VIT®

Vellore Institute of Technology

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

OPERATING SYSTEMS

TEAM MEMBERS:

DISHANK JHAVERI	18BCE0991
LAYA RATHOD	18BCE2162
AMAAN CHADHA	18BCE2155
HRITHIK DHOKA	18BCE0994

PROJECT REVIEW 3

PROJECT TITLE

IMPLEMENTATION OF SCHEDULING ALGORITHMS IN THE WORKING OF RESTAURANT CONVEYOR BELTS

Contents covered in the Review:

1. PROJECT TITLE

IMPLEMENTATION OF SCHEDULING ALGORITHMS IN THE WORKING OF RESTAURANT CONVEYOR BELTS

2. SLOT

F2+TF2
Lab 19+20

3. ABSTRACT

The Central Processing Unit (CPU) scheduling in today's generation plays a deep-seated role during designing of latest processors and units. Not only it is the premium desire of today's processing systems to efficiently switch between processes to maximize throughput, but also to do it in a way so that the idle time is reduced to a bare minimum. By switching the CPU among various processes, these targets are attempted to be achieved.

The performance of any CPU majorly depends on the scheduling algorithm used by the Processor. Primarily, the CPU is one of the most essential computer resources we have today. Not much can be done with the efficiency in executing the processes, since they take almost similar times, but a lot can be done with the scheduling. And, since the round robin scheduling algorithm is considered to be one of the most widely used algorithms, a new proposed variant of this algorithm is attempted to be established. We shall attempt improving upon the Round Robin CPU Scheduling System incorporating Shortest Job First Scheduling, achievement of stability in the Round Robin Scheduling, using the concept of dynamic Time Quantum.

4. KEYWORDS

- Operating System,
- Scheduling,
- Round Robin algorithms,
- Shortest Job First,
- Waiting time,
- Turnaround time,

5. INTRODUCTION

As processor is the most important resource that essentially determines the working capacity of a machine, CPU scheduling becomes very important in accomplishing the operating system design goals.

The target of any Operating System should be **"To allow maximum processes running at all the times in order to make best use of the CPU."** The efficiency of any CPU scheduler primarily depends on the design of the well-structured scheduling algorithms which goes well with the scheduling goals.

In this paper, we propose an algorithm which can handle all types of process with optimum scheduling criteria.

The efficiency of scheduling algorithm is measured by the following performance factors:

1. **Throughput:**
   Defined as the number of processes executed in one unit time.
2. **Waiting- time:**
   It is the time a process has to wait for CPU in ready-queue to come in main memory.
3. **Turn-around time:**
   Defined as the amount of time required by any process to complete its execution.
4. **Response time:**
   The time between generation of a request and the first response.
5. **Context Switching:**
   The process of switching CPU between processes, also known as pre-emption.

The **CPU Utilization** is a measure of maximum usage of CPU, or how effectively CPU is made busy. The performance and effectiveness of the Round Robin algorithm is largely dependent on the value of **time quantum** selected.

If the value of time quantum is too small then the number of context switches will be more and algorithm will not be effective. If the value of the time quantum is too large, then the algorithm will work more or less like FCFS algorithm.

Improved Round Robin with dynamic Choosing an optimum time quantum can significantly decrease the number of context switches, maintaining the RR nature and also can improve performance. The number of context switches can further be reduced if there is a strategy to execute processes with smallest remaining burst- times.

CPU scheduling is the basis of all operating systems. **The technique used for controlling the order of job which is to be performed by a CPU of a computer is called Scheduling.**

Most CPU scheduling algorithms concentrate on:

- <u>MAXIMIZING:</u>

1. CPU utilization

2. Throughput

   And,

- MINIMIZING:

1. Turnaround time
2. Response time
3. Waiting time
4. Number of context switching for a set of requests.

1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive)
5. Priority Scheduling (Non Pre-emptive)
6. Round Robin

6. LITERATURE SURVEY

The References have been listed in the References Section. An in-depth study of the papers and journals has been done, and the following survey has been established. We looked into the reviews, papers and journals presented by the earlier authors, and gathered following points of our own regarding the reviews.

[1]

This was one of the First greatest advancements made in the Round Robin Scheduling Algorithm. Proposed by Sir Manish Kumar Mishra in 2012, the paper describes a new improved version of RR. Improved Round Robin picks the first process from the ready queue and allocate the CPU to it for a time interval of up to 1 time quantum. Once a process's time quantum is elapsed, it checks the remaining CPU burst times of the processes currently in execution. If the CPU burst time remaining of the current process is less than 1 TQ, the CPU shall again be allocated to the process currently in execution for the remaining burst time.

[2]

In year 2013, Aashna Bisht [2] performed an analysis, and proposed a work Enhanced Round Robin(ERR) model, in which, the time quantum of only those processes which require a slightly greater time than the allotted time quantum cycle were modified. The remaining process will be executed in the already proposed Round robin manner.

[3]

The Next advancement came in the year 2011, by Saroj Hiranwal. In this, first of all we arrange the processes according to the execution time/burst time in increasing order that is smallest the burst time higher the priority of the running process. The smart time slice, then calculated, is the median process burst time of all CPU burst times in the ready queue.

[4]

Many other advancements came along the years between 2011 and 2014. In year 2011, H.S. Behera proposed an improved process scheduling algorithm by using dynamic quantum time along with its weighted mean.

In year 2011, Rakash Mohanty & Manas Das performed a work in which a new variant of Round Robin scheduling algorithms was created by executing the processes according to the calculated Fit factor and they also used the concept of dynamic quantum time.

