CMPT 300 Fall 2011 Project #2: CPU Scheduling in Nachos

In this project you will learn how to schedule CPU for threads. You are given a simple scheduling system skeleton in Nachos and your tasks are:

1) **Compile Nachos and run the system with pre-implemented Round Robin, Non-preemptive Priority CPU scheduling algorithms.**
2) **Read the code and understand how the given CPU scheduling algorithms are implemented.**
3) **Implement the First Come First Serve, Preemptive Priority and Multilevel Queue CPU scheduling algorithms in Nachos. Recompile and run the system to test your implementation.**
4) **Explain the results and answer some questions.**

Please don't be overwhelmed by the sheer amount of code provided. In fact you don't need to worry about most of it. The parts that you need to read or modify are given in the following instructions. Please read them carefully, and follow the steps.

Task 1: Run Nachos with Pre-implemented Scheduling System Skeleton

Step 1: Download Nachos source code of this project

```
wget http://www.cs.sfu.ca/CourseCentral/300/zonghuag/projects/project2.tar.gz
```

Step 2: Extract the source code

```
tar zxvf project2.tar.gz
```

Step 3: Compile the code

Enter the folder "project2" and then run "make"

Step 4: Run Nachos

This program was designed to test 5 scheduling algorithms namely First Come First Serve(FCFS), Non-preemptive Priority(PRIO_NP), Preemptive Priority(PRIO_P), Round Robin(RR), and Multilevel Queue(MLQ). To cover all these cases, we do not run the executable file 'nachos' directly. Instead, we run 'test0' through 'test4' to test each of 5 scheduling algorithms respectively.

For example, you can run 'test1' to test Non-preemptive priority scheduling algorithm.

```
./test1
```

If you succeed in running 'test1', you will see the following messages:

```
Nonpreemptive Priority scheduling

Starting at Ticks: total 10

```

```
Queuing threads.

Queuing thread threadA at Time 10, priority 7

Queuing thread threadB at Time 20, priority 5

Queuing thread threadC at Time 30, priority 2

Queuing thread threadD at Time 40, priority 8

Queuing thread threadE at Time 50, priority 6

threadD, Starting Burst of 7 Ticks: total 70

threadD, Still 6 to go Ticks: total 80

threadD, Still 5 to go Ticks: total 90

threadD, Still 4 to go Ticks: total 100

............(We omitted some output here.)............

threadC, Still 1 to go Ticks: total 660

threadC, Still 0 to go Ticks: total 670

threadC, Done with burst Ticks: total 670

No threads ready or runnable, and no pending interrupts.

Assuming the program completed.

Machine halting!


Ticks: total 670, idle 0, system 670, user 0

Disk I/O: reads 0, writes 0

Console I/O: reads 0, writes 0

Paging: faults 0

Network I/O: packets received 0, sent 0


Cleaning up...
```

To be concise, we omitted several output lines.
The following table would give very useful information to you.

Executable	**Source**	**Corresponding**	**Already**

File	File	Algorithm	Implemented?
test0	test.0.cc	FCFS	No.
test1	test.1.cc	Non-preemptive Priority	Yes.
test2	test.2.cc	Preemptive Priority	No.
test3	test.3.cc	Round Robin	Yes.
test4	test.4.cc	Multi Level Queue	No.

You can run test1and test3 to test the pre-implemented algorithms. However, because FCFS, PRIO_P and Multilevel Queue algorithms are not yet implemented, if you run test0, test2 and test4 to test the given system skeleton, there will be an error. You can view the source code of test files in test.0.cc, test.1.cc, test.2.cc, test.3.cc test.4.cc respectively.

Step 5: Read the code

Please read the code carefully. Try to understand how the given scheduling algorithms are implemented. You only need to focus on *`scheduler.h`*, *`scheduler.cc`*, *`list.h`* `and` *`list.cc`*. Here we provide you some notes about the code.

The CPU scheduling algorithms are mainly implemented in 3 functions: `ReadyToRun(), FindNextToRun(), ShouldISwitch()`, in scheduler.cc.

1) *`ReadyToRun()`* decides the policy of placing a thread into ready queue (or multilevel queues) when the thread gets ready. For example, in round robin we simply append the thread to the end the ready queue, while in priority scheduling we insert the thread to the queue according to its priority.

2) *`FindNextToRun()`* decides the policy of picking one thread to run from the ready queue. For example, in both round robin and non-preemptive priority scheduling, we fetch the first thread in ready queue to run.

3) *`ShouldISwitch`*`()` decides whether the running thread should preemptively giveup to a new forked thread. In both round robin and non-preemptive priority scheduling, the running thread does not preemptively give up its CPU resources. Note that only in preemptive algorithms, it is needed to decide whether the running thread should give up or not. In other algorithms, you can simply return false.

