IOS103

OPERATING SYSTEM
Processes and CPU Scheduling
Module 2

Objectives

At the end of the course, the student should be able to:

- *Explain process and types of computer processes;*
- *Discuss different states of a process;*
- *Define concurrent processes, parent and child processes;*
- *Discuss the concept of scheduling;*
- *Explain the CPU scheduler and different scheduling algorithms.*

Processes and CPU Scheduling

Process Concept

- A ***process*** is a program in execution. A program by itself is not a process. A program is a *passive entity*, such as the contents of a file stored on disk while a process is an *active entity*.

- A computer system consists of a collection of processes: ***operating-system processes*** execute system code, and ***user processes*** execute user code.

Processes and CPU Scheduling

Process Concept

- Although several processes may be associated with the same program, they are nevertheless considered separate execution sequences.
- All processes can potentially execute concurrently with the CPU (or CPUs) multiplexing among them (***time sharing***).
- A process is actually a cycle of CPU execution (***CPU burst***) and I/O wait (***I/O burst***). Processes alternate back and forth between these two states.

Processes and CPU Scheduling

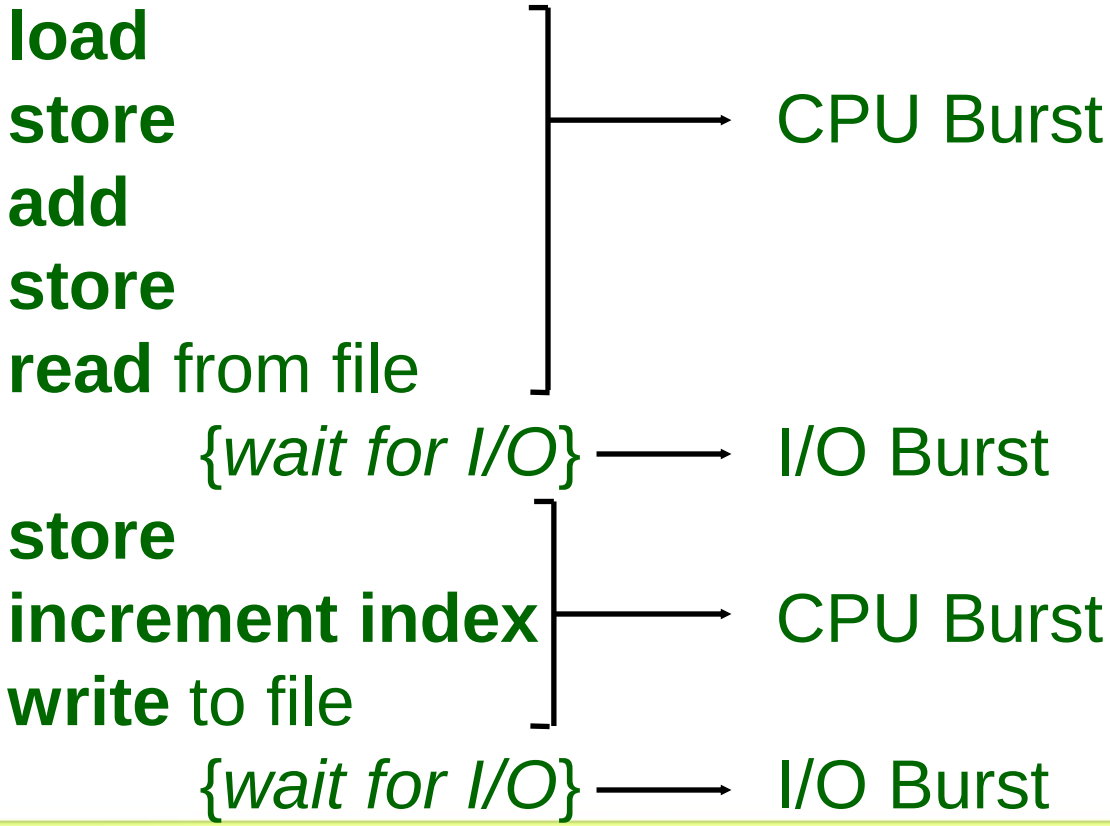
Process Concept

- Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the last CPU burst will end with a system request to terminate execution.

Processes and CPU Scheduling

Process Concept

Example:



Processes and CPU Scheduling

Process Concept

- A process is more than the program code plus the current activity (as indicated by contents the ***program counter*** and the CPU's registers). A process generally also includes the:

1. ***process stack*** containing temporary data (such as subroutine parameters, return addresses, and local variables), and a
2. ***data section*** containing global variables.

Processes and CPU Scheduling

Process Concept

- As a process executes, it changes ***state***. The current activity of a process party defines its state. Each sequential process may be in one of following states:

1. ***New***. The process is being created.

2. ***Running***. The CPU is executing its instructions.

Processes and CPU Scheduling

Process Concept

3. ***Waiting***. The process is waiting for some event to occur (such as an I/O completion).

