

CH – 5

PROCESS SCHEDULING

What is process scheduling?

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Why is it necessary?

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.
 - For a single processor system, there will never be more than one running process.
 - If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

Scheduling Queues

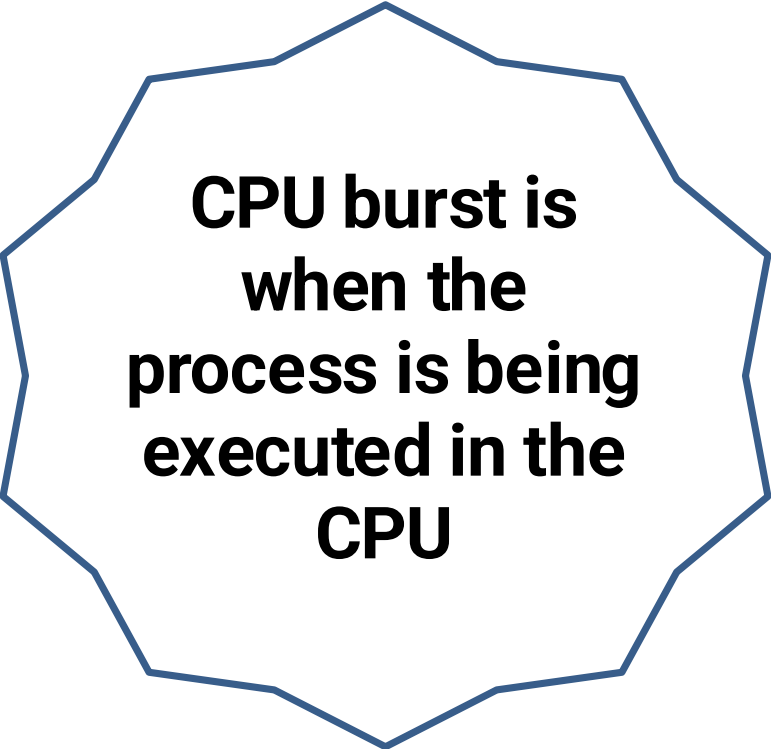
JOB QUEUE: As processes enter the system, they are put into a job queue(secondary memory), which consists of all processes in the system.

READY QUEUE: The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.

CPU and I/O Burst Cycles

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** end with a system request to terminate execution



**CPU burst is
when the
process is being
executed in the
CPU**



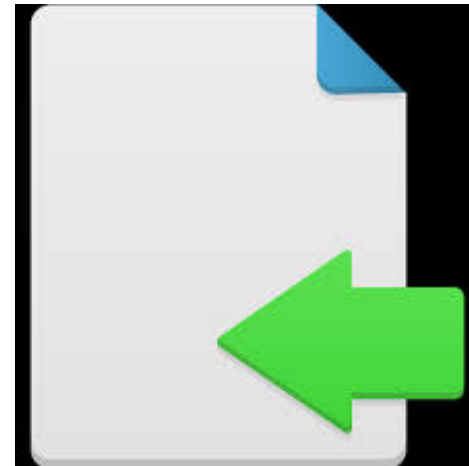
**I/O burst is when
the CPU is
waiting for I/O for
further execution**

Context Switch

- Interrupts cause the operating system to change a CPU from its current task and to run a kernel routine.
- Such operations happen frequently on general-purpose systems.
- **When an interrupt occurs, the system needs to save the current context of the process currently running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it.**
- The context is represented in the PCB (Process Control Block) of the process.

Context Switch

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**.

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 main-memory that are ready to execute and allocates the CPU to that process.

DISPATCHER

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Preemptive & Non-preemptive scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running state** to the **waiting state**.
 2. When a process switches from the **running state** to the **ready state** (for example, when an interrupt occurs).
 3. When a process switches from the **waiting state** to the **ready state** (for example, at completion of I/O).
 4. When a **process terminates**.
- For situation 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.
- However, there is a choice for situations 2 and 3.

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is Non-preemptive or cooperative; otherwise it Preemptive.

CPU Scheduling: Scheduling Criteria

➤ **CPU Utilization**

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

➤ **Throughput**

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time.

➤ **Turnaround Time**

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process.

➤ **Waiting Time**

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

➤ **Response Time**

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Some Important Terminologies

Burst time/execution time/running time(BT) :-

Time required by a process for CPU execution.

