VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled **"CPU Scheduling with dynamic quantum time"** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore-14 towards the partial fulfillment of the requirements for the course CSE4011- Virtualization is a record of bonafide work carried out by me under the supervision of **Prof. Jothi K R.** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for any other course or purpose of this institute or of any other institute or university.

Signature

Name:LAKSHIT MANISH SANGHRAJKA

Reg.No: 19BCB0031

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled "**CPU SCHEDULING WITH DYNAMIC QUANTUM TIME**" is prepared and submitted by **LAKSHIT MANISH SANGHRAJKA (Register No: 19BCB0031),** has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the course CSE4011-Virtualization in Vellore Institute of Technology, Vellore-14, India.

Guide
(Name & Signature)

ACKNOWLEDGEMENT

The project "**CPU Scheduling with dynamic quantum time"** was made possible because of inestimable inputs from everyone involved, directly or indirectly. I would first like to thank my guide, **Prof. Jothi K R,** who was highly instrumental in providing not only a required and innovative base for the project but also crucial and constructive inputs that helped make my final product. My guide has helped me perform research in the specified area and improve my understanding in the area of web development and internet of things and I am very thankful for her support all throughout the project.

I would also like to acknowledge the role of the HOD, **Dr. V Santhi** who was instrumental in keeping me updated with all necessary formalities and posting all the required formats and document templates through the mail, which I was glad to have had.

It would be no exaggeration to say that the Dean of SCOPE, **Prof Saravanan R**, was always available to clarify any queries and clear the doubts I had during the course of my project.

Finally, I would like to thank Vellore Institute of Technology, for providing me with a flexible choice and execution of the project and for supporting my research and execution related to the project.

TABLE OF CONTENT

1 ABSTRACT: -

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.

➢ 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.

➢ A time quantum is set and the processes utilize the resources available for that time quantum only. Once the time quantum is crosses, the control is passed to the next process.

➢ The purpose of this project is to introduce an optimised variant to the round robin scheduling algorithm. Every algorithm works in its own way and has its own merits and demerits.

➢ A new CPU scheduling algorithm has been proposed, named as DABRR (Dynamic Average Burst Round Robin). This uses dynamic time quantum instead of static time quantum which is used in Round Robin.

➢ The proposed algorithm overcomes the short falls of the existing scheduling algorithms in terms of waiting time, turnaround time, throughput and number of context switches.

➢ The algorithm is pre-emptive and works based on the priority of the associated processes.

➢ The performance of the proposed algorithm is experimentally compared with traditional RR and some existing variants of RR.

2 INTRODUCTION :

Existing Algorithm :

- The traditional Round-Robin is a scheduling algorithm where a time quantum is decided and time slices are assigned to each process in equal proportions and in a circular approach.

- It is a simple algorithm and the development is effortless, hence is used in many applications.

- The above said algorithm is a concept of operating system which proves to be starvation-free and can be used to perform several scheduling tasks, such as engaging data packets in any networking scenario.

- The mean Waiting Time in round robin scheduling algorithm proves to be longer than the others many a times.

- Each process is given a fixed time period to execute which is known time slice. Once the process has executed for the decided time quantum, it is preempted and the control is given to the next process for computation. This happens in a circular fashion.

- Once we have waiting times, we can compute turnaround time tat[i] of a process as sum of waiting and burst times, i.e., wt[i] + bt[i].

Our attempt is to modify the existing algorithm and provide an optimum solution for the problem identified .

