An Improved Context Switching algorithm for Round Robin (RR) Scheduler with Dynamic Time Quantum Assignment

Gaurav Kumar Singh (19BCE2119)

***Abstract*— Process Management is considered a very important function of any OS as it is the backbone of the system. There are several kinds of scheduling algorithms to perform the given task each of which have their own pros and cons. Round Robin is among the most prominent scheduling algorithms. Round Robin schedulers preempt the task based on a specific time quantum and switches to next task and run it for the same time quantum and repeat this process until no tasks are left in the ready queue. The performance and efficiency of a Round Robin scheduler is heavily dependent upon one's choice of the time quantum. If time quantum is too high, the scheduler tends to morph into a First in First Out scheduler while if it is too small, it causes a lot of context switches wasting a lot of time performing unnecessary task. This project revolves around how to dynamically pick an appropriate time quantum for one cycle and at the same time prevent context switching for tasks that use a very small portion of time quantum otherwise. Also, if the process is made to preempt right before it was about to be completed, not only does it result in an additional context switch later, it also makes that process wait for more time than needed to be completed. In this paper, we will be dealing with these problems to propose a better Round Robin algorithm applicable even for real time systems**

Index Terms— Message, Whatsapp, Mobile phones, Communication, Texting, User Study, Interview, Survey, Cliques, Social network, Network information

INTRODUCTION

Every Operating System needs scheduler to properly manage the processes entering into the system. The performance of any Operating system is largely dependent on the scheduling algorithms used. Its how the sequence of execution of processes is decided. There are many scheduling algorithms, most basic and famous of which are First-In-First-Out (FIFO), Shortest-Job-First (SJF), Priority-Scheduling (PS), Shortest-Remaining-Time-First (SRTF), and one of the most commonly used is Round-Robin (RR). FIFO and SJF are non-preemptive scheduling algorithms i.e., once a process is assigned to CPU, it will be run through to its completion without any interruption from other processes in the ready queue. SRTF and RR are preemptive scheduling algorithms as they switch between the tasks even before their completion to complete a task through multiple short bursts of time. It gives the feel of multitasking, boosts response time of the processes and ensure that no process waits for obscure amount of time before it finally gets a chance to run.

Priority Scheduling (PS) has both a preemptive and a non-preemptive variation.

FIFO as the name suggests means the task which entered the ready queue first will be executed first to its completion. In SJF the process with the shortest burst time in the ready queue will be executed first. Its principle is to get as many tasks over in as little time as possible which might sometimes make the longer tasks to wait for large amount of time. In Priority Scheduling, every task is assigned a priority and the order of execution is decided based on that priority. Its concept is fundamentally better than that of FIFO and SJF specially because it also has a preemptive variation and the tasks which are more important are given priority in it which is good to improve the overall functionality of the operating system. SRTF can be considered as a preemptive variation of SJF where every task is run for one cycle and the task to be run is selected on the amount of remaining burst time it has. Task with the shortest remaining burst time is given priority when selecting task to be run for the next cycle. RR is one of the most famous scheduling algorithms with the flexibility of being able to run a task with preemption for more than one cycle in a burst while also being able to incorporate different hybrid methodologies as its way of selecting the next process to run.

Since RR is one of the most widely used scheduling algorithm, this paper is based on highlighting the ways in which we can improve the traditional Round Robin scheduler by specifically tackling the issues any preemptive scheduling algorithm, specifically Round Robin has to improve its efficiency whilst not compromising any of its existing pros.

The first thing we can do is to improve the way any preemptive scheduler handles its context switching. The way it works in normal preemptive scheduling algorithms is that the scheduler switches to a different task as soon as it is run for its allotted number of cycles, but what it means for the operating system that if the task is preempted right before it was about to be completed, the operating system will have to waste an additional context switch later which wastes some time, also the process's waiting time would increase and ready queue would just get more and more crowded. Its solution is to just make the task run to its completion if the remaining burst time of the process is very small compared to the time quantum of the RR.

Another way to improve its overall context switching and reduce the average waiting time and average turnaround time is by the use of Dynamic time quantum in which the time quantum is dynamically calculated before every preemption with the remaining burst time of the process to best cater the process' requirement.

The above methodologies can significantly boost the performance of the RR depending on the choices of time quantum made for both algorithm given minimum time quantum of our improved round robin is the same as that of

normal RR.

I. LITERATURE SURVEY

Summary/Gaps/Limitations/Future Work identified in the Survey

1) The study propose a Priority based Round Robin scheduling algorithm which maintains the advantages of having a Round Robin scheduling algorithm by reducing starvation and adds to that the advantage of priority scheduling. It proposes a scheduling model which is more scalable for real time operating systems by improving the operating system's waiting time, response time, turnaround time, and throughput. It throws light upon different kinds of existing scheduling algorithms such as First in First Out, Shorter First, Round Robin, Priority Based Scheduler, Rate Monotonic, Dead Monotonic etc. It emphasizes on the drawbacks of traditional Round Robin based systems such as high average waiting and turnaround times.

In future work, it is planned to be able use hardware architecture based on FPGA (Field Programmable Gate Array). Hardware accelerators can be used to speed up the sorting process in the ready queue. Hybrid approaches can be utilized alongside this proposition to suit different real time systems

2) This paper emphasizes on the how time quantum of a Round Robin Scheduler affects its performance and context switching time. They propose an algorithm where the time quantum is selected dynamically and numerous experiments have been conducted to prove its efficiency. The performance of a Round Robin Scheduler is heavily dependent upon the choice of Time Quantum selected. A very large TQ might mean the RR scheduler morphed into FCFS and when it becomes too small, the context switch skyrockets. However, this algorithm does not work where priority-based scheduling is required and also the selection of time quantum by the system becomes obnoxious towards the end of the processes if the remaining burst times of the processes are very distinct after every cycle resulting in heavily compromised waiting times for some set of processes depending on type of processes.

In future work they aim to develop hybrid adaptive algorithm which combines all the scheduling algorithms to get their pros and tackle most of their cons to develop a close to ideal scheduling algorithm.

3) In this Research paper, the authors propose a Round Robin with Dynamic Time Quantum (RRDTQ) system for enhancing the CPU performance using dynamic time quantum assignment. In their proposed algorithm the burst time of all the processes in the ready queue are noted and the TQ's value is assigned based on that average value. It reduces the number of context switches required however might not be the best if the burst times of all processes are very close to each other but not equal to mean.

For future work, they plan to embed dynamic time quantum round robin algorithm to more real time systems with better context switching algorithms. The very issue with directly using mean of remaining burst time of processes as time quantum is that the algorithm itself varies massively between behaving like a First-In-First-Out and having very low time quantum for processes with large burst times leading to starvation of large processes if their concentration is very low when compared to processes with very small burst times.

4) This Research paper also uses RRDTQ to reduce the number of context switches but instead of using mean to calculate TQ of a certain process, Median is used to calculate effective time quantum. Of all the statistical aggregation techniques, the author used median which can be very polarizing in obnoxious process ready queues since median is usually not the best representative aggregation value for most dataset. Its relevance is highly dependent on the distribution of data in the dataset which in case of operating systems, users have little to no control over.

Using the avg. waiting time, avg. turnaround time, and context switch as performance parameters, the above experimental study shows that our suggested IRR scheduling method outperforms the standard RR scheduling technique. Soft and hard real-time systems can be used to expand our suggested approach.

5) In this report the authors compare various aspects of a CPU scheduling algorithms with each other, namely waiting time, turnaround time, CPU utilization, Throughput etc. amongst FCFS, PS and SJF schedulers to find optimum scheduling algorithms for different kind of needs.
It gives in-depth details about the prominent scheduling algorithms of its era

and helps us to formulate how much progress we have made in the decade in the field of scheduling and its algorithms. It still remains very relevant since it covers all the basic CPU scheduling algorithms and gives visualizations to easily compare their performance on different metrics.

For future work, it aims to compare more developed scheduling algorithms on the same metrics or even some new metrics if discovered to find best use case of the algorithms.

6) This study throw light upon different RR algorithms and their performance statistics such as RR with static time quantum, RR with Dynamic time quantum, RRDTQ using Manhattan distance (RRDTQ), Improved time quantum RR (static TQ RR), Adaptive RR and Best Time Quantum RR to give insight upon the pros and cons of each type of Round Robin Scheduler. It compares the Average waiting time, Average turnaround time, response time etc. of all the predominant existing variations of round robin scheduling algorithm and suggests which algorithm is best in which use case since each have its own set of pros and cons.

For future work, it aims to compare and visualize more developing round robin scheduling algorithm variation to find one which best fits a particular use case or fits well in most cases for better insights about round robin algorithm and its offset variations.