[5]

In year 2012,Ishwari Singh Rajput, Deepa Gupta [5] proposed priority based Round-robin CPU scheduling algorithms is based on the integration of round robin and priority scheduling. It still holds the advantage of round robin in reducing starvation and also integrates the advantage of sjf scheduling.

[6]

In year 2012, P.Surendra Varma[6] performed a work. In this paper the quantum time is computed with the help of median and highest burst time.

[7]

In year 2012, H.S. Behera & Brajendra Kumar Swain[7] performed a work it gives precedence to all processes according to their priority and burst time, then applies the Round Robin algorithm on it. This Proposed algorithm is developed by taking mean of dynamic quantum time in account.

7. PROPOSED MODEL AND ARCHITECTURE

The project presents a vast scope for future advancements and improvements. Each and every day, the world is growing better with the advancements in Technology. To follow up with the ever-increasing efficiencies, new scheduling algorithms are need to be presented and developed. Better the CPU scheduling algorithms in their efficiency, faster would be the jobs taken up by the processor, and hence faster the execution. Using these CPU scheduling algorithms we aim to reduce the time taken to deliver food on conveyor belts. This project utilises various CPU Scheduling Algorithms to drastically reduce this time and provide efficient results.

8. SOURCECODE

```
#include<stdio.h>
#include<stdlib.h>
```

```
float arr[6];

typedef struct process{
        int name[5];
        int bt;
        int at;
        int prt;
        int wt,ta;
        int flag;
}processes;


void b_sort(processes temp[],int n)
{
        processes t;
        int i,j;
        for(i=1;i<n;i++)
                for(j=0;j<n-i;j++){
                        if(temp[j].at > temp[j+1].at){
                                t = temp[j];
                                temp[j] = temp[j+1];
                                temp[j+1] = t;
                        }
                }
}

int accept(processes P[]){
        int i,n;
        printf("\n Enter total no. of orders : ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("\n ORDER [%d] \n",i+1);
                printf(" Enter order no : ");
                scanf("%s",&P[i].name);
                printf(" Enter amount of time on conveyor belt : ");
                scanf("%d",&P[i].bt);
                printf(" Enter the time the order enters the conveyor belt : ");
                scanf("%d",&P[i].at);
                printf(" Enter token no. of order : ");
                scanf("%d",&P[i].prt);
        }
        printf("\nORD NO.\tB.T.\tA.T.\tTOKEN NO.");
        for(i=0;i<n;i++)
                printf("\n %s\t%d\t%d\t%d",P[i].name,P[i].bt,P[i].at,P[i].prt);
        return n;
}

// FCFS Algorithm
void FCFS(processes P[],int n){
        processes temp[10];
        int sumw=0,sumt=0;
        int x = 0;
        float avgwt=0.0,avgta=0.0;
        int i,j;
        for(i=0;i<n;i++)
                temp[i]=P[i];

        b_sort(temp,n);

                printf("\n\nORD NO.\tB.T.\tA.T.");
                for(i=0;i<n;i++)
                        printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at);
                sumw = temp[0].wt = 0;
                sumt = temp[0].ta = temp[0].bt - temp[0].at;

                for(i=1;i<n;i++){
                        temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;;
                        temp[i].ta = (temp[i].wt + temp[i].bt);
                        sumw+=temp[i].wt;
                        sumt+=temp[i].ta;
                }
                avgwt = (float)sumw/n;
                avgta = (float)sumt/n;
                printf("\n\nORD NO.\tB.T.\tA.T.\tW.T\tT.A.T");
                for(i=0;i<n;i++)
                        printf("\n
%s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta);

                printf("\n\n GANTT CHART\n ");
                for(i=0;i<n;i++)
                        printf("   %s   ",temp[i].name);
                printf("\n ");

                printf("0\t");
                for(i=1;i<=n;i++){
                        x+=temp[i-1].bt;
                        printf("%d      ",x);
                }
                printf("\n\n Average waiting time = %0.2f\n Average turn-around =
%0.2f.",avgwt,avgta);

}


//SJF Non Pre-emptive
void SJF_NP(processes P[],int n){
        processes temp[10];
        processes t;
        int sumw=0,sumt=0;
        int x = 0;
```

```
	float avgwt=0.0,avgta=0.0;
	int i,j;

	for(i=0;i<n;i++)
		temp[i]=P[i];

	b_sort(temp,n);

	for(i=2;i<n;i++)
		for(j=1;j<n-i+1;j++){
			if(temp[j].bt > temp[j+1].bt){
				t = temp[j];
				temp[j] = temp[j+1];
				temp[j+1] = t;
			}
		}

	printf("\n\nORD NO.\tB.T.\tA.T.");
		for(i=0;i<n;i++)
			printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at);

		sumw = temp[0].wt = 0;
		sumt = temp[0].ta = temp[0].bt - temp[0].at;

		for(i=1;i<n;i++){
			temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;;
			temp[i].ta = (temp[i].wt + temp[i].bt);
			sumw+=temp[i].wt;
			sumt+=temp[i].ta;
		}
		avgwt = (float)sumw/n;
		avgta = (float)sumt/n;
		printf("\n\nORD NO.\tB.T.\tA.T.\tW.T\tT.A.T");
		for(i=0;i<n;i++)
			printf("\n
%s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta);

		printf("\n\n GANTT CHART\n ");
		for(i=0;i<n;i++)
			printf("   %s   ",temp[i].name);
		printf("\n ");

		printf("0\t");
		for(i=1;i<=n;i++){
			x+=temp[i-1].bt;
			printf("%d      ",x);
		}
		printf("\n\n Average waiting time = %0.2f\n Average turn-around =
%0.2f.",avgwt,avgta);
```

