Machine Problem 5: Kernel-Level Thread Scheduling

Design :

In this programming assignment, we implement the scheduling of multiple kernel-level threads. Firstly, we design a very basic FIFO scheduler for kernel level threads. To implement this functionality, we create a simple class "Queue" that has this property of FIFO queue. It will maintain a list of Thread structures and the next pointer for thread. This class provides two basic queue operations which are enqueue and dequeue. For enqueuing a thread object in the ready queue, we just check if there is an existing member in the queue already or not, if not we update the next pointer. For dequeue operation, we take the top thread and return.

The scheduler class declares a queue object and also has a size member. The yield api will first check if there is any thread available to switch and then dequeue the thread from queue. The resume and add in the scheduler are very similar where we enqueue the given thread to the queue. The terminate api loops through the queue and finds the thread with same thread id and then delete it.

Bonus parts implemented : Option 1 and 2

Option 1 : We call the enable interrupts api while we start the thread.

Option 2 : Round-robin scheduling

We generate a timer based interrupt at every 50 ms to allow make the running thread yield. To make this work, we update the dispatch interrupt api to allow EOI signal after the interrupt has occurred and then in the handle interrupt api, we do a yield to next thread.

Functions used in scheduler.C/H :

The definition of the data structure that we are using here :

```
classQueue
{

                      private:
```

```
        Thread * thread;
        Queue  * next;

    public:

        Queue() {
            thread   = NULL;
            next     = NULL;
        }

        Queue(Thread * new_thread ){
            thread = new_thread;
            next   = NULL;
        }

        void enqueue (Thread * new_thread) {
            if (thread == NULL) {
                thread = new_thread;
            } else  {
               if (next == NULL) {
                   next = new Queue(new_thread); //add at the end
               } else {
                   next->enqueue(new_thread);
               }
            }
        }

        Thread *dequeue() {
            if (!thread)
                return NULL;

            Thread * first = thread;

            if (next) {
                thread               =  next->thread;
                Queue * temp         =  next;
                next                 =  next->next;
                delete temp;
           } else {
                thread = NULL;
           }

           return first;
        }
```

- **Yield**

  ```
  size--;
  Thread* cur_thread = queue.dequeue();
  Thread::dispatch_to(cur_thread); //run this thread
  ```

- **Resume and add**

  ```
  queue.enqueue(_thread);
  size++;
  ```

- **Terminate**

  We use a for loop to traverse the list of threads and dequeue the thread that is passed to the function.

In thread.C we enable interrupts in `thread_start() - Machine::enable_interrupts();`

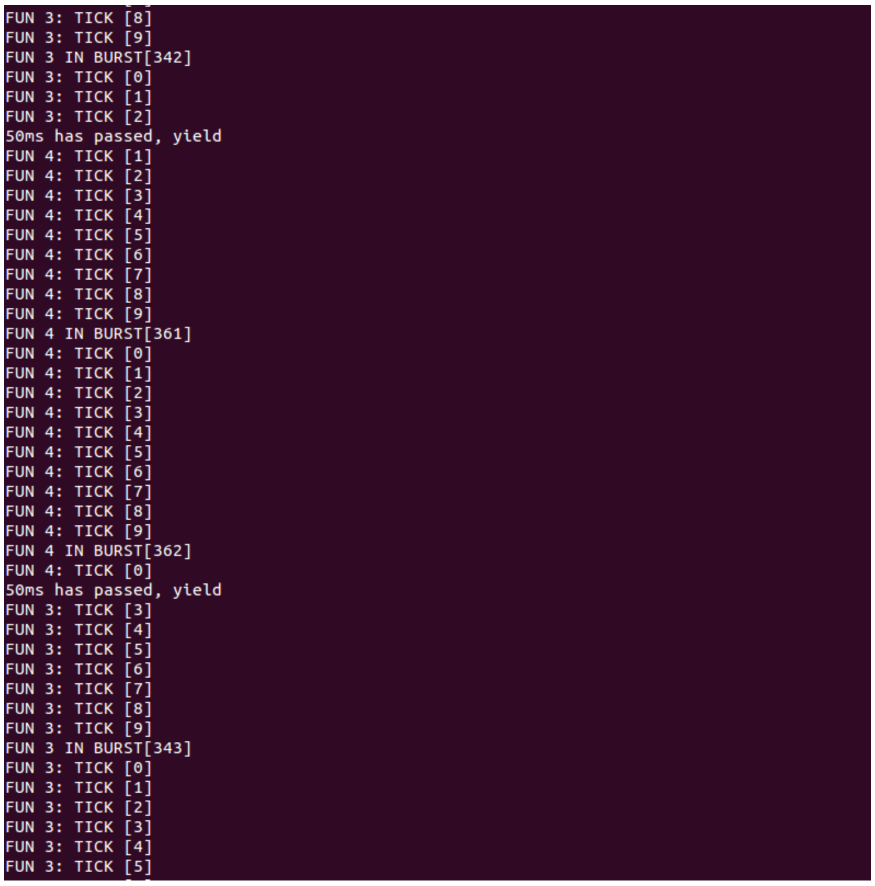
To implement the Bonus option 2, we make changes to simple_timer.C so that we get interrupts every 50ms.

```
if (ticks >= hz/20 ) { //interrupts every 50 ms
        Console::puts("50ms has passed, yield\n");
        SYSTEM_SCHEDULER->resume(Thread::CurrentThread());
        SYSTEM_SCHEDULER->yield();
}
```

Testing :

To test the basic functionality of scheduler, we first define the macro _USES_SCHEDULER_ in kernel.C which will start calling the new scheduler. We also define the macro _TERMINATING_FUNCTIONS_ for testing the terminating part.

The screenshot of the test for Round-robin functionality is posted below :



```
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 3 IN BURST[342]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
50ms has passed, yield
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 4 IN BURST[361]
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 4 IN BURST[362]
FUN 4: TICK [0]
50ms has passed, yield
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 3 IN BURST[343]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
```

The following files have been modified for this MP :

- Scheduler.C/H
- Thread.C
- Simple_timer.C/H
- Interrupts.C
- Kernel.C (mainly for testing)
- Console.C (for printing logs into console)
- Boschsrc.bxrx (for printing logs into console)