Document

Zhiren Zhou

November 2022

1 Options Implemented

Beyond the simple FIFO Scheduler, I implemented the **OPTION1 Correct handling of interrupts** and **OPTION2 Round-Robin Scheduling**.

2 Design Concept

1. Implemented the **Scheduler** class:

   This is a very simple scheduler class. Most of its function is virtual function. We will not use this class to create the scheduler object. The only use of this class is to define the ready queue structure for its subclass.

2. Implemented the **FIFOScheduler** class:

   FIFOScheduler is the subclass of class **Scheduler**. Most function in class **Scheduler** is virtual function which can be redefined by subclass such as FIFOScheduler or RRScheduler.

3. Implemented the **FIFOScheduler::FIFOScheduler()** function:

   This is the constructor of FIFOScheduler. It prints one piece of construction information to the console.

4. Implemented the **FIFOScheduler::yield()** function:

   This function is called by current running thread to voluntarily yield CPU to the most front thread in ready queue. This function pops the first thread in ready queue and makes it the next thread to take over the CPU. It then calls **Thread::dispatch_to()** to make a context switch. After returning from this function, CPU starts to execute another thread.

5. Implemented the **FIFOScheduler::resume()** function:

   This function puts one thread to the end of the ready queue. This function

is mainly called when a blocked thread is waked up by some certain event or before one current running thread has to yield the CPU.

6. Implemented the **FIFOScheduler::add()** function:

   This function is used to add a new thread to the end of ready queue. This function is mainly called after one new thread is created.

7. Implemented the **Thread::thread_shutdown()()** function:

   When one thread function terminates, that thread will call the **thread_shutdown() function**. This function will call the **FIFOScheduler::terminate()** function, asking what to do next, then release the memory of that thread(including its stack).

8. Implemented the **FIFOScheduler::terminate()** function:

   This function rules what to do when the thread returns from the thread function. If that thread is the current running thread, the scheduler just need to execute yield() and then release the memory of that thread control block. If that thread is the not the current running thread, the scheduler needs to remove that thread from the ready queue and then release the memory of tcb.

9. **OPTION1**:Modified the **Thread::setup_context** function:

   During the context switch, this function pushes nonzero ELAG, specifically setting 9th bit which is the IE flag to enable interrupts for the option 1.

10. **OPTION2** Implemented the **EOQTimer** class:

    This class is a subclass of SimpleTimer. Round Robin Scheduler needs this EOQTimer object to trigger preemption.

11. **OPTION2** Implemented the **EOQTimer::handle_interrupt** function:

    This function is a interrupt handler and it will be called when the system goes through a tick, which equals to one second/hz. When the system goes through certain ticks, the quantum of current running thread is used up. As a result, this function will call the **RRScheduler::quantum_handler()** to forcely make current running thread preempted.

12. **OPTION2** Implemented the **RRScheduler** class:

    This Round Robin Scheduler class is also a subclass of Scheduler class. With a designed EOQTimer, the RRScheduler triggers an preemtion after every quantum.

13. **OPTION2** Implemented the **RRScheduler::yield()** function:

    This function needs to distinguish between a voluntary yield action and passive yield action. If the current running thread calls yield() voluntarily, this function does the same thing as the **FIFOScheduler::yield()** does. Otherwise, this function is called by **RRScheduler::quantum_handler()** and it needs to send an info to tell the EOQ handler to return first, after which it can do the context switch. Or we can just enable interrupt in **yield()** so that the life can be much easier.

14. **OPTION2** Implemented the **RRScheduler::resume()** function:

    As same as **FIFOcheduler::resume()** function.

15. **OPTION2** Implemented the **RRScheduler::add()** function:

    As same as **FIFOcheduler::add()** function.

16. **OPTION2** Implemented the **RRScheduler::terminate()** function:

    As same as **FIFOcheduler::terminate()** function.

17. **OPTION2** Implemented the **RRScheduler::quantum_handler()** function:

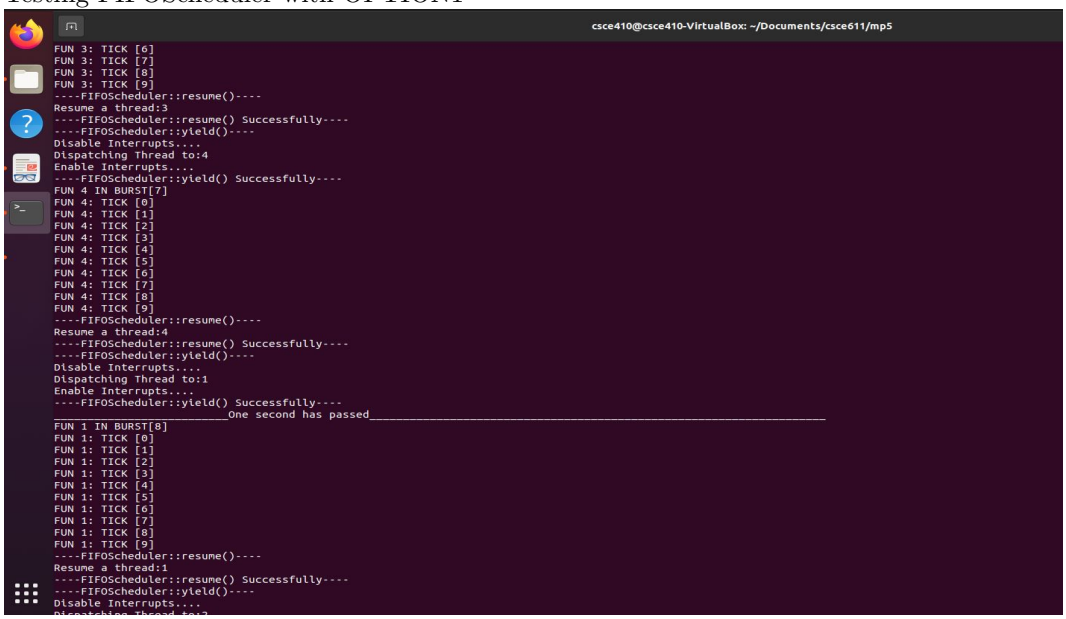
    This function is called by the **EOQTimer::handle_interrupt** when the quantum of current running thread is used up. This function will call the **EOQTimer::yield()** to make current running thread forcely preempted.

3 Files Modified

1. **scheduler.H**
2. **scheduler.C**
3. **thread.C** (for **OPTION1**)
4. **simple_timer.H** (for **OPTION2**)
5. **simple_timer.C** (for **OPTION2**)
6. **kernel.C** (for Test:Comment the macro _FIFO_SCHDEULING in kernel.C to select the RRScheduler.)

4 Testing Results

1. Testing FIFOScheduler with OPTION1



2. Testing RRScheduler with OPTION1