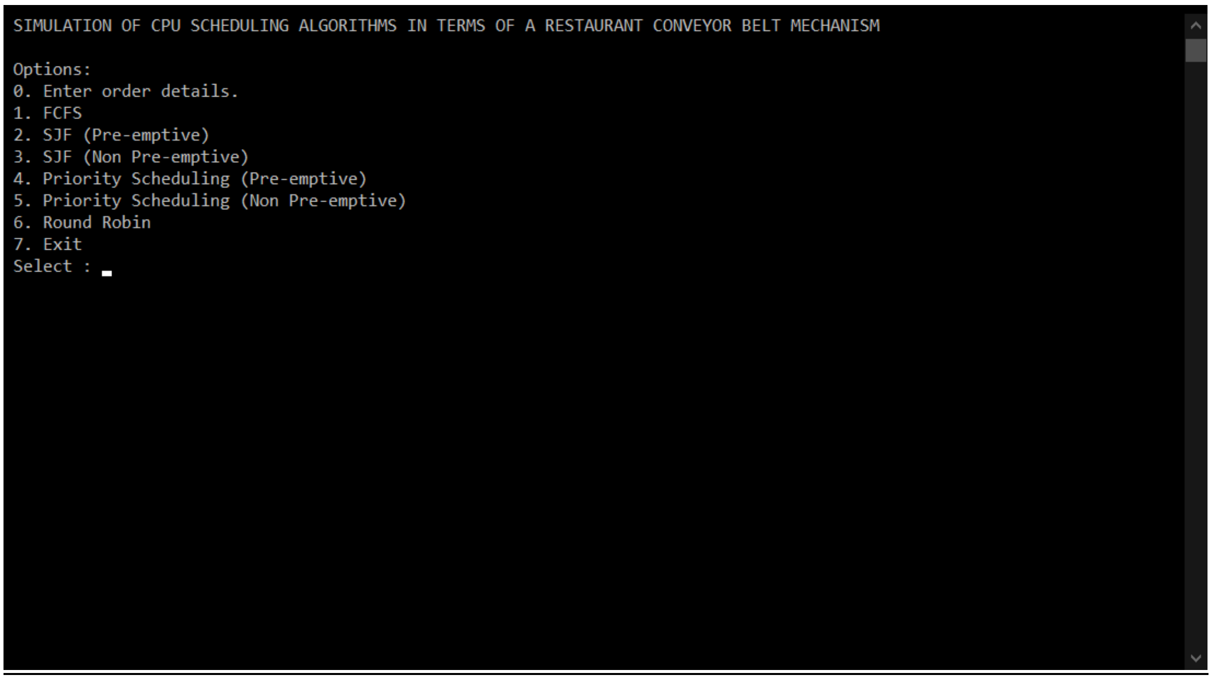
```
}

//Priority Non Pre-emptive
void PRT_NP(processes P[],int n)
{
        processes temp[10];
        processes t;
        int sumw=0,sumt=0;
        float avgwt=0.0,avgta=0.0;
        int i,j;
        int x = 0;

        for(i=0;i<n;i++)
                temp[i]=P[i];

        b_sort(temp,n);

        for(i=2;i<n;i++)
                for(j=1;j<n-i+1;j++){
                        if(temp[j].prt > temp[j+1].prt){
                                t = temp[j];
                                temp[j] = temp[j+1];
                                temp[j+1] = t;
                        }
                }

        printf("\n\nORD NO.\tB.T.\tA.T.");
                for(i=0;i<n;i++)
                        printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at);

                sumw = temp[0].wt = 0;
                sumt = temp[0].ta = temp[0].bt - temp[0].at;

                for(i=1;i<n;i++){
                        temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;;
                        temp[i].ta = (temp[i].wt + temp[i].bt);
                        sumw+=temp[i].wt;
                        sumt+=temp[i].ta;
                }
                avgwt = (float)sumw/n;
                avgta = (float)sumt/n;
                printf("\n\nORD NO.\tB.T.\tA.T.\tW.T\tT.A.T");
                for(i=0;i<n;i++)
                        printf("\n
%s\t\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta);

                printf("\n\n GANTT CHART\n ");
                for(i=0;i<n;i++)
                        printf("   %s   ",temp[i].name);
                printf("\n ");
```

```
		printf("0\t");
		for(i=1;i<=n;i++){
			x+=temp[i-1].bt;
			printf("%d      ",x);
		}
		printf("\n\n Average waiting time = %0.2f\n Average turn-around = 
%0.2f.",avgwt,avgta);

}

//Round Robin Scheduling
void RR(processes P[],int n)
{
	int pflag=0,t,tcurr=0,k,i,Q=0;
	int sumw=0,sumt=0;
	float avgwt=0.0,avgta=0.0;
	processes temp1[10],temp2[10];

	for(i=0;i<n;i++)
		temp1[i]=P[i];

	b_sort(temp1,n);

	for(i=0;i<n;i++)
		temp2[i]=temp1[i];

	printf("\n Enter quantum time : ");
	scanf("%d",&Q);

	for(k=0;;k++){
		if(k>n-1)
			k=0;
		if(temp1[k].bt>0)
			printf("  %d  %s",tcurr,temp1[k].name);
		t=0;
		while(t<Q && temp1[k].bt > 0){
			t++;
			tcurr++;
			temp1[k].bt--;
		}
		if(temp1[k].bt <= 0 && temp1[k].flag != 1){
			temp1[k].wt = tcurr - temp2[k].bt - temp1[k].at;
			temp1[k].ta = tcurr - temp1[k].at;
			pflag++;
			temp1[k].flag = 1;
			sumw+=temp1[k].wt;
			sumt+=temp1[k].ta;
		}
		if(pflag == n)
```

