Applied Operating System
Study Guide

Module Number
2

SUBTOPIC 1: PROCESS MANAGEMENT

OBJECTIVES

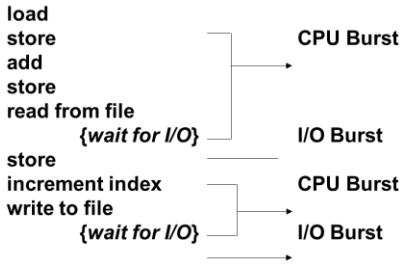
Upon completion of this module, the student will be able to:

- Explain the process concept, architecture and types of computer processes;
- Describe different states of a process and the process control block;
- Discuss concurrent processes;
- Give the concept of scheduling and CPU scheduler;
- Solve CPU scheduling problems using different CPU scheduling algorithms/techniques;

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**.
- 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.
- 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.

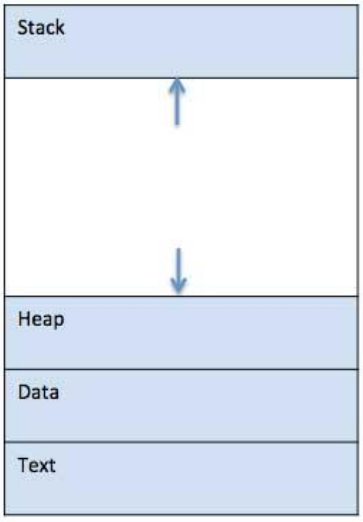
Example:



PROCESS ARCHITECTURE

- To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The figure shows a simplified layout of a process inside main memory



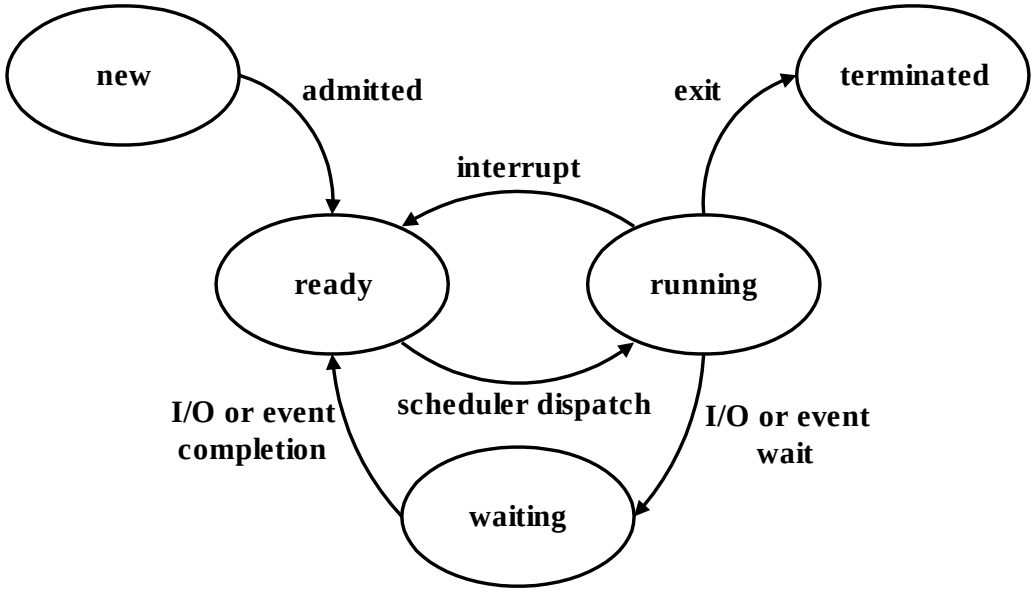
Stack	**The process Stack contains the temporary data such as method/function parameters, return address and local variables.**
Heap	**This is dynamically allocated memory to a process during its run time.**
Text	**This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.**
Data	**This section contains the global and static variables.**

PROCESS STATE

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.
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.

