Multilevel Feedback Queue: CPU Scheduling Algorithm Simulation

Christopher Jan E. Riños
BS Computer Science
University of the Philippines Visayas Tacloban College
Magsaysay Blvd., Tacloban City, 6500
Email: cerinos@up.edu.ph

Faith Cerena
BS Computer Science
University of the Philippines Visayas Tacloban College
Magsaysay Blvd., Tacloban City, 6500
Email: fcerena@up.edu.ph

***Abstract*—In order for the operating system to efficiently use its resources, processes must be properly scheduled. Hence, there is need to implement a mechanism called CPU scheduling. CPU scheduling is a process which allows one process to utilize the CPU while other processes are waiting for the availability of I/O resources, making full use of the CPU [1].**

***Keywords*— CPU Scheduling, Process, Operating System, FCFS, SJF, SRTF, NPP, PP, Round Robin, Multilevel Feedback Queues**

I. Introduction

The Different operating systems have different types of scheduling algorithms which best fits the requirements and demands of the system. In this report, we will show the execution of different types of CPU scheduling algorithms namely: First Come First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Non-preemptive Priority (NPP), Preemptive Priority (PP), and Round Robin.

II. CPU Scheduling

The operating system is responsible for selecting one processes in the ready to be executed whenever the CPU becomes idle. The short-term scheduler, more known as the CPU scheduler is the one which carries out the selection process. Another component involved in CPU scheduling is the Dispatcher. It is the module that gives of the CPU to process selected by the CPU scheduler. Hence, the dispatcher should be as quick as possible since it is invoked during every process switch [**?**].

III. First Come First Serve

First Come First Serve (FCFS) algorithm is by far the simplest CPU scheduling algorithm at hand. As the name suggests, the process that requests the CPU first is the first to be allocated to the CPU. The implementation of the FCFS is similar to the FIFO (First In First Out) Queue data structure, where the element added to the queue first, is the first one to exit the queue [2].

In the context of CPU Scheduling, whenever a scheduler takes the FCFS strategy, then the process that arrives first will be the first one to be scheduled on the processor regardless of other properties. Since this algorithm is non-preemptive, once we allocate the process to the CPU, we can not stop until that process is finished executing [3].

Advantages of FCFS

FCFS algorithm is simple and east to understand and implement.

Disadvantages of FCFS

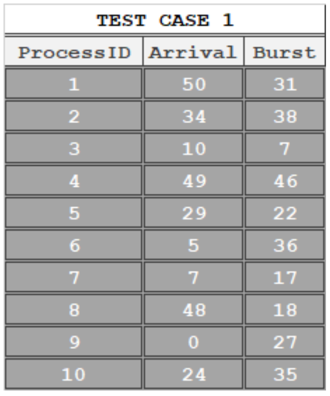
This often leads to the *Convoy effect*— a phenomenon where short processes at the back of the queue have to wait for long processes to finish executing, thus resulting to poor resource utilization [2].

Implementation of FCFS

1) Input the arrival time (at), and burst time (bt) of each process.
2) Sort the all the processes in increasing order according to the arrival time.
3) Execute process according the time of arrival, the first to arrive gets to execute first.

A. Test case 1

In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 1.1 for the information of each process.*

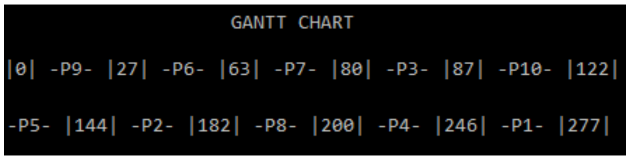


TEST CASE 1

ProcessID	Arrival	Burst
1	50	31
2	34	38
3	10	7
4	49	46
5	29	22
6	5	36
7	7	17
8	48	18
9	0	27
10	24	35

Figure 1.1: FCFS (10 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 1.2.

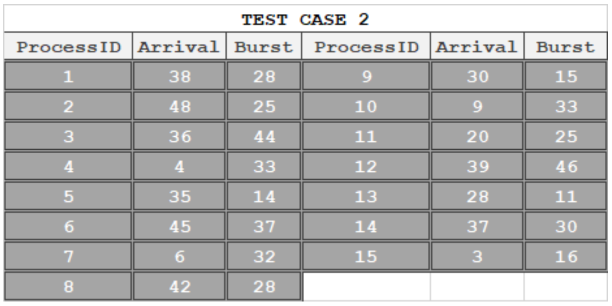


```
                    GANTT CHART

|0| -P9- |27| -P6- |63| -P7- |80| -P3- |87| -P10- |122|

-P5- |144| -P2- |182| -P8- |200| -P4- |246| -P1- |277|
```

Figure 1.2: FCFS (Gantt Chart for Test case 1)

B. Test case 2

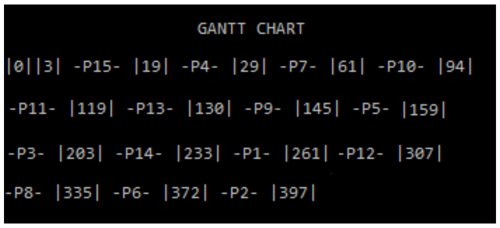
In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 2.1 for the information of each process.*



TEST CASE 2					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	38	28	9	30	15
2	48	25	10	9	33
3	36	44	11	20	25
4	4	33	12	39	46
5	35	14	13	28	11
6	45	37	14	37	30
7	6	32	15	3	16
8	42	28			

Figure 2.1: FCFS (15 Processes)

Given the information above, the Gantt chart of the test case 2 will be shown in Figure 2.2.

