CS 354 (Park) **Midterm** **Oct. 19 (Mon.), 2015**

Remarks: Keep the answers compact, yet precise and to-the-point. Long-winded answers that do not address the key points are of limited value. Binary answers that give little indication of understanding are no good either. Time is not meant to be plentiful. Make sure not to get bogged down on a single problem.

PROBLEM 1 (36 pts)

(a) What are the responsibilities of upper and lower halves of modern kernels? For each half, give an example of XINU code (i.e., function) that performs a specific task. Where does the scheduler (resched() in XINU) fit in? Explain your reasoning.

(b) What hardware support is provided by modern computing systems to help a kernel achieve isolation/protection? In x86, how are the 2-bit IOPL field of EFLAGS and the CPL field of the CS register used for isolation/protection? How are the values of these fields configured in XINU, and why? How are they configured in Linux and Windows kernels which provide isolation/protection?

(c) Why are counting semaphores preferred over interrupt disabling for achieving mutual exclusion in operating systems? As a practical matter, when is interrupt disabling acceptable as a feasible solution? Counting semaphores are viewed as software primitives, in contrast to tset and interrupt disabling which invoke direct hardware support. Are counting semaphores, indeed, a purely software feature? Explain.

PROBLEM 2 (40 pts)

(a) Explain the rationale behind how UNIX Solaris dynamically adjusts TS process priorities to achieve equitable sharing of CPU cycles by app processes. Since the Solaris scheduler is but a heuristic, it cannot guarantee that some processes may not receive CPU cycles for prolonged periods (i.e., starve). What safety mechanism does Solaris implement to prevent starvation? Before Linux adopted a fair scheduling approach using CFS a few years back, how did it differ in its approach from many other kernels including Solaris? (Hint: Think of the 2x2 table that describes how priority and time slice are qualitatively adjusted.) While reasonable, what issue did the Linux approach encounter?

(b) XINU's ctxsw(), which is invoked by the scheduler resched(), is tasked with checkpointing the state of the current (i.e., old) process that is context switched out, and retoring the state of the (new) process that is context switched in. What is the sequence of steps performed—specify what each step does in words (i.e., no need to write assembly code)—to save the state of the old process? Restoring the state of the new process from its previously checkpointed state proceeds in reverse order, with the exception of two steps whose order does not. What are these two steps? Describe an example of what can go wrong (i.e., we have a kernel bug) if the exception is not followed.

PROBLEM 3 (24 pts)

In addition to hardware support, software support in the form a kernel space run-time stack is needed to achieve isolation/protection. That is, using a process's user space run-time stack when executing kernel functions during a system call allows a hacker to circumvent isolation/protection. Describe a specific example where a hacker can by-pass isolation/protection by employing two app pocesses—one of which makes a system call—that share their user space memory including their user space run-time stacks.

BONUS PROBLEM (10 pts)

Producer/consumer buffers used for IPC/IO by operating systems use two semaphores to achieve orderly access. How are the two semaphores initialized, and why? A single counting semaphore with its initial value set to 1 may also be used to achieve orderly access through mutual exclusion of a producer/consumer buffer. Why is the latter considered a less desirable solution?