PROCESS STATE DIAGRAM



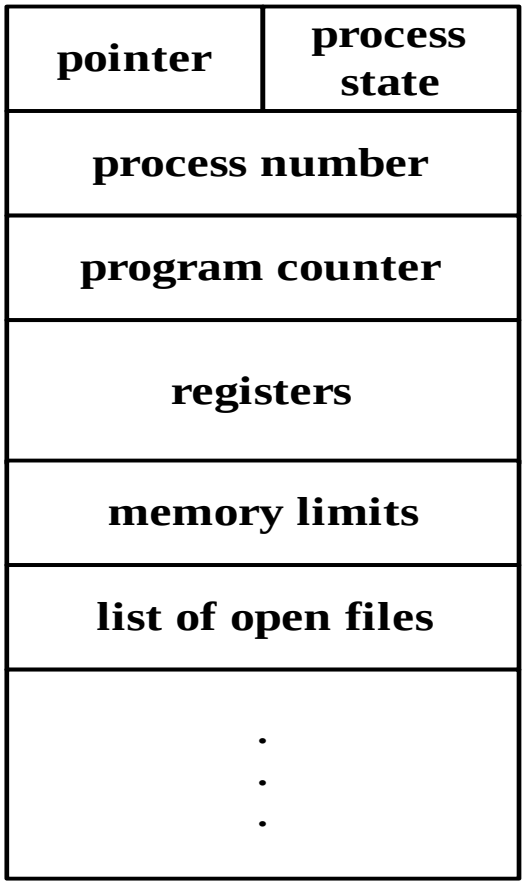
PROCESS CONTROL BLOCK

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.
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.

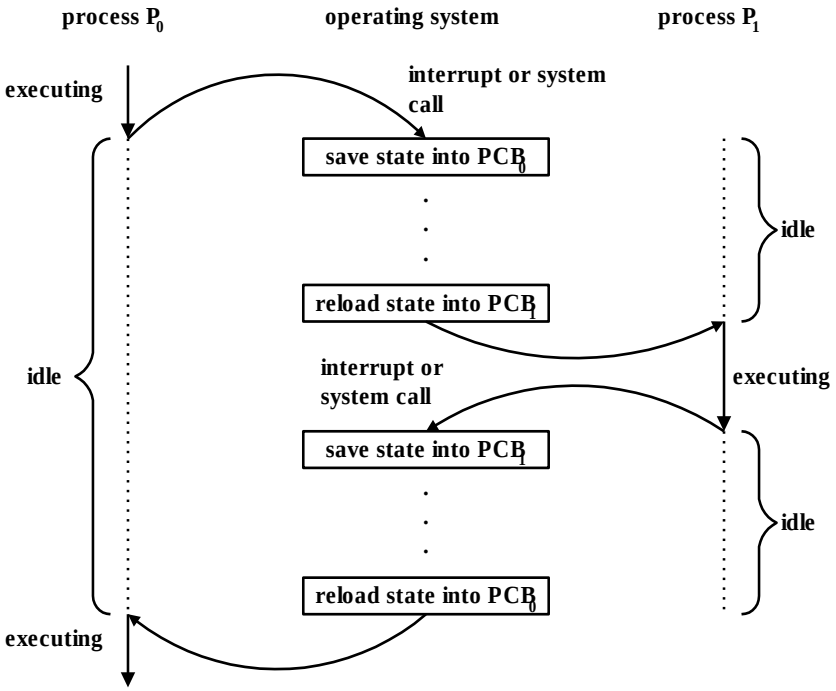
4. **CPU Scheduling Information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
5. **Memory Management Information.** This information includes limit registers or page tables.
6. **Accounting Information.** This information includes 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.

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



PROCESS CONCEPT

Example of the CPU being switched from one process to another. This is also known **as Context Switch Diagram**



CONCURRENT PROCESSES

- 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.
- When a process creates a sub-process, the sub-process may be able to obtain its resources directly from the OS 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.

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.

- 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.
    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**.

The concurrent processes executing in the OS may either be independent processes or cooperating processes.

- 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.
    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.

- 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.

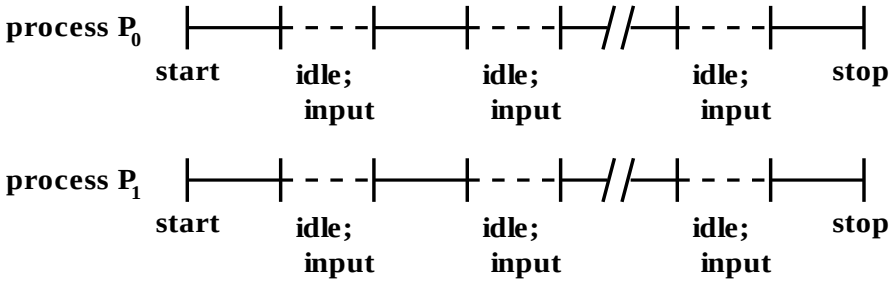
Concurrent execution of cooperating process requires mechanisms that allow processes to communicate with one another and to synchronize their actions.

