CIS 657 (POS) fall 2013

Lab 4: Priority based Roundrobin Scheduler with Single Run-queue

# 1. The 4.4BSD Scheduler

All threads that are runnable are assigned a scheduling priority that determines in which [*run queue*](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry325) they are placed. In selecting a new thread to run, the system scans the run queues from best to worst priority and chooses the first thread on the first nonempty queue. If multiple threads reside on a queue, the system runs them [*round robin*](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry316); that is, it runs them in the order that they are found on the queue, with equal amounts of time allowed. If a thread blocks, it is not put back onto any run queue. When a thread is inserted in a run queue, it is placed at the end of the queue from which it came, and the thread at the front of the best priority queue is selected to run.

#### Time-Share Thread Scheduling

The FreeBSD time-share-scheduling algorithm is based on [*multilevel feedback queues*](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry216). The system adjusts the priority of a thread dynamically to reflect resource requirements (e.g., being blocked awaiting an event) and the amount of resources consumed by the thread (e.g., CPU time). Threads are moved between run queues based on changes in their scheduling priority (hence the word feedback in the name [*multilevel feedback queue*](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry216)). When a thread other than the currently running thread attains a higher priority (by having that priority either assigned or given when it is awakened), the system switches to that thread immediately if the current thread is in user mode. Otherwise, the system switches to the higher-priority thread as soon as the current thread exits the kernel. The system tailors this [*short-term scheduling algorithm*](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry352) to favor interactive jobs by raising the scheduling priority of threads that are blocked waiting for I/O for 1 or more seconds and by lowering the priority of threads that accumulate significant amounts of CPU time.

(Excerpts from the textbook: The Design and implementation of FreeBSD Operating System by Marshall Kirk McKusick)

# 

# 2. Priority based RoundRobin Scheduler with Single Run-queue

Priority based RoundRobin Scheduler with Single Run-queue, as the name suggests, uses only one queue to keep threads of all priorities. The scheduling invariant for the system is that the best priority thread is always running, and among a set of threads with the equal best priority are executed in round robin fashion. The threads in that single queue are kept sorted on the basis of their scheduling priorities. The thread at the head of the queue is selected for execution.

runq



All priorities

# 3. Tasks [90]

In this lab, students need to implement a Priority based RoundRobin Scheduler with Single Run-queue in FreeBSD. Following is the list of tasks to be accomplished:

1. Add all the threads with various scheduling priorities in a single run-queue.
2. Add them in sorted order. If the thread to be added has same priority as some existing thread in the queue, decide whether to add the thread before or after the existing thread with same priority. Give justification for the choice you made.

For example, consider the situation of runq below. The numbers in bracket represent thread priorities. Suppose if another thread with priority 10 comes up, where you would insert it, at location one or two and why:

Runq

Location 1

Location 2

1. Choose the thread from the head of queue as the next thread to run.
2. Remove thread from the head of the queue.
3. Place KTR dumps at appropriate places to show that you are adding/choosing/removing thread from a single queue. KTR dumps should also mention priority and process id of thread.
4. Your comments on single run-queue based scheduler. Is it good or bad compared to FreeBSD 64 run-queue based scheduler?

# 4. Submission [10]

Create and attach a README (txt/word/pdf) file at the end of the lab. It doesn't need to be comprehensive, but it should at least cover the following content:

Which tasks are done, and which are not?

What’s your basic idea to achieve this task?

Where is your main function? Which files you have modified and under which function?

If you can only finish some of the tasks in this project, please make sure that your code can at least be compiled and installed and also clearly state in the README file about the missing parts of your project.

You also need to attach the .tgz file.To create this file, there is script on the website named “tar and Gzip Source code” under resources tab.

Checklist: To submit your project, you need to:

• Attach the \*.tgz file (make sure kernel compiles and runs with this snapshot).

• Create and attach a README file report.

• Send this email to the TAs keeping Dr. Chapin < chapin@syr.edu> in the CC with subject line “CIS657: Lab 4”

Saurabh Sabnis < spsabnis@syr.edu >

Prasoon Pandya < [pdpandya@syr.edu](mailto:pdpandya@syr.edu) >