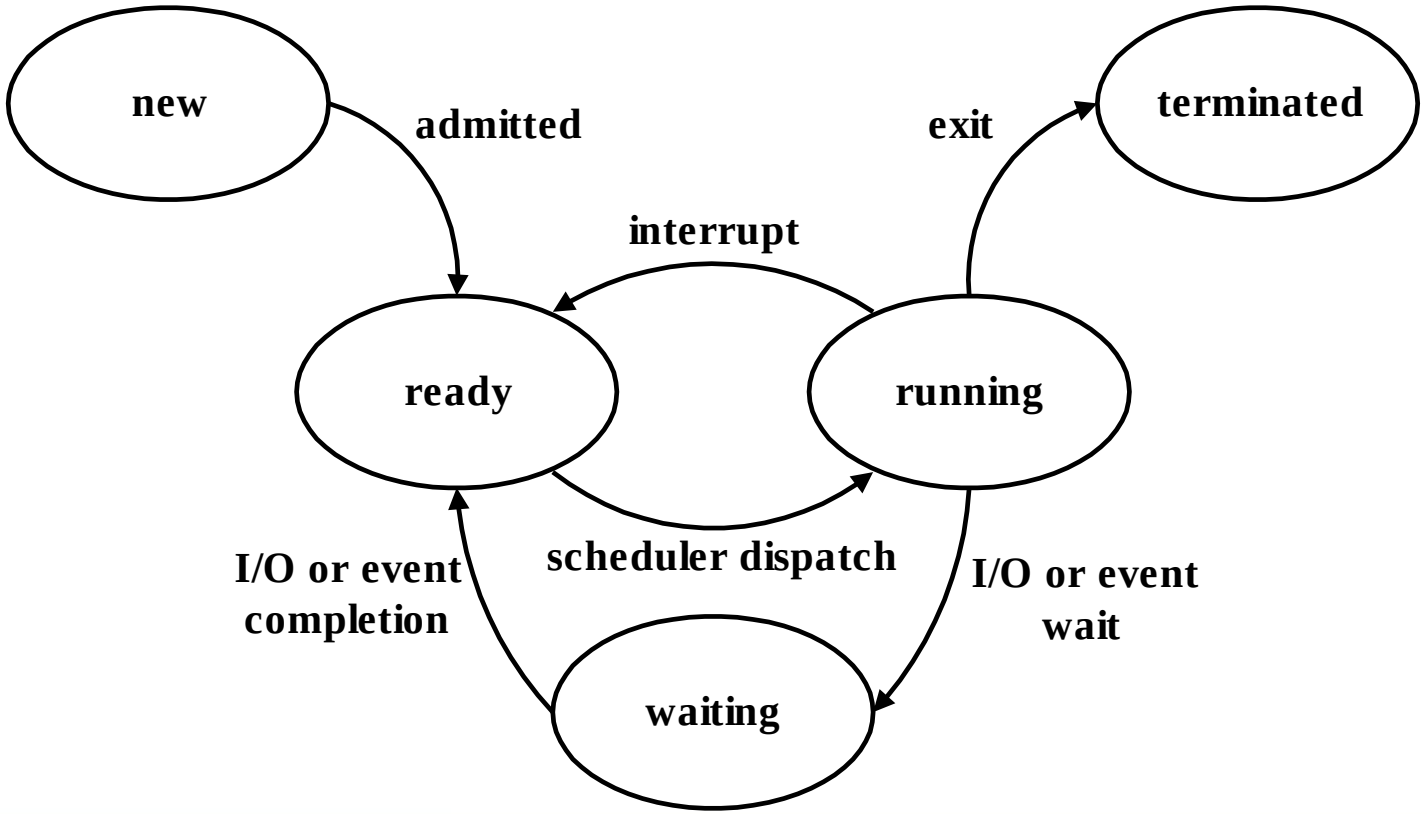
4. ***Ready***. The process is waiting for the OS to assign a processor to it.

5. ***Terminated***. The process has finished execution.

Processes and CPU Scheduling

Process Concept

- Process state diagram:



Processes and CPU Scheduling

Process Concept

- Each process is represented in the operating system by a ***process control block*** (PCB) – also called a ***task control block***. A PCB is a data block or record containing many pieces of the information associated with a specific process including:

1. ***Process state***. The state may be new, ready, running, waiting, or halted.
2. ***Program Counter***. The program counter indicates the address of the next instruction to be executed for this process.

Processes and CPU Scheduling

Process Concept

3. ***CPU Registers***. These include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
4. ***CPU Scheduling Information***. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Processes and CPU Scheduling

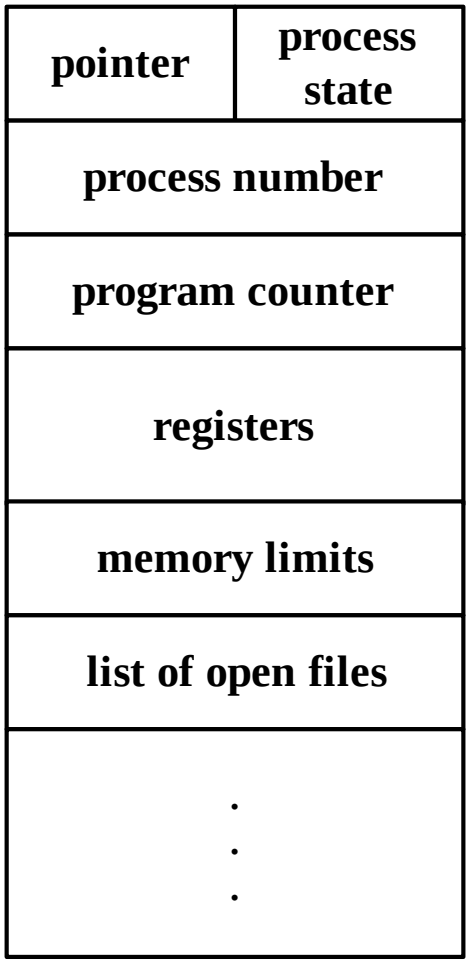
Process Concept

5. ***Memory Management Information***. This information includes limit registers or page tables.
6. ***Accounting Information***. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
7. ***I/O Status Information***. This information includes outstanding I/O requests, I/O devices (such as disks) allocated to this process, a list of open files, and so on.

