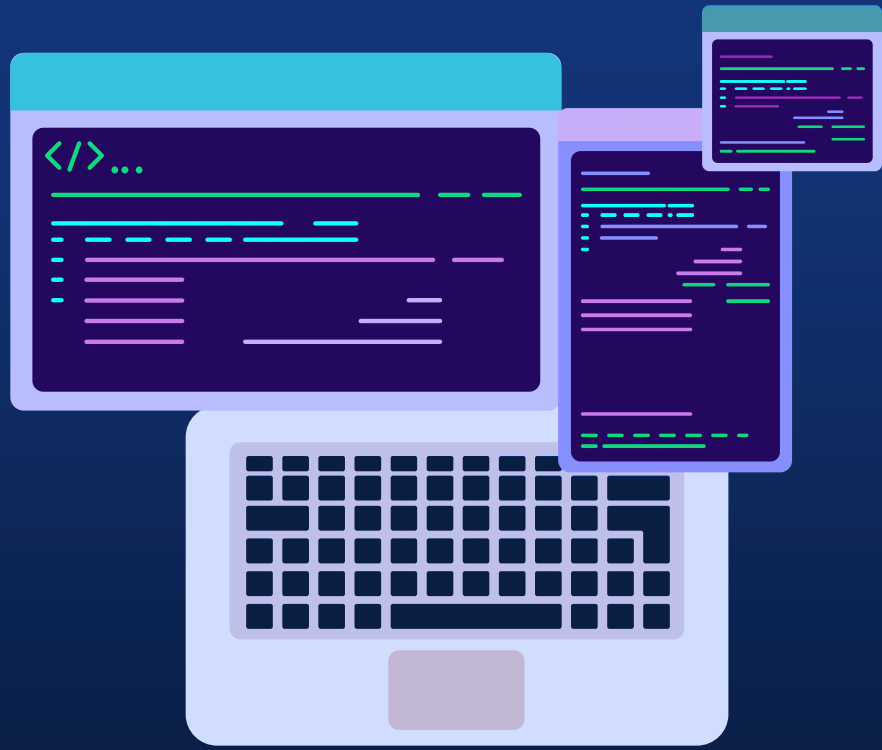
OPERATING SYSTEM

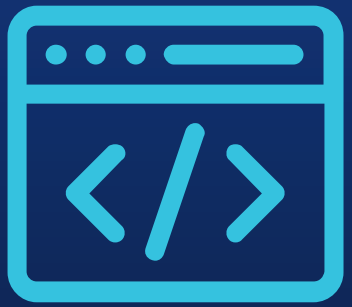
CCCS 225 PROJECT

Multilevel Feedback Queue



NAMES, IDS AND TASKS OF GROUP MEMBERS

N A M E	I D
Teif Saleh	2111370
Ruba Alsulami	2110618
Arjwan Alharbi	2110826



Topics

- Introduction
- Code & Explain
- Show The Output

Tasks

- Teif : TAT, AVG TAT, PCB
- Ruba : RT, AVG RT, Throughput
- Arjwan : WT, AVG WT, User Input

01

INTRODUCTION

MLFQ

“Processes in Multilevel Feedback Queue Scheduling (MFQS) can move between the queues.
Multiple CPUs are available for load sharing. This procedure helps to avoid starvation by moving the waiting processes from the low-priority queue to the high-priority queue, and the processes that take up too much CPU time to the lower-priority queue.”

In This Project :

Queue 1 apply RR with q = 8 ms
Queue 2 apply RR with q = 16 ms
Queue 3 apply FCFS

02

CODE

```c
#include <stdio.h>
#include <stdlib.h>

struct Process { //struct to store information of process
    int num; //name of process
    int cpu_burst; //burst time for process
    int waiting; //waiting time for each process
    int response; //response time for each process
    int turnaround; //turnaround time for each process
};
```

