OS Lab 11

Omar Harb 900201063
Sara Mohamed 900203032
Mohamed Shaalan 900201539

Roles

Omar Harb: **Multi-Level Feedback Queue,**

Sara Mohamed: **Round Robin,**

Mohamed Shaalan: **FCFS,**

“Process” Struct Attributes

- PID
  - to keep track of the processes
- Burst Time
  - needed for all scheduling algorithms, to properly move the time t
- Remaining Time
  - need to check when process gets done executing
- Arrival Time
  - needed to keep track of what processes can be scheduled, according to the current time t
- Response
- Waiting
- Turnaround

Dependencies

```
Process *get_n_processes(int n)
{
    Process *processes = new Process[n];
    for (i to n-1)
    {
        processes[i].pid = i + 1;
        processes[i].burst_time = rand(1-100);
        processes[i].arrival_time = rand(0-100);
        processes[i].remaining_time = processes[i].burst_time;
    }
    sort(processes);    //sort processes by arrival time
    return processes;
}
```

Pseudocode: Round Robin

```
void getRRwaitingtime(Process processes[], int n, int waitingtime[], int turningtime[], int
responsetime[], int quantum)
{
    int remaining_burstTimes[n];
    for (i = 0 to n-1)
        remaining_burstTimes[i] = processes[i].burst_time;

    int t = 0;  int finishedProcesses = 0;   //initializing time t and num of finished processes
    for (;;){
        int processesStarted = 0;   //number of processes started in this for loop over the processes
        for (i = 0 to n-1){
            if (arrivaltime(i)<= t & remaining_burstTimes(i) > 0)   //if process arrived and not finished
                    if (first time process is running)
                        responsetime[i] = t;       //evaluation of process response time
                    processesStarted++;   //increment num of  processes started
                    if (remaining_burstTimes[i] > quantum)
                        t += quantum;
                        remaining_burstTimes[i] -= quantum;
```

Pseudocode: Round Robin


```
                else                                          // if remaining burst time less than/equal to quantum
                    t = t + remaining_burstTimes[i];
                    waitingtime[i] = t - processes[i].arrival_time - processes[i].burst_time;
                    remaining_burstTimes[i] = 0;
                    finishedProcesses++;
        }   //end of inner for loop


        if (finishedProcesses == n)   //if all processes done, break from the infinite for loop
            break;

        if (processesStarted == 0)   //if none of the processes started in the past for loop, increment t
            t++;
        }

    for (int i = 0; i < n; i++)   //turnaround time is simply wait time + burst time
      turnarountime[i] = waitingtime[i] + processes[i].burst_time;

}
```

Pseudocode: FCFS


```
void run_fcfs(Process processes[], int n, int waitingtime[], int turnaroundtime[], int
responsetime[] ){

    currentTime = 0;
    waitingtime[0] = 0;  // first process has no waiting time

    if (processes[0].arrival == 0){  //if first process arrives at time 0, add its burst time to currentTime
        currentTime = currentTime + processes[0].burst_time;}
    else {   // if first process doesn't immediately arrive, jump to its arrival time + burst time
        currentTime = currentTime + processes[0].arrival_time + processes[0].burst_time;}

    for (i to n-1)
    {
        if (processes[i].arrival_time < t){   // if process arrives before currentTime, it has to wait
            waitingtime[i] = t - processes[i].arrival_time;   //it waited from when it arrived till currentTime
            t += processes[i].burst_time; }

        else {// if process arrives after or on currentTime, it has no waiting time
            waitingtime[i] = 0; //process did not wait
            t = processes[i].arrival_time + processes[i].burst_time;}
    }
```

Pseudocode: FCFS


```
    for (int i = 0; i < n; i++){   //turnaround time is simply wait time + burst time
        turnaroundtime[i] = waitingtime[i] + processes[i].burst_time; }
    for (int i = 0; i < n; i++){   //response time in FCFS is equal to waiting time
        responsetime[i] = waitingtime[i]; }
}
```

