

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**COURSE TITLE: OPERATING SYSTEM**

**COURSE CODE: CSE2005**

**PROJECT TITLE:**

**ROUND ROBIN CPU SCHEDULING WITH DYAMIC QUANTUM TIME**

**SUBMITTED TO:**

**PROF. GOPINATH M.P**

**SUBMITTED BY:**

**NAME: RAHUL ANAND**

**REG NO.: 18BCE0953**

**LAB SLOT: L51+L52**

**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). That uses dynamic time quantum instead of static time quantum used in RR. 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 performance of the proposed algorithm is experimentally compared with traditional RR and some existing variants of RR.

**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 pre-empted and the control is given to the next process for computation. This happens in a circular fashion.

**Proposed Algorithm**

* Round Robin scheduling algorithm has been modified to propose and introduce a new methodology which aims in taking the efficiency of the CPU to a higher level by reducing the mean WT and the mean TAT.
* The CPU is allotted to the process for a specific time-quantum to perform the authorized functions. An optimized variant to the round robin scheduling algorithm is introduced.
* The algorithm is pre-emptive and works based on the priority of the associated processes. The priority is decided on the basis of the residual 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 computation, a time quantum is initially set. In case if the residual burst time of a particular process is lower 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. This will reduce the mean WT and mean TAT of the process.

**Theoretical explanation:**

**1. With Zero arrival time**

Assume that the burst time is in increasing order

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| P1 | 0 | 40 |
| P2 | 0 | 55 |
| P3 | 0 | 60 |
| P4 | 0 | 90 |
| P5 | 0 | 102 |

**Round Robin:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** | **P1** | **P2** | **P3** | **P4** | **P5** | **P2** | **P3** | **P4** | **P5** | **P4** | **P5** | **P5** |
| 25 | 50 | 75 | 100 | 125 | 140 | 165 | 190 | 215 | 240 | 245 | 255 | 280 | 305 | 320 | 345 | 347 |

**Average Turnaround Time =** (140+245+255+320+347)/5 = 1307/5 = 261.4

**Average Waiting time =** ((0+100)+(25+90+75)+(50+90+55)+(100+90+40+15+0))/5

= (100+190+195+230+245)/5

= 192

**Dynamic Average Burst Time Round Robin:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** | **P4** | **P5** | **P5** |
| **40** | **95** | **155** | **224** | **293** | **314** | **341** | **347** |

**Average Turnaround Time =** (40+95+155+314+347)/5=951/5=190.2

**Average Waiting Time =** (0+40+95+(155+69)+(224+21))/5 = (0+40+95+224+245)/5

=604/5

=120.8

**Comparison between RR, DABRR Algorithm**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Round robin** | **DABRR** |
| Time quantum | 25 | 69,27,6 |
| Context Switch | 16 | 7 |
| Average Waiting time | 192 | 120.8 |
| Average Turn Around Time | 261.4 | 190.2 |

**2. Without Zero arrival time**

**Let us assume the process’s Burst time is in increasing order**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| P1 | 0 | 27 |
| P2 | 3 | 32 |
| P3 | 5 | 55 |
| P4 | 7 | 82 |
| P5 | 9 | 110 |

**Round Robin:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** | **P1** | **P2** | **P3** | **P4** | **P5** |
| 25 | 50 | 75 | 100 | 125 | 127 | 134 | 159 | 184 | 209 |

**Dynamic Average Burst Time Round Robin:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** | **P4** | **P5** | **P5** |
| 27 | 59 | 114 | 183 | 252 | 265 | 292 | 306 |

**Average Turnaround Time** = (27+(59-3)+(114-5)+(265-7)+(306-9))/5=747/5=149.4

**Average Waiting Time**= (0+(27-3)+(59-5)+(114+69-7)+(183+13+0-9))/5 =   
 (0+24+54+176+187)/5  
 = 441/5  
 = 88.2

**DYNAMIC AVERAGE BURST TIME ROUND ROBIN (DABRR) ALGORITHM:**

**Variables assumed:**

Time\_Quant: Time Quantum

Ready\_Quant: Ready Queue

n: number of process

Pro\_i: Process at ith index

i, j: used as index of ready queue

Tot\_Burst\_Time: Total Burst Time

**ALGORITHM:**

1. Arrange the processes in ascending order.

2. n = number of processes in RQ

3. i=0, Tot\_Burst\_Time =0

4. Repeat step 5 and 6 till i < n

5. Tot\_Burst\_Time += burst time of process Pro\_i

6. i++

7. TQ = Tot\_Burst\_Time /n

8. j = 0

9. Repeat from step 12 to 19 till j<n

10. if (burst time of Pro\_i) <= Time\_Quant

11. Execute the process

12. Take the process out of RQ

13. n--

14. Else

15. Execute the process for a time interval up to 1 Time\_Quant

16. Burst time of Pro\_i = Burst time of Pro\_i – Time\_Quant

17. Add the process to ready queue for next round of execution

18. j++

19. If new process arrives

20. goto step 1

21. If Ready\_Quant is not empty

22. goto step 2

**Code:**

#include<stdio.h>

#include<stdlib.h>

void clrscr()

