AHMEDABAD UNIVERSITY

School of Engineering and Applied Science

CSC341- Operating Systems Lab

Submission of Project Report

November 17, 2017

Project Title: ***CPU Scheduling Algorithm***

Group Number

02

Group Members

Charmi Chokshi (201501021)

Hena Ghonia (201501032)

Table of Content

Brief Description

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc., thereby making full use of CPU. **The aim of CPU scheduling is to make the system efficient, fast and fair**.

There are many different criteria to check when considering the "**best**" scheduling algorithm:

- CPU utilization
- Throughput
- Waiting time
- Turnaround time
- Starvation
- Context Switching
- Possibility of Deadlock

This project is aimed at **Implementation and Analysis** of the following CPU scheduling Algorithms based on above criteria:

Uniprocessor Scheduling Algorithms:

- **First-Come-First-Served (FCFS)**
- **Round Robin (RR)**
- **Shortest-Job-First (SJF)**
- **Priority Scheduling (PS)**

Real Time Uniprocessor Scheduling Algorithms:

- **Earliest Deadline First (EDF)**
- **Rate Monotonic Scheduling (RMS)**

The project also include implementation of **Modified Round Robin Algorithm**. Based on done analysis, it has approx. 40% higher throughput, approx. 20% less average wait time and average turnaround time and 30-50% less context switches as compared to simple RR algorithm.

There is a **Simulator** which runs different CPU scheduling algorithms and gives different Utilization matrices:

- Waiting time of each process and average waiting time
- Turn-around time of each process and average turnaround time
- Number of context switches in case of RR and Modified RR algorithms
- Throughput for given time span
- Gantt Chart

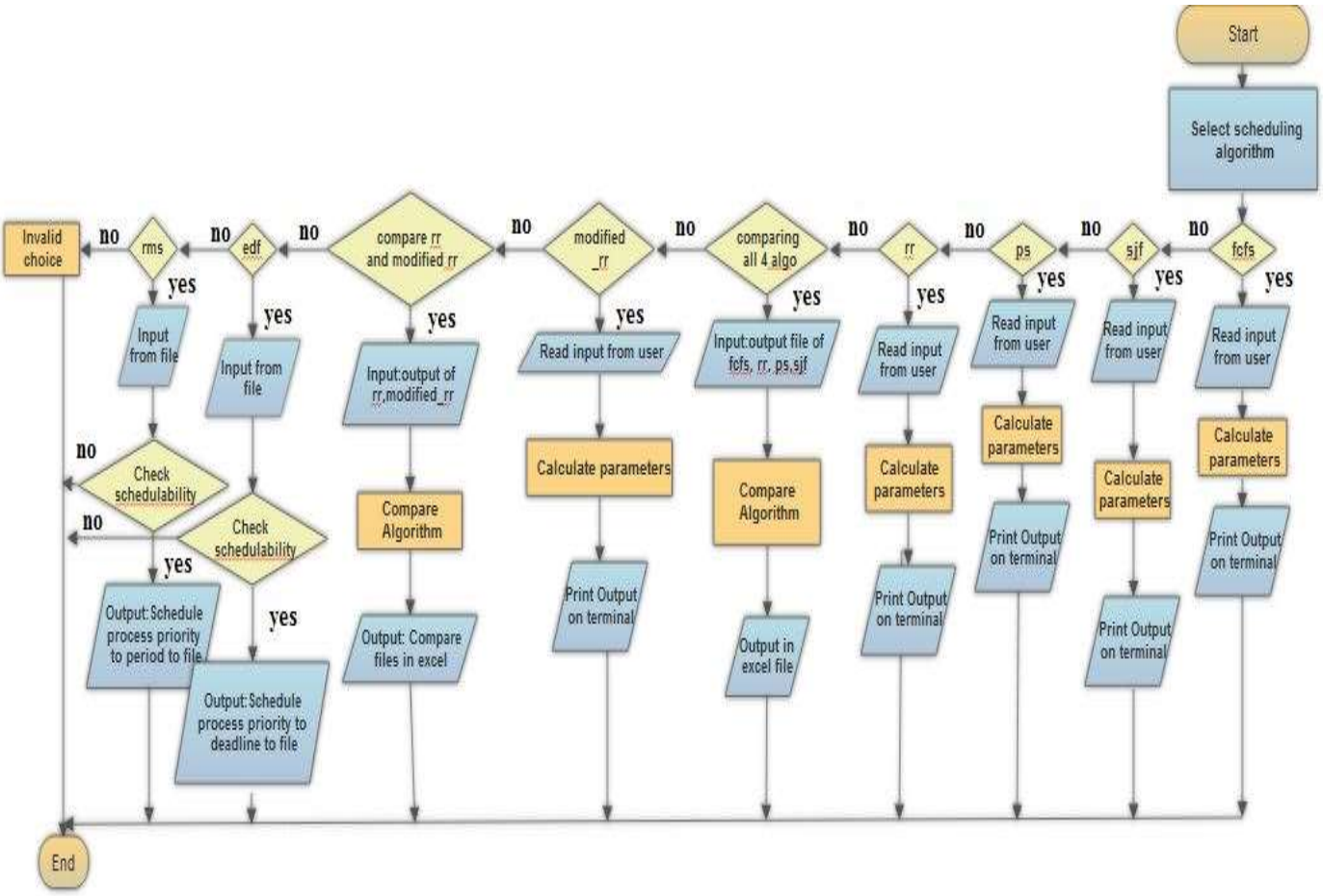
The project also include comparison of above mentioned 4 Uniprocessor Scheduling algorithms as well as comparison of Round Robin and Modified Round Robin algorithm for different test sets and gives output in form of Bar Graphs.

Architecture/ Diagram

Main Menu of project

```
        ***WELCOME TO THE CPU SCHEDULING ALGORITHM SIMULATOR***

              ~~~MENU~~~

     1. First Come First Serve (FCFS)
     2. Shortest Job First (SJF)
     3. Priority Scheduling (PS)
     4. Round Robin (RR)
     5. Compare All 4 Algorithms
     6. Modified Round Robin
     7. Compare RR and Modified RR
     8. Earliest Deadline First (EDF)
     9. Rate Monotonic Scheduling (RMS)
     0. Exit

Which Algorithm/ Option you want to Choose? 0

        ***THANK YOU FOR USING CPU SCHEDULING ALGORITHM SIMULATOR***
```