PROPOSED METHODOLOGY

Introduction

Round robin scheduling is very commonly used in operating system scheduling. It is a simple, easy-to-implement, and starvation free algorithm to preemptively schedule the processes in the operating systems. It has many variations around it depending on its use case such as weighted round robin and others. It ensures fairness of the scheduler and is most often used in network packet scheduling. Its variation such as deficit RR, weighted RR, weighted fair queueing etc. Most operating systems combine several scheduling algorithms, and many of them already implement round robin as one of its part even if not in its original form. Windows use a hybrid of round robin and priority scheduling to ensure that the operating system is running as efficiently as possible and that the core processes have higher priority than others. We will be comparing basic Round Robin to our improved round robin algorithm in this paper where we use the concept of dynamic time quantum assignment and force the process to its completion if it satisfies certain criteria to reduce context switching while still maintain its fairness and response time and simultaneously, also minimizing waiting times and turnaround times of the processes.

ROUND ROBIN

To facilitate the same the following steps are followed of the algorithm designed:

1. Importing the Libraries: stdio.h, unistd.h, sys/types.h, stdlib.h, string.h, to use system commands, manipulate strings and handle input and outputs as:

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>
```

2. Then, we proceed to create a struct names ‘Process’ which contains every detail of each and every process we are simulating in our system.
3. Initialize all the functions to calculate the completion time, turnaround time, waiting time, average waiting time and average turnaround time.
4. Prepare the ready queue of Process struct type which manages the ready queue and another array of type struct process to store information of all processes in.
5. Take input for process id, arrival time, process burst time and time quantum for the round robin scheduler and run the functions as declared before.
6. Append the data to the csv file to later use for the visualization.

IMPROVED ROUND ROBIN

To facilitate the same the following steps are followed of the algorithm designed:

1. Importing the Libraries: stdio.h, unistd.h, sys/types.h, stdlib.h, string.h, to use system commands, manipulate strings and handle input and outputs as:

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>
```

2. Then, we proceed to create a struct names 'Process' which contains every detail of each and every process we are simulating in our system.
3. Initialize all the functions to calculate the completion time, turnaround time, waiting time, average waiting time and average turnaround time. Also, initialize the functions such that they manage the time quantum in a way which makes sure the time quantum for each cycle is between a particular range of integers.
4. Create a function which checks for the remaining burst time of the process in the ready queue and compare it to the time quantum of that cycle to decide whether the process should be run to its completion
5. Prepare the ready queue of Process struct type which manages the ready queue and another array of type struct process to store information of all processes in.
6. Take input for process id, arrival time, process burst time, minimum time quantum and maximum time quantum, for the round robin scheduler and run the functions as declared before.
7. Append the data to the csv file to later use for the visualization.

VISUALIZATION

To facilitate the same the following steps are followed

1. Importing the Libraries: os, pandas, plotly, to read the csv file and perform system operations using our python program as:

```
import os
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
import os.path
```

2. We check if the necessary files exist in proper format and then use read_csv function of pandas to create a data frame of the data.
3. We then use plotly express module from plotly to plot scatter plot with a LOWESS trendline to make the data easier to interpret.
4. We repeat the same step to compare completion time, waiting time and turnaround times of the traditional Round Robin and out Improved Round Robin algorithm.
5. We now change the visualization technique to bar chart to compare the average waiting time and average turnaround time for both the algorithms.
6. Plotly express graphs can be exported offline in html format so we make another html webpage where we use object tags to show all the exported graphs under one webpage as frontend for easier interpretation.

Framework, Architecture or Modules of the Proposed System:

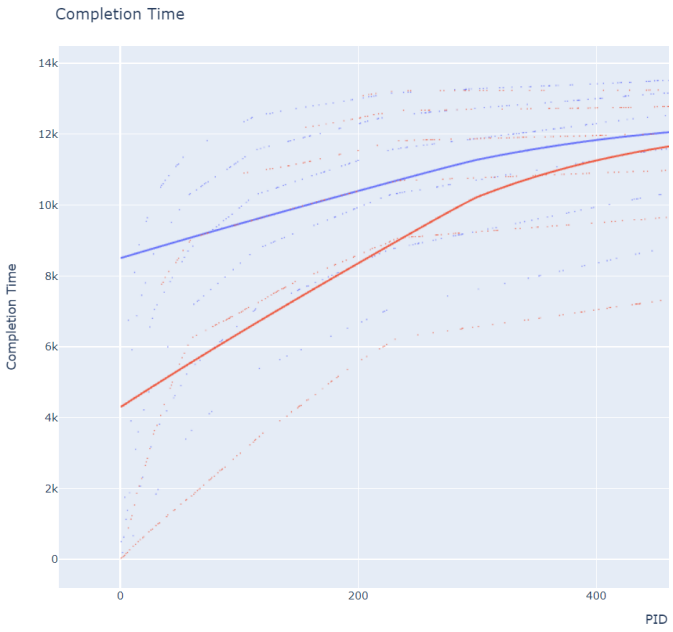
Requirements

- GCC compiler
- GCC IDE (Codeblocks)
- Python 3.8
- Pandas
- Plotly
- Text editor or Python IDE (Visual Studio Code, Jupyter Notebook, notepad etc.)

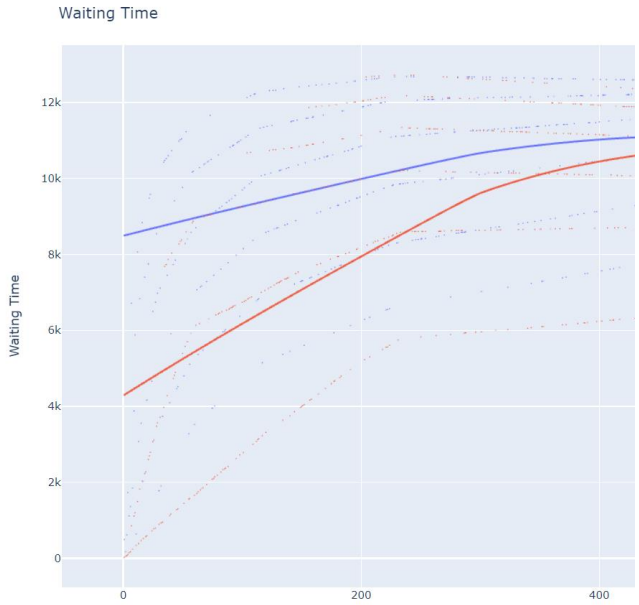
Proposed System Model :

After Analyzation the following models are obtained:

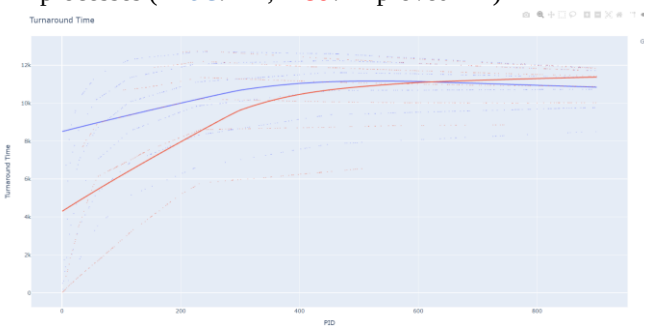
1. Completion Times for randomly generated 900 processes (Blue: RR, Red: Improved RR)



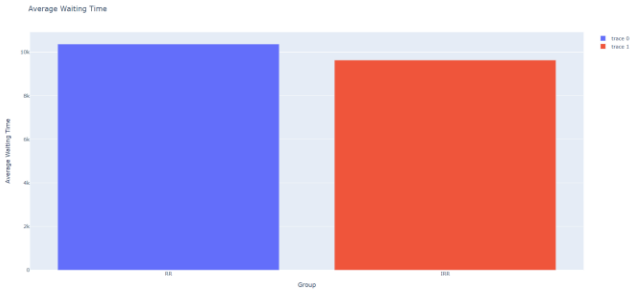
2. Waiting Times for randomly generated 900 processes (Blue: RR, Red: Improved RR)



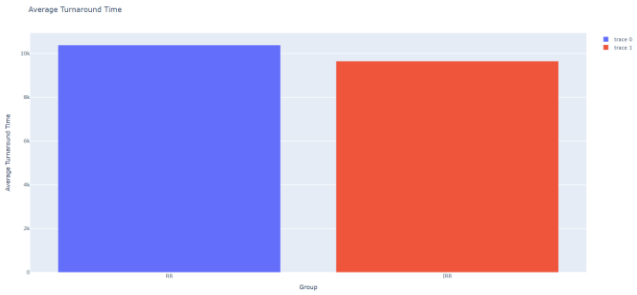
3. Turnaround Times for randomly generated 900 processes (Blue: RR, Red: Improved RR)



4. Average Waiting Times for RR and Improved RR



5. Average Turnaround Times for RR and Improved RR



PROPOSED SYSTEM ANALYSIS AND DESIGN

We used a list of 900 randomly generated processes each of which enters into the system at an offset of 2ms for consistency and randomly generated value of burst time for each process but the initial data of the process remains consistent for both the algorithms to enable easier translation. For our example, we used the value of 2ms as time quantum for normal RR and a rage of 2-10ms for our Dynamic Time Quantum RR. As it can be seen from the graphs, the average waiting time and turnaround time is about 7-10% better than normal RR and it only increases with more and more uncertainty in the burst times of the processes which enter the system which for our automated query had a fixed upper and lower limit.

