CPU Scheduling

Contents of Lecture:

References for This Lecture:

- ✓ *Abraham Silberschatz, Peter Bear Galvin and Greg Gagne, Operating System Concepts, 9th Edition,* ***Chapter 6***

Basic Concepts

- ❖ In a single-processor system, only one process can run at a time.
  - ✓ Others must wait until the CPU is free and can be rescheduled.
- ❖ The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

- ❖ A process is executed until it must wait. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process.

- ❖ Scheduling of this kind is a fundamental operating-system function. Almost all computer resources are scheduled before use.
  - ✓ The CPU is, one of the primary computer resources. Thus, its scheduling is central to operating-system design.

1. CPU –I/O Burst Cycle

- ❖ The success of CPU scheduling depends on an observed property of processes:
  - ✓ Process execution consists of a **cycle** of CPU execution and I/O wait.
  - ✓ Processes alternate between these two states.
  - ✓ Process execution begins with a **CPU burst**. That is followed by an **I/O burst**, which is followed by another CPU burst, then anotherI/O burst, and so on.
  - ✓ Eventually, the final CPU burst ends with a system request to terminate execution next figure (Figure 6.1).

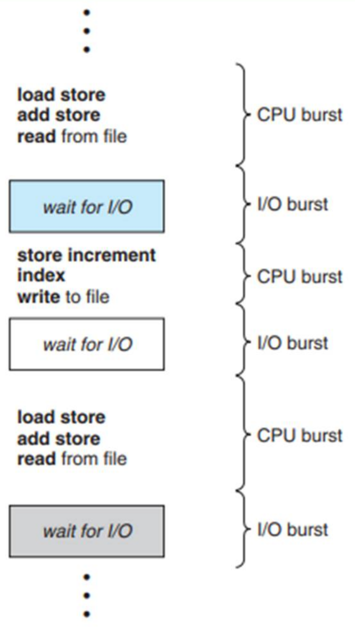


Figure 6.1 Alternating sequence of CPU and I/O bursts.

- ❖ The durations of CPU bursts have been measured extensively. Although they vary greatly from process to process and from computer to computer, they tend to have a frequency curve similar to that shown innext figure (Figure 6.2).
  - ✓ The curve is generally characterized as exponential or hyperexponential, with a large number of short CPU bursts and a small number of long CPU bursts.
  - ✓ An I/O-bound program typically has many short CPU bursts. A CPU-bound program might have a few long CPU bursts.
  - ✓ This distribution can be important in the selection of an appropriate CPU-scheduling algorithm