2.1LITERATURE SURVEY

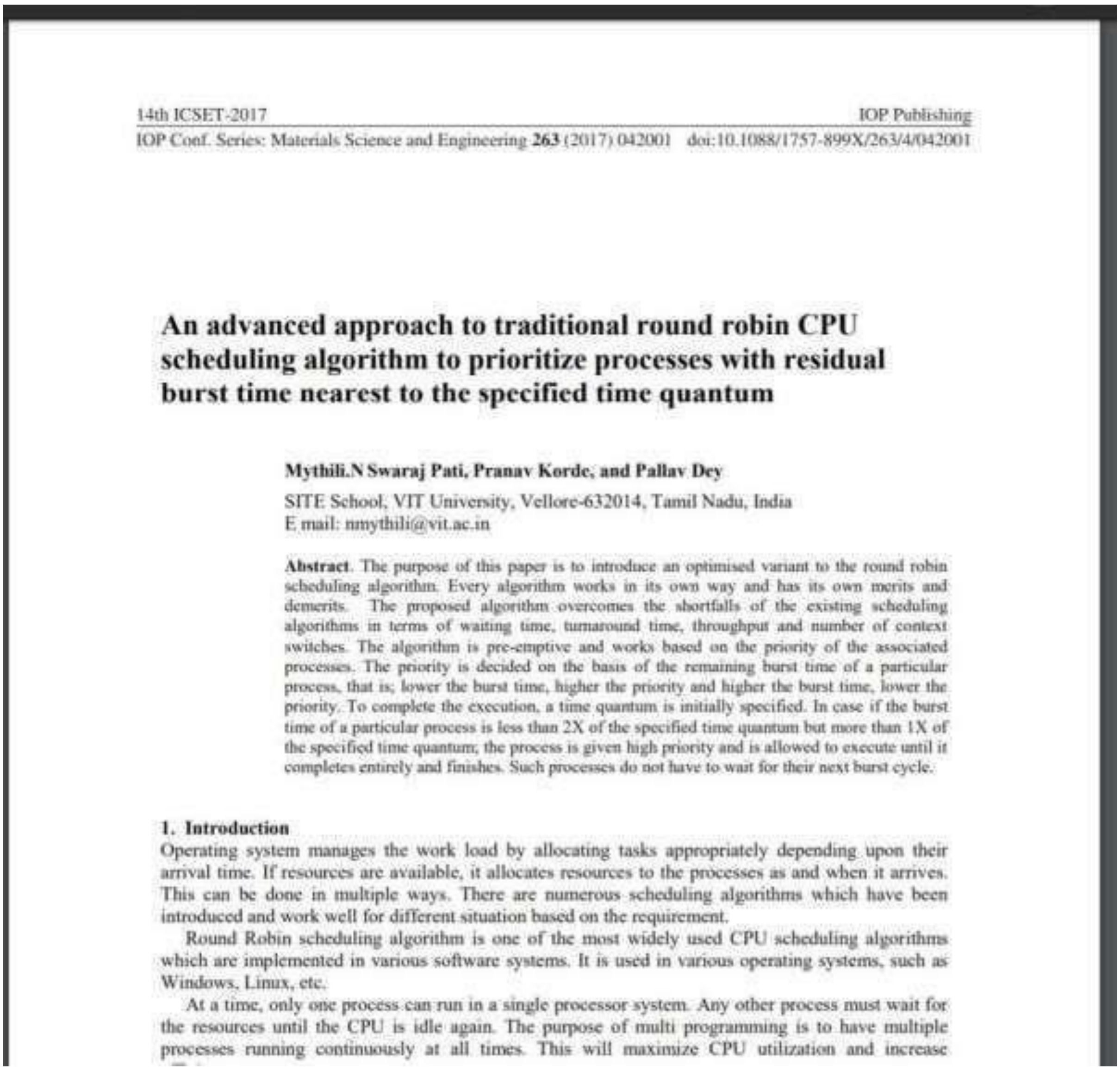
- An advanced approach to traditional round robin CPU scheduling algorithm to prioritize processes with residual burst time nearest to the specified time quantum.

- To cite this article:

➢ By Amar Ranjan Dash, Sanjay Kumar Samantra, Sandipta Kumar Sahu

https://arxiv.org/abs/1605.00362

Mythili N. Swaraj Pati et al 2017 IOP Conf.Ser.:
Mater. Sci. Eng.263 042001



14th ICSET-2017 IOP Publishing
IOP Conf. Series: Materials Science and Engineering **263** (2017) 042001 doi:10.1088/1757-899X/263/4/042001

An advanced approach to traditional round robin CPU scheduling algorithm to prioritize processes with residual burst time nearest to the specified time quantum

Mythili.N Swaraj Pati, Pranav Korde, and Pallav Dey

SITE School, VIT University, Vellore-632014, Tamil Nadu, India
E mail: nmythili@vit.ac.in

Abstract. The purpose of this paper is to introduce an optimised variant to the round robin scheduling algorithm. Every algorithm works in its own way and has its own merits and demerits. The proposed algorithm overcomes the shortfalls of the existing scheduling algorithms in terms of waiting time, turnaround time, throughput and number of context switches. The algorithm is pre-emptive and works based on the priority of the associated processes. The priority is decided on the basis of the remaining burst time of a particular process, that is; lower the burst time, higher the priority and higher the burst time, lower the priority. To complete the execution, a time quantum is initially specified. In case if the burst time of a particular process is less than 2X of the specified time quantum but more than 1X of the specified time quantum; the process is given high priority and is allowed to execute until it completes entirely and finishes. Such processes do not have to wait for their next burst cycle.

