CS 354 (Park) **Midterm** **Mar. 12 (Wed.), 2025**

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 makes the objectives of scheduling in operating systems different from scheduling in other domains? How are priority and time slice qualitatively assigned to processes in UNIX Solaris TS to achieve the above objectives? How does Solaris deal with starvation? If an operating system were to manage a workload comprised of homogenous processes—e.g., all processes predominantly require CPU cycles—what scheduler would be well-suited that is both simple and efficient? Is this scheduler well-suited for mixed workloads? Discuss your reasoning.

(b) What is a simple scenario of app processes and semaphores discussed in class that results in a deadlock? Describe the sequence of events that lead to deadlock. How do we detect deadlock in a resource graph? What is the overhead (i.e., time complexity)? Operating systems do not consider deadlock detection an essential service that must be provided to applications. What is the underlying rationale? As a kernel programmer what is a method that, if followed, guarantees that a deadlock will not arise?

PROBLEM 2 (40 pts)

(a) When checkpointing a process that is being context-switched out, XINU's ctxsw() kernel function pushes EBP, EFLAGS, then the 8 general purpose registers. When context-switching in a process, ctxsw() pops the 8 general purpose registers but reverses the sequence of restoring EFLAGS and EBP. Explain the reason behind restoring EBP first before EFLAGS is restored. ctxsw() is called by XINU's scheduler resched(). Does ctxsw(), upon completion, always return to resched()? Explain.

(b) What is the main advantage of interrupt disabling as a process coordination/synchronization mechanism, what is its main disadvantage? How does `tset` attempt to mitigate the disadvantage of interrupt disabling? What is the main disadvantage of `tset`? How do counting semaphores alleviate the main disadvantage of `tset`? In what way are producer/consumer buffers that utilize two semaphores an optimization over mutual exclusion achieved using a single semaphore? Are counting semaphores as implemented by XINU's wait() and signal() system calls pure software primitives that do not require hardware support? Explain your reasoning.

PROBLEM 3 (20 pts)

How are XINU system calls fundamentally different from system calls in x86 Linux and Windows? How is XINU's GDT configuration different from that of Linux/Windows? In lab2 we redesigned XINU system calls to approach the design of traditional Linux/Windows system calls. What were the two main simplifications—essential for isolation/protection in commodity operating systems—that afforded ease of implementation by avoiding additional complications? Suppose you were asked to redesign x86 XINU's create() system call so that it resembles the Linux clone() (or fork()) system call where a newly created process starts its execution in user mode. Describe the main elements of your changes to create() and ctxsw() to achieve this objective. A high-level sketch is sufficient without providing pseudo-code. (*Hint: Consider how create( ) and ctxsw( ) handle newly created processes, and our discussion of untrapping for implementing isolation/protection in x86.*)

BONUS PROBLEM (10 pts)

What is asynchronous IPC with callback function? How is its design and implementation affected by the need to preserve isolation/protection?