**SCHEDULING ALGORITHM**

**ABSTRACT:**

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 finishing time, waiting time, total time taken for the processes to

complete, number of rounds, etc are calculated.

**KEYWORDS:**

Scheduling

Preemptive

Non preemptive

Finishing time

Waiting time

**INTRODUCTION**

Multiprogramming is one of the most important aspects of operating systems. It requires several processes to be kept simultaneously in memory, the aim of which is maximum CPU utilization. If these several processes in the memory are ready to run at the same time, the operating system must choose which one among them to run first. Making this decision is CPU scheduling. CPU scheduling is the basis of multiprogramming systems. It refers to a set of policies and mechanisms to control the order of work to be performed by a computer system. It is made by the part of the operating system called the scheduler, using a CPU scheduling algorithm [9].

CPU scheduling is the way by which a resource is assigned to a process or task. For scheduling task, many scheduling algorithms are used such as FCFS, SJF, RR, and Priority CPU scheduling algorithm. The processes are scheduled as per to the given arrival time, burst time and priority. The execution of processes needs number of resources such as Memory, CPU time etc. [1]. A scheduling decision implies to the theory of selecting the next process to execute. In each scheduling decision, a context switch may occur, meaning that the current process will stop executing and put back to the ready queue (or some other place) and another process will be dispatched. We define the scheduling overhead cost which includes context switching time. Since processes are switching the so CPU performance will be decreased. Scheduling algorithms are extensively used in communication networks and in operating systems to assign resources to competing processes.

*CPU scheduling algorithms*

The fundamental CPU scheduling algorithms are First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR) and Priority Scheduling (PS). The FCFS scheduling is the simplest CPU scheduling algorithm, which allocates CPU to the processes on the basis of their arrival to the ready queue. Arriving processes are inserted into the tail (rear) of the ready queue and the process to be executed next is removed from the head (front) of the ready queue. A long CPU-bound process may dominate the CPU and may force shorter CPU bound processes to wait prolonged periods. In the SJF, the scheduler arranges processes according to shortest burst times in the ready queue, so that the process with least burst time is scheduled first. If two processes have equal burst times, then FCFS procedure is followed. Long running processes may wait for prolonged periods, because the CPU has a steady supply of short processes. It has been proven to be the fastest scheduling algorithm, but it suffers from one important problem: How does the scheduler know how long the next

CPU burst is going to be? [7]. The PS associates each process with a priority number. The CPU is allocated to the process with the highest priority. If there are multiple processes with same priority, then FCFS will be used to allocate the CPU. Lower priority processes may starve, because the CPU may have a steady supply of higher priority processes. Round Robin (RR) is specially designed for time-sharing systems; each process gets a small unit of CPU time (time quantum). This algorithm will allow the first process in the ready queue to run until its time quantum expires, and then run the next process in the ready queue. In a situation where the process needs more time, the process runs for the full length of the time quantum and then it is preempted and then added to the tail of the queue.

*SCHEDULING CRITERIA*

Different CPU scheduling algorithms have its own properties. The choice of a specific algorithm may favor one class of processes over another. For choice of an algorithm for a particular situation, the characteristics of various algorithms must be taken in consider [4]. Many criteria have been proposed for comparing CPU scheduling algorithms. These characteristics are used to compare and to make a significant difference in which algorithm is judged to be the best one. The criteria are the following:

1. Context Switch Time: Time taken in the process of storing and restoring context of a pre-empted process, so that execution can be resumed from exactly same point at a later time.

2. Throughput: Number of processes completed per unit time.

3. CPU Utilization: This is a measure of how much time CPU is busy.

4. Turnaround Time: Total time taken by the CPU to execute a process from time of entering.

5. Waiting Time: It is the total time a process has been in waiting state.

6. Response Time: Time at which a process get the CPU at first time from the time of entering.

So, a good scheduling algorithm should have the following features [2]:

 Maximum throughput.

 Maximum CPU utilization.

 Minimum context switches.

 Minimum turnaround time.

 Minimum waiting time.

 Minimum response time.

In this project we will examine several popular CPU scheduling algorithms by means of analysis and comparing their results under different workloads with C.

