

IMPROVISATION OF CPU SCHEDULING
ALGORITHM FOR REAL LIFE
SYSTEMS

J-Component Project Report

Operating System

CSE2005

Winter Semester 2020-21

By Project Members:

Deepsi Kumari (19BCE1064)

Ishita Kumar (19BCE1551)

Heetakshi Fating(19BAI1095)

Faculty: Prof. Santoshi Ganala



Abstract:

In this project, we plan to come up with an optimized variant to the Round Robin scheduling algorithm and try to improve its efficiency. The CPU scheduler is an important component of the operating system. Processes must be properly scheduled, or else the system will make inefficient use.

Introduction:

1. One of the most crucial problems in operating systems concepts is Scheduling the CPU to different processes and to design a particular system that will attain accurate results in scheduling.
2. Priorities can be either dynamic or static. Static priorities are allocated during creation, whereas dynamic priorities are assigned depending on the behavior of the processes while in the system. To illustrate, the scheduler could favor input/output (I/O) intensive tasks, which lets expensive requests to be issued as soon as possible.
3. In case of priority based Round Robin scheduling algorithm when similar priority jobs arrive, the processes are executed based on FCFS. The processes have a specific burst time associated.
4. A time quantum is set, and the processes utilize the resources available for that time quantum only. Once the time quantum is crossed, the control is passed to the next process.
5. The purpose of this project is to introduce an optimized variant to the round robin scheduling algorithm. Every algorithm works in its own way and has its own merits and demerits.

Existing Systems:

Scheduling usually refers to selecting which process to run next - but it can also refer to which input/output (or disk) operation to do next. (Usually, if no modifier is attached, scheduling refers to process scheduling.).

The Round Robin algorithm is generally used in time sharing environments.

Windows uses a round-robin technique with a multi-level feedback queue for priority scheduling ever since NT, Though in Vista there were some smart heuristic improvements to

ensure that some processes, such as the disk defragmenter, are at a lower priority to not interfere with foreground processes.

In case of Linux, the default scheduler is the Completely Fair Scheduler. But you can select from several other, alternative schedulers when you compile the kernel and/or tweak their parameters, even at runtime. The design is quite complex and, in my view, not suitable for RTOS.

A common technique in real-time systems is rate-monotonic scheduling, because it has strong guarantees if certain assumptions hold (e.g. static task priorities and fixed execution time and rate).

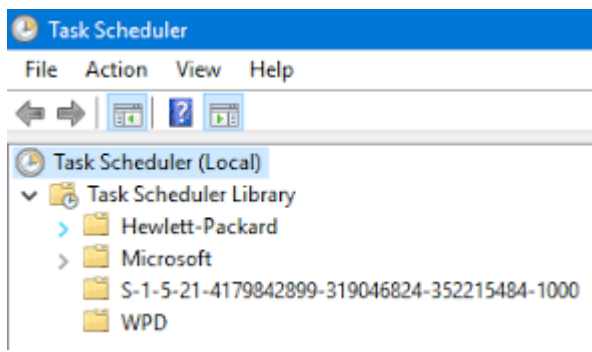


Fig. 1 Task Scheduler

Problem Statement:

1. Problem Defined:

Reducing the average waiting time and turnaround time to enhance the performance of Round robin algorithm.

Limitation:

Large waiting time: If a process gets executed for a fixed time quantum and the remaining burst time is only slightly greater than the time slice it will have its context saved and will go into the ready queue again and would have to wait for a long time to be completely executed even though only a short amount of its burst time is remaining. If the time quantum is too small then the algorithm becomes time-consuming. Too many context switches depending on the burst time and the time quantum.

2. Problem Defined:

Eliminating the problem of starvation and also reducing the average turnaround time and average waiting time.

Limitation:

In preemptive priority scheduling algorithm, the lower priority tasks may never execute, and this is called starvation. These lower priority processes have a high waiting time.

3. Problem Defined:

An Improved Time Varying Quantum Round Robin CPU Scheduling Algorithm.

Limitation:

Round Robin (RR) algorithm has been regarded as impartial algorithm this is because it uses the same quantum time for all processes on the queue irrespective of their burst time. Questions on optimal time quantum to be used by RR and Shortest Job First (SJF) have also been on the mind of researchers. This is because time quantum determines the performance of the algorithm. If the time quantum assigned is relatively high, it may lead to

First Come First Serve while high context switch is obtained with low time quantum.

4. Problem Defined:

Priority Preemptive scheduling algorithm leads to the problem of starvation which happens when processes with lower priority are not given any chance of CPU utilization due to continuous CPU usage by processes with higher priorities.

