`

For Testing FIFO scheduler,

1) Please uncomment the normal SYSTEM_SCHEDULER= new Scheduler() in kernel.C and comment the SYSTEM_SCHEDULER= new RRScheduler(50)
2) Also please uncomment the **if(ticks>= hz)** in handle_interrupt(REGS* r) for in simpleTimer.c to test FIFO scheduler and comment **modified if case** for RR scheduler

The github link is https://github.tamu.edu/kausht14/OS_MP5

Design Steps:

1)Everytime a thread is added, yielded, resumed, interrupts are checked and properly handled.

2)A FIFO queue is implemented as a linkedlist and new threads are added and deleted as they come and go

3)In Kernel.C operator delete is overloaded to handle the threads and linker error has gone after overloading delete

4) All the modifications to .C /.H files have been clearly mentioned and results have been attached.

For Thread.H

1) Included scheduler. H and extern Scheduler * Scheduler is declared to make use of Scheduler initialised in Kernel.c

For thread.C

2) The thread shutdown function and thread_start function are properly handled by making use of extern Scheduler declared in .H file

For Scheduler.H

3) A data structure to store the queue of threads is declared and also derived RRscheduler from Scheduler.

For Scheduler.C

4) First thread.H is included , the functions have been properly handled by maintaining the links appropriately when a thread yields the CPU.
5) Also in Kernel. C uncommented uses scheduler and uses terminating functions

For SimpleTimer.C

6) scheduler.H, extern Scheduler* SYSTEM_SCHEDULER ,thread.H and the function handle_interrupts is modified to reflect for quantum elapse and switching to next thread in RR fashion

For kernel.c

7) The delete operator is overloaded to delete threads that have yieded CPU
8) Also please comment out the lines 265-268

```
//SYSTEM_SCHEDULER=new
Scheduler();
                                // The following is for the use of Round Robin
                        Scheduler/
                                // Comment it to test the functionality of FIFO
                        scheduler and uncomment the FIFO scheduler
                                SYSTEM_SCHEDULER = new RRScheduler(50);
```

9) Also in Kernel.C the terminating functions have been used

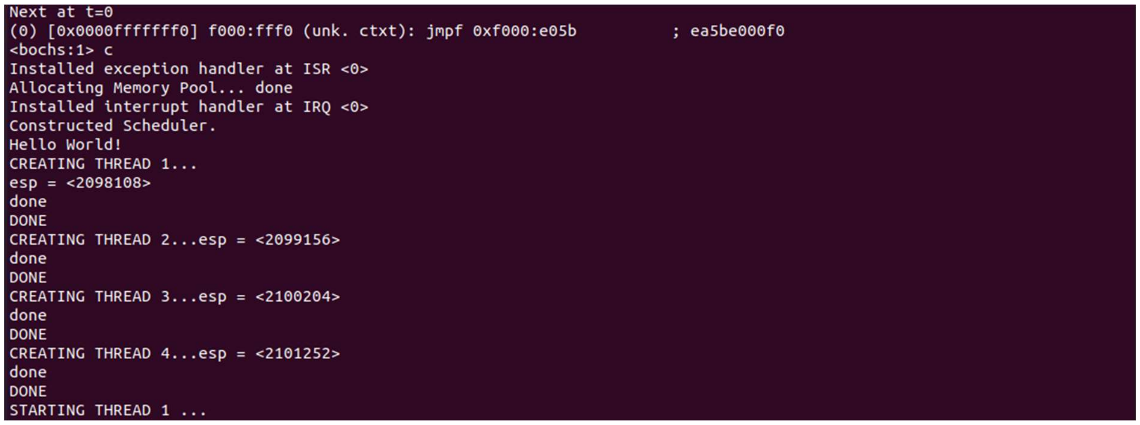
10) The console and bochs files for redirecting output to console have been used and the modifications were made as directed by the Professor during MP3

