CSCI 40300/ECE 40800 — Operating Systems
Project 1: Thread Scheduling
Due Date: Sunday, Oct 6, 2018, 11:59pm

1 Overview

In this assignment your job will be to enhance the thread scheduler to implement the following three scheduling algorithms: **Round Robin (RR), Shortest Job First (SJF), and Priority (PRIO) Scheduling**. For SJF and PRIO, you will be implementing both the preemptive and non-preemptive versions. Currently, Nachos uses First Come First Served (FCFS) scheduling. After you make your enhancements, FCFS should still run exactly as it did before you started.

2 The Assignment

2.1 Preparation

2.1.1 Nachos related background

The suggested method of preparation requires a lot of background reading. After reading this handout, please read the paper titled: "A Road Map through Nachos." It is a comprehensive overview of Nachos and it will guide you while reading the source code. Please not that this documentation is for the C++ version. The Java version was ported from the C++ version and includes the minimal changes necessary which are explained in the README file that comes with the source code. The source code is given as an attachment to this assignment on Canvas. It is tarred and gzipped (indicated with the tgz extension). (MP stands for Machine Problem.) In order to extract it, type the following command at the shell (on tesla (tesla.cs.iupui.edu — do it on this particular server and not some other cs department linux machine— the java compiler may not be installed on other servers.):

```
tar xvfz mp1given.tgz
```

You will see tar listing out the files it's creating for you; the output will be placed in the directory mp1given. **Do not issue this command again from this directory unless you want to overwrite all changes you have made!** If you wish to replace individual files, untar the mp1given.tgz file in a different directory and copy the relevant files.

For further details on Nachos refer to http://www.cs.washington.edu/homes/tom/nachos/.

Caution: Because unix/linux systems handle end-of-line differently than Windows systems, it is recommended that you untar and unzip this archive file on tesla instead of on your own PC. That is, copy mp1given.tgz onto tesla and then expand it on tesla. Expanding it on your PC first and copying the expanded directory/folder structure to tesla, may create problems. Generally, this is one big source of problems with a code that compiles and works on your personal computer (typically Windows) and then fail to compile on your submitted code because we test your submitted code on tesla. If the compilers handle the end-of-line differently, this may create problems.

2.1.2 Project related background

To compile the code, cd into the **threads** directory and type **make**. Ignore the deprecation warnings or fix them. Run the program test0 to see a simple test for FCFS. (If the current directory "." is not on your directory search path, type ./test0; if you don't know what I am talking about here, just type ./test0).

As mentioned before, FCFS scheduling has been done for you in Nachos as the default implementation. So you can run the test case, test0 and see the test case results, which are also provided to you in the file test.0.std. The test routine (In SchedulerTest.java) tries to simulate the arrival of processes in a system using kernel threads. In some future MPs we will be working with "user level" programs rather than kernel threads. This will explain the calls to OneTick() in the thread body; kernel threads in Nachos do not update the clock and hence we simulate the clock by calling OneTick().

I have also put sample results for the other test cases as test.?.std file, where ? stands for the number of the test case. Compare the output of your implementation to this std output given. I don't expect these to match character by character, but generally the output produced by your implementation should be correct scheduling. Use these files as

a guide, but do not waste a lot of time trying to exactly duplicate them byte by byte (I've had students do that in the past).

Here are a few pointers that may help as you begin your reading.

- The main routine for Nachos is in Nachos.java. You should be able to follow the code from there.
- Note the values of the static variables in the Nachos class such as THREADS to help you read the code.
- Each test case simulates the arrival of different "threads" into the system.
- The scheduling policy is setup for you. Check the static variable Scheduler.policy at run time to find out which policy you are running.

2.2 Problem: Implementation of scheduling algorithms

Files to be modified: Scheduler.java

Look for the places in the source code given to you where it says MP1. It means you might (or might not according to your implementation) have to add some code at that place.

Look at Scheduler.findNextToRun() in Scheduler.java. This is the function which implements the scheduling algorithm. Based on the scheduling policy set for you, implement the appropriate algorithm.

Look at Scheduler::shouldISwitch() in Scheduler.java. Upon arrival of a new thread, this function decides whether or not to switch to the new thread.

2.2.1 Round Robin

Implement the Round Robin scheduling algorithm. There are hints given in the code for how to do this. You will need to schedule an interrupt upon arrival of a new thread for the preemption of that thread. You will do this with a call to

```
Interrupt.schedule(Runnable handler,int fromNow, int type)
```

method. This is defined in machine/Interrupt.java file. handler is the interrupt handler you will need to define in threads/Scheduler.java. You will make the call to Interrupt.schedule() from Scheduler.java as well.

For uniformity, only do this if you have more than 4 ticks (which works out to 40 "ms") remaining and set the interrupt for 40 ms. Each thread has a field that tells how much time is left. Since this is practically impossible to know, you may assume that each thread has associated with it, an estimated running time which happens to always be right. This algorithm is tested by test1.

2.2.2 Shortest Job First

Implement the SJF non-preemptive and preemptive scheduling algorithms. Base your scheduling on the time left field mentioned above. These scheduling algorithms correspond to test2 and test3 respectively.

2.2.3 Priority

Implement priority non-preemptive and preemptive scheduling. Each thread has a priority associated with it. The priority values range from 0 to 2. Threads of priority 0 are the highest priority and threads of priority 2 are the lowest priority. These scheduling algorithms correspond to test4 and test5 respectively.

2.3 Testing, Delivery and submission

The final version of your project (correctly compiling and running, and properly debugged) should be submitted on the project due date. For the final version of your project code, you must submit the entire source directory as described below, so that we can uncompress, compile and test it.

Your code must compile and run on the tesla.cs.iupui.edu for this and all the following assignments. It should not matter where you develop your code for this MP but please test it on tesla before submitting it. Try all 6 test cases including the test0 for FCFS which is already implemented; but you do not want to break it while implementing the other parts. Make sure they work before handing in.

When you are satisfied that your code works, submit your solution on Canvas. In order to submit your MP 1 implementation, run

```
make cleanclass
```

under the threads directory, which will remove all the compiled (.class) files. Then go to the mp1given/.. directory (i.e., one level above the mp1given source code) and give the command:

```
tar cvfz mp1given.tgz mp1given
```

Then, submit this tarred and compressed source code archive on Canvas, in the assignment section to be graded.

The due date for submission of the assignment is Friday, March 7, 2014. You should submit your project by 11:55pm on March 7, 2014.

2.4 Grading criteria:

Your grade will be based on the correctness of your code and largely based on your code's ability to produce correct output in the test cases.

The final project submission	
Shortest Job First (Preemptive)	20%
Shortest Job First (Non-preemptive)	20%
Round Robin	20%
Priority (Preemptive)	20%
Priority (Non-preemptive)	20%
Total	100%

3 Some Nachos-Specific Details

The files to look at for this assignment are:

Nachos.java – sets up the system and invokes a function SchedulerTest.ThreadTest().

SchedulerTest.java – runs the appropriate test for the scheduling algorithm specified on the command line.

NachosThread.java – thread data structures and thread operations such as thread fork, thread sleep and thread finish.

Scheduler.java – manages the list of threads that are ready to run.

List.java – generic list management (LISP in Java). This is here because it's useful, not because it contains deep operating system secrets.

System.java – Nachos startup/shutdown.

Interrupt.java – manage enabling and disabling interrupts as part of the machine emulation.

Statistics.java – collect interesting statistics.