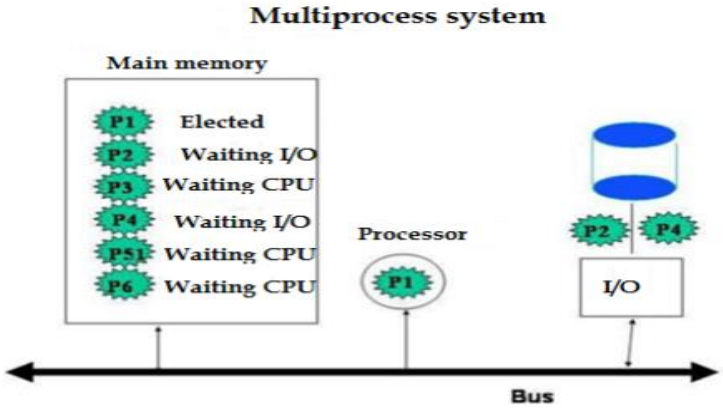
Level Students:2nd year bachlor's degree in Computer Science **Matter**: Operating System 1

Chapter 2: Process management (Prossessor Management)

1. Introduction

- Process management is a core function of an Operating System (OS). It deals with creating, scheduling, and coordinating processes to ensure efficient CPU utilization and smooth system performance. [1]
  - Single-tasking systems are easy to manage since only one process runs at a time.
  - Multiprogramming/multitasking systems are more complex, as multiple processes need to share the CPU efficiently.
  - Active processes may share memory and other resources, requiring careful management.
  - Process synchronization is necessary when processes interact or communicate to avoid conflicts.



2. Process definition

A process can be defined as a running program. In other words, a program by itself is not a process. A program is a passive entity, like the contents of a file stored on disk, while a process is an active (dynamic) entity, with an instruction counter specifying the next instruction to be executed and a set of associated resources.

Obviously, it is possible to have several different processes associated with the same program. This is the case, for example, of several users who each run a copy of the messaging program. It is also common for one process to in turn spawn multiple processes while running.

3. Process Management Tasks

Process management is a key part in operating systems with multi-programming or multitasking.

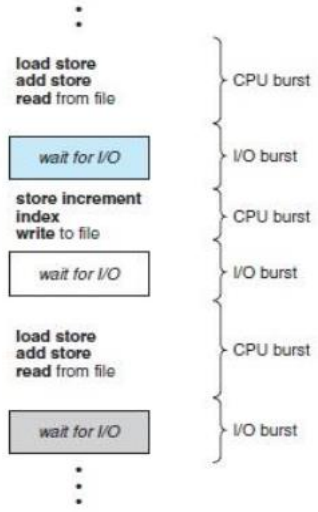
- **Process Creation and Termination**: Process creation involves creating a Process ID, setting up Process Control Block, etc. A process can be terminated either by the operating system or by the parent process. Process termination involves clearing all resources allocated to it.
- **CPU Scheduling**: In a multiprogramming system, multiple processes need to get the CPU. It is the job of Operating System to ensure smooth and efficient execution of multiple processes.
- **Deadlock Handling**: Making sure that the system does not reach a state where two or more processes cannot proceed due to cyclic dependency on each other.
- **Inter-Process Communication**: Operating System provides facilities such as shared memory and message passing for cooperating processes to communicate.
- **Process Synchronization:** Process Synchronization is the coordination of execution of multiple processes in a multiprogramming system to ensure that they access shared resources (like memory) in a controlled and predictable manner.

3.1 CPU Scheduling

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. The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try to use this time productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. 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.

3.2. 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 another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution (Figure 4.1).



Alternating sequence of CPU and I/O bursts

3.3. CPU Scheduler

Whenever 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. Note that the ready queue is not necessarily a first-in, first-out (FIFO) queue. As we shall see when we consider the various scheduling algorithms, a ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. Conceptually, however, 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*** (**PCB**s) of the processes.

