College of Computer and Cyber Sciences
Department of Computer Science

CS221- Fundamental of Operating System

Multilevel CPU Scheduling Simulation

Team Members:

Abeer Jelani – 4010368

Leen Khalil – 4010409

Shatha Faraj – 4010378

Salwa Shamma – 4010405

Samah Shamma – 4010403

L03

Instructor:

Dr. Fazilh Haron

Ms. Amal Albalawi

1 December 2022

Abstract

This report discusses the CPU scheduling which is the basic of multiprogram operating systems. By switching the CPU among processes, the operating system can make the computer more productive. Several CPU scheduling strategies are presented in this project along with an introduction to fundamental CPU scheduling concepts. The algorithms that are implemented in this project are first come first serve and Round Robin. Moreover, it will demonstrate the algorithm's pseudocode, test cases and results. In addition, it comprehends a comparison of the algorithms and a conclusion.

Keywords: *FCFC, Round Robin, Multiprogram, Algorithm.*

TABLE OF CONTENTS

I. INTRODUCTION

As is well known, various CPU scheduling algorithms exist to address the issue of which processes in the ready queue should be selected by the CPU scheduler to be executed in the CPU. The scope of the report focuses on two of them which are the First Come First Serve (FCFS), and Round Robin (RR) algorithms that will be used to implement multilevel queue scheduling with certain conditions and constraints. The next section of the report will briefly highlight the definitions of these different scheduling algorithms.

There are some concepts that should be underlined before going into details. The operating system can increase computer productivity by switching the CPU between processes which is the foundation of multiprogramming operating systems. The CPU scheduler performs the switch, choosing one of the processes in memory that are prepared to run and allocating the CPU to that process. There are various scheduling algorithms as it is mentioned before: a ready queue can be implemented as a FIFO queue, a priority queue, and so on. [1]

We have to take in our consideration also, there are two types of algorithms which are pre-emptive and non-preemptive. Once the CPU has been assigned to a process under non-preemptive scheduling, the process keeps the CPU until it releases it by terminating or by moving to the waiting state. All the algorithms that are going to disused under the non-primitive group. [1]

This report discusses multilevel CPU scheduling simulation in technical way. In addition, the pseudocode of solutions, test cases, and the analysis of result are provided.

II. CPU SCHEDULING ALGORITHIMS

A. First Come First Served (FCFS):

First-come first-served scheduling is the most basic scheduling algorithm ever created. The CPU is allotted to processes using this technique in the order in which they make their requests. In essence, there is just one queue of prepared processes. first process instantly begins to run for whatever long it wants to and it is not interrupted.[1]

In this scheduling, the CPU will begin executing the process that comes in front of it first. It is a non-pre-emptive form of scheduling algorithm, which means that in this scheduling algorithm, the priority of processes is irrelevant; rather, processes will be run in the order in which they were presented to the CPU.[2] FIFO scheduling is another name for FCFS.

B. Round-Robin (RR):

Round-Robin Planning is one of the most well-known, fairest, and easiest algorithms. Each process is given a time window known as its quantum within which it is permitted to operate. The CPU is pre-empted and given to another process if the process is still active at the conclusion of the quantum. Naturally, the CPU switches off when the process blocks if it has been blocked or finished before the quantum has passed. Round robin is simple to use, the procedure gets moved to the bottom of the list after it exhausts it quantum.[1]

The pre-emptive scheduling algorithm is round robin scheduling. Each process in this scheduling is given a specific time slice, or time quantum, to which it is assigned, and is then executed in cycles. The queue contains all the processes that wish to run. The process is given a CPU for that time quantum. Now, if the process completed its execution within that amount of time, it will be terminated; but, if it did not, it will be put to the ready queue once more and the prior process will have to wait for its turn to finish execution.[2]

C. Multilevel:

Designers eventually noticed that it was more effective to occasionally provide CPU-bound programs a big quantum but on the other hand, giving all processes a big quantum would result in slow response times. They came up with the idea of creating priority classes. For example, the most advanced processes were conducted for one quantum. The next-highest class processes were run for two quanta. The following one ran processes for four quanta, etc. A process was degraded one class after it used up all of the quanta allotted to it.[1]

There are seven distinct lines in the ready queue. Depending on the priority of the process, factors like memory size, etc., the process is separated into various queues. The scheduling algorithms employed by the various queues can be similar or dissimilar. There are numerous processes that we cannot place in a singular queue, which is resolved by this scheduling (multilevel) since we can now put them in different queues, which comes with a low scheduling overhead.[2]

III. PSEUDOCODE

1. **Start**
2. **Ask the user how he/she will enter the data**

   2.1 Enter the data interactively.

   2.2.1 Take 6 inputs from the user (Number of processes, Arrival time, Burst time, Quantum time for Round Robin, Quantum time for Round Robin 1 of Multilevel Queue, Quantum time for Round Robin 2 of Multilevel Queue).
3. **Enter the data using input file.**
4. **FCFS (First Come First Served) algorithm function:**
   1. Find waiting time:

      1.1 For the 1st process waiting time = 0.

      1.2 For all the next processes, waiting time = start time of executing – arrival time.
   2. Find turnaround time:

      2.1 For the 1st process turnaround time = burst time.

      2.2 For all the next processes, turnaround time = end time of executing – arrival time.
   3. Output: average waiting time = total waiting time/ no. processes.
   4. Output: average turnaround time = total turnaround time/ no. processes.
5. **RR (Round Robin) algorithm function:**
   1. For each process:

      Burst remaining time = burst time.
   2. While (the number of completed processes is not equal to no. processes):

      if (burst remaining time <= quantum time):

      current time = current time + burst remaining time

      burst remaining time = 0

      if (burst remaining time > quantum time):

      current time = current time + quantum time

      burst remaining time = burst remaining time – quantum time

      if (burst remaining time = 0):

      Turnaround time = current time – arrival time

Waiting time = turnaround time – burst time

Increment the number of completed processes

3. Output: average waiting time = total waiting time/ no. processes.

   Output: average turnaround time = total turnaround time/ no. processes

6. Multilevel Scheduling (1 RR, 1 FCFS) algorithm function:

1. For (each process).

   Burst remaining time= burst time.

