Design Document - Operating Systems - Anshul Sharma

==

In this machine problem we tackle the following problems:

1. We implement a first in first out(FIFO) scheduler to schedule the non-terminating threads.
2. Modify FIFO scheduler to schedule and handle(terminate) the terminating threads.
3. [Bonus Option 1] Enable and Disable Interrupts
4. [Bonus Option 2] We implement a Round Robin scheduler

To handle these cases we implement the following:
In scheduler.c:

1. Scheduler::add() - responsible for adding the thread to the ready queue.
2. Scheduler::yield() - yields the thread in the queue, then dispatches it to the thread dispatcher for further execution.
3. Scheduler::resume() - resumes execution of the dispatched thread.

We also implement the ready queue using a new queue.H class. We implement the following functions in that class:

1. enqueue() - adds the thread at the end of the ready queue.
2. deque() - removes the thread from the top of the ready queue.

We also implement the following function in the Scheduler.C to handle the terminating threads:
Scheduler::terminate() - searches the thread in the ready queue and uses the thread id and is removed from the ready queue once found.

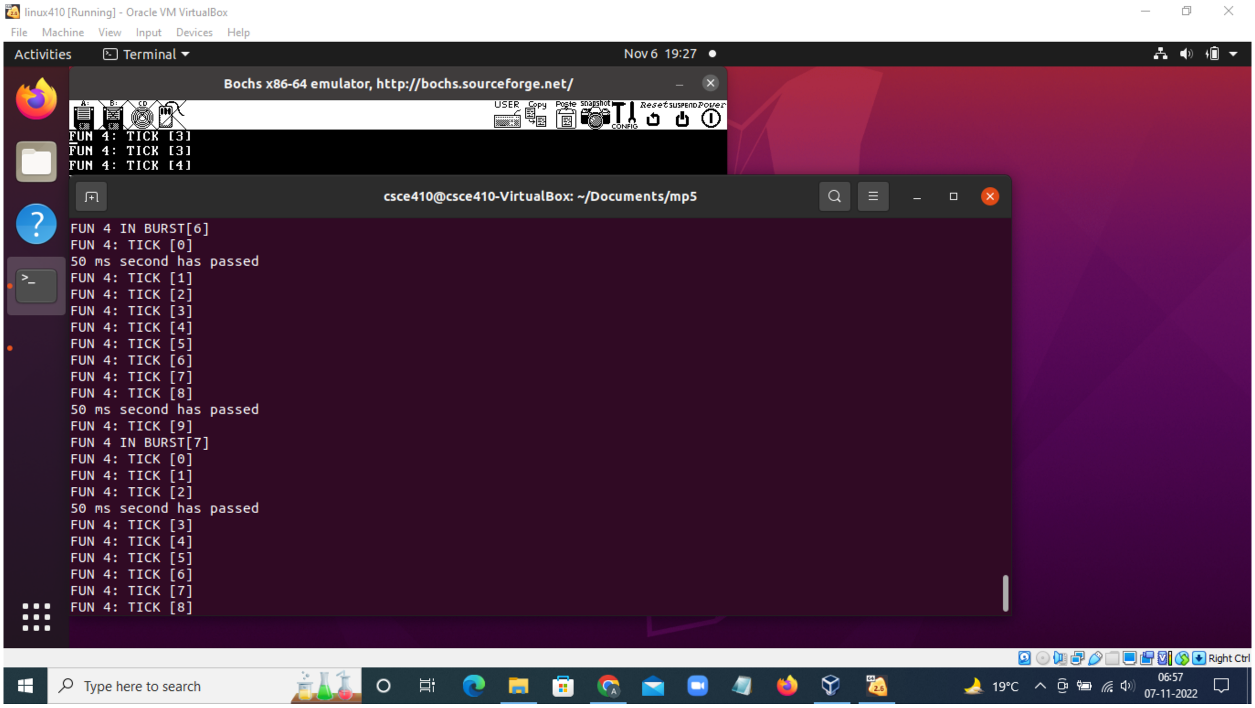
We implement the enabling and disabling of interrupts using changes in thread.C and scheduler.c

Finally the second bonus option of implementing the round-robin scheduler is by creating a round-robin scheduler class that inherits from the scheduler and the interrupt handler class and uses the already implemented ready queue functionality. In this class, we are using the time quanta of 50ms to implement the RR scheduling. Once the time quanta of the currently executing thread run out, it is pooped and pushed back into the end of the ready queue. Here are some functionalities of the RR scheduler class.