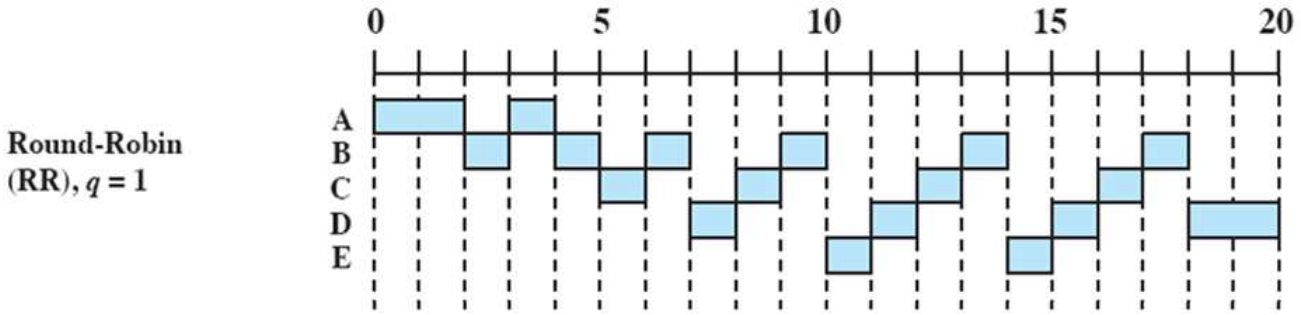
This Menu follows Flowchart given below to generate proper output



- Here, the scheduling techniques are implemented using the **C Programing Language**.
- This project has one main.c file which contains header files of all other algorithms to be implemented.
- **Simulation** of various algorithms generate output on terminal when user gives input.
- For comparison of different algorithms, pthread is used to ac-hieve parallelism. Its input is given from file and output is in form of Bar Graphs which are generated using **Libxlsxwriter** which is **a C library for creating Excel XLSX files**.
- To detect and remove Deadlocks and to avoid Floating Point Errors such as, division by 0, Signaling concept is being used. Its output will be printed on terminal as well as in text file.
- Interrupt handling is also used to avoid unwanted behavior of program at the time of interrupt generation such as ctrl + C or ctrl + Z.

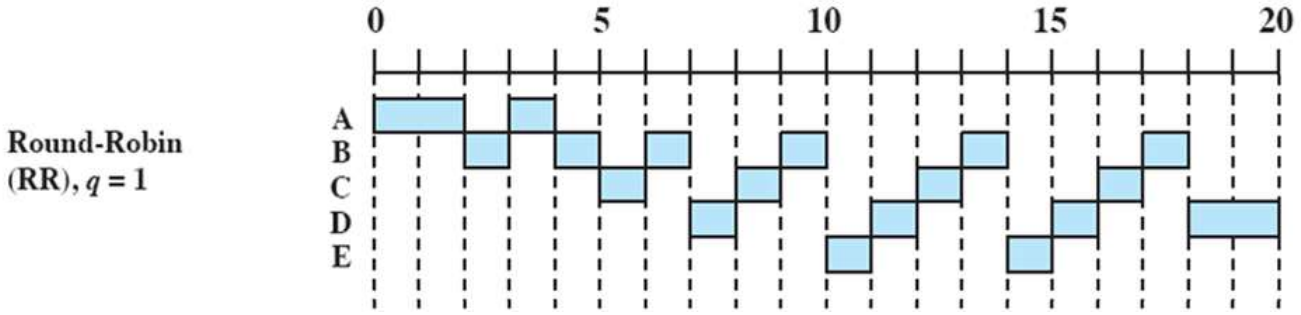
Architecture of implemented Algorithms

First Come First Served:



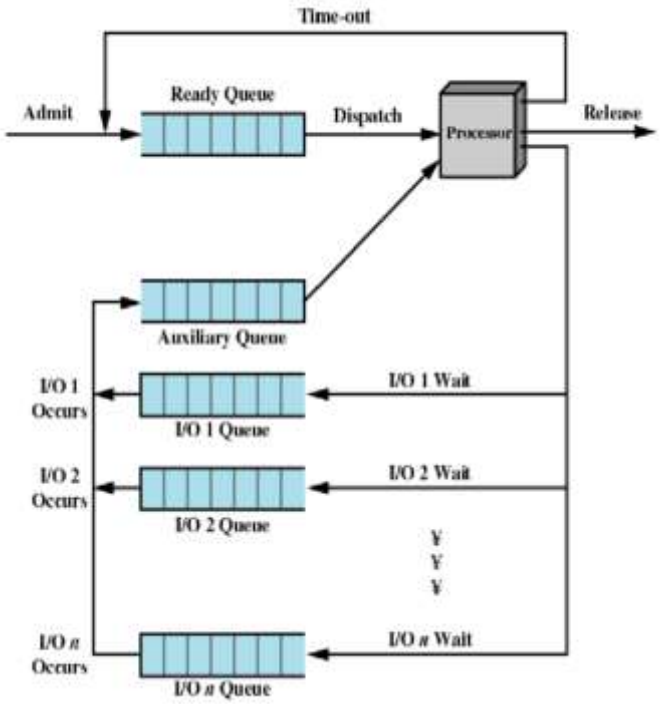
- It is a **non-preemptive** scheduling algorithm.
- Jobs are executed on first come, first served basis.
- Poor in performance as average wait time is high.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- The First Come First Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

Round Robin:

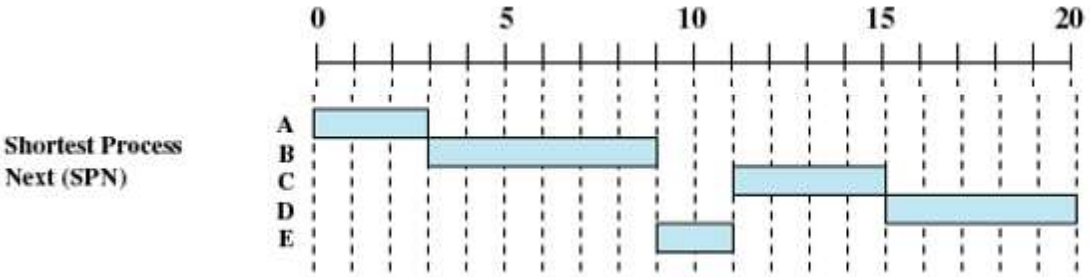


- Round Robin is the **preemptive** process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.
- This method is quite same as the FCFS but the difference is the in this case the processor will not process the whole job (process) at a time.

- **Auxiliary queue:** The new feature is an FCFS auxiliary queue to which processes are moved after being released from an I/O block.
- When a dispatching decision is to be made, processes in the auxiliary queue get preference over those in the main ready queue.
- It runs no longer than a time equal to the basic time quantum minus the total time spent running since it was last selected from the main ready queue.



