DR. GAYATHRI P

Additionally Modified Round Robin Algorithm

PROJECT REVIEW – III

L11+L12

Saumye Agarwal -17BCB0067
Shubham gupta -17BCE0199
Patel yash vijaykumar -17BCE0335
Rajat rathi -17BCE0900
Anuraj Srivastava -17BCE2006

1. Abstract

Multiprogramming is a process or method of executing multiple processes simultaneously in the memory. Its primary aim is to minimize the average waiting time, average turnaround time and maximize the CPU utilization. There are various CPU scheduling algorithms are used to performed multiprogramming like First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling (PS) and Round Robin (RR).

This project deals with the simulation of CPU scheduling algorithms with C.

The following algorithms are simulated:

• First Come First Serve (FCFS)

• Shortest Job First

• SRTF Algorithm

• Round Robin

• Our innovative algorithm

The metrics such as waiting time and turnaround time taken for the processes to complete,

number of rounds, etc are calculated.

The target of this project is to also propose a new CPU scheduling algorithm which will perform superior than current round robin algorithm and in most cases better than other algorithms as well in terms of minimizing average waiting time, average turnaround time and number of context switches.

2. Keywords

Context switching, CPU scheduling, Gantt chart, Response time, Round Robin CPU scheduling algorithm, Turnaround time, Waiting time, quantum time, burst time

3. Introduction

In a lone processor system, processor can run only a solitary methodology at some random minute and diverse techniques need to hold up until the point that the moment that the CPU is free and these strategies can be rescheduled yet again.

The objective of multiprogramming is to run numerous process at once and to open up the employments of CPU. Planning is a central working framework work. In multiprogramming figuring frameworks, wastefulness is frequently caused by ill-advised utilization of CPU. In multiprogramming frameworks, various procedures are being kept in memory for most extreme usage of CPU use can be opened up by exchanging CPU among holding up procedures in the memory and running some procedure constantly. Which process ought to be chosen next for administration, is a critical inquiry, since it influences the adequacy of the administration. it impacts the practicality of the organization. The crucial point of the CPU planning calculations is to boosting CPU usage and throughput and limiting turnaround time, holding up time, reaction time and number of settings exchanging for an arrangement of solicitations.

CPU scheduling algorithm are utilized to dispense the CPU to the procedures holding up in the prepared line. A portion of the famous CPU booking calculations are First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin (RR).

CPU SCHEDULING ALGORITHMS

A. FIRST-COME-FIRST-SERVED :

This is a non-preemptive scheduling algorithm. FCFS as the name show doles out need to the procedures in the request in which they ask for the procedures. The process which asks for the CPU first, CPU is designated to that procedures first. FCFS finishes the occupations all together their entry.

❖ First-Come-First-Served calculation is the scheduling algorithm is the minimum complex scheduling algorithm. Procedures are dispatched by their landing time on the ready queue. Being a non preemptive order, once a process has a CPU, it hurries to fruition. The FCFS planning is reasonable in the formal sense or human feeling of reasonableness yet it is unjustifiable as in long employments make short occupations pause and immaterial occupations make imperative employments pause.

❖ FCFS is more unsurprising than the greater part of different plans since it offers time. FCFS plot isn't valuable in booking intelligent clients since it can't ensure great reaction time.

The code for FCFS planning is easy to compose and get it. One of the significant disadvantage of this plan is that the normal time is frequently very long.

The First-Come-First-Served count is on occasion used as a pro arrangement in present day working systems anyway generally introduced inside various plans.

B. SHORTEST-JOB-FIRST

❖ In SJF CPU is relegated to the process which have the littlest CPU errand. This system is very same as the FCFS yet the thing that maters is that the activity with the most limited burst time executed first. At the point when a vocation comes up it is embedded it in the prepared line contingent upon its length. It gives the base normal holding up time and least normal turnaround time for a given arrangement of procedures

❖ The process with the most elevated need ought to dependably be the one that be as of now using the processor or CPU. In the event that a process or employment is as of now utilizing the processor and another process with a higher need enters the CPU, or the prepared rundown, the process or occupation on the processor ought to be expelled and come back to the prepared rundown until the point when it is by and by the most elevated need process in the framework.

❖ The SJF calculation bolsters short occupations (or processors) at the drawback of longer ones. The certain issue with SJF is that it requires correct learning of to what degree an occupation or process will run, and this information isn't commonly available. The best SJF count can do is to rely upon customer assessments of run times.

C. SRTF (Shortest Running Time First)

The SRT is the preemtive partner of SJF and valuable in time-sharing condition

❖ In SRTF scheduling, the process with the littlest assessed run-time to fruition is kept running straightaway, including new intoductions

❖ In SJF , once a vocation start executing, it hurries to consummation.

❖ In SJF conspire, a running process might be appropriated by another entry process with most limited assessed run-time.

❖ The calculation SRTF has higher overhead than its partner SJF.

❖ The SRTF must screen the passed time of the running procedure and must deal with intermittent seizures.

❖ In this plan, entry of little process will run very quickly. Be that as it may, longer employments have considerably longer mean holding up time.

D. ROUND ROBIN

Round Robin

One of the set up , least difficult, most attractive and most comprehensively utilized calculation is round robin (RR).

❖ In the round robin scheduling, forms are dispatched in a FIFO way yet are given a prohibitive measure of CPU time called a period cut or a quantum.

❖ If a procedure does not finish before its CPU-time slips , the CPU is seized and given to the following process holding up in ready queue. The seized process is then set at the back of the prepared rundown.

❖ Round Robin Scheduling is preemptive (at the complete of time-cut) thusly it is ground-breaking in time-sharing circumstances in which the system needs to guarantee sensible response times for savvy customers.

❖ The just fascinating issue with round robin plot is the length of the quantum. Setting the quantum too short causes an excessive number of setting switches and lower the CPU effectiveness. Then again, setting the quantum too long may cause poor reaction time and appoximates FCFS.