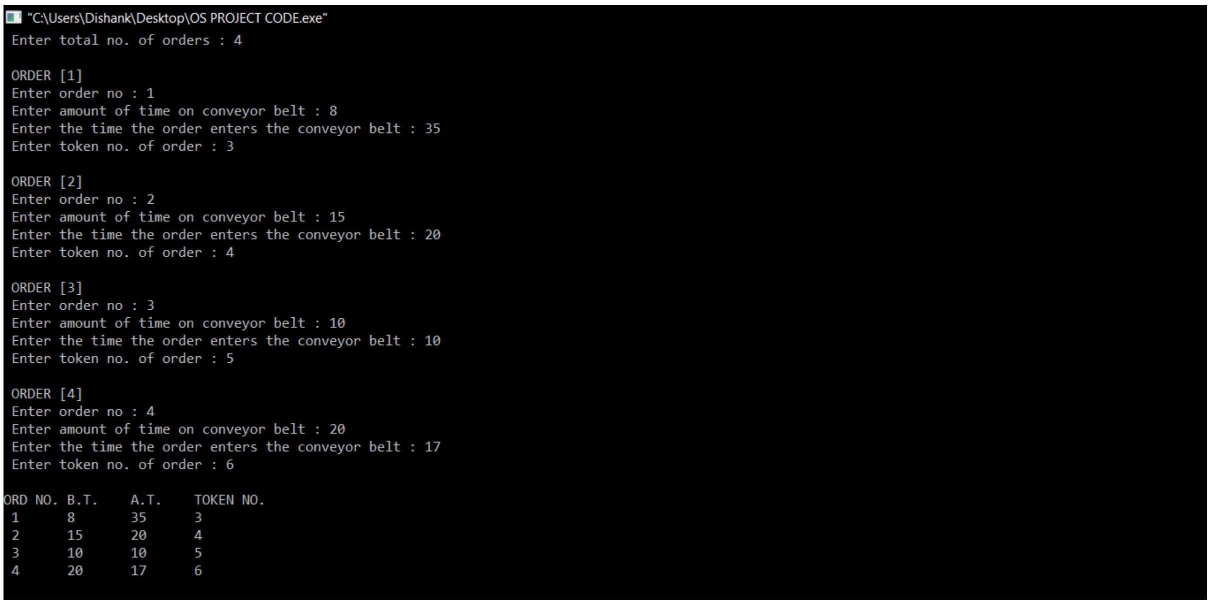
```
                        break;
            }
            printf("  %d",tcurr);
            avgwt = (float)sumw/n;
            avgta = (float)sumt/n;
            printf("\n\n Average waiting time = %0.2f\n Average turn-around =
%0.2f.",avgwt,avgta);

}

//Shortest Job First - Pre-emptive
void SJF_P(processes P[],int n){
            int i,t_total=0,tcurr,b[10],min_at,j,x,min_bt;
            int sumw=0,sumt=0;
            float avgwt=0.0,avgta=0.0;
            processes temp[10],t;

            for(i=0;i<n;i++){
                        temp[i]=P[i];
                        t_total+=P[i].bt;
            }

            b_sort(temp,n);

                        printf("\n\nORD NO.\tB.T.\tA.T.");
                        for(i=0;i<n;i++)
                                    printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at);
                        sumw = temp[0].wt = 0;
                        sumt = temp[0].ta = temp[0].bt - temp[0].at;

                        for(i=1;i<n;i++){
                                    temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;;
                                    temp[i].ta = (temp[i].wt + temp[i].bt);
                                    sumw+=temp[i].wt;
                                    sumt+=temp[i].ta;
                        }
                        avgwt = (float)sumw/n;
                        avgta = (float)sumt/n;
                        printf("\n\nORD NO.\tB.T.\tA.T.\tW.T\tT.A.T");
                        for(i=0;i<n;i++)
                printf("\n
%s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta);

            for(i=0;i<n;i++)
                        b[i] = temp[i].bt;

            i=j=0;
            printf("\n GANTT CHART\n\n %d %s",i,temp[i].name);
            for(tcurr=0;tcurr<t_total;tcurr++){
                if(b[i] > 0 && temp[i].at <= tcurr)
                        b[i]--;

                if(i!=j)
                        printf(" %d %s",tcurr,temp[i].name);

                if(b[i]<=0 && temp[i].flag != 1){

                        temp[i].flag = 1;
                        temp[i].wt = (tcurr+1) - temp[i].bt - temp[i].at;
                        temp[i].ta = (tcurr+1) - temp[i].at;
                        sumw+=temp[i].wt;
                        sumt+=temp[i].ta;
                }
                j=i;    min_bt = 999;
                for(x=0;x<n;x++){

                        if(temp[x].at <= (tcurr+1) && temp[x].flag != 1){

                                if(min_bt != b[x] && min_bt > b[x]){
                                        min_bt = b[x];
                                        i=x;
                                }
                        }
                }

        }
        printf(" %d",tcurr);
        avgwt = (float)sumw/n;          avgta = (float)sumt/n;
        printf("\n\n Average waiting time = %0.2f\n Average turn-around =
%0.2f.",avgwt,avgta);

}


void PRT_P(processes P[],int n){
        int i,t_total=0,tcurr,b[10],j,x,min_pr;
        int sumw=0,sumt=0;
        float avgwt=0.0,avgta=0.0;
        processes temp[10],t;

        for(i=0;i<n;i++){
                temp[i]=P[i];
                t_total+=P[i].bt;
        }

        b_sort(temp,n);

        for(i=0;i<n;i++)
                b[i] = temp[i].bt;
        i=j=0;
        printf("\n GANTT CHART\n\n %d %s",i,temp[i].name);
        for(tcurr=0;tcurr<t_total;tcurr++)
        {

                if(b[i] > 0 && temp[i].at <= tcurr)
                        b[i]--;

                if(i!=j)
                        printf(" %d %s",tcurr,temp[i].name);

                if(b[i]<=0 && temp[i].flag != 1)
                {
                        temp[i].flag = 1;
                        temp[i].wt = (tcurr+1) - temp[i].bt - temp[i].at;
                        temp[i].ta = (tcurr+1) - temp[i].at;
                        sumw+=temp[i].wt;
                        sumt+=temp[i].ta;
                }
                j=i;
                min_pr = 999;
                for(x=0;x<n;x++){

                        if(temp[x].at <= (tcurr+1) && temp[x].flag != 1){

                                if(min_pr != temp[x].prt && min_pr > temp[x].prt){
                                        min_pr = temp[x].prt;
                                        i=x;
                                }
                        }
                }

        }
        printf(" %d",tcurr);
        avgwt = (float)sumw/n;
        avgta = (float)sumt/n;
        printf("\n\n Average waiting time = %0.2f\n Average turn-around =
%0.2f.",avgwt,avgta);

}


int main(){

        processes P[10];
        int ch,n;
        do{
                printf("\n\n SIMULATION OF CPU SCHEDULING ALGORITHMS IN
TERMS OF A RESTAURANT CONVEYOR BELT MECHANISM\n");
```

```
                printf("\n Options:");
                printf("\n 0. Enter order details.");
                printf("\n 1. FCFS");
                printf("\n 2. SJF (Pre-emptive)");
                printf("\n 3. SJF (Non Pre-emptive)");
                printf("\n 4. Priority Scheduling (Pre-emptive)");
                printf("\n 5. Priority Scheduling (Non Pre-emptive)");
                printf("\n 6. Round Robin");
                printf("\n 7. Exit\n Select : ");
                scanf("%d",&ch);
                switch(ch){
                        case 0:
                                n=accept(P);
                                break;
                        case 1:
                                FCFS(P,n);
                                break;
                        case 2:
                                SJF_P(P,n);
                                break;
                        case 3:
                                SJF_NP(P,n);
                                break;
                        case 4:
                                PRT_P(P,n);
                                break;
                        case 5:
                                PRT_NP(P,n);
                                break;
                        case 6:
                                RR(P,n);
                                break;
                        case 7:exit(0);
                }
        }while(ch != 7);

return 0;
}
```