The following observations can be made from the above graphs:

i. As for Completion time, the completion time of the process which enter the system earlier is lower in improved RR because of how we made the process run to completion if its remaining burst time is calculated to be much lower than the time quantum. However, since the burst times of all the processes is same for both programs, the completion time for the processes which enter late is delayed a little bit more because the time of completion of all tasks is fixed and equal to sum of burst times of all processes.

ii. For Waiting times, it seems to be following the same trend as completion time and turnaround time but its sum unlike completion time is not fixed and can be improved. If we check the scale, even though the waveform seems similar but the scale is not and this is where our improved round robin algorithm gains back

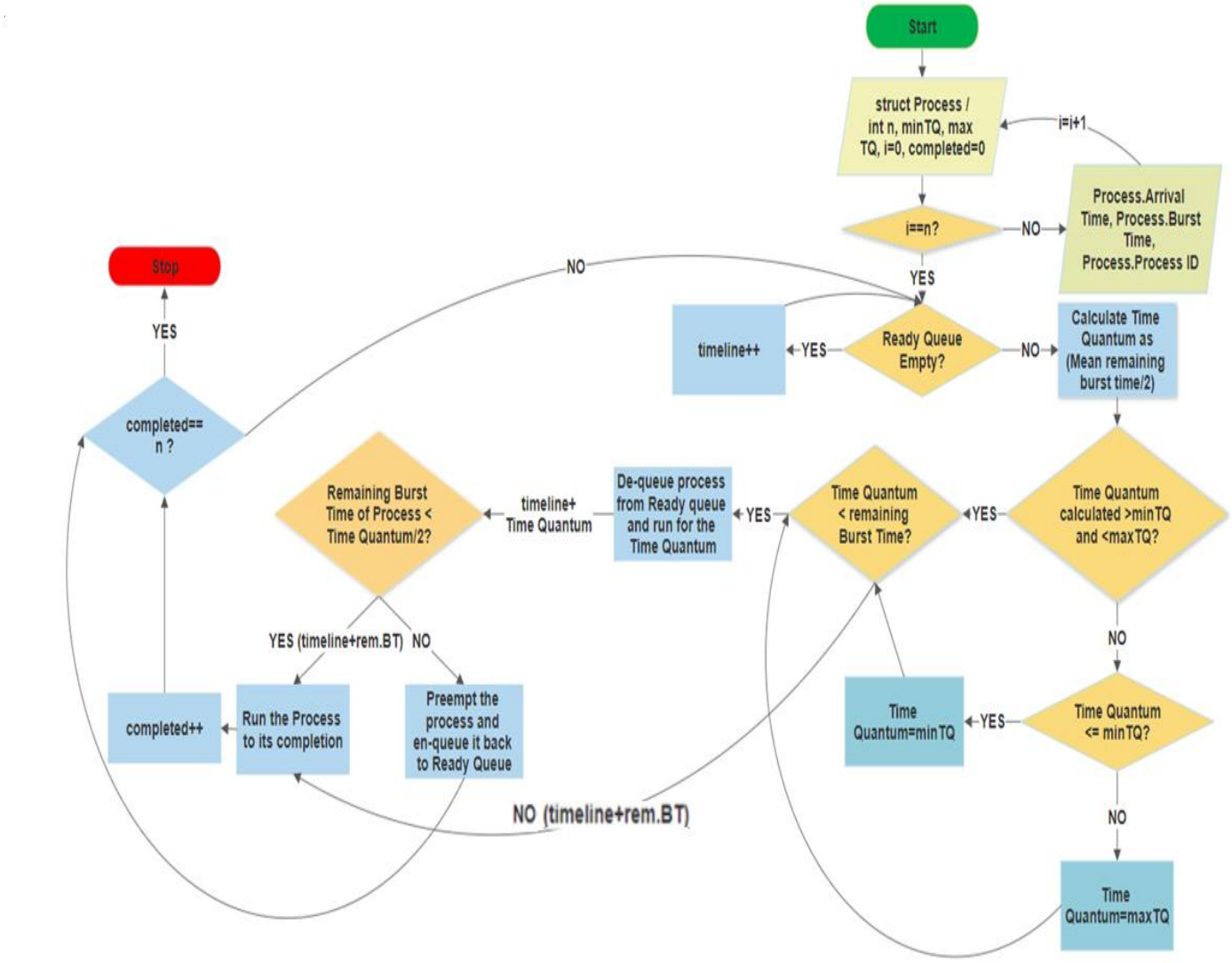
some ground. The waiting time of initial process is much lower in our algorithm than traditional RR and even though the waiting time for the processes which entered system late is more than that of traditional RR it's not even close to being high enough to tip the balance in the favor of traditional RR.

iii. Same trend as Waiting Times can be found in turnaround times. Scale changes again but our improved RR is still way better than traditional RR.

iv. The bar graphs for average turnaround times and waiting times is perfect proof of point no. 2 and 3 as from both those figures it can be calculated that our improved round robin is at least 7-10% better than traditional RR algorithm.

DESIGN

Flowchart for Improved Round Robin



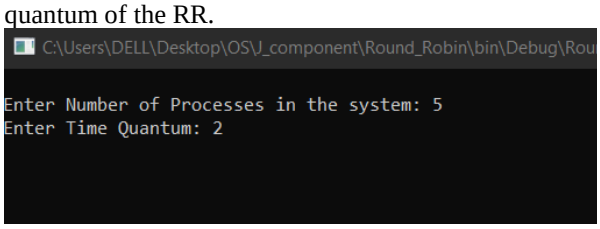
IMPLEMENTATION

The first step, is to compile and run both the programs (RR and improved RR)

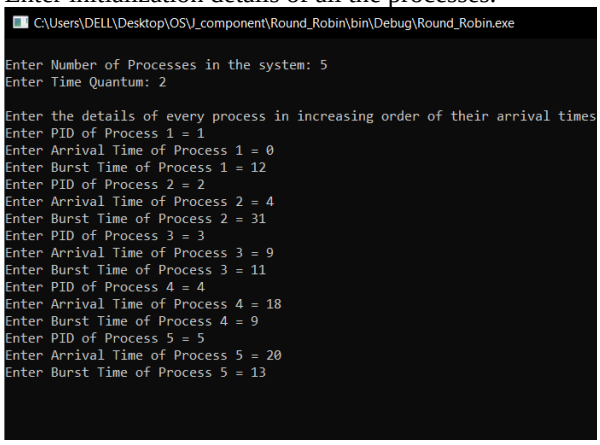
Round Robin

1) Compile and run the program.