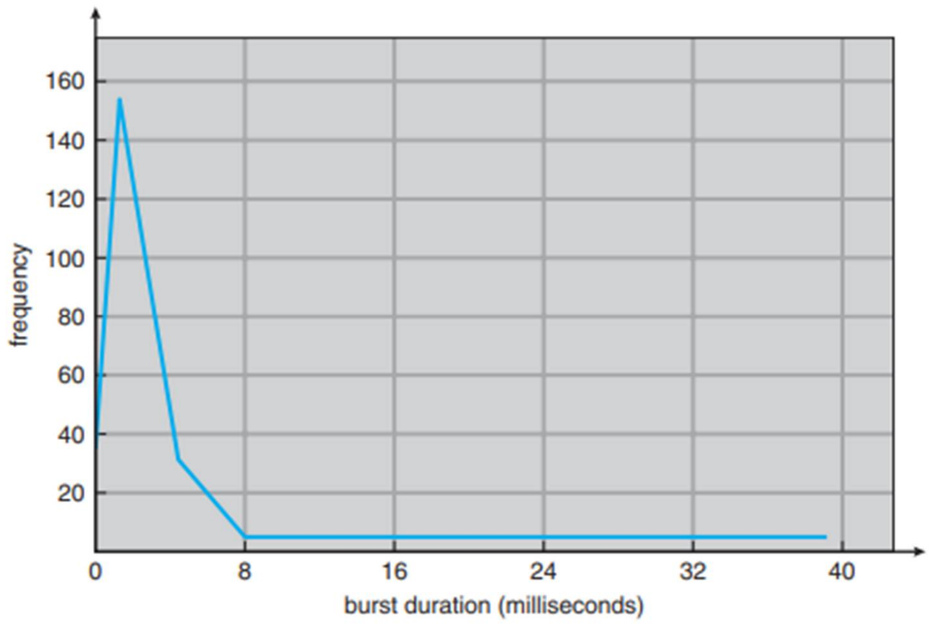


Figure 6.2 Histogram of CPU-burst durations.

2. CPU Scheduler

- When the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the **short-term scheduler**, or **CPU scheduler**.
  - ✓ The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

- A ready queue can be implemented in various ways as:
  - ✓ A FIFO queue
  - ✓ A priority queue
  - ✓ A tree
  - ✓ An unordered linked list.

- All the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally **process control blocks (PCBs)** of the processes.

3. Preemptive Scheduling

- CPU-scheduling decisions may take place under the following:
  1. When a **process switches from the running state to the waiting state** (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)
  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, at completion of I/O)
  4. When a **process terminates**

- Scheduling under 1 and 4 is nonpreemptive.There is a choice, however, for situations 2 and 3 (All other scheduling is **preemptive**):
  - ✓ Consider access to shared data
  - ✓ Consider preemption while in kernel mode
  - ✓ Consider interrupts occurring during crucial OS activities

4. Dispatcher

- Another component involved in the CPU-scheduling function is the dispatcher.
- The dispatcher is the module that gives control of the CPUto the process selected by the short-term scheduler. This function involves the following:
  - ✓ Switching context
  - ✓ Switching to user mode
  - ✓ Jumping to the proper location in the user program to restart that program

- **dispatch latency:** Time it takes for the dispatcher to stop one process and start another running.
  - ✓ The dispatcher should be as fast as possible, since it is invoked during every process switch.

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.

- ❖ Many criteria have been suggested for comparing CPU-scheduling algorithms. The criteria include the following:

  - ✓ **CPU utilization:** keep the CPU as busy as possible
  - ✓ **Throughput:** number of processes that complete their execution per time unit
  - ✓ **Turnaround time:** amount of time to execute a particular process
  - ✓ **Waiting time:** amount of time a process has been waiting in the ready queue
  - ✓ **Response time:** amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

- ❖ Scheduling Algorithm Optimization Criteria:
  - ✓ Max CPU utilization
  - ✓ Max throughput
  - ✓ Min turnaround time
  - ✓ Min waiting time
  - ✓ Min response time

Scheduling Algorithms

- ❖ CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU-scheduling algorithms:

  1. First-Come, First-Served Scheduling
  2. Shortest-Job-First Scheduling
  3. Priority Scheduling
  4. Round-Robin Scheduling

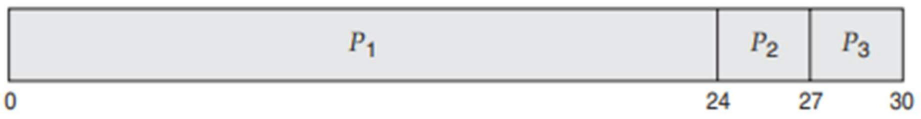
1. First-Come, First-Served Scheduling

- ❖ With this scheme, the process that requests the CPU first is allocated the CPU first.
- ❖ **Non preemptive**: the process continues until the burst cycle end.

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

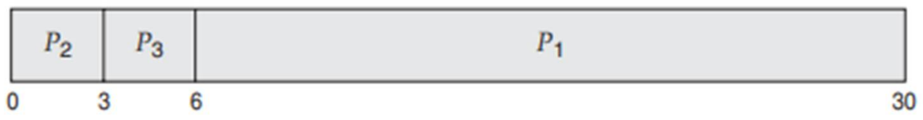
Process	Burst Time
P_1	24
P_2	3
P_3	3

- If the processes arrive in **the order P1, P2, P3**, and are served in FCFS order, get the result shown in the following:



- The waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2, and 27 milliseconds for process P3.
- Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds.