Waiting time(WT):-

Time spend by the process , in ready state , waiting for CPU to free.
(process is ready, processor is busy)

Arrival time(AT):-

The time at which process gets ready for execution.

Exit time/terminatng time(ET):-

The instance at which process leaves the system.

Turn around time(TAT):-

Total time spend by the process in the system.

First Come First Serve Scheduling

- In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.
- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.

It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.

A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

First Come First Serve Scheduling

- For every scheduling algorithm, Average waiting time is a crucial parameter to judge its performance.
- AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.
- *Lower the Average Waiting Time, better the scheduling algorithm.*
- $\text{Turn Around Time} = \text{Completion Time} - \text{Arrival Time}$
- $\text{Waiting Time} = \text{Turnaround time} - \text{Burst Time}$

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

Schedulina



First Come First Serve Scheduling

The average waiting time will be 18.75 ms

For the above given processes, first **P1** will be provided with the CPU resources, Hence, waiting time for **P1** will be 0

P1 requires 21 ms for completion, hence waiting time for **P2** will be 21 ms

Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be $(21 + 3) \text{ ms} = 24 \text{ ms}$.

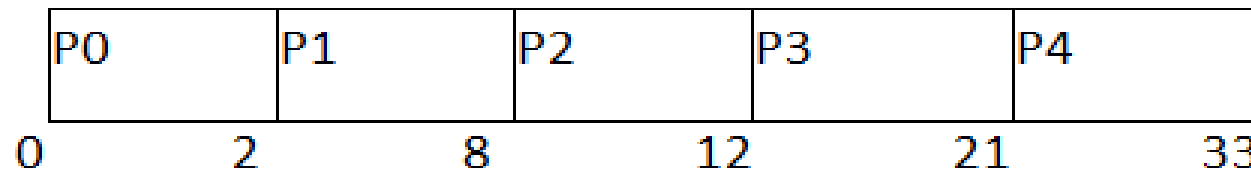
For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

Example:

Let's take an example of The FCFS scheduling algorithm. The processes and their respective Arrival and Burst time are given in the following table.

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



$$TA = FT - AT$$

$$TA \text{ for } P0 = 2 - 0 = 2$$

$$TA \text{ for } P1 = 8 - 1 = 7$$

$$TA \text{ for } P2 = 12 - 2 = 10$$

$$TA \text{ for } P3 = 21 - 3 = 18$$

$$TA \text{ for } P4 = 33 - 4 = 29$$

$$WT = TA - BT$$

$$WT \text{ for process } P0 = 2 - 2 = 0$$

$$WT \text{ for process } P1 = 7 - 6 = 1$$

$$WT \text{ for process } P2 = 10 - 4 = 6$$

$$WT \text{ for process } P3 = 18 - 9 = 9$$

$$WT \text{ for process } P4 = 29 - 12 = 17$$

Problems with FCFS Scheduling

Advantages of FCFS

- Simple
- Easy
- First come, First serve

Disadvantages of FCFS

- The scheduling method is non preemptive, the process will run to the completion.
- Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
- Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter. If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.

Not optimal Average Waiting Time.

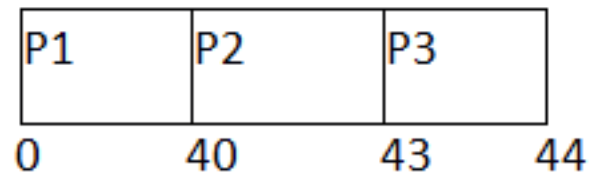
Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

Convoy effect

FCFS may suffer from the **convoy effect** if the burst time of the first job is the highest among all. As in the real life, if a convoy is passing through the road then the other persons may get blocked until it passes completely. This can be simulated in the Operating System also.

If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **convoy effect** or **starvation**.

Processes		Arrival Time	Burst Time
P0	0	40	
P1	1	3	
P2	1	1	



In the First scenario, The Process P1 arrives at the first in the queue although; the burst time of the process is the highest among all. Since, the Scheduling algorithm, we are following is FCFS hence the CPU will execute the Process P1 first.

In this schedule, the average waiting time of the system will be very high. That is because of the convoy effect. The other processes P2, P3 have to wait for their turn for 40 units of time although their burst time is very low. This schedule suffers from starvation.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	0	40	40	40	0
2	1	3	43	42	39
3	1	1	44	43	42

Avg waiting Time = $81/3$

Example 1: In the Second scenario, If Process P1 would have arrived at the last of the queue and the other processes P2 and P3 at earlier then the problem of starvation would not be there.