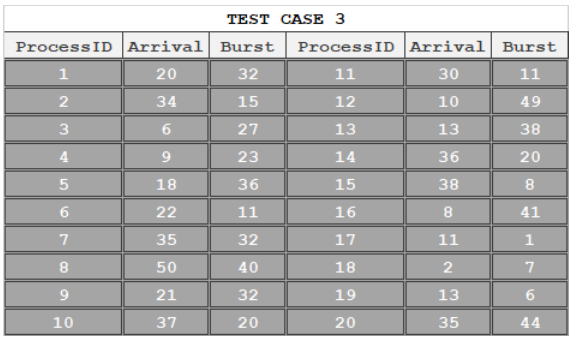


```
              GANTT CHART
|0||3| -P15- |19| -P4- |29| -P7- |61| -P10- |94|

-P11- |119| -P13- |130| -P9- |145| -P5- |159|

-P3- |203| -P14- |233| -P1- |261| -P12- |307|

-P8- |335| -P6- |372| -P2- |397|
```

Figure 2.2 FCFS (Gantt Chart for Test case 2)

C. Test case 3

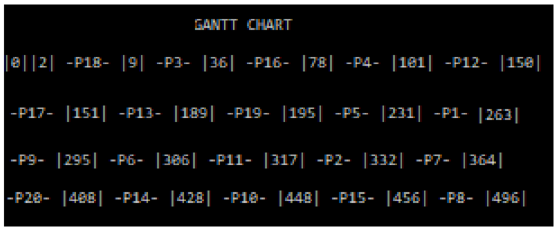
In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 3.1 for the information of each process.*



TEST CASE 3					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	20	32	11	30	11
2	34	15	12	10	49
3	6	27	13	13	38
4	9	23	14	36	20
5	18	36	15	38	8
6	22	11	16	8	41
7	35	32	17	11	1
8	50	40	18	2	7
9	21	32	19	13	6
10	37	20	20	35	44

Figure 3.1: FCFS (20 Processes)

Given the information above, the Gantt chart of the test case 3.2 will be shown in Figure 3.2.



```
                    GANTT CHART

|0||2| -P18- |9| -P3- |36| -P16- |78| -P4- |101| -P12- |150|

-P17- |151| -P13- |189| -P19- |195| -P5- |231| -P1- |263|

-P9- |295| -P6- |306| -P11- |317| -P2- |332| -P7- |364|

-P20- |408| -P14- |428| -P10- |448| -P15- |456| -P8- |496|
```

Figure 3.2 FCFS (Gantt Chart for Test case 3)

IV. Shortest Job First

Another approach to CPU scheduling is what we call the Shortest Job First (SJF) algorithm. This type of scheduling algorithm selects the process for execution which has the smallest amount of time remaining until its completion. One requisite of this strategy is that we should know the execution time of each process before running [4].

This strategy of scheduling is a non-preemptive discipline in which the process with the smallest burst time will be executed first but also basing on the arrival time of each process.

Advantages of SJF

The implementation of the SJF CPU scheduling algorithm results to the minimum average waiting time to each process.

Disadvantages of SJF

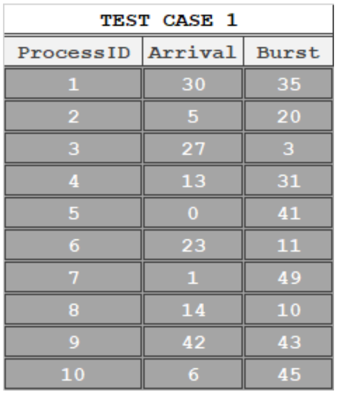
The burst time of each process should be known prior to the execution resulting to additional costs. The implementation of the SJF algorithm also creates the problem of *Starvation*—occurs when a process waits for an indefinite period of time to get the resources it requires.

Implementation of SJF

1) Input the arrival time, and burst time of each process.
2) Sort the all the processes in increasing order according to the burst time.
3) Then simply, apply the FCFS algorithm.

A. Test case 1

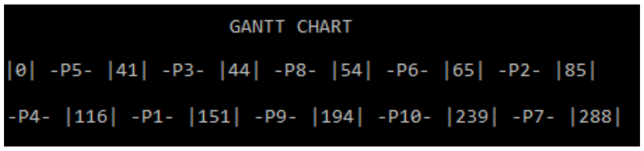
In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 4.1 for the information of each process.*



TEST CASE 1		
ProcessID	Arrival	Burst
1	30	35
2	5	20
3	27	3
4	13	31
5	0	41
6	23	11
7	1	49
8	14	10
9	42	43
10	6	45

Figure 4.1: SJF (10 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 4.2.

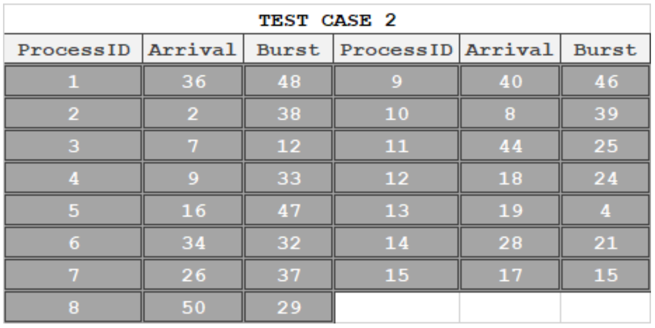


```
                    GANTT CHART
|0| -P5- |41| -P3- |44| -P8- |54| -P6- |65| -P2- |85|
-P4- |116| -P1- |151| -P9- |194| -P10- |239| -P7- |288|
```