**2. LITERATURE REVIEW**

Various improvements to RR CPU scheduling algorithm have been proposed by many authors. These modifications can be classified as follows:

*Statically allocated time quantum*

Ajit et al [3] proposed an algorithm that allocates the CPU to every process in RR fashion for an initial time quantum (say k units). After completing first cycle, it doubles the initial time quantum (2k units) and allocates the CPU to the processes in SJF format. It alternates the doubling and halving of the time quantum if processes remain in the ready queue after completing any execution cycle.

Ishwari and Deepa [5] proposed an algorithm that allocates the CPU to every process in RR fashion for only one time quantum. The CPU is then allocated to the remaining processes in the ready queue after completion of the execution in SJF fashion.

Manish and AbdulKadir [6] proposed an algorithm that allocates the CPU to processes in RR fashion. After executing each process for one time quantum, it checks if the remaining burst time of the currently running process is less than the time quantum. If so, it allocates the CPU to the process for the remaining burst time, else it moves the process to the tail of the ready queue.

*Dynamically Determined Time Quantum:*

Behera et al [10] suggested an algorithm that sorts the processes in the ready queue in ascending order of burst time and time quantum is calculated. To get an optimal time quantum, it takes the median of the burst time of the processes in the ready queue. The time quantum is recalculated using remaining burst time of the processes in each cycle.

Lalit et al [11] suggested an algorithm that sort the processes in ascending order of burst time and put time quantum for RR equals to the average of the burst times of the processes. This algorithm assumes that all processes are arrived at the time t=0.

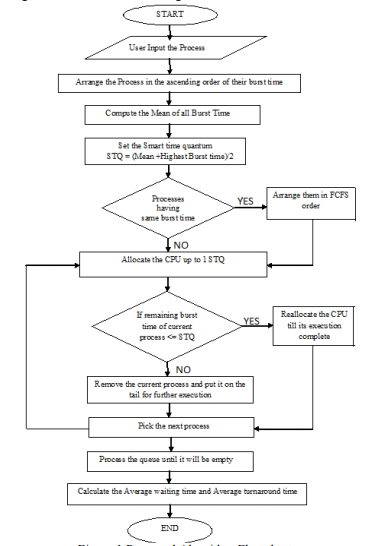
Soraj and Roy [8] suggested an algorithm that sort the processes in ascending order of burst time, and then it calculate the smart time slice (STS), which is mostly dependant on the number of processes. STS is equal to the burst time of the mid process. process when number of processes is odd and average of the processes burst times when the number of processes is even. This algorithm assumes that all processes arrive at the time t=0.

**3. Proposed Method/System**

The proposed algorithm focuses on the improvement on CPU scheduling algorithm. The algorithm reduces the waiting time and turnaround time drastically compared to the other Scheduling algorithm and simple Round Robin scheduling algorithm. This proposed algorithm works in a similar way as but with some modification. It executes the shortest job having minimum burst time first instead of FCFS simple Round robin algorithm and it also uses Smart time quantum instead of static time quantum. Instead of giving static time quantum in the CPU scheduling algorithms, our algorithm calculates the Smart time quantum itself according to the burst time of all processes. The proposed algorithm eliminates the discrepancies of implementing simple round robin architecture.

* In the first stage of the innovative algorithm CPU scheduling algorithms all the processes are arranged in the increasing order of CPU burst time. It means it automatically assign the priority to the processes. Process having low burst time has high priority to the process have high burst time.
* Then in the second stage the algorithm calculates the mean of the CPU burst time of all the processes. After calculating the mean, it will set the time quantum dynamically i.e. (**average of mean and highest burst time)/2**.
* Then in the last stage algorithm pick the first process from the ready queue and allocate the CPU to the process for a time interval of up to 1 Smart time quantum. If the remaining burst time of the current running process is less than 1 Smart time quantum then algorithm again allocate the CPU to the Current process till it execution. After execution it will remove the terminated process from the ready queue and again go to the stage 3.

**4. FLOWCHART**



**5. IMPLEMENTATION:**

We have compared the proposed algorithm with another known algorithms manually using GANTT CHART and also implemented using C PROGRAMMING.