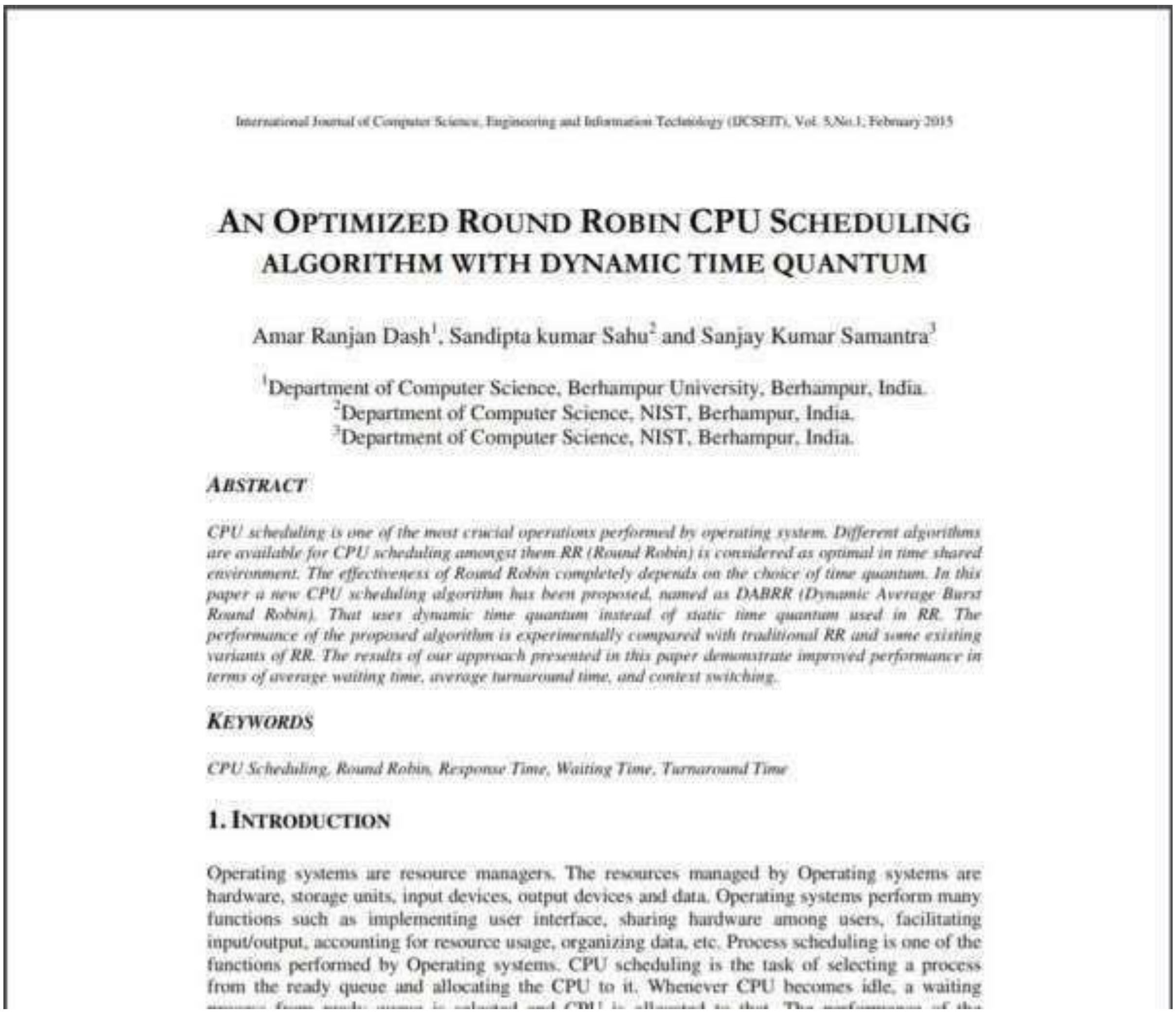
1. Introduction

Operating system manages the work load by allocating tasks appropriately depending upon their arrival time. If resources are available, it allocates resources to the processes as and when it arrives. This can be done in multiple ways. There are numerous scheduling algorithms which have been introduced and work well for different situation based on the requirement.

Round Robin scheduling algorithm is one of the most widely used CPU scheduling algorithms which are implemented in various software systems. It is used in various operating systems, such as Windows, Linux, etc.

At a time, only one process can run in a single processor system. Any other process must wait for the resources until the CPU is idle again. The purpose of multi programming is to have multiple processes running continuously at all times. This will maximize CPU utilization and increase

- An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum



International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol. 5,No.1, February 2015

AN OPTIMIZED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH DYNAMIC TIME QUANTUM

Amar Ranjan Dash[1], Sandipta kumar Sahu[2] and Sanjay Kumar Samantra[3]

[1]Department of Computer Science, Berhampur University, Berhampur, India.
[2]Department of Computer Science, NIST, Berhampur, India.
[3]Department of Computer Science, NIST, Berhampur, India.

ABSTRACT

CPU scheduling is one of the most crucial operations performed by operating system. Different algorithms are available for CPU scheduling amongst them RR (Round Robin) is considered as optimal in time shared environment. The effectiveness of Round Robin completely depends on the choice of time quantum. In this paper a new CPU scheduling algorithm has been proposed, named as DABRR (Dynamic Average Burst Round Robin). That uses dynamic time quantum instead of static time quantum used in RR. The performance of the proposed algorithm is experimentally compared with traditional RR and some existing variants of RR. The results of our approach presented in this paper demonstrate improved performance in terms of average waiting time, average turnaround time, and context switching.

