Process Management

Jayesh Deep Dubey
Technical Officer
C-DAC Patna

What is a Process?

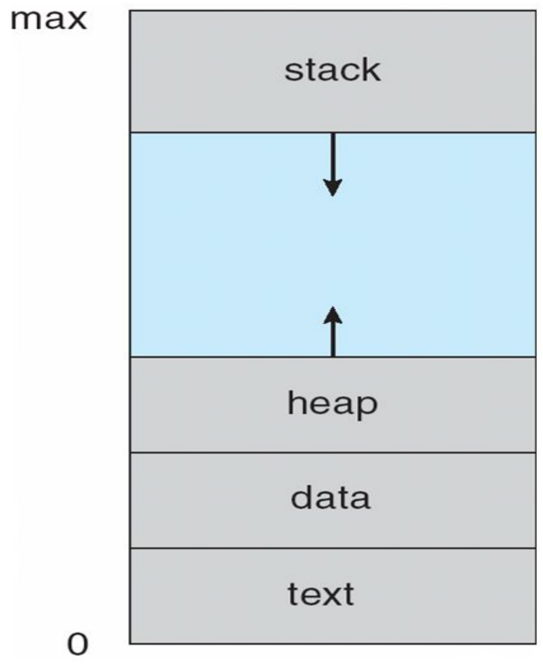
- A process is an instance of a program that is currently executing on a computer system.
  - Process – a program in execution
- Program is passive entity stored on disk (executable file), process is active
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

Parts of a Process

- The program code, also called **text** section
- Current activity including program counter, processor registers
- **Stack** containing temporary data
  - Function parameters, return addresses, local variables
- **Data** section containing global variables
- **Heap** containing memory dynamically allocated during run time

Process in memory

-

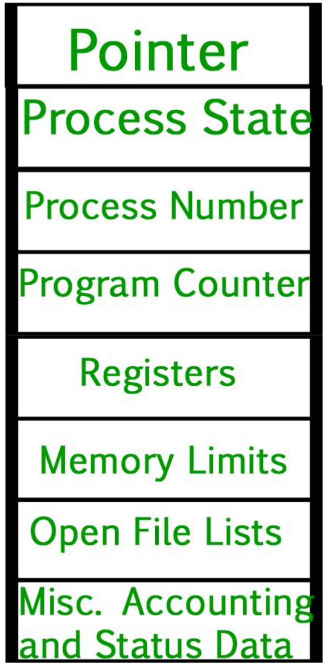


Preemptive and Non-Preemptive Process

- Preemptive and non-preemptive processes refer to the way processes are managed by an operating system.
- In a preemptive system, the operating system can interrupt a running process to allow another process to run
- In a non-preemptive system, a process runs until it voluntarily relinquishes control of the CPU.

Process Control Block

- The process control block (PCB) is a data structure used by an operating system to manage each process
- It contains all the information about a process that is necessary for the operating system to manage the process



Process Control Block

Process Control Block

- **Process Identification (ID) Section**: This section contains the unique identifier (PID) for the process.
- **Process State Section:** It stores the respective state of the process such as whether it is running, blocked, or ready to run
- **Program counter Section:** It stores the counter which contains the address of the next instruction that is to be executed for the process.
- **Register Section:** It contains information such as the values of CPU registers which includes: accumulator, base registers and general purpose registers.

Process Control Block

- **Memory Management Information Section**: This section contains information about the memory resources allocated to the process, such as the starting address, size.
- **Open files list Section**: This information includes the list of files opened for a process.
- **Pointer Section:** It contains pointers to various resources that the process may need.The purpose of the Pointer Section is to provide the operating system with quick access to the resources associated with a process, such as memory, files, or I/O devices.