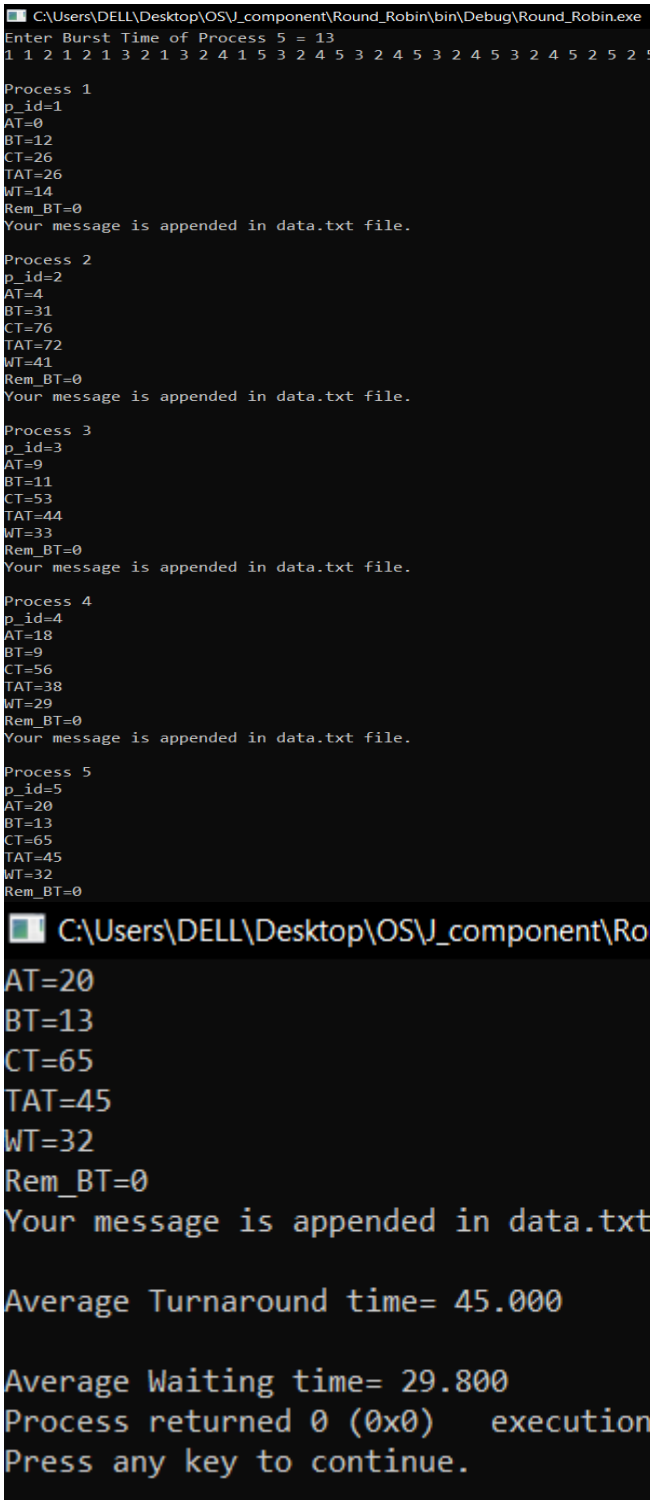
2) Give input for number of tasks and the time quantum of the RR.



3) Enter initialization details of all the processes.



4) Obtain Output



Improved Round Robin

1) Compile and run the program.

2) Give input for number of tasks and the min. time quantum and max. time quantum of the IRR.

```
C:\Users\DELL\Desktop\OS\J_component\Improved_RR\bin\Debug\Imp

Enter Number of Processes in the system: 4
Enter min Time Quantum: 2
Enter max Time Quantum: 8
```

3) Enter initialization details of all the processes.

```
C:\Users\DELL\Desktop\OS\J_component\Improved_RR\bin\De

Enter Number of Processes in the system: 4
Enter min Time Quantum: 2
Enter max Time Quantum: 8

Enter the details of every process in increasing
Enter PID of Process 1 = 1
Enter Arrival Time of Process 1 = 0
Enter Burst Time of Process 1 = 12
Enter PID of Process 2 = 2
Enter Arrival Time of Process 2 = 1
Enter Burst Time of Process 2 = 32
Enter PID of Process 3 = 4
Enter Arrival Time of Process 3 = 5
Enter Burst Time of Process 3 = 3
Enter PID of Process 4 = 5
Enter Arrival Time of Process 4 = 12
Enter Burst Time of Process 4 = 11
```

4) Obtain Output

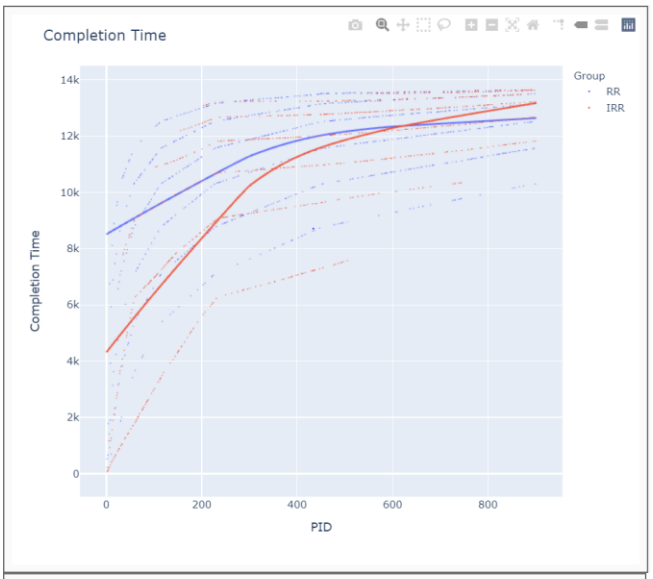
```
C:\Users\DELL\Desktop\OS\J_component\Improved_RR\b
1 2 4 1 5 *5 2 2 2 2 2 2

Process 1
p_id=1
AT=0
BT=12
CT=21
TAT=21
WT=9
Rem_BT=0
Your message is appended in data.txt file.

Process 2
p_id=2
AT=1
BT=32
CT=58
TAT=57
WT=25
Rem_BT=0
Your message is appended in data.txt file.

Process 3
p_id=4
AT=5
BT=3
CT=15
TAT=10
WT=7
Rem_BT=0
Your message is appended in data.txt file.

Process 4
p_id=5
AT=12
BT=11
CT=32
TAT=20
WT=9
Rem_BT=0
Your message is appended in data.txt file.

Average Turnaround time= 27.000

Average Waiting time= 12.500
Process returned 0 (0x0)   execution time : 1
Press any key to continue.
```

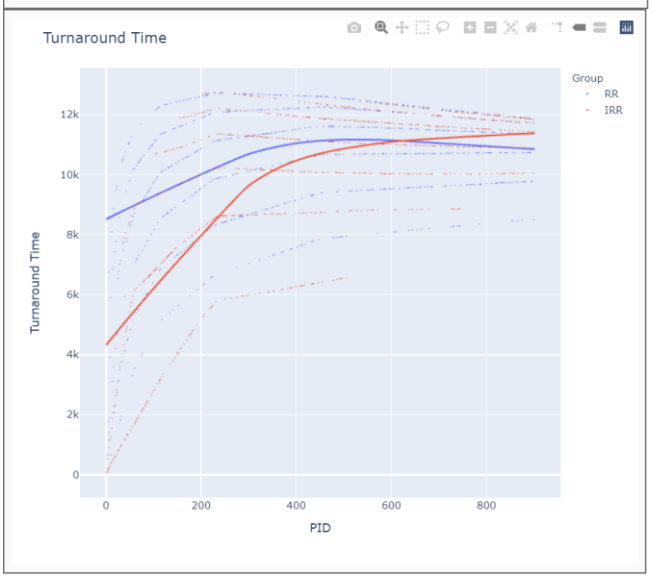
Next step is to run the python file for visualization of mentioned improvements.

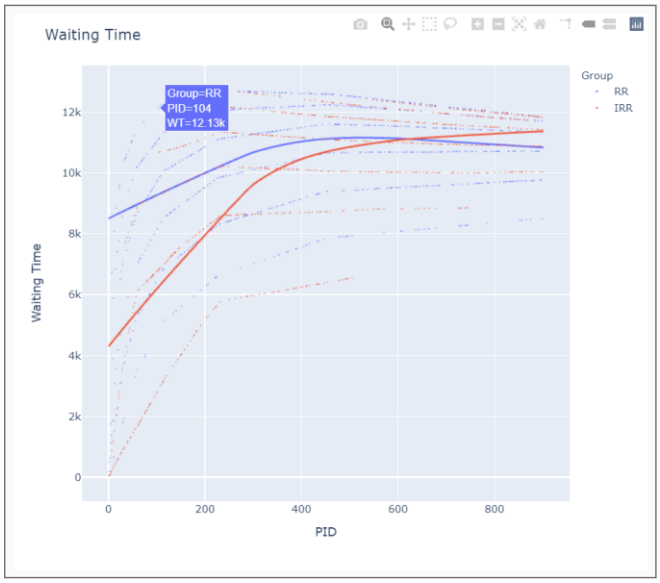
1) Run the python file 'Comparison.py'
2) Wait for it to process the data.txt as csv formatted file and locally save the visualization graphs as html files for interactive and lossless visualization.
3) If you wish to clear out the csv for the creation of new data, choose said option towards the end of the program.