2. While (the number of completed processes < no. process):

   1. Enter Round Robin queue

      If (burst remaining time < = quantum time):

      Waiting time = current time - arrival time - burst time

      Turnaround time = current time – arrival time

      else (burst remaining time > quantum time):

      Burst remaining time= burst remaining time - quantum time

      Process moves to FCFS queue

   2. If (arrival time to next time > current time)

      Move the CPU to FCFS queue.

      Turnaround time = current time – arrival time

      Waiting time = turnaround time - burst time

3. If (FCFS queue not empty):

   Move the CPU to FCFS queue.

4. Output: average turnaround time = total turnaround time/No. of processes.

   Output: average waiting time = total waiting time/No. of processes.

7. Multilevel Scheduling (2 RR, 1 FCFS) algorithm function:

1. For (each process):

   1. If (burst remaining time < = quantum time):

      Waiting time = current time - arrival time - burst time

      Turnaround time = current time – arrival time.

      Else (burst remaining time > quantum time):

      Burst time= burst time – small quantum time

      Process moves to second RR queue

   2. If (arrival time to next time > current time):

      Move the CPU to second RR queue.

If (burst remaining time < = quantum time):

Waiting time = current time - arrival time - burst time

Turnaround time = current time – arrival time.

Else (burst remaining time > quantum time):

Burst time= burst time – large quantum time

Process moves to FCFS queue.

3. If (arrival time to next time > current time):

Move the CPU to FCFS queue.

Execute processes in the order they arrive until finished.

Turnaround time = current time – arrival time

Waiting time = turnaround time - burst time

4. Output: average turnaround time = total turnaround time/No. of processes.

Output: average waiting time = total waiting time/No. of processes.

8. Stop

IV. CODE

A. FCFS

```
/* ----------------------------------------------------------------------------------------------*/
/* ----------------------------------------FCFS Methods------------------------------------------*/
/* ----------------------------------------------------------------------------------------------*/

/* ----------------------calc_StartEndTime----------------------------*/
void calc_StartEndTime(proc p[]){
        for (int i=0 ; i< num_process ; i++){
            if (i==0){
            p[0].start_t=0;
            p[0].end_t=p[0].burst;
            }else{
            p[i].start_t=p[i-1].end_t;
            p[i].end_t=p[i-1].end_t+p[i].burst;
            }
        }
}
/*-----------------------calc_Time-----------------------------------*/
void calc_Time(proc p[]){
printf("\nProcess No \t\t Burst Time \t\t Turnaround Time \t\t Waiting Time ");
/*-----------------------------calc_AvgTime_FCFS-----------------------*/
void calc_AvgTime_FCFS(proc p[]){
    int tot_wait_t=0 ,
    tot_turnaround_t=0;

for (int i=0;i<num_process;i++){
        tot_wait_t+=p[i].wait_t;
        tot_turnaround_t += p[i].turnaround_t;
}
printf("\nAverage Turn Around Time: %6.1f",(float)tot_turnaround_t/num_process);
printf("\nAverage Waiting Time    : %6.1f \n",(float)tot_wait_t/num_process);

fprintf(fh,"\nAverage Turn Around Time: %6.1f",(float)tot_turnaround_t/num_process);
fprintf(fh,"\nAverage Waiting Time    : %6.1f \n",(float)tot_wait_t/num_process);
}

/*--------------------------------FCFS Algo----------------------------*/
void FCFS_ALgo(proc p[]){

        calc_StartEndTime(p);
        calc_Time(p);
        calc_AvgTime_FCFS(p);
}
```

B. RR

```
/*------------------------------RR-----------------------------------*/
void RR(proc p[]){
printf("\nProcess No \t\t Burst Time \t\t Turnaround Time \t\t Waiting Time ");
int y; //number of process
// temp is the remaining time
// sum is current time
for(sum=0, i = 0; y!=0; )
{
    if(temp[i] <= quant && temp[i] > 0) // if the remaining time between 0 and quantum time
    {
    sum = sum + temp[i];
    temp[i] = 0; // assign remaining time to 0
    counter=1;
    }
    else if(temp[i] > 0) // if the remaining time is greater than quantum time
    {
        temp[i] = temp[i] - quant;
        sum = sum + quant;
    }

    if(temp[i]==0 && counter==1) // enter block if the first condtion is met
    {
        y--; //decrement the process no.
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, p[i].burst, sum-p[i].arriv, sum-p[i].arriv-p[i].burst);
        wt = wt+sum-p[i].arriv-p[i].burst;
        tat = tat+sum-p[i].arriv;
        counter =0;
    }

    if(i==num_process-1 ) // the last process
    {
        i=0;
    }
    else if(p[i+1].arriv<=sum) // move to next process
    {
        i++;
    }
    else // if arrival time to the next process is greater than current time
    {
        i=0;
    }
   }
}
```

```
/*------------------------------calc_AvgTime_RR-----------------------------------*/
void calc_AvgTime_RR(proc p[]){
// represents the average waiting time and Turn Around time
avg_wt = wt * (1.0/num_process);
avg_tat = tat * (1.0/num_process);
printf("\nAverage Turn Around Time: %6.1f", avg_tat);
printf("\nAverage Waiting Time: %6.1f \n", avg_wt);

fprintf(fh,"\nAverage Turn Around Time: %6.1f", avg_tat);
fprintf(fh,"\nAverage Waiting Time: %6.1f \n", avg_wt);
}

/*--------------------------------RR Algo------------------------------------------*/
void RR_ALgo(proc p[]){
        RR(p);
        calc_AvgTime_RR(p);
}
```

