7.7 Consider a possible optimization: instead of using *ready*, arrange for *semdelete* to examine the priority of each waiting process before the process is placed on the ready list. If none of the processes has higher priority than the current process, do not reschedule, but if any of them has a higher priority, call *resched*. What is the cost of the optimization and what is the potential savings?

7.8 The example code uses a FIFO policy for semaphores. That is, when a semaphore is signaled, the process that has been waiting the longest becomes ready. Imagine a modification in which the processes waiting for a semaphore are kept on a priority queue ordered by process priority (i.e., when a semaphore is signaled, the highest priority waiting process becomes ready). What is the chief disadvantage of a priority approach?

7.9 Languages meant specifically for writing concurrent programs often have coordination and synchronization embedded in the language constructs directly. For example, it might be possible to declare functions in groups such that the compiler automatically inserts code to prohibit more than one process from executing a given group at a given time. Find an example of a language designed for concurrent programming, and compare process coordination with the semaphores in the Xinu code.

7.10 When a programmer is required to manipulate semaphores explicitly, what types of mistakes can a programmer make?

7.11 When it moves the current process to the waiting state, *wait* sets field *prsem* in the process table entry to the ID of the semaphore on which the process is waiting. Will the value ever be used?

7.12 If a programmer makes a mistake, it is more likely that the error will produce 0 or 1 than an arbitrary integer. To help prevent errors, change *newsem* to begin allocating semaphores from the high end of the table, leaving slots 0 and 1 unused until all other entries have been exhausted. Suggest better ways of identifying semaphores that increase the ability to detect errors.

7.13 Function *semdelete* behaves in an unexpected way when deleting a semaphore with a non-negative count. Identify the behavior and rewrite the code to correct it.

7.14 Draw a call graph of all operating system functions from Chapters 4 through 7, showing which functions a given function invokes. Can a multi-level structure be deduced from the graph? Explain.