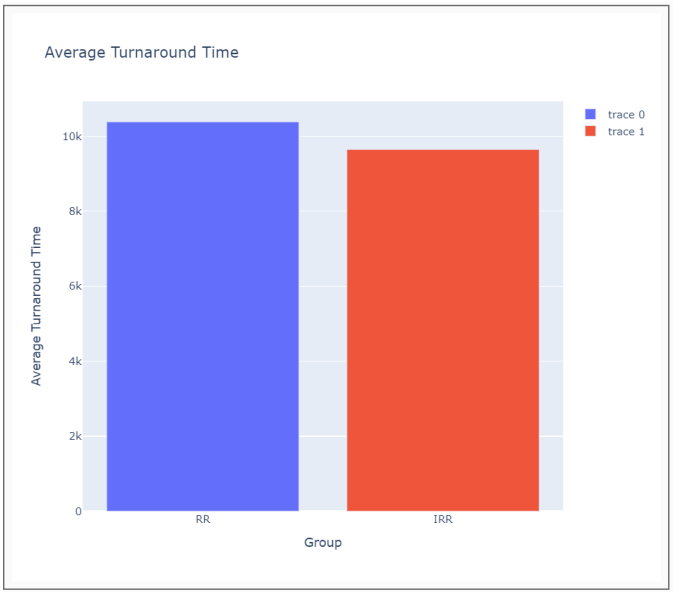
RESULTS AND DISCUSSION

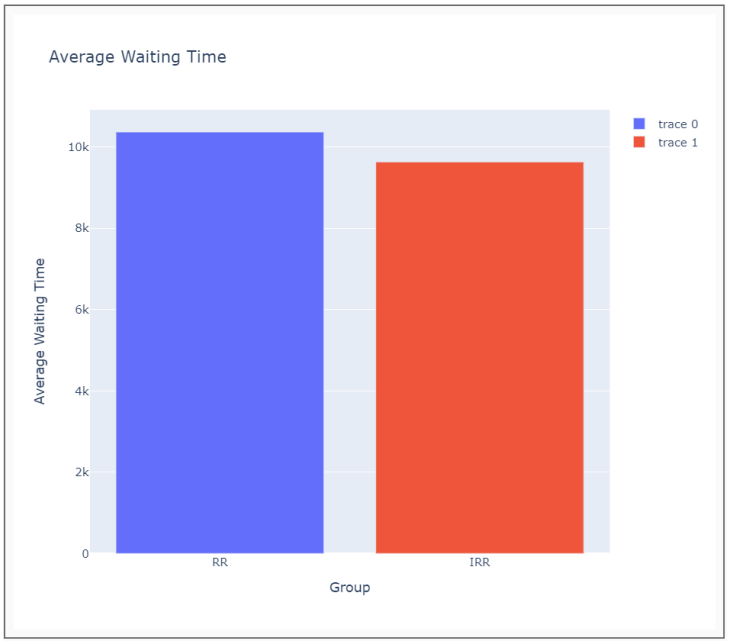
After successful running of the code, we will come across the following graphs:











II. CONCLUSION, LIMITATIONS AND SCOPE FOR FUTURE WORK

In conclusion, it is clear that our proposed methodology toward approaching Round Robin based schedulers is definitely more efficient than traditional approach. The efficiency of our model depends on our selection of time quantum and our algorithm helps to tone that issue down quite considerably since it decides time quantum dynamically, it grants us flexibility with our choice of time quantum for a particular cycle. Average waiting time and turnaround time can be reduced a lot depending on the choice of our minimum and maximum time quantum given us a wider range of time quantum to work on. The more flexible we are with our time quantum range, the better the algorithm acts. For future work, a more statistically appropriate metric can be used to determine time quantum and run to completion check in our algorithm. It can be hybridized with other approaches such as priority queue-based scheduling so that core system applications don't starve for long as can be seen in many already existing operating systems such as windows and Linux. Using statistically complicated but fast calculations and machine learning algorithms for systems performing repeated tasks, we can bypass the need for having to manually determine the range of time quantum whilst opting for the most efficient sequence time quantum to run our processes in a constrained environment with great efficiency.

REFERENCES

1. Sonia Zouaoui , Lotfi Boussaid , Abdellatif Mtibaa (February 2019). PRIORITY BASED ROUND ROBIN CPU SCHEDULING ALGORITHM (PBRR) CPU SCHEDULING ALGORITHM https://www.researchgate.net/publication/332546783_Priority_based_round_robin_PBRR_CPU_scheduling_algorithm

2. Tithi Paul Rahat Hossain Faisal Md. Samsuddoha (September 2019). IMPROVED ROUND ROBIN SCHEDULING ALGORITHM WITH PROGRESSIVE TIME QUANTUM https://www.researchgate.net/publication/335879703_Improved_Round_Robin_Scheduling_Algorithm_with_Progressive_Time_Quantum

3. Sohrawordi , Ehasn Ali , Palash Uddin and Mahabub Hossain (Feb 2019). A MODIFIED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH DYNAMIC TIME QUANTUM. https://www.researchgate.net/publication/332031238_A_MODIFIED_ROUND_ROBIN_CPU_SCHEDULING_ALGORITHM_WITH_DYNAMIC_TIME_QUANTUM

4. Debashree Nayak, Sanjeev Kumar Malla, Debashree Debadarshini (January 2012). IMPROVED ROUND ROBIN SCHEDULING USING DYNAMIC TIME QUANTUM https://www.researchgate.net/publication/258650593_Improved_Round_Robin_Scheduling_using_Dynamic_Time_Quantum#:~:text=Improved%20Round%20Robin%20scheduling%20using%20Dynamic%20Time%20Quantum%20%5B13%5D%20uses,in%20ascending%20order.%20...&text=Scheduling%20is%20considered%20one%20of,of%20real%2Dtime%20embedded%20systems.

5. O. Oyetunji and E. Oluleye (August, 2009). PERFORMANCE ASSESSMENT OF SOME CPU SCHEDULING ALGORITHMS. https://www.researchgate.net/publication/42368534_Performance_Assessment_of_Some_CPU_Scheduling_Algorithms

6. Abdulaziz A. Alsulami , Qasem Abu Al-Haija , Mohammed Thanoon , Qian Mao (April 2019). PERFORMANCE EVALUATION OF DYNAMIC ROUND ROBIN ALGORITHMS FOR CPU SCHEDULING https://ieeexplore.ieee.org/document/9020439

APPENDIX

ROUND ROBIN

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
void file_write(char* str)
{
   FILE *fp;
   fp = fopen("../data.txt", "a");
   fprintf(fp, "\n%s",str);
   printf("Your message is appended in data.txt file.");
   fclose(fp);
}

struct Process {
 int p_id;
```

```
 int AT;
 int BT;
 int rem_BT;
 int CT;
 int TAT;
 int WT;
}buffer;
void SeT(struct Process Ready[], int n){
 buffer.p_id=-1;
 for(int i=0;i<n;i++){
 Ready[i].p_id=-1;
 Ready[i].AT=-1;
 Ready[i].BT=-1;
 Ready[i].rem_BT=-1;
 Ready[i].CT=-1;
 Ready[i].TAT=-1;
 Ready[i].WT=-1;
 }
}void display(struct Process Process_Array[], int n){
 for (int i=0; i<n;i++) {
 int Pno=i+1;
 printf("\n\nProcess %d\n",Process_Array[i].p_id);
 printf("p_id=%d\n",Process_Array[i].p_id);
 printf("AT=%d\n",Process_Array[i].AT);
 printf("BT=%d\n",Process_Array[i].BT);
 printf("CT=%d\n",Process_Array[i].CT);
 printf("TAT=%d\n",Process_Array[i].TAT);
 printf("WT=%d\n",Process_Array[i].WT);
 printf("Rem_BT=%d\n",Process_Array[i].rem_BT);
 char str[300]="";
 char Group[5]="RR,";
 char PID[10];
 char AT[10];
 char BT[10];
 char CT[10];
 char TAT[10];
 char WT[10];
 sprintf(PID, "%d,", Process_Array[i].p_id);
 sprintf(AT, "%d,", Process_Array[i].AT);
 sprintf(BT, "%d,", Process_Array[i].BT);
 sprintf(CT, "%d,", Process_Array[i].CT);
 sprintf(TAT, "%d,", Process_Array[i].TAT);
 sprintf(WT, "%d", Process_Array[i].WT);
 strcat(str, Group);
 strcat(str, PID);
 strcat(str, AT);
 strcat(str, BT);
 strcat(str, CT);
 strcat(str, TAT);
 strcat(str, WT);
 file_write(str);
  }
}
void getStats(struct Process Process_Array[], int n){
 printf("\nEnter the details of every process in increasing order of their arrival times.\n");
 for (int i=0; i<n;i++) {
 printf("Enter PID of Process %d = ",i+1);
 scanf("%d",&Process_Array[i].p_id);
 printf("Enter Arrival Time of Process %d = ",i+1);
 scanf("%d",&Process_Array[i].AT);
 printf("Enter Burst Time of Process %d = ",i+1);
```

