CPS 210 Midterm Exam
Spring 2009

Answer all questions. Please do not waste any words: answers are graded on content, not style. A short paragraph and/or a fragment of pseudocode is sufficient to answer any of the questions. Where code is required, any kind of pseudocode is fine as long as its meaning is clear. You may assume common primitives (e.g., linked lists and standard synchronization primitives), but explain any primitives whose meanings are not obvious. Each question is worth 25 points. You get 25 points for writing your name on every page, for a total of 200 points.

Part 1: Concurrency and Synchronization

1. Operators are standing by. The customer service line at MegaMurk staffs a phone bank to answer questions from customers about its products. Customers calling in are put on hold, where they wait for the next available customer service advocate to answer their call “in the order it was received”. The economy is in a downturn, so business is sometimes slow: if there are no customers waiting, available advocates just wait around for the phone to ring.

   (a) Write pseudocode to synchronize the customers and advocates by implementing two procedures: customer() and advocate(). A customer calls customer() when it arrives to request service. An advocate thread calls advocate() whenever the advocate is available to serve a customer. Synchronize using mutexes and condition variables. (15 points) Extra credit: do it again using semaphores.

   (b) If you get part (a) wrong, would you expect that the Eraser race detection tool (or MUVI) could detect the problem? Why or why not? (10 points)

2. RISCy spinlocks. The Alpha and MIPS 4000 processor architectures have no atomic read-modify-write instructions, i.e., no test-and-set-lock instruction. Atomic update is supported by pairs of load-locked (LDL) and store-conditional (STC) instructions. Executing an LDL Rx, y instruction loads the memory at the specified address (y) into the specified general register (Rx), and holds y in a special per-processor (i.e., per CPU core) lock register. STC Rx, y stores the contents of the specified general register (Rx) to memory at the specified address (y), but only if y matches the address in the CPU’s lock register. If STC succeeds, it places a one in Rx; if it fails, it places a zero in Rx. Several kinds of events can cause the machine to clear the CPU lock register, including traps and interrupts. Moreover, if any CPU in a multicore/multiprocessor system successfully completes a STC to address y, then every other processor’s lock register is atomically cleared if it contains the value y.

   Show how to use LDL and STC to implement safe busy-waiting, i.e., write spinlock Acquire() and Release() primitives using “pseudo” machine instructions.

Part 2: Resource Allocation

3. Fairness of Lotteries. Lottery scheduling is primarily concerned with “fair” allocation of the CPU among contending threads. Summarize the “fairness” properties assured by lottery scheduling, and the factors that influence the amount of CPU time that a thread receives.

4. Waking up fast. CPU schedulers in modern operating systems may boost the priority of a thread temporarily if it blocks and relinquishes the CPU before its previous quantum expires. For example, Windows
boosts priority of a thread that waits on a lock or an event such as I/O completion. Is this policy “unfair”? What purpose does it serve?

5. **Handoff.** OS-X has an internal kernel primitive for a running thread to yield or “hand off” control of the CPU to some other thread that the yielding thread designates. The receiver of the handoff may be in a different task (addressing domain or process). The receiver inherits the scheduling attributes of the yielding thread, including its quantum and priority. Why might such a handoff primitive be useful? How might it be used, and under what conditions?

Part 3: Protection

6. **Don’t be evil.** Safedrive and Nooks (*Shadow Drivers*) provide isolation mechanisms that “sandbox” extensions (e.g., device drivers) to prevent them from damaging the rest of the operating system kernel. In both systems, the isolation mechanisms are designed to protect against extensions that are “buggy but not malicious”. What limitations of these mechanisms make them unsuitable to defend against “malicious” extensions that deliberately attempt to subvert or crash the kernel?

Part 4: Your Project

7. **Pressure points.** In a short paragraph, outline your proposed semester project for CPS 210.

*Have a great break!*