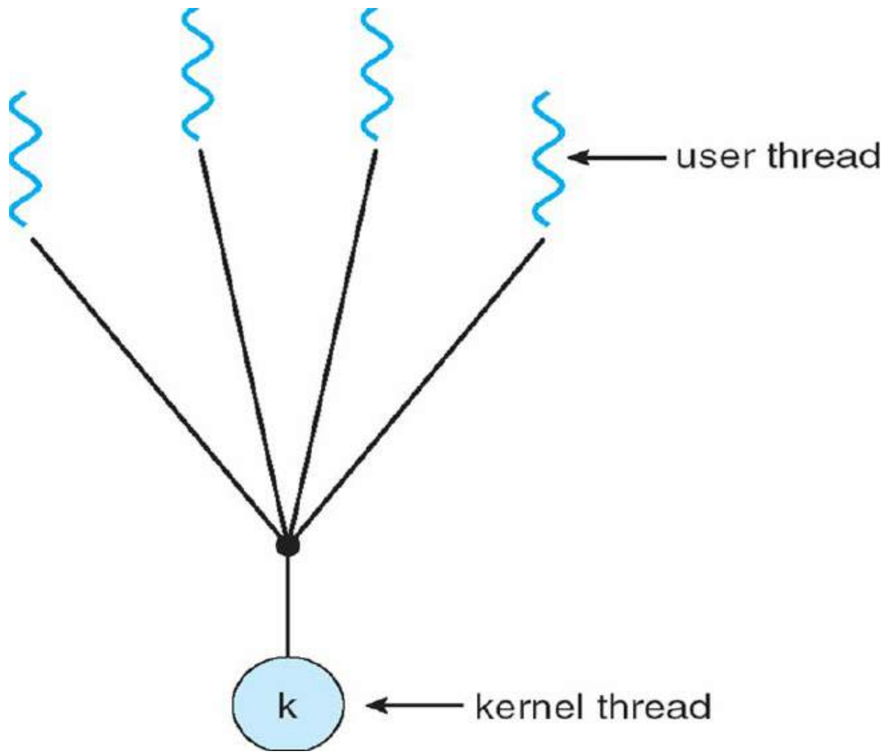
- In hybrid approach, both kernel level threads and user level threads are implemented.
- For e.g.: Solaris 2.

Multi-Threading Models

- Depending on the support for user and kernel threads, there are three multithreading models:
  - Many-to-One Model
  - One-to-One Model
  - Many-to-Many Model

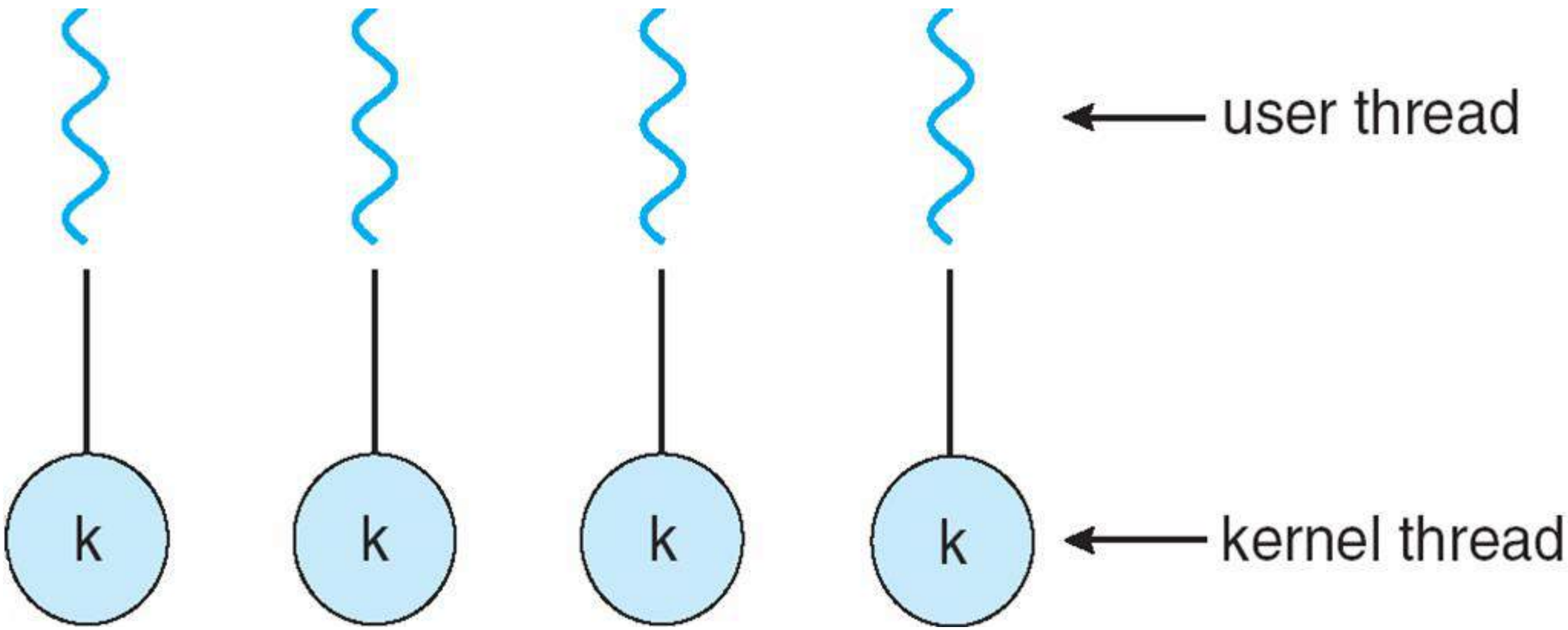
Many-to-One Model

- In this model, many user level threads are mapped to one kernel level thread.
- Threads are managed in user space.