```
 scanf("%d",&Process_Array[i].BT);
 Process_Array[i].rem_BT=Process_Array[i].BT;
 }
}
void queue(struct Process Ready[], struct Process Process_Array){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 } Ready[i]=Process_Array;
}
struct Process dequeue(struct Process Ready[]){
 struct Process temp=Ready[0];
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 int j=0;
 for (j; j<i+1;j++){
 Ready[j]=Ready[j+1];
 }
 return temp;
}
int Ready_No(struct Process Ready[]){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 return i;
}
void calcCT(struct Process Process_Array[], struct Process Ready[], int Time_Quantum, int n){
 int timeline=0;
 int counter=0;
 int Process_counter=0;
 while(counter !=n){
 for(int i=Process_counter;i<n;i++){
 if(Process_Array[i].AT<=timeline){ queue(Ready, Process_Array[i]);
 Process_counter++;
 }
 }
 int ReadyQueueCheck=Ready_No(Ready);
 if(ReadyQueueCheck==0){
 timeline++;
 }
 else{
 struct Process temp=dequeue(Ready);
 if(temp.rem_BT<=Time_Quantum){
 for(int i=0;i<n;i++){
 if(Process_Array[i].p_id==temp.p_id){
       printf("%d ",Process_Array[i].p_id);
 timeline=timeline+Process_Array[i].rem_BT;
 Process_Array[i].rem_BT=0;
 Process_Array[i].CT=timeline;
 counter++;
 }
 }
 }
 else{
 for(int i=0;i<n;i++){
 if(Process_Array[i].p_id==temp.p_id){
    printf("%d ",Process_Array[i].p_id);
 timeline=timeline+Time_Quantum;
```

```
 for(int i=Process_counter;i<n;i++){
 if(Process_Array[i].AT<=timeline){
 queue(Ready, Process_Array[i]);
 Process_counter++;
 }
 }
 Process_Array[i].rem_BT=Process_Array[i].rem_BT-Time_Quantum; queue(Ready, Process_Array[i]);
 }
 }
 }
 }
 }
}
void calcTAT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
 }
}
void calcWT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
 }
}
void calcAvgTAT(struct Process Process_Array[],int n){
 float sumTAT=0;
 for (int i=0; i<n;i++) {
 sumTAT = sumTAT + Process_Array[i].TAT;
 }
 printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}
void calcAvgWT(struct Process Process_Array[],int n){
 float sumWT=0;
 for (int i=0; i<n;i++) {
 sumWT = sumWT + Process_Array[i].WT;
 }
 printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}int main(){
 int n, Time_Quantum;
 printf("\nEnter Number of Processes in the system: ");
 scanf ("%d",&n);
 printf("Enter Time Quantum: ");
 scanf("%d",&Time_Quantum);
 struct Process Process_Array[100];
 struct Process Ready[100];
 SeT(Ready, 100);
 struct Process temp=dequeue(Ready);
 getStats(Process_Array, n);
 calcCT(Process_Array,Ready, Time_Quantum, n);
 calcTAT(Process_Array, n);
 calcWT(Process_Array, n);
 display(Process_Array, n);
 calcAvgTAT(Process_Array, n);
 calcAvgWT(Process_Array, n);


 return 0;
}
```

IMPROVED ROUND ROBIN

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
```

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

void file_write(char* str)
{
   FILE *fp;
   fp = fopen("../data.txt", "a");
   fprintf(fp, "\n%s",str);
   printf("Your message is appended in data.txt file.");
   fclose(fp);
}
struct Process {
 int p_id;
 int AT;
 int BT;
 int rem_BT;
 int CT;
 int TAT;
 int WT;
}buffer;
void SeT(struct Process Ready[], int n){
 buffer.p_id=-1;
 for(int i=0;i<n;i++){
 Ready[i].p_id=-1;
 Ready[i].AT=-1;
 Ready[i].BT=-1;
 Ready[i].rem_BT=-1;
 Ready[i].CT=-1;
 Ready[i].TAT=-1;
 Ready[i].WT=-1;
 }
}void display(struct Process Process_Array[], int n){
 for (int i=0; i<n;i++) {
 int Pno=i+1;
 printf("\n\nProcess %d\n",Pno);
 printf("p_id=%d\n",Process_Array[i].p_id);
 printf("AT=%d\n",Process_Array[i].AT);
 printf("BT=%d\n",Process_Array[i].BT);
 printf("CT=%d\n",Process_Array[i].CT);
 printf("'TAT=%d\n",Process_Array[i].TAT);
 printf("WT=%d\n",Process_Array[i].WT);
 printf("Rem_BT=%d\n",Process_Array[i].rem_BT);
 char str[300]="";
 char Group[10]="IRR,";
 char PID[10];
 char AT[10];
 char BT[10];
 char CT[10];
 char TAT[10];
 char WT[10];

 sprintf(PID, "%d,", Process_Array[i].p_id);
 sprintf(AT, "%d,", Process_Array[i].AT);
 sprintf(BT, "%d,", Process_Array[i].BT);
 sprintf(CT, "%d,", Process_Array[i].CT);
 sprintf(TAT, "%d,", Process_Array[i].TAT);
 sprintf(WT, "%d", Process_Array[i].WT);
 strcat(str, Group);
 strcat(str, PID);
 strcat(str, AT);
 strcat(str, BT);
```

```
 strcat(str, CT);
 strcat(str, TAT);
 strcat(str, WT);
 file_write(str);
 }
}
//Input for every process
void getStats(struct Process Process_Array[], int n){
 printf("\nEnter the details of every process in increasing order of their arrival times.\n");
 for (int i=0; i<n;i++) {
 printf("Enter PID of Process %d = ",i+1);
 scanf("%d",&Process_Array[i].p_id);
 printf("Enter Arrival Time of Process %d = ",i+1);
 scanf("%d",&Process_Array[i].AT);
 printf("Enter Burst Time of Process %d = ",i+1);
 scanf("%d",&Process_Array[i].BT);
 Process_Array[i].rem_BT=Process_Array[i].BT;
 }
}
//queue into ready queue
void queue(struct Process Ready[], struct Process Process_Array){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 } Ready[i]=Process_Array;
}
//Dequeue from ready queue
struct Process dequeue(struct Process Ready[]){
 struct Process temp=Ready[0];
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 int j=0;
 for (j; j<i+1;j++){
 Ready[j]=Ready[j+1];
 }
 return temp;
}
//No of elements in ready queue
int Ready_No(struct Process Ready[]){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 return i;
}
void calcCT(struct Process Process_Array[], struct Process Ready[], int min_Time_Quantum, int max_Time_Quantum, int n) {
  int timeline = 0;
  int Time_Quantum=0;
  int counter = 0;
  int Process_counter = 0;
  while (counter != n) {
    for (int i = Process_counter; i < n; i++) {
      if (Process_Array[i].AT <= timeline) {
        queue(Ready, Process_Array[i]);
        Process_counter++;
      }
    }
    int ReadyQueueCheck = Ready_No(Ready);
    if (ReadyQueueCheck == 0) {
```

```
      timeline++;
    }
    else {
      Time_Quantum=DTQ(Ready, Ready_No(Ready),min_Time_Quantum,max_Time_Quantum );
      struct Process temp = dequeue(Ready);
      if (temp.rem_BT <= Time_Quantum) {
        for (int i = 0; i < n; i++) {
          if (Process_Array[i].p_id == temp.p_id) {
             printf("%d ",Process_Array[i].p_id);
            timeline = timeline + Process_Array[i].rem_BT;
            Process_Array[i].rem_BT = 0;
            Process_Array[i].CT = timeline;
            counter++;
          }
        }
      } else {
        for (int i = 0; i < n; i++) {
          if (Process_Array[i].p_id == temp.p_id) {
             printf("%d ",Process_Array[i].p_id);
            timeline = timeline + Time_Quantum;
            for (int i = Process_counter; i < n; i++) {
              if (Process_Array[i].AT <= timeline) {
                queue(Ready, Process_Array[i]);
                Process_counter++;
              }
            }
            Process_Array[i].rem_BT = Process_Array[i].rem_BT - Time_Quantum;
            if (Process_Array[i].rem_BT < (Time_Quantum/2)){
                 printf("*%d ",Process_Array[i].p_id);
                timeline=timeline+Process_Array[i].rem_BT;
                Process_Array[i].rem_BT=0;
                Process_Array[i].CT = timeline;
                counter++;
            }
            else{
                queue(Ready, Process_Array[i]);
            }
          }
        }
      }
    }
  }
}
int DTQ(struct Process Ready[], int  nready, int minTQ, int maxTQ){
    int sum=0;
    int dtq=0;
    for (int i=0; i<nready; i++){
        sum=sum+Ready[i].rem_BT;
    }
    dtq=sum/nready;
    dtq=dtq/2;
    if(dtq<minTQ){
        dtq=minTQ;
    }
    if(dtq>maxTQ){
        dtq=maxTQ;
    }
    return (dtq);
}

void calcTAT(struct Process Process_Array[],int n){
```

```
 for (int i=0; i<n;i++) {
 Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
 }
}
void calcWT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
 }
}
void calcAvgTAT(struct Process Process_Array[],int n){
 float sumTAT=0;
 for (int i=0; i<n;i++) {
 sumTAT = sumTAT + Process_Array[i].TAT;
 }
 printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}
void calcAvgWT(struct Process Process_Array[],int n){
 float sumWT=0;
 for (int i=0; i<n;i++) {
 sumWT = sumWT + Process_Array[i].WT;
 }
 printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}int main(){
 int n, min_Time_Quantum, max_Time_Quantum;
 printf("\nEnter Number of Processes in the system: ");
 scanf ("%d",&n);
 printf("Enter min Time Quantum: ");
 scanf("%d",&min_Time_Quantum);
 printf("Enter max Time Quantum: ");
 scanf("%d",&max_Time_Quantum);
 struct Process Process_Array[100];
 struct Process Ready[100];
 SeT(Ready, 100);
 struct Process temp=dequeue(Ready);
 getStats(Process_Array, n);
 calcCT(Process_Array,Ready, min_Time_Quantum, max_Time_Quantum, n);
 calcTAT(Process_Array, n);
 calcWT(Process_Array, n);
 display(Process_Array, n);
 calcAvgTAT(Process_Array, n);
 calcAvgWT(Process_Array, n);
 return 0;
}
```

Comparison.py