{

system("@cls||clear");

}

struct proc

{

int id;

int bt;

int rem;

int wait;

int turnaround;

};

void sort(struct proc p[],int n)

{

int i=0,j=0;

struct proc temp;

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

{

for(j=i+1;j<n;j++)

if ( p[j].bt<p[i].bt )

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

}

}

}

void allocate(struct proc bt[],int n,struct proc sev1[],struct proc sev2[],struct proc sev3[])

{

int i,j;

for(j=0;j<n/3;j++)

{

sev1[j]=bt[j];

}

for (j=n/3;j<2\*n/3;j++)

sev2[j-n/3]=bt[j];

for (j=2\*(n/3);j<n;j++)

sev3[j-2\*(n/3)]=bt[j];

}

void runSeverRR(int n,int quant,struct proc sev[])

{

int i,j,total=0,count=n;

float avgtat=0.0,avgwait=0.0;

printf("\t\nRunning processes using Round Robin");

printf("\nProcess | wait time | TurnAround Time");

while(count)

{

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

{

if (sev[i].rem!=0)

{

if (sev[i].rem-quant>0)

{

total=total+quant;

sev[i].rem=sev[i].rem-quant;

}

else

{

total=total+sev[i].rem;

sev[i].rem=0;

count--;

sev[i].turnaround=total;

sev[i].wait=total-sev[i].bt;

printf("\nP%d | %d | %d ",sev[i].id,sev[i].wait,sev[i].turnaround);

}

}

}

}

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

{

avgtat=avgtat+1.0\*sev[i].turnaround;

avgwait= avgwait +1.0\*sev[i].wait;

}

avgtat=avgtat/(1.0\*n);

avgwait=avgwait/(1.0\*n);

printf("\n\nThe average turn around time of the processes is : %f",avgtat);

printf("\n\nThe average waiting time of the processes is : %f\n",avgwait);

}

void runSeverDRR(int n,struct proc sev[])

{

int i,j,total=0,count=n;

float avgtat=0,avgwait=0;

int quant=0;

int flag=1;

printf("\t\nRunning processes using Dynamic Round Robin");

printf("\nProcess | wait time | TurnAround Time");

while(count)

{

if (flag)

{

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

{

quant=quant+sev[i].rem;

}

quant=quant/count;

flag= 0;

}

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

{

if (sev[i].rem!=0)

{

if (sev[i].rem-quant>0)

{

total=total+quant;

sev[i].rem=sev[i].rem-quant;

}

else

{

total=total+sev[i].rem;

sev[i].rem=0;

count--;

sev[i].turnaround=total;

sev[i].wait=total-sev[i].bt;

printf("\nP%d | %d | %d ",sev[i].id,sev[i].wait,sev[i].turnaround);

flag=1;

}

}

}

}

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

{

avgtat=avgtat+sev[i].turnaround;

avgwait= avgwait +sev[i].wait;

}

avgtat=avgtat/(1.0\*n);

avgwait=avgwait/(1.0\*n);

printf("\n\nThe average turn around time of the processes is : %f",avgtat);

printf("\n\nThe average waiting time of the processes is : %f\n",avgwait);

}

void compalgo(int n,struct proc sev[],float tat[],float awt[],int quant)

{

int i,j,total=0,count=n;

float avgtat=0,avgwait=0;

while(count)

{

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

{

if (sev[i].rem!=0)

{

if (sev[i].rem-quant>0)

{

total=total+quant;

sev[i].rem=sev[i].rem-quant;

}

else

{

total=total+sev[i].rem;

sev[i].rem=0;

count--;

sev[i].turnaround=total;

sev[i].wait=total-sev[i].bt;

}

}

}

}

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

{

avgtat=avgtat+sev[i].turnaround;

avgwait= avgwait +sev[i].wait;

}

avgtat=avgtat/(1.0\*n);

avgwait=avgwait/(1.0\*n);

tat[0]=avgtat;

awt[0]=avgwait;