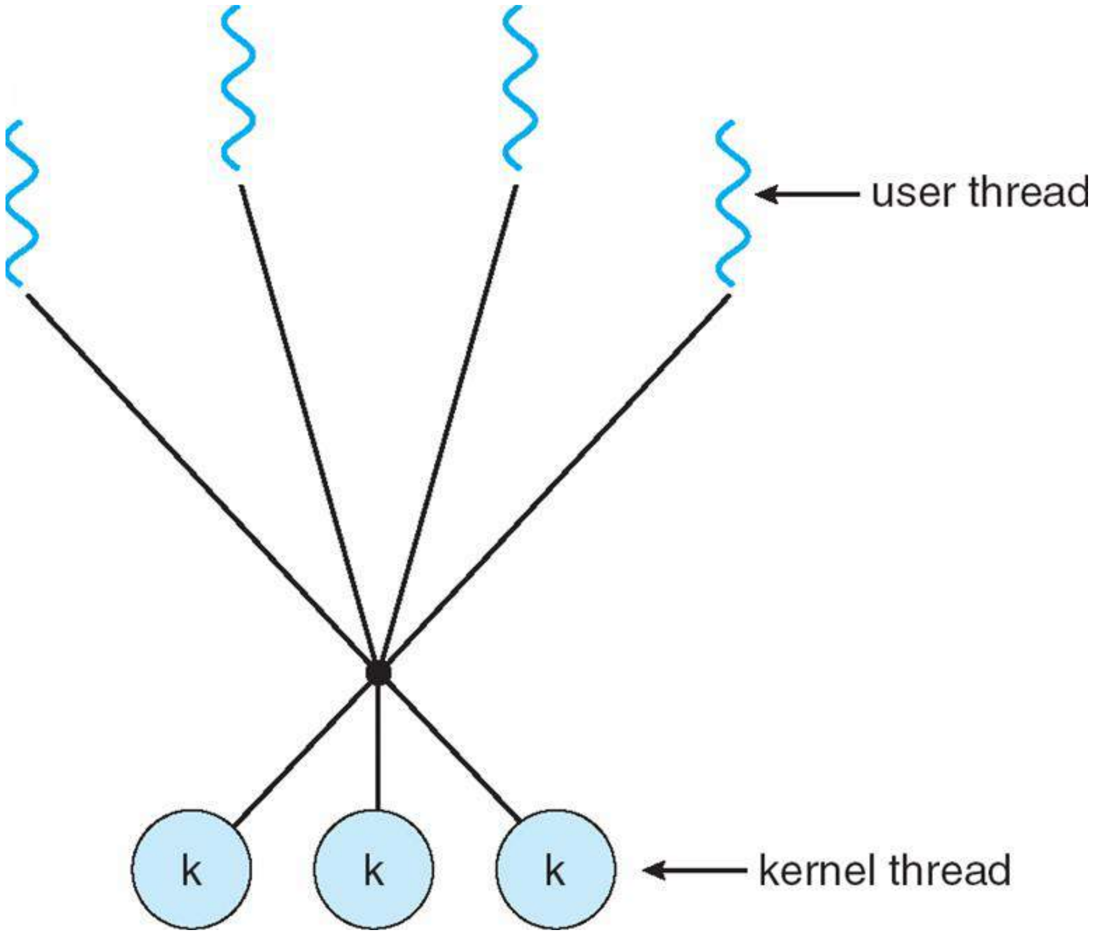
One-to-One Model

- In this model, each user level thread is mapped to one kernel level thread.



Many-to-Many Model

- In this model, many user level threads are mapped to many kernel level threads.



Scheduling Algorithms

Scheduling Algorithms

- CPU Scheduling algorithms deal with the problem of deciding which process in ready queue should be allocated to CPU.

