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) Modern kernels may be subdivided into upper and lower halves. What are the roles of the two halves? Who runs each half in XINU? Where does the scheduler fit in the overall kernel organization? In the case of XINU, give an example scenario that leads to calling resched() by the upper half. Do the same in the case of the lower half.

(b) What are the chief challenges of implementing RMS/EDF support for compressed real-time video streaming in operating systems? What is the advantage of EDF over RMS? What is its drawback? Is XINU well-suited for implementing real-time scheduling? Explain.

(c) One approach to “fair sharing” of CPU cycles in kernels is using round-robin scheduling where a scheduler sweeps over ready processes, allocating the same time slice to each process. What is the overhead (i.e., time complexity) of this approach? Why is round-robin scheduling not considered sufficient for real-world workloads? What is a popular approach followed by modern kernels such as UNIX and Windows?

(d) Suppose two processes A and B use a special tset instruction to achieve mutual exclusion of their critical section code. On a uniprocessor system, assume A executes tset first but is context switched out (e.g., due to quantum depletion) in the middle of its critical section. What happens when B is context switched in by the scheduler? Is the tset based approach meaningful in a uniprocessor system? What about a multicore system? Explain.

PROBLEM 2 (36 pts)

(a) Multithreading support may be implemented entirely in user space. What are the pros/cons compared to multithreading implemented with kernel support? In the former, what is involved in saving the state of a user space thread during a context switch? Can we use XINU’s ctxsw() code to checkpoint the state of a user space thread in x86? What impact, if any, does the sensitive instruction popfl have on restoring the state of a user space thread?

(b) What is asynchronous IPC with callback function? What issues must be dealt with when providing this service as part of a modern kernel? What is one approach for handling the issues? What drawback does it have?

(c) Suppose a hypervisor implements full virtualization and emulates two guest kernels (e.g., Linux/Windows). How is “context switching” between guest kernels related to context switching between a pair of processes within a native kernel running directly over hardware? Point out the similarities and differences. What happens if a guest kernel attempts to execute a privileged instruction such as cli in x86? Is the IF flag of EFLAGS set to 0? Explain.

PROBLEM 3 (24 pts)

(a) What is a deadlock? Give an example using 2 processes and 2 semaphores. What is the overhead associated with deadlock detection? How do kernels approach the problem of deadlocks of user processes? Why is this approach justified? How may a kernel handle “internal deadlocks,” i.e., deadlocks involving processes running kernel code in kernel mode?

(b) What are the hardware and software support features needed to achieve isolation/protection? In x86, what role do CPL and IOPL play? How does XINU configure these values? How are XINU system calls different from UNIX/Linux/Windows system calls in their design and implementation? Why do system calls incur more overhead than regular function calls?

BONUS PROBLEM (10 pts)

Based on the material covered thus far, list four simplifications—sorted in decreasing order of their importance based on your evaluation—that XINU (the initial version distributed at the start of the semester) implements when compared to modern kernels such as UNIX/Linux and Windows. Explain your choice of the most important simplification.