Limitation:

Priority Preemptive scheduling algorithm works by executing the process which has the highest priority first, even when a new process with higher priority arrives it stops the execution of the currently running process and gives chance to the newly arrived higher priority process. However, it undergoes a severe problem of high wait in g time for low priority processes. This happens when higher priority processes with large burst times are executed first and continues CPU utilization instead of giving chance to processes with lower priority even after having low burst time.

5. Problem Defined:

Modifying Round robin algorithm to prevent longer waiting time and response time and also to increase system throughput.

Limitation:

This is not a new algorithm just a previously existing algorithm has been modified. Round robin gives equal priority to all the processes which results in the shorter processes being executed in a less efficient manner. Consequently, the waiting time and response time increases.

6. Problem Defined:

Modifying the round robin algorithm to change the time of execution for the processes. Improving round robin to decrease the waiting time and turnaround time.

Limitation:

As round robin is the most commonly used algorithm, no new algorithm can be found using round robin. We can only modify the existing one.

Introduction to proposed model:

The proposed algorithm significantly reduces the average waiting time and average turn around time in comparison to the existing algorithms. However it also reduces the chances of starvation significantly, minimizes or eliminates the time lag caused due to context switching and is not biased towards particular processes.

The modifying Round robin algorithm will prevent longer waiting time and response time and also to increase system throughput. The improved round robin algorithm is slightly different than the existing round robin. A process is selected from ready queue and allotted to the CPU to be processed for a time quantum of 1 second. After executing for the given time quantum, the algorithm checks the remaining CPU burst time of the current process. If the CPU burst time is less than the time quantum then the current process is allowed to be executed completely and then it will be removed from the ready queue and the scheduler will proceed to the next process. Otherwise, if the remaining CPU burst time of the current process is more than the time quantum, the process' execution state will be saved and then it will be put in the end of the ready queue after which the scheduler will move on to select the next process.

Proposed Solution:

Design–PSEUDOCODE:

1. START
2. INPUT NO OF PROCESSES, THEIR ARRIVAL TIME, BURST TIME, PRIORITY
3. INPUT THE DATA
4. CALCULATE MODIFIED PRIORITY (MEAN OF BURST TIME AND PRIORITY INPUT BY USER).

5. ACCORDING TO THE ARRIVAL TIME,PUT THE PROCESS IN THE READY QUEUE.
6. CONVERT THE MPT(MODIFIED PRIORITY) TO NEAREST SMALLEST INTEGER
LOAD THE PROCESSES IN READY QUEUE ACCORDING TO THEIR ARRIVAL
TIME
7. IF(SAME ARRIVAL TIME) && IF(SAME MPT).
8. COMPARE ACCORDING TO BURST TIME //To overcome problem of STARVATION
9. AND EXECUTE.
10. ELSE
11. EXECUTE THE PROCESS WITH MIN NUMBER IE MAX PRIORITY.
12. ELSE
13. EXECUTE SIMPLY
14. IF(NOT SAME MPT AND SAME ARRIVAL TIME)
15. COMPARE ACCORDING TO MPT//To overcome problem of STARVATION
16. AND EXECUTE THE MAX PRIORITY PROCESS

Components of the Model:

1. Scheduler:

The scheduler is the component of the kernel that selects which process to run next.

2. CPU:

It executes the processes which are allocated to it.

3. Real-life Systems:

A real time system is the one whose applications are mission-critical, where real-time tasks should be scheduled to be completed before their deadlines.

Block Diagram:

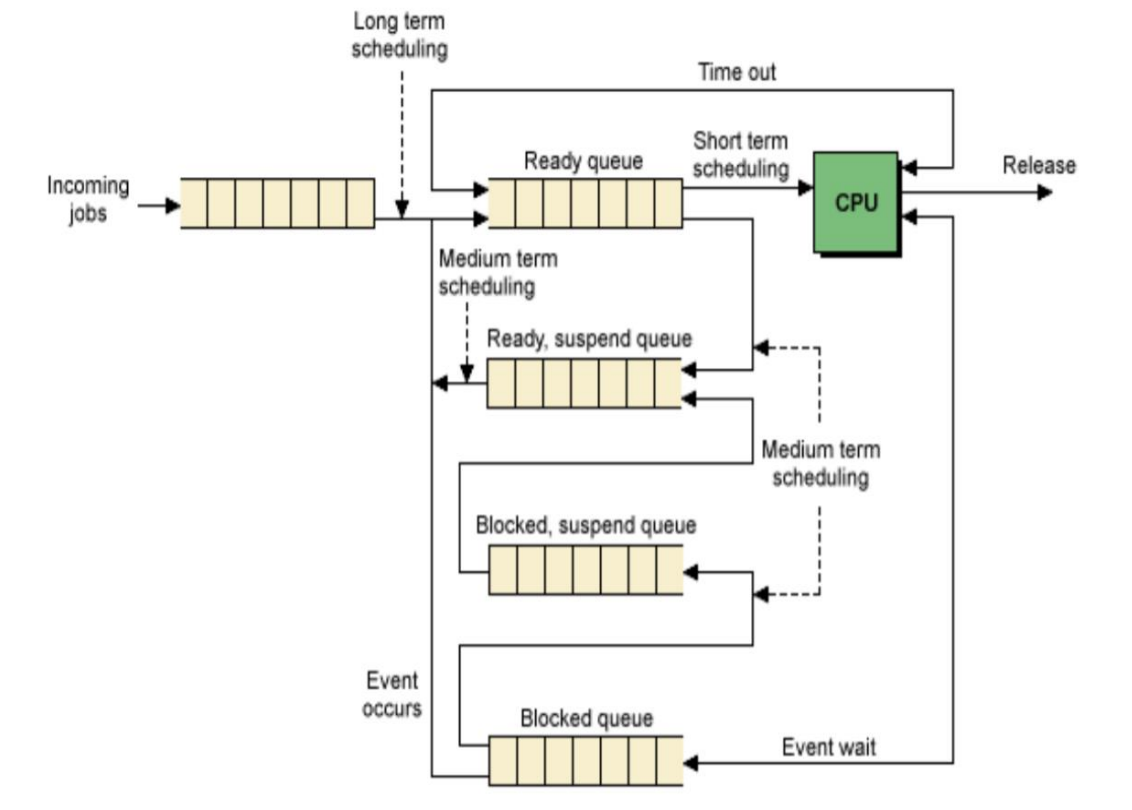


Fig. 2 Block Diagram

Explanation:

Our approach is not to change the philosophy of simple round robin algorithm, but we add one more step in this algorithm by which we decide the priority of processes which comes in a single time unit. Simply we can say our proposed Round Robin algorithm is a mesh up of Simple Round Robin algorithm and the Priority Scheduling Algorithm.

The long-term scheduler or job scheduler selects processes from the job pool and loads them into memory for execution. The short-term scheduler, or CPU scheduler selects from among the processes that are ready to execute, and allocates CPU to one of them. The medium-term scheduler removes processes from memory and reduces the degree of multiprogramming results in the scheme of swapping.

Ready queue – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

Turnaround Time: Amount of time to execute a particular process.

Waiting Time: Amount of time a process has been waiting in the ready queue.

Response Time: Amount of time it takes from when a request was submitted until the first

response is produced, not output (for time-sharing environment).

Description of the Components:

1. Real-Time Systems:

A real time system is the one whose applications are mission-critical, where real-time tasks should be scheduled to be completed before their deadlines. Most real-time systems control unpredictable environments and may need operating systems that can handle unknown and changing tasks. So, not only a dynamic task scheduling is required, but both system hardware and software must adapt to unforeseen configurations. There are two main types of real-time systems: Hard Real-Time System, Firm or Soft Real-Time System. In Hard Real-Time System, it requires that fixed deadlines must be met otherwise disastrous situation may arise whereas in Soft Real-Time System, missing an occasional deadline is undesirable, but nevertheless tolerable. System in which performance is degraded but not destroyed by failure to meet response time constraints is called soft real time systems.

2. Scheduler:

The scheduler is the component of the kernel that selects which process to run next. Operating systems may feature up to three distinct types of schedulers, a long term scheduler, a mid-term or medium term scheduler and a short-term scheduler. The long-term scheduler or job scheduler selects processes from the job pool and loads them into memory for execution. The short-term scheduler, or CPU scheduler selects from among the processes that are ready to execute, and allocates CPU to one of them. The medium term scheduler removes processes from memory and reduces the degree of multiprogramming results.

3. System Designer:

It must consider a variety of factors in designing a scheduling algorithm, such as type of systems used and what are user's needs. Depending on the system, the user and designer might expect the scheduler to :

Maximize throughput: A scheduling algorithm should be capable of servicing the maximum number of processes per unit of time.

Avoid indefinite blocking or starvation: A process should not wait for unbounded time before or while process service.

Minimize overhead: Overhead causes wastage of resources. But when we use system

resources effectively, then overall system performance improves greatly.

Enforcement of priorities: if system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.

Achieve balance between response and utilization: The scheduling mechanism should keep resources of system busy.

Favor processes exhibits desirable behavior.

x Degrade gracefully under heavy load.