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

{

sev[i].rem=sev[i].bt;

}

avgtat=0.0;

avgwait=0.0;

total=0;

count=n;

quant=0;

int flag=1;

while(count)

{

if (flag)

{

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

{

quant=quant+sev[i].rem;

}

quant=quant/count;

flag= 0;

}

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

{

if (sev[i].rem!=0)

{

if (sev[i].rem-quant>0)

{

total=total+quant;

sev[i].rem=sev[i].rem-quant;

}

else

{

total=total+sev[i].rem;

sev[i].rem=0;

count--;

sev[i].turnaround=total;

sev[i].wait=total-sev[i].bt;

flag=1;

}

}

}

}

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

{

avgtat=avgtat+sev[i].turnaround;

avgwait= avgwait +sev[i].wait;

}

avgtat=avgtat/(1.0\*n);

avgwait=avgwait/(1.0\*n);

tat[1]=avgtat;

awt[1]=avgwait;

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

{

sev[i].rem=sev[i].bt;

}

}

int main()

{

int s=0;

printf("\n \t\t\t\t ROUND ROBIN CPU SCHEDULING WITH DYNAMIC QUANTUM TIME\n ");

for(s=0;s<1000000000;s++);

clrscr();

int n,tempN,i,ip=0,tquant;

printf("\t\tWelcome to the Project on Comparison between Round Robin And Dynamic Round Robin Algorithm\n");

printf("\nEnter the number of processes you want to insert: ");

scanf("%d",&n);

clrscr();

struct proc process[n];

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

{

printf("\nEnter the Burst Time process: ");

scanf("%d",&process[i].bt);

process[i].rem=process[i].bt;

process[i].id=i;

process[i].wait=0;

process[i].turnaround=0;

}

sort(process,n);

struct proc sev1[n/3],sev2[2\*(n/3)],sev3[n-2\*(n/3)];

allocate(process,n,sev1,sev2,sev3);

clrscr();

while(ip!=7)

{

printf("\nEnter the operation you want to perform");

printf("\n1. Run server 1");

printf("\n2. Run server 2");

printf("\n3. Run server 3");

printf("\n4. Run All servers ");

printf("\n5. exit");

printf("\nenter you choice: ");

scanf("%d",&ip);

if (ip==1)

{

printf("\nEnter the time quantum for the server: ");

scanf("%d",&tquant);

clrscr();

printf("\nRunning server 1");

runSeverRR(n/3,tquant,sev1);

allocate(process,n,sev1,sev2,sev3);

runSeverDRR(n/3,sev1);

for(s=0;s<2000000000;s++);

clrscr();

}

else if (ip==2)

{

printf("\nEnter the time quantum for the server: ");

scanf("%d",&tquant);

clrscr();

printf("\nRunning server 2");

runSeverRR(n/3,tquant,sev2);

allocate(process,n,sev1,sev2,sev3);

runSeverDRR(n/3,sev2);

for(s=0;s<2000000000;s++);

clrscr();

}

else if (ip==3)

{

printf("\nEnter the time quantum for the server: ");

scanf("%d",&tquant);

clrscr();

printf("\nRunning server 3");

runSeverRR(n-2\*(n/3),tquant,sev3);

allocate(process,n,sev1,sev2,sev3);

runSeverDRR(n-2\*(n/3),sev3);

for(s=0;s<2000000000;s++);

clrscr();

}

else if (ip==4)

{

float tat[2],awt[2];

printf("\nEnter the time quantum for the server: ");

scanf("%d",&tquant);

clrscr();

compalgo(n,process,tat,awt,tquant);

printf("\n Comparison between the algorithms ");

printf("\nThe average Turn Around Time of Round robin is : %f",tat[0]);

printf("\nThe average Turn Around Time of Dynamic Round robin is : %f",tat[1]);

printf("\nThe average Waiting Time of the Round Robin is : %f",awt[0]);

printf("\nThe average Waiting Time of the Dynamic Round Robin is : %f",awt[1]);

for(s=0;s<2000000000;s++);

clrscr();

