CPU Scheduling

CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling

Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

Basic Concepts

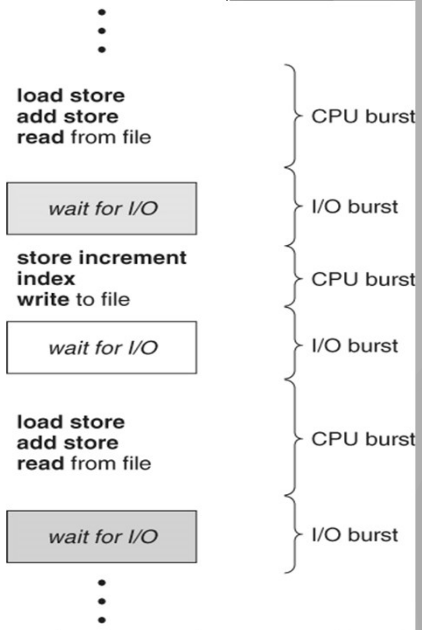
- In a single-processor system, only one process can run at a time.
- Others must wait until the CPU is free and can be rescheduled.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- Every time one process has to wait, another process can take over use of the CPU.
- Scheduling of this kind is a fundamental operating-system function.
- Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design.

What is CPU–I/O Burst Cycle?

- CPU Burst : It is the amount of time required by a process or can be said the amount of time required by the process to finish.
- We can not estimate the time taken by the process before running it. So most of the problem is related to the burst time.
- Burst Time= Turn around Time(Completion Time)-Waiting Time
- I/O Burst : While the process is in the running state, it may ask for i/o , thus the process goes to the block or wait state, where the i/o will be processed and then it will be sent back to the ready state

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern

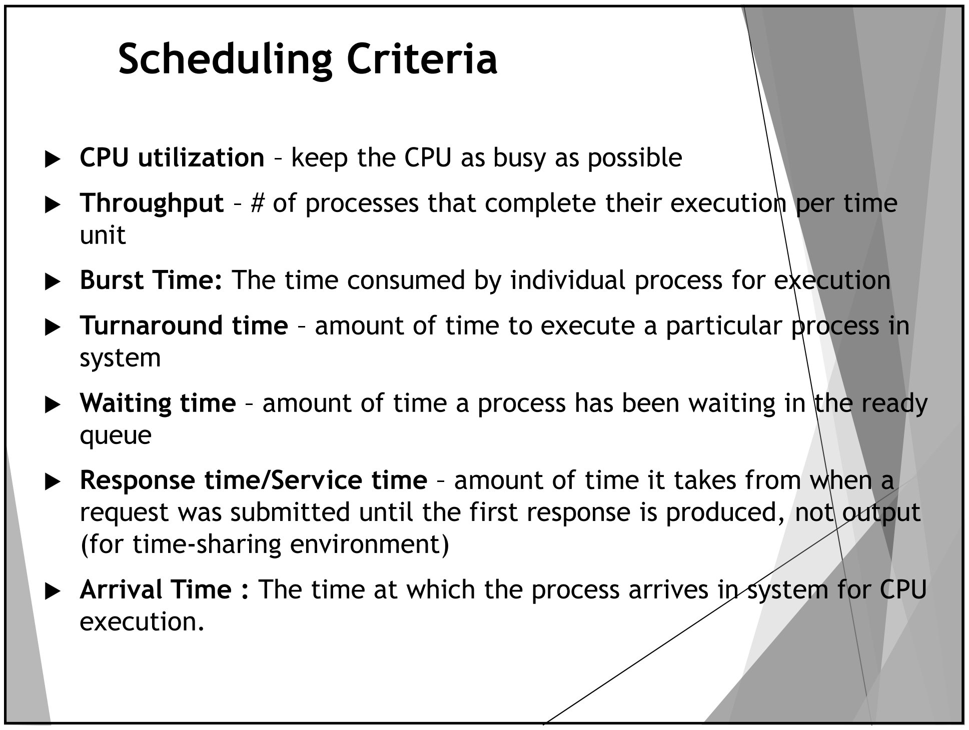


CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **non-preemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