❖ In any occasion, the normal holding up time under round robin scheduling is regularly very long.

4. Literature survey (for minimum 20 recent year papers taken from reputed journals)

Over a period of time, researchers have developed a number of CPU scheduling mechanisms which have been used for predictable allocation of CPU. Some of the important works are listed below.

IRR

In one proposed Improved Round Robin (IRR) CPU arranging count works like Round Robin (RR) with a little change . IRR picks the primary methodology from the ready queue and allocate the CPU to it for a period break of up to 1 time quantum After finish of process' opportunity quantum, it checks the rest of the CPU burst time of the at present running process. In the event that the rest of the CPU burst time of the right now running procedure is under 1 time quantum,

the CPU again distributed to the as of now running process for outstanding CPU burst time. For this situation this procedure will complete execution and it will be expelled from the prepared line. The scheduler at that point continues to the following procedure in the prepared line. Something else, if the rest of the CPU burst time of the right now running procedure is longer than 1 time quantum, the procedure will be put at the tail of the prepared line. The CPU scheduler will then choose the following procedure in the prepared line.

AAIRR

An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm centers around enhancing more on the enhanced Round Robin CPU scheduling algorithm. The algorithm diminishes the waiting up time and turnaround time definitely contrasted with the basic Round Robin. This proposed calculation works comparably as yet with some adjustment. It works in three phases:

Stage 1: It picks the principal procedess that touches base to the prepared line and assigns the CPU to it for a period interim of up to 1 time quantum. After finishing of process' opportunity quantum, it checks the rest of the CPU burst time of the as of now running procedess. On the off chance that the rest of the CPU burst time of the as of now running process is less or equivalent to 1 time quantum, the CPU is again dispensed to the presently running process for outstanding CPU burst time. For this situation this process will complete execution and it will be expelled from the prepared line. The scheduler at that point continues to the following most limited process in the ready queue. Something else, if the rest of the CPU burst time of the as of now running procedure is longer than 1 time quantum, the procedure will be put at the tail of the prepared line.

Stage 2: The CPU scheduler will then choose the following most limited process in the prepared line, and do the procedure in stage 1.

Stage 3: For the entire execution of the considerable number of process, organize 1 and Stage 2 must be rehashed.

Some other proposed counts starting late in like manner consolidate A New Proposed Two Processor

Based CPU Scheduling Algorithm with Varying Time Quantum for Real Time Systems, and Burst Round Robin (BRR)

5. Gaps identified and observations made from the literature survey

Algorithms for example, IRR, AAAIRR, BRR and A New Proposed Two Processor Based CPU Scheduling Algorithm with Varying Time Quantum for Real Time Systems were great as far as advancing the round robin calculations..

They truly gave lesser waiting time and turnaround time when contrasted and unique round-robin calculation in every single conceivable instance of procedures: burst time in expanding request, diminishing request or in any arbitrary request which made them valuable.

Algorithm	Average Waiting Time(ms)	Average Turnaround Time (ms)	Number of Context Switch
RR	38.4	57.8	10
IRR	30.4	49.8	7
AAIRR	26.4	45.8	6

However, one thing was regular for every one of them: they all were utilizing static quantum time for roundrobin planning. They were not utilizing dynamic quantum time for round robin-booking. Utilizing static or settled quantum time is useful for just a few cases. Yet, when forms have immense contrast in their burst timing, utilizing a settled quantum time may not be useful in limiting the pausing and turnaround time. In such cases, if the time cut is progressively utilized for each procedure, it might be exceptionally useful in advancing the planning