KEYWORDS

CPU Scheduling, Round Robin, Response Time, Waiting Time, Turnaround Time

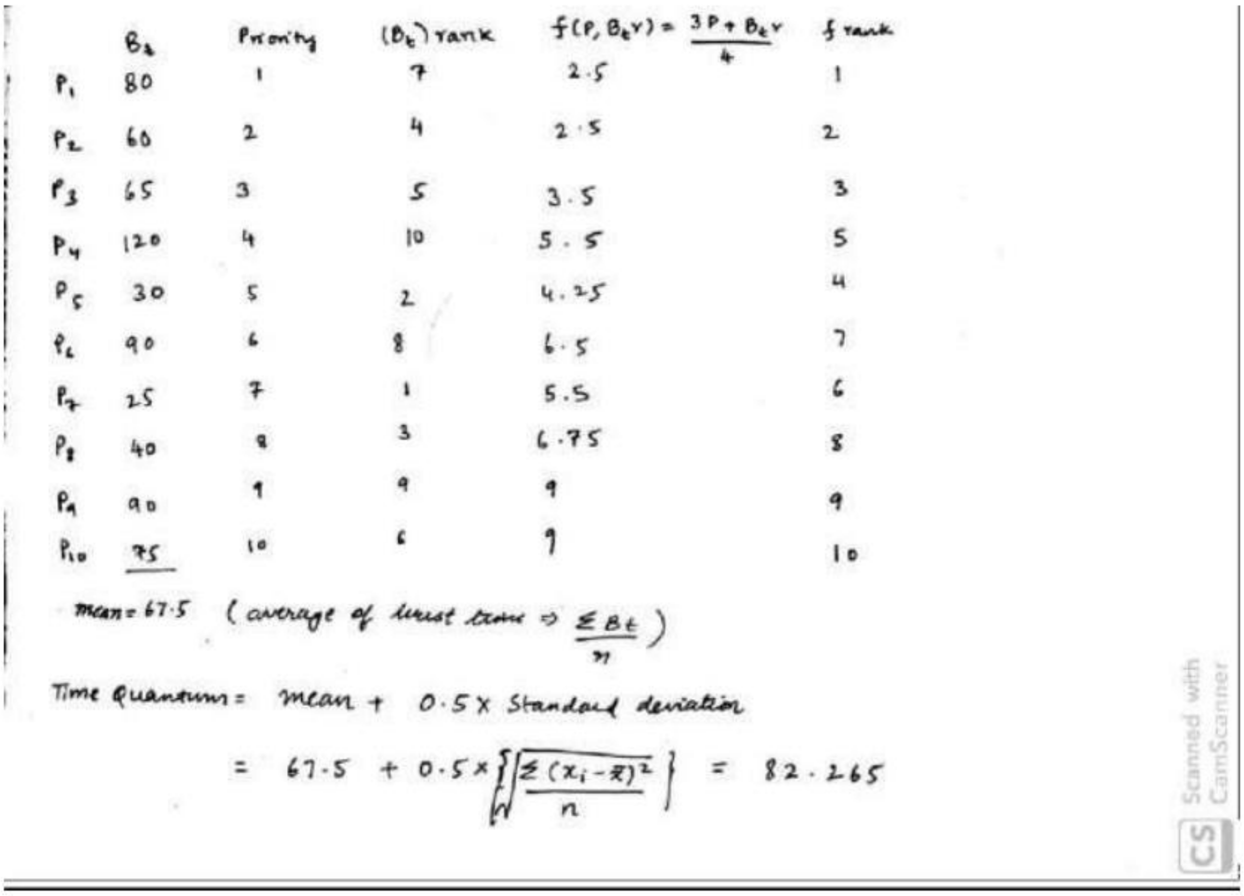
1. INTRODUCTION

Operating systems are resource managers. The resources managed by Operating systems are hardware, storage units, input devices, output devices and data. Operating systems perform many functions such as implementing user interface, sharing hardware among users, facilitating input/output, accounting for resource usage, organizing data, etc. Process scheduling is one of the functions performed by Operating systems. CPU scheduling is the task of selecting a process from the ready queue and allocating the CPU to it. Whenever CPU becomes idle, a waiting

2.2 Design of the proposed system (Algorithm):

2. Take the burst time and priority of the processes and calculate the new priority rank by using 50% of priority and 50% of rank according to burst time for optimum results.Also the weightage that has to be given to priority can be user defined.
3. Arrange the processes according to the calculated rank.
4. Now we calculate the smart time quanta(STQ) by the formula:
5. Mean of all burst time + 0.5*SD of burst time
6. Now start executing the processes by round robin algorithm with STQ time quanta.
7. If a process is not finished in one go, we add the process to the other wait queue.
8. Now after all the processes are over in our executing queue, we execute all the processes in the waiting queue.