Figure 4.2: SJF Gantt Chart for Test case 1

B. Test case 2

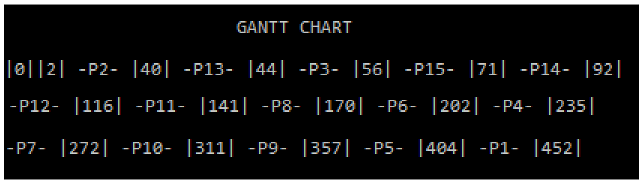
In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 5.1 for the information of each process.*



TEST CASE 2					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	36	48	9	40	46
2	2	38	10	8	39
3	7	12	11	44	25
4	9	33	12	18	24
5	16	47	13	19	4
6	34	32	14	28	21
7	26	37	15	17	15
8	50	29			

Figure 5.1: SJF (15 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 5.2.

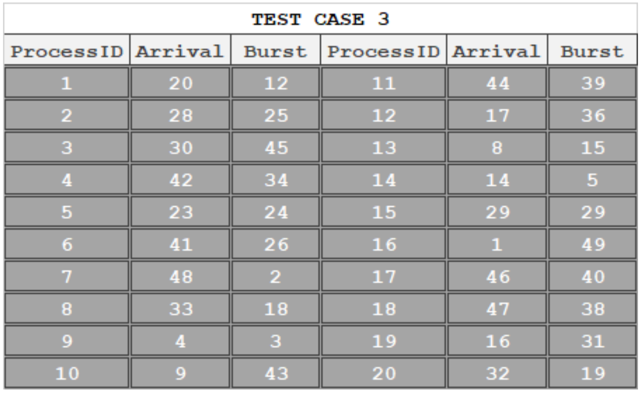


```
                    GANTT CHART
|0||2| -P2- |40| -P13- |44| -P3- |56| -P15- |71| -P14- |92|
-P12- |116| -P11- |141| -P8- |170| -P6- |202| -P4- |235|
-P7- |272| -P10- |311| -P9- |357| -P5- |404| -P1- |452|
```

Figure 5.2: SJF Gantt Chart for Test case 2

C. Test case 3

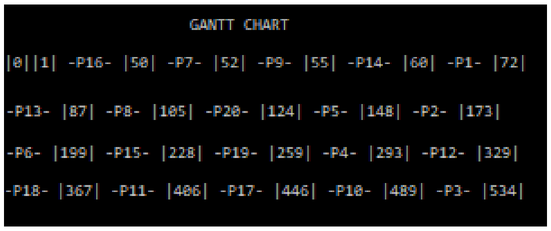
In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 6.1 for the information of each process.*



TEST CASE 3					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	20	12	11	44	39
2	28	25	12	17	36
3	30	45	13	8	15
4	42	34	14	14	5
5	23	24	15	29	29
6	41	26	16	1	49
7	48	2	17	46	40
8	33	18	18	47	38
9	4	3	19	16	31
10	9	43	20	32	19

Figure 6.1: SJF (20 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 6.2.



```
                    GANTT CHART
|0||1| -P16- |50| -P7- |52| -P9- |55| -P14- |60| -P1- |72|
-P13- |87| -P8- |105| -P20- |124| -P5- |148| -P2- |173|
-P6- |199| -P15- |228| -P19- |259| -P4- |293| -P12- |329|
-P18- |367| -P11- |406| -P17- |446| -P10- |489| -P3- |534|
```

Figure 6.2: SJF Gantt Chart for Test case 3

V. Shortest Remaining Time First

The Shortest Remaining Time First (SRTF) algorithm is the preemptive version of the Shortest Job First. In SRTF, the selection of a job is similar to the SJF, but the difference is that in SJF, processes will run until completion, while in SRTF, the processes will run until completion or a new process arrives with a smaller execution time than the remaining execution time of the process currently running [5].

Advantages of SRTF

Similar to the SJF, but the SRTF CPU scheduling algorithm results to lesser minimum average waiting time and shorter jobs will be scheduled more quickly.

Disadvantages of SRTF

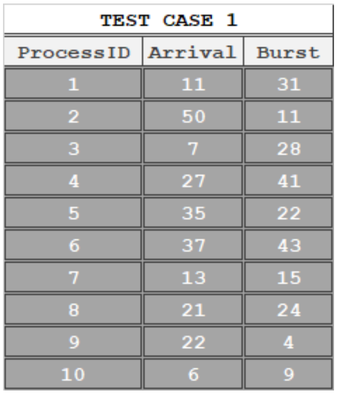
Just like the SJF, the SRTF algorithm can also result to starvation. In other words, whenever a system contains a lot of short jobs and one long job, then there is a possibility that the long process might never be able to run [6].

Implementation of SRTF

1) Input the arrival time, and burst time of each process.
2) Traverse until all process gets completely executed..
    - Find process with minimum remaining time at every single time lap.
    - Reduce its time by 1.
    - Check if its remaining time is 0
    - Increment the counter of process completion
    - Completion time (ct) of current process = ct + 1;

A. Test case 1

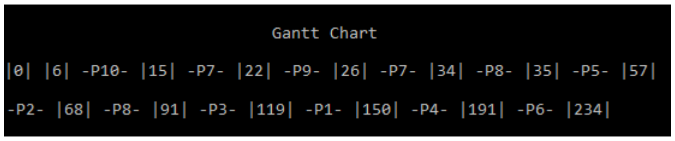
In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 7.1 for the information of each process.*



TEST CASE 1		
ProcessID	Arrival	Burst
1	11	31
2	50	11
3	7	28
4	27	41
5	35	22
6	37	43
7	13	15
8	21	24
9	22	4
10	6	9

Figure 7.1: SRTF (10 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 7.2.