C. 3 MLQ

```
/* -------------------------------------------------------------------------------------------*/
/* ----------------------------------------3 multilevel scheduling -------------------------------------*/
/* -------------------------------------------------------------------------------------------*/

/*----------------------calc_AvgTime_3 level MLQ------------------------------------*/
void calc_AvgTime_level_3_MLQ(){
int tot_wait_t=0 ,
tot_turnaround_t=0;

for (int i=0;i<num_process;i++){
tot_wait_t+=Q1[i].wait_t;
tot_turnaround_t += Q1[i].turnaround_t;

tot_wait_t+=Q2[i].wait_t;
tot_turnaround_t += Q2[i].turnaround_t;

tot_wait_t+=Q3[i].wait_t;
tot_turnaround_t += Q3[i].turnaround_t;
}

printf("\nAverage Turn Around Time: %6.1f",(float)tot_turnaround_t/num_process);
printf("\nAverage Waiting Time    : %6.1f \n",(float)tot_wait_t/num_process);

fprintf(fh,"\nAverage Turn Around Time: %6.1f", (float)tot_turnaround_t/num_process);
fprintf(fh,"\nAverage Waiting Time: %6.1f \n", (float)tot_wait_t/num_process);

}
```

```
/*---------------------------------3 level MLQ-------------------------------------*/
void level_3_MLQ(){
time=Q1[0].arriv;
printf("Process in first queue following RR with time quantum=%d \n",quant1);
printf("\nProcess NO\t\tBurst\t\tWaiting\t\tTurnaround\tComplete");
for(int i=0;i<num_process;i++){
  if(Q1[i].remaining_t<=quant1){
        if(time < Q1[i].arriv){
                time = Q1[i].arriv;
        }
       time+=Q1[i].burst;//from arrival time of first process to completion of this process/
       Q1[i].wait_t=time-Q1[i].arriv-Q1[i].burst;//amount of time process has been waiting in the first queue/
       Q1[i].turnaround_t=time-Q1[i].arriv;//amount of time to execute the process/
      printf("\nProcess NO[%d]\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,Q1[i].burst,Q1[i].wait_t,Q1[i].turnaround_t,time);
        }

  else{//process moves to queue 2 with qt=25/
      if(time < Q1[i].arriv){
              time=Q1[i].arriv;
      }
      time+=quant1;
      Q1[i].remaining_t-=quant1;
      Q2[k].remaining_t=Q1[i].remaining_t;
      Q2[k].burst=Q1[i].burst;
      Q2[k].arriv = Q1[i].arriv;
      k=k+1;
  }
}

printf("\nProcess in second queue following RR with time quantum=%d \n",quant2);
printf("\nProcess NO\t\tBurst\t\tWaiting\t\tTurnaround\tComplete \n");

for(int i=0;i<k;i++){
      if(Q2[i].remaining_t<=quant2){
      time+=Q2[i].remaining_t;//from arrival time of first process +burst of this process/
      Q2[i].wait_t=time-Q2[i].arriv-quant1-Q2[i].remaining_t;//amount of time process has been waiting in the ready queue/
      Q2[i].turnaround_t=time-Q2[i].arriv;//amount of time to execute the process/
     printf("\nProcess NO[%d]\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,Q2[i].burst,Q2[i].wait_t,Q2[i].turnaround_t,time);
    }

    else{//process moves to queue 3 with FCFS/
     Q3[r].arriv=Q2[i].arriv;
     time+=quant2;
     Q2[i].remaining_t-=quant2;
     Q3[r].remaining_t=Q2[i].remaining_t;
     Q3[r].burst=Q2[i].burst;
     r=r+1;
    }
}

printf("\nProcess in third queue following FCFS\n ");
printf("\nProcess NO\t\tBurst\t\tWaiting\t\tTurnaround\tComplete\n");
for(int i=0;i<r;i++)
{
    if(i==0)
            Q3[i].complete_t=Q3[i].remaining_t+time;
        else
            Q3[i].complete_t=Q3[i-1].complete_t+Q3[i].remaining_t;

}
for(int i=0;i<r;i++){
        Q3[i].turnaround_t=Q3[i].complete_t - Q3[i].arriv;
        Q3[i].wait_t=Q3[i].turnaround_t - Q3[i].burst;  printf("\n Process NO[%d]\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,Q3[i].burst,Q3[i].wait_t,Q3[i].turnaround_t,Q3[i].complete_t);

    }
        calc_AvgTime_level_3_MLQ();
}
```