Pseudocode: Multi-Level Feedback Queue


```
void run mlq(vector<Proc_Queue> &queues, processes[], mlq_waiting_time[], mlq_turnaround_time[],
mlq_response_time[], n)
{
    m = queues.size();
    t = 0;  // current time, starting at 0

    // all processes initially assigned to first queue.
    for (i = 0 to n-1)
        queues[0].processes.push(processes[i]);

    for (i = 0 to m-1)
    {
        if (i == m - 1 and m > 1)
            sort_queue(queues[i]);

        min arrival = INF;
        num moved = 0;
        while (!queues[i].processes.empty())
        {
            &process = queues[i].processes.front();
            if (process.arrival_time <= t)
            {
                // so it doesn't get overwritten when assigned to lower queue
                if (i == 0)
                    process.response = t - process.arrival_time;

                int quantum = min(process.remaining_time, queues[i].time_quantum);
                t += quantum;
                process.remaining_time -= quantum;

                queues[i].processes.pop();
```

```
struct Proc_Queue
{
    int time_quantum;
    queue<Process> processes;
};
```

Pseudocode: Multi-Level Feedback Queue


```
                if (process done executing)
                {
                   process.turnaround = t - process.arrival time;
                   process.waiting = process.turnaround - process.burst_time;
                }
                else
                {
                   if (i < m - 1)
                     push process into next queue
                }
          }
          else
          {
                move process to end of queue
                min arrival = min(min_arrival, process.arrival_time);
                num moved++;
                if (num_moved == queues[i].processes.size()) // process is now in original order
                {
                    t = min arrival;
                    num moved = 0;
                    min_arrival = INF;
                 }
         }
      }
   }
   get_results(processes, mlq_waiting_time, mlq_turnaround_time, mlq_response_time, n);
}
```

Sample Test Case

Process ID	Arrival Time	Burst Time
1	2	5
2	8	2
3	8	2
4	9	9
5	10	1

Testing Round-Robin

Time-quantum = 4

Gantt Chart:

X	P1	x	P2	P3	P4	P5	P1	P4	P4

0 2 6 8 10 12 16 17 18 22 25

```
PS C:\Users\Mohamed Shaalan\Desktop\Uni\Course Material\OS_Lab\Lab11> ./main
Processes  Burst time  Turn around time  Response time  Waiting time
  1            5          5                 0              0
 2             2          2                 0              0
 3             2          4                 2              2
 4             9          13                3              4
 5             1          7                 6              6
```

Testing FCFS

Gantt Chart:

X	P1	x	P2	P3	P4	P5

0 2 7 8 10 12 21 22

```
PS C:\Users\Mohamed Shaalan\Desktop\Uni\Course Material\OS_Lab\Lab11> ./main
Processes  Burst time  Turn around time  Response time  Waiting time
  1            5          5                0              0
 2             2          2                0              0
 3             2          4                2              2
 4             9          12               3              3
 5             1          12               11             11
```

Testing MLFQ

Time-quantum = {3, 5, INF}

Queue 1

X	P1	x	P2	P3	P4	P5

0 2 5 8 10 12 15 16

Queue 2

P1	P4

16 18 21

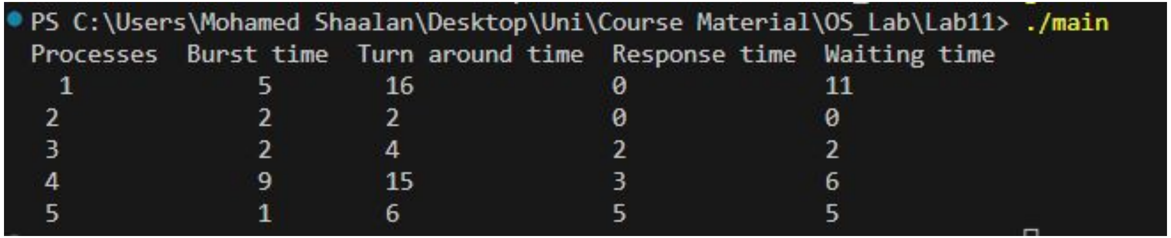
Queue 3

P4

21 24

Results (Computations)

PID	Turnaround	Response	Waiting
1	18 - 2 = 16	2 - 2 = 0	16 - 5 = 11
2	10 - 8 = 2	8 - 8 = 0	2 - 2 = 0
3	12 - 8 = 4	10 - 8 = 2	4 - 2 = 2
4	24 - 9 = 15	12 - 9 = 3	15 - 9 = 6
5	16 - 10 = 6	15 - 10 = 5	6 - 1 = 5



```
PS C:\Users\Mohamed Shaalan\Desktop\Uni\Course Material\OS_Lab\Lab11> ./main
Processes  Burst time  Turn around time  Response time  Waiting time
  1            5          16                 0              11
 2             2          2                  0              0
 3             2          4                  2              2
 4             9          15                 3              6
 5             1          6                  5              5
```

