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) XINU has a null (or idle) process, as do many other operating systems. What is the role of the null process? Is it a necessary feature? Are there computing systems where kernels may not utilize a null process?
(b) How does a modern kernel (e.g., UNIX/Linux, Windows) that provides isolation/protection and runs on a x86 machine prevent user mode processes from executing privileged instructions? Since XINU does not implement isolation/protection, what is the privilege level configuration that makes it so?
(c) Deadlock detection is not a service provided by typical kernels. What are the two main reasons for this design decision?
(d) A multi-level feedback queue is a useful data structure for implementing TS scheduling. In what sense is the scheduling overhead constant (i.e., $O(1)$)? Suppose we have a scheduler that does not implement round-robin scheduling among equal priority ready processes. Instead, it chooses one that has the least memory footprint (uses the least amount of memory). What is the resultant scheduling overhead assuming a multi-level feedback queue is still used?

PROBLEM 2 (36 pts)
(a) What are the benefits and main drawback of user space multithreading support over its kernel space counterpart in a single-CPU/core system? What must a user space multithreading solution do to overcome the drawback? Which approach is favored in a multi-core system?
(b) Suppose a kernel uses low-level (i.e., basic or hardware independent) memory management to dynamically allocate/free memory for processes as in XINU. Suppose the free memory region starts at location $lowmem$ and ends at $lowmem + 1000$. Using XINU’s $getmem()$ and $freemem()$ (the first argument of $freemem$ specifies the beginning address of memory to be freed), give an example of system calls invoked by a process that results in external fragmentation. What is memory compaction and how significant is its overhead? What other critical complication does compaction introduce?
(c) Implementing RMS or EDF scheduling of periodic tasks, in particular, real-time video streaming, is challenging despite it being a “killer app.” What are the two application related features in the period task model that make it so? What are some kernel related features that make guaranteeing hard deadlines difficult? What design feature makes XINU not well-suited as a real-time kernel to implement RMS/EDF?

PROBLEM 3 (24 pts)
(a) We discussed asynchronous I/O with callback function in the context of interprocess communication (IPC). In XINU where isolation/protection is not a concern, what is the simplest design solution? In a kernel that provides isolation/protection, when and by whom should the receiver’s callback function be executed to preserve isolation/protection? What complication arises in the implementation?
(b) Suppose Linux runs as a guest kernel under full virtualization on x86. Suppose an application process in the Linux kernel executes code that does not contain privileged instructions (which is the norm) and makes no system calls (unusual). What is the emulation/virtualization overhead of running the app process in the Linux guest kernel over a hypervisor in x86? That is, compared to running Linux directly over x86 hardware without virtualization. What is the overhead if the process does make system calls but they do not contain privileged instructions? What happens if a system call contains privileged instructions, say, $cli$? What complication is introduced by sensitive instructions?

BONUS PROBLEM (10 pts)
Both EDF and Linux’s CFS incur a logarithmic scheduling overhead. Why is that so? In EDF a ready process with the earliest deadline is chosen to run whereas in CFS a ready process with the least CPU time consumed is chosen. We discussed “running a highest priority ready process” as the scheduling invariant enforced by most operating systems. Although on the surface they may seem different, why can EDF and CFS be viewed as being compliant with the scheduling invariant?