- If the processes arrive in **the order P2, P3, P1**, the results will be as shown in the following:



- The average waiting time is (6 + 0 + 3)/3 = 3 milliseconds.
- **Advantages:**
  - ✓ Simple
  - ✓ Fair(as long as co process hongs the CPU, every process will eventually run)
- **Disadvantages:**
  - ✓ Waiting time depends on arrival order
  - ✓ Short processes stuck waiting for long process to complete (**Convoy effect**)

2. Shortest-Job-First Scheduling

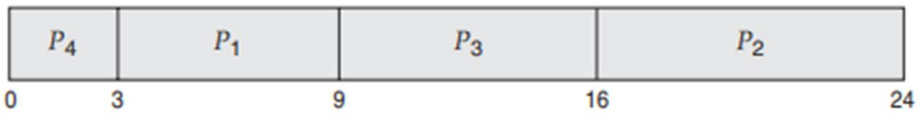
- This algorithm associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
  - ✓ **No preemption:** the process continues to execute until its CPU burst complete.
  - ✓ **With preemption:** the process may get preempted when anew process arrives.

SJF no preemption:

- **Example of SJF scheduling no preemption**:
- Consider the following set of processes, 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

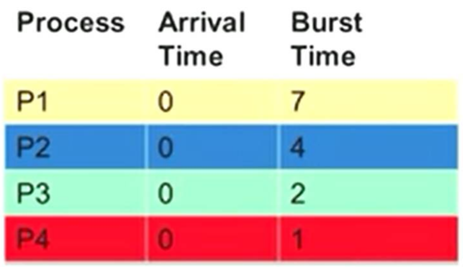
- Using SJF scheduling, would schedule these processes according to the following:



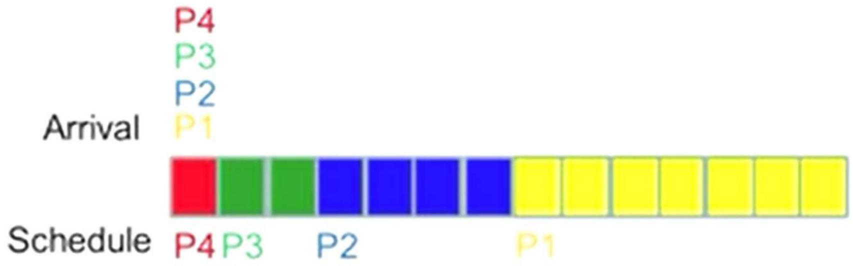
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

❖ **<u>Example of SJF scheduling no preemption:</u>**

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



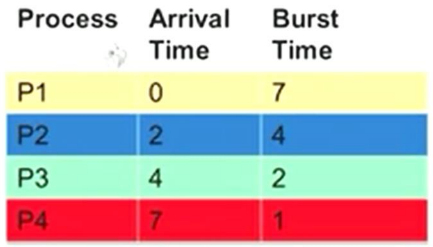
Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	1



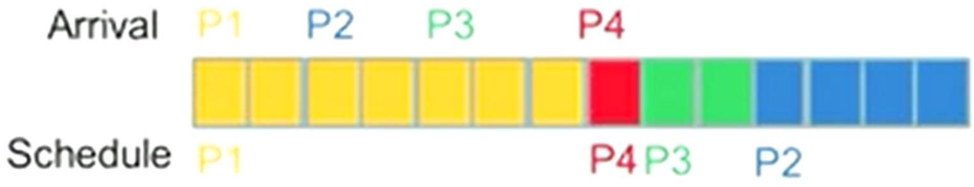
❖ Average waiting time = (7 + 3 + 1 + 0) / 4 = 2.71