4. States of a Process in Operating Systems

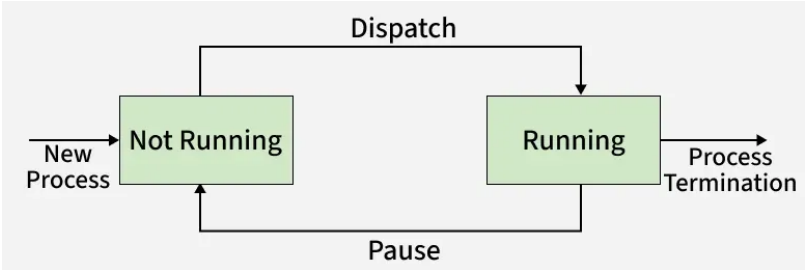
A process in an operating system passes through multiple states as it begins execution, waits for resources, gets scheduled, runs, and eventually finishes. These stages collectively describe the complete lifecycle of a process. Common types of process state models include Two State, Five State and Seven State models.

4.1 Two-State Model

The Two-State Model divides the process lifecycle into only two possible conditions. A process is either actively using the CPU to execute instructions or waiting until it gets a chance to run. This model helps illustrate the basic interaction between a process and the CPU.

The two states are

- **Running:** The process is currently using the CPU to execute its tasks.
- **Not Running:** The process is not using the CPU. It may be waiting for input, waiting for resources, or simply paused.

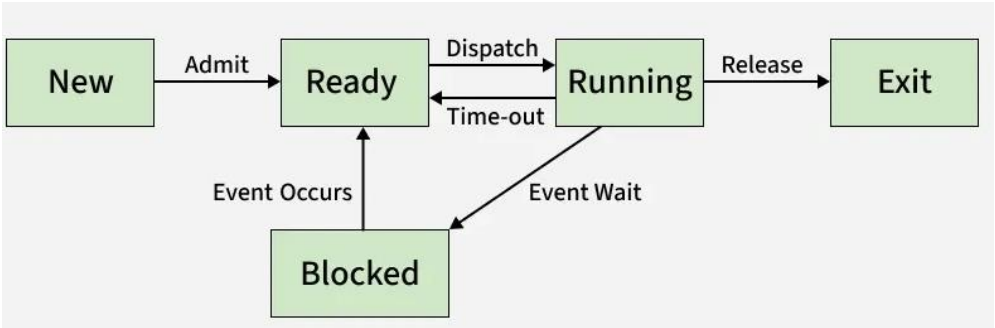


When a new process is created, it starts in the not running state. Initially, this process is kept in a program called **the dispatcher**. here's what happens step by step:

1. **Not Running State**: A newly created process starts here since it is not yet using the CPU.
2. **Dispatcher Role:** The dispatcher checks if the CPU is free for a new process.
3. **Moving to Running State**: If the CPU is available, the dispatcher loads the process onto it, shifting it to the running state.
4. **CPU Scheduler Role:** The scheduler selects which process should run next based on the OS's scheduling policy.

4.2 The Five-State Model

The five-state process lifecycle expands the basic two-state idea by separating processes that are simply waiting for CPU time from those that cannot run because they are waiting for an external event. This makes the model more accurate for real operating systems, where processes may pause for input, data, or device responses before continuing.

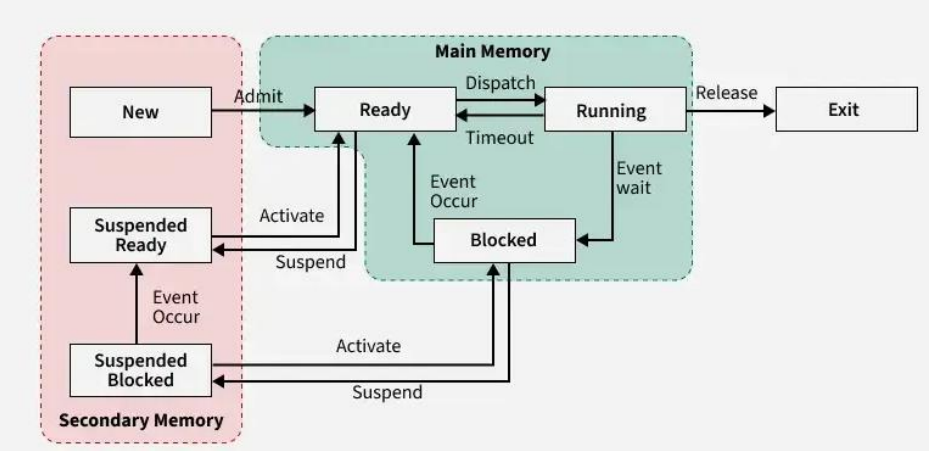


The five states are:

- **New**: The process has just been created. It hasn't started running yet, but its PCB (Process Control Block) is prepared.
- **Ready**: The process is loaded into memory and prepared to run as soon as the CPU becomes free.
- **Running**: The CPU is currently executing this process. With a single CPU system, only one process can be in this state at a time.
- **Blocked/Waiting**: The process cannot continue execution because it is waiting for an event like I/O completion or data input.
- **Exit/Terminate**: The process has completed its execution or has been stopped, and the operating system removes it from memory.

4.3 The Seven-State Model

A process moves through different states as it is created, prepared for execution, runs on the CPU, waits for resources, or completes. These states help the operating system manage how processes use memory, CPU, and I/O devices.



The main ***process states*** are:

- **New**: The process is in the creation phase. The program exists in secondary memory, and the OS prepares its PCB before loading it into main memory.
- **Ready**: The process has been loaded into main memory and is prepared to run. It is waiting in the ready queue until the CPU becomes available.
- **Running**: The CPU is currently executing the instructions of the process. Only one process can be in this state at a time on a single-processor system.
- **Blocked/Waiting**: The process cannot continue because it is waiting for an event such as I/O completion, user input, or access to a locked resource. Once the event is complete, it moves back to the ready state.
- **Terminated**: The process has completed its execution or has been stopped. The OS deletes its PCB and frees all resources allocated to it.
- **Suspend Ready**: A ready process that has been swapped out of main memory and placed in secondary storage due to memory shortage. When brought back to main memory, it returns to the ready state.
- **CPU-Bound / I/O-Bound**: A CPU-bound process spends most of its time performing computations, while an I/O-bound process spends more time waiting for input/output operations.

4.5 Movement of a Process From One State to Another

A process transitions between different states depending on its progress and the availability of system resources. These movements help the operating system manage tasks efficiently.

- **New → Ready:** A process is created, resources are allocated, and it is loaded into main memory.
- **Ready → Running:** The scheduler selects a ready process and assigns the CPU to it.
- **Running → Blocked (Waiting):** The process must wait for an event or resource (e.g., I/O, user input, system call).
- **Blocked → Ready:** The event completes or the needed resource becomes available, so the process is ready to run again.
- **Running → Ready:** The OS preempts the running process—often because a higher-priority process becomes ready.
- **Running → Terminated:** The process completes its execution or is forcefully stopped.
- **Blocked → Terminated:** The process waiting for an event is aborted or killed by the OS or another process.

(General Rule): A process may move between **ready**, **running**, and **blocked** many times,but **new** and **terminated** happen only once in its lifetime.

5. Types of multiprogramming

We have many processes ready to run. There are two types of multiprogramming:

- **Preemption:** Process is forcefully removed from CPU. Pre-emption is also called time sharing or multitasking.

- **Non-Preemption:** Processes are not removed until they complete the execution. Once control is given to the CPU for a process execution, till the CPU releases the control by itself, control cannot be taken back forcibly from the CPU.

***Remark**: The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming, e.g., if the degree of programming = 100, this means 100 processes can reside in the ready state at maximum.*

6. Operation on The Process

- **Creation:** A process is created and placed in the ready queue (main memory), where it waits to be executed.
- **Scheduling:** The operating system selects one process from the ready queue to run next. This selection step is called scheduling.
- **Execution:** The CPU starts running the chosen process. If the process needs to wait for an event or resource, it becomes blocked, and the CPU switches to another process.
- **Termination:** Once the process completes its task, the OS terminates it and clears its context.
- **Blocking:** When a process waits for an event or resource (like I/O), it enters the blocked state and cannot continue until that event is completed.
- **Context Switching:** When the OS switches from one process to another, it saves the current process's context and loads the next one. This switching is called context switching.
- **Inter-Process Communication (IPC):** Processes often need to exchange data or coordinate. The OS supports IPC using mechanisms like shared memory, message passing, and synchronization tools.