Bonus Sections

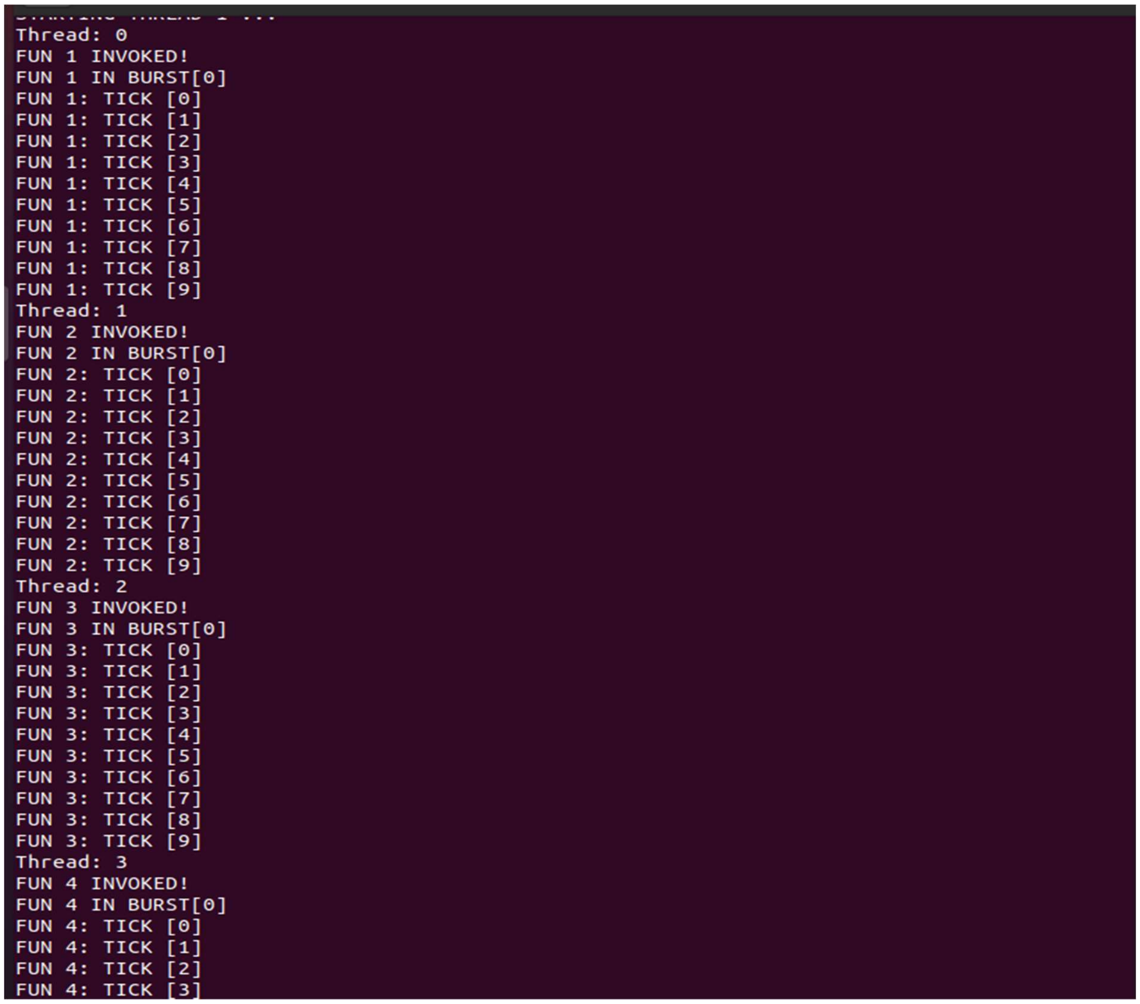
1) **Correct Handling of Interrupts:**
   a) Interrupts are properly enabled and disabled by proper conditional statements and all the test cases have been attached as screenshots that describe the proper handling of interrupts
   b) Also the interrupts are checked everytime a new thread is scheduled, thereby making sure that the interruption is handled correctly