Processes and CPU Scheduling

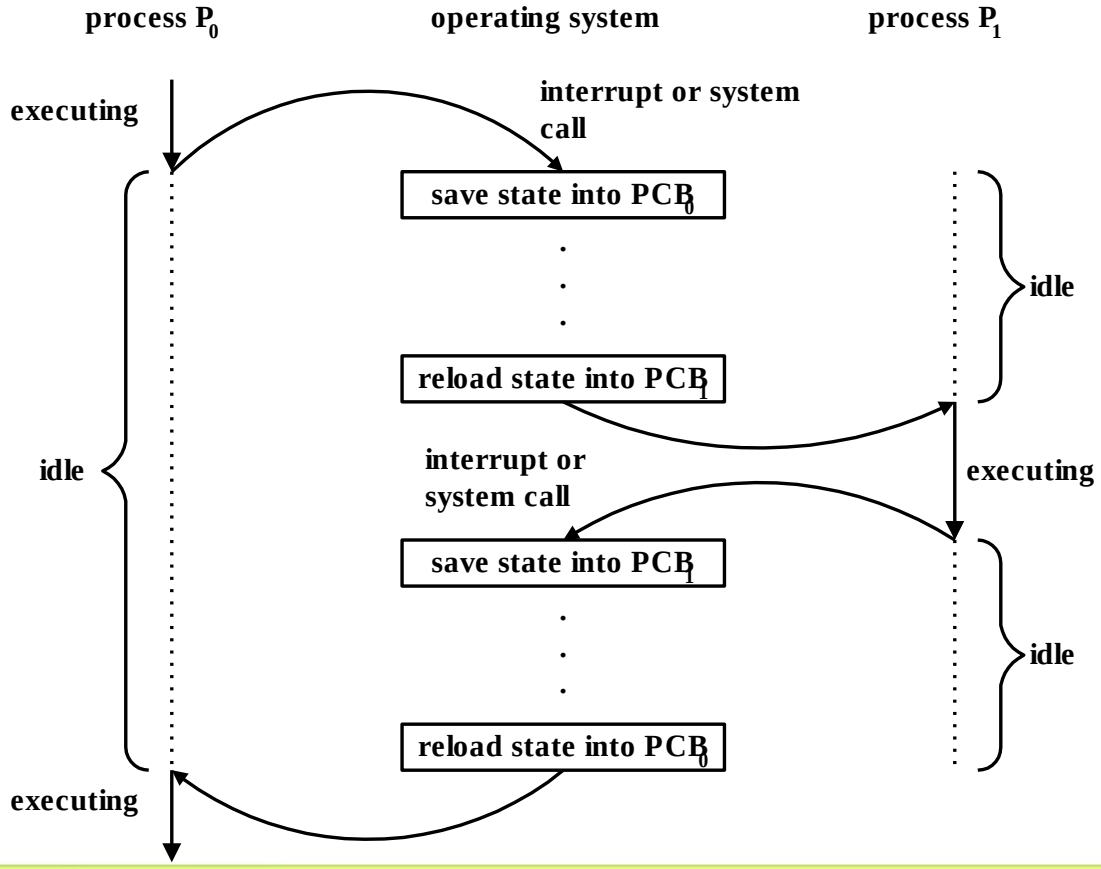
Process Concept

- The PCB simply serves as the repository for any information that may vary from process to process.



Processes and CPU Scheduling

- Example of the CPU being switched from one process to another.



Processes and CPU Scheduling

Concurrent Process

- The processes in the system can execute concurrently; that is, many processes may be multitasked on a CPU.
- A process may create several new processes, via a ***create-process*** system call, during the course of execution. Each of these new processes may in turn create other processes.
- The creating process is the ***parent*** process whereas the new processes are the ***children*** of that process.

Processes and CPU Scheduling

Concurrent Process

- When a process creates a sub-process, the sub-process may be able to obtain its resources directly from the operating system or it may use a subset of the resources of the parent process. Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many processes.

Processes and CPU Scheduling

Concurrent Process

- When a process creates a new process, two common implementations exist in terms of execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until all its children have terminated.

Processes and CPU Scheduling

Concurrent Process

- A process terminates when it finishes its last statement and asks the operating system to delete it using the ***exit*** system call.
- A parent may terminate the execution of one of its children for a variety of reason, such as

1. The child has exceeded its usage of some of the resources it has been allocated.

Processes and CPU Scheduling

Concurrent Process

2. The task assigned to the child is no longer required.

3. The parent is exiting, and the OS does not allow a child to continue if its parent terminates. In such systems, if a process terminates, then all its children must also be terminated by the operating system. This phenomenon is referred to as ***cascading termination***.

Processes and CPU Scheduling

Concurrent Process

- The concurrent processes executing in the operating system may either be ***independent processes*** or ***cooperating processes***.

Processes and CPU Scheduling

Concurrent Process

- A process is independent if it cannot affect or be affected by the other processes. Clearly, any process that does not share any data (temporary or persistent) with any other process is independent. Such a process has the following characteristics:

1. Its execution is deterministic; that is, the result of the execution depends solely on the input state.

Processes and CPU Scheduling

Concurrent Process

2. Its execution is reproducible; that is, the result of the execution will always be the same for the same input.

3. Its execution can be stopped and restarted without causing ill effects.

Processes and CPU Scheduling

Concurrent Process

- A process is cooperating if it can affect or be affected by the other processes. Clearly, any process that shares data with other processes is a cooperating process. Such a process has the following characteristics:
  1. The results of its execution cannot be predicted in advance, since it depends on relative execution sequence.
  2. The result of its execution is nondeterministic since it will not always be the same for the same input.