Following example shows the deviation in the waiting times of both the scenarios. Although the length of the schedule is same that is 44 units but the waiting time will be lesser in this schedule.

Processes		Arrival Time	Burst Time
P0	1	40	
P1	0	3	
P2	1	1	

Example 2: For the given scenario calculate average waiting and average turnaround time.

Processes		Arrival Time	Burst Time
P0	0	3	
P1	1	2	
P2	2	1	
P3	3	4	
P4	4	5	
P5	5	2	

Example 3 For the given problem draw the Gantt chart and calculate average waiting and turnaround time using FCFS scheduling.

Processes	Arrival Time	Burst Time
A	3	4
B	5	3
C	0	2
D	5	1
E	4	3

```
/* Simple C++ program for implementation of FCFS scheduling */
```

```
void findWaitingTime(int processes[], int n, int bt[], int wt[])  
{  
    // waiting time for first process will be 0  
    wt[0] = 0;  
    // calculating waiting time  
    for (int i = 1; i < n ; i++)  
        { wt[i] = bt[i-1] + wt[i-1];  
        }  
}
```

```
// function to calculate turn around time
```

```
void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])  
{  
    // calculating turnaround time by adding  
    // bt[i] + wt[i]  
    for (int i = 0; i < n ; i++)  
    {  
        tat[i] = bt[i] + wt[i];  
    }  
}
```

```

void findAverageTime( int processes[], int n, int bt[])
    { int wt[n], tat[n], total_wt = 0, total_tat = 0;
// function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);
// function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);
// display processes along with all details
    cout << "Processes " << " Burst time " << " Waiting time " << " Turn around
time\n";

// calculate total waiting time and total turn around time
for (int i = 0; i < n; i++)
{
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    cout << " " << i+1 << "\t\t" << bt[i] << "\t " << wt[i] << "\t\t " << tat[i] << endl;
}
cout << "Average waiting time = " << (float)total_wt / (float)n;
cout << "\nAverage turn around time = " << (float)total_tat / (float)n;
}

```

Shortest Job First

- Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.
- In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.
- However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Q1. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using SJF scheduling.

Processes	Burst Time
A	4
B	3
C	2
D	1
E	2

D	C	E	B	A	12
0	1	3	5	8	

T.A of Process A=12

B=8

C=3

D=1

E=5

W.T. for process A=12-4=8

B=8-3=5

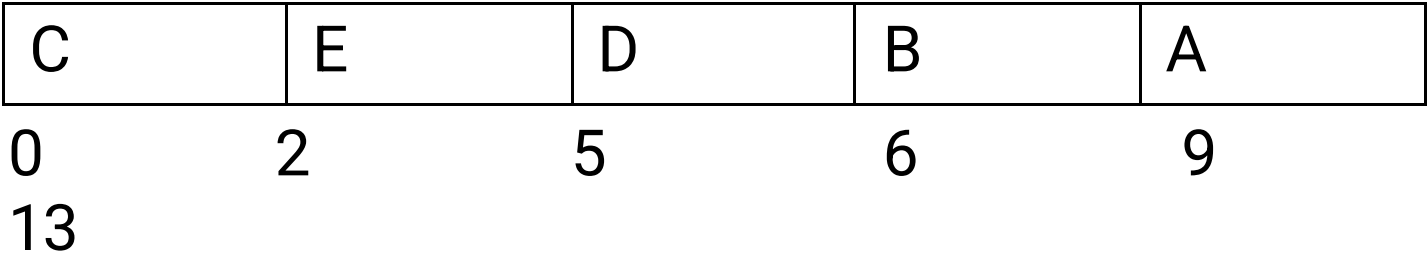
C=3-2=1

D=1-1=0

E=5-2=3

Q2. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using Non Pre-emptive SJF scheduling.

Processes	Arrival Time	Burst Time
A	2	4
B	5	3
C	0	2
D	5	1
E	2	3



Q3. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using Pre-emptive SJF scheduling.

Processes	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

P1	P2	P4	P1	P3
0	1	5	10	17

Turn Around time for process P1=17-0=17
Turn Around time for process P2=5-1=4
Turn Around time for process P3=26-2=24
Turn Around time for process P4=10-3=7

Waiting Time for Process P1=17-8=9
Waiting Time for Process P2=4-4=0
Waiting Time for Process P3=24-9=15
Waiting Time for Process P4=7-5=2

Q4. Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is pre-emptive shortest remaining time first. The average turn around time of these processes is -----milliseconds.

