

# CPS 210 Qualifying Exam

January 9, 2002

Answer all questions. Allocate your time carefully. Your answers will be graded on content, not style. Any kind of pseudocode is fine as long as its meaning is clear. You have 180 minutes.

**Problem 1.** Show how to emulate general (counting) semaphores safely using binary semaphores.

**Problem 2.** You are to implement the core of a process manager for a multiprogrammed operating system kernel. The process manager is a class called *Process* that tracks parent/child/sibling relationships among processes as needed, and coordinates the process-related kernel services (e.g., the *fork*, *exit*, and *wait* system calls in Unix). Your solution must implement three key methods to operate on a *Process* object *p*.

*p->Birth(Process\* parent)* registers this process *p* as a child of its parent.

*p->Death(int status)* indicates that this process *p* has exited with the specified status.

*int status = p->Join()* waits for *p* to exit (i.e., to call *Death*), and returns *p*'s exit status.

*To help guide your solution.* Write only what you need to implement these three methods, assuming the following context. Each *Process* object *P* has an associated thread that animates the process. The thread bound to *P* calls *Birth* and *Death* on *P* as part of the implementation of the system calls for process creation (e.g., *fork*) and destruction (e.g., *exit*). *Join* on *P* may be called only by the thread bound to the parent of *P*. You do not need to create these threads or enforce these restrictions; they are intended to simplify the problem.

*Required synchronization.* *Death* on *P* blocks until (1) the children of *P* have exited, and (2) the parent of *P* has exited or has executed a *Join* on *P*. The intent is that the exit status and process object for *P* may be safely destroyed after *Death* returns. Be sure that your solution is free of race conditions, deadlock, and dangling references. To simplify matters I suggest that you use a single mutex and condition variable for your synchronization.

You may assume any reasonable set of list primitives if their meaning is clear, and you may add other methods for *Process* as needed to support the functions of *Birth*, *Death*, and *Join*.

**Problem 3.** Sketch rough graphs (“back of napkin”) illustrating the relationships among the following quantities. Note any important assumptions underlying your analysis.

- a) Page fault rate as a function of page size. What page sizes are typical for current processors and operating systems?
- b) Disk access latency as a function of rotation speed. What rotation speeds and access latencies are typical for current disk drives?

- c) Peak disk read bandwidth and read latency as a function of read block size (request size). What peak bandwidths are typical for current disk drives?
- d) Peak read bandwidth, I/O operation throughput (IOPS), and access latency as a function of the number of drives in a RAID storage unit.
- e) Utilization, response time, and throughput as a function of request arrival rate for a queueing service center.
- f) I/O block cache hit ratio as a function of cache size, for random references uniformly distributed across the I/O block space.
- g) I/O block cache hit ratio as a function of cache size, for sequential references.
- h) Network bandwidth as a function of per-message CPU overhead.

**Problem 4.** Most modern processors and operating systems enforce protection boundaries that prevent programs from interfering with one another or with the operating system, and that allow the operating system to securely mediate and monitor all accesses to shared resources in accordance with a security policy. Briefly summarize the most important mechanisms underlying OS protection and security. Your answer should be comprehensive and precise, but succinct.

**Problem 5.** Distributed file systems use caching to improve performance and scalability. NFSv3 uses block caching with validation, AFS/Coda uses whole-file caching with callbacks, and other systems use block caching with file leases (e.g., NQ-NFS and NFSv4). Suppose that there is a fixed universe of files, each client generates requests at a fixed rate, client failures occur with some fixed probability per time unit, and all files have a fixed probability of reference for each request. How will each of these schemes behave as the number of clients increases and/or as their distance from the server increases? Explain your answer.