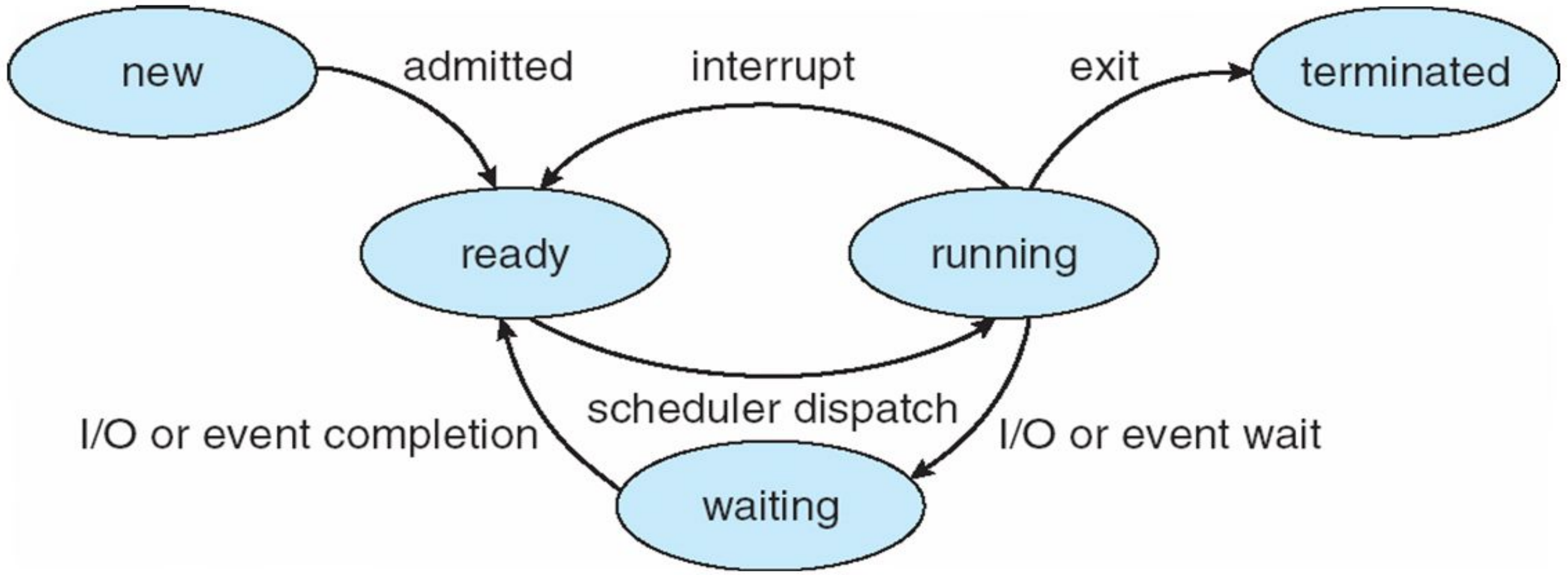
Process Control Block

- **Accounting Information Section**: This section contains information about the resources used by the process, such as CPU time used, memory usage, and I/O operations performed.
- **Input/Output (I/O) Status Information Section**: This section contains information about the I/O devices used by the process, such as the files and devices the process has opened, the status of I/O operations, and the I/O requests made by the process

Process Life Cycle

- The process life cycle consists of several stages: new, ready, running, blocked, and terminated.
- As a process executes, it changes state
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Process Life Cycle



Introduction to Schedulers

- Schedulers are an essential part of process management. They determine which process should run next and for how long.
- They are responsible for managing and coordinating the execution of multiple processes in a way that optimizes system resources and ensures fair and efficient execution of tasks.
- There are three types of schedulers - short-term, medium-term, and long-term

Two Types of Processes

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Short Term Scheduler

- The short-term scheduler, also known as the CPU scheduler, determines which process should run next in the ready queue.
- It runs frequently, typically every few milliseconds, to ensure that the CPU is used efficiently

Long Term Scheduler

- The long-term scheduler, also known as the job scheduler, determines which processes should be admitted into the system
- Long-term scheduler is responsible for selecting processes from the pool of incoming processes and adding them to the pool of processes that are ready to be executed
- Long-term scheduler is invoked infrequently (seconds, minutes) (may be slow)
- Long-term scheduler strives for good process mix(I/O bound and CPU bound)

Medium Term Scheduler

- Medium-term scheduler (also known as swapping scheduler) is responsible for managing the process's memory allocation. It determines which processes should be swapped out of main memory and onto secondary storage.
- It is responsible for managing memory and disk space.
- When the number of processes in the system exceeds the available physical memory, the medium-term scheduler selects processes that have been idle for a long time and moves them to the disk. This helps to free up memory space and improve system performance.

