CS 354 (Park) **Midterm** **March 8 (Fri.), 2013**

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) Xinu system calls begin by disabling all interrupts and end by re-enabling them before returning. What purpose does this serve? In the case of the `create()` system call which updates (i.e., writes to) the process table, what may go wrong if interrupts are not disabled? Why is interrupt disabling not a method preferred by modern operating systems?

(b) What are the hardware features of processors, including x86 CPUs, that help operating systems such as UNIX, Linux and Windows achieve isolation/protection? What role do system calls play for realizing isolation/protection?

(c) In x86 Xinu, when context-switching out a process, what pieces of information does `ctxsw()` save? Most of the pieces are saved on the process's run-time stack, however, the stack pointer is recorded in the process table. Why is this a reasonable strategy as opposed to, say, saving everything in the process table?

PROBLEM 2 (32 pts)

(a) UNIX Solaris, among other operating systems, implements time-shared (TS) process scheduling by changing the priority and time slice of a process based on whether it makes a blocking system call or depletes its allocated time slice. How are the adjustments qualitatively made and what is the rationale behind them? Are there potential issues? How does Solaris go about preventing starvation?

(b) What are the pros/cons of implementing multithreading entirely in user space versus with kernel support? Which approach may be more suited for today's multicore processors?

PROBLEM 3 (32 pts)

(a) Kernel code must be extremely efficient, preferably incurring only constant overhead (i.e., constant time), and, if not, logarithmic overhead. What is the overhead of Xinu's default scheduler used in Lab1? What is the overhead of TS scheduling which uses a multi-level feedback queue? Explain how you arrive at the overhead estimation.

(b) When aiming to achieve mutual exclusion by guarding critical section code of processes that operate on shared data structures, what is the advantage of using `tset` over interrupt disabling, and what is the advantage of using semaphores over both `tset` and interrupt disabling? Might using semaphores have a drawback? Can semaphores be considered a purely software-based solution or is there a hidden hardware dependence? Explain.

BONUS PROBLEM (10 pts)

In modern operating systems, it is not enough that a process has a private run-time stack to manage user space function calls. To achieve isolation/protection, a process must also be provided with a kernel space run-time stack that is used when executing system calls. Why is this the case?