**CODE FOR IMPLEMENTATION:**

#include <stdio.h>

#include<math.h>

#include<string.h>

int wt[100],bt[100],at[100],tat[100],n,p[100];

float awt[5],atat[5];

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,tq;

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=0;

int sum,t,a,ap;

ap=0;

while(num>0){

a=0;

max=0;

sum=0;

t=0;

//sorting in ascending order

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

for(j=0;j<n;j++){

if(b[i]<b[j] && at[i]<=time){

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=b[j];

b[j]=b[i];

b[i]=temp;

temp=bt1[j];

bt1[j]=bt1[i];

bt1[i]=temp;

}}

}

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

if(at[i]<=time && b[i]!=0){

a++;

if(b[i]>max)

max=b[i];

sum+=b[i];

}}

if(a!=ap){

tq=sqrt((sum/a)\*max);

ap=a;

}

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

if(at[i]<=time && b[i]!=0)

{

if(b[i]<tq)

{

t+=b[i];

b[i]=0;

}

else

{

t+=tq;

b[i]-=tq;

}

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]);

}

}

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

twt+=wt[i];

awt[4]=twt/n;

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

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,x[10],b[10],count=0,time,smallest;

double avg=0,tt=0,end;

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

{

x[i]=bt[i];

}

bt[9]=9999;

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

{

smallest=9;

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

{

if(at[i]<=time && bt[i]<b[smallest] && bt[i]>0 )

smallest=i;

}

bt[smallest]--;

if(bt[smallest]==0)

{

count++;

end=time+1;

avg=avg+end-at[smallest]-x[smallest];

tt= tt+end-at[smallest];

}

}

awt[3]=avg/n;

atat[3]=tt/n;

}

void display(int c)

{

int i;

printf("Average Waiting Time: %f\nAverage Turn Around Time:%f",awt[c-1],atat[c-1]);

}

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("Welcome to CPU Scheduling:\n\n");

input();

int c,choice;

changeArrival();

printf("Choice\tAlgorithm used\n1\tFCFS Algorithm\n2\tSJF Algorithm\n3\tRound robin\n4\tSRTF Algorithm\n5\tOur innovative algorithm\n");

do

{

printf("Enter your choice from the above table");

scanf("%d",&c);

switch(c)

{

case 1:fcfs();break;

case 2:sjf();break;

case 3:rr();break;

case 4:srtf();break;

case 5:innovative();break;

default: printf("Please enter choice from 1 to 5 only\n");break;

}

display(c);

printf("\n\nEnter 1 to continue 0 to stop");

scanf("%d",&choice);

}while(choice==1);

int a[5][2],i;

for(i=0;i<5;i++)

{

a[i][0]=awt[i];

a[i][1]=atat[i];

}

}

**RESULT AND DISCUSSION:**

**CPU Burst Time in Increasing Order**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Average | Average | Number |
|  | Waiting | Turnaround | of Context |
|  | Time(ms) | Time (ms) | Switch |
| FCFS | 38.4 | 57.8 | 10 |
| RR | 30.4 | 49.8 | 7 |
| SJF | 26.4 | 45.8 | 6 |
| XX | 24.4 | 43.7 | 4 |

0 0 2 10 16 24 0 0 6 14 0 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P2 | P3 | P4 | P5 | P4 | P5 |

0 5 15 25 35 45 47 57 67 77 83 97

Gantt chart representation of FCFS

0 0 0 10 16 24 0 0 0



P1

P2

P3

P4

P5

P3

P4

P

5

0 5 17 27 37 47 57 73 97

Gantt chart representation of RR

0 0 0 0 16 24 0 0



P1

P2

P3

P4

P5

P4

P5

0 5 17 37 47 57 73 97

Gantt chart representation of SJF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 |

0 5 17 37 63 97

Gantt chart representation of XX

***CPU Burst Time in Decreasing Order:***

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Average | Average | Number |
|  | Waiting | Turnaround | of |
|  | Time(ms) | Time (ms) | Context Switch |
| FCFS | 58 | 77.4 | 11 |
| RR | 49 | 68.4 | 8 |
| SJF | 34.4 | 53.8 | 7 |
| XX | 24.4 | 43.79 | 4 |

* 1. 24 16 10 2 0 14 6 0 0 4 0 0



P1

P2

P3

P4

P5

P1

P2

P3

P4

P1

P2

P1

0 10 20 30 40 45 55 65 75 77 87 93 97

Gantt chart representation of FCFS

0 24 16 10 0 0 14 0 0 0



P1

P2

P3

P4

P5

P1

P2

P3

P1

0 10 20 30 42 47 57 73 83 97

Gantt chart representation of RR

0 24 0 0 0 16 14 0 0



P1

P5

P4

P3

P2

P1

P2

P1

0 10 15 27 47 57 67 81 97

Gantt chart representation of SJF

0 0 0 0 0 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P5 | P4 | P3 | P2 | P1 |

