CSE 410 - Computer Systems
Homework 6

Assigned: Friday, May 21, 2004

Due: Friday, May 28, 2004
At the start of class

Name: _____

Student #: _____

1. On most modern systems, there are usually several "ready queues" and several "wait queues." None of the tasks (threads) on any of these queues is actually executing instructions. For the following questions, describe a set of circumstances that would cause the given scenario to take place. There are many such possibilities.

a. Describe a set of circumstances under which a task would move from a wait queue to a ready queue. Include a description of the particular wait queue before the transition, the event that triggers the transition and a description of the particular ready queue after the transition.

b. Describe a set of circumstances under which a task would move from a ready queue to being the running task, executing instructions on a CPU. Include a description of a particular ready queue before the transition, the event that triggers the transition, and a description of how the scheduler picks this particular task to run next.

2. For scheduling purposes, a useful characterization of tasks is often that they are "I/O bound" or "CPU bound".

a. Give an example of a desktop PC task that is likely to be I/O bound often. Why does this task fit this description?

b. Give an example of a desktop PC task that is likely to be CPU bound for relatively long periods. Why does this task fit this description?

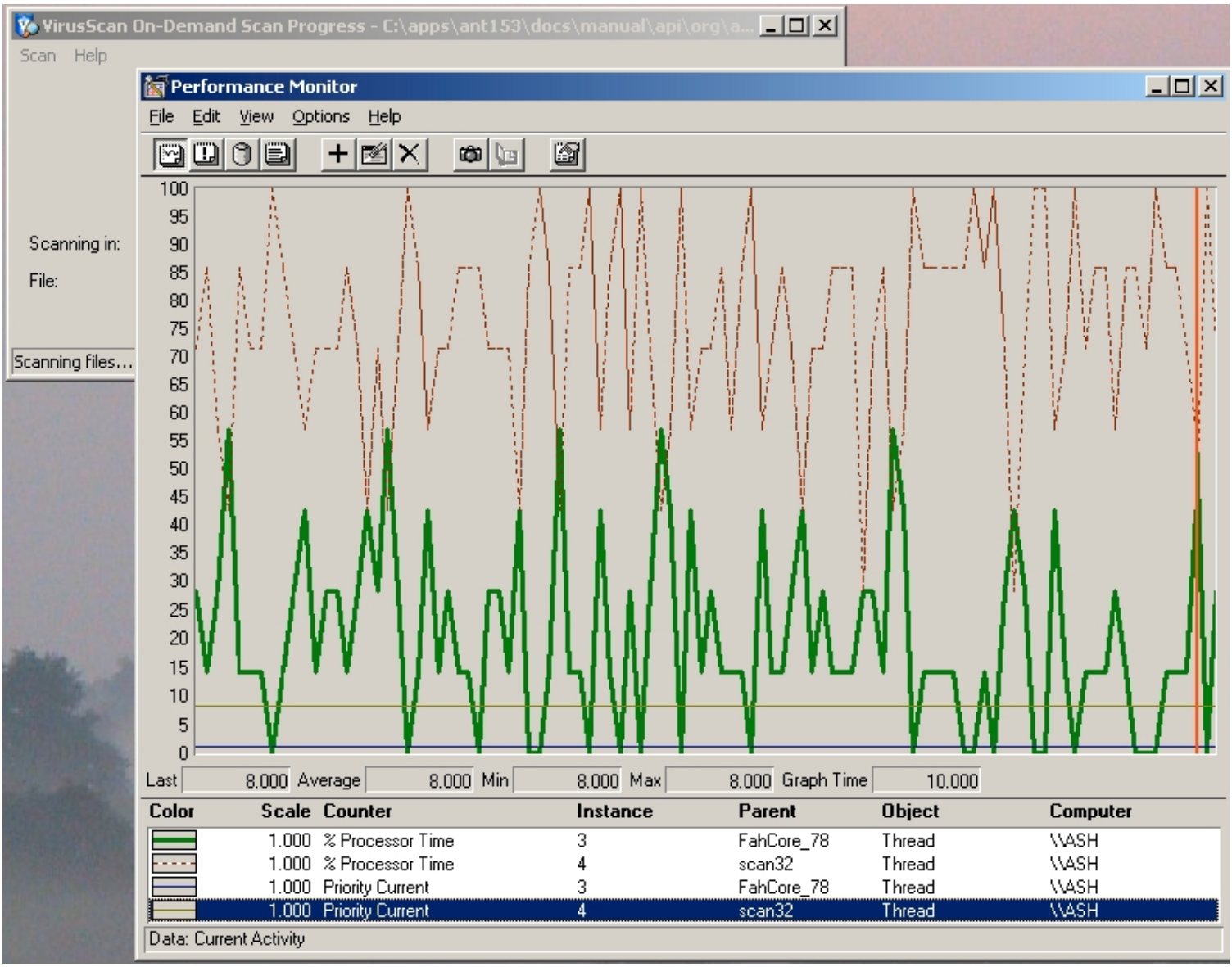
3. The lecture dated May 17 is about scheduling on Windows 2000. On slide 9 there is a description of theWin2K priority structure. Refer to that page for information on process priority classes.

