

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) Today's x86 PCs support both real mode and protected mode, the former for backward compatibility with pre-286/386 legacy applications. Which mode does Xinu use? How can you determine the answer by looking at the source code? What does "system calls in Xinu are procedure calls" mean in this context?
- (b) What is the role of the programmable interval timer (PIT) in desktop operating systems such as Linux? What is the role of the real-time clock (RTC)? Could their roles be reversed? Why is the interval timer functionality of PIT not actively utilized by the kernel?
- (c) Desktop operating systems employ both process priority and time slicing to achieve equitable sharing of CPU among user processes. What is the fundamental approach followed by operating systems such as Solaris and Windows XP? From a kernel implementation perspective, what is the overhead associated with realizing the approach?
- (d) A null (or idle) process is implemented by operating systems to achieve the scheduling invariant: there is always a ready process to schedule, even if all user and system processes are blocked. Supposing there is no special hardware support, how would you modify Xinu so that a null process is not employed? From a performance perspective, how does your "no null process design" fare against Xinu's existing design?

PROBLEM 2 (30 pts)

- (a) What are internal and external fragmentation, and which does paging try to reduce? Does segmentation—in the sense of variable-length pages—have any advantages over paging with respect to fragmentation? What is multi-level paging, and why is it a necessity in today's desktop systems? Given the multiple levels of indirection introduced by paging, how is effective memory access time kept low?
- (b) A deadlock between two processes p and q can arise if p acquires a semaphore or lock R , q acquires S , then p waits on S while q blocks on R . From a kernel service design perspective, is providing deadlock detection service for user processes an essential component? Explain your reasoning. What is the preferred method for preventing deadlocks inside the kernel? Illustrate the method using the above example.
- (c) Xinu employs interrupt disabling/restoring to prevent critical sections inside system calls from being disrupted by interrupt handling routines. Suppose that critical sections inside system calls containing accesses to shared data structures are protected by semaphores instead. What complications can semaphores introduce in relation to interaction with interrupt handlers that do not exist with interrupt disabling? How should they be handled? Which method—interrupt disabling or semaphores—is preferable and why?

PROBLEM 3 (30 pts)

- (a) Describe, step-by-step, the worst-case scenario that can arise with respect to overhead in a paging system, triggered by a single memory access. Can the process that issued the memory access be switched out as a consequence? How many disk I/O operations can ensue? How many memory accesses can result in total? What are things that a kernel designer can do—besides improving the page replacement algorithm—to try to reduce paging cost? In the context of page replacement, what is Belady's page replacement "algorithm," and why is it not implementable in real systems? Can LRU be accurately and efficiently implemented without hardware support? Why, in practice, is LRU intimately related to Belady's criterion? How are first and second chance page replacement related to LRU?
- (b) What is the motivation behind splitting an interrupt handler of an I/O device into two parts: top half and bottom half? What is the relation of *upper* half and *lower* half of a kernel to top and bottom halves of device drivers (besides being not the best names ever invented)? What are two typical ways by which bottom halves are implemented? What are their pros/cons? Is it safe for a bottom half routine to make a blocking call such as `wait`? When the

lower half reads from a shared producer/consumer buffer written by the *upper* half, how can synchronization using a counting semaphore *sem* be affected such that the lower half does not need to make a blocking call? Show the code segment implemented by the reader and writer.

BONUS PROBLEM (10 pts)

What are the pros/cons of microkernel vs. monolithic kernel design? Among the the factors to consider are speed and modularity. How would you categorize Xinu? Suppose that you are given the responsibility of designing a new microkernel from scratch targeted at today's desktop market. What would be the components of your kernel and why? In your opinion, does kernel design play a significant role with respect to meeting perceived user demand?