2) **Handling RRScheduler:**
   a) In kernel.c the normalFIFO scheduler is commented and new RRscheduler is called by giving a EOQ value of 50ms
   b) The class is derived from the Scheduler and also the simple Timer and interrupts for every quantum elapse are handled similar to the one second elapse as given in handout by minor modifications to reflect for the 50ms switching.
   c) The functionality is also supported by minor modifications to handle_interrupts functionality in SimpleTimer.C by making minor modifications by checking for 50ms elapse and switching to a new thread in queue in RR fashion.
   d) The screenshots with and without terminating functions for RR scheduler have been attached.

Initialisation with FIFO scheduler and terminating functions

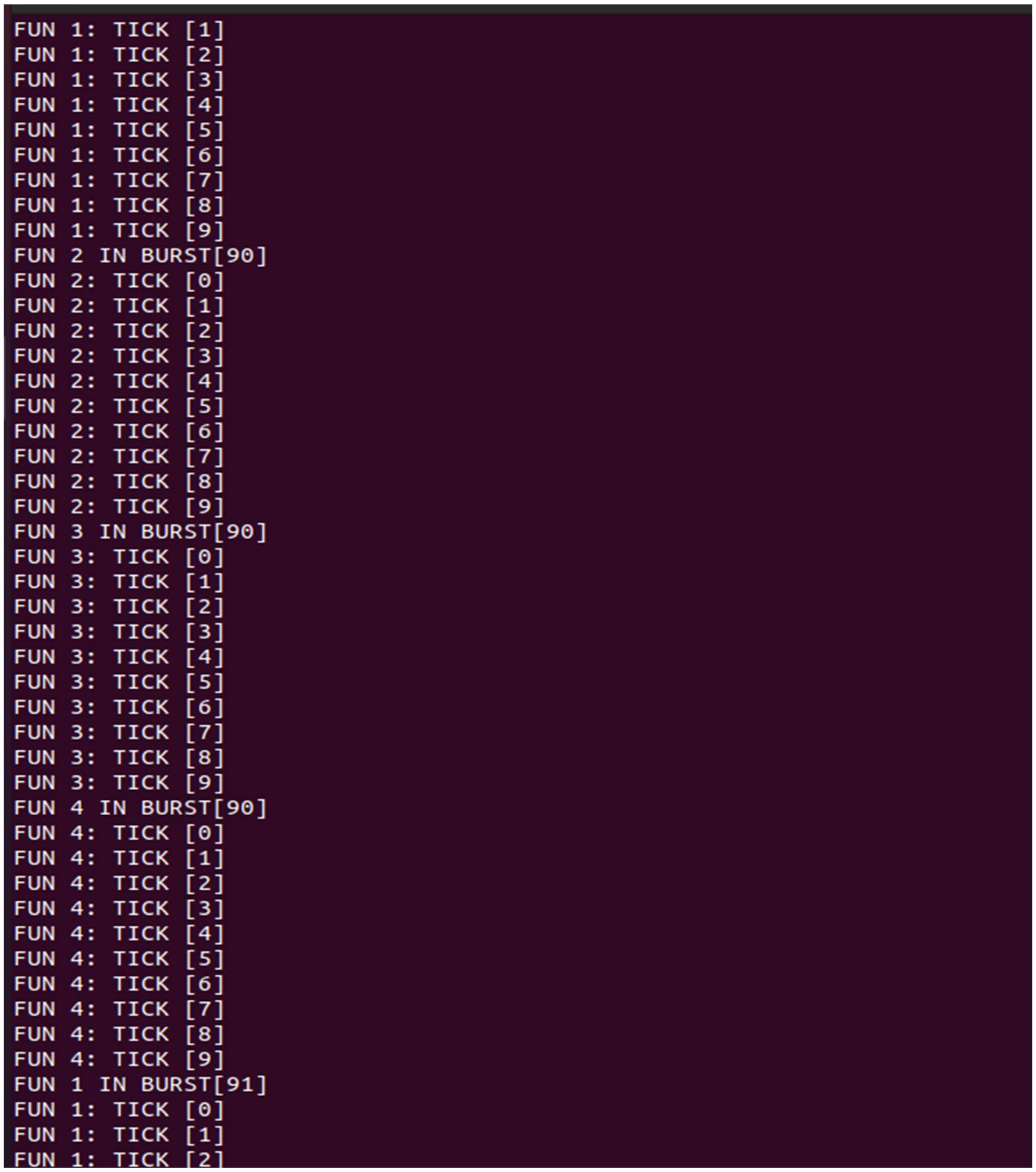


```
Next at t=0
(0) [0x0000fffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b          ; ea5be000f0
<bochs:1> c
Installed exception handler at ISR <0>
Allocating Memory Pool... done
Installed interrupt handler at IRQ <0>
Constructed Scheduler.
Hello World!
CREATING THREAD 1...
esp = <2098108>
done
DONE
CREATING THREAD 2...esp = <2099156>
done
DONE
CREATING THREAD 3...esp = <2100204>
done
DONE
CREATING THREAD 4...esp = <2101252>
done
DONE
STARTING THREAD 1 ...
```



```
Thread: 0
FUN 1 INVOKED!
FUN 1 IN BURST[0]
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]
Thread: 1
FUN 2 INVOKED!
FUN 2 IN BURST[0]
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]
Thread: 2
FUN 3 INVOKED!
FUN 3 IN BURST[0]
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]
Thread: 3
FUN 4 INVOKED!
FUN 4 IN BURST[0]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
```