6. Proposed model – explain your model with architecture

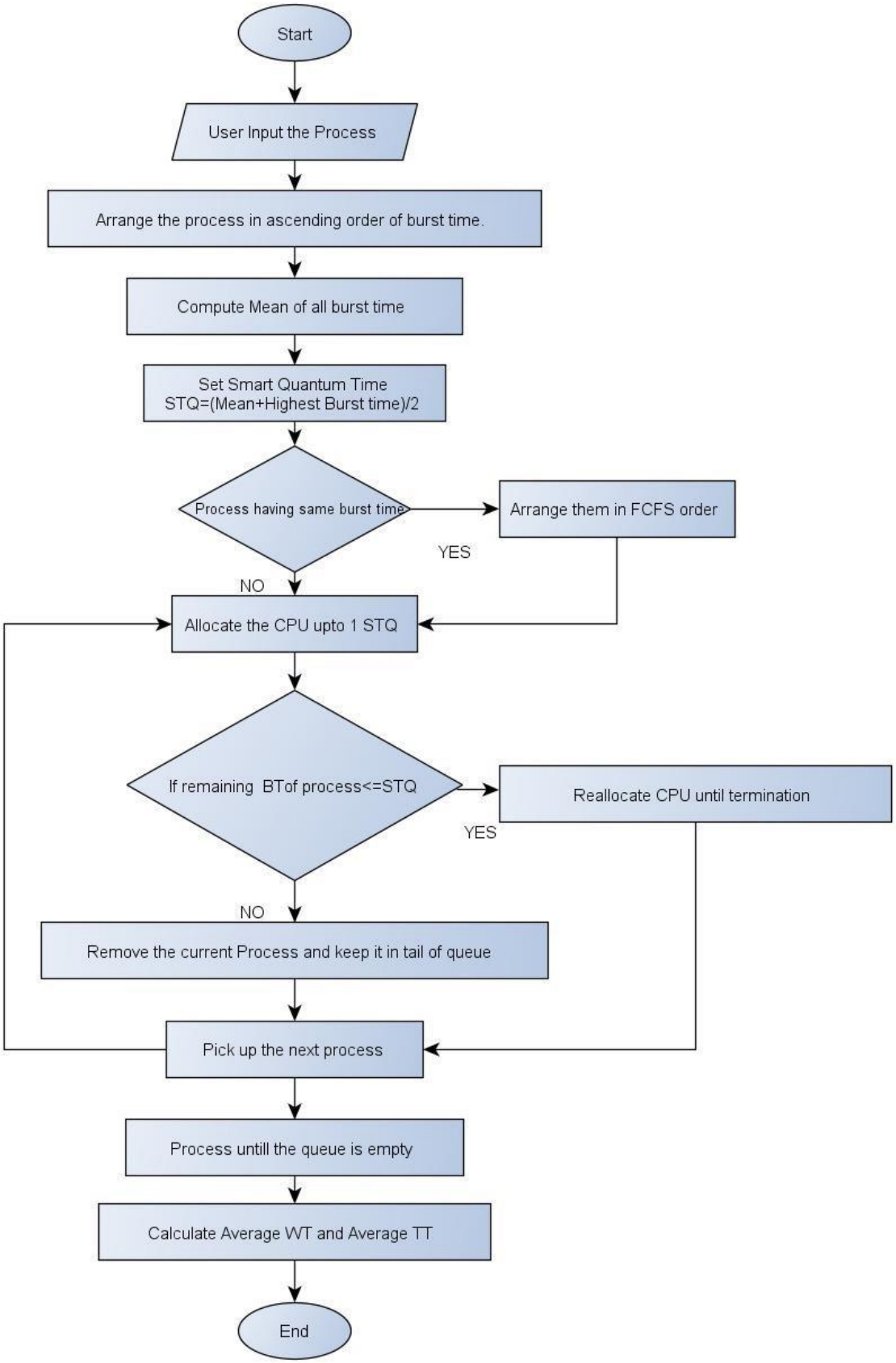
The proposed algorithm centers around the change on CPU scheduling algorithmn. The calculation diminishes the waiting time and turnaround time definitely contrasted with the other Scheduling calculation and straightforward Round Robin booking calculation. This proposed calculation works comparably as yet with some adjustment. It executes the most limited activity having least blasted time first rather than FCFS basic Round robin calculation and it additionally utilizes Smart time quantum rather than static time quantum. Rather than giving static time quantum in the CPU scheduling time, our calculation ascertains the Smart time quantum itself as per the burst time all things considered. The proposed calculation disposes of the inconsistencies of actualizing basic round robin engineering.

• In the main phase of the inventive calculation CPU scheduling algorithm every one of the procedures are organized in the expanding request of CPU burst time. It implies it naturally relegate the need to the procedures. Process having low blasted time has high need to the procedure have high blasted time.

• Then in the second stage the calculation figures the mean of the CPU burst time of the number of process Subsequent to ascertaining the mean, it will set the time quantum progressively i.e. (normal of mean and most elevated burst time)/2.

• Then in the last stage algorithm pick the principal process from the prepared line and dispense the CPU to the procedure for a period interim of up to 1 Smart time quantum. In the event that the rest of the burst time of the present running procedure is under 1 Smart time quantum then calculation again dispense the CPU to the Current procedure till it execution. After execution it will expel the ended procedure from the prepared line and again go to the stage 3.

•**The flowchart for proposed algorithm appeares beneath**:



Stage 1: START

Stage 2: Arrange the process in the expanding request of CPU burst time in the ready queue.

Stage 3: Calculate the Mean of the CPU burst time of the total number of Processes

Mean (M)=(P1+P2+P3… … .Pn)/n;

Stage 4: Set the Smart time quantum (STQ) as indicated by the accompanying technique

STQ=(Mean+ Highest burst time)/2;

Stage 5: Allocate the CPU to the First process in the Ready Queue for the age of 1 Smart Time Quantum.

Stage 6: If the rest of the CPU burst time of the present process is less at that point or equivalent to 1 time quantum. Reallocate the CPU to current process again for the rest of the burst time. After the total execution of the present process, expel it from the ready queue.

o Otherwise expel the present process from the ready queue and put it on the tail of the ready queue for further execution.

Stage 7: Pick the following process from the ready queue and give the CPU to it up to the age of 1 Smart time quantum and afterward again to the stage 6.

Stage 8: Process the line until the point when it will be vacant.

Stage 9: Calculate the Average Waiting time and Average Turnaround time of all procedure.