These states—new, ready, running, waiting, and terminated—represent different stages in a process's life cycle. By transitioning through these states, the operating system ensures that processes are executed smoothly, resources are allocated effectively, and the overall performance of the computer is optimized.

6.1 Process Control Block in OS

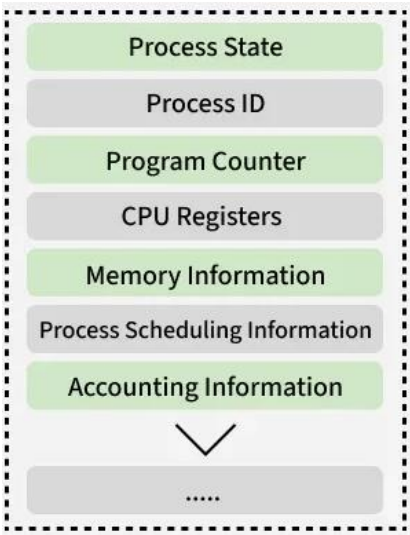
A Process Control Block (PCB) is a data structure used by the operating system to keep track of process information and manage execution. It helps the OS monitor and control process execution

- Each process is given a unique Process ID (PID) for identification.
- The PCB stores details such as process state, program counter, stack pointer, open files, and scheduling info.
- During a state transition, the OS updates the PCB with the latest execution data.
- It also includes register values, CPU quantum, and process priority.
- The Process Table is an array of PCBs that maintains information for all active processes.

The Process Control Block (PCB) is stored as a par of OS so that normal users can't access. This is because it holds important information about the process. Some operating systems place the PCB at the start of the kernel stack for the process, as this is a safe and secure spot.

6.2 Terminologies used in Process Control Block

- **Process state:** Stores whether the process is running, waiting, ready, or terminated
- **Process number Or PID:** Every process is assigned a unique id known as process ID or PID.
- **Program counter:** Program Counter stores the address of the next instruction that is to be executed for the process.
- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.



6.3 Applications of PCB

- Helps the operating system schedule processes and allocate CPU resources efficiently.
- Enables efficient resource utilization and sharing by providing detailed resource-usage information.
- Supports context switching through stored CPU registers and stack pointer information.
- Assists in process synchronization by keeping track of waiting states and required resources.
- Tracks process states and resource usage to determine which process should be executed next.

7. Scheduling queues:

To manage the processes during their stay, the OS maintains several queues. We can cite among others:

- Queue of ready processes: This queue contains all the processes waiting for the processor.
- Device Queue: To regulate allocation requests from different devices, one can imagine a queue for each device. When a process requests an I/O operation, it is put in the relevant queue.

7.1. The scheduler:

The scheduler is a program of the OS which takes care of choosing, according to a given scheduling policy, a process among the processes ready to allocate the processor to it. That is, the scheduler is a module of the operating system which assigns control of the CPU in turn to the various processes in competition according to a policy defined in advance by the designers of the system.

7.2. Context switching:

Switching the processor to another process requires saving the state of the old process and loading the state saved by the new process. This task is known as context switching.

7.3. Scheduling criteria:

Different processor scheduling algorithms have different properties and may favor one class of process over another. To choose which algorithm to use in a particular situation, we need to consider the properties of the various algorithms.

Several criteria have been proposed to compare and evaluate the performance of processor scheduling algorithms. The criteria most often used are:

- **CPU usage**: A good scheduling algorithm will be one that keeps the processor as busy as possible.
- **Processing capacity**: It is the quantity of completed processes per unit of time.
- **Waiting time**: This is the time spent waiting in the ready process queue.
- **Response time**: This is the time spent in the queue of ready processes before the first response.
- **Turnaround time**: also known as residence time, is the time taken by a process from its arrival to its termination.