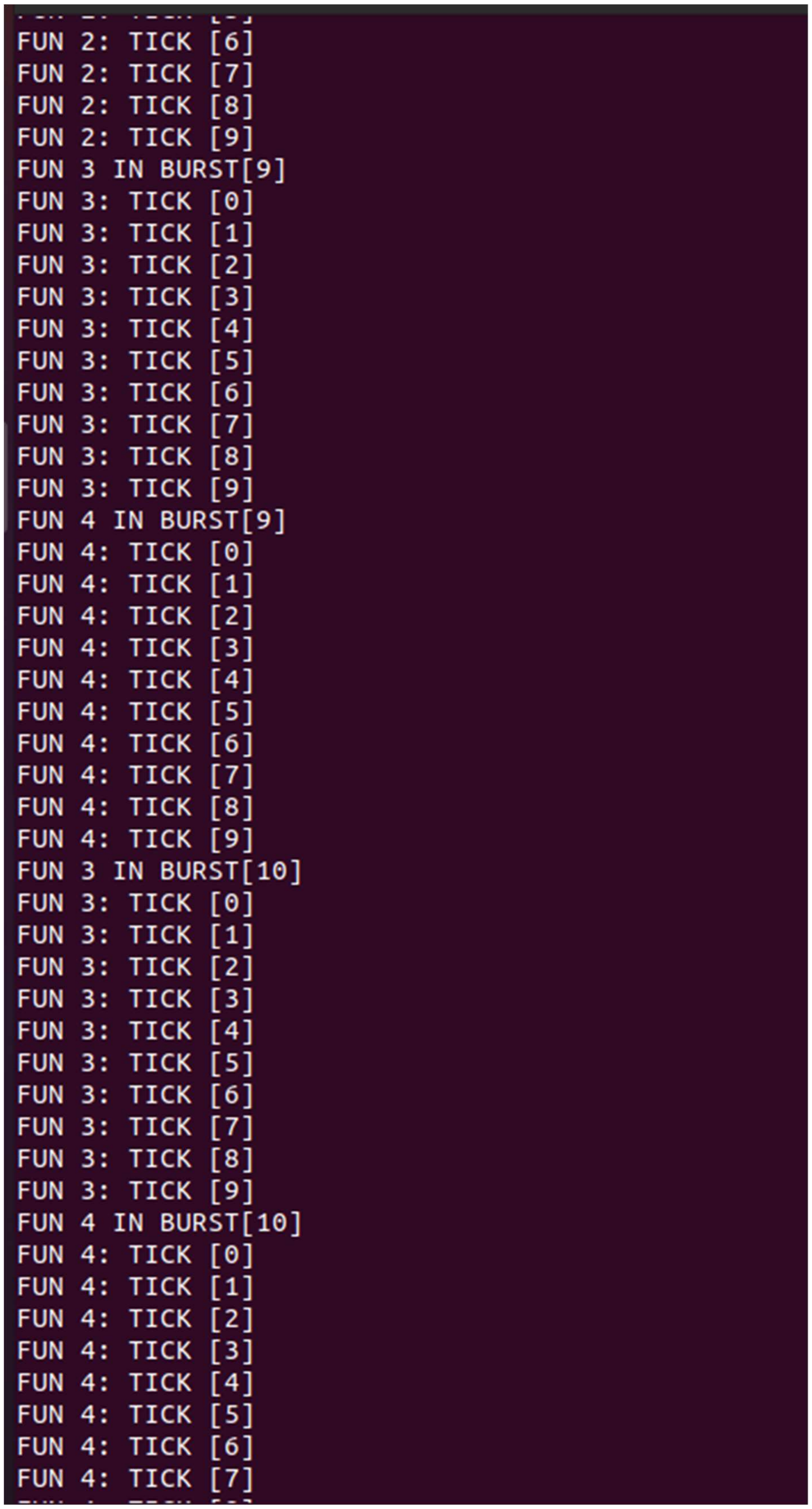
Using FIFO Scheduler without terminating functions



```
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[90]
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]
FUN 3 IN BURST[90]
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[90]
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 1 IN BURST[91]
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
```