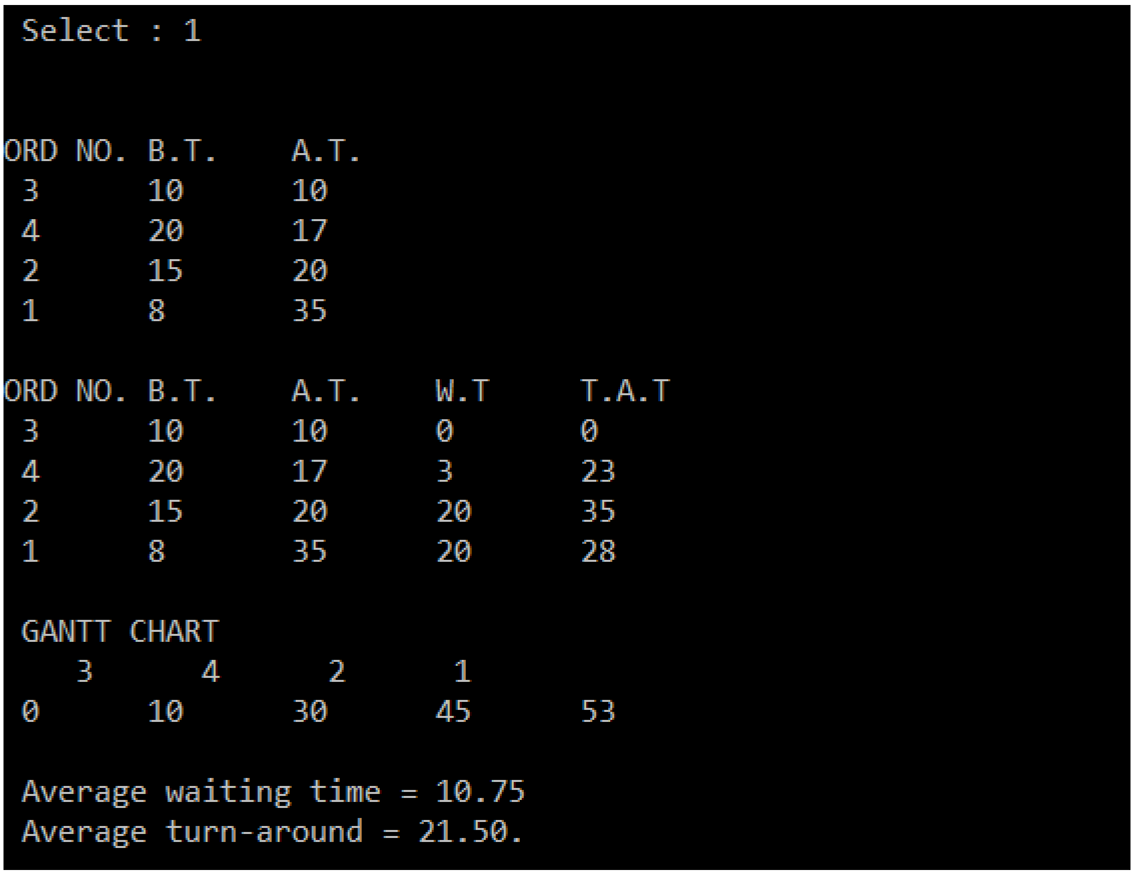
OUTPUT SCREENSHOTS OF THE IMPLEMENTATION



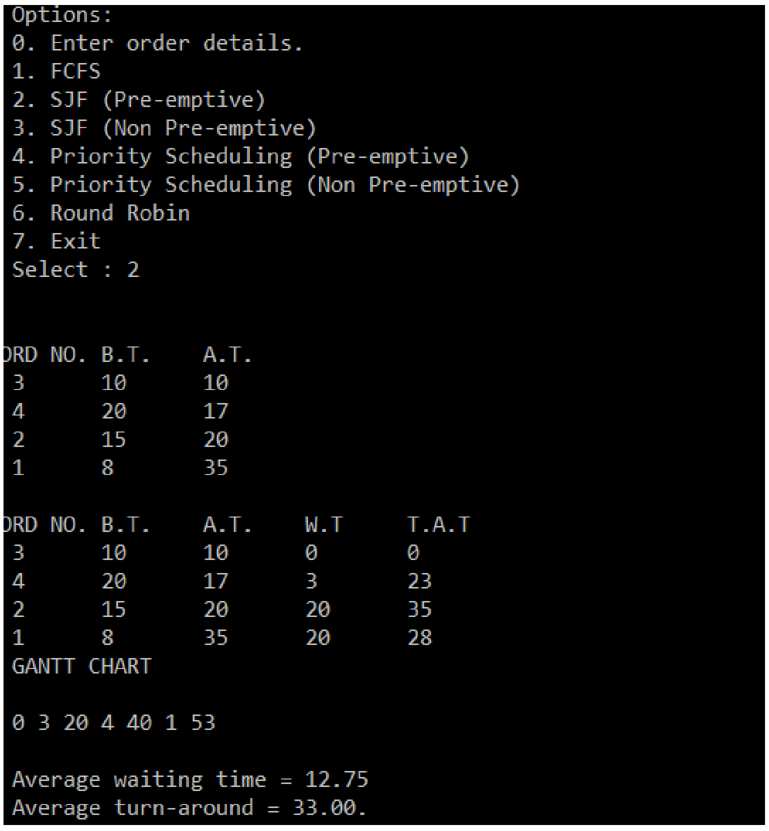
```
SIMULATION OF CPU SCHEDULING ALGORITHMS IN TERMS OF A RESTAURANT CONVEYOR BELT MECHANISM

Options:
0. Enter order details.
1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive)
5. Priority Scheduling (Non Pre-emptive)
6. Round Robin
7. Exit
Select : _
```



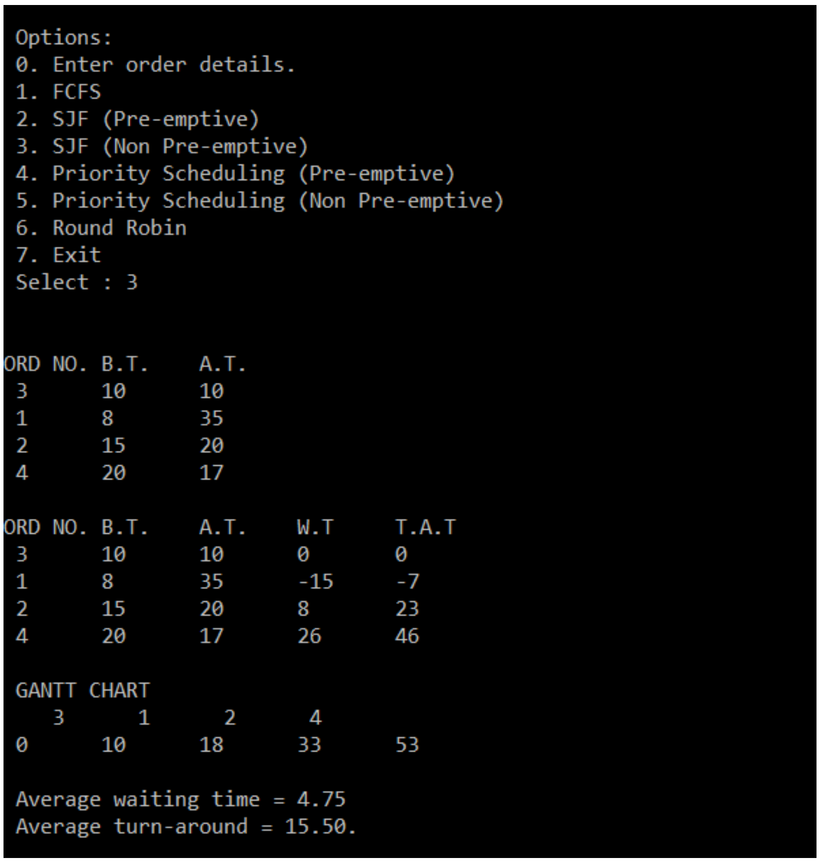
```
Enter total no. of orders : 4

ORDER [1]
Enter order no : 1
Enter amount of time on conveyor belt : 8
Enter the time the order enters the conveyor belt : 35
Enter token no. of order : 3

ORDER [2]
Enter order no : 2
Enter amount of time on conveyor belt : 15
Enter the time the order enters the conveyor belt : 20
Enter token no. of order : 4

ORDER [3]
Enter order no : 3
Enter amount of time on conveyor belt : 10
Enter the time the order enters the conveyor belt : 10
Enter token no. of order : 5

ORDER [4]
Enter order no : 4
Enter amount of time on conveyor belt : 20
Enter the time the order enters the conveyor belt : 17
Enter token no. of order : 6

ORD NO. B.T.    A.T.    TOKEN NO.
 1      8       35      3
 2      15      20      4
 3      10      10      5
 4      20      17      6
```