```
import os
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
import os.path
from os import path
patha=os.getcwd()
print(patha)
path_txt=patha+"\data.txt"
path_csv=patha+"\data.csv"
# with open(path+'\data.txt', 'r') as f:
#     print(f.read())
if(os.path.isfile(path_txt)):
    os.rename(path_txt,path_csv)
```

```
df=pd.read_csv(path_csv)
RR=df[df["Group"]=="RR"]
IRR=df[df["Group"]=="IRR"]
print(RR)
print(IRR)

fig = px.scatter(df
                 , x='PID'
                 , y='CT'
                 , color='Group'
                 , trendline='lowess'
                 )
fig.update_layout(
    title="Completion Time",
    xaxis_title="PID",
    yaxis_title="Completion Time"
)
fig.update_traces(marker=dict(size=1.5))
if(os.path.isfile(patha+"\\CT.html")):
    os.remove(patha+"\\CT.html")
fig.write_html(patha+"\\CT.html")

fig1 = px.scatter(df
                 , x='PID'
                 , y='TAT'
                 , color='Group'
                 , trendline='lowess'
                 )
fig1.update_layout(
    title="Turnaround Time",
    xaxis_title="PID",
    yaxis_title="Turnaround Time"
)
fig1.update_traces(marker=dict(size=1.5))
if(os.path.isfile(patha+"\\TAT.html")):
    os.remove(patha+"\\TAT.html")
fig1.write_html(patha+"\\TAT.html")

fig2 = px.scatter(df
                 , x='PID'
                 , y='WT'
                 , color='Group'
                 , trendline='lowess'
                 )
fig2.update_layout(
    title="Waiting Time",
    xaxis_title="PID",
    yaxis_title="Waiting Time"
)
fig2.update_traces(marker=dict(size=1.5))
if(os.path.isfile(patha+"\\WT.html")):
    os.remove(patha+"\\WT.html")
fig2.write_html(patha+"\\WT.html")

rr_avgtat=[]
irr_avgtat=[]
rr_avgwt=[]
irr_avgwt=[]
rr_avgtat.append(RR['TAT'].mean())
rr_avgwt.append(RR['WT'].mean())
irr_avgtat.append(IRR['TAT'].mean())
```

```
irr_avgwt.append(IRR['WT'].mean())

fig3 = go.Figure(data=[
    go.Bar(x=pd.Series(RR['Group']).unique(), y=rr_avgtat),
    go.Bar(x=pd.Series(IRR['Group']).unique(), y=irr_avgtat)
])
fig3.update_layout(
    title="Average Turnaround Time",
    xaxis_title="Group",
    yaxis_title="Average Turnaround Time"
)
if(os.path.isfile(patha+"\\avgTAT.html")):
    os.remove(patha+"\\avgTAT.html")
fig3.write_html(patha+"\\avgTAT.html")

fig4 = go.Figure(data=[
    go.Bar(x=pd.Series(RR['Group']).unique(), y=rr_avgwt),
    go.Bar(x=pd.Series(IRR['Group']).unique(), y=irr_avgwt)
])
fig4.update_layout(
    title="Average Waiting Time",
    xaxis_title="Group",
    yaxis_title="Average Waiting Time"
)
if(os.path.isfile(patha+"\\avgWT.html")):
    os.remove(patha+"\\avgWT.html")
fig4.write_html(patha+"\\avgWT.html")

if(os.path.isfile(path_csv)):
    os.rename(path_csv,path_txt)

choice=input("Clear data for fresh use next time? (y/n): ")
if(choice=='y'):
    if(os.path.isfile(path_txt)):
        os.remove(path_txt)
    if(os.path.isfile(path_csv)):
        os.remove(path_csv)
    with open(path_txt, 'w') as fp:
        fp.write("Group,PID,AT,BT,CT,TAT,WT")
    print("Data deleted")
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .plots{
            background-color: transparent;
            width: 50%;
            min-width: 50%;
            height: calc(25vw * 1.72);
            margin: 20px;
            border: 2px solid rgb(109, 108, 108);

        }
```

```
        .containers{
            width:100%;
            height:fit-content;
            justify-content:center;
            align-items:center;
            display:flex;
            flex-direction: column;
            background-color: rgb(250, 250, 250);
        }
    </style>
</head>
<body>
    <div class="containers">
        <div class=" plots " id="CT">
            <object width=100% height=100% data="CT.html"></object>
        </div>

