MACHINE PROBLEM 5

OBJECTIVE - To add scheduling of multiple kernel-level threads to existing code base.

OVERVIEW -
We need to implement the below functions in scheduler.C

1) Scheduler () -> constructor for class Scheduler.
2) yield () -> Called by the currently running thread in order to give up the CPU.
3) resume (Thread * _thread) ->Add the given thread to the ready queue of the scheduler.
4) add (Thread * _thread) ->Add the given thread to the ready queue of the scheduler.
5) terminate (Thread * _thread) ->Remove the given thread from the scheduler.

We also need to implement 2 functions in thread.C and add a function in thread.C
1) thread_shutdown() - > terminates a thread
2) thread_start() -> just enable interrupts -> for bonus question 1
3) EOQ() -> new function called at end of quantum for bonus question 2

Added files -
1) queue.H-> contains basic implementation of FIFO queue with enqueue and dequeue function.

Other Modified files -
1) thread.H-> added static function EOQ.
2) scheduler.H -> declared private members(thread_queue and queue_size).
3) makefile -> added queue.H in makefile.
4) simple_timer.C-> changed the function handle_interrupt to generate a periodic interrupt after a quantum has passed.
5) simple_timer.H-> added a private variable.
6) interrupts.C -> dispatch_interrupt function was modified to send EOI signal before calling the handle interrupt function which inturn calls the EOQ function. This required for bonus 2 question.
7) kernel.C - pass_on_CPU function was commented to allow the round robin scheduler to work.

IMPLEMENTATION -
Bonus 1 and Bonus 2 questions are also implemented.
Changes in scheduler.C -

Scheduler() - queue_size is initialized to zero.

yield() - if queue_size is non-zero, new thread is dispatched from the FIFO queue.

resume() - thread is added to the ready queue.

add() - thread is added to the ready queue.

terminate() - the thread passed as argument is searched in the fifo queue and removed from the queue.

Changes in thread.C -

thread_shutdown() - terminates the thread using schedulers terminate function and then frees up the memory by deleting the thread, then it calls the scheduler's yield function to dispatch the next thread.

thread_start() - enabled interrupts using Machine::enable_interrupts() function for bonus question 1.

EOQ() - this function is called at the end of the quantum, this function calls the resume function with the current thread as an argument to add the current thread to the ready queue and then calls the yield function to dispatch the next thread. This function is periodically called from the simple_timer.C and causes the scheduler to switch jobs after every quantum in round robin fashion for bonus question 2.

Changes in simple_timer.C -

handle_interrupt() - defined a new variable fifty_ms, which is incremented every 50ms. Changed the if condition so that the EOQ function is called every 50ms.

Changes in interrupts.C -

dispatch_interrupt() - modified to send EOI signal before calling the handle interrupt function which inturn calls the EOQ function. This is required for bonus 2 question.

TESTING - testing is done using kernel.C, when 1 quantum is passed thread is switched
As shown in below snapshots

```
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 1 IN BURST[2]

*************************1 quantum has passed*********************
FUN 2 IN BURST[2]
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]

*************************1 quantum has passed*********************
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 3 IN BURST[3]
FUN 3: TICK [0]

*************************1 quantum has passed*********************
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
```