Hung-Hsueh Shih (UIN: 132006330) CSCE611 Operating System
Machine Problem 5 Kernel-Level Thread Scheduling

Introduction:
In this machine problem I will add scheduling of multiple kernel-level threads to my code base. The emphasis is on scheduling. I implement RR scheduler and FIFO scheduler with interrupt handling(Option 1 and Option 2)

I have modified :

Scheduler.C

Scheduler.H

Thread.C

Simple_timer.C

Kernel.C

For FIFO scheduler

Scheduler.H:

I declare a struct “queue” which includes a thread pointer and a pointer point to next queue object for my scheduler class.

Scheduler.C:

1. Constructor

In my constructor I have a head pointer points to the first item in my ready queue and a tail pointer which points to the last item in my ready queue. I initialize both pointer to NULL which means the ready queue is empty. Besides, I have a pointer point to current running thread.

2. Yield

If ready queue is not empty, I remove first item in ready queue and update current running thread to the first thread in ready queue. In the end dispatch the next thread to cpu.

3.Resume

I simply call enqueue function to add thread into the end of my ready queue

4. Add

I simple call enqueue function to add thread into the end of ready queue.

5. Terminate
First of all, I check whether the thread I'm going to remove is currently running in cpu, if it is, I simply call yield function. If it's not, means the thread we want to remove is inside the ready queue. Therefore, I travers the ready queue and remove the thread.

6. Enqueue

If the head or tails pointers are NULL means the ready queue is empty. I simply add a need queue object into ready queue. If the ready queue is not empty, I simply add the new queue object into the end of ready queue.

Thread.C

1. thread_shutdown

I disable interrupt and call the terminate function remove thread.

2. thread_start

I enable interrupts.

Bonus:

Option1

For correct handling interrupts, in yield function I check the interrupts first. If interrupts enabled at the beginning of yield function I disable it. If the interrupts are disabled in the end of yield function I enable it.

For the resume and add function, I do the same thing.

Option2 : implement RR scheduler

In the scheduler.C I also declare RRScheduler class which is similar to FIFO scheduler except for the RR scheduler constructor I need an argument for simple timer to set up time quantum

In the simple timer, I modify the handle_interrupt to make sure that when the time expire, if the thread has not finished, it will resume itself to the end of ready queue and give up cpu by calling yield.

In the interrupts.C I modify dispatch_interrupt() to let the interrupt controller (PIC) know that the interrupt has been handled. Since the interrupt has been handled before calling handle interrupt.

In the kernel.C I change the FIFO scheduler to RR scheduler which I declare in scheduler.C with 50 msec. Besides, because I'm using RR scheduler I do not need to manually dispatch thread in my thread function. Thus, I uncomment the dispatch in each thread function. In the end, I uncomment _USES_SCHEDULER_ and _TERMINATING_FUNCTIONS_ to use RR scheduler and let thread able to terminate.