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 not 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 are the pros/cons of RMS vs. EDF for supporting real-time scheduling of periodic tasks in operating systems? What are the main implementation challenges for supporting real-time scheduling of video streaming in real-world systems? Why is XINU not well-suited for supporting real-time tasks?

(b) What were the roles of the three software layers in the trapped system call implementation of XINU system calls in lab2? Why did reusing legacy XINU system calls in the design not cause issues despite XINU system calls disabling interrupts upon entry and reenabling interrupts upon exit? The system calls implemented in lab2 were non-blocking. Would implementing trapped blocking system calls, say sleepms(), have resulted in additional complications and resultant coding changes? Explain your reasoning.

PROBLEM 2 (40 pts)
(a) For what system environment is interrupt disabling suited for achieving mutual exclusion? For what environment is test-and-set suited? What performance gain does a software solution using semaphores aim to achieve over test-and-set? The x86 instruction xchg performs an atomic swap of the contents of its two operands. Describe in words how xchg can be used to code a function, int tset(int *), that implements test-and-set where the argument specifies the address of a semaphore variable in main memory.

(b) Explain why the overhead (i.e., time complexity) of multilevel feedback queue scheduling is constant. Why is overhead of Linux CFS logarithmic? Linux CFS uses virtual runtime which measures CPU usage to determine which process to run next. What two additional issues must be handled heuristically as a result of using CPU usage as indicator of priority in process scheduling? In multiprocessor/core scheduling load in the form of processor utilization is a primary balancing criterion. What is a second balancing criterion that is relevant for effecting more CPU time allocation to high priority processes over low priority processes? What property of real-world workloads needs to be considered when performing process migration, and why?

PROBLEM 3 (20 pts)
How does XINU handle context-switching in a process that has never executed on a CPU differently from one that has executed before and was context-switched out? What is the role of create() in this? Why does ctxsw() restore EBP first before updating EFLAGS when context-switching in a process? Suppose lab2 were extended so that newly created processes start execution in user mode and transition to kernel mode when executing upper or lower half kernel code. Keeping in mind that the esp0 and ss0 fields of the TSS data structure pointed to by the TSS entry in GDT (e.g., x86 Linux and Windows) enable hardware supported stack switching when transitioning between user mode/kernel mode, what modification of ctxsw() is needed to save the context of an old process and change to the context of a newly created process? What modification to create() is needed so that a newly created process starts executing in user mode? Is any modification needed when the process to be context-switched in is not a newly created process (i.e., has executed before)? Explain.

BONUS PROBLEM (10 pts)
Suppose we were to implement isolation/protection in XINU by utilizing x86 hardware support for stack switching between user mode and kernel mode. Noting that XINU executes lower half code by borrowing the context of the current process, do we need a third stack—separate from per-process user and kernel stacks—to use as kernel stack during interrupt handling? Explain your reasoning.