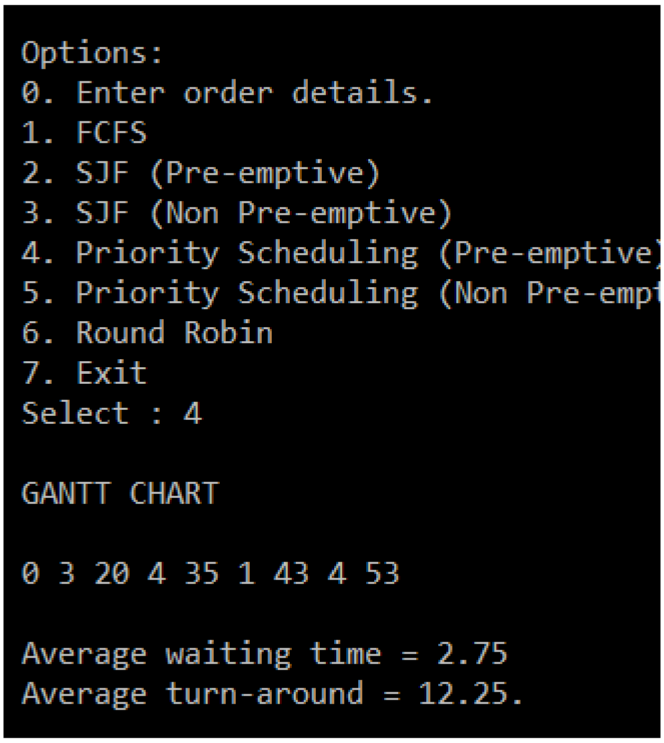
```
 Select : 1


ORD NO. B.T.    A.T.
 3      10      10
 4      20      17
 2      15      20
 1      8       35

ORD NO. B.T.    A.T.    W.T     T.A.T
 3      10      10      0       0
 4      20      17      3       23
 2      15      20      20      35
 1      8       35      20      28

 GANTT CHART
    3      4      2      1
 0      10      30      45      53

 Average waiting time = 10.75
 Average turn-around = 21.50.
```



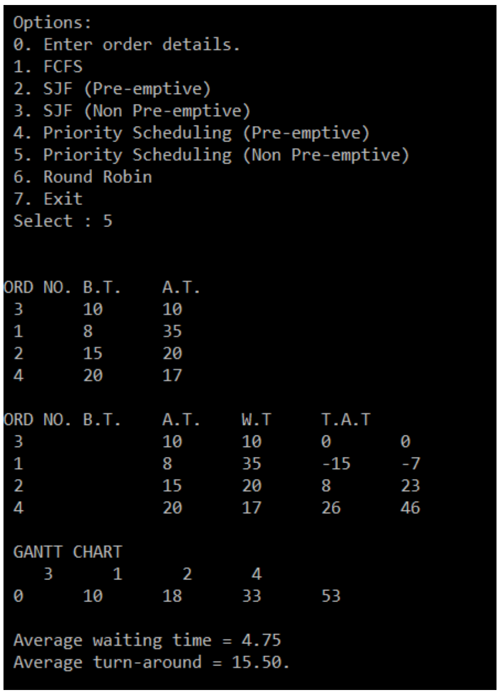
```
 Options:
 0. Enter order details.
 1. FCFS
 2. SJF (Pre-emptive)
 3. SJF (Non Pre-emptive)
 4. Priority Scheduling (Pre-emptive)
 5. Priority Scheduling (Non Pre-emptive)
 6. Round Robin
 7. Exit
 Select : 2


ORD NO. B.T.    A.T.
 3      10      10
 4      20      17
 2      15      20
 1      8       35

ORD NO. B.T.    A.T.    W.T     T.A.T
 3      10      10      0       0
 4      20      17      3       23
 2      15      20      20      35
 1      8       35      20      28
 GANTT CHART

 0 3 20 4 40 1 53

 Average waiting time = 12.75
 Average turn-around = 33.00.
```