- (A)8.25
- (B)10.25
- (C).6.35
- (D).4.25

Processes	Arrival Time	Burst Time
P1	0	10
P2	3	6
P3	7	1
P4	8	3

Q5. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using Non Pre-emptive and Pre-emptive SJF scheduling.

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

Algorithm:

$n=5$

1. Enter no. of process, Arrival time, CPU Burst Time of each of the n process.
2. completed = 0 ;

current_time = 0;

while(completed != n) // all n processes are not completed

{

find process with **minimum CPU burst** from
the Ready Queue till current time

if(such process is found)

{

//find all metrics e.g. CT, TAT, WT, RT

// Throughput, CPU utilization

completed++

mark that process as completed

current_time = ct ;

}

else

current_time++;

} //end of while

3. Print the required Outputs

Proces s	Arrival Time	CPU Burst Time
P1	0	5
P2	8	4
P3	9	6
P4	8	2
P5	10	3

Algorithm:

1. Enter no. of process, Arrival time, CPU Burst Time of each of the n process.

2. completed = 0 ;

current_time = 0;

while(completed != n) // all n processes are not completed

{

find process with minimum CPU burst from the Ready Queue till current time

if(min_index != -1)

{

//find all metrics e.g. CT, TAT, WT, RT

// Throughput, CPU utilization

mark that process as completed

completed++

current_time = ct ;

}

else

current_time++;

}

3. Print the required Outputs

	AT	BT
P1	0	5
P2	8	4
P3	9	6
P4	8	2
P5	10	3

```
int min_index = -1;
int minimum = INT_MAX;
for(int i = 0; i < n; i++) {
    if(ps[i].at <= current_time && is_completed[i] == 0)
    {
        if(ps[i].bt < minimum) {
            minimum = ps[i].bt;
            min_index = i;
        }
        if(ps[i].bt == minimum) {
            if(ps[i].at < ps[min_index].at) {
                minimum = ps[i].bt;
                min_index = i;
            }
        }
    }
}
```

SJF Scheduling:

```
#include <iostream>
using namespace std;
```

```
//Shortest Process on Arrival Time
```

```
int shortest(int *burst,int *arrival,int time,int number)
{
    int index = -1;
    for(int i = 0;i < number;i++)
    {
        if(arrival[i] <= time && burst[i] != 0 )
        {
            if(index == -1)
                index = i;
            else if(burst[index] > burst[i])
                index = i;
        }
    }
    return index;
}
```



```
//Controller
void controller(int *burst,int *arrival,int n)
{
    int *finish = new int[n];
    int *temp_burst = new int[n];
    for(int i=0;i < n;i++)
        temp_burst[i] = burst[i];
    int temp,control = 1;

    //Finding the Starting time
    int start=arrival[0];

    for(int i = 1;i < n;i++)
        if(arrival[i] < start)
            start = arrival[i];

    //Finding Maximum Time Required
    int max = start;
    for(int i = 0;i < n;i++)
        max += burst[i];
}
```

//Process Control at Every Second

```
cout<<"*****Gantt Chart*****\n";
```

```
cout<<"Start\tProgram No.\tEnd\n";
```

```
for(int i = start;i < max;i++)
```

```
{
```

```
    temp = shortest(burst,arrival,i,n);
```

```
    if(temp != -1)
```

```
        control = temp;
```

```
    if(control != -1)
```

```
        burst[control]--;
```

```
    if(burst[control] == 0)
```

```
        finish[control] = i+1;
```

```
    cout<<i<<"\t "<<control+1<<"\t\t"<<(i+1)<<"\n";
```

```
}
```

```

//Printing Turn-around and Waiting time
double turn = 0,wait = 0;
cout<<"*****Final Chart*****\n";
cout<<"Program No.\tTurnAround\tWaiting\n";

for(int i = 0;i < n;i++)
{
    cout<<i+1<<"\t\t";
    temp = finish[i] - arrival[i];
    cout<<temp<<"\t\t";
    turn = turn + temp;
    temp = finish[i] - (arrival[i] + temp_burst[i]);
    cout<<temp<<"\n";
    wait = wait + temp;
}

cout<<"\n\nAverage Waiting Time : "<<(wait/n)<<"\n";
cout<<"Average Turnaround Time : "<<(turn/n)<<"\n";
}