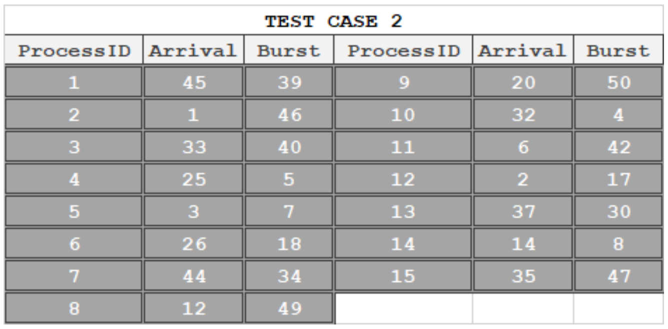


```
                         Gantt Chart
|0| |6| -P10- |15| -P7- |22| -P9- |26| -P7- |34| -P8- |35| -P5- |57|
-P2- |68| -P8- |91| -P3- |119| -P1- |150| -P4- |191| -P6- |234|
```

Figure 7.2: SRTF Gantt Chart for Test case 1

B. Test case 2

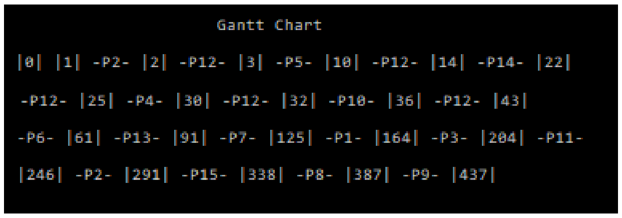
In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 8.1 for the information of each process.*



TEST CASE 2					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	45	39	9	20	50
2	1	46	10	32	4
3	33	40	11	6	42
4	25	5	12	2	17
5	3	7	13	37	30
6	26	18	14	14	8
7	44	34	15	35	47
8	12	49			

Figure 8.1: SRTF (15 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 8.2.

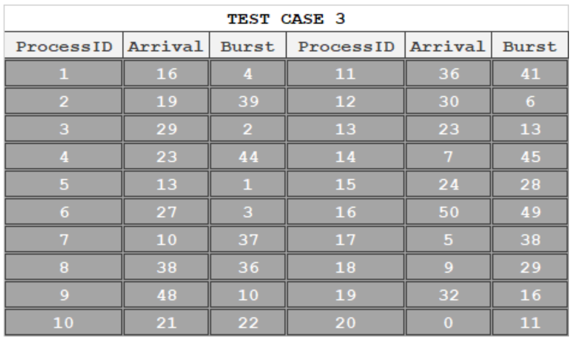


```
                   Gantt Chart
|0| |1| -P2- |2| -P12- |3| -P5- |10| -P12- |14| -P14- |22|
-P12- |25| -P4- |30| -P12- |32| -P10- |36| -P12- |43|
-P6- |61| -P13- |91| -P7- |125| -P1- |164| -P3- |204| -P11-
|246| -P2- |291| -P15- |338| -P8- |387| -P9- |437|
```

Figure 8.2: SRTF Gantt Chart for Test case 2

C. Test case 3

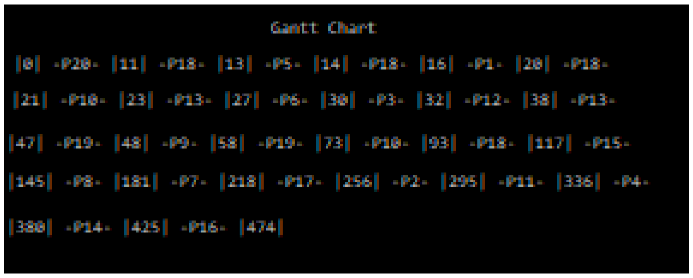
In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 9.1 for the information of each process.*



TEST CASE 3					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	16	4	11	36	41
2	19	39	12	30	6
3	29	2	13	23	13
4	23	44	14	7	45
5	13	1	15	24	28
6	27	3	16	50	49
7	10	37	17	5	38
8	38	36	18	9	29
9	48	10	19	32	16
10	21	22	20	0	11

Figure 9.1: SRTF (20 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 9.2.



```
                        Gantt Chart
|0| -P20- |11| -P18- |13| -P5- |14| -P18- |16| -P1- |20| -P18-
|21| -P10- |23| -P13- |27| -P6- |30| -P3- |32| -P12- |38| -P13-
|47| -P19- |48| -P9- |58| -P19- |73| -P10- |93| -P18- |117| -P15-
|145| -P8- |181| -P7- |218| -P17- |256| -P2- |295| -P11- |336| -P4-
|380| -P14- |425| -P16- |474|
```

Figure 9.2: SRTF Gantt Chart for Test case 3

VI. Priority Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, which is different from other types of scheduling. Priority scheduling can be either preemptive or non-preemptive. Each process is assigned a priority. Processes with the highest priority is to be executed first and so on [7]. In case there are processes with the same priority, the strategy of First Come First Serve takes place.

Advantages of Priority Scheduling

Jobs with the highest priority are executed first. Process taking long time will delay low priority but short processes.

Disadvantages of Priority Scheduling

Since a process may have very low priority. Major problem with priority scheduling is starvation.

Implementation of Priority Scheduling

1) Input the arrival time, burst time, and priority.
2) Sort the processes, burst time and priority according to the priority.
3) Then simply, apply the FCFS algorithm.

