Textbook Readings

Sec. 6.3  **CPU scheduling algorithms.** (Continuation of Sec. 6.1-6.2 from assignment 5 readings)

Sec. 4.1  **Threads.** A thread is a “lightweight process” that shares memory and other resources (such as files) with the parent process. Threads can be implemented more efficiently than processes, with faster thread creation and faster context switches.

Sec. 14.1-14.5, 14.7  **Protection.** Protection domain. Two ways to implement the access matrix: access lists and capabilities. Revocation of access rights.

Sec. 15.2.4; course reader pp. 60-66  
**Buffer overflow attack** – an example of a security breach. [Unfortunately, we do not have time to cover the many other interesting security topics in Chapter 15. These include user authentication, Trojan horses, trap doors, worms, viruses, intrusion detection, and encryption.]

Sec. 2.7, 16.1, 16.3  **Operating System structure.** OS code can be organized into layers, or into a kernel and processes. Virtual machines.

Optional review in Chapters 18 (Linux) and 19 (Windows 7). These chapters provide excellent review of course material.

You do not need to memorize any aspects of Linux or Windows 7 for the final exam, but you are invited to skim parts of these chapters, depending on what topics interest you. I will discuss parts of these chapters in lecture during the last week of term.

**Questions about CPU Scheduling**

1) Under a round-robin scheduling strategy, what is the advantage of choosing a short time quantum? What is the advantage of choosing a long time quantum?

2) $N$ processes are sharing a CPU in round-robin fashion. The time-quantum $Q$ is the length of time a process is allowed to use the CPU. Each process switch takes $S$ seconds. (This process switch time is not counted as part of $Q$.) How long should $Q$ be, so that all processes get a full time slice at least once every $T$ seconds?

$$Q \leq \text{expression involving } N, S, T$$

3) You are given the following information about five processes that are being executed. Only one CPU burst is described for each process.

<table>
<thead>
<tr>
<th>process Name</th>
<th>Arrival Time</th>
<th>CPU burst length</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>P5</td>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>

Show the scheduling of these jobs under the following policies. Use a timeline format similar to the Gantt charts shown in Section 6.3 of the textbook. Compute the **average turnaround time** under each of these scheduling algorithms.

- **FCFS (First-Come, First-Served).**
- **Round-robin with time quantum $Q$ equal to 1.** (You can put a newly-arrived process at the front or the end of the ready queue. Either policy is fine, just pick one policy and use it consistently.)
- **Round-robin with time quantum equal to 4.**
- **SJF (Shortest-Job-First).** Use the non-preemptive version of shortest-job-first (at the start of section 6.3.2), rather than the preemptive version (shortest-remaining-time-first, at the end of section 6.3.2).

Remember to compute the average turnaround time for the schedule resulting from each of these scheduling algorithms.
**Question about Threads**

4) Which of the following are private to a thread, and which are shared with other threads:

- the program counter
- the stack
- the code area
- the data area (which contains global variables)

For help, refer to the first paragraph of textbook Section 4.1.

**Questions about Protection**

5) Briefly describe how a buffer overflow can be used to force the victim computer to execute arbitrary code. See pages 60-66 of the course reader. You don't have to understand all the details in the course reader. It’s enough if you write a brief overview of how a buffer overflow attack works, describing how the stack is involved.

6 a) Briefly define what access lists and capabilities are.

b) Process P creates a file F and wants to give other processes the right to access this file. However, sometime in the future process P may decide that it wants to selectively revoke those access rights, so that certain processes can no longer access file F. How could you handle this revocation using access lists? How could you handle this revocation using capability-based protection?

c) Suppose we are interested in fine-grain protection, in which each procedure can operate on a strict “need-to-know” basis. Which mechanism (access lists or capabilities) would you recommend? Why?

d) Why is it desirable to incorporate certain operating systems security functions directly into hardware or microcode? Briefly discuss the hardware support needed for capability-based protection systems: describe the mechanisms for distinguishing capabilities from other data (see textbook section 14.5.3).

**Question about Operating System Structure**

7) Can parts of an operating system be implemented as processes? Can the kernel of an operating system be a process?

**Memory management review problem: does not have to be handed in**

(Review) When a process accesses a segmented memory, the OS/hardware must check that the segment offset is in range. Describe how information from the segment table is used to carry out this test. Also describe how the segment table can be used to store protection information.

[You do not have to hand in an answer. Study the answer provided with the assignment 6 solution.]