• **Arrival Time:** Time at which the process arrives in the ready queue.
• **Completion Time:** Time at which process completes its execution.
• **Burst Time:** Time required by a process for CPU execution.
• **Turnaround Time (T.A.T):** Time Difference between completion time and arrival time.
Turn Around Time = Completion Time – Arrival Time
• **Waiting Time (W.T):** Time Difference between turnaround time and burst time.
Waiting Time = Turnaround Time – Burst Time

7.4. Scheduling algorithms

The CPU scheduler is used to decide which process (in the ready processes queue) to assign CPU control to. Scheduling strategies can be grouped into two categories: without CPU requisition (non-preemptive strategy) or with CPU requisition (preemptive strategy). Objectives of Process Scheduling Algorithm:

- Utilization of CPU at maximum level. Keep CPU as busy as possible.
- Allocation of CPU should be fair.
- Throughput should be Maximum. i.e. Number of processes that complete their execution per time unit should be maximized.
- Minimum turnaround time, i.e. time taken by a process to finish execution should be the least.
- There should be a minimum waiting time and the process should not starve in the ready queue.
- Minimum response time. It means that the time when a process produces the first response should be as less as possible.

7.5. Scheduling modes

Two classes of scheduler:

A. ***Non-preemptive:*** (without requisition) Selects a process, then lets it run until it blocks (either on an I/O or waiting for another process) or voluntarily frees the processor. Even if it runs for hours, it won't be forcibly suspended. No scheduling decisions are made during clock interrupts.
B. ***Preemptive:*** (with requisition) selects a process and lets it run for a specified time. If the process is still running at the end of this delay, it is suspended and the scheduler selects another process to execute.

8.1 Scheduling without CPU requisition (without preemption)

In this category, a process retains control of the CPU until it crashes or terminates. This approach corresponds to the needs of batch jobs (batch systems). There are several algorithms in this category:

8.1.1 First Come First Served (FCFS: First come First Served) (also called FIFO)

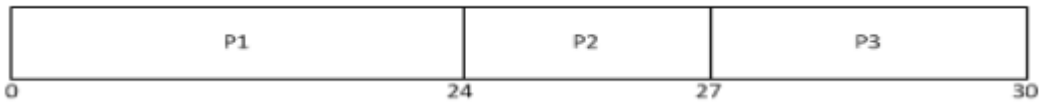
The simplest processor scheduling algorithm is the First Come First Served (FCFS) algorithm. With this algorithm, the processor is allocated to the first process that requests it. FCFS policy implementation is easily managed with a FIFO queue. When a process enters the ready process queue, its PCB is chained to the tail of the queue. When the processor becomes free, it is allocated to the processor at the top of the queue.

Example :

•We consider all the following processes:

Process	CPU Burst
P1	24
P2	3
P3	3

It is assumed that the processes arrive at time 0 and admitted in the order: P1 , P2 , P3 •The Gantt chart for the FCFS scheduling of this set is:



•Waiting time for P1 = 0; P2=24; P3 = 27
•Average waiting time: (0 + 24 + 27)/3 = 17
If the processes had arrived in the order P2, P3 and P1, the results would be different:

The average waiting time would be: (0+3+6)/3=3 units.
Thus the average waiting time with an FCFS policy is generally not minimal and can vary substantially if the execution times of the processes vary a lot.

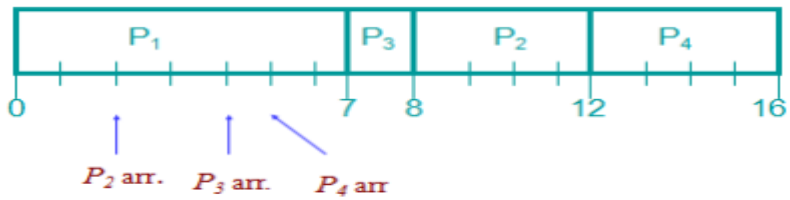
Critique of the method:
The FCFS method tends to penalize short jobs: The FCFS algorithm does not perform requisitions. That is to say that once the processor has been allocated to a process, this one keeps it until it releases it, either by terminating, or after having requested an I/O.
The FCFS algorithm is particularly inconvenient for time-sharing systems, where it is important for the user to obtain the processor at regular intervals. It may seem disastrous to allow a process to hold onto the processor for an extended period of time.