Code explanation

First, we created a structure called Process to hold different data types of process information.

```c
int main(){
    printf("    > Enter total number of processes: "); //Get total
        number of processes from the user
    int SIZE;
    scanf("%d", &SIZE); //input total number of processes
    while(SIZE<=0){
       printf("    > Enter again: ");
       scanf("%d", &SIZE);
       }

    int* remaining_cpu = malloc(sizeof(int)*SIZE); //declare
        remaining burst time array
    struct Process** processes = malloc(sizeof(struct Process
        *)*SIZE); //processes array
    //Queue 1 using RR algorithm with 8 time quantum
    struct Process** Queue1 = malloc(sizeof(struct Process*)*SIZE);
    for(int i=0; i<SIZE; i++){ //loop to get information for all
        processes
        int temp; //store burst time for each process
        processes[i] = malloc(sizeof(struct Process)); //declare
```

Code explanation

We start asking the user for the number of processes. Then we have a declaration for several arrays.

```c
for(int i=0; i<SIZE; i++){ //loop to get information for all
    processes
    int temp; //store burst time for each process
    processes[i] = malloc(sizeof(struct Process)); //declare
        each process
    printf("    > Enter burst time %d: ", i + 1); //enter burst
        time for each process
    scanf("%d", &temp); //get burst time for each process
    while(temp<=0){
       printf("    > Enter again: ");
       scanf("%d", &temp);
   }
    processes[i]->num = i; //store name for each process
    processes[i]->cpu_burst = temp; //store cpu time for each
        process
    remaining_cpu[i] = processes[i]->cpu_burst; //store
        remaining time for each process
    Queue1[i] = processes[i]; //store each process in Queue 1
}
```

Code explanation

Here is a for-loop that:
Prompt the user to enter the burst time for each process (according to the number entered in the previous step)
Store information about each process (process name/number - burst time - remaining time)
Note: Arrival time was set to zero
Enter and store each process in the first queue

```
39      int arrival = 0; // initialize arrival time to zero
40      int counter = 0; //initialize timer to zero
41      int total_wait = 0; //initialize total wait time to zero
42      int total_turnaround = 0; //initialize total turnaround time to
            zero
43      int total_response = 0; //initialize total response time to
            zero
44      int x = 0; //variable to store size of queue 1
45      int y = 0; //variable to store size of queue 2
46      int z = 0; //variable to store size of queue 3
47      int i = 0; //variable to iterate queue 2
48      int j = 0; //variable to iterate queue 3
49
50      //Queue 2 with RR algorithm with 16ms time quantum
51      struct Process** Queue2 = malloc(sizeof(struct Process*) * SIZE
            );
52      //Queue 3 with FCFS scheduling
53      struct Process** Queue3 = malloc(sizeof(struct Process*) * SIZE
            );
54
55      int executed = 0; //total executed processes
```

Code explanation

We set the timer (counter) and the total of waiting time, turnaround time, response time, and processes that had been executed to zero. Also, we declared the second and third queue.

```
56    while (executed != SIZE) { //while not all processes are executed
57        if (x < SIZE) { //if queue 1 has processes
58            Queue1[x]->response = counter; //response time for each process
59            total_response += counter; //calculate response time
60            if (remaining_cpu[Queue1[x]->num] <= 8) { //if remaining time is less than 8ms
61
62                counter += remaining_cpu[Queue1[x]->num]; //increase timer to remaining time for process
63                executed++; //increase number of executed processes by 1
64                Queue1[x]->waiting = counter - Queue1[x]->cpu_burst ;  //waiting time for each process
65                Queue1[x]->turnaround = Queue1[x]->cpu_burst + Queue1[x]->waiting; //turnaround time for each process
66                total_wait += counter - Queue1[x]->cpu_burst; //calculate total wait time
67                 total_turnaround += Queue1[x]->cpu_burst + Queue1[x]->waiting; //calculate total turnaround time
68                 remaining_cpu[Queue1[x++]->num] = 0; //set remaining time of process to zero
69            }
```

Code explanation

While – loop with nested if – else statements to execute all processes. If the Q1 has processes, then start calculating the response time for each process & its total. If the remining time of the process <= the quantum 8, timer and number of executed processes will increase. Then, start calculating the WT & AVG WT, TAT & AVG TAT.

Note : set remining time to zero.

```
70          else {
71              counter += 8; //set timer to quantum of queue 1
72              Queue2[y++] = Queue1[x]; //move process to queue 2
73              remaining_cpu[Queue1[x++]->num] -= 8; //set
                    remaining time of process
74          }
75      }
```

Code explanation

If the remining time != the quantum 8, then increase the timer with quantum 8 and move the process from Q1 to Q2.