0 5 17 37 63 97

Gantt chart representation of XX

***CPU Burst Time in Random Order:***

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Average | Average | Number of |
|  | Waiting | Turnaround | Context |
|  | Time(ms) | Time (ms) | Switch |
| FCFS | 48 | 67.4 | 11 |
| RR | 40.4 | 59.8 | 8 |
| SJF | 31 | 50.4 | 6 |
| XX | 24.4 | 43.7 | 4 |

1. 10 24 0 2 16 0 14 0 6 4 0 0



P1

P2

P3

P4

P5

P1

P2

P3

P5

P2

P5

P2

0 10 20 25 35 45 55 65 67 77 87 93 97

Gantt chart representation of FCFS

0 10 24 0 0 16 0 14 0 0



P1

P2

P3

P4

P5

P1

P2

P5

P1

0 10 20 30 42 47 57 73 83 97

Gantt chart representation of RR

0 0 0 0 16 24 0 0



P1

P3

P4

P5

P2

P5

P2

0 20 25 37 47 57 73 97

Gantt chart representation of SJF

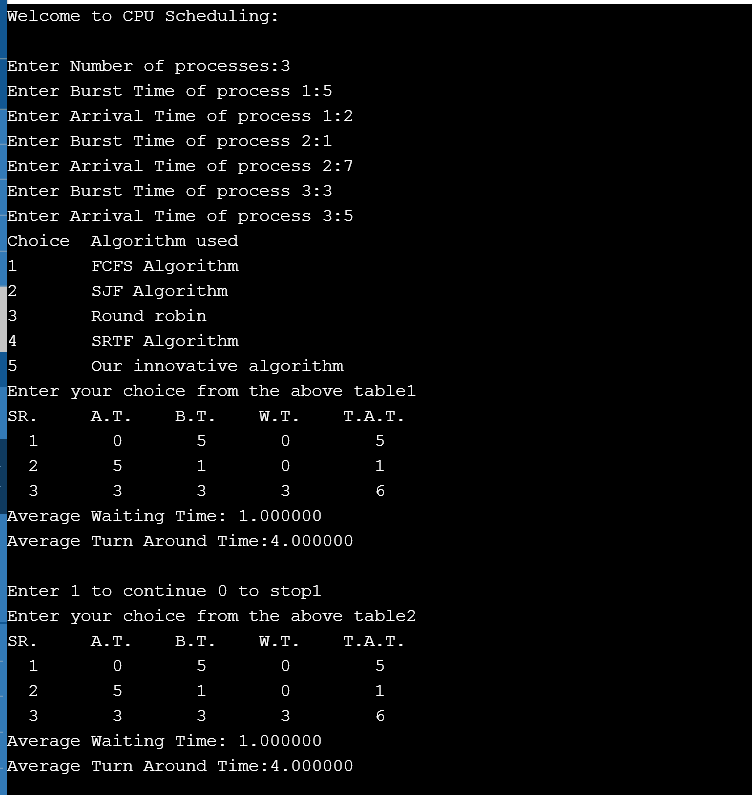
0 0 0 0 0 8

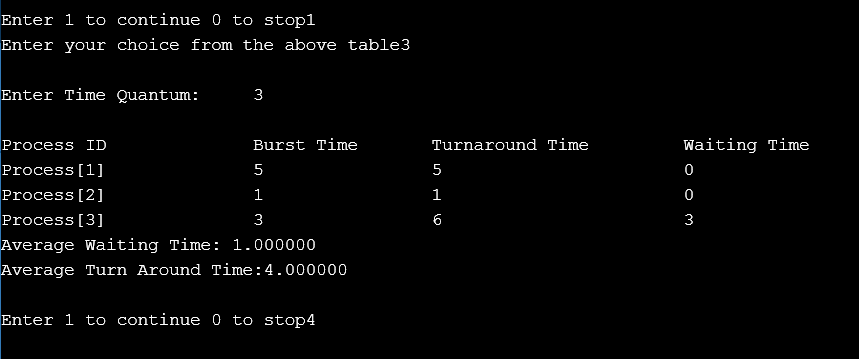
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P3 | P4 | P1 | P5 | P2 |