8.1.2 Shortest first: Shortest Job First (SJF)

This algorithm (in English Shortest Job First: SJF) assigns the processor to the process with the shortest execution time. If several processes have the same duration, a FIFO policy will then be used to decide between them.
Example :

Process	Arrival Time	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

The Gantt chart:



Average waiting time = (0+(8-2)+(7-4)+(12-5))/4
(0 + 6 + 3 + 7)/4 = 4
Critique of the method:
The SJF algorithm has been proven to be time-optimal in the sense that it achieves the shortest waiting time for a given set of processes. However, this algorithm is difficult to implement for a simple reason: How can we know the execution time of a process in advance?
Solution: Prediction technique.

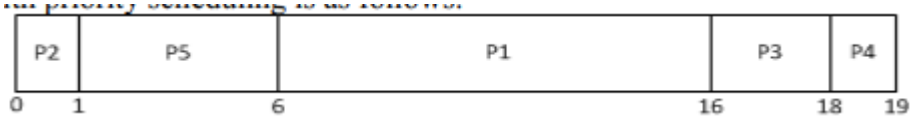
8.1.3 Scheduling with priority:

This algorithm associates each process with a priority, and the processor will be assigned to the process with the highest priority. This priority varies between systems and can range from 0 to 127.
Priorities can be defined according to several parameters: the type of process, time limits, memory limits, etc.

Example: We have 5 processes with different priorities, as shown in this table:

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

The Gantt chart with priority scheduling is as follows:



The average waiting time is = (0+1+6+16+18)/5=8.2 time units.

Critique of the method:

A non-preemptive priority scheduling algorithm places the new process at the head of the ready process queue:

•A scheduling algorithm with priority poses a major problem –Infinite blocking or starvation (starvation) –Can leave low priority processes waiting indefinitely

•Solution:

– Aging: A technique of gradually increasing the priority of processes that have been waiting in the system for a long time.

8.2 Scheduling with CPU requisition (with preemption)

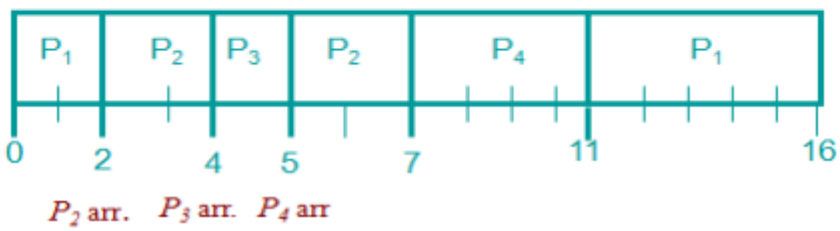
In this category, the scheduler can remove the CPU from a process before the latter hangs or terminates in order to allocate it to another process. At each quantum (unit of time) the scheduler chooses from the list of ready processes a process to which the CPU will be allocated. This approach corresponds to interactive systems.

8.2.1 The shortest remaining time (SRTF: Shortest Remaining Time First)

It is the preemptive version of the SJF algorithm. At the start of each quantum, the CPU is allocated to the process that has the smallest remaining execution time.

Example of SJF with preemption (SRTF)

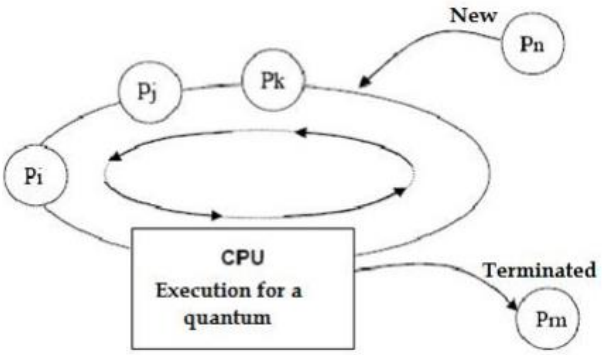
Process	Arrival Time	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4