A. Non-preemptive Priority

The non-preemptive priority simply puts the new process at the head of the ready queue. When a process enters the state of running, the process is not removed from the scheduler until it finishes its burst time [7].

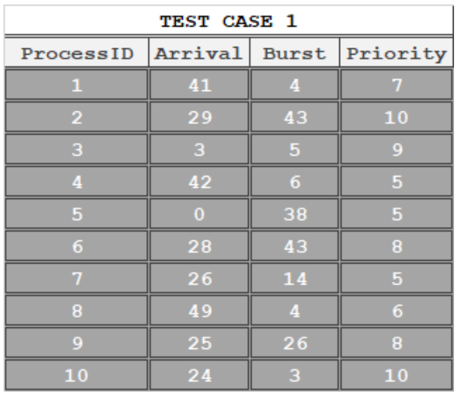
Advantages of Non-preemptive Priority

The non-preemptive priority scheduling algorithm reduces context-switch overhead and also reduces stack size.

Disadvantages of Non-preemptive Priority

The utilization bound under non preemptive scheduling drops to zero. Scheduling reduces schedulability introducing blocking delays in high priority tasks [8].

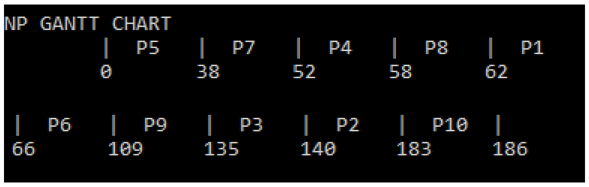
1) Test Case 1: In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 10.1 for the information of each process.*



TEST CASE 1			
ProcessID	Arrival	Burst	Priority
1	41	4	7
2	29	43	10
3	3	5	9
4	42	6	5
5	0	38	5
6	28	43	8
7	26	14	5
8	49	4	6
9	25	26	8
10	24	3	10

Figure 10.1: NPP (10 Processes)

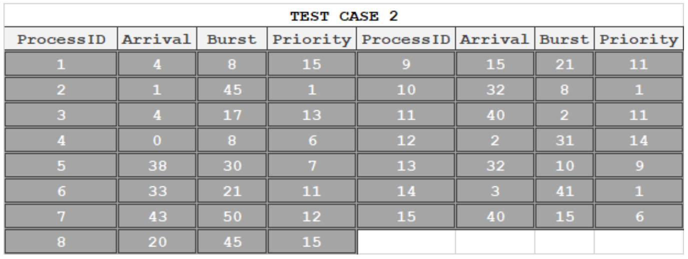
Given the information above, the Gantt chart of the test case 1 will be shown in Figure 10.2.



```
NP GANTT CHART
         |  P5    |  P7    |  P4    |  P8    |  P1
         0        38       52       58       62

 |  P6    |  P9    |  P3    |  P2    |  P10   |
 66       109      135      140      183      186
```

Figure 10.2: NPP Gantt Chart for Test case 1

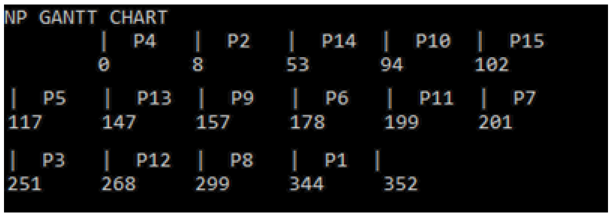
2) Test Case 2: In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 11.1 for the information of each process.*



TEST CASE 2							
ProcessID	Arrival	Burst	Priority	ProcessID	Arrival	Burst	Priority
1	4	8	15	9	15	21	11
2	1	45	1	10	32	8	1
3	4	17	13	11	40	2	11
4	0	8	6	12	2	31	14
5	38	30	7	13	32	10	9
6	33	21	11	14	3	41	1
7	43	50	12	15	40	15	6
8	20	45	15				

Figure 11.1: NPP (15 Processes)

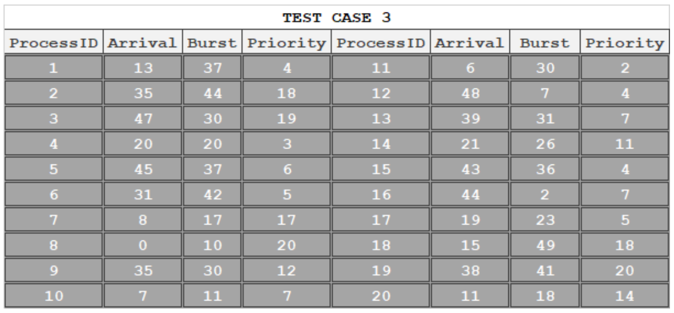
Given the information above, the Gantt chart of the test case 1 will be shown in Figure 11.2.



```
NP GANTT CHART
         |  P4    |  P2    |  P14   |  P10   |  P15
         0        8        53       94       102

|  P5    |  P13   |  P9    |  P6    |  P11   |  P7
117      147      157      178      199      201

|  P3    |  P12   |  P8    |  P1    |
251      268      299      344      352
```

Figure 11.2: NPP Gantt Chart for Test case 2

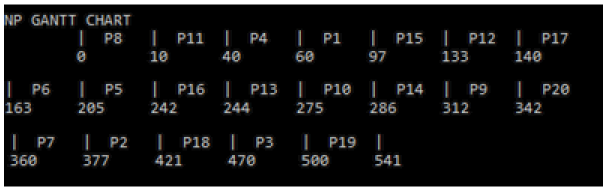
3) Test Case 3: In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 12.1 for the information of each process.*