0 5 17 37 63 97

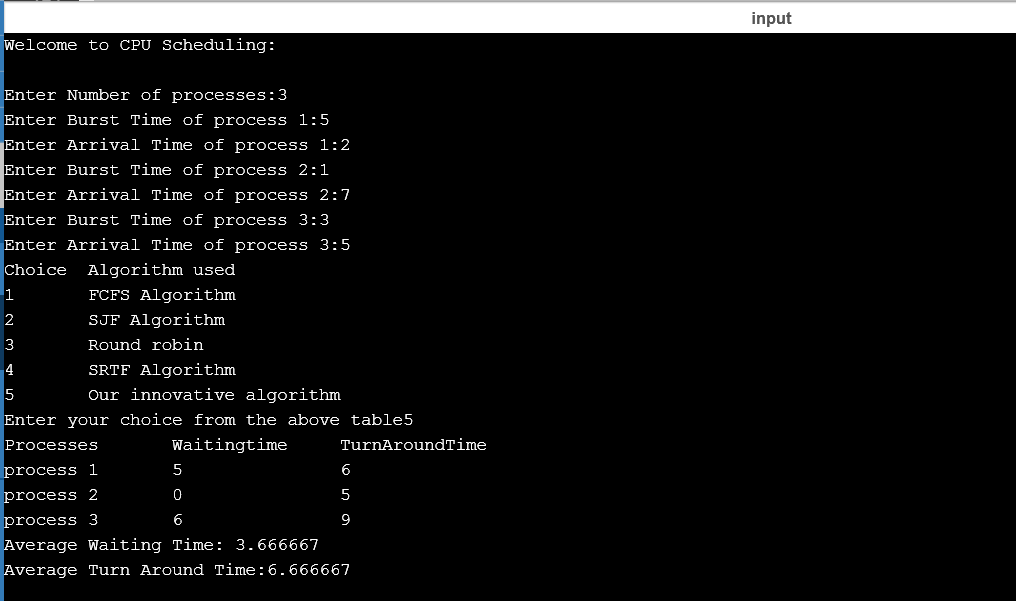
Gantt chart representation of XX

**For four algorithms:**





**For our prposed algorithm:**



**CONCLUSION:**

Results have shown that the proposed algorithm gives better results in terms of average waiting time, average turnaround time and number of context switches in all cases of process categories than the simple Round Robin CPU scheduling algorithm.

In all these proposed algorithms time quantum is static due to which in these cases the number of context switches, average waiting time and average turnaround time will be very high and in our proposed algorithm, time quantum is calculated dynamically according to the burst time of all processes and it will find out a smart time quantum for all processes which gives good performance as compared to FCFS, RR, SJF and XX.

**REFERENCES:**

[1] Seltzer, M P. Chen and J outerhout, 1990.Disk scheduling revisited in USENIX. Winter technical conference. Shamim H M 1998. Operating system, DCSA-2302.

[2] E.O. Oyetunji, A. E. Oluleye,” Performance Assessment of Some CPU Scheduling Algorithms”, Research Journal of Information Technology,1(1): pp 22-26,

[3] 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.

[4] 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), Vol. 1 Issue 3, pp 1-11.

[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), Vol. 1 Issue 3, pp 1-11.

[6] Manish K. M. and Abdul Kadir K. (2012): An Improved Round Robin CPU Scheduling Algorithm, Journal of Global Research in Computer Science, ISSN: 2229371X, Volume 3, No. 6, pp 64-69.

[7] Operating Systems\_ CPU Scheduling, http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSyst ems/5\_CPU\_Scheduling.html‎‎,accessed‎ 8th October 2013.

[8] Rakesh Kumar Yadav, Abhishek K Mishra, Navin Prakash, Himanshu Sharma,” An Improved Round Robin Scheduling Algorithm for CPU Scheduling”, (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 04, 1064-1066, 2010

[9] Suri, P.K and Sumit, M (2012): Design of Stochastic Simulator for Analyzing the Impact of Scalability on CPU Scheduling Algorithms, International Journal of Computer Applications (0975 – 8887) Volume 49, No.17, pp 4-9.

[10] Lalit, K, Rajendra, S and Praveen, S (2011): Optimized Scheduling Algorithm, International Journal of Computer Applications, pp 106109.

[11] Soraj, H and Roy, K.C: Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice", International Journal of Data Engineering (IJDE).