Average waiting time = (9 + 1 + 0 +2)/4 = 3; P1 waits from 2 to 11, P2 from 4 to 5, P4 from 5 to 7

8.2.2 Round Robin Algorithm:

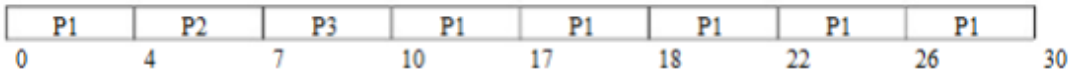
CPU control is assigned to each process for a slice of time Q, called a quantum, in turn. When a process runs during a quantum, the context of the latter is saved and the CPU is allocated to the next process in the list of ready processes (see the figure below). In practice, the quantum ranges between 10 and 100 ms.



Example: We have 3 processes P1, P2 and P3 having execution times of 24, 3 and 3 ms respectively.

Process	CPU Cycle (ms)
P1	24
P2	3
P3	3

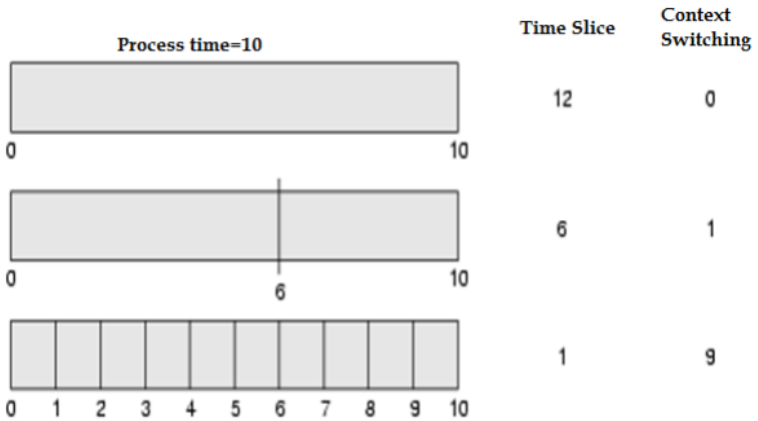
Using a Round Robin algorithm, with a quantum of 4 ms, we obtain the following Gantt chart:



The average waiting time is: (6+4+7)/3 = 17/3 = 5.66 ms

Critique of the method

The performance of this policy depends on the value of the quantum. A too large quantum increases the response time, the Round Robin policy would be similar to that of the FCFS. A quantum that is too small multiplies the switchings of context, which, from a certain frequency, can no longer be neglected. **Example**: We have a process P whose execution time is 10 ms. Let's calculate the number of context switches needed for a quantum equal respectively to: 12, 6 and 1.

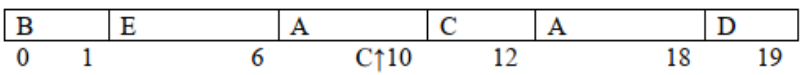


8.2.3 Priority with preemption

Example:

It is assumed in this example th18at the smallest value corresponds to the highest priority.

PID	Arrival time	CPU Burst	Priority
A	0	10	3
B	0	1	1
C	10	2	2
D	0	1	4
E	0	5	2



Average waiting time= (8+0+0+18+1)/5=5,4

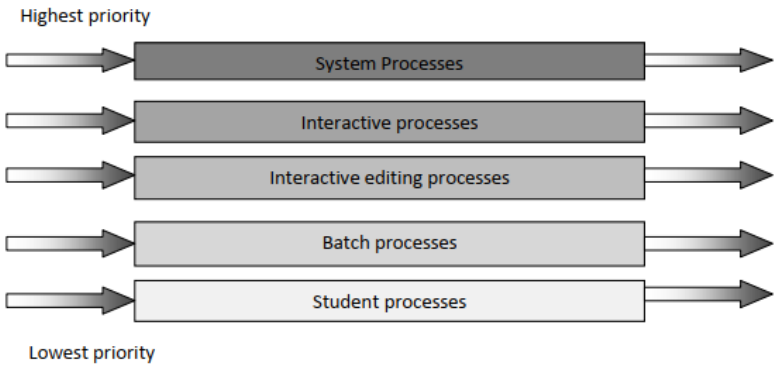
Preemptive vs. Non-Preemptive Scheduling