D. 2 MLQ

```
/* ------------------------------------------------------------------------------------------*/
/* ------------------------------------2 multilevel scheduling -----------------------------------*/
/* ------------------------------------------------------------------------------------------*/

/*----------------------RR_MLV2------------------------------------*/
void RR_MLV2(process Q11[],process Q22[]){
  if(Q11[r1].remaining_t<=quant1){
       time1+=Q11[r1].burst;//from arrival time of first process to completion of this process/
       Q11[r1].pointer=time1; // <---- this added
       Q11[r1].wait_t=time1-Q11[r1].arriv-Q11[r1].burst;//amount of time process has been waiting in the first queue/
       Q11[r1].turnaround_t=time1-Q11[r1].arriv;//amount of time to execute the process /

       printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n",r1,Q11[r1].burst,Q11[r1].wait_t,Q11[r1].turnaround_t,time1);
  } else {//process moves to queue 2 with qt=2
      time1+=quant1;
      Q11[r1].pointer=time1; // <---- this added
      Q11[r1].remaining_t-=quant1;
      Q22[k1].remaining_t=Q11[r1].remaining_t;
      Q22[k1].burst=Q11[r1].burst;
      Q22[k1].arriv=Q11[r1].arriv;
      k1=k1+1; // number of process in Q2
      }
    r1++; // to be ready to next iteration
}
```

```c
/*----------------------FCFS_MLV2-----------------------------------*/
void FCFS_MLV2( process Q11[],process Q22[]){
int j=0;
        do{
        printf("\nProcess in second queue FCFS_MLV2 ");
        printf("\nProcess\t\tBT\t\tWT\t\tTAT\t\tCT");

        if(j_1==0){
                    Q22[j_1].complete_t=Q22[j_1].remaining_t+time1;
                    time1=Q22[j_1].complete_t;//??

         }else{
                    if (flag1==1){
                    Q22[j_1].complete_t=Q11[r1-1].pointer+Q22[j_1].remaining_t;
                    flag1= 0;
                    }
                    else{
                    Q22[j_1].complete_t=Q22[j_1-1].complete_t+Q22[j_1].remaining_t;
                    }
                }

         Q22[j_1].turnaround_t=Q22[j_1].complete_t - Q22[j_1].arriv;
         Q22[j_1].wait_t=Q22[j_1].turnaround_t - Q22[j_1].burst;
         printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n",j,Q22[j_1].burst,Q22[j_1].wait_t,Q22[j_1].turnaround_t,Q22[j_1].complete_t);
         j_1++;

        } while((Q22[j_1].pointer < Q11[r1].arriv) && (j_1<k1)) ;
}
```

```c
void FCFS2( process Q11[],process Q22[]){

do{
printf("\nProcess in second queue following FCFS_MLV2 ");
printf("\nProcess\t\tBT\t\tWT\t\tTAT\t\tCT");
    if(j_1==0){
            Q22[j_1].complete_t=Q22[j_1].remaining_t+time1;
        }else{
            if (flag1==1){
            printf("\n- here --%d>\n", Q11[r1-1].pointer);
            Q22[j_1].complete_t=Q11[r1-1].pointer+Q22[j_1].remaining_t;
            flag1= 0;
            }
            else{
            Q22[j_1].complete_t=Q22[j_1-1].complete_t+Q22[j_1].remaining_t;
            }
}

        Q22[j_1].turnaround_t=Q22[j_1].complete_t - Q22[j_1].arriv;
        Q22[j_1].wait_t=Q22[j_1].turnaround_t - Q22[j_1].burst;


        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n",j_1,Q22[j_1].burst,Q22[j_1].wait_t,Q22[j_1].turnaround_t,Q22[j_1].complete_t);
j_1++;
} while((j_1<k1)) ;
}
```

```c
/*---------------calc_AvgTime_2 level MLQ--------------------------*/
void calc_AvgTime_level_2_MLQ(process Q11[],process Q22[]){
int tot_wait_t=0 ,
tot_turnaround_t=0;

for (int i1=0;i1<num_process1;i1++){
tot_wait_t+=Q11[i1].wait_t;
tot_turnaround_t += Q11[i1].turnaround_t;

tot_wait_t+=Q22[i1].wait_t;
tot_turnaround_t += Q22[i1].turnaround_t;
        }

printf("\nAverage Turn Around Time: %6.1f",(float)tot_turnaround_t/num_process1);
printf("\nAverage Waiting Time     : %6.1f \n",(float)tot_wait_t/num_process1);

fprintf(fh,"\nAverage Turn Around Time: %6.1f",(float)tot_turnaround_t/num_process1);
fprintf(fh,"\nAverage Waiting Time     : %6.1f \n",(float)tot_wait_t/num_process1);

}

/*------------------------2 level MLQ-----------------------------*/
void MLV2(process Q11[],process Q22[]){
time1=Q11[0].arriv;
printf("\nProcess\t\tBurst\t\tWaiting\t\tTurnaround\tComplete");

do{
        RR_MLV2(Q11, Q22);
        if(Q11[r1].arriv>Q11[r1-1].pointer && (k1!=0)){ // will not call FCFS_MLV2 until arrive time greater than finish time
        FCFS_MLV2(Q11, Q22);
        flag1 = 1;
        m1++;
        }
s1++;
}while(s1 < num_process1);

if (k1!= m1) //will not call FCFS_MLV2 until there are some process not executed yet
FCFS2(Q11, Q22);

calc_AvgTime_level_2_MLQ(Q11, Q22);
}
```

