Consider the following questions – that is, consider the answer(s) to each of them. Although there is no guarantee that these questions will appear on the exam – or that these are the only questions that might appear on the exam – studying these questions will give you good preparation for the examination.

1. An operating system has two major purposes. What are they?
2. What is the difference (or differences) between a program and a process?
3. Can more than one process execute the same code?
4. Describe the typical memory layout for a process. Identify the various memory regions, how large they are, where they are relative to each other and the virtual (or physical) address space of the system, and the type of protection (e.g. read-only, read-write) that would be appropriate for the region, if the system architecture permits it.
5. Describe the content of a stack frame for any reasonable system – the x86 architecture with the gcc compiler and ld linker is a good choice, since that’s what we’re using in the class. Ideally you should be able to describe what’s located where in the stack frame.
6. With what is a stack frame for a function invocation associated: the process executing the function, or the code for the function? Explain.
7. When a process makes a system call – to Xinu or any other system – various things must be saved, and then later restored just before the system call returns to the process. Obviously all of the general purpose registers that are usable by the process must be saved. In the case of the x86 (e.g. the Galileo), that includes the program counter (eip), and many other registers (e.g. eax, ebx, ecx, esp, …) Where are these register’s contents saved? Are they all in the same place? If not, why not.
8. When a process P2 is waiting for process P1, what should P2 be doing? That is, should P2 be using the processor, or not?
9. Consider the terms exception, fault, interrupt, and other similar terms. Identify what these terms mean. Do they have the same meaning for all processors? If not, why?
10. Describe actions that will likely cause an exception to be generated? Include some that are not obvious, like a page fault, invalid instruction execution, and division by zero.
11. Describe the “interesting” features of the Xinu queue mechanism. In particular, discuss how it is used to construct the ready queue.
12. What does the resched function do in Xinu? Does this function ever return to its caller? If so, when does it return?
13. What is the minimum number of processes that must be ready to run at any time?
14. The API of an operating system – the set of system calls – provides the interface used by applications to access the extended services provided by
the operating system. What’s better: more system calls or fewer system calls? Why?

15. Assume a system call is being executed. In the context of which process is the system call executed? Is there any system call that is not “clearly” executed in the context of a single process?

16. Consider a system like Linux or Xinu. In both of these systems there are at least three techniques that could be used to terminate a process (normally or abnormally). Identify these.

17. What are the modes in which a typical processor executes? What is the difference between these modes?

18. Application processes almost always run with interrupts disabled. How, then, is it possible to generate an interrupt that causes entry into the kernel of the system?

19. In Xinu, suppose there are multiple processes waiting on a semaphore. What is the policy used to select the process that is awakened first?

20. In general, describe the difference between a policy and a mechanism?

21. The count associated with a semaphore is usually a non-negative integer. Xinu does something a little different, and allows the count for a semaphore to become negative. What does a negative semaphore count mean?

22. In many systems, like Linux and Windows, when a process terminates, all heap space associated with the process – that is, all storage allocated as a result of things like malloc or new (in appropriate languages/libraries) – is reclaimed by the operating system.
   a. Is the same property exhibited when a Xinu process terminates? Why, or why not?
   b. What makes reclamation of dynamically-allocated (heap) space easy in Windows and Linux?

23. Explain what Comer means by “first fit” and “last fit” when discussing memory allocation.

24. Xinu memory management is separated into “high level” and “low level” management. Why?

25. What happens when a semaphore is deleted? In particular, what happens to the processes that are waiting on a semaphore queue when the semaphore is deleted? What other approaches are possible?

26. Describe, in general, the organization of the page table for the Intel x86 architecture. How is the location of the page table found?

27. Comer indicates that some systems may include a page table length register. Why would such a register be of value? The x86 architecture does not have a page table length register. How is the effect of a page table length register reflect in the x86?

28. Comer suggests that paging the page table isn’t necessarily a good idea. Why does he say that?

29. What is meant by the term “interrupt mask?” How is the interrupt mask manipulated by kernel functions in Xinu?
30. How are interrupts and/or exceptions directed to the function in the kernel that deals with them?
31. Suppose an event could cause more than one process to move from a blocked state (in Xinu) to the ready state. How does Xinu avoid rescheduling before all of the processes have been moved to the ready state? Why should we care?
32. When should the resched function be called in Xinu?
33. What is meant by the terms “synchronous” and “asynchronous?”
34. The ctxsw function in Xinu is, as its name suggests, used to effect a context switch. When does that occur? Or perhaps more appropriately, why does a context switch occur?
35. How large are the low-level messages in Xinu? How long is the message queue associated with low-level messages? Is low-level message passing synchronous or asynchronous for the sender and/or receiver?
36. Describe the processor scheduling policy used in Xinu.
37. Describe the policy used to determine the order in which processes waiting on a semaphore are awakened.
38. We’ve seen a prototype for the code used for a Xinu system call. What is always the first thing that should be done by a system call? What’s the second thing?
39. We observed an obvious flaw in the code for the mkbufpool. What was that flaw? How you would rate the “seriousness” of the flaw on a scale of 1 to 10, where 1 is a minor problem and 10 is a very serious system-killing failure?
40. There are several single-bit status indicators normally associated with each page of memory for a process. Specifically we’re talking about the presence bit, the modified/dirty bit, and the referenced bit. (a) Describe each of these bits and how the hardware uses them, in general. (b) How are these bits used by various virtual memory systems in an OS? Is the content of any of these bits ever modified by the OS? Why, or why not?