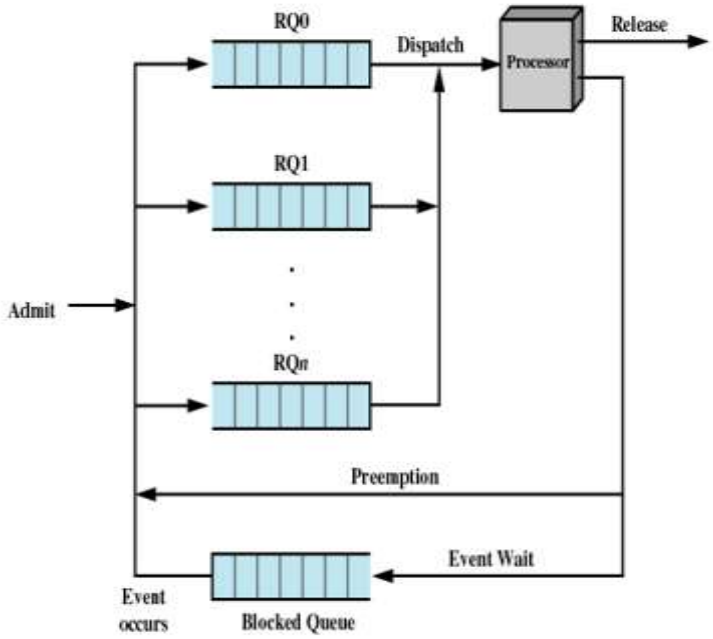
Shortest Job First:



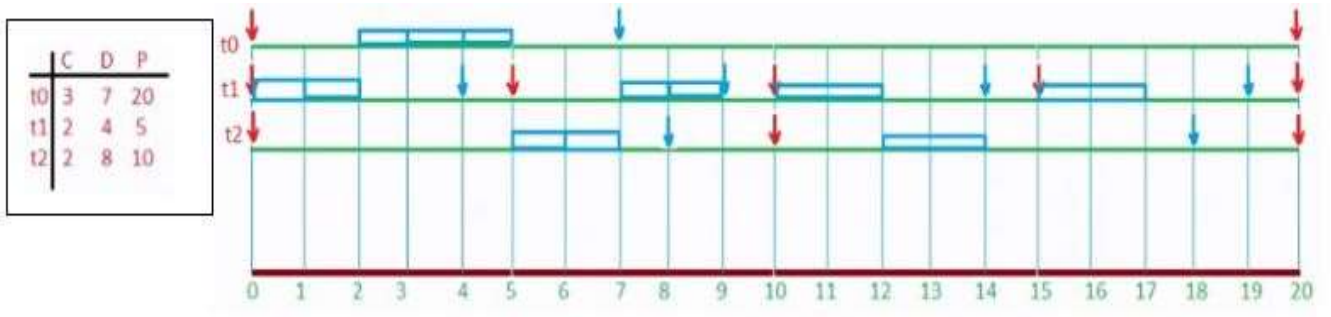
- This is also known as shortest job Next, or SJ
- This is a non-preemptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.
  This method is quite same as the FCFS but the difference is the in this case the processor will not process the jobs (processes) as they will come.

Priority Scheduling:

- Priority scheduling is **a non-preemptive** algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm.

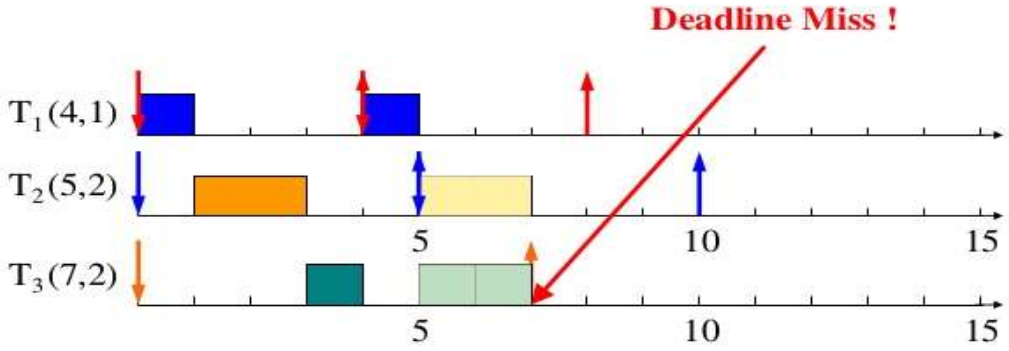


Earliest Deadline First:



- It uses dynamic priority assignment
- Priority is given to the process which is ready and has to complete first i.e tasks with earlier deadlines will be executed at higher priority.
- The current executed task is preempted whenever another periodic instance with earlier deadline becomes active.
- EDF has a utilization bound of 100%.
- The algorithm is also difficult to implement in hardware and there is a tricky issue of representing deadlines in different ranges

Rate Monotonic Scheduling:



- Assume n tasks running periodically
- Periods T1,T2,...Tn
- Worst-case execution time(burst time): C1,C2,...Cn
- Pre-emptive with fixed/static priority
- Theorem: If any static priority yields a feasible schedule, then priorities ordered by period (smallest period has highest priority) also yields a feasible schedule.

Technical Specifications/ Details

Assumptions:

- Arrival time for each processes in FCFS, PS, RR, SJF and Modified RR Algorithm is 0 ms
- Deadline and Time Periods in EDF and RMS Algorithm are of type integer
- Number of processes to compare all 4 algorithms is set to 4
- Number of processes to compare RR and Modified RR is set to 5
- The value of Time Quantum in Modified RR is doubled after first iteration

Used Operating Systems Concepts:

- **Uniprocessor Scheduling:** A multiprogramming system is a basic form of parallel processing in which multiple programs are run at the same time on a uniprocessor. As a result of having only a single processor, concurrent execution of multiple programs is impossible. Rather, the operating system must alternate between the programs, e.g. giving some processing time to one process, then switching to a different process.
- **Interrupt Handling**: An interrupt is a signal from a device attached to a computer or from a program within the computer that requires the operating system to stop and figure out what to do next. After the interrupt signal is sensed, the computer either resumes running the current program or begins running another program.
    - Interrupt handling is used using signals such as SIGINT and SIGTSTP to avoid unwanted behavior of program at the time of interrupt generation such as ctrl + C or ctrl + Z.
- **Inter Process Communication (Using Signalling)**: Signals are a limited form of inter-process communication (IPC), typically used in Unix, Unix-like operating systems. A signal is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.
    - To avoid Floating Point Errors such as, division by 0, SIGFPE is used in this project.
- **POSIX Thread:** POSIX Threads, usually referred to as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API.
    - To compare all 4 uniprocessor scheduling algorithms at run time, 4 threads are created
- **File Management:** A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.
    - At run time, take inputs such as number of processes, burst time, throughput etc. and after execution, to store the output, text files are being used.

- **Context Switching:** In computing, a context switch is the process of storing and restoring the state (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time. This enables multiple processes to share a single CPU and is an essential feature of a multitasking operating system.
- **Starvation:** In computer science, starvation is a problem encountered in concurrent computing where a process is perpetually denied necessary resources to process its work. Starvation may be caused by errors in a scheduling or mutual exclusion algorithm, but can also be caused by resource leaks.
    - Here, the process which has to wait for longer period of time to get the CPU access is process which starves the most.
- **Possibility of Deadlock:** Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.
    - In EDF and RMS Algorithms, when more than 1 processes arrive at the same time, there is a possibility of deadlock.
    - This is avoided using Hold and Wait which means put one process in wait queue while the other is executing. The selection of process to be executed first is based on priority such as earliest deadline or shortest period.
- **CPU Utilization:** To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput:** It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.
- **Turnaround Time:** It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process (Wall clock time).
- **Waiting Time:** The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Advantages and Disadvantages of various algorithms based on done Analysis

First Come First Serve (FCFS)

- **Advantages:**
    - FCFS algorithm doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one.
    - Hence, FCFS is pretty simple and easy to implement.
    - Eventually, every process will get a chance to run, so starvation doesn't occur.
- **Disadvantages:**
    - There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends.