Note : Decrement the remaining time by 8

```
76      else if (i < y) {  //if queue 2 has processes
77          if (remaining_cpu[Queue2[i]->num] <= 16) { //if
                remaining time is less than 16ms
78
79              counter += remaining_cpu[Queue2[i]->num];
                    //increase timer to remaining time for process
80              Queue2[i]->waiting = counter - Queue2[i]->cpu_burst
                    ;  //waiting time for each process
81              Queue2[i]->turnaround = Queue2[i]->cpu_burst +
                    Queue2[i]->waiting; //turnaround time for each
                    process
82              total_wait += counter - Queue2[i]->cpu_burst;
                    //calculate total wait time
83              total_turnaround += Queue2[i]->cpu_burst +
                    Queue2[i]->waiting; //calculate total
                    turnaround time
84              remaining_cpu[Queue2[i++]->num] = 0; //set
                    remaining time of process to zero
85              executed++; //increase number of executed processes
                    by 1
86          }
```

Code explanation

While – loop with nested if – else statements to execute all processes. If the Q2 has processes, then start calculating the response time for each process & its total. If the remining time of the process <= the quantum 16, timer and number of executed processes will increase. Then, start calculating the WT & AVG WT, TAT & AVG TAT.

```
87          else {
88              counter += 16; //set timer to quantum of queue 2
89              Queue3[z++] = Queue2[i]; //move process to queue 3
90              remaining_cpu[Queue2[i++]->num] -= 16; //set
                    remaining time of process
91          }
92      }
```

Code explanation

If the remining time != the quantum 16, then increase the timer with quantum 8 and move the process from Q2 to Q3.

Note : Decrement the remaining time by 16

```
93      else if (j < z) { //if queue 3 has processes
94
95          counter += remaining_cpu[Queue3[j]->num]; //set timer
                to remaining time of process
96          Queue3[j]->waiting = counter - Queue3[j]->cpu_burst;
                //waiting time for each process
97          Queue3[j]->turnaround = Queue3[j]->cpu_burst +
                Queue3[j]->waiting; //turnaround time for each
                process
98          total_wait += counter - Queue3[j]->cpu_burst;
                //calculate total wait time
99          total_turnaround += Queue3[j]->cpu_burst + Queue3[j]
                ->waiting; //calculate total turnaround time
00          remaining_cpu[Queue3[j++]->num] = 0; //set remaining
                time of process to zero
01          executed++; //increase number of executed processes by
                1
02      }
03  }
```

Code explanation

While – loop with nested if – else statements to execute all processes. If the Q3 has processes, then start calculating the response time for each process & its total. If the remining time of the process > the quantum 16, timer and number of executed processes will increase. Then, start calculating the WT & AVG WT, TAT & AVG TAT.

```
105     for (int i = 0; i < SIZE; i++) {
106         printf("     _____________________________________\n");
107         printf("    |\t       PCB-PROCESS %d              |\n", (i
                + 1));
108         printf("    |_____________________________________|\n");
109         printf("    |>> Burst time     [process %d] : %.2d    |\n",
                processes[i]->num + 1, processes[i]->cpu_burst);
110         printf("    |>> Waiting time   [process %d] : %.2d    |\n",
                processes[i]->num + 1, processes[i]->waiting);
111         printf("    |>> Trnaround time [process %d] : %.2d    |\n",
                processes[i]->num + 1, processes[i]->turnaround);
112         printf("    |>> Response time  [process %d] : %.2d    |\n",
                processes[i]->num + 1, processes[i]->response);
113         printf("    |_____________________________________|\n");
114         printf("\n");
115     }
```