Processes and CPU Scheduling

Concurrent Process

- Concurrent execution of cooperating process requires mechanisms that allow processes to communicate with one another and to synchronize their actions.

Processes and CPU Scheduling

Scheduling Concepts

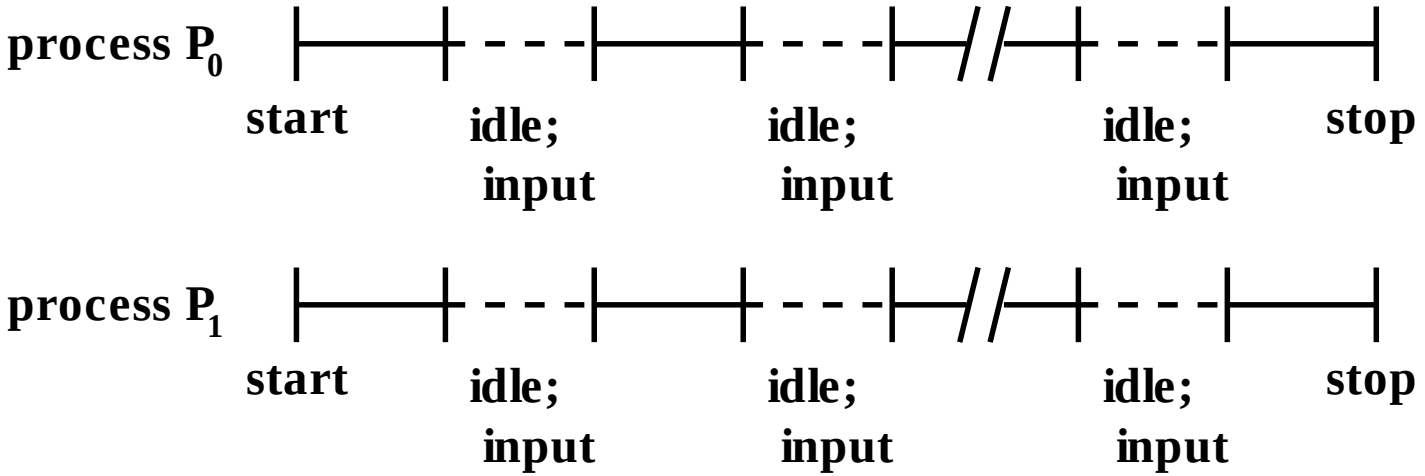
- The objective of ***multiprogramming*** is to have some process running at all times, to maximize CPU utilization. Multiprogramming also increases throughput, which is the amount of work the system accomplishes in a given time interval (for example, 17 processes per minute).

Processes and CPU Scheduling

Scheduling Concepts

Example:

Given two processes, P_0 and P_1.



If the system runs the two processes sequentially, then CPU utilization is only 50%.

Processes and CPU Scheduling

Scheduling Concepts

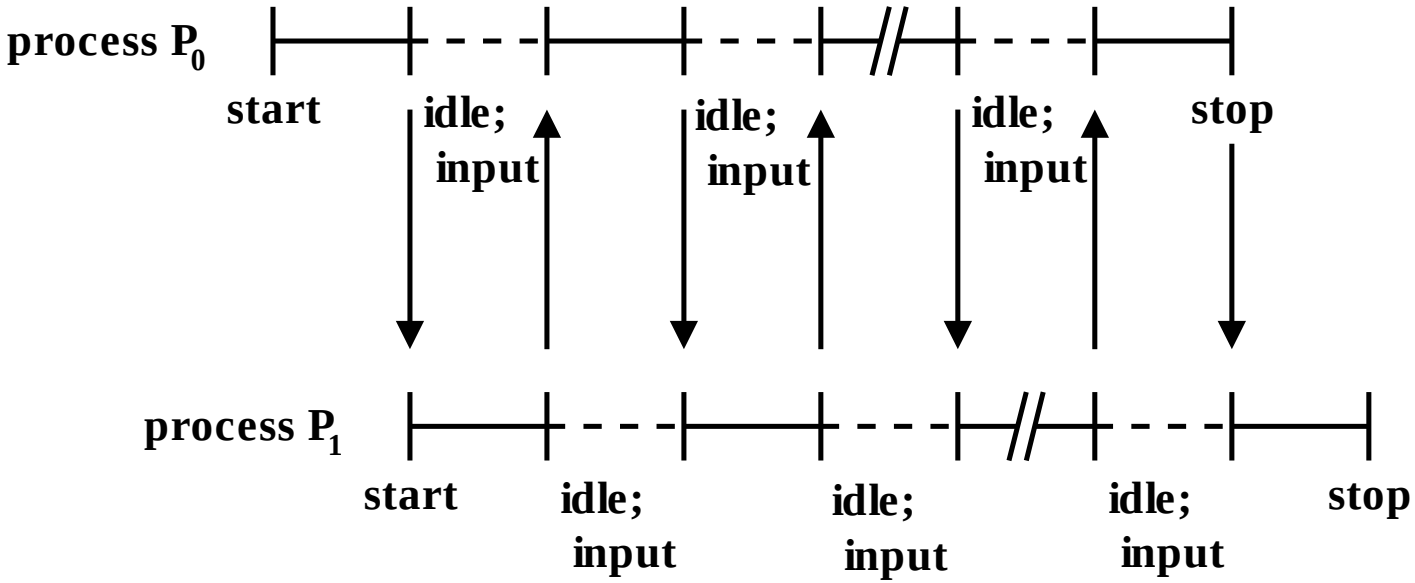
- The idea of multiprogramming is if one process is in the *waiting* state, then another process which is in the *ready* state goes to the *running* state.