V. RESULTS AND DISCUSSION

A. Test Cases: FCFS, RR, and 2 MLQ

Usual Cases									
Test Case #	**Input**	**Test Case Description**	**Anticipated Output for FCFS**	**Output for FCFS**	**Anticipated Output for RR QT=2**	**Output for RR QT=2**	**Anticipated Output for 2 MLQ QT =2**	**Output for 2 MLQ QT =2**	**Pass or failed**
1	Arrival:0, Burst:3 Arrival:0, Burst:4 Arrival:0, Burst:2	All processes arrive at 0	AWT=3.3 ATAT=6.3	AWT=3.3 ATAT=6.3	AWT=4.3 ATAT=7.3	AWT=4.3 ATAT=7.3	AWT=4.3 ATAT=7.3	AWT=4.3 ATAT=7.3	Pass
2	Arrival:0, Burst:3 Arrival:2, Burst:4 Arrival:3, Burst:3	Processes arrives in different times with small difference	AWT=1.7 ATAT=5.0	AWT=1.7 ATAT=5.0	AWT=3.7 ATAT=7	AWT=3.7 ATAT=7	AWT=3.7 ATAT=7.0	AWT=3.7 ATAT=7.0	Pass
3	Arrival:0, Burst:3 Arrival:1, Burst:3 Arrival:3, Burst:3	All processes have same burst time	AWT=1.7 ATAT=4.7	AWT=1.7 ATAT=4.7	AWT=3.7 ATAT=6.7	AWT=3.7 ATAT=6.7	AWT=3.7 ATAT=6.7	AWT=3.7 ATAT=6.7	Pass
4	Arrival:0, Burst:24 Arrival:0, Burst:3 Arrival:0, Burst:3	All processes have same arrival time but process with big burst arrive first	AWT=17 ATAT=27	AWT=17.0 ATAT=27.0	AWT=6.3 ATAT=16.3	AWT=6.3 ATAT=16.3	AWT=19 ATAT=29	AWT=19.0 ATAT=29.0	Pass
Unusual Cases									
5	Arrival:0, Burst:60 Arrival:3, Burst:50 Arrival:4, Burst:40 QT=2	Processes have big burst time	AWT=54.3 ATAT=104.3	AWT=54.3 ATAT=104.3	AWT=85.6 ATAT=135.6	AWT=85.6 ATAT=135.6	AWT=55.0 ATAT=105.0	AWT=55.0 ATAT=105.0	Pass
6	Arrival:0, Burst:50 Arrival:3, Burst:150 Arrival:5, Burst:250 QT=15	Processes have big burst time with big difference	AWT=80.7 ATAT=230.7	AWT=80.7 ATAT=230.7	AWT=161.7 ATAT=311.7	AWT=161.7 ATAT=311.7	AWT=83.7 ATAT= 233.7	AWT=83.7 ATAT= 233.7	Pass
7	Arrival:2, Burst:3 Arrival:0, Burst:4 Arrival:0, Burst:2 TQ=15	Time quantum is much more than burst time of all processes	AWT=2.0 ATAT=5.0	AWT=2.0 ATAT=5.0	AWT=2.0 ATAT=5.0	AWT=2.0 ATAT=5.0	AWT=2.0 ATAT=5.0	AWT=2.0 ATAT=5.0	Pass

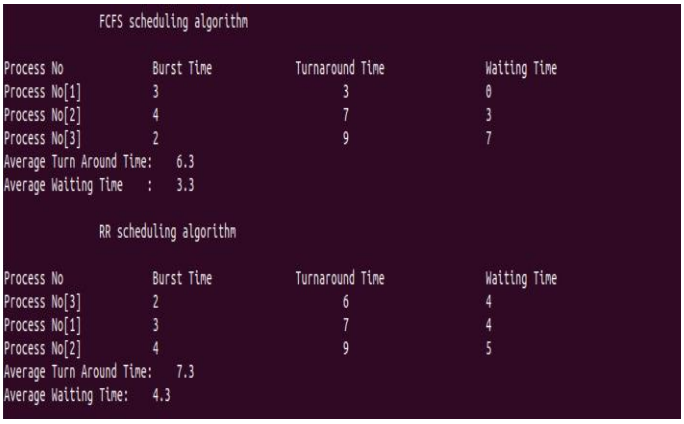
B. 3 MLQ

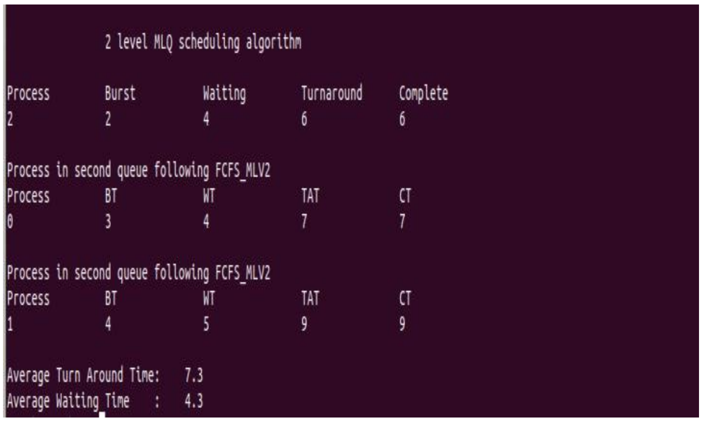
Usual Cases					
Test Case #	**Input**	**Test Case Description**	**Anticipated Output for FCFS**	**Output for FCFS**	**Pass or failed**
1	Arrival:0, Burst:53 Arrival:0, Burst:17 Arrival:0, Burst:68 Arrival:0, Burst:24 QT =17 QT=25	All processes arrive at 0	AWT=73.8 ATAT=114.2	AWT=73.8 ATAT=114.2	Pass
2	Arrival:0, Burst:6 Arrival:0, Burst:3 Arrival:0, Burst:1 QT =1 QT=2	All processes arrive at 0	AWT=3.3 ATAT=6.6	AWT=3.3 ATAT=6.7	Pass
3	Arrival:0, Burst:6 Arrival:1, Burst:3 Arrival:2, Burst:1 QT =1 QT=2	Processes arrives in different times with small difference	AWT=2.3 ATAT=5.6	AWT=2.3 ATAT=5.7	Pass
Unusual Cases					
4	Arrival:0, Burst:3 Arrival:0, Burst:4 Arrival:0, Burst:2 QT =15 QT=17	Time quantum is much more than burst time of all processes	AWT=3.3 ATAT=6.3	AWT=3.3 ATAT=6.3	Pass

