OS611_MP5_Report

Name: Wei-Che Hsu
UIN: 826001725

In this MP, I implement Round-Robin scheduler. However, I am not confident on my Round-Robin scheduler, so I submit 7 files. Use Macro definitions to enable Round-Robin scheduler. I hope grader could follow these steps to test my MP:

For FIFO scheduler + option1 (Fix interrupts management): Just Replace **thread.C, thread.H, scheduler.C, and scheduler.H**.

For option2 (Modify the scheduler to implement round-robin with a 50 msec time quantum.): Replace **thread.C, thread.H, scheduler.C, scheduler.H**, **simple_timer.C, interrupts.C, and kernel.C.** Then make sure that three below Macros are defined. Please **uncomment them manually** because they are commented in default.
In **simple_timer.C**: Line15, #define _RR_MODE_TIMER_
In **interrupts.C**: Line14, #define _RR_MODE_INT_
In **kernel.C**: Line24, #define _USE_RR_
If these Macros are all defined, it runs option2, Round-Robin scheduler with 50 ms quantum.
If all of these macros are “not” defined, it will run FIFO scheduler + option1. This is default setting.

If not replacing kernel.C, I think it is OK. Because I only make it not compile “pass_on_CPU()” in every thread function. This is to make sure the only way for thread changing is round-robin scheduling.

Next, I would like to explain what I do in each file:
In **scheduler.H** and **scheduler.C**: Use List structure to model queue behavior. Each Node in the list has a thread pointer and a pointer to next node. When add/resume, this queue attach this thread pointer to the next node of the tail of the list. When yield, this queue pop it’s the head node. When terminate, it compare thread ID in the queue and remove the node with the same requested thread ID. Here, the queue only takes dequeue and enqueue to remove the requested node rather than directly change next pointer of the node which is prior of the requested node in the middle of the list.

In **thread.H** and **thread.C**: Incude Machine.H, and enable interrupts in thread_start()

because when thread creating sets Interrupt Enable Flag in EFLAG as 0, and then return back from fake exception, the interrupt disable. Therefore, I choose hread_start() to enable interrupts here. Besides, when thread shutdown, I check the scheduler queue whether there is node with this thread id, and remove this node. Then, delete this thread to release memory and call scheduler->yield to execute next thread in the ready queue.

In **simple_Timer.C**, I change timer counts part from hz to hz/20, to obtain 50ms trigger. Each trigger if the pointer of the current thread is not NULL, sets scheduler->resume (pointer of current thread) to save the pointer of current thread into ready queue, and then scheduler->yield() to obtain next thread in ready queue.

In **interrupts.C**, to make the interrupt controller know that the interrupt has been handled, I adjust the order in dispatch_interrupt() to send EOI before entering handler->handle_interrupt. When I use timer interrupts to trigger round-robin schedule, it may happen other interrupts as thread changing. For fitting the description in MP5 handout: let PIC knows the interrupts has been processed, and not let original thread's interrupt task finished response much later, I reorder the code flow in dispatch_interrupt().