Stage 10: END

Code:

```
#include <stdio.h>
 #include <conio.h>
 #include <stdio.h>
 #include <conio.h>
 #include<math.h>
 #include<string.h>
 int wt[100],bt[100],at[100],tat[100],n,p[100];
 float awt[5],atat[5];
 int temp1,temp2,temp3,sqt,avg;

 void input()
 {
      printf("Enter Number of processes:");
      scanf("%d",&n);
      int i;
         for(i=0;i<n;i++)
                  p[i]=i+1;
      for(i=0;i<n;i++)
      {
        printf("Enter Burst Time of process %d:",i+1);
        scanf("%d",&bt[i]);
        printf("Enter Arrival Time of process %d:",i+1);
        scanf("%d",&at[i]);
      }
         for(i=0;i<5;i++)
         {
                  awt[i]=0.0;
                  atat[i]=0.0;
```

```
        }

}
void changeArrival(){
        int a=at[0];
        int i;
        for(i=0;i<n;i++){
                if(at[i]<a)
                        a=at[i];
}
if(a!=0){
        for(i=0;i<n;i++)
                at[i]=at[i]-a;
}
}
void fcfs(){
  wt[0]=0;
  atat[0]=tat[0]=bt[0];
  int btt=bt[0];
  int i;
  for(i=1;i<n;i++){
    wt[i]=btt-at[i];
    btt+=bt[i];
    awt[0]+=wt[i];
    tat[i]= wt[i]+bt[i];
    atat[0]+=tat[i];
  }
  atat[0]/=n;
  awt[0]/=n;
```

```
   printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");
   for(i=0;i<n;i++)
   {
   printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);
   }
}
void innovative()
{
        int bt1[n],i,j,temp,qt;
        int b[n];
float twt,ttat;
for(i=0;i<n;i++)
        bt1[i]=bt[i];
for(i=0;i<n;i++)
        b[i]=bt[i];
int num=n;
int time=0;
int max;
int sum,t,a,ap;
ap=0;
        //sorting in ascending order
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n-i- 1; j++)
         {
            if (bt[j] > bt[j + 1])
            {
                temp1 = bt[j];
                temp2 = p[j];
```

```
            temp3 = at[j];
            bt[j] = bt[j + 1];
            p[j] = p[j + 1];
            at[j] = at[j + 1];
            bt[j + 1] = temp1;
            p[j + 1] = temp2;
            at[j + 1] = temp3;
        }
      }
  }
max=bt[n-1];
sum=0;
for(i=0;i<n;i++)
{
   sum=sum+bt[i];
}
avg=sum/n;

//printf("\n %d  %d \n",max,avg);

qt=(avg+max)/2;
printf("\nDynamic Quantum time calculated is : %d\n",qt);

   while(num>0){
   a=0;
   max=0;
   sum=0;
   t=0;
   for(i=0;i<n;i++){
```

```
        if(at[i]<=time && b[i]!=0)
        {
                if(b[i]<qt)
                {
                        t+=b[i];
                        b[i]=0;
                }
                else
                {
                        t+=qt;
                        b[i]-=qt;
                }
                if(b[i]<qt && b[i]!=0)
                {
                        t+=b[i];
                        b[i]=0;
                }
                if(b[i]==0){
                        wt[i]=(time+t)-bt1[i];
                        tat[i]=time+t;
                        num--;
                }
        }
}
time+=t;
}
printf("Processes\tWaitingtime\tTurnAroundTime\n");
for(j=1;j<=n;j++)
{
```

```
            for(i=0;i<n;i++)
            {
               if(j==p[i])
                                    printf("process %d\t%d\t\t%d\n",p[i],wt[i],tat[i]);
                           }
                  }

      twt=0;
      ttat=0;
         for(i=0;i<n;i++)
                  {twt=twt+wt[i];}
         awt[4]=twt/n;
         for(i=0;i<n;i++)
                  {ttat=ttat+tat[i];}
         atat[4]=(ttat/n);

}
 void rr()
 {
int i, total = 0, x, counter = 0, time_quantum;
         int wait_time = 0, turnaround_time = 0, temp[100];
x=n;
for(i = 0; i < n; i++)
         {
         temp[i] = bt[i];
         }
         printf("\nEnter Time Quantum:\t");
         scanf("%d", &time_quantum);
   printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
```

```
	for(total = 0, i = 0; x != 0;)
	{
		if(temp[i] <= time_quantum && temp[i] > 0)
		{
			total = total + temp[i];
			temp[i] = 0;
			counter = 1;
		}
		else if(temp[i] > 0)
		{
			temp[i] = temp[i] - time_quantum;
			total = total + time_quantum;
		}
		if(temp[i] == 0 && counter == 1)
		{
			x--;
				printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n", i + 1, bt[i], total -
at[i], total - at[i] - bt[i]);
			wait_time = wait_time + total - at[i] - bt[i];
			turnaround_time = turnaround_time + total - at[i];
			counter = 0;
		}
		if(i == n - 1)
		{
		i = 0;
	}
		else if(at[i + 1] <= total)
		{
		i++;
```

```
        }
                else
                {
                i = 0;
        }
}
awt[2] = wait_time * 1.0 / n;
atat[2] = turnaround_time * 1.0 / n;
 }

 void srtf()
 {
        int i,j,x[10],b[10],count=0,time,smallest;
        int avg1=0,tt1=0,end1;
        int bt2[100];
        for(i=0;i<n;i++)
        bt2[i]=bt[i];
 for(i=0;i<n;i++)
 {
x[i]=bt2[i];
 }
 b[9]=9999;

 for(time=0;count!=n;time++)
 {
   smallest=9;
  for(i=0;i<n;i++)
  {
   if(at[i]<=time && bt2[i]<b[smallest] && bt2[i]>0 )
```

```
 smallest=i;
 }
 bt2[smallest]--;
 if(bt2[smallest]==0)
 {
  count++;
  end1=time+1;
  avg1=avg1+end1-at[smallest]-x[smallest];
  tt1= tt1+end1-at[smallest];
 }
}
awt[3]=avg1/n;
atat[3]=tt1/n;
printf("Processes\tWaitingtime\tTurnAroundTime\n");
        for(j=1;j<=n;j++)
        {
           for(i=0;i<n;i++)
           {
              if(j==p[i])
                                printf("process %d\t%d\t\t%d\n",p[i],wt[i],tat[i]);
                        }
                   }

}

 void display(int c)
 {
   int i;
   printf("Average Waiting Time: %f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);
```