C. Snapshots of Results

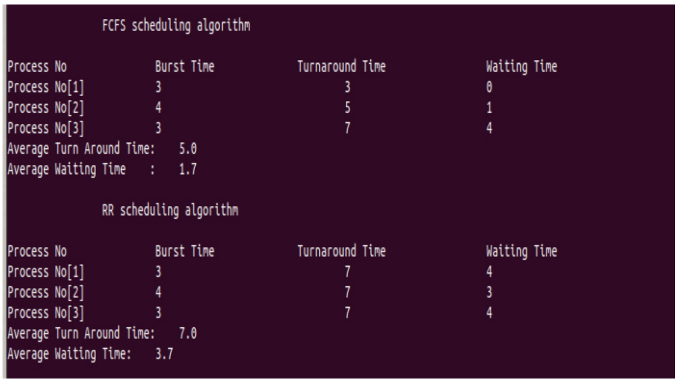
A. Test cases: FCFS, RR, and 2 MLQ

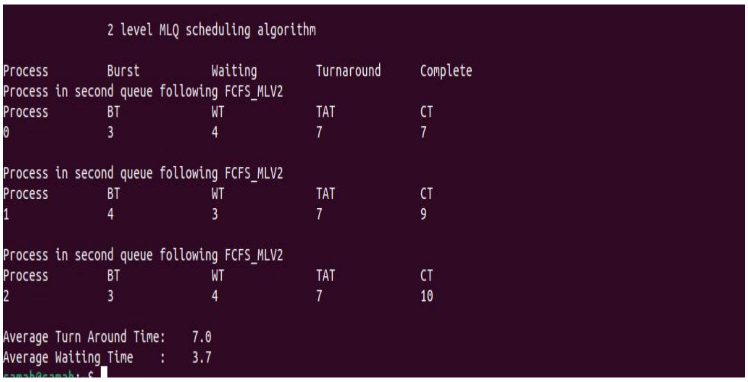
1. Test case #1:


```
FCFS scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[1]   3               3                    0
Process No[2]   4               7                    3
Process No[3]   2               9                    7
Average Turn Around Time:   6.3
Average Waiting Time   :   3.3

RR scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[3]   2               6                    4
Process No[1]   3               7                    4
Process No[2]   4               9                    5
Average Turn Around Time:   7.3
Average Waiting Time:   4.3
```

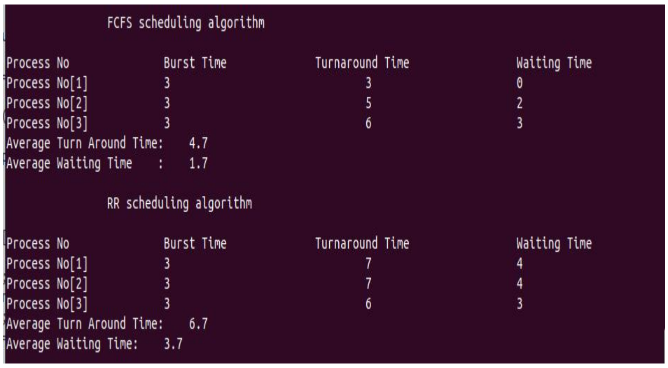

```
2 level MLQ scheduling algorithm

Process     Burst     Waiting     Turnaround     Complete
2           2         4           6              6

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
0           3         4           7              7

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
1           4         5           9              9

Average Turn Around Time:   7.3
Average Waiting Time   :   4.3
```

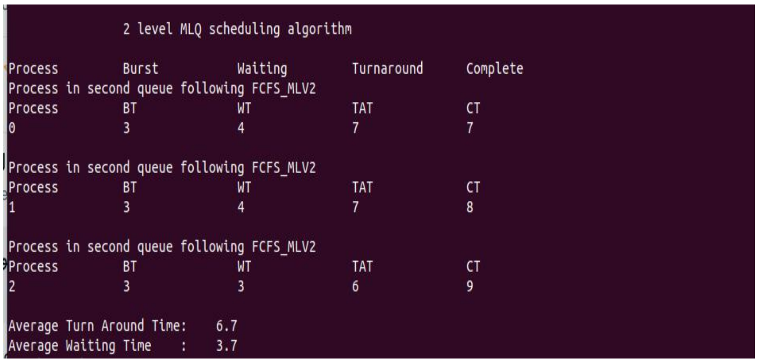
2. Test case # 2:


```
FCFS scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[1]   3               3                    0
Process No[2]   4               5                    1
Process No[3]   3               7                    4
Average Turn Around Time:   5.0
Average Waiting Time   :   1.7

RR scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[1]   3               7                    4
Process No[2]   4               7                    3
Process No[3]   3               7                    4
Average Turn Around Time:   7.0
Average Waiting Time:   3.7
```


```
2 level MLQ scheduling algorithm

Process     Burst     Waiting     Turnaround     Complete
Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
0           3         4           7              7

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
1           4         3           7              9

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
2           3         4           7              10

Average Turn Around Time:   7.0
Average Waiting Time   :   3.7
```

3. Test case #3:


```
FCFS scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[1]   3               3                    0
Process No[2]   3               5                    2
Process No[3]   3               6                    3
Average Turn Around Time:   4.7
Average Waiting Time   :   1.7

RR scheduling algorithm

Process No      Burst Time      Turnaround Time      Waiting Time
Process No[1]   3               7                    4
Process No[2]   3               7                    4
Process No[3]   3               6                    3
Average Turn Around Time:   6.7
Average Waiting Time:   3.7
```


```
2 level MLQ scheduling algorithm

Process     Burst     Waiting     Turnaround     Complete
Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
0           3         4           7              7

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
1           3         4           7              8

Process in second queue following FCFS_MLV2
Process     BT        WT          TAT            CT
2           3         3           6              9

Average Turn Around Time:   6.7
Average Waiting Time   :   3.7
```

4. Test case #4:

```
            FCFS scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       24                  24                       0
Process No[2]       3                   27                       24
Process No[3]       3                   30                       27
Average Turn Around Time:   27.0
Average Waiting Time    :   17.0

            RR scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[2]       3                   9                        6
Process No[3]       3                   10                       7
Process No[1]       24                  30                       6
Average Turn Around Time:   16.3
Average Waiting Time:    6.3
```

```
            2 level MLQ scheduling algorithm

Process         Burst           Waiting         Turnaround      Complete
Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
0               24              4               28              28

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
1               3               26              29              29

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
2               3               27              30              30

Average Turn Around Time:   29.0
Average Waiting Time    :   19.0
```