Process Scheduling

- Process scheduling is an essential operating system function that determines the order and timing of executing different processes or threads
- The primary purpose of process scheduling is to allocate the system's resources, such as CPU time, memory, and input/output devices, to the processes in a fair and efficient manner.

Benefits of Process Scheduling

Here are some of the uses and benefits of process scheduling:

- **Fair resource allocation**: Process scheduling ensures that each process gets a fair share of the available resources, such as CPU time and memory. This ensures that no process monopolizes the system resources, which can lead to poor system performance and stability.
- **Optimal system performance**: By scheduling processes effectively, the operating system can maximize the utilization of the CPU and other resources, which can result in improved system performance.

Benefits of Process Scheduling

- **Prioritization of processes**: Process scheduling allows the operating system to prioritize critical processes, such as those involved in system maintenance or security, over other less important tasks.
- **Multitasking**: Process scheduling enables the operating system to perform multiple tasks simultaneously, even on a single-core processor. This is achieved by rapidly switching between different processes, giving the illusion of parallelism.
- **Time-sharing**: Process scheduling allows the operating system to time-share the CPU among multiple processes. This means that each process gets a small time slice of the CPU's attention, which is typically a few milliseconds long. This provides the illusion of each process running continuously, even though they are sharing the CPU with other processes.

Process Scheduling Algorithms

- Process scheduling algorithms are the methods and techniques used by the operating system to determine which process should be executed next.
- The scheduling algorithm is responsible for selecting the next process from the pool of processes waiting to be executed and allocating system resources, such as the CPU, to that process.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**

Process Scheduling Algorithms

- First-Come, First-Serve (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling
- Round Robin (RR)
- Multilevel Queue (MLQ)

Some Important Terms w.r.t Process Scheduling

In process scheduling, different types of time are used to measure and manage the execution of processes:

- **Arrival time**: This is the time when a process arrives in the ready queue and requests the CPU for execution.
- **Burst time**: This is the time required by a process to complete its execution once it gets the CPU
- **Completion time**: This is the time when a process completes its execution, i.e., when it finishes its last instruction

Some Important Terms w.r.t Process Scheduling

Different types of time are used to measure and manage the execution of processes:

- **Waiting time**: This is the total amount of time a process spends in the ready queue waiting for the CPU to become available.
- **Turnaround time**: This is the total amount of time a process takes from when it arrives in the ready queue to when it completes its execution.

Process Scheduling Algorithms

- First Come First Serve (FCFS)
    - First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first.
    - It is implemented by using the FIFO queue
    - Tasks are always executed on a First-come, First-serve concept.
    - FCFS is easy to implement and use.
    - This algorithm is not much efficient in performance, and the wait time is quite high

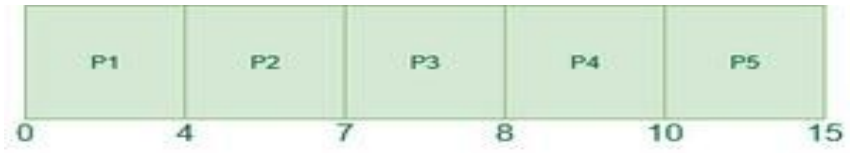
An example (FCFS)

Example-1: Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Processes	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	1
P4	3	2
P5	4	5

An example (FCFS)