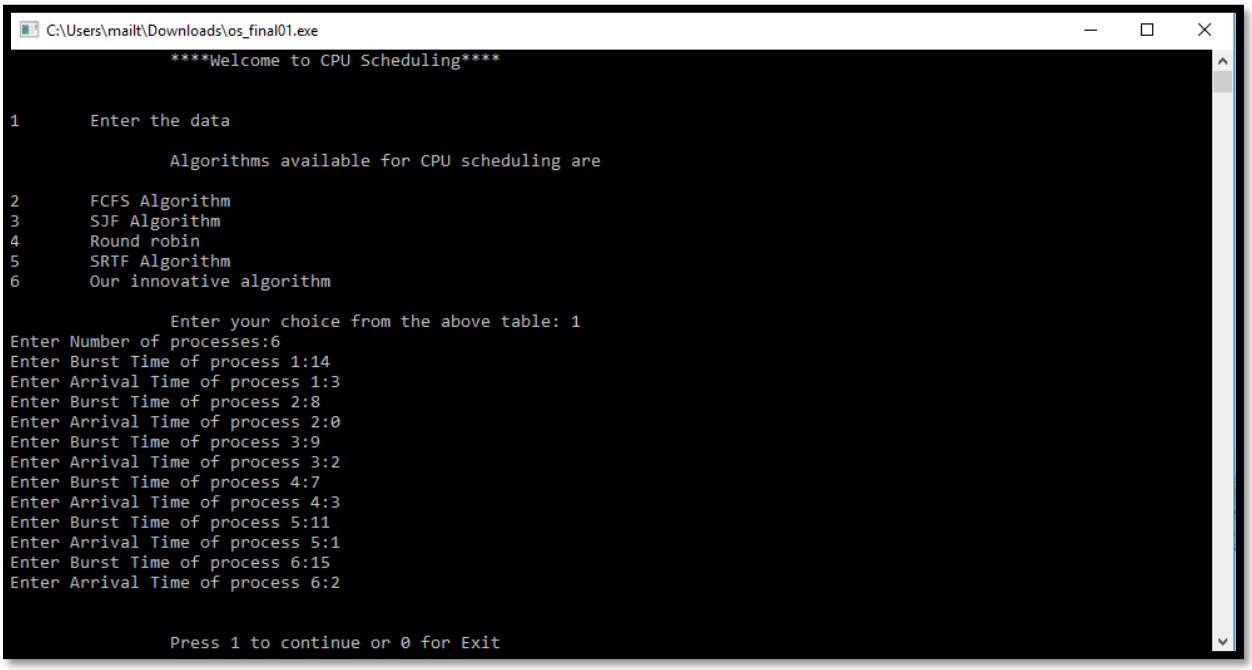
```
 }
void sjf()
{
float wavg=0,tavg=0,tsum=0,wsum=0;
        int i,j,temp,sum=0,ta=0;
 for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
        if(at[i]<at[j])
        {
        temp=p[j];
        p[j]=p[i];
        p[i]=temp;

        temp=at[j];
        at[j]=at[i];
        at[i]=temp;

        temp=bt[j];
        bt[j]=bt[i];
        bt[i]=temp;
        }
}
}
 int btime=0,min,k=1;
for(j=0;j<n;j++)
{
btime=btime+bt[j];
```

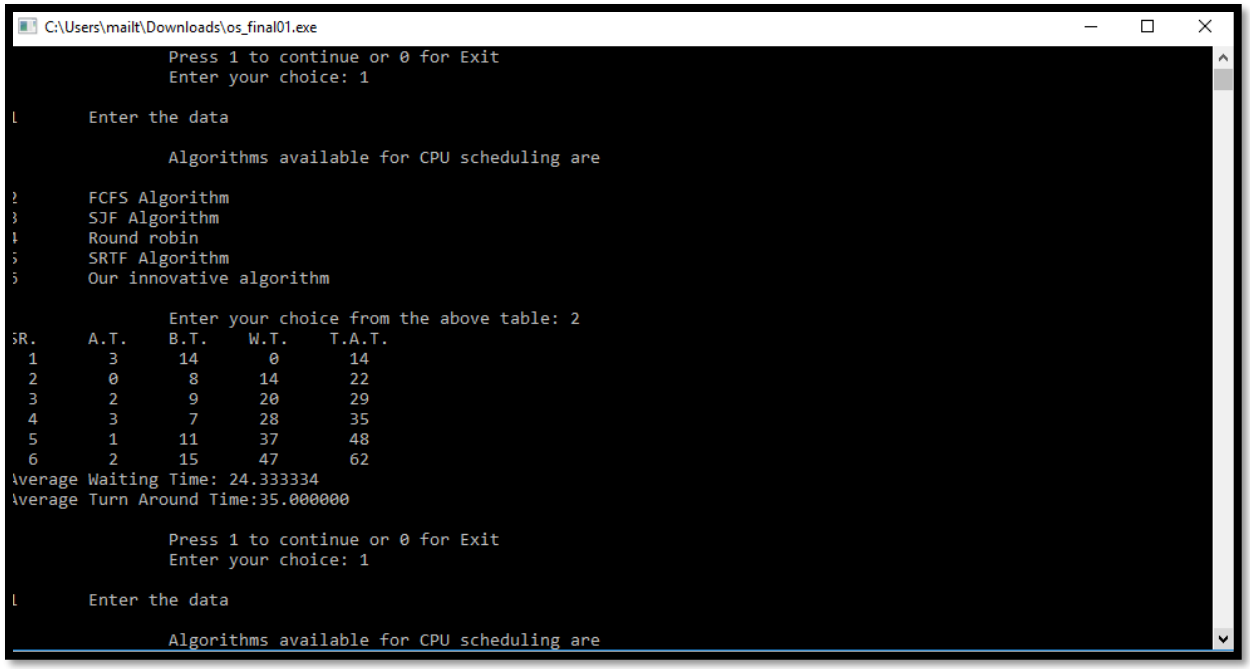
```
min=bt[k];
for(i=k;i<n;i++)
{
if (btime>=at[i] && bt[i]<min)
{
temp=p[k];
p[k]=p[i];
p[i]=temp;
temp=at[k];
at[k]=at[i];
at[i]=temp;
temp=bt[k];
bt[k]=bt[i];
bt[i]=temp;
}
}
k++;
}
wt[0]=0;
for(i=1;i<n;i++)
{
sum=sum+bt[i-1];
wt[i]=sum-at[i];
wsum=wsum+wt[i];
}

awt[1]=(wsum/n);
for(i=0;i<n;i++)
{
```