5. Test case #5:

```
            FCFS scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       60                  60                       0
Process No[2]       50                  107                      57
Process No[3]       40                  146                      106
Average Turn Around Time:  104.3
Average Waiting Time    :  54.3

            RR scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[3]       40                  118                      78
Process No[2]       50                  139                      89
Process No[1]       60                  150                      90
Average Turn Around Time:  135.7
Average Waiting Time:  85.7
```

```
            2 level MLQ scheduling algorithm
Process         Burst           Waiting         Turnaround      Complete
Process in second queue FCFS_MLV2
Process         BT              WT              TAT             CT
0               60              0               60              60

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
- here --64>

1               50              59              109             112

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
2               40              106             146             150

Average Turn Around Time:   105.0
Average Waiting Time    :    55.0
```

6. Test case #6:

```
            FCFS scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       50                  50                       0
Process No[2]       150                 197                      47
Process No[3]       250                 445                      195
Average Turn Around Time:  230.7
Average Waiting Time    :  80.7

            RR scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       50                  146                      96
Process No[2]       150                 344                      194
Process No[3]       250                 445                      195
Average Turn Around Time:  311.7
Average Waiting Time:  161.7
```

```
            2 level MLQ scheduling algorithm

Process         Burst           Waiting         Turnaround      Complete
Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
0               50              6               56              56

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
1               150             50              200             203

Process in second queue following FCFS_MLV2
Process         BT              WT              TAT             CT
2               250             195             445             450

Average Turn Around Time:   233.7
Average Waiting Time    :   83.7
```

7. Test case #7:

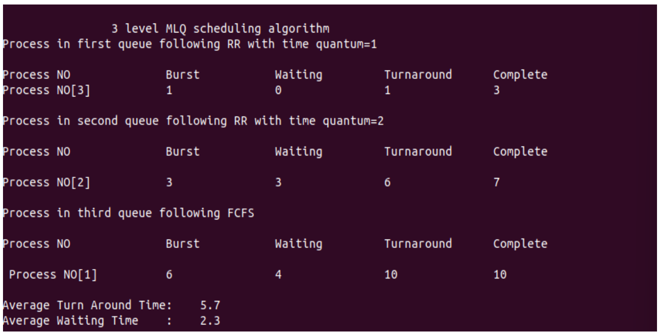
```
            FCFS scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       2                   2                        0
Process No[2]       4                   6                        2
Process No[3]       3                   7                        4
Average Turn Around Time:    5.0
Average Waiting Time    :    2.0

            RR scheduling algorithm

Process No          Burst Time          Turnaround Time          Waiting Time
Process No[1]       2                   2                        0
Process No[2]       4                   6                        2
Process No[3]       3                   7                        4
Average Turn Around Time:    5.0
Average Waiting Time:    2.0
```

```
            2 level MLQ scheduling algorithm

Process     Burst       Waiting     Turnaround  Complete
0           2           0           2           2

1           4           2           6           6

2           3           4           7           9

Average Turn Around Time:   5.0
Average Waiting Time    :   2.0
```

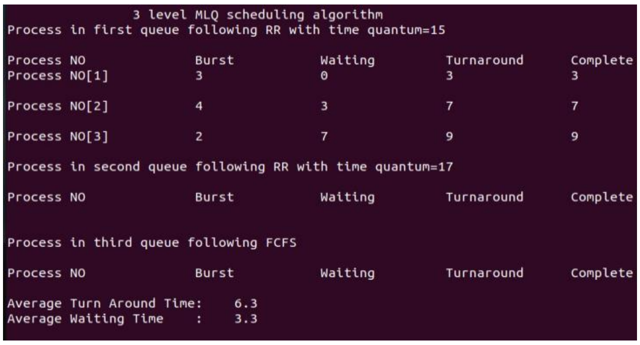
B. Test cases: 3 MLQ

```
            3 level MLQ scheduling algorithm
Process in first queue following RR with time quantum=17

Process NO          Burst       Waiting     Turnaround      Complete
Process NO[2]       17          17          34              34

Process in second queue following RR with time quantum=25

Process NO          Burst       Waiting     Turnaround      Complete

Process NO[3]       24          101         125             125

Process in third queue following FCFS

Process NO          Burst       Waiting     Turnaround      Complete

 Process NO[1]      53          83          136             136

 Process NO[2]      68          94          162             162

Average Turn Around Time:  114.2
Average Waiting Time    :   73.8
```

```
            3 level MLQ scheduling algorithm
Process in first queue following RR with time quantum=1

Process NO          Burst       Waiting     Turnaround      Complete
Process NO[3]       1           2           3               3

Process in second queue following RR with time quantum=2

Process NO          Burst       Waiting     Turnaround      Complete

Process NO[2]       3           4           7               7

Process in third queue following FCFS

Process NO          Burst       Waiting     Turnaround      Complete

 Process NO[1]      6           4           10              10

Average Turn Around Time:    6.7
Average Waiting Time    :    3.3
```



```
          3 level MLQ scheduling algorithm
Process in first queue following RR with time quantum=1

Process NO          Burst       Waiting     Turnaround  Complete
Process NO[3]       1           0           1           3

Process in second queue following RR with time quantum=2

Process NO          Burst       Waiting     Turnaround  Complete

Process NO[2]       3           3           6           7

Process in third queue following FCFS

Process NO          Burst       Waiting     Turnaround  Complete

 Process NO[1]      6           4           10          10

Average Turn Around Time:    5.7
Average Waiting Time    :    2.3
```



```
          3 level MLQ scheduling algorithm
Process in first queue following RR with time quantum=15

Process NO          Burst       Waiting     Turnaround  Complete
Process NO[1]       3           0           3           3

Process NO[2]       4           3           7           7

Process NO[3]       2           7           9           9

Process in second queue following RR with time quantum=17

Process NO          Burst       Waiting     Turnaround  Complete


Process in third queue following FCFS

Process NO          Burst       Waiting     Turnaround  Complete

Average Turn Around Time:    6.3
Average Waiting Time    :    3.3
```

D. Comparison of the Algorithms

There are various test cases that show the robustness of our code. During the testing stage, we discovered the following: the waiting time of FCFS depends on the arrival order of the process, and short processes can be stuck waiting until long processes are complete, proving what was learned in lecture time (as we will see later), whereas the waiting time of RR depends on choosing the quantum time, which is critical. If the quantum time is very long, as shown in test cases 5 and 6, the RR can adopt the FCFS approach.