}

else

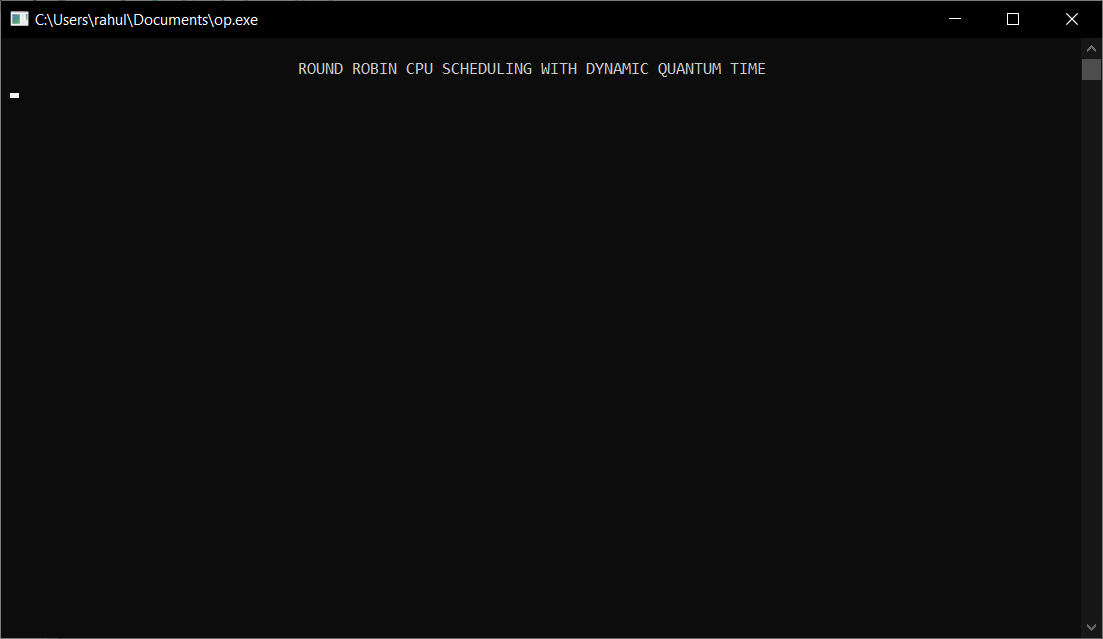
ip=7;