```

```
Int main()
{
    cout<<"Enter the Number of Processes : ";
    int n;
    cin>>n;
    cout<<"Enter Details of each Program...\n";
    int *burst = new int[n];
    int *arrival = new int[n];

    for(int i = 0;i < n;i++)
    {
        cout<<"Burst Time of Program "<<i+1<<" : ";
        cin>>burst[i];
        cout<<"Arrval Time of Program "<<i+1<<" : ";
        cin>>arrival[i];
    }
    controller(burst,arrival,n);
    return 0;
}
```

Prediction of CPU Burst Time for a process in SJF

- The SJF algorithm is one of the best scheduling algorithms since it provides the maximum throughput and minimal waiting time but the problem with the algorithm is that the OS has no way of determining the burst time of the next process prior to its execution.
- So we can predict the approximate CPU burst time for a process. There are various techniques which can be used to assume the CPU Burst time for a process. Our Assumption needs to be accurate in order to utilize the algorithm optimally.

There are the following techniques used for the assumption of CPU burst time for a process.

Techniques used for the assumption of CPU burst time

1. Static Techniques

Process Size

We can predict the Burst Time of the process from its size. If we have two processes **T_OLD** and **T_New** and the actual burst time of the old process is known as **20 secs** and the size of the process is **20 KB**. And if we know that the size of **P_NEW** is **21 KB**. Then the probability of **P_New** having the similar burst time as **20 secs** is maximum.

If, $P_OLD \rightarrow 20\text{ KB}$

$P_New \rightarrow 21\text{ KB}$

$BT(P_OLD) \rightarrow 20\text{ Secs}$

Then,

$BT(P_New) \rightarrow 20\text{ secs}$

Hence, in this technique, we actually predict the burst time of a new process according to the burst time of an old process of similar size as of new process.

Process Type

Process type –

We can predict Burst-Time depending on the Type of Process. Operating System process (like scheduler, dispatcher, segmentation, fragmentation) are faster than User process (Gaming, application softwares).

Burst-Time for any New O.S process can be predicted from any old O.S process of similar type and same for User process.

Note – Static method for burst time prediction is not reliable as it is always not predicted correctly.

Dynamic Techniques

➤ Exponential Averaging or Aging

Let, A_n be the actual burst time of n th process.

$E(n)$ be the predicted burst time for n th process then the CPU burst time for the next process ($n+1$) will be calculated

$$\text{as, } E(n+1) = \alpha \cdot A_n + (1-\alpha) \cdot E(n)$$

Where, α is the smoothing. Its value lies between 0 and 1.

Smoothing factor (α) – It controls the relative weight of recent and past history in our prediction.

If $\alpha = 0$, $E^{n+1} = E^n$ i.e. no change in value of initial predicted burst time.

If $\alpha = 1$, $E^{n+1} = A^n$ i.e. predicted Burst-Time of new process will always change according to actual Burst-time of n^{th} (previous) process.

So we keep value of $\alpha = 1/2$, recent and past history are equally weighted rather than keeping the value 0 or 1.

Que 1

Calculate the expected value of E5?

Process	BT
P1	4
P2	5
P3	5
P4	6

Given: $\alpha=0.5$ $E1=5$ and actual burst time of four processes given in table

SOLUTION: $E(5) = \alpha \cdot A4 + (1-\alpha) \cdot E(4)$

$$E5 = (0.5) 6 + (0.5) E4 \text{-----(1)}$$

$$E4 = (0.5) 5 + (0.5) E3 \text{-----(2)}$$

$$E3 = (0.5) 5 + (0.5) E2 \text{-----(3)}$$

$$E2 = (0.5) 4 + (0.5) E1$$

$$E2 = 2.0 + 0.5 \cdot 5$$

$$E2 = 2.0 + 2.5 = 4.5$$

Putting value of E2 in eq 3, we can get value of E3 and so on.

Priority Based Scheduling

- Priority scheduling 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 first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.
- Disadvantage:- Indefinite blocking or starvation. A priority scheduling can leave some low priority waiting processes indefinitely for CPU. If the system eventually crashes then all unfinished low priority processes gets lost.