Task 2: Implement Three Scheduling Algorithms

In this task, you are required to implement the remaining three scheduling algorithms including FCFS, Preemptive Priority and Multi Level Queue, and then test your implementation. To achieve these, you needn't modify any source file other than *`scheduler.cc`*. You are supposed to add some code in the following three functions.

```
Scheduler::ReadyToRun
Scheduler::FindNextToRun
Scheduler::ShouldISwitch
```

They are so-called CALLBACK functions that will be invoked by Nachos scheduler.

Note: Be very careful of **`cases`** in **`switch`** block(s) in each of those functions. Make sure you put your code in the right place.

Since you have to operate one or more Lists, you could refer to *`list.h`* and *`list.cc`* to get familiar with List operations. If you make good use of appropriate List operations, it is expected that you add **no more than 30 code lines** in all to *scheduler.cc*. However, it's only an advice, so feel free to add any number of reasonable code lines.

Step 1. Implement FCFS Scheduling

In this step, you are supposed to add some code with respect to FCFS Algorithm in **`caseSCHED_FCFS`** in each function. In FCFS, the threads should be scheduled in a first-come first-served manner.

Hint:This algorithm is easy to implement because you only need to append the thread to the end of readyList when a thread gets ready and return the first thread in readyList when scheduler needs to pick a thread to run.

Then, you should run "make clean" and then "make" to recompile the code and run test0 to check the output.

```
./test0
```

Step 2. Implement Preemptive Priority Scheduling

In this step, you are supposed to add some code with respect to Preemptive Priority in **`caseSCHED_PRIO_P`** in each function. In Preemptive Priority algorithm, the thread with the highest priority (largest priority number) in readyList should be scheduled for running all the time. If there are more than one thread with the same highest priority, they must be scheduled in FCFS manner.

Some notes are given to you.

1. The priority of a thread is an integer between MIN PRIORITY and MAX PRIORITY (defined in class *thread*). If no explicit priority is given to a new thread when the thread is forked, it has the same priority as its parent. The first thread created in a process is set to NORM PRIORITY.

2. Do NOT use function SetPriority to change the priority of a thread dynamically in your own test file.

Hint: You can insert the thread to readyList according to its priority when a thread gets ready. Therefore, it can be guaranteed that the first thread in readyList is the thread with the highest priority.

Then, you should run "make clean" and then "make" to recompile the code and run test2 to check the output.

```
./test2
```

Step 3. Implement Multi Level Queue Scheduling

In this step, you are supposed to add some code with respect to Multi Level Queue Algorithm in **`caseSCHED_MLQ`** in each function. In Multi Level Queue algorithm, threads are placed on different queues with different priorities instead of ready queue. The threads placed in a queue with higher priority have higher priority than those who are placed in a queue with lower priority. The threads in different queues should be scheduled by their priorities, while the threads in the same queue should be scheduled in **FCFS** manner (**Note: Different from the lecture slides, we adopt FCFS instead of Round-Robin here for simplicity.**).

Some notes are given to you.

1. Multi Level Queue Algorithm is different from and much simpler than Multi Level Feedback Queue Algorithm. **You are NOT asked to do the later one**.

2. The number of priority queues is specified by the function Scheduler::SetNumQueue(). When the number of queues is set to NumOfLevel, you can visit those NumOfLevel priority queues by using MultiLevelList[0] to MultiLevelList[NumOfLevel-1]. (Reminder: **readyList is not used in Multi Level Queue Algorithm**)

3. The thread with largest priority number has the highest priority.

4. The priority of a thread is specified in the "priority" field in class Thread. It should be the same as the priority of the queue in which the thread is placed.

Hint: You can scan the queues from high priority to low priority until you find a thread which can be put to run.

Then, you should run "make clean" and then "make" to recompile the code and run test4 to check the output.

```
./test4
```

Step 4. Save the output

After finishing your implementation of all the algorithms, please save your output of test0, test2, test4 to `project2_test0.txt`, `project2_test2.txt`, `project2_test4.txt` respectively. For example, you can save your output of test0 to project2_test0.txt by

```
./test0  >project2_test0.txt
```

Please keep project2_test0.txt, project2_test2.txt, project2_test4.txt and your source code scheduler.cc for grading.

Task 3: Explain the Result

Understand the output of test2 (preemptive priority scheduling) and answer the following question.

1) How many times does the running thread preemptively give up to the new thread with higher priority?
2) Can you point out in which lines in project2_test2.txt does these actions happen?

Please write the answer in project2_report.txt.

After Finishing These Tasks:

1) Please generate a single tar.gz file and submit it.
2) The name of the ZIP should be "proj2_********.tar.gz", using your student ID to replace star symbols.
3) The following files should be included insides the ZIP:

File Name	Description
scheduler.cc	Source file you have accomplished by the end of Task 2
project2_test0.txt	Output of test0
project2_test2.txt	Output of test2
project2_test4.txt	Output of test4
project2_report.txt	The answer to the questions in Task 3