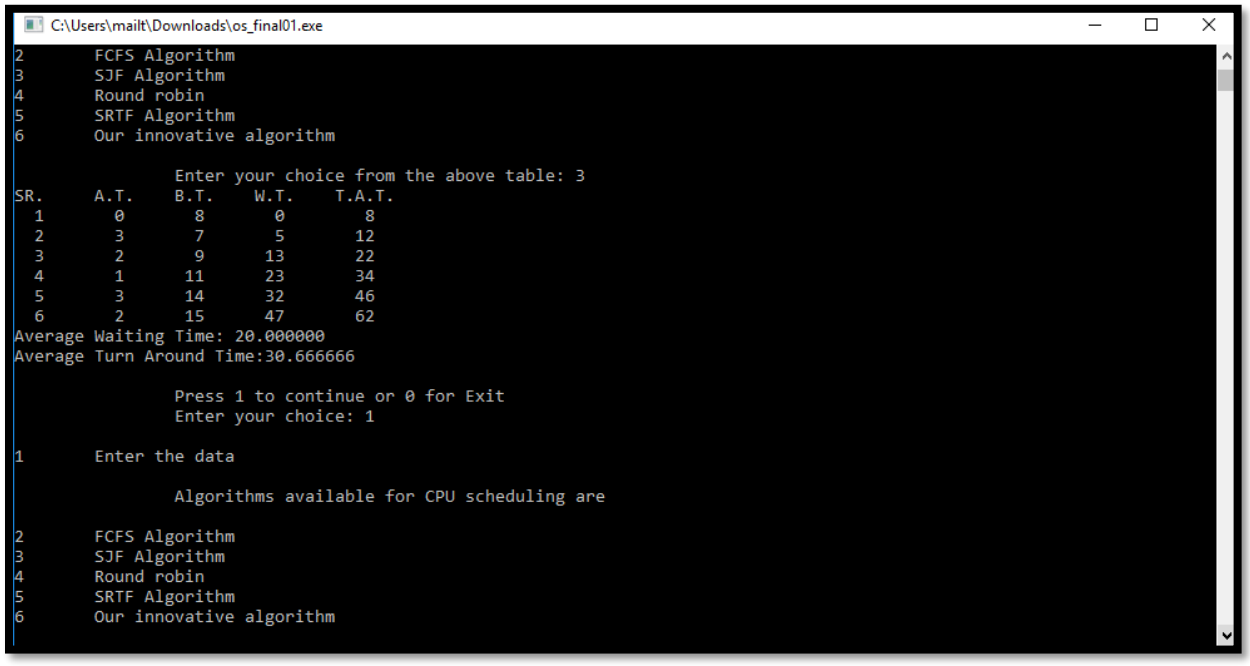
```
ta=ta+bt[i];
tat[i]=ta-at[i];
tsum=tsum+tat[i];
}

atat[1]=(tsum/n);
printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");
    for(i=0;i<n;i++)
    {
    printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);
    }
 }
 int main(){
       printf("\t\t****Welcome to CPU Scheduling****\n\n");
   int c,choice;
   do
   {
       printf("\n1\tEnter the data \n\n\t\tAlgorithms available for CPU scheduling are\n \n2\tFCFS Algorithm\n3\tSJF Algorithm\n4\tRound robin\n5\tSRTF Algorithm\n6\tOur innovative algorithm\n");
   printf("\n\t\tEnter your choice from the above table: ");
   scanf("%d",&c);
       switch(c)
       {
        case 1:input();changeArrival(); break;
            case 2:fcfs();display(c);break;
        case 3:sjf();display(c);break;
            case 4:rr();display(c);break;
            case 5:srtf();display(c);break;
            case 6:innovative();display(c);break;
```

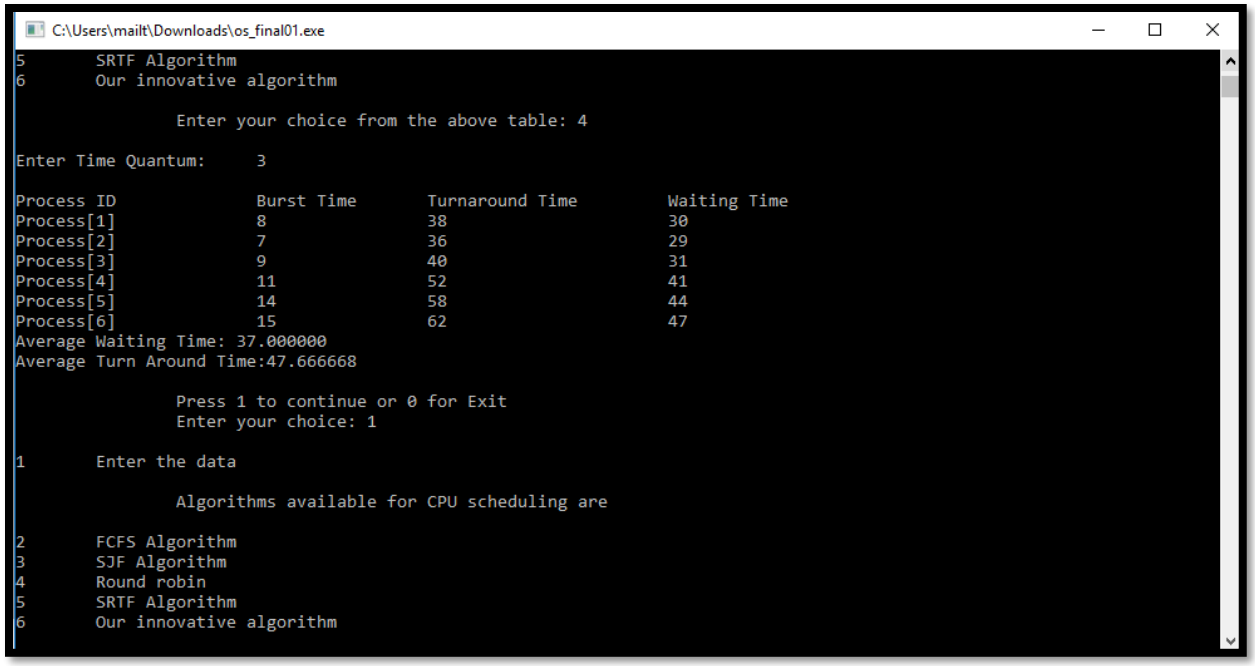
```
        default: printf("Please enter choice from 1 to 6 only\n");break;
  }
    printf("\n\n\t\tPress 1 to continue or 0 for Exit\n\t\tEnter your choice: ");
    scanf("%d",&choice);
    if(choice==0)
     printf("\nThank You!\n");
                }while(choice==1);

}
```