Note: To run the RR scheduler functionality, please **uncommen**t the following line(line# 42)
#define _RR_SCHEDULER_

1. handle_interrupt(): if the timer reaches the 50ms value we put the currently running thread at the end of the ready queue and start the execution of the next thread in the ready queue and reset the timer to 0.
2. yield(): ensures that 50ms of time quanta is given to each executing thread and also resets the ticks count.

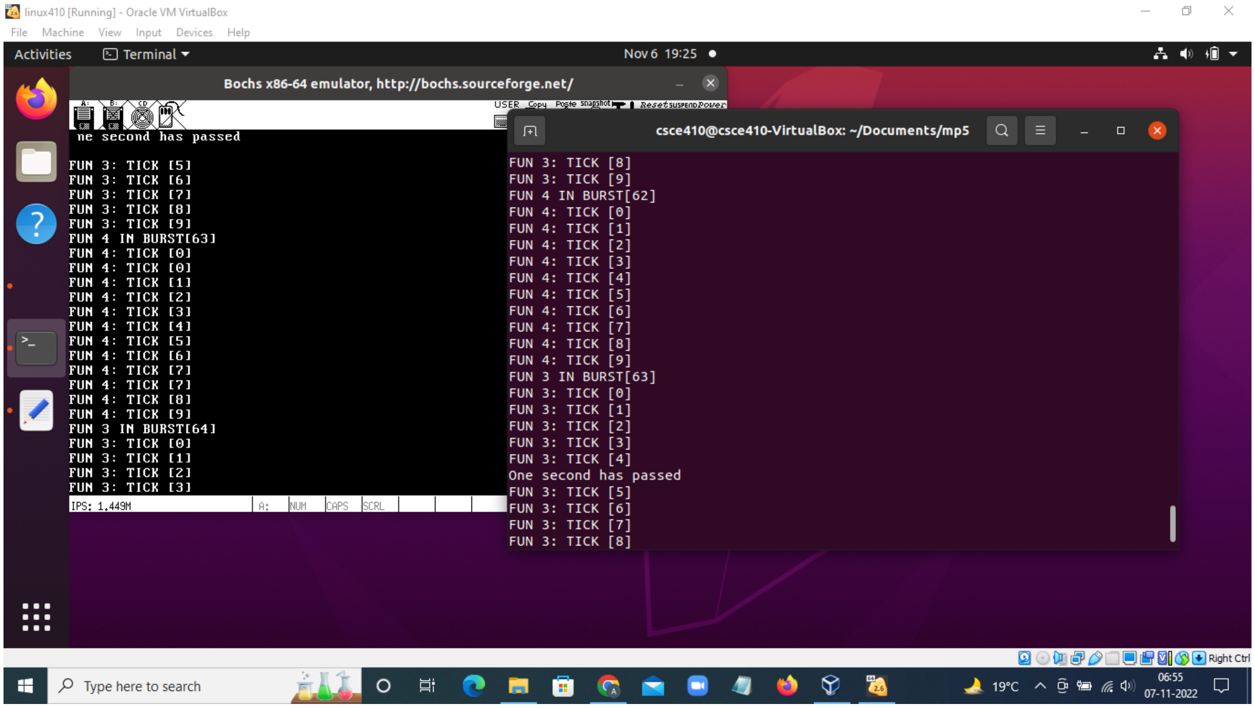
Here’s the screenshot of normal FIFO scheduling:


linux410 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Nov 6 19:27
Bochs x86-64 emulator, http://bochs.sourceforge.net/
csce410@csce410-VirtualBox: ~/Documents/mp5
Type here to search
19°C
06:57
07-11-2022
Right Ctrl

```
FUN 4: TICK [3]
FUN 4: TICK [3]
FUN 4: TICK [4]
```

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

Here’s the screenshot of RR Scheduling:


linux410 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Nov 6 19:25
Bochs x86-64 emulator, http://bochs.sourceforge.net/
csce410@csce410-VirtualBox: ~/Documents/mp5
IPS: 1.449M
A:
NUM
CAPS
SCRL
Type here to search
19°C
06:55
07-11-2022
Right Ctrl

```
 ne second has passed

FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[63]
FUN 4: TICK [0]
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 [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 3 IN BURST[64]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
```

```
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[62]
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[63]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
One second has passed
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
```