Working Mechanism:

1. CONTEXT SWITCH AFTER EVERY 1 TIME QUANTAM AND SAVE THE STATE OF PRIOR EXECUTION IN THE TEMPORARY FILE
2. CREATE NEW FILE FOR EVERY PROCESS AT THEIR RESPONSE TIME AND EDIT THEM WHEN THEY COME ACCESS AGAIN AFTER PRE-EMPTION. (NO LOCKS APPLIED)
3. LOCK THE RUNNING PROCESS FILE AND CREATE NEW FILE FOR EVERY NEW PROCESS. (LOCKS ARE GIVEN). PERMISSION WILL ONLY BE ALLOWED TO THE PROCESS WHO LOCKED THE FILE.
4. AFTER PROCESS GET ITS ALL RESOURCES MOVE THAT FILE TO CACHE (ACCORDING TO LRU ALGO) AND TERMINATE (EXIT)
5. RESOURCE FREE
6. EVERY PROCESS EXECUTED
7. ALL RESOURCES ARE OCCUPIED BACK BY OS
8. EXECUTION SUCCESSFUL
9. CALCULATE COMPLETION TIME, TURN AROUND TIME (COMPLETION TIME - ARRIVAL TIME), WAITING TIME (TURN AROUND TIME - BURST TIME)
10. STOP
11. END

Theoretical Explanation:

Process	AT	PT	BT	MPT
P1	0	1	22	11
P2	1	2	10	6
P3	2	4	5	4
P4	4	5	5	5
P5	5	3	6	4
P6	0	2	12	7
P7	2	6	3	4

Table 1 Calculation of MPT for 7 processes

Here Modified Priority = $(PT+BT)/2 = (22+1)/2 = 11$

Here less value = maximum priority

If same value of MPT, then accommodate on basis of BT to avoid effect convoy effect.

P6	P2	P7	P7	P7	P3	P3	P5	P4	P2	P6	P1	Complete
0	1	2	3	4	5	6	10	16	21	30	41	63

Table 2 Ready Queue

At t=0:

P1 and P6 are there in ready queue. Since MPT of P6 is less, it will go first, At t=1:

P2 also enters and has MPT =6

Since P1, P2, and P6 are there with MPT = 1, 6, 7 respectively, P2 will be executed.

At t=2:

P3 and P7 arrives. So MPT are 11, 6, 4, 7, 4

Since MPT of P3 and P7 are same, we will see BT and burst time of P7 is less, hence it will get executed first.

No process at t=3, so P7 will execute again for one more time.

Now at t=4, P4 also arrives. So P7 will be executed again.

At t=5:

P5 arrives. MPT are 11,6, 4, 5, 4, 7. Since MPT is same between P3 and P5, we check wrt to BT so P3 will be executed.

(Also overcomes the problem of starvation between P3 and P5)

Now, P5 will execute followed by P4, P2, P6 and P1 respectively.

Proce ss	C T	TAT=CT -AT	WT=TAT -BT
P1	6 3	63	41
P2	3 0	29	19
P3	1 0	8	3
P4	2 1	17	12
P5	1 6	11	5
P6	4 1	41	29
P7	5	3	0
SUM =		24.57	15.57

Table 3 Calculation of Turn Around Time and Waiting Time using Modified Scheduling Algorithm

Context Switches = 8

Flow Diagram:

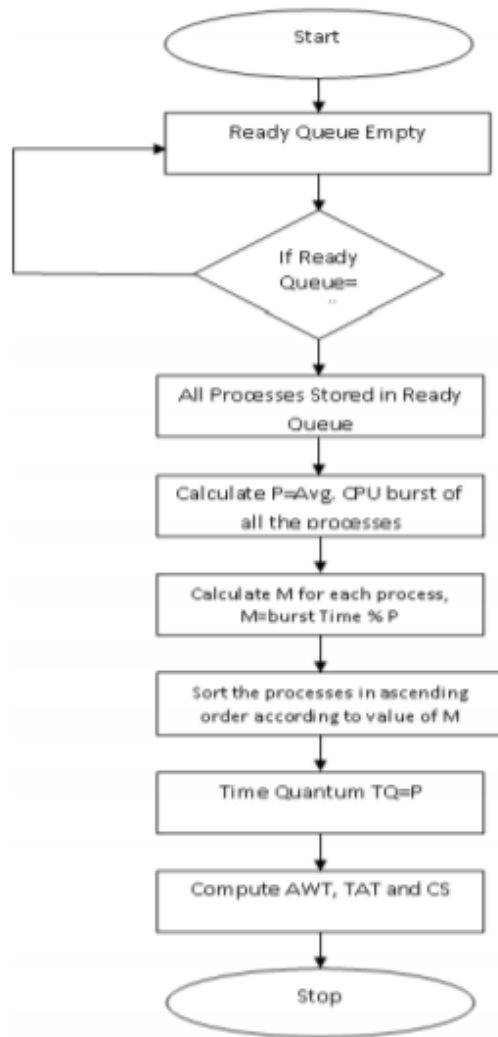


Fig. 3 Flow Chart of the Modified algorithm

Assumptions:

In our new proposed algorithm, we assume the following points. We already have number of processes and their respective burst time.

