

CSCI 3150 Introduction to Operating Systems

Bonus Assignment Three

Deadline: 23:59, Nov. 28, 2021

Total Marks: 100

1. 100 marks

Implement the MLFQ (Multi-Level Feedback Queue) scheduler in [Chapter 8](#) (Book: Operating Systems: Three Easy Pieces) by following the rules below:

- Rule 1: If $\text{Priority}(A) > \text{Priority}(B)$, A runs (B doesn't).
- Rule 2: If $\text{Priority}(A) = \text{Priority}(B)$, A & B run in round-robin fashion using the time slice (quantum length) of the given queue.
- Rule 3: When a job enters the system, it is placed at the highest priority (the topmost queue). For the jobs arriving at the same time, schedule the job with smallest pid first.
- Rule 4: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue and will be at the tail of the target queue, which means it will be scheduled last).
- Rule 5: After some time period S, move all the jobs in the system to the topmost queue, and sort all the jobs by pid. The job with the smallest pid will be scheduled first.

In this question, we assume the time unit is millisecond (ms). Please note that in this assignment, we have different time allotment for different queue.

The inputs of your program: **process.file** and **queue.cfg**

- **process.file** contains the process information and its format is as follows:

```
ProcessNum N
pid:X1, arrival_time:X11, execution_time:X12
pid:X2, arrival_time:X21, execution_time:X22
...
pid:YN, arrival_time:YN1, execution_time:YN2
```

Here, the first line denotes there are N processes to be scheduled (there is at least one space character between ProcessNum and N), and from the second line to the (N+1)*th* line, each shows the pid (process id), arrival time and execution time of a process with the format like “pid:X1, arrival_time:X11, execution_time:X12” (separated by “,”). To make it simple, you can assume that all pids are different. An example is given below.

```
ProcessNum 5
pidnum:123, arrival_time:60, execution_time:90
pidnum:13, arrival_time:70, execution_time:100
pidnum:1023, arrival_time:10, execution_time:160
pidnum:12, arrival_time:80, execution_time:28
pidnum:36, arrival_time:10, execution_time:10
```

- **queue.cfg** contains the queue information and its format is as follows:

```
QueueNum n
Period_S S
Time_Slice_QN Xn Allotmenttime_QN Yn
...
Time_Slice_Q2 X2 Allotmenttime_Q2 Y2
Time_Slice_Q1 X1 Allotmenttime_Q1 Y1
```

Here, the first line denotes the number of queues we will use in the scheduler, the second line denotes the period S to move up all jobs to the topmost queue (Rule 5), the third line denotes the time slice for Queue n that is the topmost queue with the highest priority (the smallest time slice), and the subsequent lines define other queues with descending order in terms of priority. In each line, between the keyword and number, there should be at least one space character. One example is shown below.

```
QueueNum 3
Period_S 300
Time_Slice_Q3 10 Allotmenttime_Q3 30
Time_Slice_Q2 40 Allotmenttime_Q2 80
Time_Slice_Q1 60 Allotmenttime_Q1 120
```

The output of your program: **output.log**

- Your program should output the schedule to the file, **output.log**
- The format in output.log is as follows (This is very important as we will check the output based on this format):

Time_slot:x-y, pid:x1, arrival-time:x2, remaining_time:x3

Here, Time_slot:x-y denotes the time interval starting at time x and end at time y (Commonly the time interval is the time slice of current queue. If one process finishes without using the whole time slice, the time interval will be the time it used, and the smallest time interval is 1 ms.), pid:x1 denotes the corresponding process with pid x1 is scheduled in this time interval, arrival-time:x2 and remaining_time:x3 denote the arrival time and remaining time of the process are x2 and x3, respectively. There is at least one space character between them. An example is shown below (part of the result of the given information):

```
Time_slot:10-20, pid:36, arrival-time:10, remaining_time:0
Time_slot:20-30, pid:1023, arrival-time:10, remaining_time:150
Time_slot:30-40, pid:1023, arrival-time:10, remaining_time:140
Time_slot:40-50, pid:1023, arrival-time:10, remaining_time:130
Time_slot:50-90, pid:1023, arrival-time:10, remaining_time:90
Time_slot:90-100, pid:123, arrival-time:60, remaining_time:80
Time_slot:100-110, pid:13, arrival-time:70, remaining_time:90
Time_slot:110-120, pid:12, arrival-time:80, remaining_time:18
.....
Time_slot:378-398, pid:1023, arrival-time:10, remaining_time:0
```