Run Details

Processes: **100 - 2000**

Arrival time range: **0 - 100**

Burst time range: **1 - 100**

RR time quantum: **20**

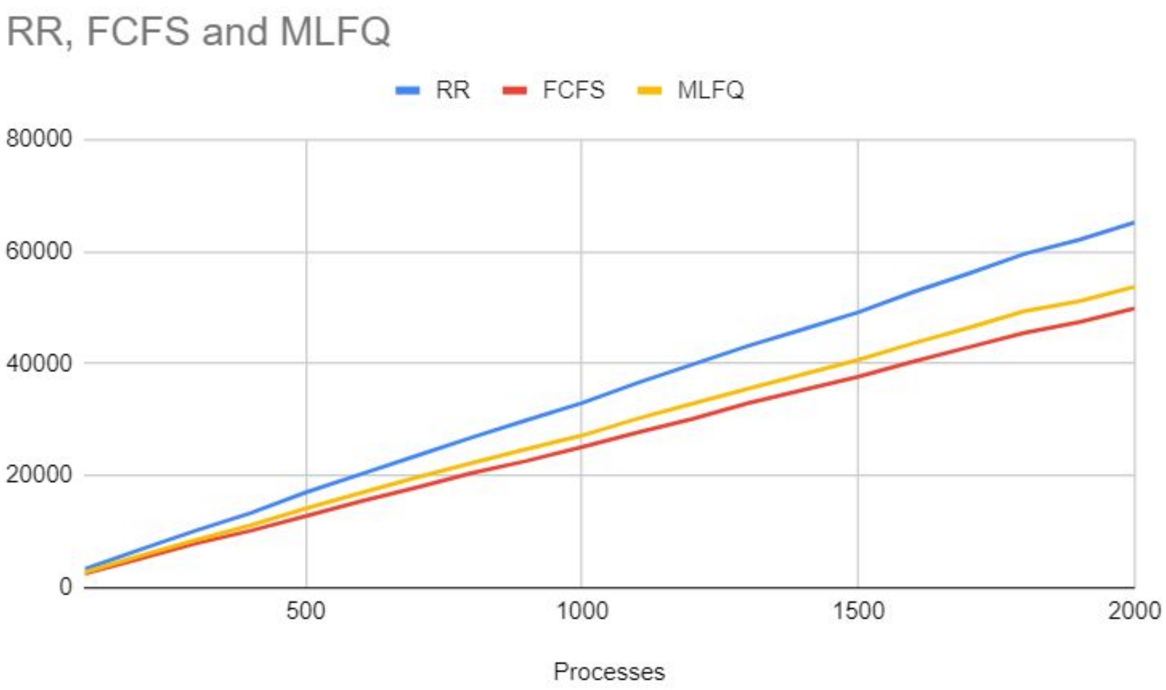
MLFQ Queue 1 time quantum: **3**

MLFQ Queue 2 time quantum: **5**

MLFQ Queue 3: **FCFS**

Waiting Time Graph

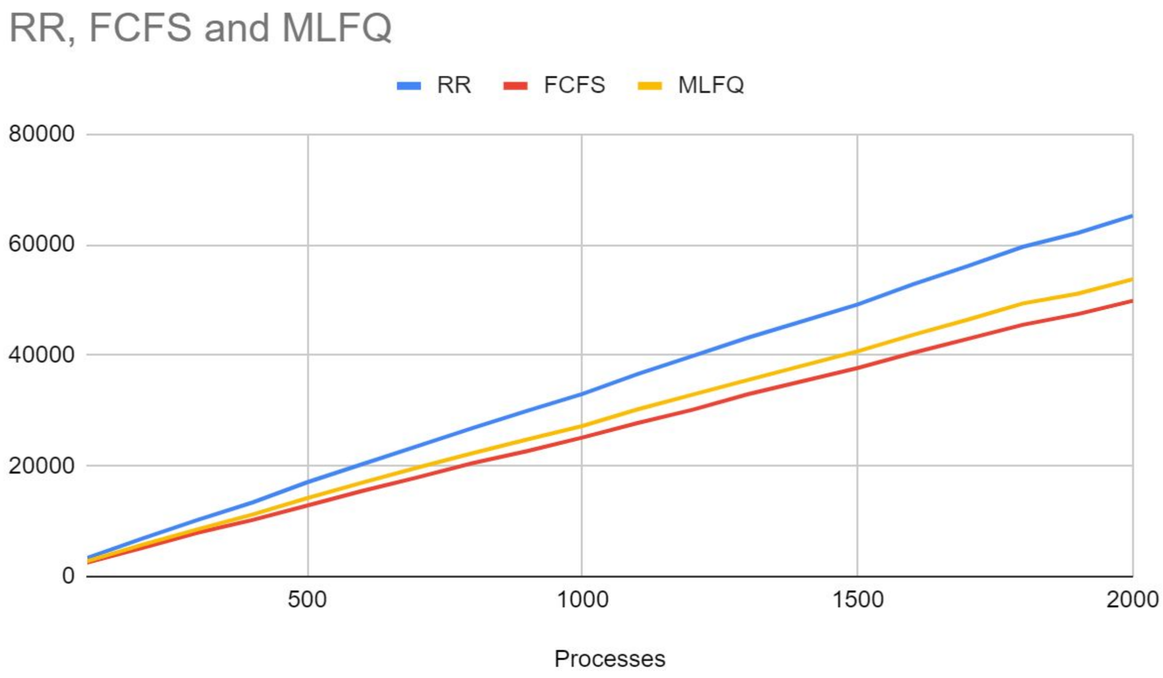
- understandably, rr takes most waiting time, followed by MLFQ and then FCFS