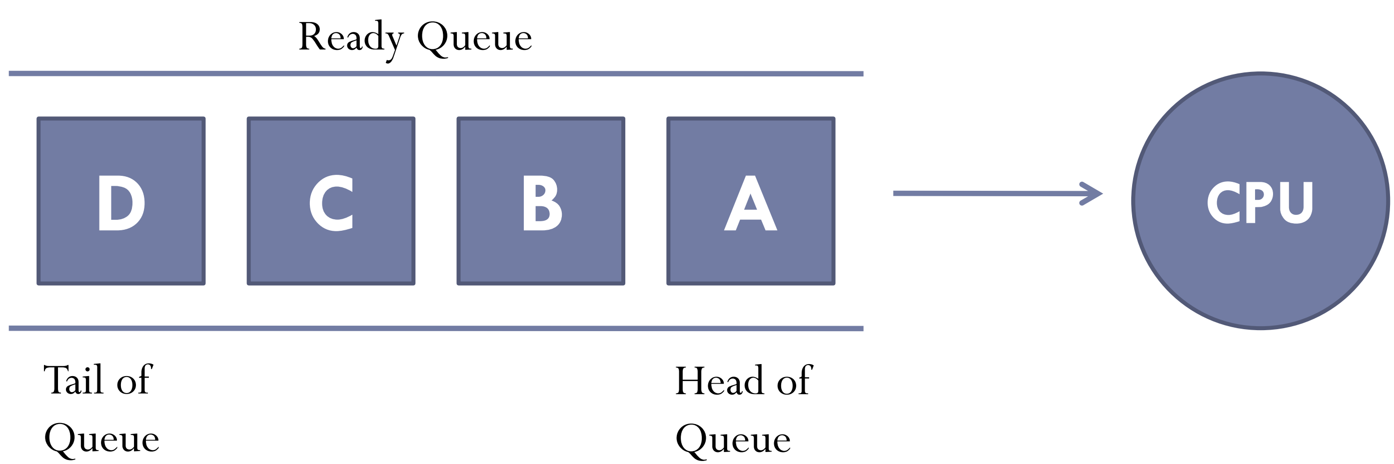
- Following are the commonly used scheduling algorithms:

Scheduling Algorithms

- First-Come-First-Served (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling
- Round-Robin Scheduling (RR)
- Multi-Level Queue Scheduling (MLQ)
- Multi-Level Feedback Queue Scheduling (MFQ)

First-Come-First-Served Scheduling (FCFS)

- In this scheduling, the process that requests the CPU first, is allocated the CPU first.
- Thus, the name *First-Come-First-Served*.
- The implementation of FCFS is easily managed with a FIFO queue.



First-Come-First-Served Scheduling (FCFS)

- When a process enters the ready queue, its PCB is linked to the tail of the queue.
- When CPU is free, it is allocated to the process which is at the head of the queue.
- FCFS scheduling algorithm is ***non-preemptive***.
- Once the CPU is allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by I/O request.

Example of FCFS Scheduling

- Consider the following set of processes that arrive at time 0 with the length of the CPU burst time in milliseconds:

Process	Burst Time (in milliseconds)
P_1	24
P_2	3
P_3	3

Example of FCFS Scheduling

- Suppose that the processes arrive in the order: P_1, P_2, P_3.

P_1	24
P_2	3
P_3	3

- The Gantt Chart for the schedule is:

- Waiting Time for P_1 = 0 milliseconds
- Waiting Time for P_2 = 24 milliseconds
- Waiting Time for P_3 = 27 milliseconds

Example of FCFS Scheduling

- Average Waiting Time = (Total Waiting Time) / No. of Processes
  = (0 + 24 + 27) / 3
  = 51 / 3
  = 17 milliseconds

Example of FCFS Scheduling

- Suppose that the processes arrive in the order: P_2, P_3, P_1.
- The Gantt chart for the schedule is:

- Waiting Time for P_2 = 0 milliseconds
- Waiting Time for P_3 = 3 milliseconds
- Waiting Time for P_1 = 6 milliseconds

Example of FCFS Scheduling

- Average Waiting Time = (Total Waiting Time) / No. of Processes

  = (0 + 3 + 6) / 3

  = 9 / 3

  = 3 milliseconds

- Thus, the average waiting time depends on the order in which the processes arrive.

First Come, First Served Scheduling

Process	Burst Time
P1	5
P2	3
P3	2

Timer

0

Shortest Job First Scheduling (SJF)

- In SJF, the process with the least estimated execution time is selected from the ready queue for execution.
- It associates with each process, the length of its next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length of next CPU burst, FCFS scheduling is used.
- SJF algorithm can be preemptive or non-preemptive.

Non-Preemptive SJF

- In non-preemptive scheduling, CPU is assigned to the process with least CPU burst time.
- The process keeps the CPU until it terminates.
- **Advantage:**
  - It gives minimum average waiting time for a given set of processes.
- **Disadvantage:**
  - It requires knowledge of how long a process will run and this information is usually not available.

Preemptive SJF

- In preemptive SJF, the process with the smallest estimated run-time is executed first.
- Any time a new process enters into ready queue, the scheduler compares the expected run-time of this process with the currently running process.
- If the new process's time is less, then the currently running process is preempted and the CPU is allocated to the new process.

