오퍼레이팅 시스템 - 003

Process Scheduling Simulation
Using
Multi-level Queue Algorithm

12180152 끼서폿
sophotk@gmail.com

I. Idea

In multi programmable systems, scheduling which processes to get the cpu resource is very crucial. Therefore, in this assignment we will try to get a better understanding of how to schedule threads with simulation.

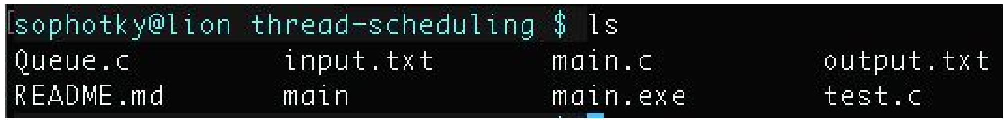
II. Approach

There are several ways to schedule processes or threads. The most effective, yet difficult to implement is the multi-level queue scheduling. In this assignment, I will be using 3 queues:

- Queue_1: First Come First Serve algorithm, highest priority queue.
- Queue_2 : Priority algorithm (the bigger the value the lower priority).
- Queue_3 : Shortest Job First algorithm, lowest priority queue.

Processes in Queue_3 will be scheduled only when Queue_2 is empty and Queue_2 can be scheduled only if Queue_1 is empty.

III. Folder Structure



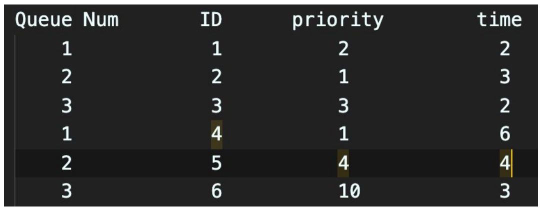
```
[sophotky@lion thread-scheduling $ ls
Queue.c         input.txt       main.c          output.txt
README.md       main            main.exe        test.c
```

In the zip file, there are:

- "input.txt" : the sample input to our program with order of
  (queue number, ID, priority, time)
- "output.txt" : result of after all threads have been scheduled
- "main.c" : main source code
- "Queue.h" : contains implementation of queue data structure with methods to select the thread to schedule based on the requirements.
- "main.exe" : executable file

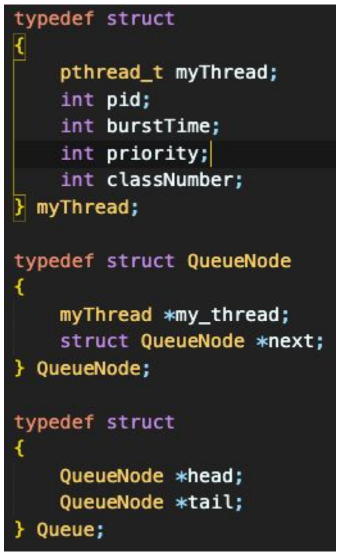
IV. Program Flow and Code Explanation

First of all, we have to create a file object that opens *"input.txt"* to read the sample inputs and store them into *myThread* which is stored in the queues.
Example:



```
Queue Num       ID      priority        time
    1            1         2              2
    2            2         1              3
    3            3         3              2
    1            4         1              6
    2            5         4              4
    3            6         10             3
```

In Queue_1 there are Process ID 1 & 4.
In Queue_2 there are Process ID 2 & 5
In Queue_3 there are Process ID 3 & 6

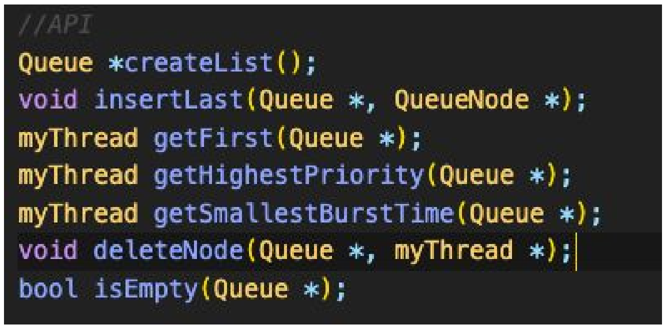


```c
typedef struct
{
    pthread_t myThread;
    int pid;
    int burstTime;
    int priority;
    int classNumber;
} myThread;

typedef struct QueueNode
{
    myThread *my_thread;
    struct QueueNode *next;
} QueueNode;

typedef struct
{
    QueueNode *head;
    QueueNode *tail;
} Queue;
```

myThread is a structure that holds input datas and *pthread_t* data. Also *myThread* is a member of *QueueNode* which will be used to implement the queue data structure.