❖ **<u>Example of SJF scheduling no preemption:</u>**

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



Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	2
P4	7	1

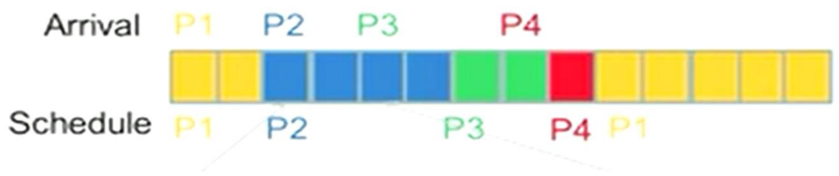


❖ Average waiting time = (0 + 8 + 4 +0) / 4 = 4

SJF with preemption:

❖ Consider the following set of processes:

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	2
P4	7	1



❖ Average waiting time = (7 +0 + 2 +0) / 4 = 2.5

❖ **Advantages:**

✓ Optimal: Minumum average wait time

❖ **Disadvantages:**

✓ Not practical: difficult to predicat burst time

3. 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.

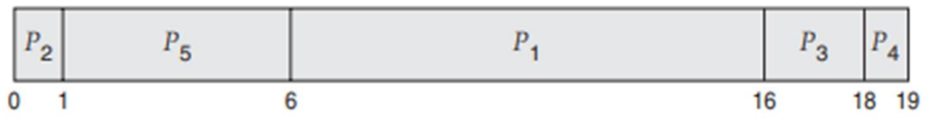
Note that

Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority

❖ As an example, consider the following set of processes, assumed to have arrived at time 0 in the order P1, P2, ···, P5, with the length of the CPU burst given in milliseconds:

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

❖ Using priority scheduling, we would schedule these processes according to the following:



❖ The average waiting time is 8.2 milliseconds

❖ **Problem: Starvation** – low priority processes may never execute

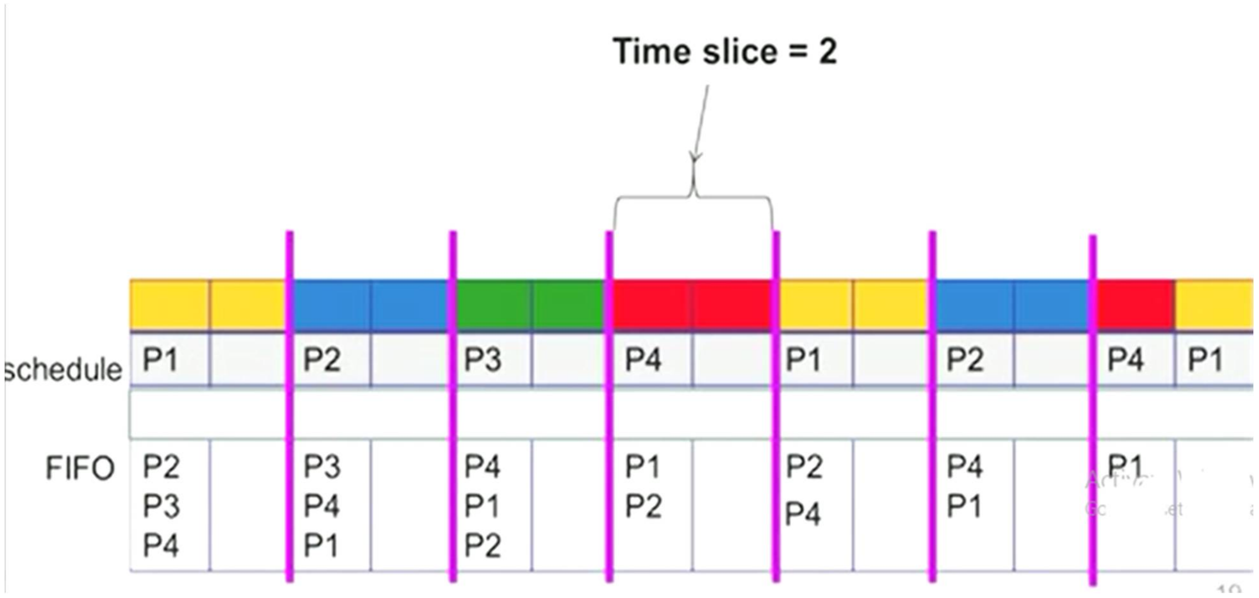
❖ **Solution: Aging** – as time progresses increase the priority of the process