EXAMPLE :-



	B_t	Priority	(B_t) rank	$f(P, B_tr) = \frac{3P + B_tr}{4}$	f rank
P_1	80	1	7	2.5	1
P_2	60	2	4	2.5	2
P_3	65	3	5	3.5	3
P_4	120	4	10	5.5	5
P_5	30	5	2	4.25	4
P_6	90	6	8	6.5	7
P_7	25	7	1	5.5	6
P_8	40	8	3	6.75	8
P_9	90	9	9	9	9
P_{10}	75	10	6	9	10

mean = 67.5 (average of burst time $\Rightarrow \frac{\Sigma B_t}{n}$)

Time Quantum = mean + 0.5 × Standard deviation

$$= 67.5 + 0.5 \times \left\{ \sqrt{\frac{\Sigma (x_i - \bar{x})^2}{n}} \right\} = 82.265$$

3.1 IMPLEMENTATION

CODE:

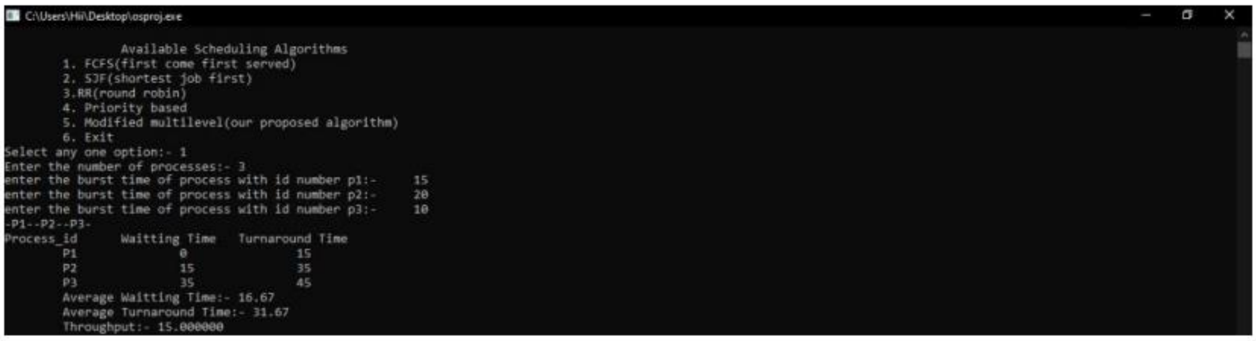
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int i,j;
struct process{
int process_id,burst_time,priority,shortness_rank,remaining_time;
float new_rank;
int waitting_time,turnaround_time;
};
int sort(struct process *p,int n,int type){
struct process temp;
if(type==1){
for(i=0;i<n-1;i++)
for(j=0;j<n-i-1;j++)
if(p[j].new_rank-p[j+1].new_rank>0){/// our new algo
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
else if(type==2){
for(i=0;i<n-1;i++)
for(j=0;j<n-1-i;j++)
if(p[j].burst_time>p[j+1].burst_time){/// sjf
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
else if(type==3){
for(i=0;i<n-1;i++)
for(j=0;j<n-i-1;j++)
if(p[j].priority>p[j+1].priority){/// priority
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
else if(type==4){
for(i=0;i<n-1;i++)
for(j=0;j<n-i-1;j++)
if(p[j].process_id>p[j+1].process_id){/// printing
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
return(1);
}
int print_pretty(struct process *p,int n,int t){
float avg_wt=0,avg_tt=0;
sort(p,n,4);
printf("\nProcess_id\tWaitting Time\tTurnaround Time\n");
for(i=0;i<n;i++){
avg_wt+=p[i].waitting_time;
avg_tt+=p[i].turnaround_time;
printf("\tP%d\t\t%d\t\t%d\n",p[i].process_id,p[i].waitting_time,p[i].turnaround_time);
}
avg_wt=(float)avg_wt/n;
avg_tt=(float)avg_tt/n;
printf("\tAverage Waitting Time:- %.2f\n",avg_wt);
printf("\tAverage Turnaround Time:- %.2f\n",avg_tt);
printf("\tThroughput:- %f\n",(float)t/n);
}
int returnIndex(struct process *p,int n,int a){
for(i=0;i<n;i++){
if(p[i].process_id==a)
return(i);
}
return(-1);
```