- **Waiting Time**
  - P1 = 0 – 0 = 0
  - P2 = 4 – 1 = 3
  - P3 = 7 – 2 = 5
  - P4 = 8 – 3 = 5
  - P5 = 10 – 4 = 6



- **Average Waiting time**

= (0 + 3 + 5 + 5+ 6)/ 5 = 19 / 5 = 3.8

An example (FCFS)

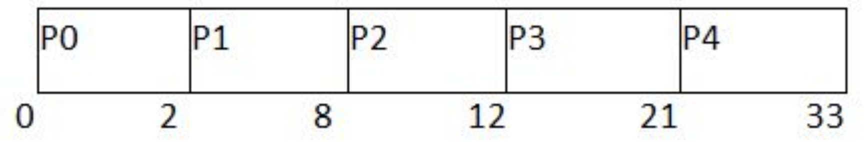
- **Turnaround Time**
  - P1 = 4 – 0 = 4
  - P2 = 7 – 1 = 6
  - P3 = 8 – 2 = 6
  - P4 = 10 – 3 = 7
  - P5 = 15 – 4 = 11

Question (FCFS)

Consider the following table of arrival time and burst time for five processes P0, P1, P2, P3 and P4.

Processes	Arrival Time	Burst Time
P0	0	2
P1	1	6
P2	2	4
P3	3	9
P4	6	12

Solution



.

Completion Time	Turnaround Time	Waiting Time
2	2	0
8	7	1
12	10	6
21	18	9
33	29	17

Process Scheduling Algorithms

- **Shortest Job First(SJF)**
  - SJF selects the waiting process with the smallest execution time to execute next.
  - Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
  - SJF can be used in specialized environments where accurate estimates of running time are available

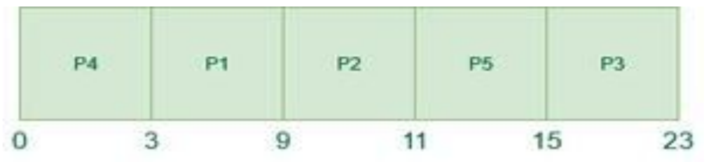
An example (SJF)

Example-1: Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Processes	Arrival Time	Burst Time
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

An example (SJF)

- **Waiting Time**
  - P4 = 0 – 0 = 0
  - P1 = 3 – 2 = 1
  - P2 = 9 – 5 = 4
  - P5 = 11 – 4 = 7
  - P3 = 15 – 1 = 14



- Average Waiting Time

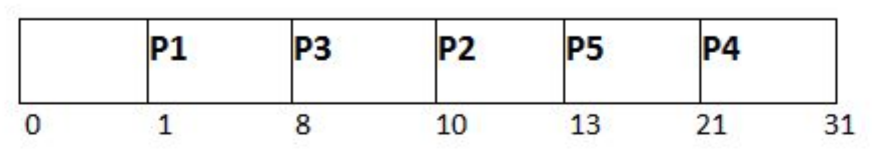
  = 0 + 1 + 4 + 7 + 14/5 = 26/5 = 5.2

Question (SJF)

Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Processes	Arrival Time	Burst Time
P1	1	7
P2	3	3
P3	6	2
P4	7	10
P5	9	8

Solution



.

Completion Time	Turnaround Time	Waiting Time
8	7	0
13	10	7
10	4	2
31	24	14
21	12	4

Process Scheduling Algorithms

- **Shortest Remaining Time First (SRTF)**
  - It is a preemptive version of the shortest job first (SJF) algorithm
  - In SRTF, scheduler selects the process with the shortest remaining burst time to execute next.
  - SRTF algorithm can lead to starvation for long-running processes because short processes may always be selected for execution.
  - To mitigate this, some operating systems use aging, where the priority of a process increases the longer it waits in the ready queue.