In all usual and unusual test case, FCRS is better than RR with a little different. However, as we mentioned before, FCFS has bad performance when the long processes are executed first, while RR has good performance in this case. This shown in 4th test case. In the multilevel schedule algorithm in this algorithm queue are classified into two groups, first containing background processes and second containing foreground processes.

In addition, as shown in the test cases, there is no such difference to be mentioned in the average waiting time and turnaround time between FCFS (first usual case) and three multilevel queues(first unusual case), also in the average waiting time and turnaround time of RR(first usual case) and two multilevel queues (first usual case) in the case of small burst time and same arrival ,but the difference and the efficiency and the performance of the algorithms appear on the large burst time process as shown in the test cases FCFS(first unusual case) is the least average waiting time and turnaround time between the RR(first unusual case) with a gap about 30ms which is huge slake of the CPU the gap in RR due to the many context switches happen although RR algorithm will make CPU busy mostly all time , FCFS(first unusual case) has no big difference between MLQ(2)(first unusual)&(3) which is about 0.7 ms but it is preferable to use MLQ due the utilization of CPU if we have many cores , on the other hand, FCFS is more likely to be used as it is easy to be implemented.

VI. CONCLUSION

To sum up, we followed the form's contents in accordance with the report's guidelines for this report. We used different algorithms which are First Come First Serve and Round Robin. Then, we have used them to implement multilevel queue scheduling. We were able to compare between these algorithms. We were able to learn many new concept and implementation of different scheduling methods. Additionally, we gave a snapshot of each case with explaining the processes. Finally, we ended the report with the references section.

REFERENCE

[1] A. Tanenbaum and H. T. Boschung, *Modern Operating Systems*, 4th ed. Etats-Unis: Pearson, 2018.

[2] D. Thakur, "Operating system scheduling algorithms," *Computer Notes*, 10-Aug-2020. [Online]. Available: https://ecomputernotes.com/fundamental/disk-operating-system/scheduling-algorithms#The_multilevel_Queue_scheduling_algorithm [Accessed: 12-Nov-2022].

[3] C. ashush (2020) Difference between first come first served (FCFS) and Round Robin (RR) scheduling algorithm, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/difference-between-first-come-first-served-fcfs-and-round-robin-rr-scheduling-algorithm/(Accessed: November 30, 2022).

[4] *Multilevel Feedback Queue - MLFQ شرح نوف البدر*. nouf badr, 2018.

[5] *Multi-level Feedback Queue Scheduling Algorithm with Example | CPU Scheduling Algorithms in OS*. 2018.

[6] *Multilevel Queue Scheduling Algorithm with Example | CPU Scheduling Algorithms in Operating Systems*. Simple Snippets, 2018.

[7] *13- C programming - Structures – Part1 - محمد يوسف*. Mohamed Yousef 2020 , محمد يوسف.

Appendices

Task	Abeer	Leen	Shatha	Salwa	Samah
Report					✓
Abstract		✓			
1.0 Introduction				✓	
2.0 CPU Scheduling Algorithms			✓		
3.0 Pseudocode	✓	✓	✓	✓	✓
4.0 Results and Discussion					
4.1 Test Cases	✓	✓	✓	✓	✓
4.2 Snapshots of Results		✓			✓
4.3 Comparison of the Algorithms		✓	✓	✓	
5.0 Conclusion		✓			
Code					
Implement algorithm 1				✓	✓
Implement algorithm 2	✓	✓	✓		
Implement multilevel algorithms 1	✓	✓	✓		
Implement multilevel algorithms 2				✓	✓
Integration the code				✓	✓

Handle error in program:

A)

```
int main(int argc){
    // opening file in writing mode
    fh = fopen("output.txt", "w");

    // exiting program
    if (fh == NULL) {
        printf("Error opening file.!\n");
        exit(1);
    }
int choice;
while (flag == 0){//<--- handel error if the user enters incorrect choice not 1 0r 2
```

B)

```
// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<num_process; i++){
printf("\nEnter the Arrival and Burst time of the Process[%d]\n", (i+1));
printf("Arrival time is: ");  // Accept arrival time
scanf("%d", &p[i].arriv);
Q1[i].arriv= p[i].arriv;
Q11[i].arriv= p[i].arriv;
printf("Burst time is: "); // Accept the Burst time
scanf("%d", &p[i].burst);
Q1[i].burst= p[i].burst;
Q11[i].burst= p[i].burst;
Q1[i].remaining_t=Q1[i].burst;
Q11[i].remaining_t=Q11[i].burst;
temp[i] = p[i].burst ; // store the burst time in temp array
        }

sortByArrival_priority();// <-----handel error if the processes not enter in correct order
```

C)

```
while (flag ==0 ){ // <------ handel error if quant2 < quant1
printf("Enter the Time Quantum for RR1 of MLQ: ");
scanf("%d", &quant1);

printf("Enter the Time Quantum for RR2 of MLQ: ");
scanf("%d", &quant2);

if (quant1<= quant2)
flag = 1;
        }
flag=0;
```