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).

Example:

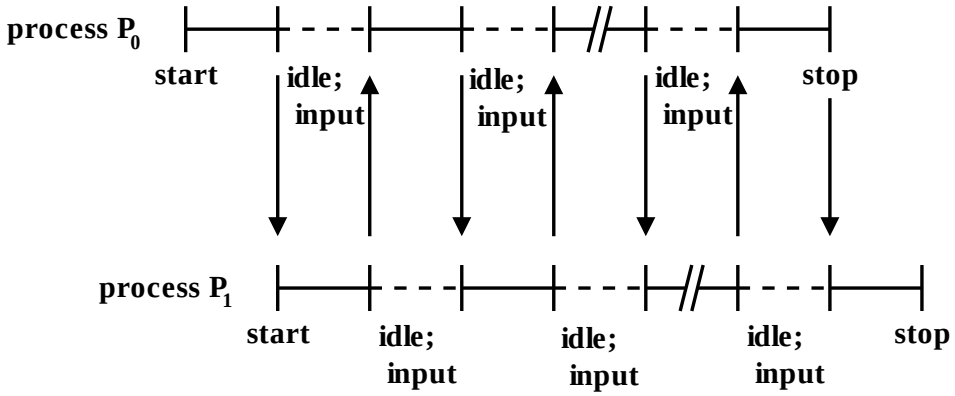
Given two processes, P_0 and P_1.



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

- 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.

Example: Applying multiprogramming to the two processes, P_0 and P_1



*Then CPU utilization increases to **100%**.*

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

TYPES OF SCHEDULER

- **Long-term scheduler** (or J**ob scheduler**) selects processes from the secondary storage and loads them into memory for execution.
- 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.
- **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.
- 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.
- **Medium-term scheduler** 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**.

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**.
- Whenever the CPU becomes idle, the OS (particularly the CPU scheduler) must select one of the processes in the ready queue for execution.

CPU SCHEDULER

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).
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.

- 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 **non-preemptive**; otherwise, the scheduling scheme is **preemptive**.

CPU SCHEDULER

- **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 switching states. **No process is interrupted** until it is **completed**, and after that processor switches to another process.

- **Preemptive scheduling** works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

CPU 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: *CPU Utilization, Throughput, Turnaround Time, Waiting Time, and Response Time*

1. **CPU Utilization** 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** is the amount of work completed in a unit of time. In other words throughput is the processes executed to number of jobs completed in a unit of time. The scheduling algorithm must look to maximize the number of jobs processed per time unit.

3. **Turnaround Time** 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.

4. **Waiting Time** is the time a job waits for resource allocation when several jobs are competing in multiprogramming system. Waiting time is the total amount of time a process spends waiting in the ready queue.

5. **Response Time** is 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.

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.
- 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.

Non-preemptive:

- First-Come First-Served(FCFS)
- Shortest Job First (SJF)
- Priority Scheduling (Non-preemptive)

Preemptive:

- Shortest Remaining Time First (SRTF)
- Priority Scheduling (Preemptive)
- Round-robin (RR)

SUBTOPIC 2: CPU SCHEDULING TECHNIQUES – NON PREEMPTIVE

FIRST-COME FIRST-SERVED (FCFS)

- **FCFS** is the **simplest** CPU-scheduling algorithm.
- The process that requests the CPU first gets the CPU first.
- The FCFS algorithm is **non-preemptive**.

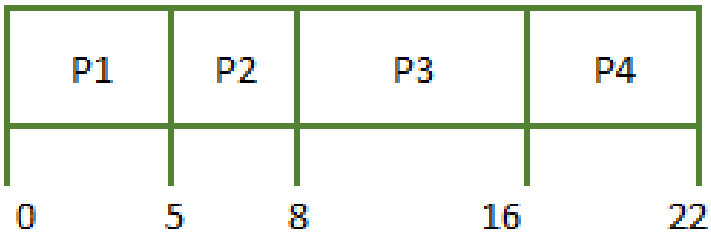
Example 1:

- Consider the following set of processes that arrive at **time 0**, with the length of the CPU burst given in milliseconds (ms).
- Illustrate the **Gantt chart** and compute for the **average waiting time** and **average turnaround time**

Given:

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	0-0 = **0**	5-0 = **5**
P2	0	3	5-0 = **5**	8-0 = **8**
P3	0	8	8-0 = **8**	16-0 =**16**
P4	0	6	16-0 = **16**	22-0 = **22**
Average			**29/4 7.25 ms**	**51/4 12.75 ms**

