Nachos Multiprogramming: Preemptive Round-Robin Scheduling with Quantum

1 An Important Reminder

Review the university's academic integrity policy (http://class.syr.edu/academic-integrity). Remember that violating academic integrity policy will significantly jeopardize your grade.

2 Learning Objectives

After completing this project assignment, students can

1. describe how timer device is emulated in Nachos, (i.e., `timer.[cc|h]`),
2. describe how timer interrupt service routine is called and how it is implemented,
3. describe Round-Robin Scheduling with Quantum,
4. implement a Round-Robin Scheduling with Quantum in Nachos,
5. describe how a Timer Interrupt Service routine works and
6. modify a Timer Interrupt Service routine as a part of Round-Robin Scheduling with Quantum.

3 Preliminary Requirement

Get a fresh copy of Nachos from `nachos.tar`. Refer to Lab 1.

4 Preemptive Scheduling with Quantum

Based on your implementation from Project 1 and Project 2, we will implement a preemptive Round-Robin Scheduler with Quantum in this project. As we learned in the class, the timer interrupt will occur at regular intervals.

5 Timer.[cc|h]

Two files, `machine/timer.[cc|h]`, implement an emulation of hardware timer. Read the comments at the beginning of `timer.cc`.

Question 1 *(6 points) What is the value of* `TimerTicks`*? How often does a timer interrupt occur?*

Question 2 *(6 points) In line 99 of* `kernel.cc`*, a timer device* `alarm` *is created by the code* `alarm = new Alarm(randomSlice);`*. What is the data type of the variable* `randomSlice` *and what is its value in this line?*

Question 3 *(6 points) Which source file in the* `machine` *directory implements the function* `OneTick()`*? What is the function's role? Two situations can cause* `OneTick()` *to be called. One of them is when a user program instruction is executed. Which function in which file contains this call when a user instruction is executed?*

Question 4 *(12 points) Make sure that the timer object* `alarm` *is created in line 99 of* `kernel.cc`*. Modify* `void Alarm::CallBack()` *function in* `alarm.cc` *as below by inserting* `printf("In Alarm::CallBack(), totalTicks = %d\n", kernel->stats->totalTicks);`

```
void Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();

    printf("In Alarm::CallBack(),
             totalTicks = %d\n", kernel->stats->totalTicks);
    if (status != IdleMode) {
        interrupt->YieldOnReturn();
    }
}
```

Compile Nachos and run the following command.

```
./nachos -x ../test1/write | more
```

If you do not use `more`*, you will see all the outputs scrolling by.* `more` *will allow you to inspect the output one page at a time. To display the next page, you can press the* `space bar`*. Explain*

the output. Explain how `void Alarm::CallBack()` *is called from the time the user program* `write` *was executing on the CPU.*

Question 5 *(20 points total) In the above question, the "*`In Alarm::CallBack(), totalTicks = x`*" message is printed out at every regular interval x.*

1. *(4 points) What is the value of x?*
2. *(4 points) Which Nachos source file defines this value of x?*
3. *(12 points) How should we modify the* `void Alarm::CallBack()` *function, if we want the message to be printed out at every $4 \times x$ interval? Give a pseudo code.*

6 Overview for Design and Implementation of Round-Robin Scheduler using Timer

6.1 Note: Before You Start the Implementation Part

We again use the same new `addrspace.[cc|h]` from Project 1 and 2. Therefore, the memory management is done for you. Make sure to use the correct `exception.cc` that takes care of `Write ()` and `Read ()` system calls.

Make sure to delete or comment out the `printf(···)` inserted for Question 4 unless you want to see all the messages printed out scrolling fast on your screen!

6.2 Implementation Requirement

You need to implement a new flag "-quantum" that is used to set the size of the timeslice for round-robin scheduling. With everything implemented, the command:

```
./nachos -quantum <quantum_size> -x ../test1/prog1 -x ../test1/prog2
```

should run `prog1` and `prog2` with round-robin scheduling.

We provide two user programs, `prog1.c`, `prog2.c`. They call `Write()` to print messages to screen and `Exit()` at the end to quit. Like in Project 1 and 2, the implementations of these system calls are provided `exception.cc`. Note: the implementations given are not full Nachos system calls implementations using Nachos console device.

7 Implementation Requirements (50 points)

1. The simulated hardware `Timer` object is created in the `Alarm` class in `alarm.h` and `alarm.cc`. By default, a timer interrupt occurs at every 100 timer ticks. Therefore, the default quantum of Nachos is 100 ticks. When the timer interrupt happens, the timer interrupt handler, which

is `Alarm::CallBack()`, is invoked. This handler forces the currently running thread to give up the CPU by setting the `interrupt->YieldOnReturn` flag. To change this behavior, you need to make sure the `interrupt->YieldOnReturn` flag is only set after the given "quantum" number of ticks.

2. Since the timer generates a timer interrupt every 100 ticks, the granularity of quantum is 100 ticks, i.e., the actual quantum is rounded to the next multiple of 100 for any value given with the "-quantum" flag.

3. When implementing the argument parsing for "-quantum" flag, you may get a hint from the parsing of "-rs" flag found in `kernel.cc`.

8 Tests and Output

Your program will be tested with the following command at minimum:

```
./nachos -quantum <quantum_size> -x ../test1/prog1 -x ../test1/prog2
```

The correct output contains messages from 2 user programs, 5 from each. The order of the messages changes with the value of `<quantum_size>`.

9 What to Submit

You must submit two files to Bb: (1) The PDF report and (2) The source code zip file.

1. **PDF report:** You need to submit a detailed project report describing what you have done to implement the requirements and your outputs. Answer all questions. Please also list the important code snippets followed by an explanation. These code snippets should include the modified part of the Nachos source code. Simply attaching code without any description will not receive credits.
2. **Source code zip file:** After "`make clean`" in `build.linux`, please compress your whole `nachos` folder, name the compressed file LastName,FirstName.zip (NOTE: .zip ONLY!)