Processes and CPU Scheduling

Scheduling Concepts

Example:

Applying multiprogramming to the two processes, P_0 and P_1.



then CPU utilization increases to 100%.

Processes and CPU Scheduling

Scheduling Concepts

- As processes enter the system, they are put into a ***job queue***. This queue consists of all processes in the system.

- The processes that are residing in main memory and are ready and waiting to execute are kept on another queue which is the ***ready queue***.

Processes and CPU Scheduling

- A new process initially goes in the ready queue. It waits in this queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of several events could occur:
  1. The process could issue an I/O request, and then be placed in an I/O queue.
  2. The process could create a new sub-process and wait for its termination.
  3. The process could be forcibly removed from the CPU, as a result of an interrupt, and put back in the ready queue.

Processes and CPU Scheduling

Scheduling Concepts

- A process migrates between the various scheduling queues throughout its lifetime. The operating system must select processes from these queues in some fashion. The selection process is the responsibility of the appropriate ***scheduler***.

Processes and CPU Scheduling

Scheduling Concepts

- The ***long-term scheduler*** (or ***job scheduler***) selects processes from the secondary storage and loads them into memory for execution. The ***short-term scheduler*** (or ***CPU scheduler***) selects process from among the processes that are ready to execute, and allocates the CPU to one of them.

Processes and CPU Scheduling

Scheduling Concepts

- The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Because of the brief time between executions, the short-term scheduler must be very fast.

Processes and CPU Scheduling

- The long-term scheduler executes much less frequently. There may be minutes between the creation of new processes in the system. The long-term scheduler controls the ***degree of multiprogramming*** – the number of processes in memory. Because of the longer interval between executions, the long-term scheduler can afford to take more time to select a process for execution.

Processes and CPU Scheduling

Scheduling Concepts

- Some operating systems may have a ***medium-term scheduler***. This removes (swaps out) certain processes from memory to lessen the degree of multiprogramming (particularly when thrashing occurs). At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called ***swapping***.

Processes and CPU Scheduling

Scheduling Concepts

- Switching the CPU to another process requires some time to save the state of the old process and loading the saved state for the new process. This task is known as ***context switch***.
- Context-switch time is pure overhead, because the system does no useful work while switching and should therefore be minimized.

Processes and CPU Scheduling

CPU Scheduler

- Whenever the CPU becomes idle, the operating system (particularly the CPU scheduler) must select one of the processes in the ready queue for execution.
- CPU scheduling decisions may take place under the following four circumstances:
  1. When a process switches from the running state to the waiting state (for example, I/O request, invocation of wait for the termination of one of the child processes)

Processes and CPU Scheduling

CPU Scheduler

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state (for example, completion of I/O).

4. When a process terminates.

Processes and CPU Scheduling

CPU Scheduler

- For circumstances 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however, for circumstances 2 and 3.
- When scheduling takes place only under circumstances 1 and 4, the scheduling scheme is ***nonpreemptive***; otherwise, the scheduling scheme is ***preemptive***.

Processes and CPU Scheduling

- Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or switching states.
- Preemptive scheduling incurs a cost. Consider the case of two processes sharing data. One may be in the midst of updating the data when it is preempted, and the second process is run. The second process may try to read the data, which are currently in an inconsistent state. New mechanisms thus are needed to coordinate access to shared data.

Processes and CPU Scheduling

Scheduling Algorithms

- Different CPU-scheduling algorithms have different properties and may favour one class of processes over another.
- Many criteria have been suggested for comparing CPU-scheduling algorithms. The characteristics used for comparison can make a substantial difference in the determination of the best algorithm. The criteria should include the following:

Processes and CPU Scheduling

Scheduling Algorithms

1. ***CPU Utilization***. This measures how busy is the CPU. CPU utilization may range from 0 to 100 percent. In a real system, it should range from 40% (for a lightly loaded system) to 90% (for a heavily loaded system).
2. ***Throughput***. This is a measure of work (number of processes completed per time unit). For long processes, this rate may be one process per hour; for short transactions, throughput might be 10 processes per second.

Processes and CPU Scheduling

Scheduling Algorithms

3. ***Turnaround Time***. This measures how long it takes to execute a process. Turnaround time is the interval from the time of submission to the time of completion. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing in the CPU, and doing I/O.

Processes and CPU Scheduling

Scheduling Algorithms

4. ***Waiting Time***. 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 a process spends waiting in the ready queue. Waiting time is the total amount of time a process spends waiting in the ready queue.

Processes and CPU Scheduling

Scheduling Algorithms

5. ***Response Time***. The time from the submission of a request until the system makes the first response. It is the amount of time it takes to start responding but not the time that it takes to output that response. The turnaround time is generally limited by the speed of the output device.

Processes and CPU Scheduling

Scheduling Algorithms

- A good CPU scheduling algorithm maximizes CPU utilization and throughput and minimizes turnaround time, waiting time and response time.