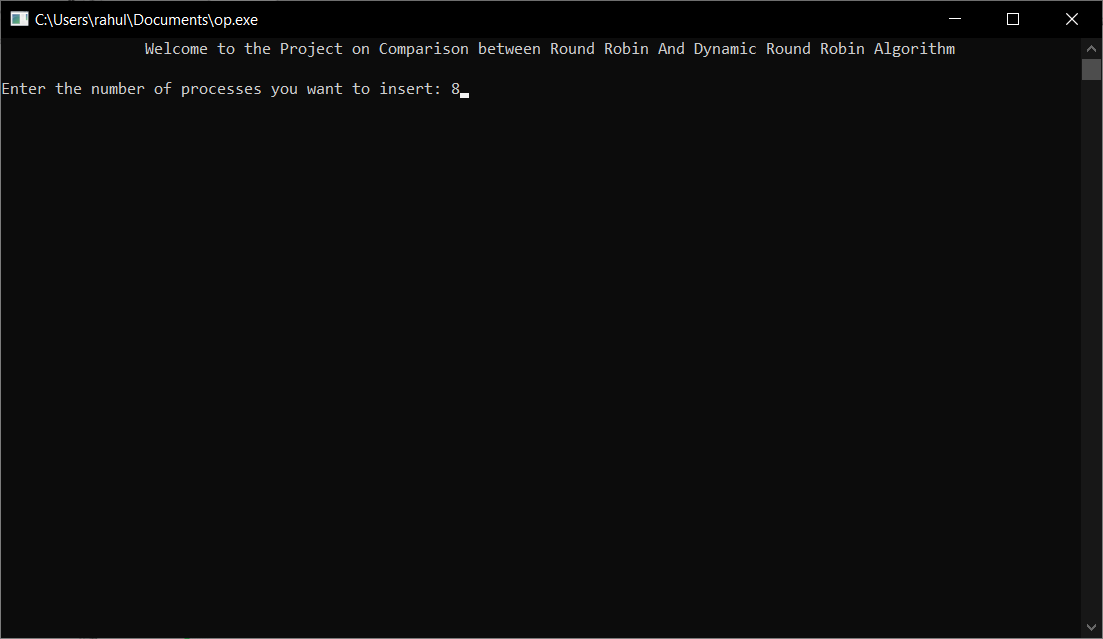
}

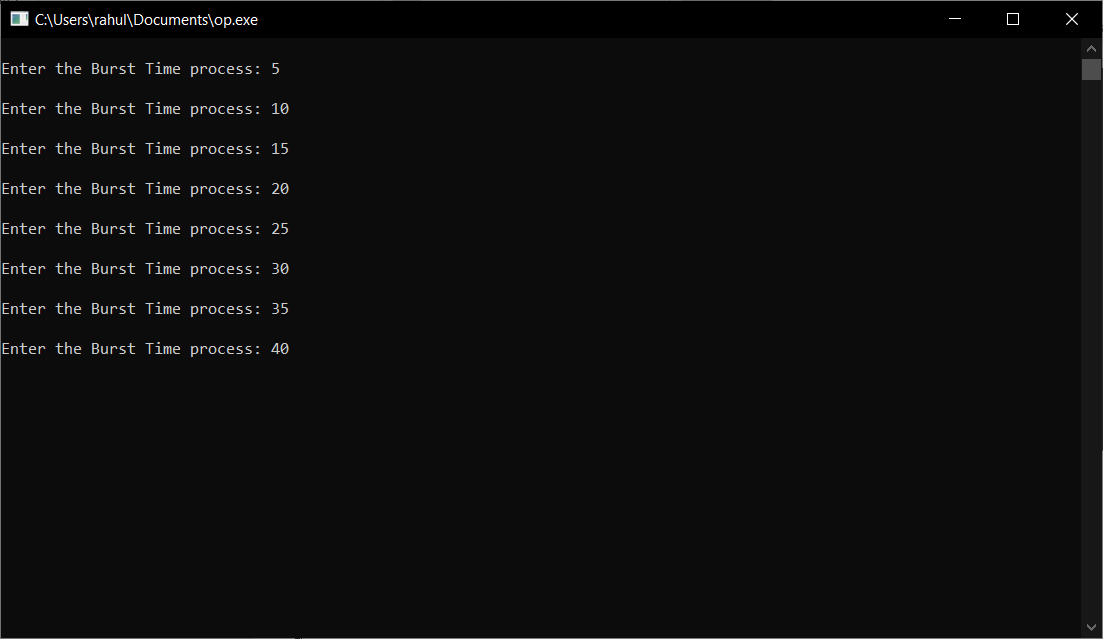
return 0;

