MP5 Design

Tsao Yuan Chang 927000791

I implemented the FIFO scheduler in MP5, but I didn't implement the bonus options. I'll cover the modification I did in scheduler.H, scheduler.C, thread.H and thread.C in this document.

scheduler.H

In scheduler.H, first I defined the class Queue to handle the ready queue, which is basically a linked list. It has functions such as enqueue and dequeue, which will be implemented in scheduler.C. I also added the size variable to keep track of the ready_queue's size, and I define the ready_queue in this file so I don't need to 'new' the ready_queue in scheduler.C.

```
class Queue{
private:
    Thread* thread;
    Queue* next;
public:
    Queue();
    Queue(Thread* t);
    Queue(Queue& q);
    void enqueue(Thread* t);
    Thread* dequeue();

};
```

scheduler.C

1.implementing the Queue class

I use a linked list ready_queue for thread scheduling. The enqueue function adds the thread to the end of the ready queue. The dequeue function removes the first thread and returns it.

2. constructor

I cannot write something like 'Queue* ready_queue = new Queue();' since we can't access the 'new' operator here. So I defined the ready_queue in scheduler.H instead. The only thing I do in the constructor is to set the size of ready_queue to 0.

3.yield()

In this function, I find the first thread in the ready queue, which is supposed to run next, and then dispatch to this thread.

4.resume() and add()

The implementation of these 2 functions are the same. I put the thread into the ready queue, and increase the size of ready_queue by one.

5. terminate()

In this function, I iterate through ready_queue to find the terminating thread. I dequeue each of the thread and save it into the variable cur. If cur is not the terminating thread, I'll use enqueue to put this thread back. Otherwise, I won't put the thread back and the ready_queue's size will decrease by 1.

thread.H

There’s no modification in this file.

thread.C

1. thread_start()

I enable the interrupts in this function. So whenever a thread is created, the interrupt will be enabled.

2. thread_shutdown()

In this function, we need to make the thread give up their resources. So first I extern the SYSTEM_SCHEDULER to deal with the threads. I use the scheduler to terminate the current thread, and then delete the thread, finally yield the CPU to other threads. After implementing this function, I can uncomment the _TERMINATING_FUNCTIONS_ macro in kernel.C and terminate the threads appropriately.

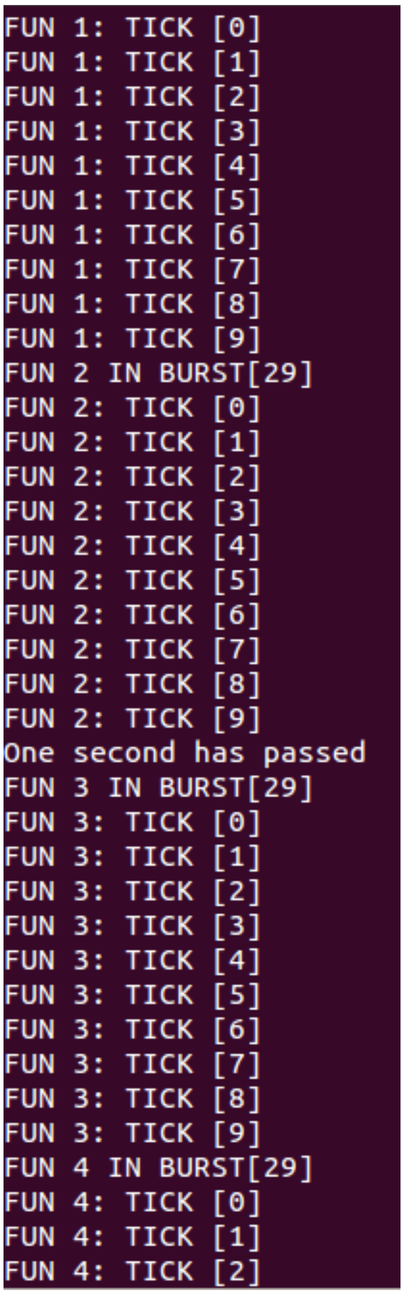
```
static void thread_shutdown() {
    /* This function should be called when the thread returns from the thread function.
       It terminates the thread by releasing memory and any other resources held by the thread.
       This is a bit complicated because the thread termination interacts with the scheduler.
     */

    SYSTEM_SCHEDULER->terminate(Thread::CurrentThread());//remove current thread from scheduler
    delete current_thread;//deletes current thread
    SYSTEM_SCHEDULER->yield();//give cpu to other threads
    /* Let's not worry about it for now.
       This means that we should have non-terminating thread functions.
    */
}
```

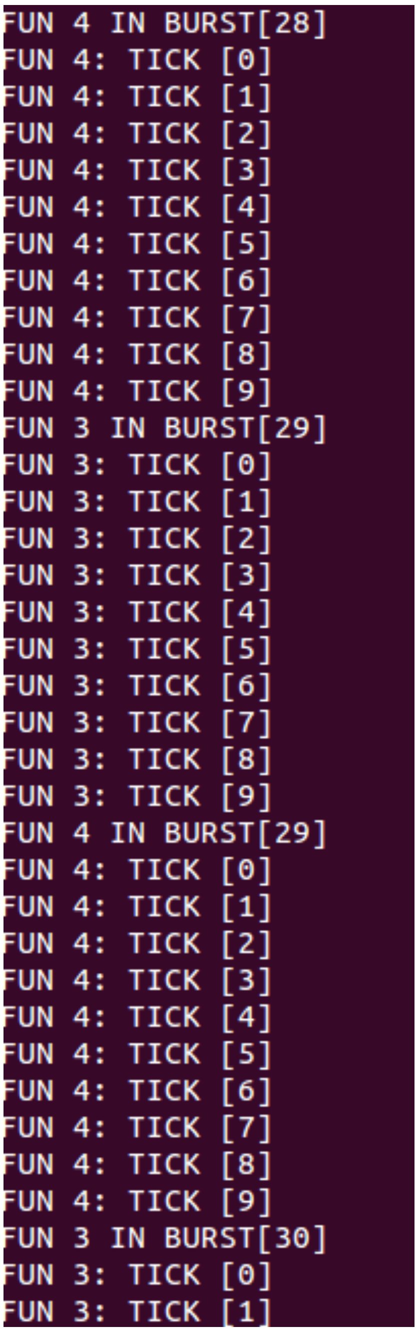
(final testing result is on the next page)

Final result:

The left one is the log before uncommenting the _TERMINATING_FUNCTIONS_ macro, so you can see thread 1 ~ thread 4 runs in turn infinitely. After uncommenting the macro, only thread 3 and thread 4 are running after a short period of time since thread 1 and thread 2 are terminated after 10 iterations.



```
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
FUN 1: TICK [3]
FUN 1: TICK [4]
FUN 1: TICK [5]
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
FUN 2 IN BURST[29]
FUN 2: TICK [0]
FUN 2: TICK [1]
FUN 2: TICK [2]
FUN 2: TICK [3]
FUN 2: TICK [4]
FUN 2: TICK [5]
FUN 2: TICK [6]
FUN 2: TICK [7]
FUN 2: TICK [8]
FUN 2: TICK [9]
One second has passed
FUN 3 IN BURST[29]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[29]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
```



```
FUN 4 IN BURST[28]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 3 IN BURST[29]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[29]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 3 IN BURST[30]
FUN 3: TICK [0]
FUN 3: TICK [1]
```