- In most cases, the average measure is optimized. However, in some cases, it is desired to optimize the minimum or maximum values, rather than the average. For example, to guarantee that all users get good service, it may be better to minimize the maximum response time.

Processes and CPU Scheduling

Scheduling Algorithms

- For interactive systems (time-sharing systems), some analysts suggests that minimizing the variance in the response time is more important than averaging response time. A system with a reasonable and predictable response may be considered more desirable than a system that is faster on the average, but is highly variable.

Processes and CPU Scheduling

First-Come First-Served(FCFS)

- This is the simplest CPU-scheduling algorithm. The process that requests the CPU first gets the CPU first.
- The average waiting time under the FCFS policy is often quite long.

Processes and CPU Scheduling

First-Come First-Served(FCFS)

- Example:

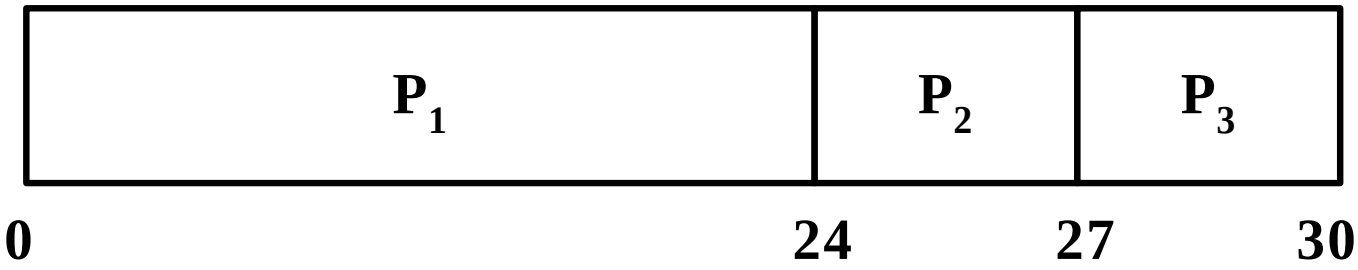
Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

Processes and CPU Scheduling

First-Come First-Served(FCFS)

- If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, the system gets the result shown in the following ***Gantt chart***:



Processes and CPU Scheduling

First-Come First-Served(FCFS)

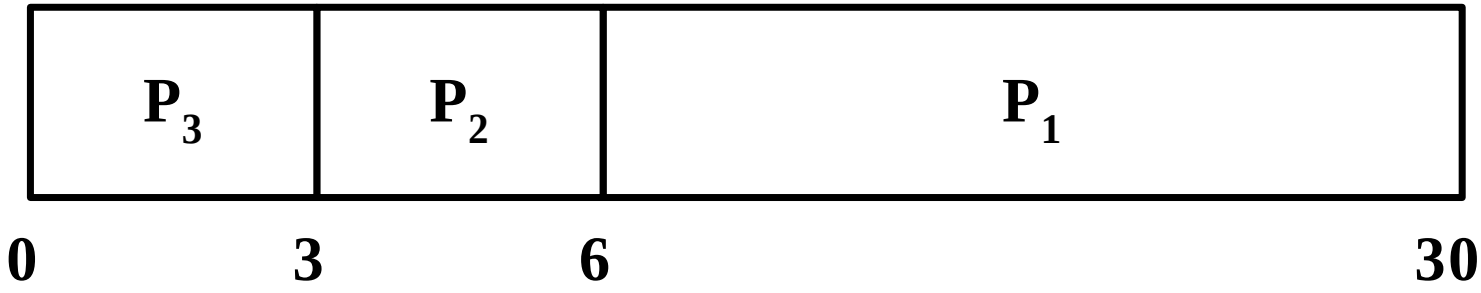
Therefore, the waiting time for each process is:

$$\text{WT for } P_1 = 0 - 0 = 0$$
$$\text{WT for } P_2 = 24 - 0 = 24$$
$$\text{WT for } P_3 = 27 - 0 = 27$$
$$\text{Average waiting time} = (0 + 24 + 27) / 3$$
$$= 17 \text{ ms}$$

Processes and CPU Scheduling

First-Come First-Served(FCFS)

- If the processes arrive in the order P_3, P_2, P_1, however, the results will be:



Processes and CPU Scheduling

First-Come First-Served(FCFS)

Therefore, the waiting time for each process is:

$$\text{WT for } P_1 = 6 - 0 = 6$$
$$\text{WT for } P_2 = 3 - 0 = 3$$
$$\text{WT for } P_3 = 0 - 0 = 0$$
$$\text{Average waiting time} = (6 + 3 + 0) / 3$$
$$= 3 \text{ ms}$$

Processes and CPU Scheduling

First-Come First-Served(FCFS)

- The average waiting time under a FCFS policy is generally not minimal, and may vary substantially if the process CPU-burst times vary greatly.
- The FCFS algorithm is ***nonpreemptive***. Once the CPU has been allocated to a process, the process keeps the CPU until it wants to release the CPU, either by terminating or be requesting I/O. The FCFS algorithm is particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.

Processes and CPU Scheduling

Shortest-Job-First(SJF)

- This algorithm associates with each process the length of the latter's 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 next CPU burst, FCFS scheduling is used to break the tie.