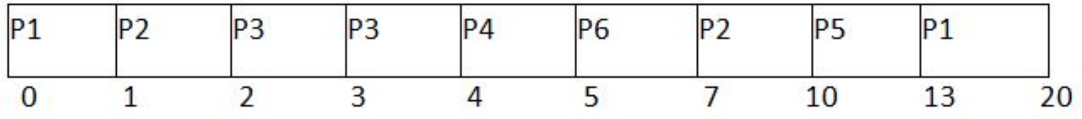
An example (SRTF)

Example-1: Consider the following table of arrival time and burst time for six processes P1, P2, P3, P4 and P5,P6

Processes	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	2
P4	3	1
P5	4	3
P6	5	2

An example (SRTF)

- **Waiting Time**
  - P1 = 12
  - P2 = 5
  - P3 = 0
  - P4 = 1
  - P5 = 6
  - P6 = 0



- **Average Waiting Time**

= (12 + 5 + 0 + 1 + 6 + 0)/6 = 24/6 = 4.

An example (SRTF)

- **Turnaround Time**
    - P1 = 20
    - P2 = 9
    - P3 = 2
    - P4 = 2
    - P5 = 9
    - P6 = 2

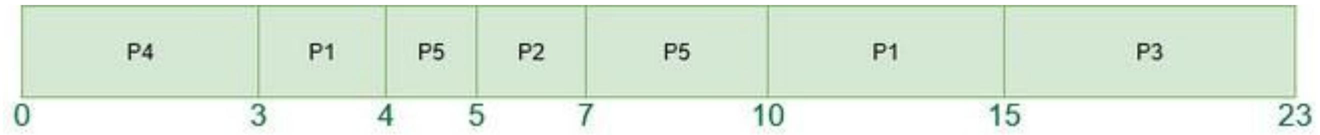
Question (SRTF)

Example-1: Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Processes	Burst Time	Arrival Time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Solution

.



Processes	Waiting Time	Turnaround Time
P1	7	13
P2	0	2
P3	14	22
P4	0	3
P5	2	6

Process Scheduling Algorithms

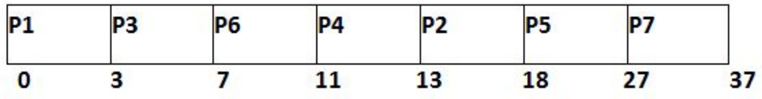
- **Priority Scheduling (Non Preemptive)**
    - In Priority scheduling, there is a priority number assigned to each process.
    - The Process with the higher priority among the available processes is given the CPU
    - In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority

An example (Priority Scheduling)

Example-1: Consider the following table of arrival time and burst time for seven processes P1, P2, P3, P4, P5, P6, P7.(Lower no. means higher priority)

Process ID	**Priority**	**Arrival Time**	**Burst Time**
P1	2	0	3
P2	6	2	5
P3	3	1	4
P4	5	4	2
P5	7	6	9
P6	4	5	4
P7	10	7	10

Example(Priority Scheduling)



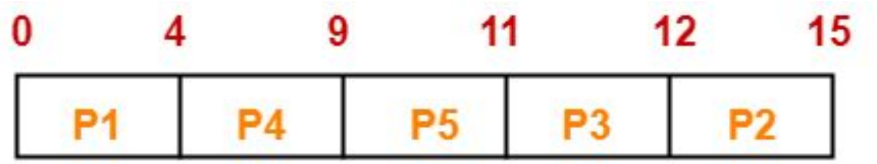
Process ID	Completion Time	Turnaround Time	Waiting Time
P1	3	3	0
P2	18	16	11
P3	7	6	2
P4	13	9	7
P5	27	21	12
P6	11	6	2
P7	37	30	18

Question(Priority Scheduling Non Preemptive)

Consider the set of 5 processes whose arrival time and burst time are given below(higher no. means higher priority)

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

Solution



.

