CPS 210 Final Exam

May 8, 1998

Answer all questions. Allocate your time carefully. Your answers will be graded on content, not style. For code answers, any kind of pseudocode is fine as long as its meaning is clear. For "essay" questions please do not waste any words. 300 points total. You have three hours.

Part 1: The Basics

1. Two approaches to synchronization are semaphores and (mutex, condition variable) pairs. These facilities are equivalent in power: any synchronization problem that can be solved with either facility can also be solved with the other. Prove this equivalence by showing how to implement each facility using the other. (40 points)

2. Most operating systems use variants of Least Recently Used (LRU) as a replacement policy for the file block cache (buffer cache) and the virtual memory system (page cache).

   a) What is the rationale for choosing LRU as the replacement policy in each case? (5 points)

   b) Outline how to implement LRU replacement for the file block cache. Consider the handling of dirty blocks. (10 points)

   c) In general, a direct implementation of LRU is not practical for the virtual memory page cache. Explain why this is so. (10 points)

   d) Explain how to approximate LRU replacement for the virtual memory page cache using the FIFO-with-second-chance algorithm. What parameters does the algorithm use to balance the quality and cost of the LRU approximation? (15 points)

3. Most operating systems include a timer or alarm facility to allow threads to pause (sleep) for some period of time. In this problem you are to implement two internal kernel procedures, *AlarmPause* and *AlarmWake*, as the basis for alarms. Alarms will be available to user processes through a *Pause (**int** howlong)* system call, which puts the calling process to sleep for *howlong* time units. *AlarmPause(**int** howlong)* is called from the *Pause* system call handler, and sleeps until the pause time has expired. *AlarmWake()* is called from the clock interrupt handler to wake up any threads whose periods have expired while sleeping in *AlarmPause*. Your solution should include any synchronization needed for *AlarmWait* and *AlarmPause* on a shared-memory multiprocessor. You may assume common primitives (e.g., linked lists), but explain any primitives whose meanings are not obvious. (35 points)

4. Explain in detail what happens when a thread invokes the *Pause* system call in problem 3, and how it exits the kernel after *AlarmPause* returns. Be sure to explain any transitions of the thread state (e.g., ready, running, blocked) and changes to the mode, space, or context of the CPU. (40 points)

Part 2: Topics

5. Explain why small writes can create performance problems in RAID systems. Hartman and Ousterhout have suggested that the "small write problem" can be resolved by using a log-structured file system (LFS) on the RAID. (This idea is used in the Zebra and xFS cluster file systems and NAC's commercial network file server). Explain how the properties of LFS can improve the performance of small writes in RAIDs. (30 points)

6. In the Mach microkernel system, most OS services — including most of the Unix system call interface — are provided by server processes running in user mode. Briefly argue for and against this structure. (30 points)

Part 3: Continuations

7. In most systems, the state of a blocked kernel operation is represented largely within the kernel stack of a thread (or process). Draves and Bershad proposed restructuring kernels to use *continuations* as an alternative. In their scheme, the state of a blocked kernel operation may be represented as a pointer to a "continuation" procedure and a list of arguments. When a thread blocks, its kernel stack is discarded; after waking up, the thread continues by calling its continuation on a freshly allocated kernel stack, passing the specified arguments.

   a) Continuations can yield a significant space savings in the kernel. This is true even though the essential state normally kept on the kernel stack must be held somewhere else in memory when the thread is blocked. Explain. (5 points)

   a) Kernel code must be significantly restructured to use continuations. Show how to restructure your *Alarm* (problem 3) to use continuations. Also outline any changes to the underlying scheme for sleep/wakeup or entering/exiting the kernel. (15 points)

   c) Draves and Bershad found that the Mach microkernel is well suited to continuations. This is because Mach systems tend to have larger numbers of blocked threads, yet these threads tend to be blocked at a small number of points in the kernel. Explain. (10 points)

8. The Global Memory Service uses a variant of continuations to handle RPC call completions, as described in the paper *Cheating the I/O Bottleneck*. In GMS, continuations are used to implement "nonblocking RPC" rather than to reduce kernel memory usage. In this context, continuations are an alternative to the the I/O daemons used for a similar purpose in NFS.

   a) Explain the role of nonblocking RPC in allowing GMS and NFS to access files at the speed of the underlying hardware (disk or network). (25 points)

   b) Explain how NFS implements nonblocking RPC using I/O daemons, and outline any limitations of this scheme. (15 points)

   c) Explain how GMS implements nonblocking RPC using continuations, and outline any limitations of this scheme. (10 points)