Input Parameters: The number of Processes which we want to execute, Burst time of processes, priority time and arrival time.

Result Parameters: Average Waiting Time and Average Turnaround Time.

Code:

```
#include<cstdlib>
```

```
#include<iostream>
```

```

using namespace std;

int main()

{

int a[10],b[10],x[10];

int waiting[10],turnaround[10],completion[10],p[10],mp[10],p1[10];

int i,j,smallest,count=0,time,n;

double avg=0,tt=0,end;

cout<<"\nEnter the number of Processes: "; cin>>n;

for(i=0;i<n;i++)

{

cout<<"\nEnter arrival time of process: "; cin>>a[i];

}

for(i=0;i<n;i++)

{

cout<<"\nEnter burst time of process: "; cin>>b[i];

}

for(i=0;i<n;i++)

{

cout<<"\nEnter priority of process: "; cin>>p1[i];

}

for(i=0;i<n;i++)

{

mp[i]=(b[i]+p1[i])/2;

mp[i]=abs(10-mp[i]);

```

```

}

for(i=0; i<n; i++) x[i]=b[i];

p[9]=-1;

for(time=0; count!=n; time++)

{

smallest=9;

for(i=0; i<n; i++)

{

if(a[i]<=time && mp[i]>mp[smallest] && b[i]>0 ) smallest=i;

}

b[smallest]--;

if(b[smallest]==0)

{

count++; end=time+1;

completion[smallest] = end;

waiting[smallest] = end - a[smallest] - x[smallest]; turnaround[smallest] = end - a[smallest];

}

}

cout<<"Process"<<"\t"<<"burst-time"<<"\t"<<"arrival-time"

<<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<<"\t"<<"completion-

time"<<"\t"<<"Priority"<<"\t"<<"Modified-Priority"<<endl;

for(i=0; i<n; i++)

{

cout<<"p"<<i+1<<"\t"<<x[i]<<"\t"<<a[i]<<"\t"<<waiting[i]<<"\t"<<t

urnaround[i]<<"\t"<<completion[i]<<"\t"<<p[i]<<"\t"<<mp[i]<<endl;

```

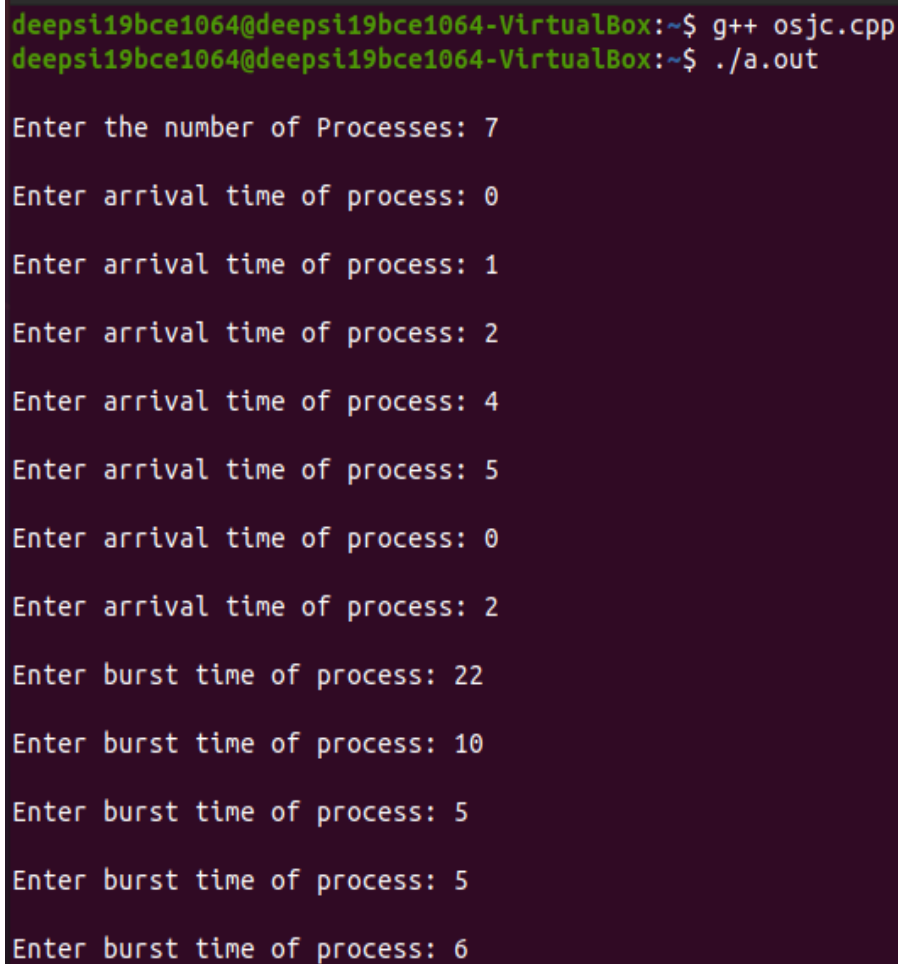
```
avg = avg + waiting[i]; tt = tt + turnaround[i];

}

cout<<"\n\nAverage waiting time ="<<avg/n; cout<<" Average Turnaround time
="<<tt/n<<endl;

}
```

