CS 354 (Park) **Midterm 1** **Feb. 24 (Fri.), 2017**

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 (45 pts)

(a) What are the responsibilities of the upper and lower halves of XINU and modern kernels such as Linux/UNIX and Windows? In XINU, what is the first task that is performed after entering the upper or lower half? In our x86 Galileo backends, what change in hardware state (think of register flags) does this software action result in? What is the main difference of XINU's upper half when compared to Linux/UNIX and Windows? Name three XINU kernel functions: one belonging to the upper half, another to the lower half, and a third belonging to neither.

(b) How do kernels determine if a process is CPU- or I/O-bound? In Solaris UNIX, what happens to a TS process when it is deemed to be CPU-bound? What happens to an I/O-bound process? What is the rationale behind these actions? In the fair scheduling implementation discussed in class (and part of lab3), how are I/O-bound processes treated when compared to CPU-bound processes? Explain your reasoning.

(c) The memory layout of XINU on a backend machine after it has created a number of concurrent processes is different from that of Linux/UNIX and Windows. In what way is this so? How is this related to the heavy-weight/light-weight process model? In principle, a process in Linux that makes a very large number of nested function calls can face what type of run-time problem? In XINU, a process that makes a comparatively moderate number of nested function calls can cause what type of problem? What x86 hardware support that is currently "disabled in software" (recall our discussion of segmentation) may be utilized to mitigate the problem in XINU?

PROBLEM 2 (34 pts)

(a) What are the hardware support features provided for achieving isolation/protection in modern operating systems? What software feature is required? If this software feature is not implemented, what can an attacker do to bypass isolation/protection and take over a computing system? Describe the steps involved. Is super user/root the same concept as kernel mode in modern kernels? When might super user/root become relevant in XINU?

(b) When XINU performs context-switching on our x86 backends, what pieces of hardware state does it save on a process's run-time stack? What information is also saved elsewhere, and why? When a function calls another function, the compiler makes sure that the return address (i.e., EIP in Galileo x86) is checkpointed so that when the callee returns the caller can resume where it left off. In XINU's context-switching code where we are switching from one process to another—a much more drastic change—the EIP of the process being context-switched out is not saved. Why is that so? Is this specific to XINU?

PROBLEM 3 (21 pts)

What is the overhead (i.e., time complexity) of XINU's static priority scheduler? State your answer with respect to the two main operations: enqueueing and dequeueing a ready process. Why can we implement TS process scheduling of Solaris UNIX in constant time with the help of the multilevel feedback queue data structure? Why is the overhead of Linux's fair scheduler logarithmic? Practically speaking, is this a major weakness when compared to constant time schedulers? Explain your reasoning.

BONUS PROBLEM (10 pts)

What is the role of the clock interrupt handler in the lower half of a kernel (including XINU) with respect to process scheduling? Describe in words what it does to assist in process scheduling. When implementing fair scheduling in lab3, do you still require the help of the clock interrupt handler? Discuss your reasoning.