- Because there is no pre-emption, if a process executes for a long time, the processes in the back of the queue will have to wait for a long time before they get a chance to be executed.

Shortest Job First (SJF)

- **Advantages:**
  - According to the definition, short processes are executed first and then followed by longer processes.
  - The throughput is increased because more processes can be executed in less amount of time.
- **Disadvantages:**
  - The time taken by a process must be known by the CPU beforehand, which is not possible.
  - Longer processes will have more waiting time, eventually they'll suffer starvation.

Priority Scheduling (PS)

- **Advantages:**
  - The priority of a process can be selected based on memory requirement, time requirement or user preference. For example, a high end game will have better graphics that means the process which updates the screen in a game will have higher priority so as to achieve better graphics performance.
- **Disadvantages:**
  - A second scheduling algorithm is required to schedule the processes which have same priority.
  - In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process. If lower priority process keeps waiting for higher priority processes, starvation occurs.

Earliest Deadline First (EDF)

- **Advantages:**
  - EDF scheduler uses 100% of CPU resource.
  - EDF is a global scheduler. If any task is not schedulable by EDF then it can not be scheduled in any algorithm.
- **Disadvantages:**
  - It needs priority queue to store deadlines, needs dynamic priorities
  - It behaves badly under overload (a jobs that missed deadline is allowed to continue-domino effect). The arrival of a new job may result in all the previous jobs missing their deadlines. Such an undesirable phenomenon, called the Domino Effect

Rate Monotonic Scheduling (RMS)

- **Advantages:**
  - Static priorities assigned according to the rate monotonic conventions (tasks with shorter
  - periods/deadlines are given higher priorities)

- **Disadvantages:**
    - Fragile - overrun may cause whole schedule to fail
    - It is In flexible and difficult to modify
    - Not very suitable for systems with both periodic and a periodic tasks.

Round Robin (RR)

- **Advantages:**
    - Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.
    - Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind.
- **Disadvantages:**
    - The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.
    - If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.

Why we opt for modifying RR Algorithm?

Round Robin reduces the penalty that short jobs suffer with FCFS by preempting running jobs periodically, and also saves starving of longer jobs and scheduling effort in case of SJF. The main advantage of Round Robin Scheduling is that every process gets the CPU and thus there is no starvation.

But it has some limitations which can be improved:

- Higher wait time
- Higher turnaround time
- Low throughput
- More context switches

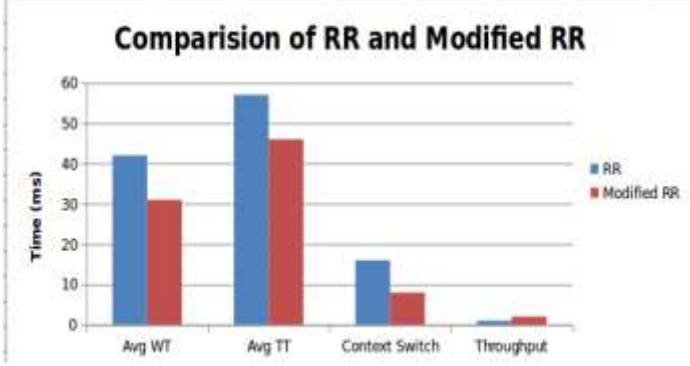
So, we have proposed modified round robin algorithm which gives better performance on above criterias.

Modified Round Robin

- Modified Round Robin is an amalgamation of:
    - Traditional Round Robin Algorithm
    - Priority Scheduling Algorithm
    - Also it uses concept of change in quantum time

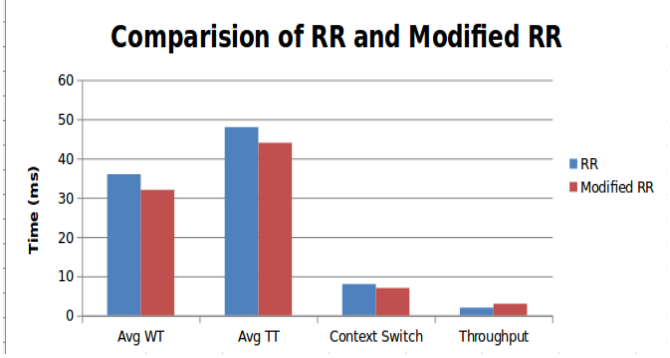
Input: 1
Number of processes: 5
Burst time of processes: 22, 18, 21, 10, 5
Time Quantum: 5
Throughput to test: 30 ms

Process	RR	Modified RR
Avg WT	42	31
Avg TT	57	46
Context Switch	16	8
Throughput	1	2



Input: 2
Number of processes: 5
Burst time of processes: 22, 9, 11, 17, 5
Time Quantum: 10
Throughput to test: 45 ms

Process	RR	Modified RR
Avg WT	36	32
Avg TT	48	44
Context Switch	8	7
Throughput	2	3



Thus, proposed algorithm has following performance matrix as compare to traditional Round Robin Algorithm:

- **Approximately 20% less average Waiting Time**
- **Approximately 15% less average Turnaround Time**
- **Approximately 30-40% less Context Switches**
- **Approximately 50-70% higher Throughput**

Algorithms

First Come First Served:

1- Input the processes along with their burst time (bt).
2- Find waiting time (wt) for all processes.
3- As first process that comes need not to wait so waiting time for process 1 will be 0 i.e. wt[0] = 0.
4- Find waiting time for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] .
5- Find turnaround time = waiting_time + burst_time for all processes.
6- Find average waiting time = total_waiting_time / no_of_processes.
7- Similarly, find average turnaround time = total_turn_around_time / no_of_processes.

Shorted Job First:

1- Sort all the processes in increasing order according to burst time.
2- Then simply, apply FCFS.

Round Robin:

1- Create an array rem_bt[] to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
  a- If rem_bt[i] > quantum
    (i) t = t + quantum
    (ii) bt_rem[i] -= quantum;
  c- Else // Last cycle for this process
    (i) t = t + bt_rem[i];
    (ii) wt[i] = t - bt[i]
    (ii) bt_rem[i] = 0; // This process is over

Priority Scheduling:

1- First input the processes with their burst time and priority.
2- Sort the processes, burst time and priority according to the priority.
3- Now simply apply FCFS algorithm

Modified Round Robin:

1- Allocate every process to CPU, a single time by applying RR scheduling with a initial time quantum (say k units).
2- After completing first cycle perform the following steps:
    a) Double the initial time quantum (2k units).
    b) Select the shortest process from the waiting queue and assign to CPU.
    c) After that we have to select the next shortest process for execution by excluding the already executed one in this phase.
3- For the complete execution of all the processes we have to repeat steps 1 and 2 in cycle.