Output Screenshots:



```
deepsil9bce1064@deepsil9bce1064-VirtualBox:~$ g++ osjc.cpp
deepsil9bce1064@deepsil9bce1064-VirtualBox:~$ ./a.out

Enter the number of Processes: 7
Enter arrival time of process: 0
Enter arrival time of process: 1
Enter arrival time of process: 2
Enter arrival time of process: 4
Enter arrival time of process: 5
Enter arrival time of process: 0
Enter arrival time of process: 2
Enter burst time of process: 22
Enter burst time of process: 10
Enter burst time of process: 5
Enter burst time of process: 5
Enter burst time of process: 6
```

Fig. 4.1 Output of Modified Algorithm in Virtual Box

```
Enter burst time of process: 6
Enter burst time of process: 12
Enter burst time of process: 3
Enter priority of process: 1
Enter priority of process: 2
Enter priority of process: 4
Enter priority of process: 5
Enter priority of process: 3
Enter priority of process: 2
Enter priority of process: 6
Process burst-time      arrival-time      waiting-time      turnaround-time complet
ion- time      Priority      Modified-Priority
p1      22      0      41      63      63      -
274169223      1
p2      10      1      19      29      30      2
```

Fig. 4.2 Output of Modified Algorithm in Virtual Box

```
Average waiting time =16.2857 Average Turnaround time =25.2857
deepsi19bce1064@deepsi19bce1064-VirtualBox:~$
```

