CS 354 (Park) **Midterm** **March 4 (Fri.), 2022**

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 (40 pts)

(a) What four hardware support features are necessary to efficiently implement isolation/protection? Why are these features not sufficient, and what additional software feature is needed for isolation/protection? In x86 how is XINU's configuration of GDT different from that of Linux and Windows? What is the reason for XINU's GDT being fundamentally different?

(b) A simple method for implementing fair scheduling in the sense of equal share seems to be round-robin scheduling using a FIFO queue. Explain what the overhead of this method is in terms of enqueue and dequeue operations. Why is this method not considered adequate for CPU scheduling in operating systems? What is the modified goal of "fairness" for real-world processes? How does UNIX Solaris try to meet the modified goal, and why is its overhead constant?

PROBLEM 2 (40 pts)

(a) Explain what it is means for XINU's receive() to be a blocking system call. Being a system call, receive() is part of the kernel's upper half. Does any of receive()'s operation depend on XINU's lower half? Does the system calls sleepms(), which is also part of the upper half, depend on XINU's lower half? Explain.

(b) Trapped system call implementation in x86 using the int software interrupt entailed three separate software layers. Using XINU system call resume() explain what they are and their respective roles. In lab2 we implemented trapped versions of XINU's getprio() and chprio() system calls. In what two ways did our implementation fall short of achieving full isolation/protection?

PROBLEM 3 (20 pts)

When discussing context-switching in (i.e., making current) a process in x86 XINU, we considered two cases: a process to be ontext-switched in (i) ran before but was context-switched out, or (ii) is a newly created process that will run for the first time on the CPU. How did case (ii) impact the code of ctxsw()? What stack manipulation was performed by system call create() to support case (ii)?

BONUS PROBLEM (10 pts)

Describe a workload scenario whereby CPU-bound processes may suffer under starvation. What method is used by UNIX Solaris to mitigate this problem?