TEST CASE 3							
ProcessID	Arrival	Burst	Priority	ProcessID	Arrival	Burst	Priority
1	13	37	4	11	6	30	2
2	35	44	18	12	48	7	4
3	47	30	19	13	39	31	7
4	20	20	3	14	21	26	11
5	45	37	6	15	43	36	4
6	31	42	5	16	44	2	7
7	8	17	17	17	19	23	5
8	0	10	20	18	15	49	18
9	35	30	12	19	38	41	20
10	7	11	7	20	11	18	14

Figure 12.1: NPP (20 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 12.2.



```
NP GANTT CHART
        |  P8    |  P11   |  P4    |  P1    |  P15   |  P12   |  P17
        0        10       40       60       97       133      140

|  P6    |  P5    |  P16   |  P13   |  P10   |  P14   |  P9    |  P20
163      205      242      244      275      286      312      342

 |  P7    |  P2    |  P18   |  P3    |  P19   |
 360      377      421      470      500      541
```

Figure 12.2: NPP Gantt Chart for Test case 3

B. Preemptive Priority

The preemptive priority is a method based on priority. This scheduling involves priority assignment to every process and the process that is currently utilized is the process which has the highest priority [7].

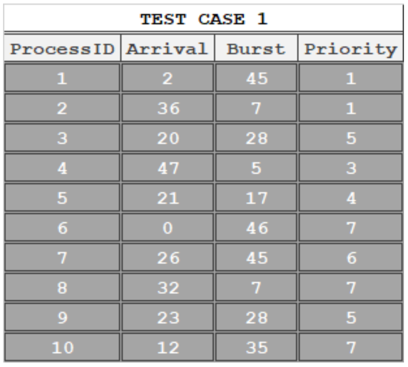
Advantages of Preemptive Priority

The preemptive priority algorithm is focused on fully preemptive systems because they allow higher responsiveness.

Disadvantages of Preemptive Priority

Context switch cost: time taken by the scheduler to suspend the running task.

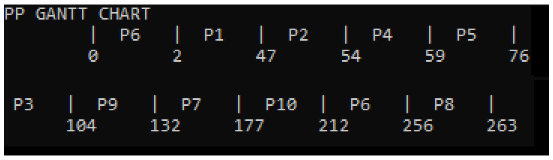
1) Test Case 1: In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 13.1 for the information of each process.*



TEST CASE 1			
ProcessID	Arrival	Burst	Priority
1	2	45	1
2	36	7	1
3	20	28	5
4	47	5	3
5	21	17	4
6	0	46	7
7	26	45	6
8	32	7	7
9	23	28	5
10	12	35	7

Figure 13.1: PP (10 Processes)

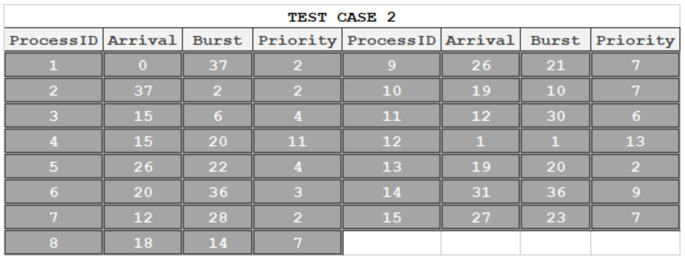
Given the information above, the Gantt chart of the test case 1 will be shown in Figure 13.2.



```
PP GANTT CHART
        |  P6    |  P1    |  P2    |  P4    |  P5    |
        0        2        47       54       59       76

P3    |  P9    |  P7    |  P10   |  P6    |  P8    |
     104      132      177      212      256      263
```

Figure 13.2: PP Gantt Chart for Test case 1

2) Test Case 2: In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 14.1 for the information of each process.*

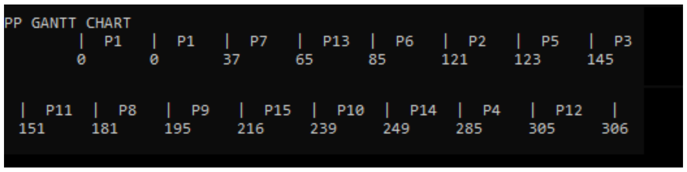


TEST CASE 2

ProcessID	Arrival	Burst	Priority	ProcessID	Arrival	Burst	Priority
1	0	37	2	9	26	21	7
2	37	2	2	10	19	10	7
3	15	6	4	11	12	30	6
4	15	20	11	12	1	1	13
5	26	22	4	13	19	20	2
6	20	36	3	14	31	36	9
7	12	28	2	15	27	23	7
8	18	14	7				

Figure 14.1: PP (15 Processes)

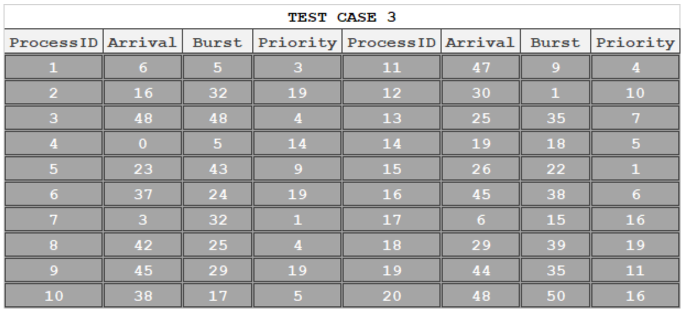
Given the information above, the Gantt chart of the test case 1 will be shown in Figure 14.2.



```
PP GANTT CHART
      |  P1   |  P1   |  P7   |  P13  |  P6   |  P2   |  P5   |  P3
      0       0       37      65      85      121     123     145

 |  P11  |  P8   |  P9   |  P15  |  P10  |  P14  |  P4   |  P12   |
 151     181     195     216     239     249     285     305     306
```