        <div class="plots " id="TAT">
            <object width=100% height=100% data="TAT.html"></object>
        </div>
        <div class="plots " id="WT">
            <object width=100% height=100% data="WT.html"></object>
        </div>
        <div class="plots " id="avgTAT">
            <object width=100% height=100% data="avgTAT.html"></object>
        </div>
        <div class="plots " id="avgWT">
            <object width=100% height=100% data="avgWT.html"></object>
        </div>
    </div>
</body>
</html>
```

Random value Generator program for comparison

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int RNG(int lower, int upper)
{
    int num = (rand() % (upper - lower + 1)) + lower;
    return num;
}

void file_write(char* str)
{
   FILE *fp;
   fp = fopen("../data.txt", "a");
   fprintf(fp, "\n%s",str);
   printf("Your message is appended in data.txt file.");
   fclose(fp);
}

struct Process {
 int p_id;
 int AT;
 int BT;
 int rem_BT;
```

```
 int CT;
 int TAT;
 int WT;
}buffer;
void SeT(struct Process Ready[], int n){
 buffer.p_id=-1;
 for(int i=0;i<n;i++){
 Ready[i].p_id=-1;
 Ready[i].AT=-1;
 Ready[i].BT=-1;
 Ready[i].rem_BT=-1;
 Ready[i].CT=-1;
 Ready[i].TAT=-1;
 Ready[i].WT=-1;
 }
}
void display(struct Process Process_Array[], int n){
 for (int i=0; i<n;i++) {
 int Pno=i+1;
 printf("\n\nProcess %d\n",Pno);
 printf("p_id=%d\n",Process_Array[i].p_id);
 printf("AT=%d\n",Process_Array[i].AT);
 printf("BT=%d\n",Process_Array[i].BT);
 printf("CT=%d\n",Process_Array[i].CT);
 printf("TAT=%d\n",Process_Array[i].TAT);
 printf("WT=%d\n",Process_Array[i].WT);
 printf("Rem_BT=%d\n",Process_Array[i].rem_BT);
 char str[300]="";
 char Group[10]="IRR,";
 char PID[10];
 char AT[10];
 char BT[10];
 char CT[10];
 char TAT[10];
 char WT[10];

 sprintf(PID, "%d,", Process_Array[i].p_id);
 sprintf(AT, "%d,", Process_Array[i].AT);
 sprintf(BT, "%d,", Process_Array[i].BT);
 sprintf(CT, "%d,", Process_Array[i].CT);
 sprintf(TAT, "%d,", Process_Array[i].TAT);
 sprintf(WT, "%d", Process_Array[i].WT);
 strcat(str, Group);
 strcat(str, PID);
 strcat(str, AT);
 strcat(str, BT);
 strcat(str, CT);
 strcat(str, TAT);
 strcat(str, WT);
 file_write(str);
 }
}
void displayRR(struct Process Process_Array[], int n){
 for (int i=0; i<n;i++) {
 int Pno=i+1;
 printf("\n\nProcess %d\n",Process_Array[i].p_id);
 printf("p_id=%d\n",Process_Array[i].p_id);
 printf("AT=%d\n",Process_Array[i].AT);
 printf("BT=%d\n",Process_Array[i].BT);
 printf("CT=%d\n",Process_Array[i].CT);
 printf("TAT=%d\n",Process_Array[i].TAT);
```

```
 printf("WT=%d\n",Process_Array[i].WT);
 printf("Rem_BT=%d\n",Process_Array[i].rem_BT);
 char str[300]="";
 char Group[5]="RR,";
 char PID[10];
 char AT[10];
 char BT[10];
 char CT[10];
 char TAT[10];
 char WT[10];
 sprintf(PID, "%d,", Process_Array[i].p_id);
 sprintf(AT, "%d,", Process_Array[i].AT);
 sprintf(BT, "%d,", Process_Array[i].BT);
 sprintf(CT, "%d,", Process_Array[i].CT);
 sprintf(TAT, "%d,", Process_Array[i].TAT);
 sprintf(WT, "%d", Process_Array[i].WT);
 strcat(str, Group);
 strcat(str, PID);
 strcat(str, AT);
 strcat(str, BT);
 strcat(str, CT);
 strcat(str, TAT);
 strcat(str, WT);
 file_write(str);
  }
}
void getStats(struct Process Process_Array[], int n){
 printf("\nEnter the details of every process in increasing order of their arrival times.\n");
 for (int i=0; i<n;i++) {
 Process_Array[i].p_id=i+1;
 Process_Array[i].AT=2*i;
 Process_Array[i].BT=RNG(10,100);
 Process_Array[i].rem_BT=Process_Array[i].BT;
 }
}
void queue(struct Process Ready[], struct Process Process_Array){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 } Ready[i]=Process_Array;
}
struct Process dequeue(struct Process Ready[]){
 struct Process temp=Ready[0];
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 int j=0;
 for (j; j<i+1;j++){
 Ready[j]=Ready[j+1];
 }
 return temp;
}
int Ready_No(struct Process Ready[]){
 int i=0;
 while(Ready[i].p_id!=-1){
 i++;
 }
 return i;
}
void calcRRCT(struct Process Process_Array[], struct Process Ready[], int Time_Quantum, int n){
```

```
 int timeline=0;
 int counter=0;
 int Process_counter=0;
 while(counter !=n){
 for(int i=Process_counter;i<n;i++){
 if(Process_Array[i].AT<=timeline){ queue(Ready, Process_Array[i]);
 Process_counter++;
 }
 }
 int ReadyQueueCheck=Ready_No(Ready);
 if(ReadyQueueCheck==0){
 timeline++;
 }
 else{
 struct Process temp=dequeue(Ready);
 if(temp.rem_BT<=Time_Quantum){
 for(int i=0;i<n;i++){
 if(Process_Array[i].p_id==temp.p_id){
       printf("%d ",Process_Array[i].p_id);
 timeline=timeline+Process_Array[i].rem_BT;
 Process_Array[i].rem_BT=0;
 Process_Array[i].CT=timeline;
 counter++;
 }
 }
 }
 else{
 for(int i=0;i<n;i++){
 if(Process_Array[i].p_id==temp.p_id){
    printf("%d ",Process_Array[i].p_id);
 timeline=timeline+Time_Quantum;
 for(int i=Process_counter;i<n;i++){
 if(Process_Array[i].AT<=timeline){
 queue(Ready, Process_Array[i]);
 Process_counter++;
 }
 }
 Process_Array[i].rem_BT=Process_Array[i].rem_BT-Time_Quantum; queue(Ready, Process_Array[i]);
 }
 }
 }
 }
 }
}
void calcRRTAT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
 }
}
void calcRRWT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
 }
}
void calcRRAvgTAT(struct Process Process_Array[],int n){
 float sumTAT=0;
 for (int i=0; i<n;i++) {
 sumTAT = sumTAT + Process_Array[i].TAT;
 }
 printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}
```

```
void calcRRAvgWT(struct Process Process_Array[],int n){
 float sumWT=0;
 for (int i=0; i<n;i++) {
 sumWT = sumWT + Process_Array[i].WT;
 }
 printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}
void calcCT(struct Process Process_Array[], struct Process Ready[], int min_Time_Quantum, int max_Time_Quantum, int n) {
  int timeline = 0;
  int Time_Quantum=0;
  int counter = 0;
  int Process_counter = 0;
  while (counter != n) {
    for (int i = Process_counter; i < n; i++) {
      if (Process_Array[i].AT <= timeline) {
        queue(Ready, Process_Array[i]);
        Process_counter++;
      }
    }
    int ReadyQueueCheck = Ready_No(Ready);
    if (ReadyQueueCheck == 0) {
      timeline++;
    }
    else {
      Time_Quantum=DTQ(Ready, Ready_No(Ready),min_Time_Quantum,max_Time_Quantum );
      struct Process temp = dequeue(Ready);
      if (temp.rem_BT <= Time_Quantum) {
        for (int i = 0; i < n; i++) {
          if (Process_Array[i].p_id == temp.p_id) {
             printf("%d ",Process_Array[i].p_id);
            timeline = timeline + Process_Array[i].rem_BT;
            Process_Array[i].rem_BT = 0;
            Process_Array[i].CT = timeline;
            counter++;
          }
        }
      } else {
        for (int i = 0; i < n; i++) {
          if (Process_Array[i].p_id == temp.p_id) {
             printf("%d ",Process_Array[i].p_id);
            timeline = timeline + Time_Quantum;
            for (int i = Process_counter; i < n; i++) {
              if (Process_Array[i].AT <= timeline) {
                queue(Ready, Process_Array[i]);
                Process_counter++;
              }
            }
            Process_Array[i].rem_BT = Process_Array[i].rem_BT - Time_Quantum;
            if (Process_Array[i].rem_BT < (Time_Quantum/2)){
                 printf("*%d ",Process_Array[i].p_id);
                timeline=timeline+Process_Array[i].rem_BT;
                Process_Array[i].rem_BT=0;
                Process_Array[i].CT = timeline;
                counter++;
            }
            else{
                queue(Ready, Process_Array[i]);
            }
          }
        }
      }
```

```
    }
  }
}
int DTQ(struct Process Ready[], int  nready, int minTQ, int maxTQ){
    int sum=0;
    int dtq=0;
    for (int i=0; i<nready; i++){
        sum=sum+Ready[i].rem_BT;
    }
    dtq=sum/nready;
    dtq=dtq/2;
    if(dtq<minTQ){
        dtq=minTQ;
    }
    if(dtq>maxTQ){
        dtq=maxTQ;
    }
    return (dtq);
}

void calcTAT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].TAT = Process_Array[i].CT - Process_Array[i].AT;
 }
}
void calcWT(struct Process Process_Array[],int n){
 for (int i=0; i<n;i++) {
 Process_Array[i].WT = Process_Array[i].TAT - Process_Array[i].BT;
 }
}
void calcAvgTAT(struct Process Process_Array[],int n){
 float sumTAT=0;
 for (int i=0; i<n;i++) {
 sumTAT = sumTAT + Process_Array[i].TAT;
 }
 printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));
}
void calcAvgWT(struct Process Process_Array[],int n){
 float sumWT=0;
 for (int i=0; i<n;i++) {
 sumWT = sumWT + Process_Array[i].WT;
 }
 printf("\n\nAverage Waiting time= %.3f", (sumWT/n));
}


int main(){
 srand(time(NULL));
 int n, Time_Quantum, min_Time_Quantum, max_Time_Quantum;
 printf("\nEnter Number of Processes in the system: ");
 scanf ("%d",&n);
 printf("Enter Time Quantum: ");
 scanf("%d",&Time_Quantum);
 struct Process Process_Array[1000];
 struct Process Process_ArrayIRR[1000];
 struct Process Ready[1000];
 struct Process ReadyIRR[1000];
 SeT(ReadyIRR,1000);
 SeT(Ready, 1000);
 struct Process temp=dequeue(Ready);
 getStats(Process_Array, n);
```

```
for (int i=0; i<n;i++){
    Process_ArrayIRR[i]=Process_Array[i];
}
calcRRCT(Process_Array,Ready, Time_Quantum, n);
calcRRTAT(Process_Array, n);
calcRRWT(Process_Array, n);
displayRR(Process_Array, n);
calcRRAvgTAT(Process_Array, n);
calcRRAvgWT(Process_Array, n);
 printf("Enter min Time Quantum: ");
scanf("%d",&min_Time_Quantum);
printf("Enter max Time Quantum: ");
scanf("%d",&max_Time_Quantum);

temp=dequeue(ReadyIRR);
calcCT(Process_ArrayIRR,ReadyIRR, min_Time_Quantum, max_Time_Quantum, n);
calcTAT(Process_ArrayIRR, n);
calcWT(Process_ArrayIRR, n);
display(Process_ArrayIRR, n);
calcAvgTAT(Process_ArrayIRR, n);
calcAvgWT(Process_ArrayIRR, n);
return 0;
}
```