```
}
int fcfs(struct process *que1,int n){
int total_time=0;
for(i=0;i<n;i++){
que1[i].waitting_time=total_time;
total_time+=que1[i].burst_time;
que1[i].turnaround_time=total_time;
printf("-P%d-",que1[i].process_id);
}
print_pretty(que1,n,total_time);
return(1);
}
int sjf(struct process *que1,int n){
sort(que1,n,2);
int total_time=0;
for(i=0;i<n;i++){
que1[i].waitting_time=total_time;
total_time+=que1[i].burst_time;
que1[i].turnaround_time=total_time;
printf("-P%d-",que1[i].process_id);
}
print_pretty(que1,n,total_time);
return(1);
}
int rr(struct process *que1,int n){
int time_quanta;
printf("Enter the time quanta:\t");
scanf("%d",&time_quanta);
int total_time=0;
int n1=n,count=0;
while(n1){
if(que1[count].remaining_time<=time_quanta && que1[count].remaining_time>0){
total_time+=que1[count].remaining_time;
que1[count].remaining_time=0;
que1[count].turnaround_time=total_time;
que1[count].waitting_time=total_time-que1[count].burst_time;
printf("-P%d-",que1[count++].process_id);
n1--;
}
else if(que1[count].remaining_time>0){
que1[count].remaining_time-=time_quanta;
total_time+=time_quanta;
printf("-P%d-",que1[count++].process_id);
}
else
count++;
if(count==n)
count=count%n;
}
print_pretty(que1,n,total_time);
return(1);
}
int priority(struct process *que1,int n){
sort(que1,n,3);
int total_time=0;
for(i=0;i<n;i++){
que1[i].waitting_time=total_time;
total_time+=que1[i].burst_time;
que1[i].turnaround_time=total_time;
printf("-P%d-",que1[i].process_id);
}
print_pretty(que1,n,total_time);
return(1);
}
int modifiedmultiLevelScheduling(struct process *que1,int n,float x){
int i,j;
int sum=0;
float mean,standard_deviation=0.0;
for(i=0;i<n;i++){
sum+=que1[i].burst_time;
int r=1;
for(j=0;j<n;j++)
if(j!=i && que1[j].burst_time<que1[i].burst_time)
r++;
```

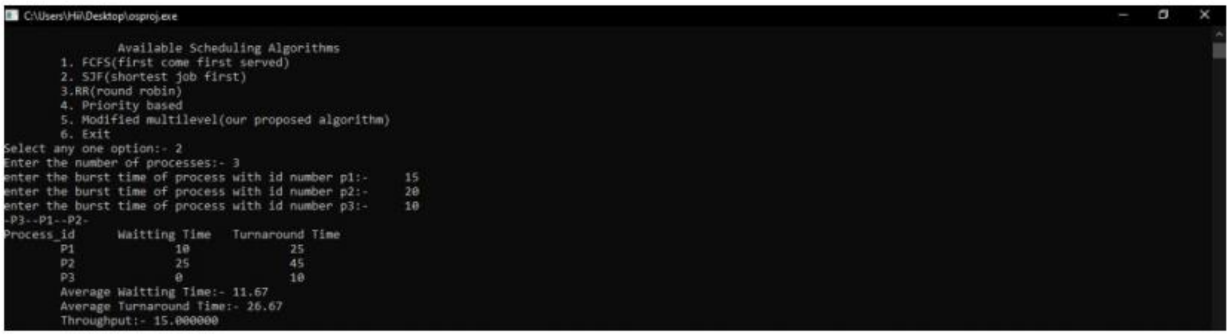
```
que1[i].shortness_rank=r;
que1[i].new_rank=(float)((x/100)*que1[i].priority+((100-x)/100)*que1[i].shortness_rank);
}
mean=(float)sum/n;
for(i=0;i<n;i++)
standard_deviation+=(que1[i].burst_time-mean)*(que1[i].burst_time-mean);
standard_deviation=sqrt(standard_deviation/n);
int time_quanta=(int)(mean+0.5*standard_deviation);
int t=n,n2,n1=n;
int total_time=0;
struct process *que2;
struct process *p=que1;
sort(que1,n,1);
while (t){
que2=(struct process*)malloc(sizeof(struct process)*n1);
n2=0;
for(i=0;i<n1;i++){
if(que1[i].remaining_time<time_quanta){
total_time+=que1[i].remaining_time;
int x=returnIndex(p,n,que1[i].process_id);
p[x].turnaround_time=total_time;
p[x].waitting_time=total_time-p[x].burst_time;
que1[i].remaining_time=0;
t--;
}
else{
total_time+=time_quanta;
que1[i].remaining_time-=time_quanta;
que2[n2++]=que1[i];
}
printf("-P%d-",que1[i].process_id);
}
que1=que2;
n1=n2;
}
print_pretty(p,n,total_time);
return(1);
```