Figure 14.2: PP Gantt Chart for Test case 2

3) Test Case 3: In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 15.1 for the information of each process.*

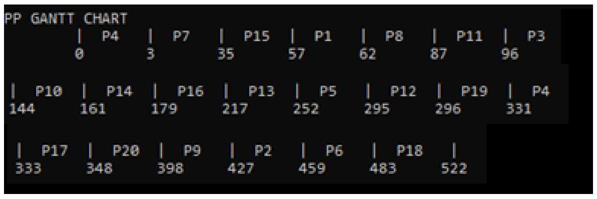


TEST CASE 3

ProcessID	Arrival	Burst	Priority	ProcessID	Arrival	Burst	Priority
1	6	5	3	11	47	9	4
2	16	32	19	12	30	1	10
3	48	48	4	13	25	35	7
4	0	5	14	14	19	18	5
5	23	43	9	15	26	22	1
6	37	24	19	16	45	38	6
7	3	32	1	17	6	15	16
8	42	25	4	18	29	39	19
9	45	29	19	19	44	35	11
10	38	17	5	20	48	50	16

Figure 15.1: PP (20 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 15.2.



```
PP GANTT CHART
      |  P4   |  P7   |  P15  |  P1   |  P8   |  P11  |  P3
      0       3       35      57      62      87      96

|  P10  |  P14  |  P16  |  P13  |  P5   |  P12  |  P19  |  P4
144     161     179     217     252     295     296     331

 |  P17  |  P20  |  P9   |  P2   |  P6   |  P18  |
 333     348     398     427     459     483     522
```

Figure 15.2: PP Gantt Chart for Test case 3

VII. Round Robin

Round Robin scheduling is a CPU Scheduling algorithm that is considered to be very fair since it uses time slices — called *quantum* — that are assigned to each process in the queue. Each process is allowed to execute for a given amount of time, and if that process does not complete its execution during the allotted time, it is preempted and moved at the back of the queue so that the next process in line can execute for the same amount of time [9].

Setting the Quantum size

Short quantum size is an advantage when there are many short jobs that do not take up the entire time-slice. It is better if the quantum size is not too short, otherwise more time will be wasted during context switching.

Advantages of Round Robin

This algorithm is easy to implement. This algorithm is fair in the sense that each job gets an equal amount of processing time.

Disadvantages of Round Robin

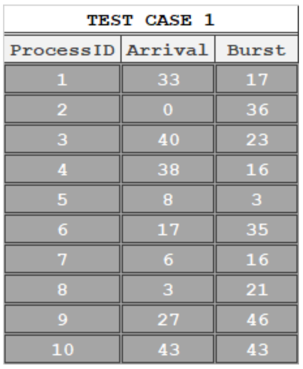
Sometimes it is better not to give jobs equal amount of processing time. Highly interactive processes will get scheduled no more frequently than CPU processes [8].

Implementation of Round Robin

1) The queue structure in ready queue is of First In First Out (FIFO) type.
2) A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.
3) The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.
4) The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.
5) The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.
6) The same steps are repeated until all the process are finished.

A. *Test case 1*

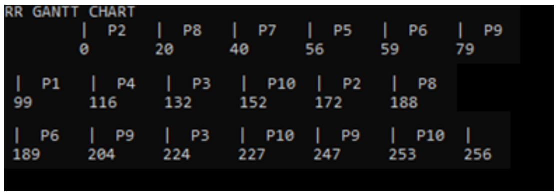
In this test case, we are given 10 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 16.1 for the information of each process.*



TEST CASE 1		
ProcessID	Arrival	Burst
1	33	17
2	0	36
3	40	23
4	38	16
5	8	3
6	17	35
7	6	16
8	3	21
9	27	46
10	43	43

Figure 16.1: RR (10 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 16.2.

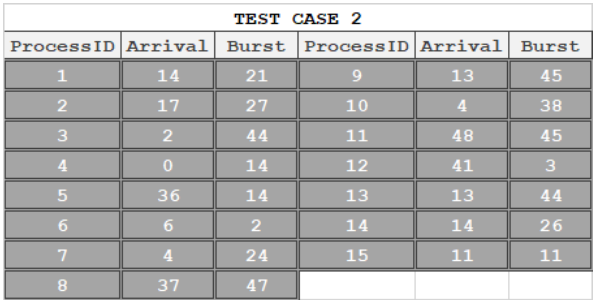


```
RR GANTT CHART
        |  P2   |  P8   |  P7   |  P5   |  P6   |  P9
        0       20      40      56      59      79

 |  P1   |  P4   |  P3   |  P10  |  P2   |  P8
99      116     132     152     172     188

 |  P6   |  P9   |  P3   |  P10  |  P9   |  P10  |
189     204     224     227     247     253     256
```

Figure 16.2: RR Gantt Chart for Test case 1

B. Test case 2

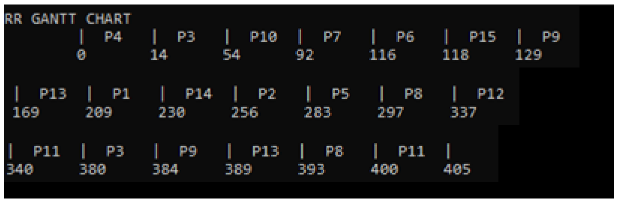
In this test case, we are given 15 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 17.1 for the information of each process.*



TEST CASE 2					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	14	21	9	13	45
2	17	27	10	4	38
3	2	44	11	48	45
4	0	14	12	41	3
5	36	14	13	13	44
6	6	2	14	14	26
7	4	24	15	11	11
8	37	47			

Figure 17.1: RR (15 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 17.2.