Earliest Deadline First:

1- Input number_process(n), execution_time(C[]),deadline(d[]) period(T[])
2- Check feasibility test for utilization(U)

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

3- If it satisfies then calculate LCM(least common multiple) of period(T[]) and call schedule_rms() else print system is not schedulable
4- Initialize earlydeadline=LCM, index=-1, remainsCapacity[]=C[], nextdeadline[]=d[], OnlineNewPeriod[]=0
5- Repeat the steps until i=LCM
    Print i,i+1(time interval)
    Repeat steps until j=n
        if remainCapacity(j)>0 and earlydeadline>nextdeadline(j) then
        earlydeadline=nextdeadline[j],
        index=j
    print index(process number)
    decreament remainCapacity(index) by 1
6- Repeat steps until j=n
    if OnlineNewPeriod(j)==T(j)-1
        then nextdeadline[j]=d[j]

remainsCapacity[j]=C[j]

OnlineNewPeriod[j]=0

Else

If nextdeadline(j)>0 then decreament nextdeadline(j) by 1

Else if remainCapacity(j)>0

then print process has no chance to complete its capacity

Increament OnlineNewPeriod(j) by 1

Rate Monotonic Scheduling:

1- Input number_process(n), execution_time(C[]), period(T[])

2- Check feasibilty test for utilization(U)

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

3- If it satisfies then calculate LCM(least common multiple) of period(T[]) and call schedule_rms() else print system is not schedulable

4- Initialize earlyrate=LCM, index=-1, remainsCapacity[]=C[], nextrate[]=T[], OnlineNewPeriod[]=0

5- Repeat the steps until i=LCM

Print i,i+1(time interval)

Repeat steps until j=n

if remainCapacity(j)>0 and earlyrate>nextrate(j) then earlyrate=nextrate[j],

index=j

print index(process number)

decreament remainCapacity(index) by 1

6- Repeat steps until j=n

if OnlineNewPeriod(j)==T(j)-1

then nextrate[j]=T[j]

remainsCapacity[j]=C[j]

OnlineNewPeriod[j]=0

Else

If nextrate(j)>0 then decreament nextrate(j) by 1

Else if remainCapacity(j)>0

then print process has no chance to complete its capacity

Increament OnlineNewPeriod(j) by 1

List of Programs and Link to Source Code

- Click here to view all the source codes listed below:

- **main.c**: main C file which includes all the header files listed below
- **fcfs.h**: header file to implement FCFS algorithm
- **rr.h**: header file to implement RR algorithm
- **sjf.h**: header file to implement SJF algorithm
- **ps.h**: header file to implement PS algorithm
- **modified_rr.h**: header file to implement Modified RR algorithm
- **edf.h:** header file to implement EDF algorithm
- **rms.h:** header file to implement RMS algorithm
- **thread.h:** header file to generate threads
- **rr_mrr_cmp.h:** header file to compare RR and Modified RR algorithms
- **plot_cmp_algos.h:** header file to plot graphs of comparison of algorithms
- **plot_rr_mrr.h:** header file to plot graphs of comparison of RR and Modified RR algorithms

- Click here to view all the Input data sets
- Click here to view all the Output data sets
- Click here to view whole Project

Test Results

Main Menu

```
        ***WELCOME TO THE CPU SCHEDULING ALGORITHM SIMULATOR***

              ~~~MENU~~~

     1. First Come First Serve (FCFS)
     2. Shortest Job First (SJF)
     3. Priority Scheduling (PS)
     4. Round Robin (RR)
     5. Compare All 4 Algorithms
     6. Modified Round Robin
     7. Compare RR and Modified RR
     8. Earliest Deadline First (EDF)
     9. Rate Monotonic Scheduling (RMS)
     0. Exit

Which Algorithm/ Option you want to Choose? 0

        ***THANK YOU FOR USING CPU SCHEDULING ALGORITHM SIMULATOR***
```

Implementation of FCFS Algorithm

```
FIRST COME FIRST SERVE SCHEDULING ALGORITHM
==========================================

Enter total number of process: 5

Enter burst time for each process:
P[1] : 10
P[2] : 15
P[3] : 5
P[4] : 3
P[5] : 7
Enter Throughput time to test for: 20

+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |     10     |       0      |        10       |
+-----+------------+--------------+-----------------+
|  2  |     15     |      10      |        25       |
+-----+------------+--------------+-----------------+
|  3  |      5     |      25      |        30       |
+-----+------------+--------------+-----------------+
|  4  |      3     |      30      |        33       |
+-----+------------+--------------+-----------------+
|  5  |      7     |      33      |        40       |
+-----+------------+--------------+-----------------+

Total Waiting Time      : 98
Average Waiting Time    : 19.60
Total Turnaround Time   : 138
Average Turnaround Time : 27.60
Throughput for 20 ms is  : 1

          * GANTT CHART *

 ------------------- ------------------------------ ---------- ------ --------------
|         P1        |              P2              |    P3    |  P4  |      P5      |
 ------------------- ------------------------------ ---------- ------ --------------
0                  10                             25         30     33             40
```

Implementation of SJF Algorithm

```
SHORTEST JOB FIRST SCHEDULING ALGORITHM
=======================================

Enter total process: 4

Enter burst time for each process:
P[1]: 10
P[2]: 15
P[3]: 20
P[4]: 3

Enter Throughput time to test for: 20

* Gantt Chart *
 ------ ------------------ ---------------------------- ------------------------------------
|  P4  |         P1         |              P2              |                  P3                  |
 ------ ------------------ ---------------------------- ------------------------------------
0      3                   13                           28                                    48

+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |     10     |       3      |        13       |
+-----+------------+--------------+-----------------+
|  2  |     15     |      13      |        28       |
+-----+------------+--------------+-----------------+
|  3  |     20     |      28      |        48       |
+-----+------------+--------------+-----------------+
|  4  |      3     |       0      |         3       |
+-----+------------+--------------+-----------------+

Total Waiting Time      : 44
Average Waiting Time    : 11.00
Total Turnaround Time   : 92
Average Turnaround Time : 23.00
Throughput for 20 ms is : 2
```