```
}
int main(){
int c,n;
float x;
while(1){
printf("\n\t\tAvailable Scheduling Algorithms\n\t1. FCFS(first come first served)\n\t2. SJF(shortest
job first)\n\t3.RR(round robin)\n");
printf("\t4. Priority based\n\t5. Modified multilevel(our proposed algorithm)\n\t6. Exit\nSelect any
one option:-\t");
scanf("%d",&c);
if(c<6){
printf("Enter the number of processes:-\t");
scanf("%d",&n);
struct process queue1[n];
if(c<4)
for(i=0;i<n;i++){
printf("enter the burst time of process with id number p%d:-\t",i+1);
scanf("%d",&queue1[i].burst_time);
queue1[i].remaining_time=queue1[i].burst_time;
queue1[i].process_id=i+1;
}
else if(c==4)
for(i=0;i<n;i++){
printf("enter the burst time and priority of process with id number p%d:-\t",i+1);
scanf("%d %d",&queue1[i].burst_time,&queue1[i].priority);
queue1[i].remaining_time=queue1[i].burst_time;
queue1[i].process_id=i+1;
}
else if(c==5){
for(i=0;i<n;i++){
printf("enter the burst time and priority of process with id number p%d:-\t",i+1);
scanf("%d %d",&queue1[i].burst_time,&queue1[i].priority);
queue1[i].remaining_time=queue1[i].burst_time;
queue1[i].process_id=i+1;
}
printf("Enter priority as 50 for optimum results \n");
printf("enter the vatage to be given to priority in percentage:");
scanf("%f",&x);
}
switch (c){
case 1: fcfs(queue1,n);
break;
case 2: sjf(queue1,n);
break;
case 3: rr(queue1,n);
break;
case 4: priority(queue1,n);
break;
case 5: modifiedmultiLevelScheduling(queue1,n,x);
break;
default: printf("Wrong option selected");
}
}
else{
exit(0);
}
}
return(0);
}
```

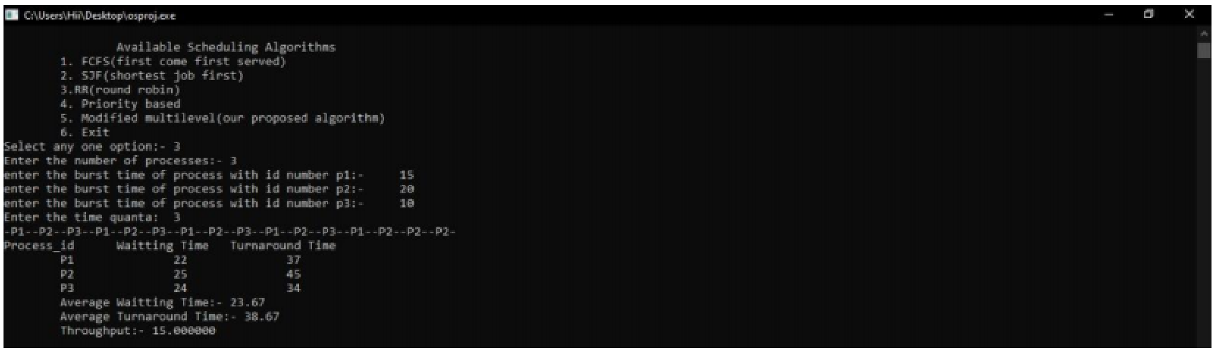
3.2 RESULT :AND DISCUSSION :



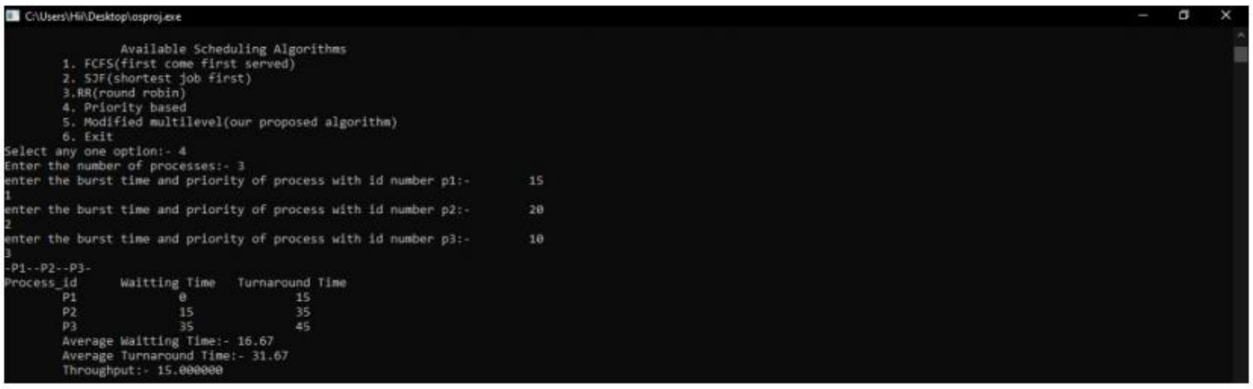
```
                Available Scheduling Algorithms
        1. FCFS(first come first served)
        2. SJF(shortest job first)
        3.RR(round robin)
        4. Priority based
        5. Modified multilevel(our proposed algorithm)
        6. Exit
Select any one option:- 1
Enter the number of processes:- 3
enter the burst time of process with id number p1:-     15
enter the burst time of process with id number p2:-     20
enter the burst time of process with id number p3:-     10
-P1--P2--P3-
Process_id      Waitting Time   Turnaround Time
        P1              0               15
        P2              15              35
        P3              35              45
        Average Waitting Time:- 16.67
        Average Turnaround Time:- 31.67
        Throughput:- 15.000000
```



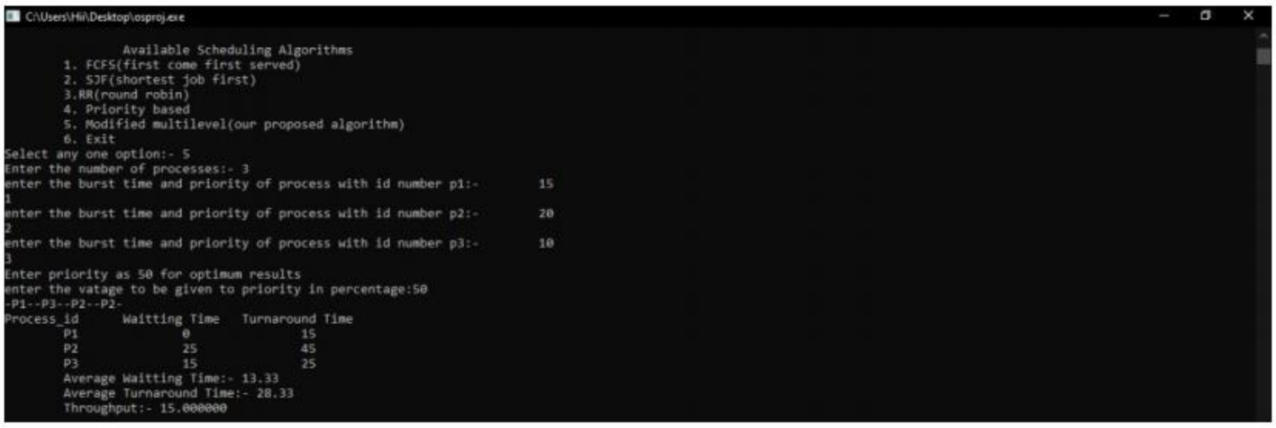
```
                Available Scheduling Algorithms
        1. FCFS(first come first served)
        2. SJF(shortest job first)
        3.RR(round robin)
        4. Priority based
        5. Modified multilevel(our proposed algorithm)
        6. Exit
Select any one option:- 2
Enter the number of processes:- 3
enter the burst time of process with id number p1:-     15
enter the burst time of process with id number p2:-     20
enter the burst time of process with id number p3:-     10
-P3--P1--P2-
Process_id      Waitting Time   Turnaround Time
        P1              10              25
        P2              25              45
        P3              0               10
        Average Waitting Time:- 11.67
        Average Turnaround Time:- 26.67
        Throughput:- 15.000000
```