Code explanation

For loop to print all the queue's information

Which is:
-Burst time
-Waiting time
-Trnaround time
-Response time

```
121     printf("    |(1) Average waiting time : %.2f      |\n", (float
            )total_wait / (float)SIZE);
122     printf("    |_____________________________________|\n");
123     printf("\n");
124     printf("     _____________________________________\n");
125     printf("    |                                     |\n");
126     printf("    |(2) Average turnaround time : %.2f   |\n", (float
            )total_turnaround / (float)SIZE);
127     printf("    |_____________________________________|\n");
128     printf("\n");
129     printf("     _____________________________________\n");
130     printf("    |                                     |\n");
131     printf("    |(3) Average response time : %.2f     |\n", (float
            )total_response / (float)SIZE);
132     printf("    |_____________________________________|\n");
133     printf("\n");
134     printf("     _____________________________________\n");
135     printf("    |                                     |\n");
136     printf("    |(4) Throughput : %.2f                |\n", (float
            )SIZE / (float)counter);
137     printf("    |_____________________________________|\n");
138     return 0;
```

Code explanation

At the end it will print all the process information

Which is:
- Average Waiting time
- Average Trnaround time
-Average Response time
-Throughput

03

OUTPUT

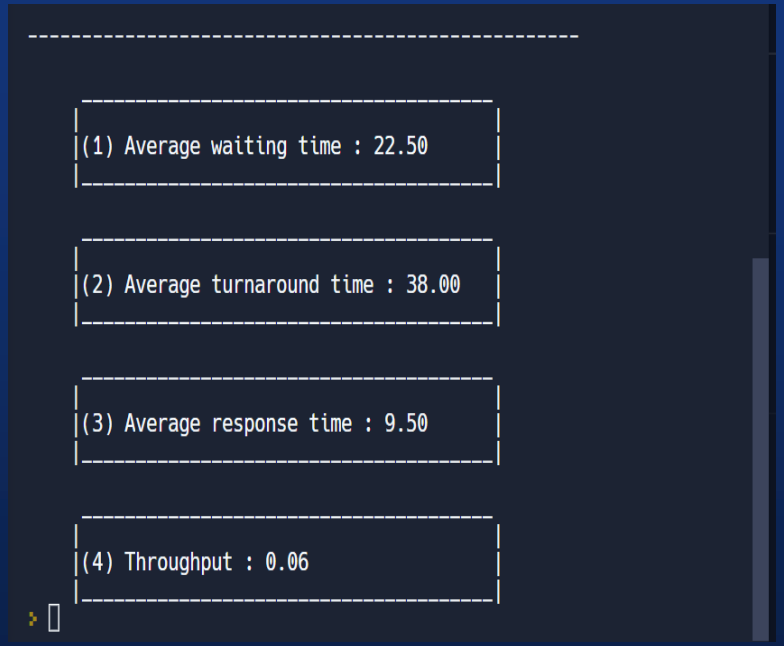
```
clang-7 -pthread -lm -o main main.c
./main
   > Enter total number of processes: 4
   > Enter burst time 1: 33
   > Enter burst time 2: 3
   > Enter burst time 3: 19
   > Enter burst time 4: 7
-----------------------------------------------------

 ____________________________________
|          PCB-PROCESS 1             |
|____________________________________|
|>> Burst time     [process 1] : 33  |
|>> Waiting time   [process 1] : 29  |
|>> Trnaround time [process 1] : 62  |
|>> Response time  [process 1] : 00  |
|____________________________________|

 ____________________________________
|          PCB-PROCESS 2             |
|____________________________________|
|>> Burst time     [process 2] : 03  |
|>> Waiting time   [process 2] : 08  |
|>> Trnaround time [process 2] : 11  |
|>> Response time  [process 2] : 08  |
|____________________________________|
```

```
 ____________________________________
|          PCB-PROCESS 3             |
|____________________________________|
|>> Burst time     [process 3] : 19  |
|>> Waiting time   [process 3] : 34  |
|>> Trnaround time [process 3] : 53  |
|>> Response time  [process 3] : 11  |
|____________________________________|

 ____________________________________
|          PCB-PROCESS 4             |
|____________________________________|
|>> Burst time     [process 4] : 07  |
|>> Waiting time   [process 4] : 19  |
|>> Trnaround time [process 4] : 26  |
|>> Response time  [process 4] : 19  |
|____________________________________|
```


(1) Average waiting time : 22.50
(2) Average turnaround time : 38.00
(3) Average response time : 9.50
(4) Throughput : 0.06