Example of Non-Preemptive SJF

- Consider the following set of processes that arrive at time 0 with the length of the CPU burst time in milliseconds:

Process	Burst Time (in milliseconds)
P_1	6
P_2	8
P_3	7
P_4	3

Example of Non-Preemptive SJF

P_1	6
P_2	8
P_3	7
P_4	3

- The Gantt Chart for the schedule is:

P_4	P_1	P_3	P_2

0 3 9 16 24

- Waiting Time for P_4 = 0 milliseconds
- Waiting Time for P_1 = 3 milliseconds
- Waiting Time for P_3 = 9 milliseconds
- Waiting Time for P_2 = 16 milliseconds

Example of Non-Preemptive SJF

- Average Waiting Time = (Total Waiting Time) / No. of Processes
  = (0 + 3 + 9 + 16) / 4
  = 28 / 4
  = 7 milliseconds

Shortest Job First Scheduling

Process	Burst Time
P1	5
P2	3
P3	2

Timer

0

Example of Preemptive SJF

- Consider the following set of processes. These processes arrived in the ready queue at the times given in the table:

Process	Arrival Time	Burst Time (in milliseconds)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Example of Preemptive SJF

- The Gantt Chart for the schedule is:

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

- Waiting Time for P_1 = 10 – 1 – 0 = 9
- Waiting Time for P_2 = 1 – 1 = 0
- Waiting Time for P_3 = 17 – 2 = 15
- Waiting Time for P_4 = 5 – 3 = 2

P	AT	BT
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Example of Preemptive SJF

- Average Waiting Time = (Total Waiting Time) / No. of Processes
  = (9 + 0 + 15 + 2) / 4
  = 26 / 4
  = 6.5 milliseconds

Explanation of the Example

- Process P_1 is started at time 0, as it is the only process in the queue.
- Process P_2 arrives at the time 1 and its burst time is 4 milliseconds.
- This burst time is less than the remaining time of process P_1 (7 milliseconds).
- So, process P_1 is preempted and P_2 is scheduled.

Explanation of the Example

- Process P_3 arrives at time 2. Its burst time is 9 which is larger than remaining time of P_2 (3 milliseconds).
- So, P_2 is not preempted.
- Process P_4 arrives at time 3. Its burst time is 5. Again it is larger than the remaining time of P_2 (2 milliseconds).
- So, P_2 is not preempted.

Explanation of the Example

- After the termination of P_2, the process with shortest next CPU burst i.e. P_4 is scheduled.
- After P_4, processes P_1 (7 milliseconds) and then P_3 (9 milliseconds) are scheduled.

Priority Scheduling

- In priority scheduling, a priority is associated with all processes.
- Processes are executed in sequence according to their priority.
- CPU is allocated to the process with highest priority.
- If priority of two or more processes are equal than FCFS is used to break the tie.

Priority Scheduling

- Priority scheduling can be preemptive or non-preemptive.
- **Preemptive Priority Scheduling:**
  - In this, scheduler allocates the CPU to the new process if the priority of new process is higher tan the priority of the running process.
- **Non-Preemptive Priority Scheduling:**
  - The running process is not interrupted even if the new process has a higher priority.
  - In this case, the new process will be placed at the head of the ready queue.

Priority Scheduling

- **Problem:**
  - In certain situations, a low priority process can be blocked infinitely if high priority processes arrive in the ready queue frequently.
  - This situation is known as ***Starvation.***

Priority Scheduling

- **Solution:**
  - ***Aging*** is a technique which gradually increases the priority of processes that are victims of starvation.
  - For e.g.: Priority of process X is 10.
  - There are several processes with higher priority in the ready queue.
  - Processes with higher priority are inserted into ready queue frequently.
  - In this situation, process X will face starvation.

Priority Scheduling

(Cont.):

- The operating system increases priority of a process by 1 in every 5 minutes.
- Thus, the process X becomes a high priority process after some time.
- And it is selected for execution by the scheduler.

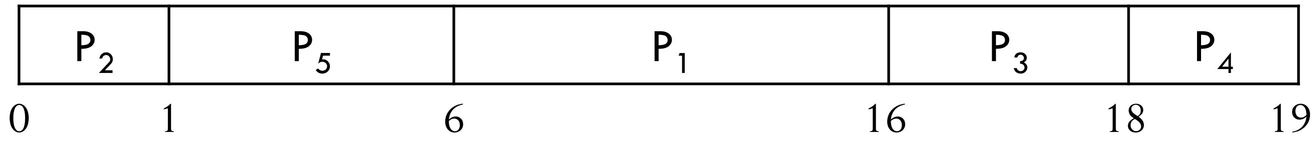
Example of Priority Scheduling

- Consider the following set of processes that arrive at time 0 with the length of the CPU burst time in milliseconds. The priority of these processes is also given:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Example of Priority Scheduling

- The Gantt Chart for the schedule is:



- Waiting Time for $P_2 = 0$
- Waiting Time for $P_5 = 1$
- Waiting Time for $P_1 = 6$
- Waiting Time for $P_3 = 16$
- Waiting Time for $P_4 = 18$

P	**BT**	**Pr**
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Example of Priority Scheduling

- Average Waiting Time = (Total Waiting Time) / No. of Processes

  = (0 + 1 + 6 + 16 + 18) / 5

  = 41 / 5

  = 8.2 milliseconds

Priority Scheduling

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Timer

0

Another Example of Priority Scheduling

- Processes P_1, P_2, P_3 are the processes with their arrival time, burst time and priorities listed in table below:

Process	Arrival Time	Burst Time	Priority
P_1	0	10	3
P_2	1	5	2
P_3	2	2	1

Another Example of Priority Scheduling

- The Gantt Chart for the schedule is:

P_1	P_2	P_3	P_2	P_1

0 1 2 4 8 17

- Waiting Time for $P_1 = 0 + (8 - 1) = 7$
- Waiting Time for $P_2 = 1 + (4 - 2) = 3$
- Waiting Time for $P_3 = 2$

P	**AT**	**BT**	**Pr**
P_1	0	10	3
P_2	1	5	2
P_3	2	2	1

Another Example of Priority Scheduling

- Average Waiting Time = (Total Waiting Time) / No. of Processes
  = (7 + 3 + 2) / 3
  = 12 / 3
  = 4 milliseconds

Round Robin Scheduling (RR)

- In Round Robin scheduling, processes are dispatched in FIFO but are given a small amount of CPU time.
- This small amount of time is known as ***Time Quantum*** or ***Time Slice***.
- A time quantum is generally from 10 to 100 milliseconds.

Round Robin Scheduling (RR)

- If a process does not complete before its time slice expires, the CPU is preempted and is given to the next process in the ready queue.
- The preempted process is then placed at the tail of the ready queue.
- If a process is completed before its time slice expires, the process itself releases the CPU.
- The scheduler then proceeds to the next process in the ready queue.

Round Robin Scheduling (RR)

- Round Robin scheduling is always preemptive as no process is allocated the CPU for more than one time quantum.
- If a process's CPU burst time exceeds one time quantum then that process is preempted and is put back at the tail of ready queue.
- The performance of Round Robin scheduling depends on several factors:
  - Size of Time Quantum
  - Context Switching Overhead

Example of Round Robin Scheduling

- Consider the following set of processes that arrive at time 0 with the length of the CPU burst time in milliseconds:

Process	Burst Time
P_1	10
P_2	5
P_3	2

- Time quantum is of 2 milliseconds.

Example of Round Robin Scheduling

- The Gantt Chart for the schedule is:

P_1	P_2	P_3	P_1	P_2	P_1	P_2	P_1	P_1

0 2 4 6 8 10 12 13 15 17

- Waiting Time for $P_1 = 0 + (6 - 2) + (10 - 8) + (13 - 12)$

  $= 4 + 2 + 1 = 7$

- Waiting Time for $P_2 = 2 + (8 - 4) + (12 - 10)$

  $= 2 + 4 + 2 = 8$

- Waiting Time for $P_3 = 4$

P	BT
P_1	10
P_2	5
P_3	2

Example of Round Robin Scheduling

- Average Waiting Time = (Total Waiting Time) / No. of Processes

  = (7 + 8 + 4) / 3

  = 19 / 3

  = 6.33 milliseconds

Round Robin Scheduling

Process	Burst Time
P1	5
P2	3
P3	2

Timer

0

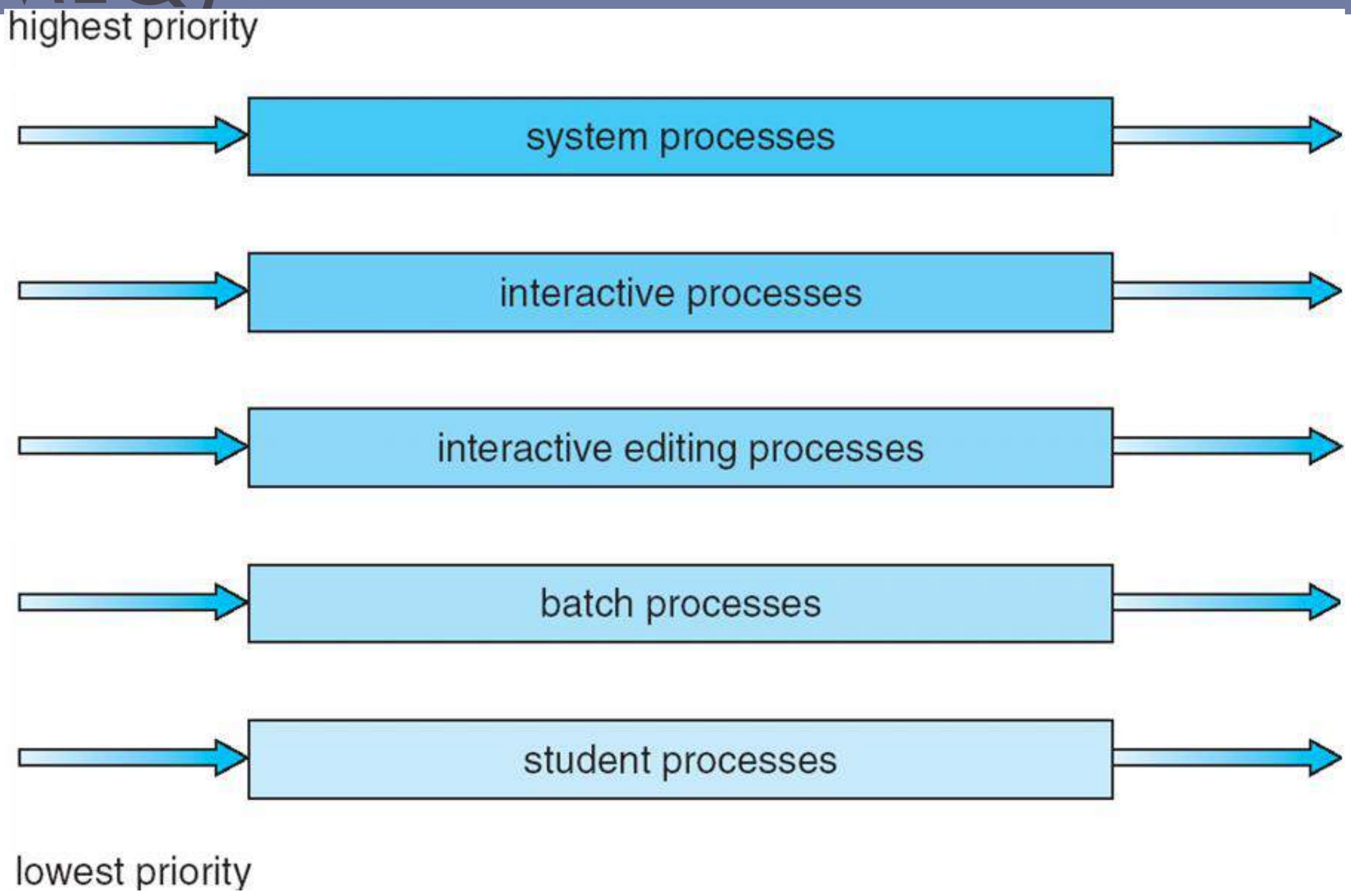
Multi-Level Queue Scheduling (MLQ)