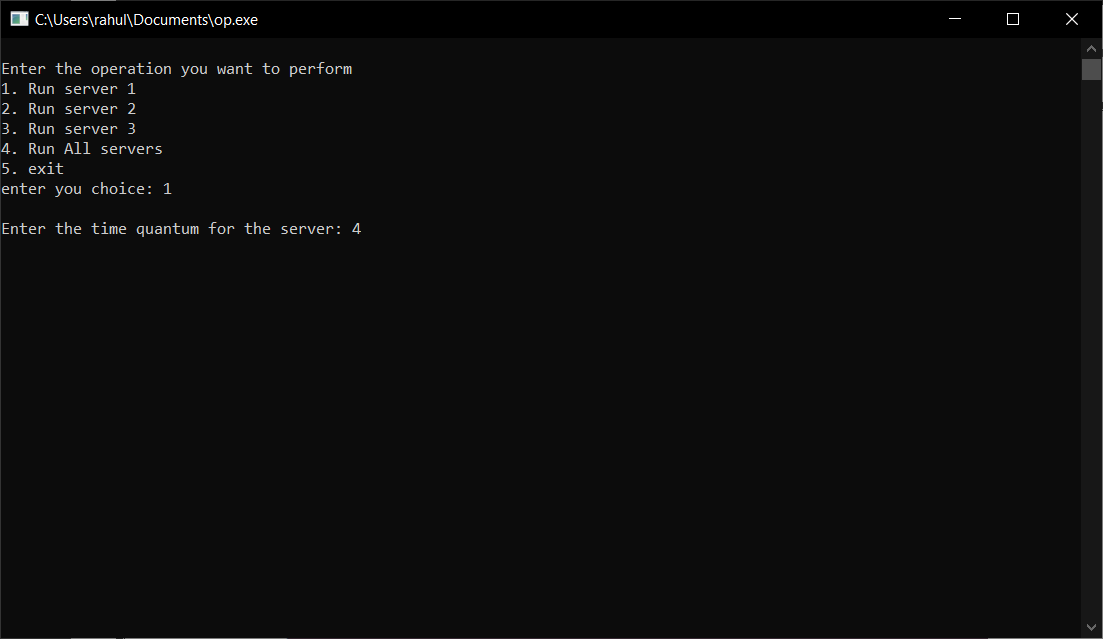
}

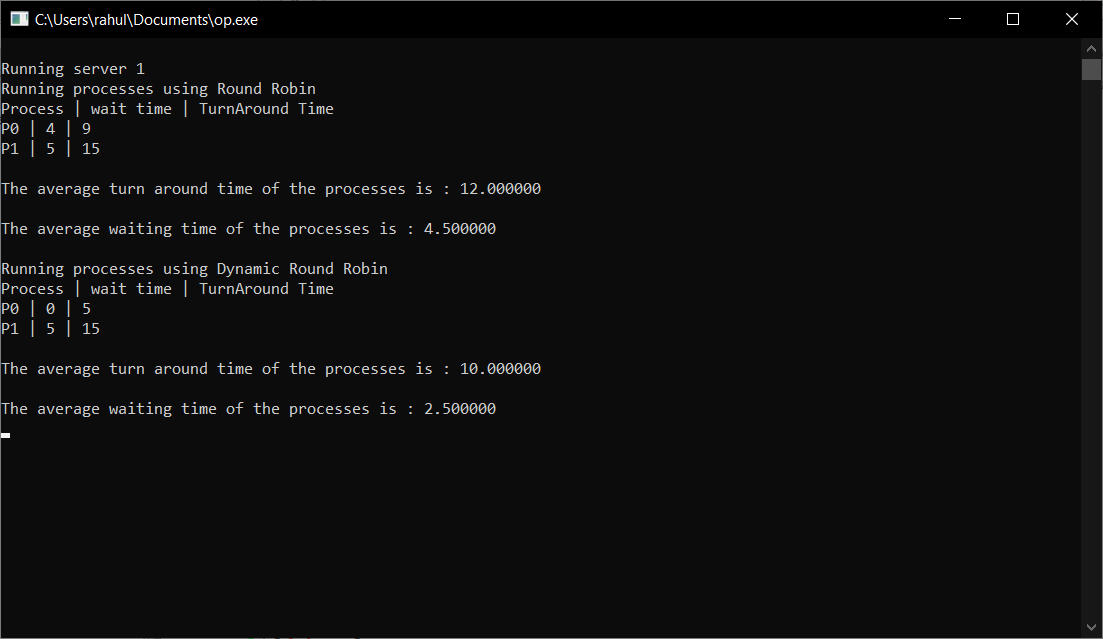
**CODE OUTPUT**

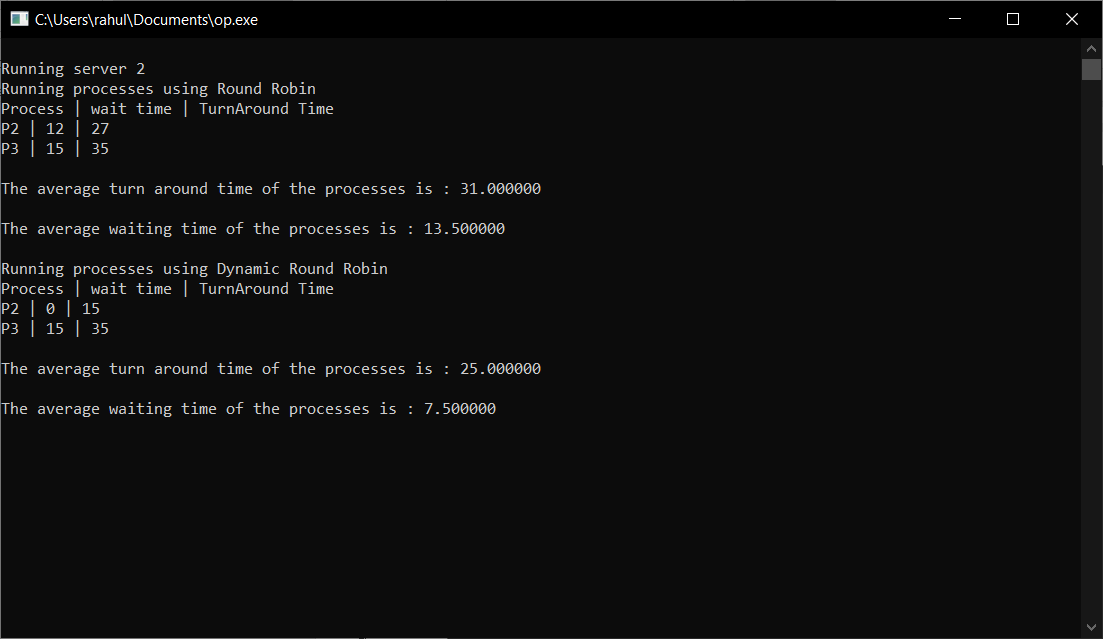


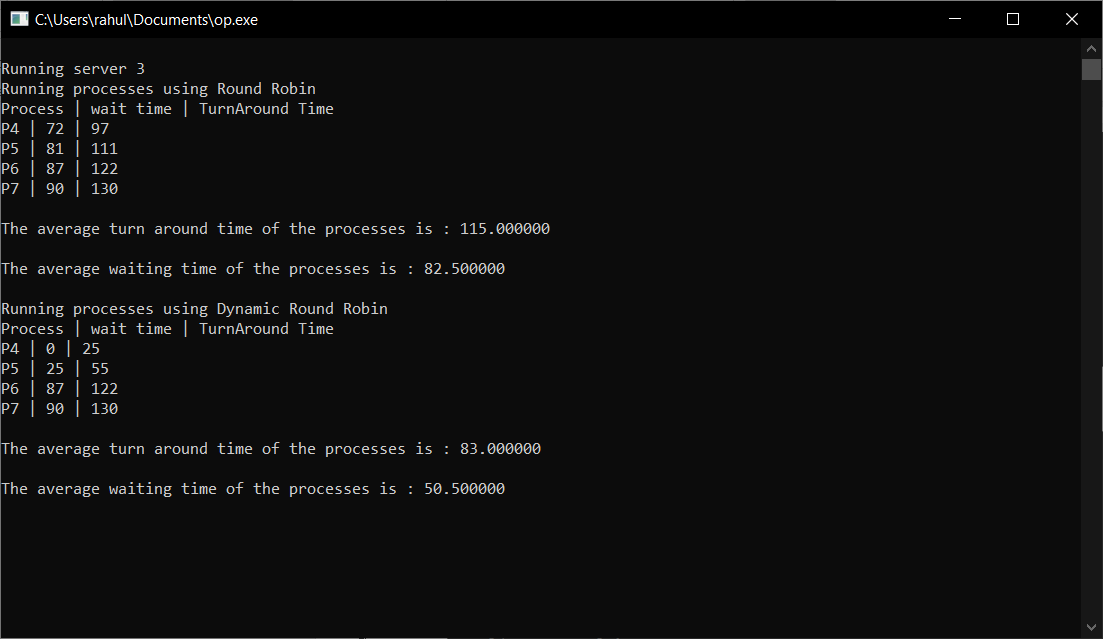


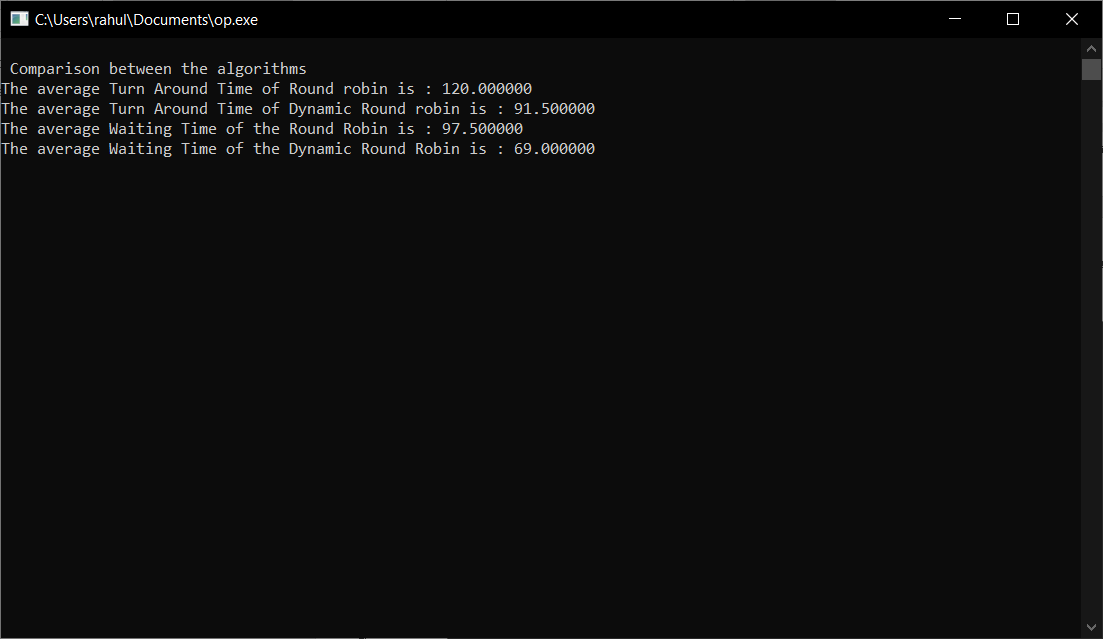






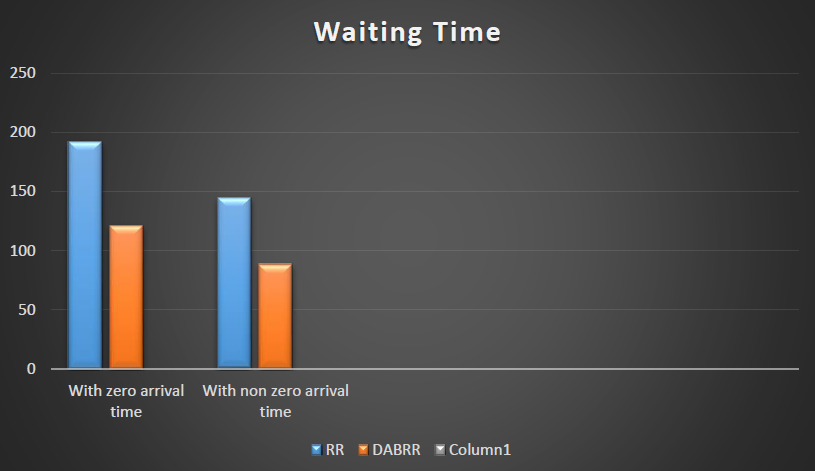


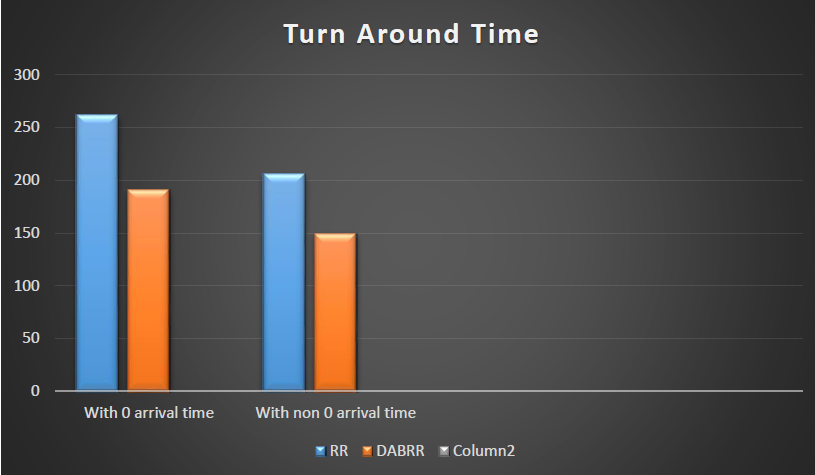




**Comparison between RR, DABRR**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Round robin** | **DABRR** |
| Time quantum | 25 | 27,69,27,14 |
| Context Switch | 15 | 7 |
| Average Waiting time | 144.4 | 88.2 |
| Average Turn Around Time | 205.6 | 149.4 |





**Conclusion:**

* The traditional Round Robin gives **higher mean Waiting Time** and **higher the mean TAT** as compared to the modified Round Robin.