Gantt Chart:



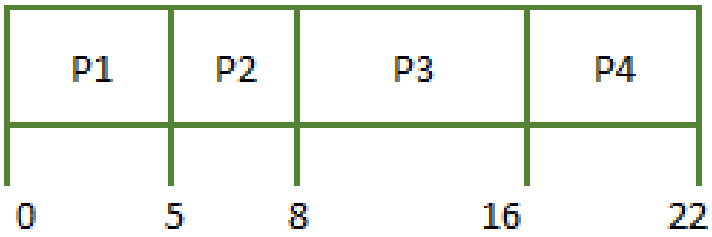
Formulas:

- **Waiting time** = Start time – Arrival time
- **Turnaround time** = Completion time – Arrival time
- **Average Waiting Time** = Sum of Waiting time / No. of processes
- **Average Waiting Time** = Sum of Turnaround time / No. of processes

Example 2:

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	0-0 = 0	5-0 = 5
P2	1	3	5-1 = 4	8-1 = 7
P3	2	8	8-2 =6	16-2 = 14
P4	3	6	16-3 = 13	22-3 = 19
Average			23/4 5.75 ms	45/4 11.25 ms

Gantt Chart:



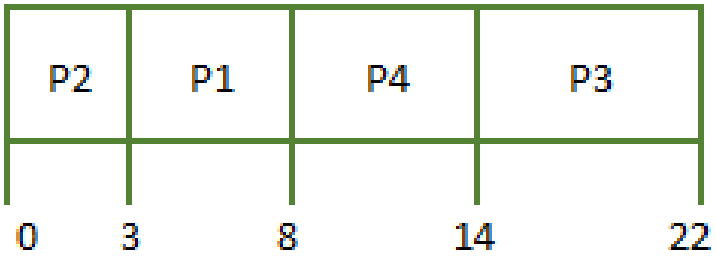
SHORTEST JOB FIRST (SJF)

- **SJF** 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**.
- SJF algorithm is **non-preemptive**.

Example 1:

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	3-0 = 3	8-0 = 8
P2	0	3	0-0 = 0	3-0 =3
P3	0	8	14-0 = 14	22-0 = 22
P4	0	6	8-0 = 8	14-0 =14
Average			**25/4 6.25 ms**	**47/4 11.75 ms**

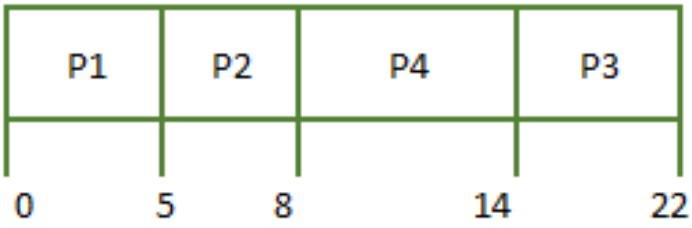
Gantt Chart:



Example 2:

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	0-0 = **0**	5-0 = **5**
P2	1	3	5-1 = **4**	8-1 = **7**
P3	2	8	14-2 =**12**	22-2 = **20**
P4	3	6	8-3 = **5**	14-3 = **11**
Average			**21/4 5.25 ms**	**43/4 10.75 ms**

Gantt Chart:



PRIORITY SCHEDULING (NP)

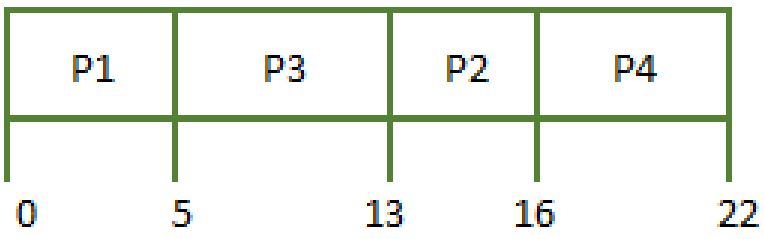
- **Priority scheduling (non-preemptive)** algorithm is one of the most common scheduling algorithms in batch systems.
- Each process is assigned a **priority**.
- Process with **highest priority** is to be **executed first** and so on.
- Processes with **same priority** are executed on **FCFS** basis.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

Example 1:

Consider the **lowest priority value** gets the **highest priority**. In this case, process with priority value (1) is the highest priority.

