

## OPERATING SYSTEMS COMPREHENSIVE EXAM, SPRING 2006

This exam contains four questions of equal value. For each question, record your answer in a separate exam booklet. Answer all questions to the best of your ability.

### 1. Synchronization

Consider the following implementation of the semaphore wait and signal functions:

```
1. wait (s)
2.   sem s;
3.   {
4.     if (value (s) == 0) {           /* is value of s zero? */
5.       putq (s, thisproc ());      /* put this proc on wait queue */
6.       block (thisproc ());        /* this proc blocks and yields */
7.     }
8.     decr (s);                       /* decrement value of s */
9.   }
10.
11. signal (s)
12.   sem s;
13.   {
14.     incr (s);                       /* increment value of s */
15.     if (! emptyq (s)) {           /* if wait queue is not empty */
16.       unblock (getq (s));         /* get a process and unblock it */
17.     }
18.   }
```

incr (s)        increments value of semaphore s  
decr (s)        decrements value of semaphore s  
value (s)       returns value of semaphore s  
putq (s, p)     puts process p on semaphore s's queue of waiting processes  
getq (s)        removes a process from semaphore s's queue and returns it  
emptyq (s)      returns true if semaphore s's waiting process queue is empty  
thisproc ()     returns the currently running process  
block (p)       puts process p in blocked state and yields to another process  
unblock (p)     puts process p in ready state

When either of these functions are called, you may assume that the process running them will not be preempted. However, the process may voluntarily give up the CPU, as in the wait function.

(i) There is a bug in the implementation; what is the bug? Identify using the line numbers where the problem is, and describe how the problem may occur.

(ii) How would you fix the problem? Describe the minimal number of changes, using the line numbers as references. (You will only get credit if you determine the absolute minimal number of changes and what they are.)

## 2. Virtual Memory

(i) Sketch how virtual address translation works on a modern architecture using a hardware-managed translation lookaside buffer (TLB) and a monolithic operating system (e.g., Linux or Windows). Imagine that we are running Emacs as an application on the system. Start with the step where Emacs executes an instruction that makes a memory reference to a page that has been paged out to disk, and end with the step where the Emacs instruction finishes execution. It is sufficient to simply list the steps, a detailed discussion is not necessary.

(ii) A number of different operating system structures have been proposed over time, including how virtual memory is managed. Again consider the case where Emacs is running as an application on a system and causes a page fault. Select **two** (and only two) of the following recent systems:

- L4
- Xen
- Exokernel

For the two systems you selected, sketch how they handle a page fault by answering the following questions:

(a) What part of the system (whose address space) receives the initial page fault exception?

(b) What part of the system (whose address space) stores and manages application page tables?

(c) What mechanism does the system use to vector a page fault from (a) to (b)?

### **3. Distributed Services**

a) Describe some of the relative advantages of using multi-threaded, multi-process, and event-driven network services. Consider issues of portability to multiple platforms, performance, and simplicity of the overall code. Please briefly justify your position on the relative merits of each approach.

b) What are some of the challenges and techniques for making a remote procedure call transparent? Consider a network that may drop, reorder, or corrupt messages and end hosts that may fail at arbitrary points. What role do idempotent operations play in this space? What about marshaling/unmarshaling arguments?

### **4. Answer the following questions about Rosenblum and Ousterhout's "The Design and Implementation of a Log-Structured File System" paper.**

(a) What trend in hard disc performance was LFS designed to harness?

(b) In analyzing the performance of LFS, the authors proposed a new metric called "write cost." How was write cost defined? How did they determine the write costs of FFS and LFS? (You do not need to recall the exact formula for LFS, but should explain the basic approach.)

(c) Why does a "hot-and-cold" file access pattern cause the greedy cleaner to clean segments at a higher average utilization?

(d) What was the purpose of the "cost-benefit" cleaning policy?