Implementation of PS Algorithm

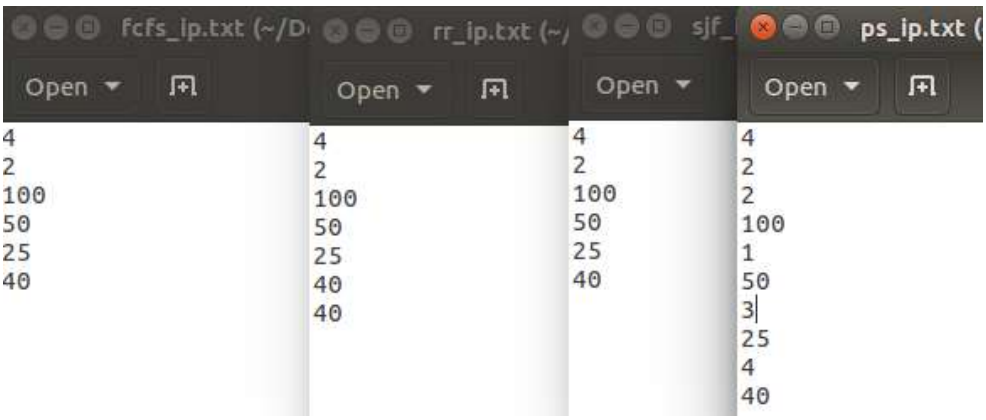
```
PRIORITY SCHEDULING ALGORITHM
=============================

Enter total number of Process: 3

Enter Burst Time and Priority for each process:

Burst Time of P[1]: 10
Priority of P[1]  : 2

Burst Time of P[2]: 20
Priority of P[2]  : 1

Burst Time of P[3]: 15
Priority of P[3]  : 1

Enter Throughput time to test for: 20


          * GANTT CHART *

 ---------------------------------------- ------------------------------ --------------------
|                  P2                    |              P3              |         P1         |
 ---------------------------------------- ------------------------------ --------------------
0                                        20                             35                   45

+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |     10     |      35      |        45       |
+-----+------------+--------------+-----------------+
|  2  |     20     |       0      |        20       |
+-----+------------+--------------+-----------------+
|  3  |     15     |      20      |        35       |
+-----+------------+--------------+-----------------+

Total Waiting Time      : 55
Average Waiting Time    : 18.33
Total Turnaround Time   : 100
Average Turnaround Time : 33.33
Throughput for 20 ms is : 1
```

Implementation of RR Algorithm

```
ROUND ROBIN SCHEDULING ALGORITHM
================================

Enter total number of process: 3

Enter Burst Time for each process:
Burst Time of P[1] : 15
Burst Time of P[2] : 20
Burst Time of P[3] : 10
Enter time Quantum: 5
Enter Throughput time to test for: 20


    ***GANTT CHART***

| P1 | -> | P2 | -> | P3 | -> | P1 | -> | P2 | -> | P3 | -> | P1 | -> | P2 | -> | P2 |


+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |     15     |      20      |        35       |
+-----+------------+--------------+-----------------+
|  2  |     20     |      25      |        45       |
+-----+------------+--------------+-----------------+
|  3  |     10     |      20      |        30       |
+-----+------------+--------------+-----------------+

Total Waiting Time              : 65
Average Waiting Time            : 21.67
Total Turnaround Time           : 110
Average Turnaround Time         : 36.67
Total number of Context Switches: 7
Throughput for 20 ms is          : 0
```

Comparison of FCFS, SJF, PS and RR Algorithm

Input: From file which contains Number of Processes, Burst time of each algorithm, Throughput time to test, Priorities of processes (in case of PS) and Quantum (in case of RR)



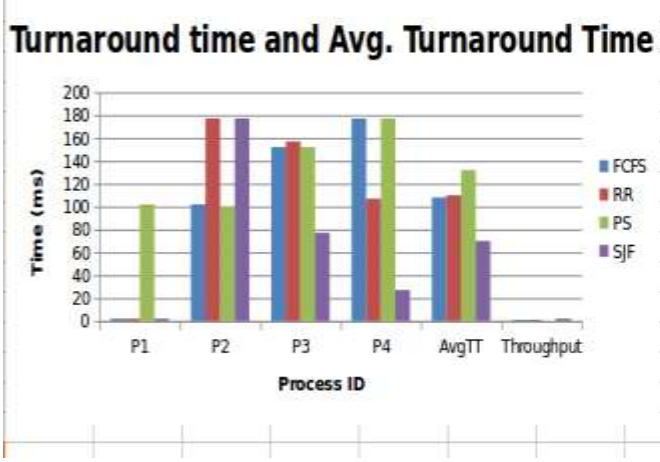
Output: Starvation Analysis

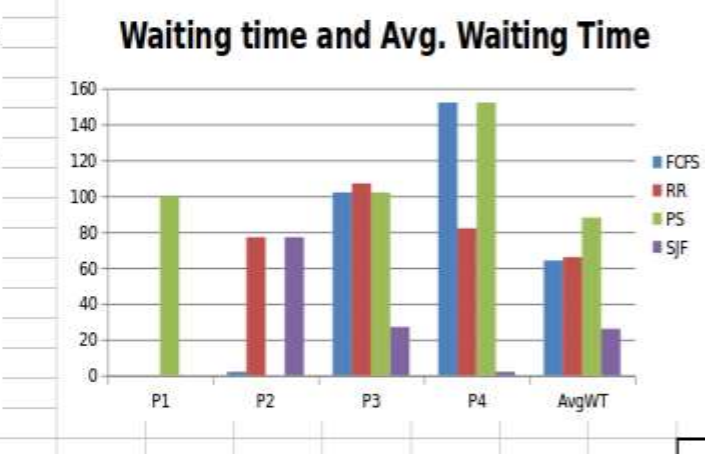
```
Starvation analysis of each Algorithm:

Algorithm: RR      Most Starved Process: P3      Wait Time (ms): 107
Algorithm: FCFS    Most Starved Process: P4      Wait Time (ms): 152
Algorithm: SJF     Most Starved Process: P2      Wait Time (ms): 77
Algorithm: PS      Most Starved Process: P4      Wait Time (ms): 152

Graphs to Compare all the Algorithms are Generated...

                    MENU
```

Output: Graphs of comparison of Wait time, Average Wait time, Turnaround time, Average Turnaround time and Throughput

Process	FCFS	RR	PS	SJF
P1	2	2	102	2
P2	102	177	100	177
P3	152	157	152	77
P4	177	107	177	27
AvgTT	108	110	132	70
Through»	1	1	0	2

Process	FCFS	RR	PS	SJF
P1	0	0	100	0
P2	2	77	0	77
P3	102	107	102	27
P4	152	82	152	2
AvgWT	64	66	88	26



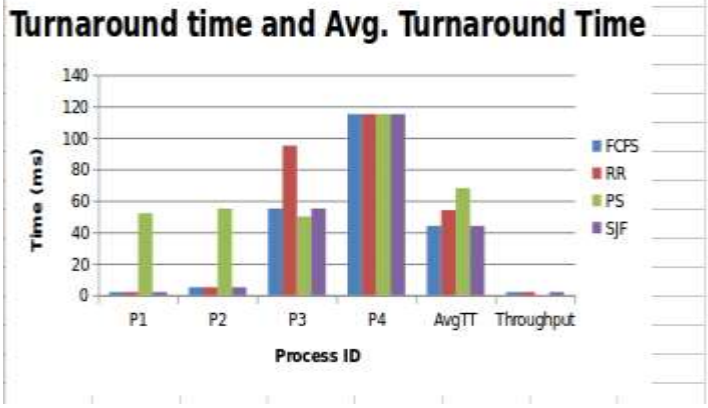


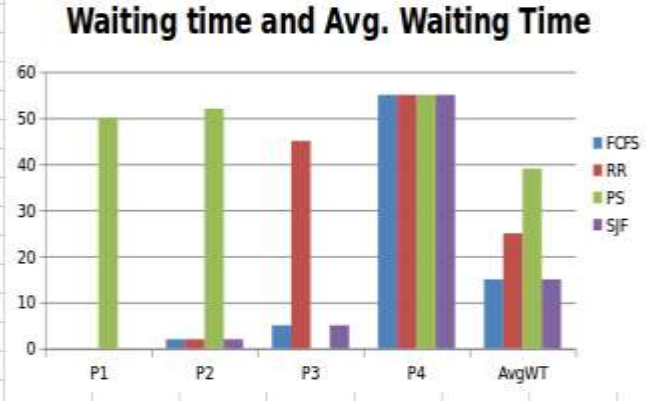
Input 2: Processes are mix of long and sort processes