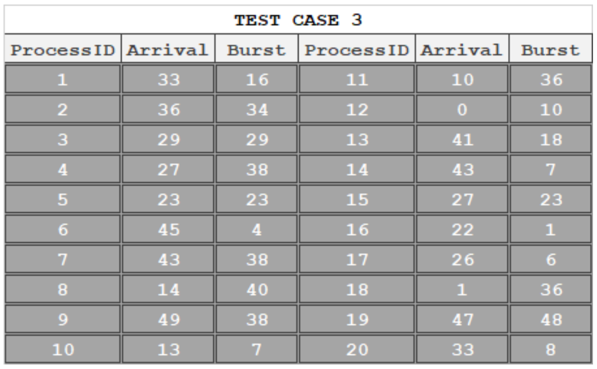


```
RR GANTT CHART
       |  P4   |  P3   |  P10  |  P7   |  P6   |  P15  |  P9
       0       14      54      92      116     118     129

 |  P13  |  P1   |  P14  |  P2   |  P5   |  P8   |  P12
169     209     230     256     283     297     337

|  P11  |  P3   |  P9   |  P13  |  P8   |  P11  |
340     380     384     389     393     400     405
```

Figure 17.2: RR Gantt Chart for Test case 2

C. Test case 3

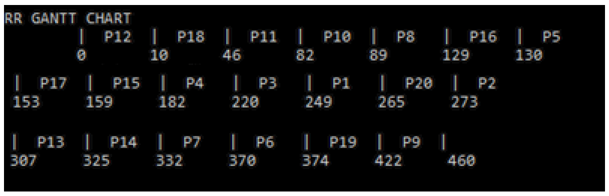
In this test case, we are given 20 processes with random arrival time ranging from 0 - 50 and random burst time ranging from 1 - 50. *See Fig. 18.1 for the information of each process.*



TEST CASE 3					
ProcessID	Arrival	Burst	ProcessID	Arrival	Burst
1	33	16	11	10	36
2	36	34	12	0	10
3	29	29	13	41	18
4	27	38	14	43	7
5	23	23	15	27	23
6	45	4	16	22	1
7	43	38	17	26	6
8	14	40	18	1	36
9	49	38	19	47	48
10	13	7	20	33	8

Figure 18.1: SJF (20 Processes)

Given the information above, the Gantt chart of the test case 1 will be shown in Figure 18.2.



```
RR GANTT CHART
       |  P12  |  P18  |  P11  |  P10  |  P8   |  P16  |  P5
       0       10      46      82      89      129     130

 |  P17  |  P15  |  P4   |  P3   |  P1   |  P20  |  P2
153     159     182     220     249     265     273

 |  P13  |  P14  |  P7   |  P6   |  P19  |  P9   |
307     325     332     370     374     422     460
```

Figure 18.2: RR Gantt Chart for Test case 3

VIII. Conclusion

One of the most major and difficult task in CPU Scheduling is choosing the appropriate scheduling algorithm to use for a specific system. There are countless of ways one can decide how to schedule process, but there are a lot of factors that also need to be considered while making such a decision.

References

[1] Studytonight, *CPU Scheduling in Operating System*, Studytonight.com, 2018. [Online]. Available: https://www.studytonight.com/operating-system/cpu-scheduling. [Accessed: 19-April-2018]

[2] Studytonight, *First Come First Serve (FCFS)*, Studytonight.com, 2018. [Online]. Available: https://www.studytonight.com/operating-system/first-come-first-serve. [Accessed: 19-April-2018]

[3] TheJavaProgrammer, *Java Program for First Come First Serve (FCFS) Scheduling Algorithm*, TheJavaProgrammer.com, October 22, 2017. [Online]. Available: https://www.thejavaprogrammer.com/java-program-first-come-first-serve-fcfs-scheduling-algorithm/. [Accessed: 19-April-2018]

[4] TheJavaProgrammer, *Java Program for Shortest Job First (SJF) Scheduling*, TheJavaProgrammer.com, November 2, 2017. [Online]. Available: https://www.thejavaprogrammer.com/java-program-shortest-job-first-sjf-scheduling/. [Accessed: 19-April-2018]

[5] S. Mittal, *Shortest Remaining Time First (Preemptive And Non Preemptive) Sjf Scheduling Algorithm With Example*, Javahungry.blogspot.com, 2013. [Online]. Available: http://javahungry.blogspot.com/2013/11/shortest-remaining-time-first-srt-preemptive-non-preemptive-sjf-scheduling-algorithm-with-example-java-program-code.html. [Accessed: 19-April-2018]

[6] Wikibooks, *Operating System Design* , wikibooks.org. [Online]. Available: https://en.wikibooks.org/wiki/Operating-System-Design/Scheduling-Processes/SPN. [Accessed: 19-April-2018]

[7] Techopedia, *Priority Scheduling*, techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/21478/priority-scheduling. [Accessed: 19-April-2018]

[8] COEP Wikim *Scheduling Algorithms*, http://foss.coep.org.in. [Online]. Available: http://foss.coep.org.in/coepwiki/index.php/Scheduling-Algorithms. [Accessed: 19-April-2018]

[9] Techopedia, *Round Robin Scheduling (RRS)*, techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/9236/round-robin-scheduling-rrs. [Accessed: 19-April-2018]