- Multi-Level Queue scheduling classifies the processes according to their types.
- For e.g.: a MLQ makes common division between the interactive processes (foreground) and the batch processes (background).
- These two processes have different response times, so they have different scheduling requirements.
- Also, interactive processes have higher priority than the batch processes.

Multi-Level Queue Scheduling (MLQ)

- In this scheduling, ready queue is divided into various queues that are called subqueues.
- The processes are assigned to subqueues, based on some properties like memory size, priority or process type.
- Each subqueue has its own scheduling algorithm.
- For e.g.: interactive processes may use round robin algorithm while batch processes may use FCFS.

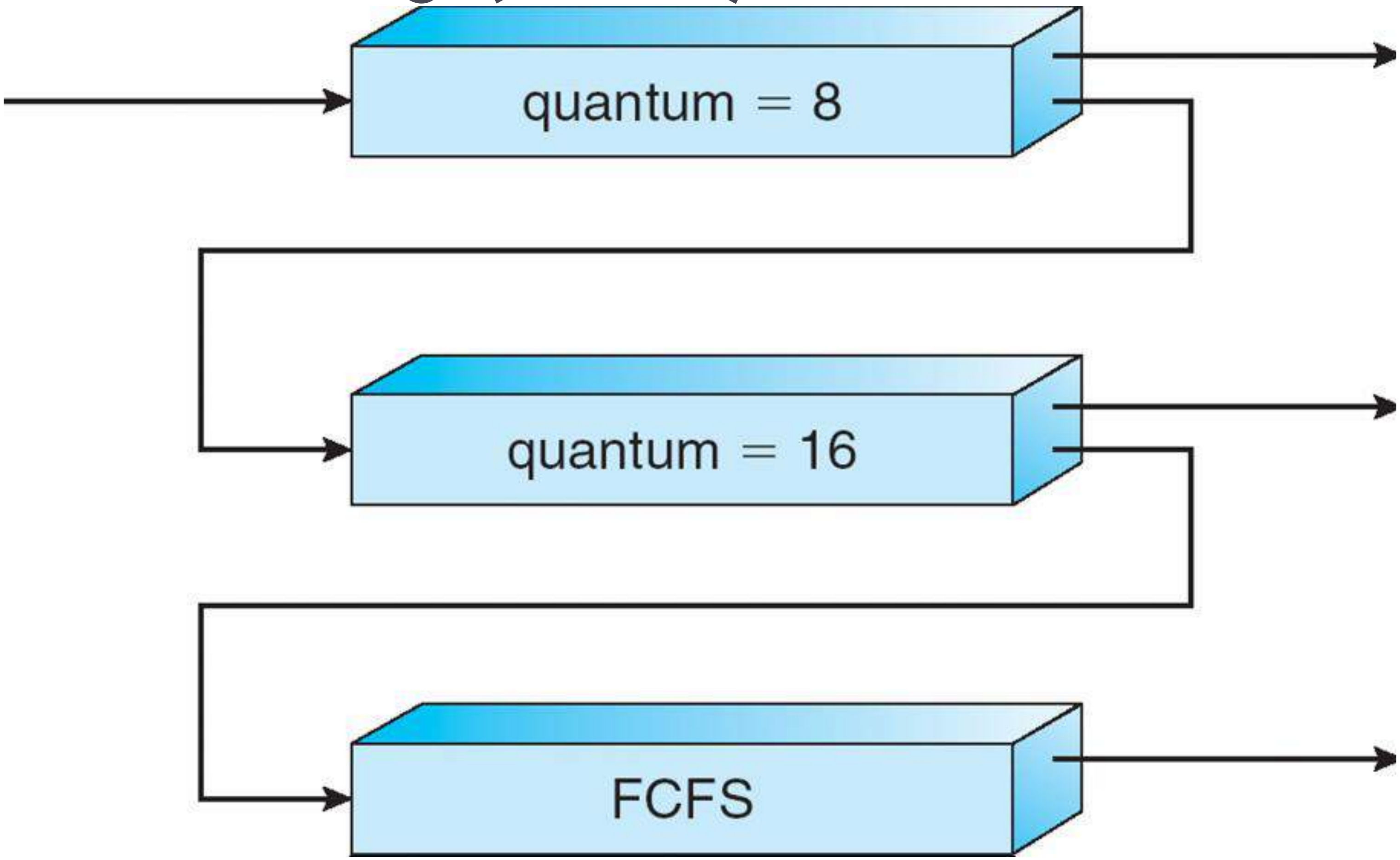
Multi-Level Queue Scheduling (MLQ)