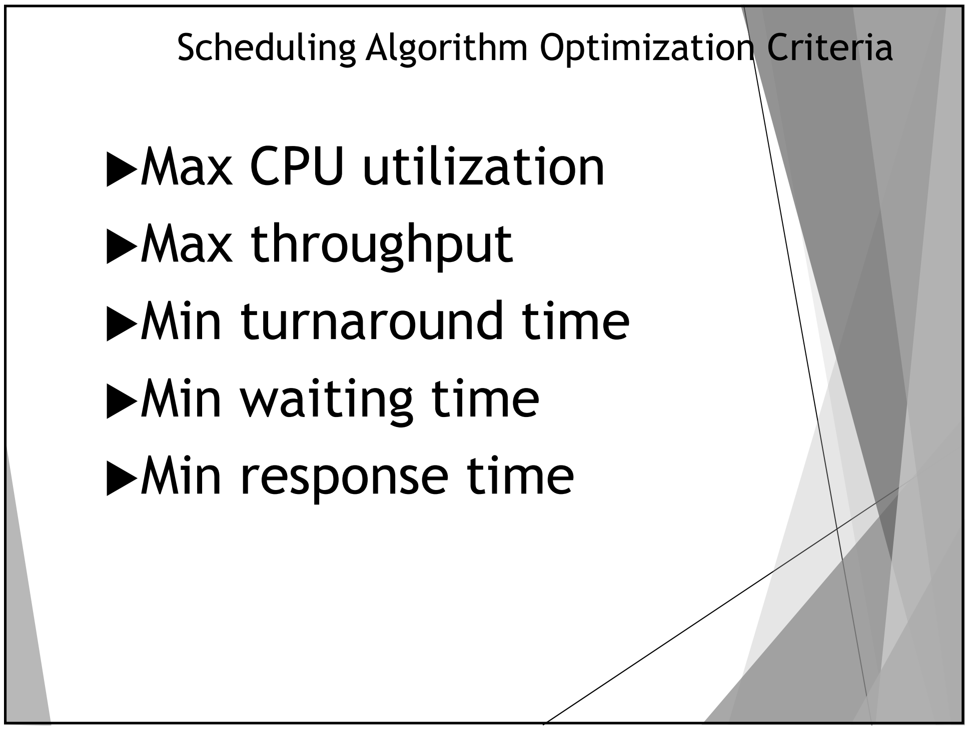
Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - 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



Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Burst Time:** The time consumed by individual process for execution
- **Turnaround time** – amount of time to execute a particular process in system
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time/Service time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
- **Arrival Time :** The time at which the process arrives in system for CPU execution.



Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Preemptive and Non-Preemptive Scheduling

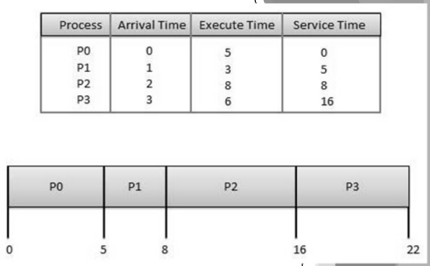
- **Preemptive Scheduling**
- Preemptive Scheduling means once a process started its execution, the currently running process can be paused for a short period of time to handle some other process of higher priority, it means we can preempt the control of CPU from one process to another if required.
- A computer system implementing this supports multi-tasking as it gives the user impression that the user can work on multiple processes.
- It is practical because if some process of higher priority is encountered then the current process can be paused to handle that process.
- **Examples:-** SRTF, Priority, Round Robin, etc.
- **Non-Preemptive Scheduling**
- Non-Preemptive Scheduling means once a process starts its execution or the CPU is processing a specific process it cannot be halted or in other words we cannot preempt (take control) the CPU to some other process.
- A computer system implementing this cannot support the execution of process in a multi task fashion. It executes all the processes in a sequential manner.
- It is not practical as all processes are not of same priority and are not always known to the system in advance.
- **Examples:-** FCFS, SJF, Priority, etc.

Difference between Preemptive and Non-Preemptive Scheduling in OS

Preemptive Scheduling	**Non-Preemptive Scheduling**
Processor can be preempted to execute a different process in the middle of execution of any current process.	Once Processor starts to execute a process it must finish it before executing the other. It cannot be paused in middle.
CPU utilization is more compared to Non-Preemptive Scheduling.	CPU utilization is less compared to Preemptive Scheduling.
Waiting time and Response time is less.	Waiting time and Response time is more.
The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.	When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.
If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Ex:- SRTF, Priority, Round Robin, etc.	Ex:- FCFS, SJF, Priority, etc.

First- Come, First-Served (FCFS) Scheduling

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



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

Wait time of each process is as follows −

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

•Average Wait Time:

•(0+4+6+13) / 4 = 5.75

Shortest-Job-First (SJF) Scheduling

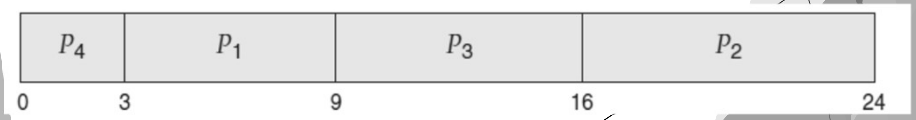
- **Shortest Job Next (SJN)**
- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive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

SJF scheduling-Non-Preemptive

- As an example of SJF scheduling, consider the following set of processes, with the length of the CPU burst given in milliseconds

Process	Burst Time	Turn Around Time	Waiting Time(TAT-BT)	Average Waiting Time	Average TAT
P1	6	9	3	(3+16+9+)) /4=7	(9+24+16+3) /4=13
P2	8	24	16		
P3	7	16	9		
P4	3	3	0		