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 0 = 4	4 – 4 = 0
P2	15	15 – 1 = 14	14 – 3 = 11
P3	12	12 – 2 = 10	10 – 1 = 9
P4	9	9 – 3 = 6	6 – 5 = 1
P5	11	11 – 4 = 7	7 – 2 = 5

Process Scheduling Algorithms

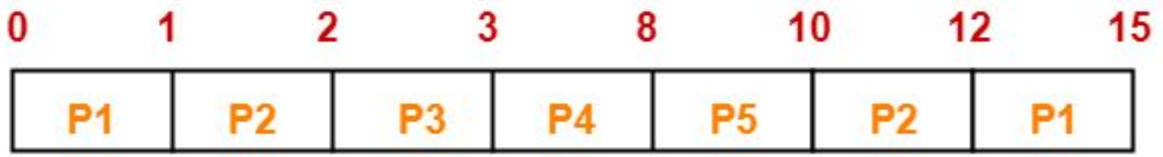
- **Priority Scheduling (Preemptive)**
    - This is a preemptive version of Priority Scheduling.
    - If a higher priority process comes in the ready queue and lower priority process is running on the CPU then it will be preempted and higher priority process will be assigned the CPU
    - In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority

An example (Priority Scheduling Preemptive)

Example-1: Consider the following table of arrival time and burst time for seven processes P1, P2, P3, P4, P5, P6, P7. (Higher No. means higher priority)

Process ID	**Priority**	**Arrival Time**	**Burst Time**
P1	2	0	4
P2	3	1	3
P3	4	2	1
P4	5	3	5
P5	5	4	2

Example(Priority Scheduling Preemptive)



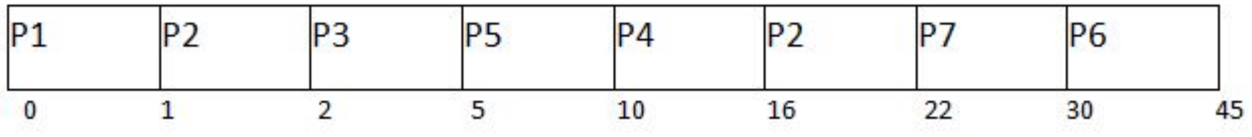
Process ID	Completion Time	Turnaround Time	Waiting Time
P1	15	15	11
P2	12	11	8
P3	3	1	0
P4	8	5	0
P5	10	6	4

Question(Priority Scheduling Preemptive)

Consider the set of 7 processes whose arrival time and burst time and priority are given below (lower No => Higher Priority)

Process Id	**Arrival time**	**Burst time**	**Priority**
P1	0	1	2
P2	1	7	6
P3	2	3	3
P4	3	6	5
P5	4	5	4
P6	5	15	10
P7	15	8	9

Solution



Process	Waiting Time	Turnaround Time
P1	0	1
P2	14	21
P3	0	3
P4	7	13
P5	1	6
P6	25	40
P7	16	24

Process Scheduling Algorithms

- **Round Robin Scheduling**
  - Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems.
  - In this algorithm, every process gets executed in a cyclic way.
  - Each process present in the ready queue is assigned the CPU for a certain time slice (time quantum), if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution

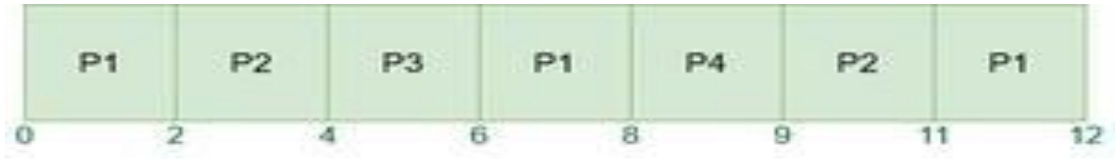
An example (RR Scheduling)

Example-1: Consider the following table of arrival time and burst time for four processes P1, P2, P3, and P4 and given Time Quantum = 2.

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

An example (RR Scheduling)

.