Multi-Level Feedback Queue Scheduling (MFQ)

- Multi-Level Feedback Queue scheduling is an enhancement of MLQ.
- In this scheme, processes can move between different queues.
- The various processes are separated in different queues on the basis of their CPU burst times.
- If a process consumes a lot of CPU time, it is placed into a lower priority queue.
- If a process waits too long in a lower priority queue, it is moved into higher priority queue.
- Such an *aging* prevents starvation.

Multi-Level Feedback Queue Scheduling (MFQ)



Multi-Level Feedback Queue Scheduling (MFQ)

- The top priority queue is given smallest CPU time quantum.
- If the quantum expires before the process terminates, it is then placed at the back of the next lower queue.
- Again, if it does not complete, it is put to the last priority queue.
- The processes in this queue runs on FCFS scheduling.
- If a process becomes a victim of starvation, it is promoted to the next higher priority queue.

CPU Scheduling

- **Scheduling** refers to selecting a process, from many ready processes, that is to be next executed on CPU.
- In multiprogramming environment, multiple processes are kept in main memory.
- When one process has to wait for I/O completion, operating system takes the CPU from that process and assigns it to another process.
- In this way, CPU is never idle and has some process

Scheduler

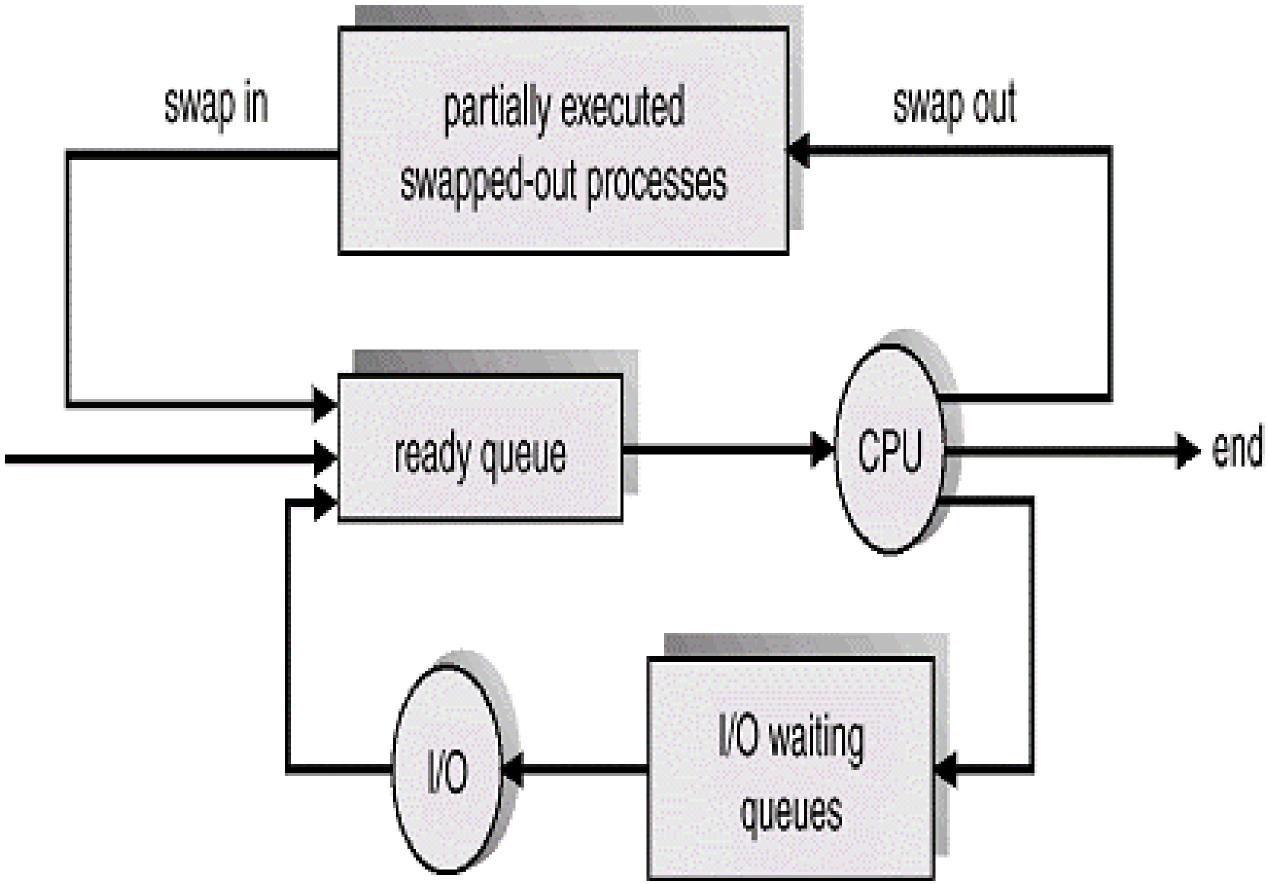
- **Scheduler** is an operating system module that selects the next job or process to be assigned to CPU.
- Thus, scheduler selects one of the many processes in memory that are ready to execute and allocates CPU to it.