Number of processes: 4
Burst time of each process: 2, 3, 50, 60
Priority: 2, 3, 1, 4
Quantum: 10
Throughput to test: 20

Process	FCFS	RR	PS	SJF
P1	2	2	52	2
P2	5	5	55	5
P3	55	95	50	55
P4	115	115	115	115
AvgTT	44	54	68	44
Through	2	2	0	2

Process	FCFS	RR	PS	SJF
P1	0	0	50	0
P2	2	2	52	2
P3	5	45	0	5
P4	55	55	55	55
AvgWT	15	25	39	15



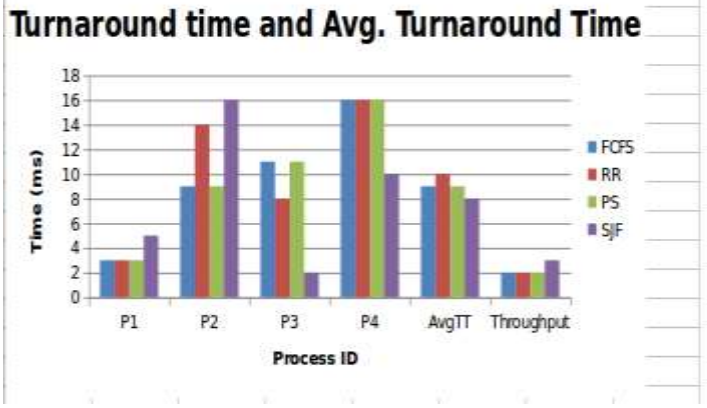


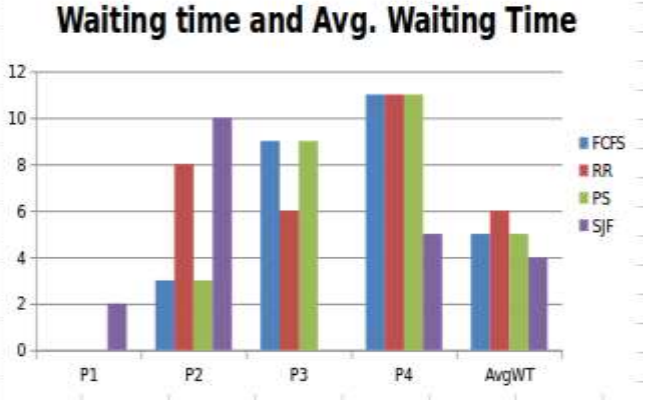
Input 3: All Processes are sort

Number of processes: 4
Burst time of each process: 3, 6, 2, 5
Priority: 1, 2, 3, 4
Quantum: 3
Throughput to test: 10

Process	FCFS	RR	PS	SJF
P1	3	3	3	5
P2	9	14	9	16
P3	11	8	11	2
P4	16	16	16	10
AvgTT	9	10	9	8
Through	2	2	2	3

Process	FCFS	RR	PS	SJF
P1	0	0	0	2
P2	3	8	3	10
P3	9	6	9	0
P4	11	11	11	5
AvgWT	5	6	5	4



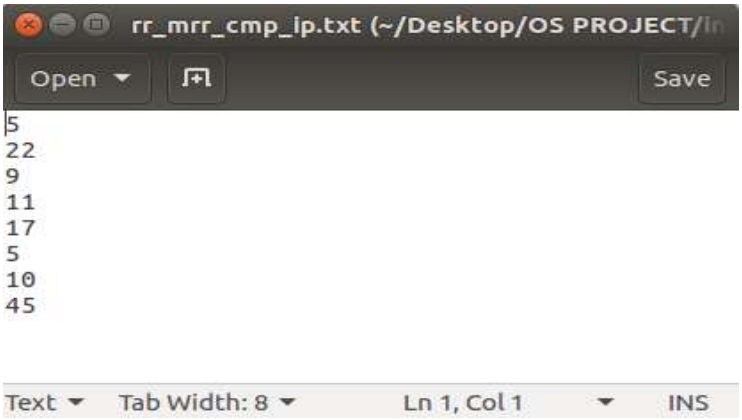


Modified RR

```
MODIFIED ROUND ROBIN SCHEDULING ALGORITHM
=========================================

Enter total number of process: 4

Enter Burst Time for each process:
Burst Time of P[1] : 30
Burst Time of P[2] : 15
Burst Time of P[3] : 5
Burst Time of P[4] : 20
Enter time Quantum: 5
Enter Throughput time to test for: 30


    ***GANTT CHART***

| P1 | -> | P2 | -> | P3 | -> | P4 | -> | P2 | -> | P4 | -> | P4 | -> | P1 | -> | P1 | -> | P1 |


+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |     30     |      40      |        70       |
+-----+------------+--------------+-----------------+
|  2  |     15     |      15      |        30       |
+-----+------------+--------------+-----------------+
|  3  |      5     |      10      |        15       |
+-----+------------+--------------+-----------------+
|  4  |     20     |      25      |        45       |
+-----+------------+--------------+-----------------+

Total Waiting Time              : 90
Average Waiting Time            : 22.50
Total Turnaround Time           : 160
Average Turnaround Time         : 40.00
Total number of Context Switches: 6
Throughput for 30 ms is          : 2
```

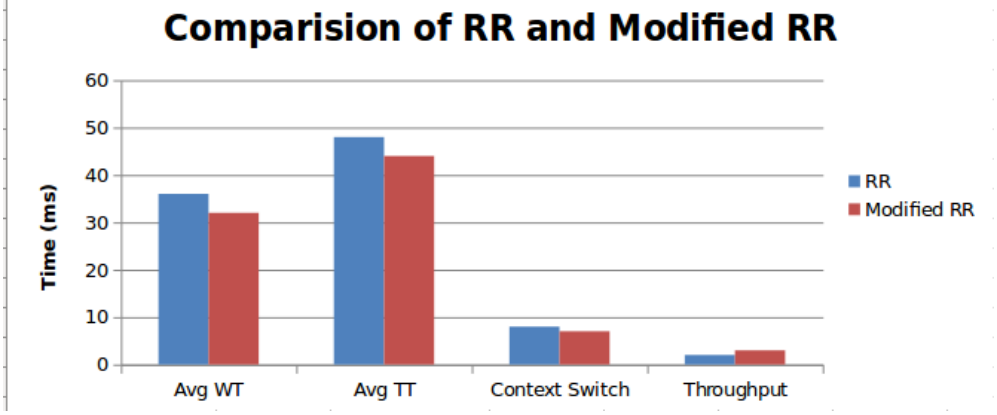
Comparison of RR and Modified RR

Input: From file which contains Number of processes, Burst time of each process, Quantum and Throughput time to test for



