CS 354 (Park) **Midterm** **March 9 (Mon.), 2020**

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 (60 pts)

(a) What is the role of the upper half of a kernel? What is the role of the lower half? Where does the scheduler fit in? Describe an example of how XINU's resched() is reached using nested function calls via the upper half. Do the same for the lower half. What is the role of the null process in XINU?

(b) How is XINU's system call implementation different from that of Linux/UNIX and Windows? What were the three software components in our reimplementation of XINU's getpid() and getprio() system calls in lab2? What were the roles of the three components? What two features were lacking in our implementation that made it fall short of how system calls are implemented in Linux/UNIX and Windows?

(c) For the two operations of inserting and removing a process from XINU's ready list, what are their overheads, and why? What are the corresponding overheads in the case of UNIX Solaris that uses a multilevel feedback queue, and why? What is the criterion used by Solaris to determine if the priority of a process should be increased or decreased? What is the the "safety net" used by Solaris scheduling?

PROBLEM 2 (40 pts)

(a) In XINU, a process is created using the create() system call and made ready using resume(). What specific content is pushed onto the stack of a newly created process, in what sequence, so that when it becomes current for the first time and is context-switched in, the process jumps to the first instruction of the function to be executed by the process? What should the value of IF flag for EFLAGS pushed onto the stack be, and why? Suppose the function to be executed by the newly created process completes. How does XINU make the process jump to epilogue code userret() upon completion? What happens when userret() is executed?

(b) In asynchronous IPC with callback function, a receiver process registers with the kernel user code—i.e., callback function—that is to be invoked if a message arrives for the receiver process. Consider the case where the receiver process is sleeping when a message arrives. Describe the steps that the kernel must perform to facilitate execution of the callback function while preserving isolation/protection. Will the callback function be executed right after the message arrives? Explain your reasoning.

BONUS PROBLEM (10 pts)

We noted that an operating system is not a process. What is a technically accurate description of an OS as a software system? How is this related to Problem 1(a)?