Scheduler

- Scheduler is of three types:

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

Scheduler


swap in
partially executed
swapped-out processes
swap out
ready queue
CPU
end
I/O waiting
queues
I/O

Long Term Scheduler

- Long Term Scheduler selects the processes from secondary storage and loads them into memory for execution.
- It is called “long term” because the time for which the scheduling is valid is long.
- The frequency of execution of a long term scheduler is usually low, as there may be minutes between the creation of new processes in the system.

Long Term Scheduler

- The primary objective of long term scheduler is to control the “**degree of multiprogramming**”.
- Degree of multiprogramming refers to the total number of processes present in the memory.
- If the degree of multiprogramming is stable, then the average rate of process creation is equal to the average terminate rate.

Long Term Scheduler

- This scheduler shows the best performance by selecting the good mixture of I/O bound and CPU bound processes.
- I/O bound processes are those that spend most of their time in I/O.
- CPU bound processes are those that spend most of their time in computations.

Medium Term Scheduler

- The medium term scheduler is required at the time when a swapped-out process is to be brought into pool of ready processes.
- A running process may be suspended because of I/O request.
- Such a suspended process is then removed from main memory and stored in secondary memory.

Medium Term Scheduler

- This is done because there is a limit on the number of active processes that can reside in main memory.
- Therefore, a suspended process is swapped-out from main memory.
- At some later time, the process can be swapped-in into the main memory.
- All versions of Windows use swapping.

Short Term Scheduler

- Short term scheduler selects one process from many ready processes that are residing in main memory and allocates CPU to one of them.
- Thus, it handles the scheduling of the processes that are in ready state.
- Short term scheduler is also known as **CPU Scheduler.**

Short Term Scheduler

- As compared to long term scheduler, a short term scheduler has to work very often.
- The frequency of execution of short term scheduler is high.
- It must select a new process for CPU frequently.

Preemptive & Non-Preemptive Scheduling

- A scheduling algorithm can be:

1. Preemptive Scheduling
2. Non-Preemptive Scheduling

Non-Preemptive Scheduling

- A scheduling is **non-preemptive** if, once a process has been given the CPU, the CPU cannot be taken away from the process.
- In other words, in non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by entering the waiting state.

Preemptive Scheduling

- A scheduling is preemptive if the CPU can be taken away from a process after being allocated.
- In other words, even if the CPU has been allocated to a certain process, it can be snatched from the process any time either due to time constraint or due to priority reason.

Dispatcher

- **Dispatcher** is a program responsible for assigning the CPU to the process, which has been selected by the short term scheduler.
- Dispatching a process involves context switching.

Scheduling Criteria

- The goal of a scheduling algorithm is to identify the process whose selection will result in the best possible system performance.
- The various scheduling criteria for evaluating an algorithm are discussed next.

Scheduling Criteria

- **CPU Utilization:**
  - CPU utilization is the average fraction of time during which the processor is busy.
  - The level of CPU utilization depends on the load on the system.
  - CPU utilization may range from 0 to 100%.

Scheduling Criteria

- **Throughput:**
  - It refers to the number of processes the system can execute in a period of time.
  - For long processes, this rate may be 1 process per hour.
  - For short processes, throughput may be 10 processes per second.
  - Thus, evaluation of throughput depends on the average length of a process.

Scheduling Criteria

- **Turnaround Time:**
  - This is the interval of time between the submission of a process and its completion.
  - Thus, turnaround time is an average period of time it takes a process to execute.
  - Turnaround time includes actual execution time plus time spent waiting for resources and doing I/O.

Scheduling Criteria

- **Waiting Time:**
  - It is the average period of time a process spends waiting.
  - Waiting time can be expressed as

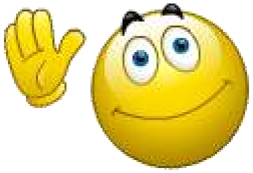
$$W(x) = T(x) - x$$

  - where, $W(x)$ is the waiting time
  - $T(x)$ is the turnaround time
  - x is the actual execution time.

Scheduling Algorithm Optimization Criteria

- The optimization criteria is:
  - Max. CPU Utilization
  - Max. Throughput
  - Min. Turnaround Time
  - Min. Waiting Time
  - Min. Response Time

Thank You



Have a Nice Day