Example 1. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using Non Pre-emptive Priority scheduling. Here we are considering 1 is the lowest 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



$$\text{Avg. TAT} = 41/5 = 8.2$$

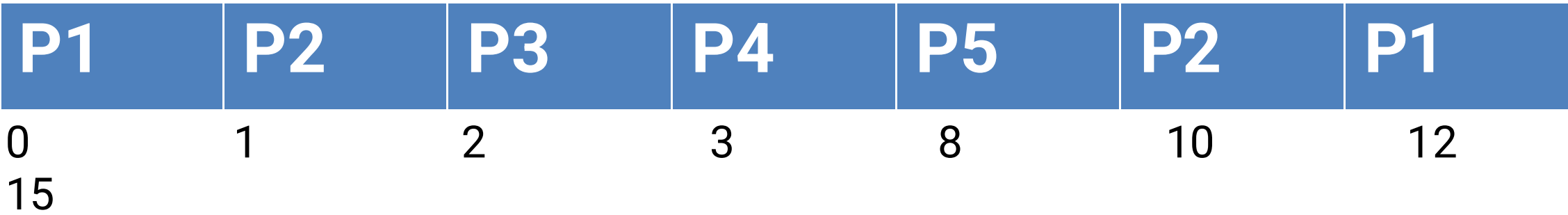
$$\text{Avg. WT} = 26/5 = 5.2 \text{ units}$$

Preemptive Priority Scheduling

- In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.
- The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.
- Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

Q2. For the given problem draw the Gantt chart and calculate average waiting and turnaround time using Pre-emptive Priority scheduling. Here we are considering 1 is the lowest 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



Avg. TAT= $38/5=7.6$
Avg. WT= $24/5=4.8$ units

Round Robin Scheduling Algorithm

Each process is assigned a fixed time(Time Quantum/Time Slice) in cyclic way.

It is designed especially for the time-sharing system. 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.

To implement Round Robin scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and dispatches the process.

The process may have a CPU burst of less than 1-time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system.

A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Round Robin Scheduling

Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2
0									10

P1 waiting time : 4

The average waiting time(AWT) : $(4+6+6)/3=5.33$

P2 waiting time: 6

P3 waiting time: 6

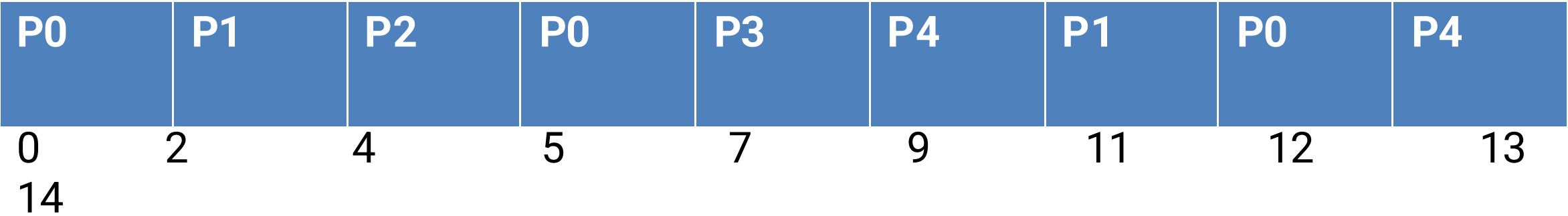
Round robin scheduling algorithm is always preemptive in nature.

It gives the best response time.

Let time quantum be:-2 units.

Performance depends on time quantum.

Process Id.	Arrival time	Burst Time	TAT=Exit time -waiting time(ET-WT)	Waiting time=TAT-BT
P0	0	5	13-0=13	13-5=8
P1	1	3	12-1=11	11-3=8
P2	2	1	5-2=3	3-1=2
P3	3	2	9-3=6	6-2=4
P4	4	3	14-4=10	10-3=7



AVG. TAT=43/5=8.6 AVG.WT=31/5=6.2 UNITS

Exercise : Consider the following set of processes with the length of next CPU burst given in milliseconds:

Process	Arrival Time	Burst Time	Priority
P1	0	2	2
P2	1	1	1
P3	2	8	4
P4	3	4	2
P5	4	5	3

1. Draw four Gantt Chart that illustrate the execution of these processes using the following scheduling algorithms: FCFS, Pre-emptive SJF, Pre-emptive Priority(a larger priority no. implies a higher priority) and RR(TQ=2 milli Sec)
2. What is the Turn around time of each process for each of the scheduling algorithms.
3. What is the Waiting time of each process for each of the scheduling algorithms.
4. Which of the algorithms results in minimum average waiting time(over all the processes)?