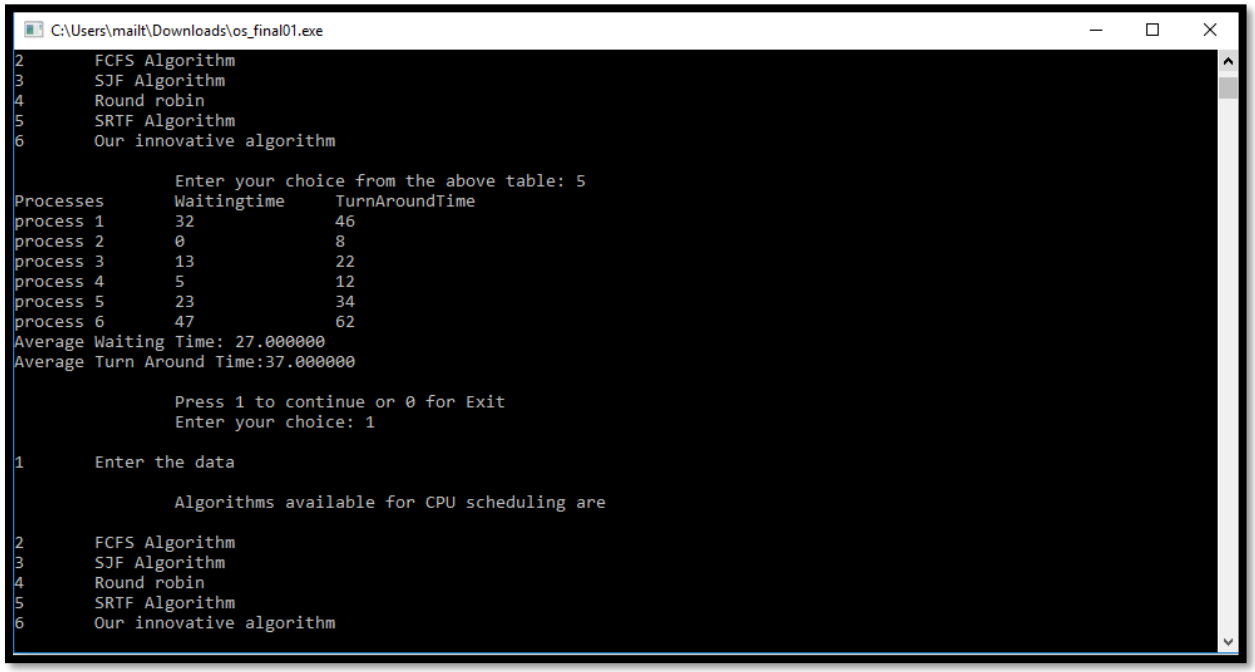
7. Results obtained

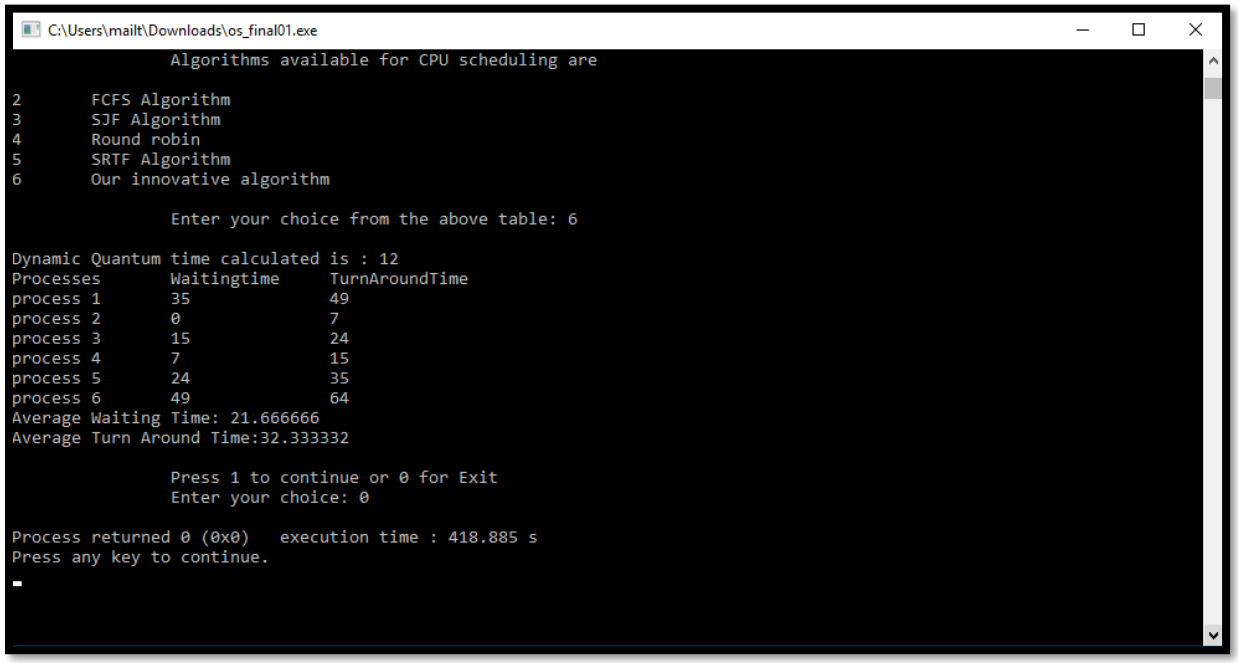



C:\Users\mailt\Downloads\os_final01.exe
Press 1 to continue or 0 for Exit
Enter your choice: 1
Enter the data
Algorithms available for CPU scheduling are
FCFS Algorithm
SJF Algorithm
Round robin
SRTF Algorithm
Our innovative algorithm
Enter your choice from the above table: 2
SR. A.T. B.T. W.T. T.A.T.
1 3 14 0 14
2 0 8 14 22
3 2 9 20 29
4 3 7 28 35
5 1 11 37 48
6 2 15 47 62
Average Waiting Time: 24.333334
Average Turn Around Time:35.000000
Press 1 to continue or 0 for Exit
Enter your choice: 1
Enter the data
Algorithms available for CPU scheduling are


C:\Users\mailt\Downloads\os_final01.exe
FCFS Algorithm
SJF Algorithm
Round robin
SRTF Algorithm
Our innovative algorithm
Enter your choice from the above table: 3
SR. A.T. B.T. W.T. T.A.T.
1 0 8 0 8
2 3 7 5 12
3 2 9 13 22
4 1 11 23 34
5 3 14 32 46
6 2 15 47 62
Average Waiting Time: 20.000000
Average Turn Around Time:30.666666
Press 1 to continue or 0 for Exit
Enter your choice: 1
Enter the data
Algorithms available for CPU scheduling are
FCFS Algorithm
SJF Algorithm
Round robin
SRTF Algorithm
Our innovative algorithm


C:\Users\mailt\Downloads\os_final01.exe
5 SRTF Algorithm
6 Our innovative algorithm
Enter your choice from the above table: 4
Enter Time Quantum: 3
Process ID Burst Time Turnaround Time Waiting Time
Process[1] 8 38 30
Process[2] 7 36 29
Process[3] 9 40 31
Process[4] 11 52 41
Process[5] 14 58 44
Process[6] 15 62 47
Average Waiting Time: 37.000000
Average Turn Around Time:47.666668
Press 1 to continue or 0 for Exit
Enter your choice: 1
1 Enter the data
Algorithms available for CPU scheduling are
2 FCFS Algorithm
3 SJF Algorithm
4 Round robin
5 SRTF Algorithm
6 Our innovative algorithm


C:\Users\mailt\Downloads\os_final01.exe
2 FCFS Algorithm
3 SJF Algorithm
4 Round robin
5 SRTF Algorithm
6 Our innovative algorithm
Enter your choice from the above table: 5
Processes Waitingtime TurnAroundTime
process 1 32 46
process 2 0 8
process 3 13 22
process 4 5 12
process 5 23 34
process 6 47 62
Average Waiting Time: 27.000000
Average Turn Around Time:37.000000
Press 1 to continue or 0 for Exit
Enter your choice: 1
1 Enter the data
Algorithms available for CPU scheduling are
2 FCFS Algorithm
3 SJF Algorithm
4 Round robin
5 SRTF Algorithm
6 Our innovative algorithm



8. Performance evaluation-Parameters identified for evaluation of proposed model