Processes and CPU Scheduling

Shortest-Job First(SJF)

- Example:

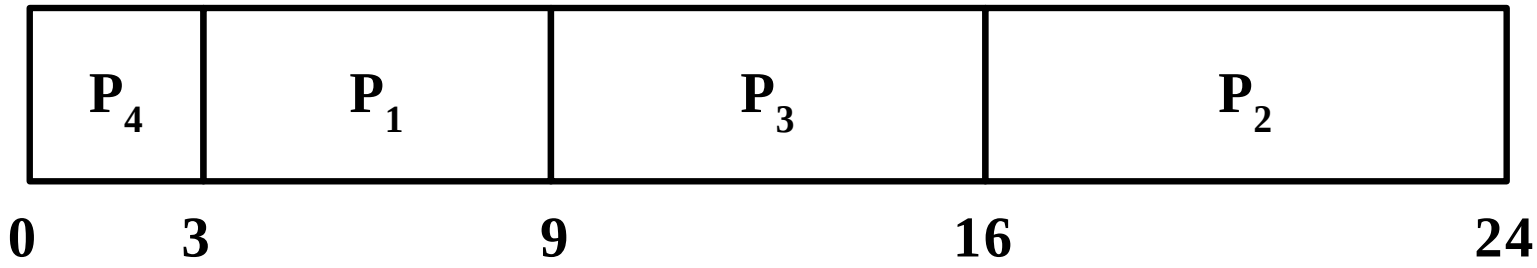
Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Processes and CPU Scheduling

Shortest-Job First(SJF)

- Using SJF, the system would schedule these processes according to the following Gantt chart:



Processes and CPU Scheduling

Shortest-Job First(SJF)

Therefore, the waiting time for each process is:

$$\text{WT for } P_1 = 3 - 0 = 3$$
$$\text{WT for } P_2 = 16 - 0 = 16$$
$$\text{WT for } P_3 = 9 - 0 = 9$$
$$\text{WT for } P_4 = 0 - 0 = 0$$

Average waiting time= (3 + 16 + 9 + 0) / 4

= 7 ms

Processes and CPU Scheduling

Shortest-Job First(SJF)

- If the system were using the FCFS scheduling, then the average waiting time would be 10.25 ms.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term scheduling. There is no way to know the length of the next CPU burst. The only alternative is to predict the value of the next CPU burst.

Processes and CPU Scheduling

Shortest-Job First(SJF)

- The SJF algorithm may be either preemptive or nonpreemptive. A new process arriving may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process. Preemptive SJF scheduling is sometimes called ***shortest-remaining-time-first*** scheduling.

Processes and CPU Scheduling

Shortest-Job First(SJF)

Example:

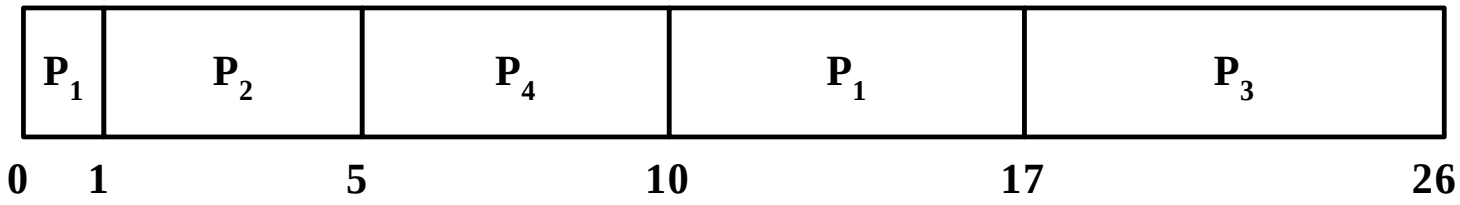
Consider the following set of processes with the length of the CPU burst given in milliseconds:

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

Processes and CPU Scheduling

Shortest-Job First(SJF)

- If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart:



Processes and CPU Scheduling

Shortest-Job First(SJF)

Therefore, the waiting time for each process is:

$$\text{WT for } P_1 = 10 - 1 = 9$$
$$\text{WT for } P_2 = 1 - 1 = 0$$
$$\text{WT for } P_3 = 17 - 2 = 15$$
$$\text{WT for } P_4 = 5 - 3 = 2$$

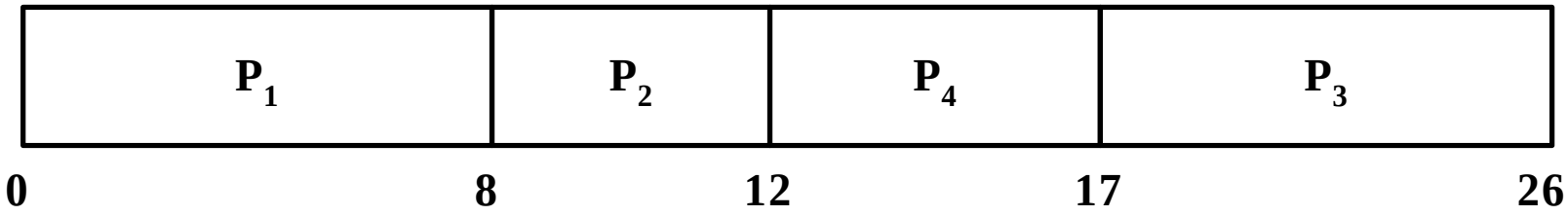
Average waiting time= (9 + 0 + 15 + 2) / 4

= 6.5 ms

Processes and CPU Scheduling

Shortest-Job First(SJF)

- Nonpreemptive SJF scheduling would result in the following schedule:



Processes and CPU Scheduling

Shortest-Job First(SJF)

Therefore, the waiting time for each process is:

$$\text{WT for } P_1 = 0 - 0 = 0$$
$$\text{WT for } P_2 = 8 - 1 = 7$$
$$\text{WT for } P_3 = 17 - 2 = 15$$
$$\text{WT for } P_4 = 12 - 3 = 9$$

Average waiting time= (0 + 7 + 15 + 9) / 4

= 7.75 ms

Processes and CPU Scheduling

Priority Scheduling

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the next CPU burst (τ).

$$p = 1 / \tau$$

- The larger the CPU burst, the lower the priority, and vice versa.