With terminating functions:

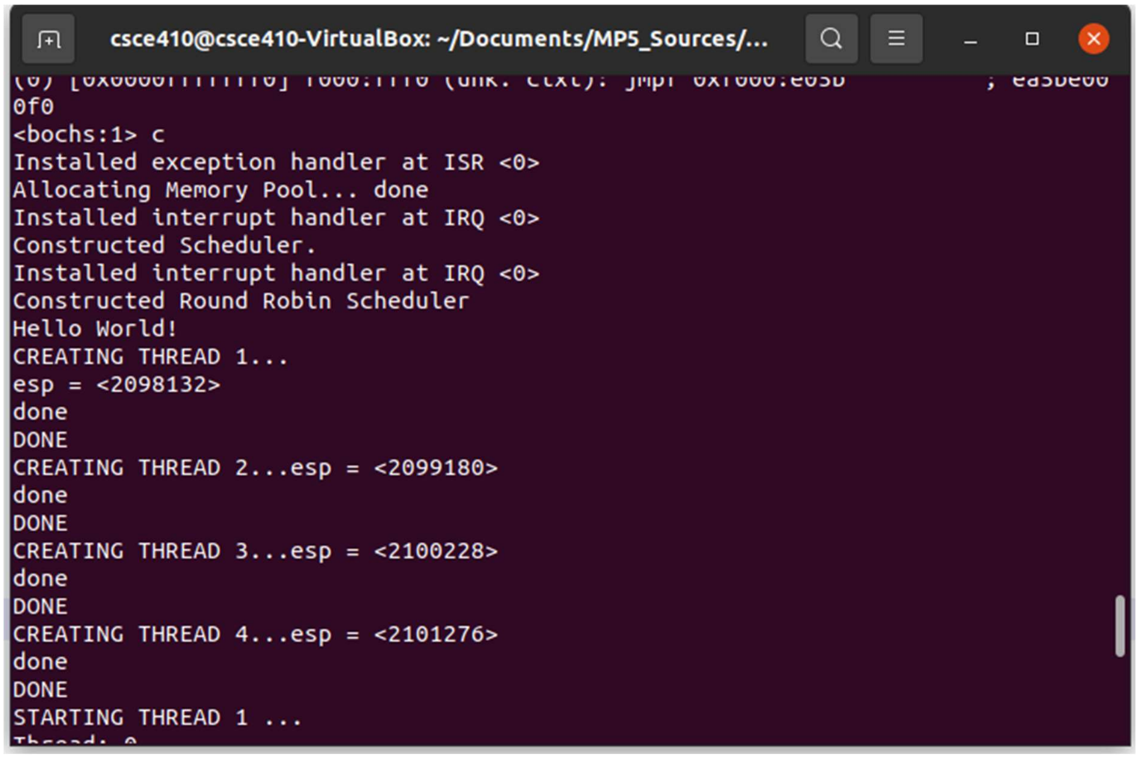
After Burst 9 thread 1 and thread 2 terminates as per the code



```
FUN 2: TICK [6]
FUN 2: TICK [7]
FUN 2: TICK [8]
FUN 2: TICK [9]
FUN 3 IN BURST[9]
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[9]
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[10]
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[10]
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]
```

