Design Document:

Machine Problem 5: Kernel-Level Thread Scheduling

Bonus Attempted: Bonus 1

Files Modified/ added:

- thread.c
- Kernel.c
- Scheduler.c
- Scheduler.h
- queue.h
- queueNode.h

The Document is divided into the following sub parts:
Part1: FIFO Scheduler with non terminating functions.
Part2: FIFO Scheduler with terminating functions.
Part3: Bonus1, enabling interrupts.

Part1:

We need to implement a queue to use it as a ready queue for the scheduler. The ready queue contains all the threads which are ready. When the currently running thread is preempted, the front of the queue is sent to the CPU and the currently running thread is made to yield the CPU and it is added at the end of the queue.

Queue Implementation: A queue is implemented using a linked list in the background. The queueNode.h file defines a single node in the linked list. The queue.h file defines the queue class by making use of the queueNode class. The functions supported are

- front(): Which returns the next ready element in the queue
- pop(): It removes the first ready element in the queue;
- push(): Adds a new element at the end of the queue.
- remove(): removes the given element in the queue if it is present.
- size(): prints out the current size of the queue. (This was used for Debugging purposes only).

Scheduler implementation: The functions defined in the scheduler.h are implemented in the scheduler.c file. We create a readyQueue which is implemented by the queue we defined before. The functions implemented are:

- add(): This function takes a thread and adds it to the ready queue.
- yield(): This function gets the first thread in the queue, pops it and then swaps it with the running thread. That is, it performs a context switch through the dispatcher function.
- resume(): It adds back a thread which is now ready coming back from a not ready state.
- terminate(): This function finds the thread in the queue and deletes it.

Testing: The kernel.c file should be edited to enable scheduler. All the 4 threads run infinitely.

Part2:

With Terminating Functions:
Now we try to make the first and second threads terminate. For this to happen without any problem, we need to properly write the thread shutdown function. First we terminate the thread which deletes any kind of presence the thread might have in the queue and then we delete the thread and yield the cpu.

The function changed for this is thread_shutdown() in thread.c file.

Testing: The kernel.c file should be edited to enable terminationg. The first 2 threads terminate after 10 bursts and the remaining 2 threads run infinitely.

Part3:

Interrupt handling:
Initially all the interrupts are diabled by default. And we want the interrupts to be in disabled state during the creation of the thread. But the interrupts should be enabled at a later point of time. The thread stack runs the following functions in order; Thread_start, Thread_function and finally thread_shutdown. We need to enable the interrupts when the thread starts. So we edit the thread_start function to enable the interrupts.

We also need to make sure that the interrupt doesn't mess with our ready queue. To achieve this we disable the interrupts before we make any changes to our

ready queue. So for all the functions in the scheduler we disable the interrupts before queue operations and re-enable them after the queue operations are done.

Testing: We can now see “1 second has passed” prompt in the output.