Preemptive Scheduling	Non-Preemptive Scheduling
In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state
Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up
If a process having high priority frequently arrives in the ready queue, a low priority process may starve	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve
It has overheads of scheduling the processes	It does not have overheads
Average process response time is less	Average process response time is high
Decisions are made by the scheduler and are based on priority and time slice allocation	Decisions are made by the process itself and the OS just follows the process's instructions
More as a process might be preempted when it was accessing a shared resource.	Less as a process is never preempted.
Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First

9. Scheduling with multilevel queues:

Another class of scheduling algorithms has been developed for situations where one can easily classify processes into different groups. For example, it would be interesting to make a distinction between foreground (interactive) processes and background (batch) processes. Indeed, these two types of processes have different needs in terms of response time and therefore may need to be scheduled differently. In addition, foreground processes can take priority over background processes.

Thus, a scheduling algorithm with multilevel queues splits the queue of ready processes into several separate queues. The following figure gives an example of the breakdown of these queues (e.g.: main server of a university):



Processes are permanently assigned to a queue usually based on certain process properties, such as: process type, memory size, priority, etc. Each queue has its own scheduling algorithm. For example, the queue of system processes is scheduled according to the highest priority algorithm, those of interactive processes are managed according to the Round Robin algorithm and the batch process queue is managed according to the

FCFS algorithm. On the other hand, there must be scheduling between the queues themselves. Consider, for example, the following set of multilevel queues:

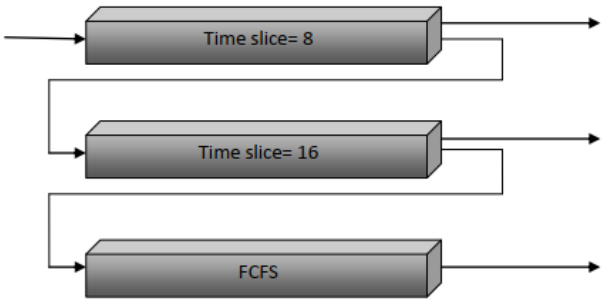
1. Systems processes
2. Interactive processes
3. Batch process
4. User processes

Each queue has absolute priority over lower level queues. For example, no process in the batch process queue will be able to run unless the system and interactive process queues are all empty. Also, if an interactive process enters the system while a batch process is running, the batch process must be terminated. Another way to do this would be to assign time slices to queues. Each queue obtains a certain part of the processor time, which must be scheduled between the different processes that compose it. For example, you can allocate 80% of the processor time to the foreground process queue and 20% to the background process queue.

10. Scheduling with multilevel queues and feedback:

Normally, in an algorithm with multilevel queues, processes are permanently assigned to a queue as soon as they enter the system. Processes do not move between queues. This organization has the advantage of a low overhead due to scheduling, but it lacks flexibility. Thus, scheduling with multilevel feedback queues allows processes to move between queues. The idea comes down to separating the processes according to the evolution of their characteristics in the system.

Example: A system has 3 multilevel queues: Queue 0, Queue 1 and Queue 2. Queue 0 has the highest priority. Queues 0 and 1 are managed according to the Round Robin policy. Queue 2 is managed using the FCFS technique.



A process entering the system will be placed in queue 0. The process is given a time slice of 8 ms. If it does not finish, it is moved to queue 1. If queue 0 is empty, a 16 ms time slice is given to the header process of queue 1. If it does not finish, it is interrupted and put in queue 2. Processes in queue 2 are executed only when queues 0 and 1 are empty.

References

[1]:https://www.geeksforgeeks.org/operating-systems/introduction-of-process-management/(View 16/02/2026)

SILBERSCHATZ, A. and PB GALVIN, Operating System Concepts. 8th Edition, Addison Wesley.2012.

Dr. D. Boukhlouf. Course materials. Module: Operating Systems I. Chapter 3: Process Scheduling. Academic year 2023/2024