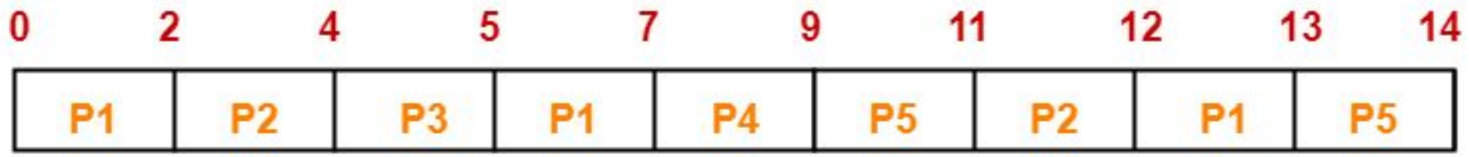
Process	Waiting Time	Turnaround Time
P1	7	12
P2	6	10
P3	2	4
P4	4	5

Question (RR Scheduling)

Consider the following table of arrival time and burst time for five processes and given Time Quantum = 2

Process	Burst Time	Arrival Time
P1	5 ms	0 ms
P2	3 ms	1 ms
P3	1 ms	2 ms
P4	2 ms	3 ms
P5	3 ms	4 ms

Solution



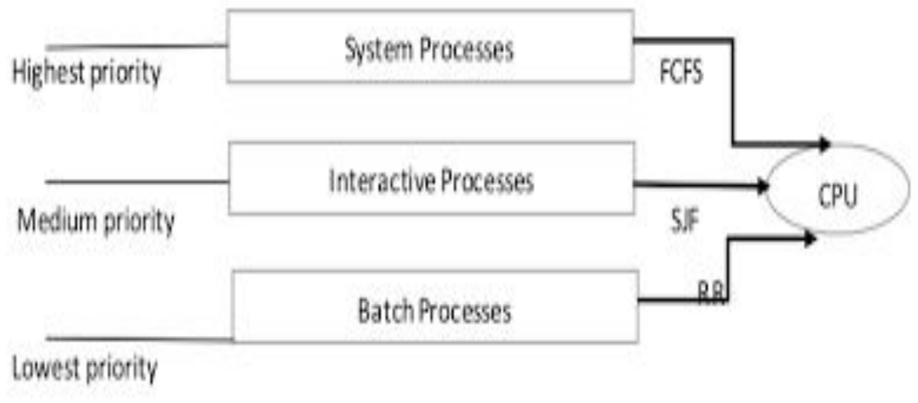
Process	Turnaround Time	Waiting Time
P1	13	8
P2	11	8
P3	3	2
P4	6	4
P5	10	7

Process Scheduling Algorithms

- **Multilevel Queue Scheduling**
  - Multilevel queue scheduling is a type of process scheduling algorithm in which processes are divided into separate queues based on different priority levels or types of processes
  - Each queue can have its own scheduling algorithm and priority level, which allows the system to manage processes more efficiently based on their characteristics and requirements
  - Processes can be moved between queues based on their priority levels, allowing the system to adapt to changing requirements and optimize resource utilization.

Process Scheduling Algorithms

- **Multilevel Queue Scheduling**



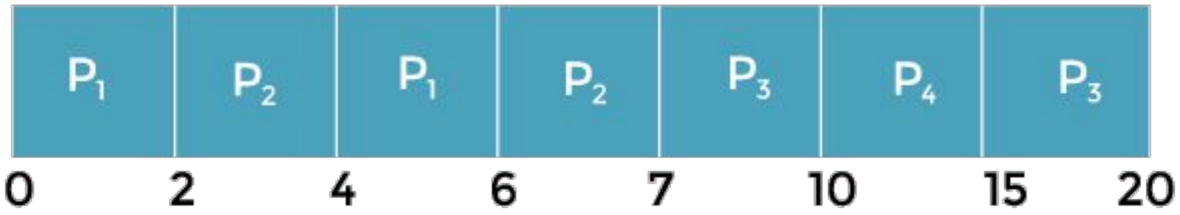
An Example (**Multilevel Queue Scheduling**)

Consider the following table of arrival time and burst time for four processes. There are two queues. Queue 1 has higher priority and uses RR (t = 2), Queue 2 has lower priority and uses FCFS.