Submission:

In this Assignment, you need to revise function

```
void scheduler(Process* proc, LinkedQueue** ProcessQueue, int proc_num, int queue_num, int period);
```

to implement the MLFQ (Multi-Level Feedback Queue) scheduler.

You only need to submit `scheduler-impl.c` that can be compiled with the same “Makefile”.

TA Zhang Kai is in charge of this bonus assignment, if you have any questions about this bonus assignment, you can enquiry with this email:

kzhang@cse.cuhk.edu.hk

Attachment:

Introduction to Codes Provided in Bonus Assignment Three

In this document, we will present how the scheduler provided works.

1. Compile and Run

Download and unzip Bonus-Assign2.zip into your current directory and compile it as follows:

make

Then you can run it as follows:

./Scheduler

After this, you will get output like this:

```
Process number: 5
36 10 10
1023 10 160
123 60 90
13 70 100
12 80 28

Queue number: 3
Period: 300
0 60 120
1 40 80
2 10 30
```

2. Implementation

Three groups of functions in the program are:

- (i) Read processes information from process.file and sort them.

```
int ReadProcessFile(); // Return the number of process in process.file, all the processes are
                        // stored in proc.
int min(int x, int y); // Return the less one between x and y.
Process MinProc(Process x, Process y); // Return the process arrive earlier; if arrive at the
                                        // same time, return the one have less pid.
void SortProcess(Process* p, int num); // Sort proc according to arrival_time and pid.
```

- (ii) Read queues information from queue.cfg file.

```
int GetQueueNum(); // Return the number of queue in queue.cfg.
int GetPeriod(); // Return the value of Period_S in queue.cfg.
void ReadQueueCfg(LinkedQueue** LQueue, int num); // Store queues into LQueue.
```

- (iii) Output functions and **scheduler function that you need to implement.**

```
void InitOutputFile(); // Create void output.log file.
void outprint(int time_x, int time_y, int pid, int arrival_time, int remaining_time);
    // Print one line to output.log file.
void scheduler(Process* proc, LinkedQueue** ProcessQueue, int proc_num, int
queue_num, int period);
```

For above functions, if you are interested in the implementation, you can look up the source code **scheduler-exec.c**.

Main function:

```
Process* proc_tmp; // Used to temporarily store the input process.
int main(){
    /*
        Following codes will read and sort processes from process.file,
        and store the sorted processes into array proc[].
    */
    int proc_num = ReadProcessFile();
    Process proc[proc_num];
    for (int i = 0; i < proc_num; i++){
        proc[i].process_id = proc_tmp[i].process_id;
        proc[i].arrival_time = proc_tmp[i].arrival_time;
        proc[i].execution_time = proc_tmp[i].execution_time;
    }
    SortProcess(proc, proc_num);

    /*
        Following codes will read queues from queue.cfg,
        and store them into ProcessQueue.
    */
    int queue_num = GetQueueNum();
    int period = GetPeriod();
    LinkedQueue** ProcessQueue = (LinkedQueue**)malloc(sizeof(LinkedQueue*) *.
        queue_num);
    ReadQueueCfg(ProcessQueue, queue_num);

    /*
        Initiate output.log file.
    */
    InitOutputFile();

    //Call scheduler() here.
    scheduler(proc, ProcessQueue, proc_num, queue_num, period);

    return 0;
}
```