Turnaround Time Graph

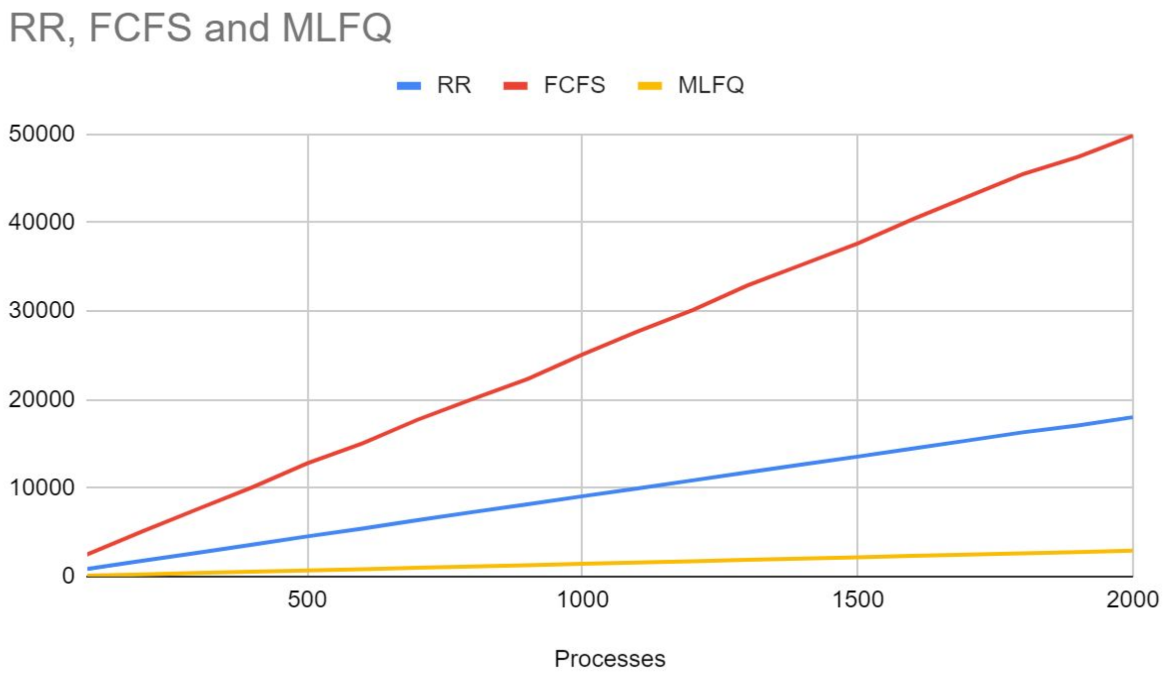
- understandably, rr takes most turnaround time, followed by MLFQ and then FCFS

- turnaround very similar to waiting, since it is evaluated by adding waiting time to burst time



Response Time Graph

- understandably, MLFQ takes the least response time, since the time quantum assigned to its first rr queue is smaller than that of the rr algorithm; followed by rr and then fcfs



Results & Analysis

- round robin always takes least response time, and differs depending on the time quantum assigned to it
- FCFS always leads to the shortest waiting and turnaround time, this is due to a relatively short burst time (less convoy effect)
- FCFS is always assigned last in the MLFQ, because a process is guaranteed to complete execution in that algorithm; so we finish off the MLFQ with FCFS to conclude the execution of all remaining processes