* The modified Round Robin proves to be better and efficient when it compares the performance aspect.
* The **execution time** also decreases in case of the modified Round Robin.
* This will in turn increase the efficiency of the system as the waiting time and TAT is reduced.
* This will be very much helpful, as the overall performance is increased through increase in throughput and Decrease in response time.

**Reference:**

* *Silberschatz, A., Galvin, P.B., Gagne, G.: Operating system principles, 7th edn.*
* *Dhamdhere, D.M.: Operating Systems A concept-based approach. Tata McGraw Hill*
* *Sinth, A., Goyal, P., Batra, S.: An optimized Round Robin Scheduling Algorithm for CPU Scheduling. IJCSE 02(07), 2383–2385 (2010)*
* *Yaashuwanth, C., Ramesh, R.: A New Scheduling Algorithm. IJCSIS 6(2) (2009)*
* *Noon, A., Kalakecch, A., Kadry, S.: A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average. IJCSI 8(3(1)) (May 2011)*
* *Matarneh, R.J.: Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes. American Journal of Applied Sciences 6(10), 1831–1837 (2009) ISSN 1546-9239*
* *Hiranwal, S., Roy, K.C.: Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice. International Journal of computer Science and Communication 2(2), 219–326 (2011)*
* *Abdullahi, I., and Junaidu, S. B (2013): Empirical Framework to Migrate Problems in Longer Job First Scheduling Algorithm (LJF+CBT), International Journal of Computer Applications (0975 – 8887) Volume 75– No.14, pp 9-14.*