On the next page of this homework there is a snapshot of a perfmon4 graph showing the folding@home calculation process running at the same time as a virus scanner program. The dotted line that bounces around 70 is the percent CPU being used by thread 4 of the scan32 (virus scanner) process. The heavy line that bounces around 25 is the percent CPU being used by thread 3 of the Fah_core (folding@home) process. The horizontal line at 8 is the current priority for thread 4 of scan32, and the horizontal line at 1 is the current priority of thread 3 of Fah_core.

a. What is the most likely process priority class for scan32? Why?

b. It is true that the Fah_core thread is CPU-bound. In fact, if there is no other activity on the machine, it will absorb 100% of the available CPU time. Would you say that the scan32 thread is also CPU-bound, or is it IO-bound? Why?

c. The priority of the Fah_core thread never gets to be higher than the priority of the scan32 thread, and yet the Fah_core thread gets significant amounts of CPU time. Using the terminology of the running thread, wait queues, and ready queues, explain how this could happen.


VirusScan On-Demand Scan Progress - C:\apps\ant153\docs\manual\api\org\a...
Scan Help
Scanning in:
File:
Scanning files...
Performance Monitor
File Edit View Options Help
100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
Last 8.000 Average 8.000 Min 8.000 Max 8.000 Graph Time 10.000
Color Scale Counter Instance Parent Object Computer
1.000 % Processor Time 3 FahCore_78 Thread \\ASH
1.000 % Processor Time 4 scan32 Thread \\ASH
1.000 Priority Current 3 FahCore_78 Thread \\ASH
1.000 Priority Current 4 scan32 Thread \\ASH
Data: Current Activity

4. Consider the slides in the May 17 lecture that describe various scheduling scenarios (slides 18 to 20). Note that the thread that comes off the ready queue to be run is always the one at the head of the queue. For this question, the term "scheduling events" means statements like "move to wait queue for device read", "quantum exhausted", "preempt by higher priority task" and so on.

a. Are the threads in these examples "dynamic" threads or "real time" threads?

b. Starting from the situation shown in slide 19, where thread B is running, describe a set of scheduling events under which thread A will get a chance to run.

c. Starting from the situation shown in slide 20, where thread C has just started running, describe a set of scheduling events under which thread E will eventually get to run.

5. Coordinated access to shared state variables is the key issue in synchronization of multiple threads.

a. On a preemptively scheduled system, is the following **`if`** statement an "atomic operation", or is it possible for a thread to be interrupted while executing the statement?

```
if (j < k) j++;
```

b. Is the "**`j++`**" portion of the statement an "atomic operation" or is it possible for the thread to be interrupted while executing the statement?

c. Assuming that j and k are shared state variables, show how you would use a lock with this code to prevent corruption due to a context swap at an inconvenient moment. You can use the same notation conventions that I used in the Synchronization lectures, namely `lock->acquire()` and `lock->release()`. This is not a trick question; the answer is simple.

6. Consider the lecture "Synchronization Part 2". On slide 12, there is an example of how Push() and Pop() functions can be implemented to provide coordinated, multi-thread access to a single shared stack.

a. Notice that the Push procedure uses **`condition->signal(lock)`** to alert any waiting thread that there is something on the stack. If that call were replaced with **`condition->broadcast(lock)`**, do you think that the procedures would still work correctly in a multi-threaded application?

b. Describe what would happen when a broadcast took place. If you think it won't work correctly, describe an example where it fails. If you think it will work correctly, describe why it won't fail.