Process	Arrival Time	Burst time	Priority	Waiting Time	Turnaround Time
P1	0	5	1	0-0 = **0**	5-0 = **5**
P2	0	3	2	13-0 = **13**	16-0 = **16**
P3	0	8	1	5-0 = **5**	13-0 = **13**
P4	0	6	3	16-0 = **16**	22-0 = **22**
Average				**34/4 8.5 ms**	**56/4 14 ms**

Gantt Chart:



SUBTOPIC 3: CPU SCHEDULING TECHNIQUES–PREEMPTIVE

SHORTEST JOB FIRST (SJF)

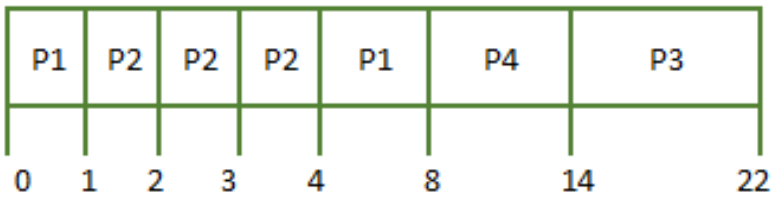
- The SJF algorithm may be either **preemptive** or **non-preemptive**.
- A new process arriving may have a shorter next CPU burst than what is left of the currently executing process.
- A preemptive SJF algorithm will **preempt** the currently executing process.
- **Preemptive SJF** scheduling is sometimes called **Shortest Remaining Time First** scheduling.

SHORTEST REMAINING TIME FIRST (SRTF)

Example :

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	(0-0) + (4-1) = 3	8-0 = 8
P2	1	3	1-1 = 0	4-1 = 3
P3	2	8	14-2 =12	22-2 = 20
P4	3	6	8-3 = 5	14-3 = 11
Average			20/4 5 ms	42/4 10.5 ms

Gantt Chart:



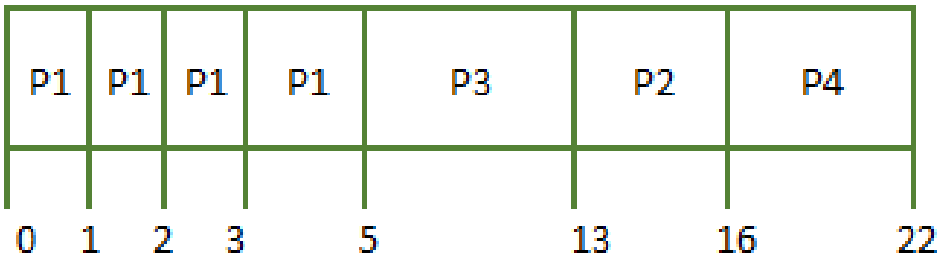
PRIORITY SCHEDULING (P)

- Priority scheduling can either be **preemptive** or **non-preemptive**.
- 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.

Example 3:

Process	Arrival Time	Burst time	Priority	Waiting Time	Turnaround Time
P1	0	5	1	0-0 = 0	5-0 = 5
P2	1	3	2	13-1 = 12	16-1 = 15
P3	2	8	1	5-2 = 3	13-2 = 11
P4	3	6	3	16-3 = 13	22-3 = 19
Average				**28/4 7 ms**	**50/4 12.5 ms**

Gantt Chart:



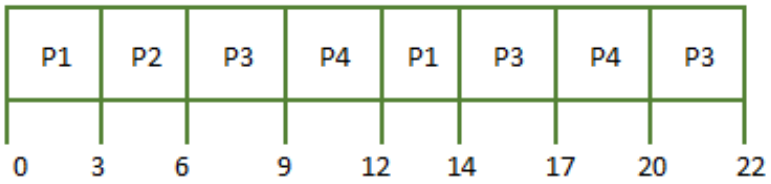
ROUND-ROBIN (RR) SCHEDULING

- 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**.

Example:
Time Quantum = 3ms

Process	Arrival Time	Burst time	Waiting Time	Turnaround Time
P1	0	5	(0-0 + 12-3) = 9	14-0 = 14
P2	1	3	3-1 = 2	6-1 = 5
P3	2	8	(6-2 + 14-9 + 20-17) =12	22-2 = 20
P4	3	6	(9-3 + 17-12) = 11	20-3 = 17
Average			34/4 8.5 ms	56/4 14 ms

Gantt Chart:



- 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**.

REFERENCES

- *Siberschatz, A. (2018). Operating System Concepts, Wiley.*
- *Tomsho, G. (2019). Guide to Operating Systems, Cengage Learning.*