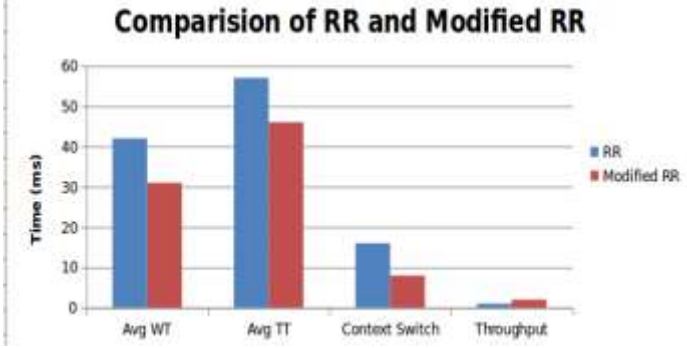
Output: Graph of comparison of Wait time, Average Wait time, Turnaround time, Average Turnaround time and Throughput

Process	RR	Modified RR
Avg WT	36	32
Avg TT	48	44
Context Switch	8	7
Throughput	2	3



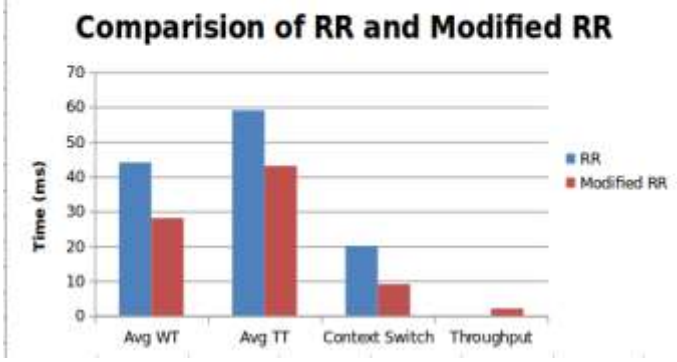
Input: 2
Number of processes: 5
Burst time of processes: 22, 18, 21, 10, 5
Time Quantum: 5
Throughput to test: 30 ms

Process	RR	Modified RR
Avg WT	42	31
Avg TT	57	46
Context Switch	16	8
Throughput	1	2



Input: 3
Number of processes: 5
Burst time of processes: 22, 18, 9, 20, 5
Time Quantum: 4
Throughput to test: 35 ms

Process	RR	Modified RR
Avg WT	44	28
Avg TT	59	43
Context	20	9
Through	0	2



Implementation of EDF Algorithm

Use of Signal SIGFPE to avoid floating point error of division by 0

```
EARLIEST DEADLINE FIRST ALGORITHM
=================================

Input Data

Pi |Ci, Di, pi
---------------
P1 | 3, 7, 0
P2 | 2, 4, 5
P3 | 2, 8, 10


Siganl Generated: Dovision by 0

This Real time system is not schedulable
```

Applying concept of deadlock detection and removal when more than 1 process arrives at the same time

```
Input Data

Pi |Ci, Di, pi
---------------
P1 | 3, 7, 20
P2 | 2, 4, 5
P3 | 2, 8, 10

CPU Utilization: 0.750000
This Real time system is schedulable because 0.750000 <= 1

LCM: 20
We can schedule the real system in 20 units.

Time    PID
-----------

Deadlock Avoided...
P1 allocated to CPU, based on selection of nearest deadline

Deadlock Avoided...
P2 allocated to CPU, based on selection of nearest deadline

(0,1):  P2
(1,2):  P2
(2,3):  P1
(3,4):  P1
(4,5):  P1
(5,6):  P3
(6,7):  P3
(7,8):  P2
(8,9):  P2
(9,10):  P0

...Deadlock Detected...
Following Processes accured at same time:
P2
P3

Deadlock Avoided...
P2 allocated to CPU, based on selection of nearest deadline
```

Implementation of RMS Algorithm

```
Time    PID
-----------

(0,1):  P2
(1,2):  P1
(2,3):  P1
(3,4):  P4
(4,5):  P4
(5,6):  P2
(6,7):  P3
(7,8):  P3
(8,9):  P3
(9,10):  P3

...Deadlock Detected...
Following Processes accured at same time:
P1
P2

Deadlock Avoided...
P2 allocated to CPU, based on selection of smallest period

(10,11):  P2
(11,12):  P1
(12,13):  P1
(13,14):  P3
(14,15):  P1

...Deadlock Detected...
Following Processes accured at same time:
P2
P4

Deadlock Avoided...
P2 allocated to CPU, based on selection of smallest period

(15,16):  P2
(16,17):  P4
(17,18):  P4
(18,19):  P1
(19,20):  P1
```

Handling of Interrupts like ctrl + C and ctrl + Z using signals SIGINT and SIGTSTP respectively

```
RATE MONOTONIC SCHEDULING ALGORITHM
===================================

Input Data

Pi |Ci, pi
-----------
P1 | 2, 10
P2 | 1, 5
P3 | 5, 30
P4 | 2, 15

CPU Utilization: 0.700000
As 0.700000 < 0.756828 , The System is surely Schedulable.

LCM: 30
We can schedule the real system in 30 units.

Time    PID
-----------

(0,1):  P2
(1,2):  P1
(2,3):  P1
^Z(3,4):  P4

Interrupt Signal Caught: 20
Interrupt Disabled

(4,5):  P4
(5,6):  P2
^C(6,7):  P3

Interrupt Signal Caught: 2
Interrupt Disabled

(7,8):  P3
(8,9):  P3
(9,10):  P3
```

Conclusion

- In this project, we simulated and learnt about different type of scheduling algorithm and implementation of them which are used in batch systems, interactive systems and real time systems.
- Also, we visualized and compared the performance of CPU scheduling algorithms with different parameters such as Average Waiting time, Throughput and average Turnaround time, number of context switches etc.
- We also proposed one algorithm named Modified Round Robin which is amalgamation of simple Round robin, priority scheduling and uses concept of dynamic quantum precision, which gives better performance than traditional Round robin.

Future work

- Algorithms we implemented was based on uniprocessor, but this could be done in multiprocessor system.
- Also complexities could be modified and multiprogramming can be considered. Even more variables and criteria could be used to provide a broader range for comparison.
- Extension of this project could be Re-Compilation of Linux Kernel using both real time and batch scheduling algorithms.

References

Research Paper:

- https://www.researchgate.net/publication/50194216_An_Optimized_Round_Robin_Scheduling_Algorithm_for_CPU_Scheduling
- https://www.ripublication.com/acst17/acstv10n6_01.pdf
- http://ipasj.org/IIJCS/Volume2Issue11/IIJCS-2014-11-17-28.pdf
- https://www.coursera.org/learn/real-time-systems/.../earliest-deadline-first-example

Websites:

- http://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/
- https://en.wikipedia.org/wiki/Scheduling_(computing)
- http://www.studytonight.com/operating-system/cpu-scheduling
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html

Reference Book:

- Operating Systems-Internals and Design principles by William Stallings.