I implemented Queue data structure methods as follows:

- getFirst() will be used with the FCFS queue.
- getHighestPriority() will be used with the Priority queue.
- getSmallestBurstTime() will be used with the SJF queue.
- deleteNode() will delete the process from queue after it is selected by the scheduler



```c
//API
Queue *createList();
void insertLast(Queue *, QueueNode *);
myThread getFirst(Queue *);
myThread getHighestPriority(Queue *);
myThread getSmallestBurstTime(Queue *);
void deleteNode(Queue *, myThread *);
bool isEmpty(Queue *);
```

After saving all *myThread* data in the queues, I created a thread to schedule all the threads in the queues. In the schedule thread, I used 3 while loops. The first while checks if Queue_1 is not empty, the first thread in the queue will be selected to execute. The second while selects any thread that has the highest priority value. The last while selects any thread that has the smallestTime. After selecting a thread, that thread will be removed from the ready queue by calling *deleteNode()* function.

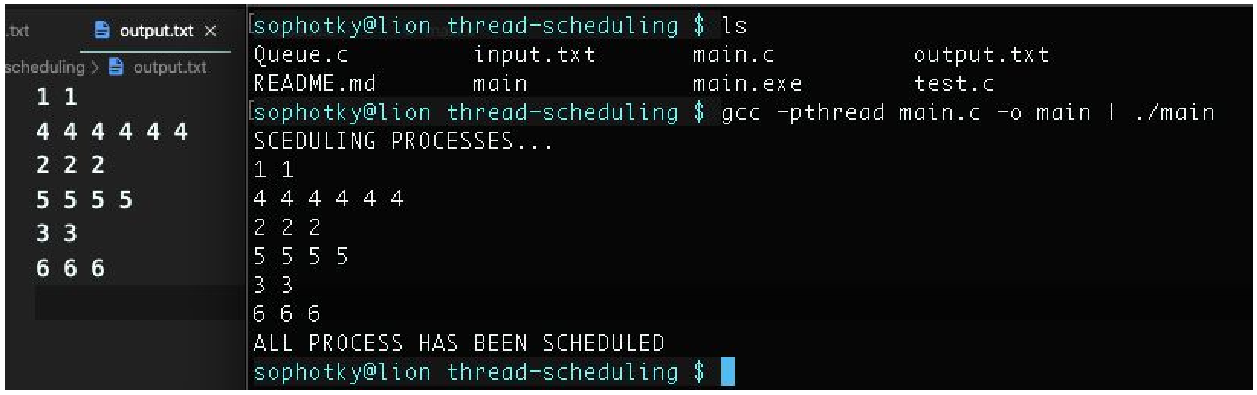
To test if everything is working fine, when selecting any thread in any queue, we will tell the thread to print out its own ID to "output.txt" and also to the terminal the amount of *time* value it is required.

V. Source Code Compilation

In terminal type:

“gcc -pthread main.c -o main | ./main”

VI. Result Screenshot & Explanation



```
1 1
4 4 4 4 4 4
2 2 2
5 5 5 5
3 3
6 6 6
```

```
sophotky@lion thread-scheduling $ ls
Queue.c          input.txt        main.c           output.txt
README.md        main             main.exe         test.c
sophotky@lion thread-scheduling $ gcc -pthread main.c -o main | ./main
SCEDULING PROCESSES...
1 1
4 4 4 4 4 4
2 2 2
5 5 5 5
3 3
6 6 6
ALL PROCESS HAS BEEN SCHEDULED
sophotky@lion thread-scheduling $
```

From the input, process ID 1 and 4 are in Queue_1, the two will have to be selected first. Also, in Queue_1 I used FCFS and process ID 1 comes in first so it is selected first. And the last process (4) in Queue_1 is selected leaving Queue_1 empty.

Queue_2 has processes ID 2 and 5. The priority value of process 2 is smaller than process 5 meaning process 2 has more priority than process 5. Therefore, process 2 is selected first and the last process (5) in Queue_2 is selected leaving Queue_2 empty.

Queue_3 has processes ID 3 and 6. The time value of process 3 is smaller than process 6 meaning that process 3 has more priority than process 6. Therefore, process 3 is selected first. And the last process (6) is selected.

VII. Conclusion

This assignment gives me a good and detailed understanding about CPU scheduling. I also learned a lot about:

- how to create a thread using POSIX library via pthread_create().
- How to wait for threads to finish its task via pthread_join().
- How to terminate a thread via pthread_exit().