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

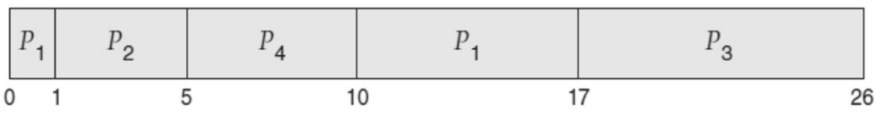


SJF scheduling-Preemptive and Non-Preemptive

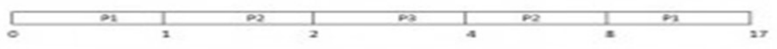
- The SJF algorithm can be either preemptive or non-preemptive.
- The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first scheduling.**

Example of SJF

Process	Arrival time	Burst Time	Turn Around Time	Waiting Time(TAT-BT)	Average Waiting Time	Average TAT
P1	0	8	17	9		
P2	1	4	5	1		
P3	2	9	26	17		
P4	3	5	10	5		



process	CPU Burst Time	Time of Arrival
p1	10	0
p2	5	1
p3	2	2



In this, the CPU will be taken away from the currently executing process whenever a process will less CPU burst time.

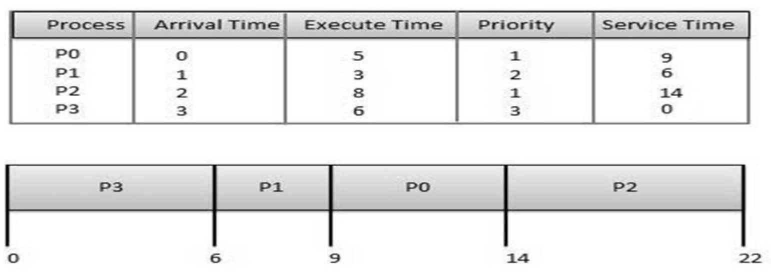
As shown in figure, the time when P2 arrives P1 needs 9 millisecond more to finish. As B's cpu burst in 5 millisecond < 9 millisecond, therefore, P1's execution will be preempted and P2 will be executed but against as P3 arrives P2's execution needs 3 more millisecond where as P3 needs only 2 millisecond to execute, thus P3 takes over P2 and so on.

Waiting time for P1 = 0+ (8-1) = 7 millisecond
Waiting time for P2 = 1+ (4-2) = 3 millisecond
Waiting time for P3 = 2 millisecond
Average waiting time = (7+3+2) / 3 = 4 millisecond

Priority Based Scheduling

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



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

Wait time of each process is as follows −
Average Wait Time: (9+5+12+0) / 4 = 6.5

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

Priority scheduling preemptive or non-preemptive

- Priority scheduling can be either preemptive or non-preemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of 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 priority of the currently running process.
- A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Disadvantages of Priority scheduling

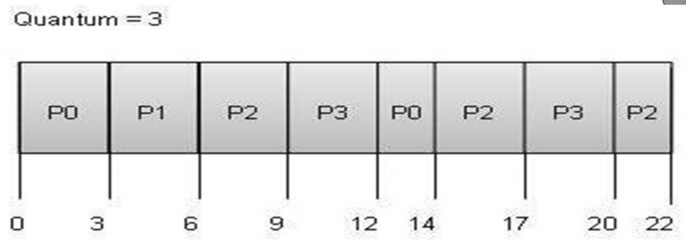
- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**.
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely.

Round Robin Scheduling

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

The Round-Robin (RR) scheduling algorithm

- The **round-robin (RR)** scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- A small unit of time, called a **time quantum** or **time slice**, is defined.
- A time quantum is generally from10 to 100 milliseconds in length.
- 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



Wait time of each process is as follows −
Average Wait Time: (9+2+12+11) / 4 = 8.5

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

Multiple Processor Scheduling

- In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible.
- However multiple processor scheduling is more **complex** as compared to single processor scheduling.
- In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling –

- One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**.
- This is simple and reduces the need of data sharing.
- This entire scenario is called **Asymmetric Multiprocessing**.
- A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**.
- All processes may be in a common ready queue or each processor may have its own private queue for ready processes.
- The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity

- Processor Affinity means a processes has an **affinity** for the processor on which it is currently running.
- When a process runs on a specific processor there are certain effects on the cache memory.
- The data most recently accessed by the process populate the cache for the processor and as a result successive memory access by the process are often satisfied in the cache memory.
- Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated.
- Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as **PROCESSOR AFFINITY**

Affinity

- There are two types of processor affinity:
- **Soft Affinity** – When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.
- **Hard Affinity** – Some systems such as Linux also provide some system calls that support Hard Affinity which allows a process to migrate between processors

Load Balancing

- Load Balancing is the **phenomena** which keeps the **workload** evenly **distributed** across all processors in an SMP system.
- Load balancing is necessary only on systems where each processor has its own private queue of process which are eligible to execute.
- Load balancing is unnecessary because once a processor becomes idle it immediately extracts a runnable process from the common run queue.
- On SMP(symmetric multiprocessing), it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor else one or more processor will sit idle while other processors have high workloads along with lists of processors awaiting the CPU.

Load Balancing

- There are two general approaches to load balancing
- **Push Migration -** In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.
- **Pull Migration -** Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.