Process	Arrival Time	Burst Time	Queue
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

An Example (**Multilevel Queue Scheduling**)

.



Fork System Call

- The fork system call is a fundamental feature of operating systems that creates a new process by duplicating the calling process
- When a fork system call is made, a new process is created, which is an exact copy of the parent process
- The new process created by the fork system call is called the child process, and the original process that called fork is called the parent process. The child process has its own process ID (PID) and runs independently of the parent process
- After the fork system call, both the parent and child processes continue to execute concurrently, with the child process receiving a return value of 0 from the fork call, and the parent process receiving the PID of the child process.

Fork System Call

- **Example 1**

```
#include <stdio.h>
#include <sys/types.h>;
#include <unistd.h>;
int main()
{
  fork();
  printf("Hello world!\n");
  return 0;
}
```

Output:

```
Hello World
Hello World
```

Fork System Call

- **Example 2**

```
#include <stdio.h>
#include <sys/types.h>;
#include <unistd.h>;
int main()
{
  fork();
  fork();
  printf("Hello world!\n");
  return 0;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
```

Fork System Call

- **For n number of system calls:**
  - Total number child processes

    $2^n - 1$

  - Total number of processes

    2^n

Fork System Call

- **Example 3**

```
void Example3()
{
        if (fork() == 0)
                printf("Child Process");
        else
                printf("Parent Process");
}
int main()
{
        Example3();
        return 0;
}
```

Output:

```
Child Process
Parent Process

(or)

Parent Process
Child Process
```

Fork System Call

- **Example 4**

```
int main()
{
          if (fork() == 0)
          {
               if(fork() > 0)
               {
                  printf("Hello");
               }
          }
}
```

exec System Call

- The exec system call is used to replace the current process image with a new process image.
- When the exec system call is invoked, the operating system replaces the current process with a new process, loading the new executable file into the current process space.
- The exec system call can be used to start a new process with a different program, arguments, and environment variables, but with the same process ID, file descriptors, and other process attributes.

exec System Call

- Some examples of how the exec system call can be used include:
    - Starting a new process with a different program
    - Starting a new process with the same program, but with different arguments
    - Replacing the current process with a different program

waitpid System Call

- The waitpid system call is used by a parent process to wait for the termination of a child process.
- When a parent process creates a child process using the fork system call, it may need to wait for the child process to complete before continuing its own execution. The waitpid system call provides a way for the parent process to block and wait for the child process to terminate.
- The waitpid system call takes three arguments: pid, status, and options.

waitpid System Call

- The pid argument specifies the process ID of the child process to wait for. The status argument is used to store information about the termination of the child process, such as the exit status. The options argument can be used to specify additional flags to control the behavior of the waitpid system call.
- When the waitpid system call is called by a parent process, it blocks until one of its child processes terminates. If there are no child processes to wait for, waitpid returns immediately with an error code.
- If there are one or more child processes to wait for, waitpid returns the process ID of the terminated child process

Orphan Process

- An orphan process is a process that has been created by another process (usually a parent process) and is still running, but the parent process has terminated or exited without waiting for the child process to terminate.
- The child process is no longer being monitored by its parent process.
- The orphan process is then adopted by the init process (the first process that is started during the boot process and is responsible for starting and managing other system processes, it has a process ID 1), which becomes its new parent process.
- Orphan processes continue to run until they complete or are terminated by the init process.

Zombie Process

- A zombie process is a process that has completed execution and terminated, but its parent process has not yet called waitpid system call to collect its exit status.
- When a process terminates, it continues to exist in the process table until its parent process retrieves its exit status. If the parent process fails to do this, the terminated process becomes a zombie process.
- Zombie processes do not use any system resources, but they can accumulate in the system if their parent processes do not retrieve their exit status
- Zombie processes can be cleaned up by their parent process by calling the waitpid system call to collect their exit status

Thank You !!