Processes and CPU Scheduling

Priority Scheduling

Example:

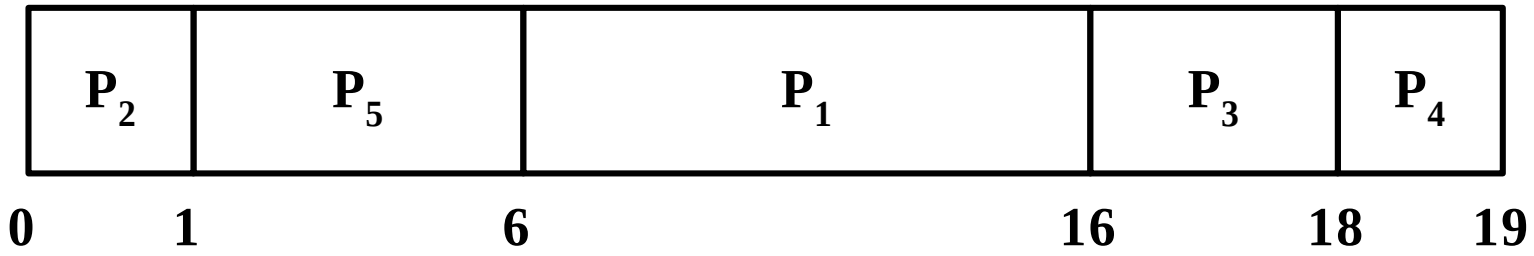
Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

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

Processes and CPU Scheduling

Priority Scheduling

- Using priority algorithm, the schedule will follow the Gantt chart below:



Processes and CPU Scheduling

Priority Scheduling

Therefore, the waiting time for each process is:

$$\begin{aligned}
\text{WT for } P_1 &= 6 - 0 = 6 \\
\text{WT for } P_2 &= 0 - 0 = 0 \\
\text{WT for } P_3 &= 16 - 0 = 16 \\
\text{WT for } P_4 &= 18 - 0 = 18 \\
\text{WT for } P_5 &= 1 - 0 = 1
\end{aligned}$$

$$\text{Average waiting time} = (6 + 0 + 16 + 18 + 1) / 5$$

$$= 8.2 \text{ ms}$$

Processes and CPU Scheduling

Priority Scheduling

- Priority scheduling can either be preemptive or nonpreemptive. When a process arrives at the ready queue, its priority is compared with the priority at the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the currently running process.

Processes and CPU Scheduling

Priority Scheduling

- A major problem with the priority scheduling algorithms is ***indefinite blocking*** or ***starvation***. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Processes and CPU Scheduling

Round-Robin (RR) Scheduling

- This algorithm is specifically for time-sharing systems. A small unit of time, called a ***time quantum*** or ***time slice***, is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. The RR algorithm is therefore preemptive.

Processes and CPU Scheduling

Round-Robin (RR) Scheduling

Example:

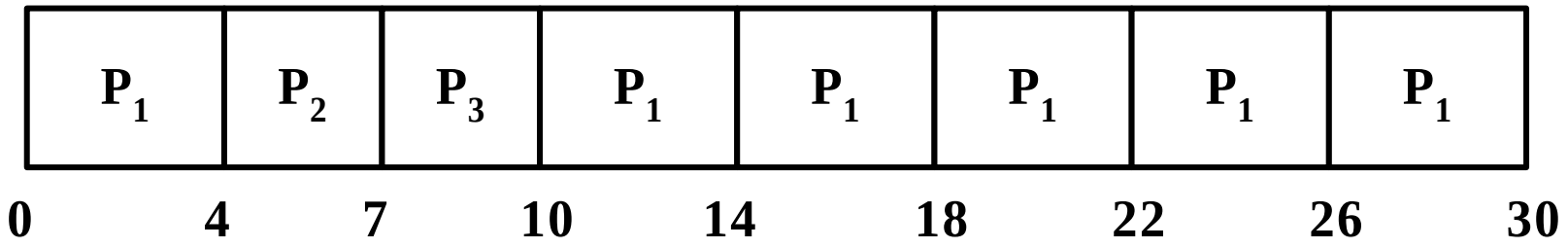
Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

Processes and CPU Scheduling

Round-Robin (RR) Scheduling

- If the system uses a time quantum of 4 ms, then the resulting RR schedule is:



Processes and CPU Scheduling

Round-Robin (RR) Scheduling

Therefore, the waiting time for each process is:

WT for P_1	=	10 - 4=	6
WT for P_2	=	4 - 0=	4
WT for P_3	=	7 - 0=	7

Average waiting time= (6 + 4 + 7) / 3
= 5.67 ms

Processes and CPU Scheduling

Round-Robin (RR) Scheduling

- The performance of the RR algorithm depends heavily on the size of the time quantum. If the time quantum is too large (infinite), the RR policy degenerates into the FCFS policy. If the time quantum is too small, then the effect of the context-switch time becomes a significant overhead.
- As a general rule, 80 percent of the CPU burst should be shorter than the time quantum.