Fig. 4.3 Output of Modified Algorithm in Virtual Box


```

Enter burst time of process: 10
Enter burst time of process: 5
Enter burst time of process: 5
Enter burst time of process: 6
Enter burst time of process: 12
Enter burst time of process: 3
Enter priority of process: 1
Enter priority of process: 2
Enter priority of process: 4
Enter priority of process: 5
Enter priority of process: 3
Enter priority of process: 2
Enter priority of process: 6
Process burst-time arrival-time waiting-time turnaround-time completion-time Priority Modified-Priority
p1 22 0 41 63 63 0 1
p2 10 1 19 29 30 0 4
p3 5 2 0 5 7 0 6
p4 5 4 12 17 21 0 5
p5 6 5 2 8 13 0 6
p6 12 0 29 41 41 0 3
p7 3 2 11 14 16 0 6

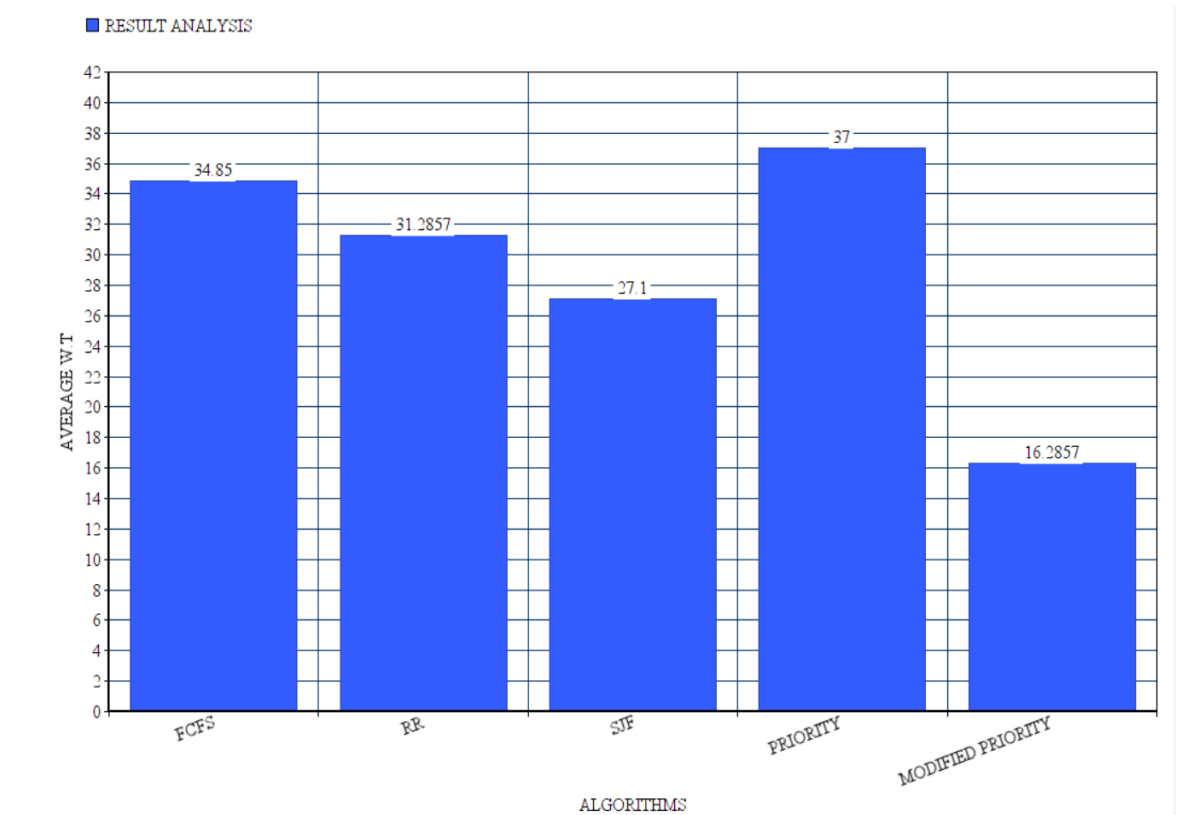
Average waiting time =16.2857 Average Turnaround time =25.2857
>

```

Fig. 4.4 Output of Modified Algorithm in Online compiler

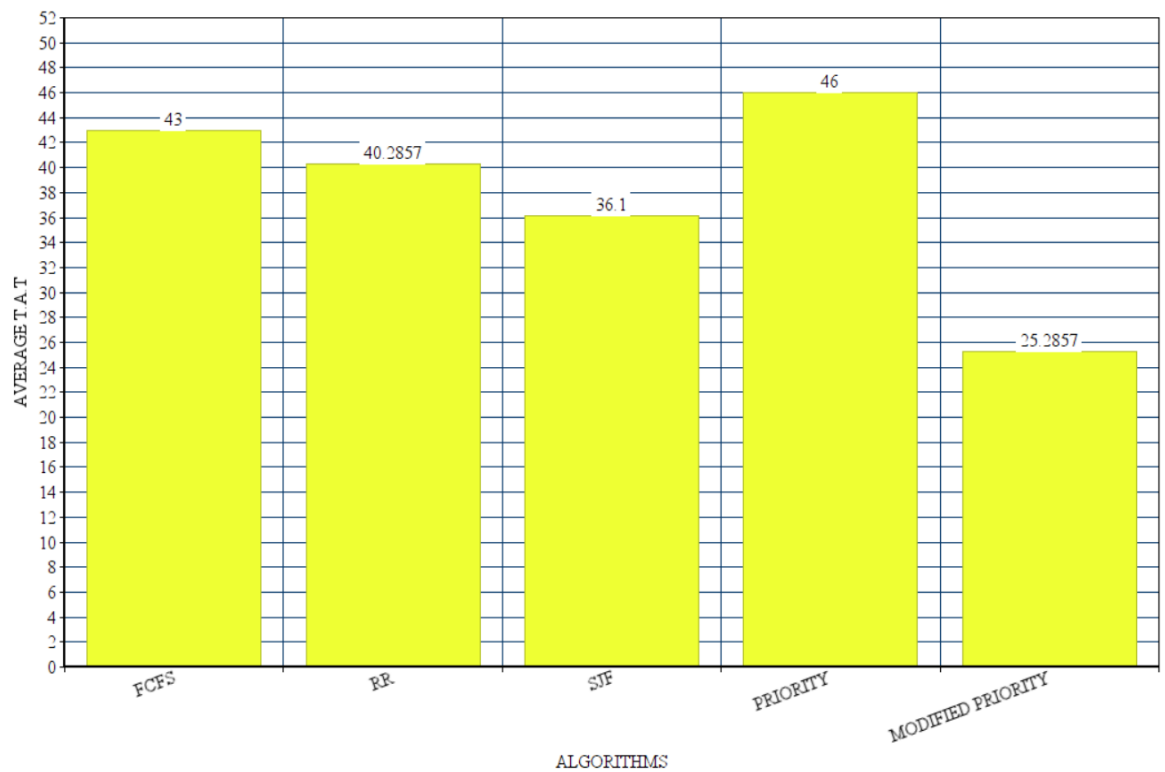
Result Analysis:

1. COMPARISON OF AVERAGE W.T. :



Graph 1 Graph comparison of Average Waiting Time of FCFS, RR, SJF, and PRIORITY with Modified Priority

2. COMPARISON OF AVERAGE T.A.T:



Graph 2 Graph comparison of Average Turn around Time of FCFS, RR, SJF, and PRIORITY with Modified Priority

Conclusion:

The proposed algorithm significantly reduces the average waiting time and average turnaround time in comparison to FCFS, SJF, Priority scheduling and Round robin.

The proposed algorithm reduces the chances of starvation significantly, eliminates the convoy effect, minimizes or eliminates the time lag caused due to context switching and is not biased towards particular processes.

ADVANTAGES:

1. REDUCED CHANCE OF STARVATION
2. HIGH RESPONSE TIME
3. NO CHANCE OF CONVOY EFFECT/SO SMALLER PROCESS DON'T HAVE TO WAIT FOR THEIR CHANCE FOR LONG
4. PROPER CHANCE TO EVERY PROCESS/EQUAL OPPORTUNITY TO EVERY PROCESS
5. EASY TO IMPLEMENT

- 6. SUPPORTABLE IN REAL LIFE SYSTEM**
- 7. ALSO PROVIDE SUPPORT TO THE HIGH PRIORITY TASK (IF THEY ARRIVE, THEY WILL BE EXECUTED FIRST)**
- 8. LESS NUMBER OF CONTEXT SWITCHES DUE TO WHICH NO TIME LAG FOR SWITCHING**
- 9. INCREASE EFFICIENCY AND MAKE CPU BUSY**
- 10. NOT BIASED COMPLETELY AND RESPECT PRIORITY OF PROCESSES SET BY USER.**

Future Works:

1. We plan to optimize our algorithm more as we get some more time in the upcoming month.
2. We will also be working on reducing space complexity of our algorithm thus requiring fewer resources.
3. Our next target then is to publish this Research Paper under the guidance of Sir.

References:

1. Abdulraza, Abdulrahim, Salisu Aliyu, Ahmad M Mustapha, Saleh E Abdullahi, “ An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm,” International Journal of Advanced Research in Computer Science and Software Engineering Volume 4, Issue 2, February 2014 ISSN: 2277 128X.
2. Manish Kumar Mishra and Abdul Kadir khan, “An Improved Round Robin CPU scheduling algorithm,” Journal of Global Research in Computer Science Volume 3, No. 6, published in June 2012.
3. Ishwari Singh Rajput and Deepa Gupta , “A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems”, International Journal of Innovations in Engineering and Technology(IJiet), published in 2012.
4. Saqib Ul Sabha, A Novel And Efficient Round Robin Algorithm With Intelligent Time Slice And Shortest Remaining Time First, Materials Today: Proceedings, published in 2018.

5. Kunal Chandiramani ,Rishabh Verma and M.Sivagami, “A Modified Priority Preemptive Algorithm for CPU Scheduling”,Procedia Computer Science Vol.165,published in 2019.
6. U. Saleem and M.Y. Javed, “Simulation of CPU Scheduling Algorithms”,TENCON Proceedings , published in 2000.
7. Supriya Raheja, Reena Dadhich and Smita Rajpal,”An Optimum Time Quantum Using Linguistic Synthesis For Round Robin Scheduling Algorithm”,International Journal of Soft Computing, published in 2012.
8. Rami J. Matarneh, “Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes”,American Journal of Applied Sciences 6, published in 2009.
9. Himanshi Arora, Bagish Goel, Deepansh Arora and Parita Jain,”An Improved CPU Scheduling Algorithm”, International Journal of Applied Information Systems (IJ AIS) Vol. 6, published in 2013.
10. H.Singh, Sachin Kumar Sarin, Arushi Patel and Supriya Sen,”Performance Analysis of Hybrid CPU Scheduling Algorithm in Multi-tasking Environment”, Smart and Innovative Trends in Next Generation Computing Technologies, published in 2018.