4. Round-Robin Scheduling

- ❖ Run process for a time slice then move back to ready queue.
  - ✓ Configure timer to interrupt periodically
  - ✓ At every timer interrupt, preempt current running process

❖ **Example**: Consider the following set of processes

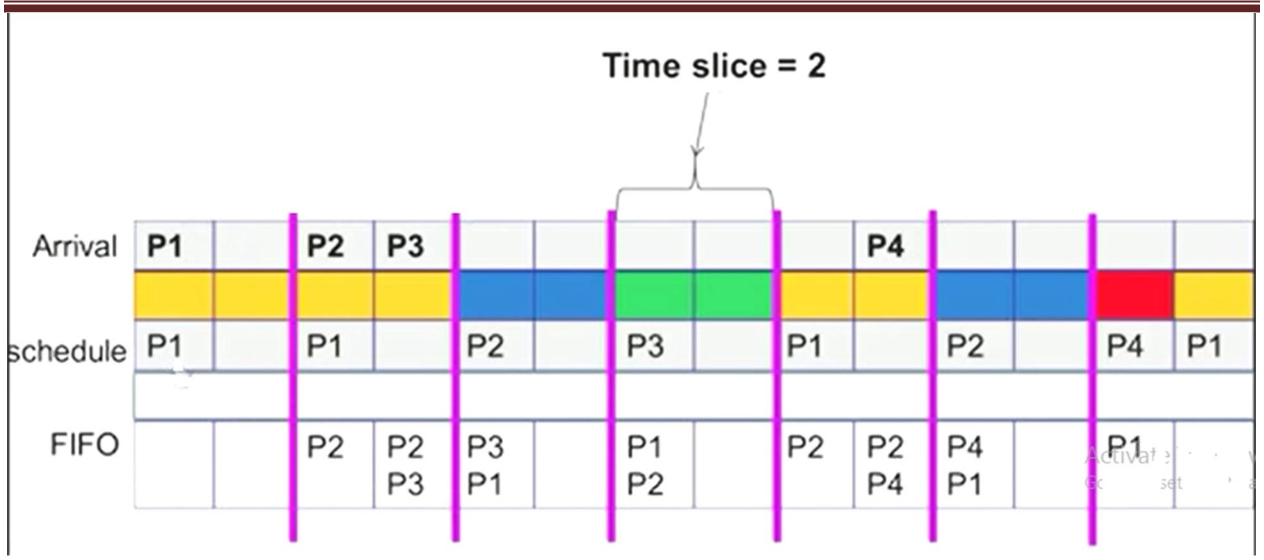
Process	Arrival Time	Burst Time
P1	0	5
P2	0	4
P3	0	2
P4	0	3



- ❖ Average waiting time = (9 +8 + 4 +10) / 4 = 7.75
- ❖ Average response time = (0 +2 + 4 +6) / 4 = 3
- ❖ Context switches = 7

❖ **Example**: Consider the following set of processes

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1



Arrival	**P1**		**P2**	**P3**						**P4**				
	🟨	🟨	🟨	🟨	🟦	🟦	🟩	🟩	🟨	🟨	🟦	🟦	🟥	🟨
schedule	P1		P1		P2		P3		P1		P2		P4	P1
FIFO			P2	P2 P3	P3 P1		P1 P2		P2	P2 P4	P4 P1		P1	

- ❖ Average waiting time = (7 +6 + 3 +3) / 4 = 4.75
- ❖ Context switches = 7