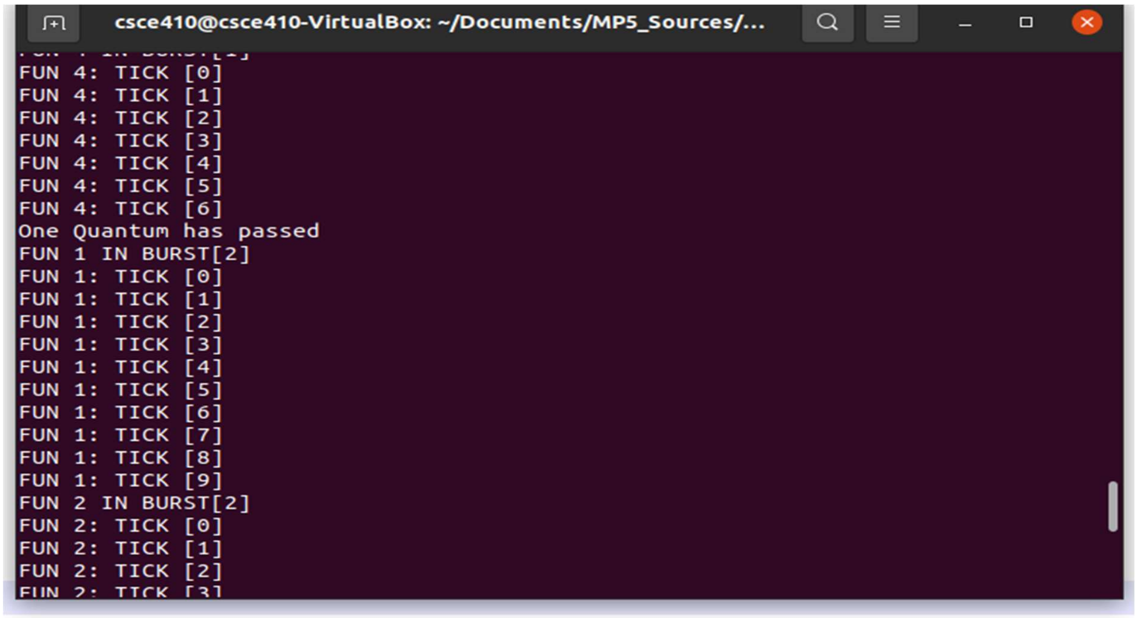
RR Scheduling:

1) Initialiastion of RR schdeuluer and register timer interrupt



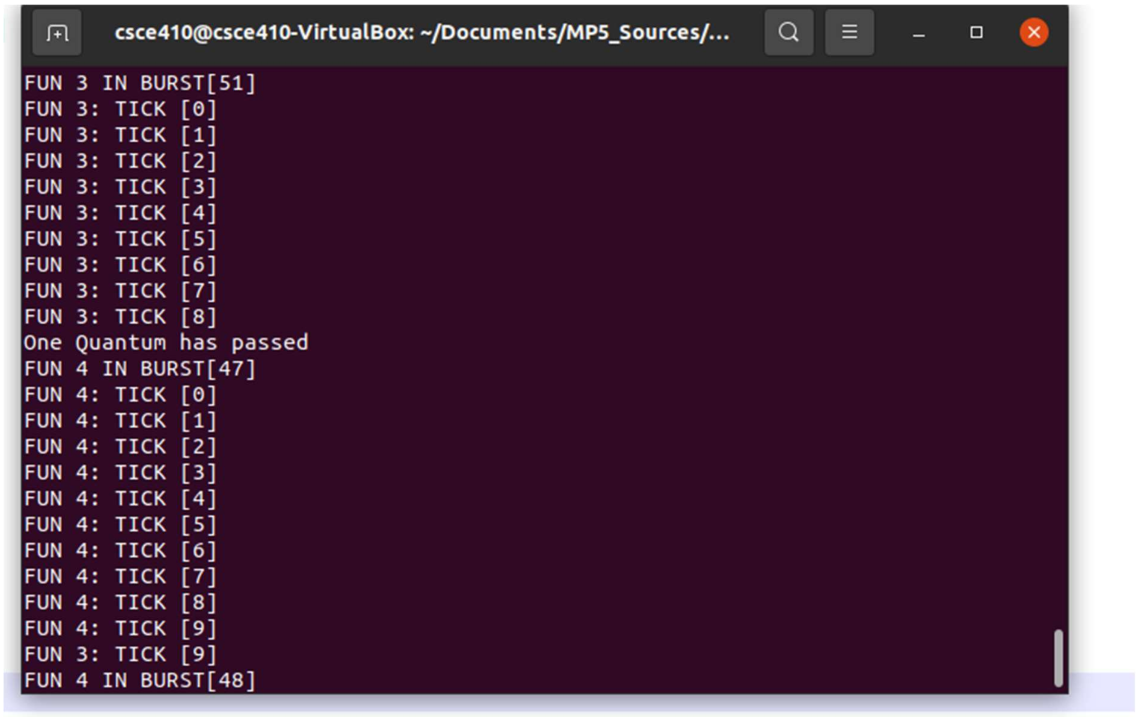
```
(0) [0x0000fffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b          ; ea5be00
0f0
<bochs:1> c
Installed exception handler at ISR <0>
Allocating Memory Pool... done
Installed interrupt handler at IRQ <0>
Constructed Scheduler.
Installed interrupt handler at IRQ <0>
Constructed Round Robin Scheduler
Hello World!
CREATING THREAD 1...
esp = <2098132>
done
DONE
CREATING THREAD 2...esp = <2099180>
done
DONE
CREATING THREAD 3...esp = <2100228>
done
DONE
CREATING THREAD 4...esp = <2101276>
done
DONE
STARTING THREAD 1 ...
Thread: 0
```

2)



```
FUN 4 IN BURST[1]
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]
One Quantum has passed
FUN 1 IN BURST[2]
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[2]
FUN 2: TICK [0]
FUN 2: TICK [1]
FUN 2: TICK [2]
FUN 2: TICK [3]
```

2) In the below screenshot, When one quantum has passed , the thread 3 is added to the end of the queue and the CPU is given to next thread in queue. From the screenshot we can see when the thread has completed its execution then another thread executes in that quantum which ensures that yielding is proper and after the quantum has expired we can see that thread 3 again executes and yields the CPU.



```
FUN 3 IN BURST[51]
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]
One Quantum has passed
FUN 4 IN BURST[47]
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: TICK [9]
FUN 4 IN BURST[48]
```