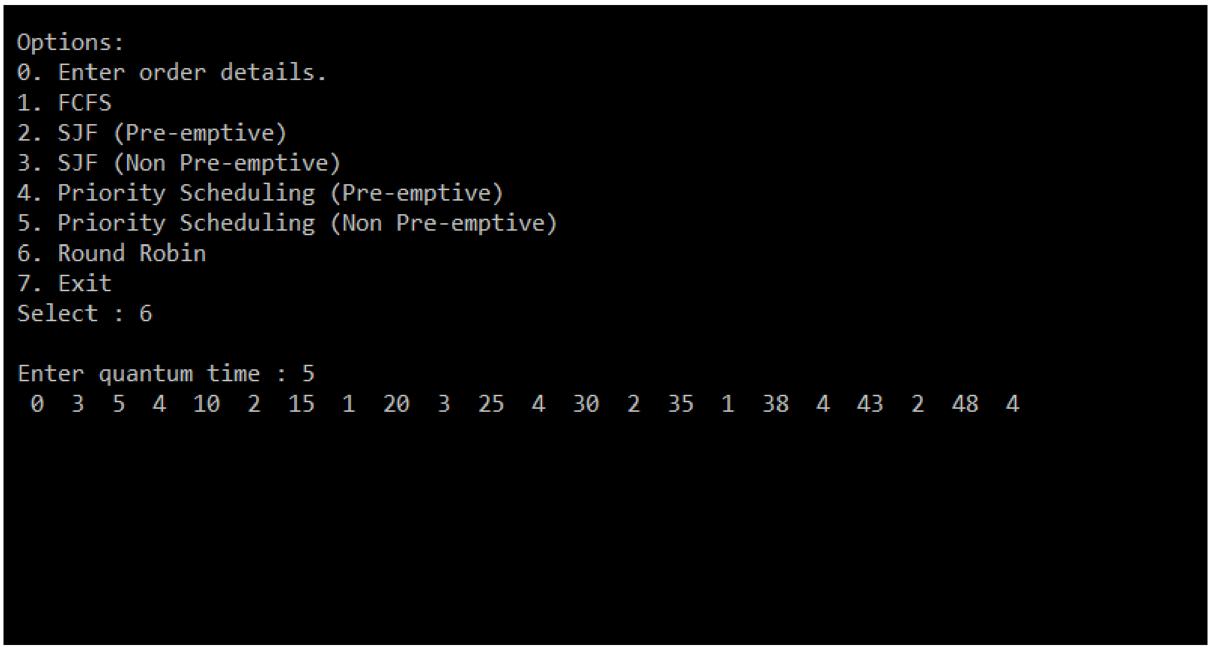
```
Options:
0. Enter order details.
1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive)
5. Priority Scheduling (Non Pre-emptive)
6. Round Robin
7. Exit
Select : 3


ORD NO. B.T.    A.T.
 3      10      10
 1      8       35
 2      15      20
 4      20      17

ORD NO. B.T.    A.T.    W.T     T.A.T
 3      10      10      0       0
 1      8       35      -15     -7
 2      15      20      8       23
 4      20      17      26      46

 GANTT CHART
    3      1       2      4
 0      10      18      33      53

 Average waiting time = 4.75
 Average turn-around = 15.50.
```



```
Options:
0. Enter order details.
1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive
5. Priority Scheduling (Non Pre-empt
6. Round Robin
7. Exit
Select : 4

GANTT CHART

0 3 20 4 35 1 43 4 53

Average waiting time = 2.75
Average turn-around = 12.25.
```



```
Options:
0. Enter order details.
1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive)
5. Priority Scheduling (Non Pre-emptive)
6. Round Robin
7. Exit
Select : 5


ORD NO. B.T.    A.T.
3       10      10
1       8       35
2       15      20
4       20      17

ORD NO. B.T.    A.T.    W.T     T.A.T
3               10      10      0       0
1               8       35      -15     -7
2               15      20      8       23
4               20      17      26      46

GANTT CHART
   3      1      2      4
0       10      18      33      53

Average waiting time = 4.75
Average turn-around = 15.50.
```



```
Options:
0. Enter order details.
1. FCFS
2. SJF (Pre-emptive)
3. SJF (Non Pre-emptive)
4. Priority Scheduling (Pre-emptive)
5. Priority Scheduling (Non Pre-emptive)
6. Round Robin
7. Exit
Select : 6

Enter quantum time : 5
 0  3  5  4  10  2  15  1  20  3  25  4  30  2  35  1  38  4  43  2  48  4
```

9 PARAMETERS IDENTIFIED FOR EVALUATION OF PROPOSED MODEL

Attributes that would be Prerequisites for evaluation will be:

1. **Order Number**
2. **Time On conveyor Belt**
3. **The time the order takes to enter the conveyor belt**
4. **Waiting time**
5. **Burst time**
6. **Arrival time**
7. **Gantt Chart**
8. **Turnaround time**

Parameters considered during the experiment:

Input Parameters

- Order Number
- Time on conveyor Belt
- The time the order takes to enters the conveyor Belt
- Token Number

Output Parameters

- Waiting Time
- Turnaround Time

- Arrival Time
- Burst Time
- Gantt Chart

We performed our experiment evaluating the performance new proposed algorithm considering a data set with different arrival time for the process.

The algorithm works efficiently for a very large number of processes.
We can test the timing for each process with a varying number of orders. This way we can analyse the timing for each order. On continuous testing we can single out how different orders are optimised for different scheduling algorithms

11. CONCLUSION

From the above Experiments, We can conclude that by testing the above process for we can use scheduling algorithms to optimise the process of delivering food using conveyor belts. We can analyse various results and change our process the way we need to.