```
                Available Scheduling Algorithms
        1. FCFS(first come first served)
        2. SJF(shortest job first)
        3.RR(round robin)
        4. Priority based
        5. Modified multilevel(our proposed algorithm)
        6. Exit
Select any one option:- 3
Enter the number of processes:- 3
enter the burst time of process with id number p1:-     15
enter the burst time of process with id number p2:-     20
enter the burst time of process with id number p3:-     10
Enter the time quanta:  3
-P1--P2--P3--P1--P2--P3--P1--P2--P3--P1--P2--P3--P1--P2--P2--P2-
Process_id      Waitting Time   Turnaround Time
        P1              22              37
        P2              25              45
        P3              24              34
        Average Waitting Time:- 23.67
        Average Turnaround Time:- 38.67
        Throughput:- 15.000000
```



```
                Available Scheduling Algorithms
        1. FCFS(first come first served)
        2. SJF(shortest job first)
        3.RR(round robin)
        4. Priority based
        5. Modified multilevel(our proposed algorithm)
        6. Exit
Select any one option:- 4
Enter the number of processes:- 3
enter the burst time and priority of process with id number p1:-        15
1
enter the burst time and priority of process with id number p2:-        20
2
enter the burst time and priority of process with id number p3:-        10
3
-P1--P2--P3-
Process_id      Waitting Time   Turnaround Time
        P1              0               15
        P2              15              35
        P3              35              45
        Average Waitting Time:- 16.67
        Average Turnaround Time:- 31.67
        Throughput:- 15.000000
```



```
C:\Users\Hii\Desktop\osproj.exe
                Available Scheduling Algorithms
        1. FCFS(first come first served)
        2. SJF(shortest job first)
        3.RR(round robin)
        4. Priority based
        5. Modified multilevel(our proposed algorithm)
        6. Exit
Select any one option:- 5
Enter the number of processes:- 3
enter the burst time and priority of process with id number p1:-       15
1
enter the burst time and priority of process with id number p2:-       20
2
enter the burst time and priority of process with id number p3:-       10
3
Enter priority as 50 for optimum results
enter the vatage to be given to priority in percentage:50
-P1--P3--P2--P2-
Process_id      Waitting Time   Turnaround Time
        P1              0               15
        P2              25              45
        P3              15              25
        Average Waitting Time:- 13.33
        Average Turnaround Time:- 28.33
        Throughput:- 15.000000
```

DISCUSSION ABOUT THE CODE:

1.) The priority weightage is user defined . The default value (if the user does not specify) is considered to be 50 %.

2.) 50% weightage to the priority gives better and optimum results.

4.1 CONCLUSION :

On comparing with the other scheduling algorithms, our proposed algorithm gives better results and is successful in saving waiting time and throughput time.

4.2 REFERENCES :

- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4531731
- **https://ieeexplore.ieee.org/document/8392238**
- Dynamic Time Quantum based Round Robin CPU

https://www.ijcaonline.org/archives/volume167/number13/ 27835-2017914569

- Scheduling Algorithm by Yosef Berhanu Department of Computer Science University of Gondar Ethiopia AbebeAlemu Department of Computer Science University of Gondar Ethiopia Manish Kumar Mishra Department of Computer Science University of Gondar Ethiopia