Our proposed algorithm endeavors to advance round – robin calculation. Parameters for assessment of our model are the essential planning criteria which are utilized for assessing proficiency of any scheduling algorithm.

There are different criteria to check the execution of CPU scheduling algorithm. These different criteria are utilized for looking at the different CPU scheduling algorithm. The criteria incorporate the accompanying:

1. **Waiting Time**: The normal timeframe a procedure spends pausing. Holding up time might be communicated as turnaround time less the genuine execution time.

2. **Turnaround time**: The interim from the season of accommodation of a procedure to the season of finishing is the turnaround time.

Comparison with existing model:

Our algorithm (AMRR) performs way better than the existing round robin algorithm and some other variations previously made in the round robin algorithm like IRR and AAIRR. Which unlike our AMRR used static quantum time.

Our algorithm is suitable for real-life problems when processes arrive almost altogether and have large burst times. Therefore, significant difference between the computational effectivenss of our algorithm and others is possible when the following conditions are met:

- Processes have close arrival time
- Processes have large burst time.
- Number of processes should be more.

Our code has been created for handling around a hundred number of processes, but it can be further extended if needed.

9. References (in APA format)

[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, June 2012.Neetu

[3] Weiming Tong and Jing Zhao, "Quantum Varying Deficit Round Robin Scheduling Over Priority Queues", International Conference on Computational Intelligence and Security, pp. 252-
256, China, 2007

[4] Ajit, S, Priyanka, G and Sahil, B (2010): An Optimized Round Robin Scheduling Algorithm for CPU Scheduling, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No. 07, 2383-2385, pp 2382-2385.

[5] Ishwari, S. R and Deepa, G (2012): A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems, International Journal of Innovations in Engineering and Technology (IJIET), ISSN: 2319 – 1058, Vol. 1 Issue 3, pp 1-11.

[6] Helmy, T. and A. Dekdouk, “Burst Round Robin as a Proportional-share Scheduling Algorithm”, IEEEGCC, http://eprints.kfupm.edu.sa/1462/, 2007.

[7] H.S. Behera, Jajnaseni Panda, Dipanwita Thakur and Subasini Sahoo, “A New Proposed Two Processor Based CPU Scheduling Algorithm with Varying Time quantum for Real Time Systems”, Journal of Global Research in Computer Science, Vol. 2, No. 4, April 2011, pp. 81-87.