1. A programmer is having problems synchronizing two POSIX threads in a program where both need to access a global integer variable named “count”. First, select all of the following which would allow him (assuming he makes no programming errors) to achieve synchronization. Second, prioritize these potential solutions from most desirable to least desirable and give your reasons.

   1a. Use Peterson’s solution with `sched_yield()`.
   1b. Declare the variable as “static int count;”.
   1c. Use a POSIX mutex to synchronize the threads.
   1d. Use a System V IPC semaphore to synchronize the threads.
   1e. Put the modification of `count` within a function call, e.g.:
       ```c
       void set_count(int i) { count=i; return; }
       ```

2. In the System V IPC suite we can also let processes communicate via shared memory. What are both the advantages and disadvantage of using shared memory rather than using message queues for interprocess communication?

3. In most operating systems hardware interrupts and exceptions are converted (by the interrupt handlers) into messages for one or more kernel message queues. Why is this done?

4. A hardware interrupt from a hardware device controller causes a level-1 handler to run. The memory accessed by this handler must be non-paged. That is, it must be memory which is always resident in core and never swapped. Why?

5. What is a Process Control Block (PCB)? What information in addition to the context of the process does it hold?

6. For each of the following options, what will be the effect (significant decrease, slight decrease, almost no change, slight increase, significant increase) on the run times for a system with only one disk array and for which almost all of the processes are I/O-bound. You must give your reasons:

   6a. Upgrade to a faster CPU.
   6b. Upgrade to a disk array which has a much larger cache.
   6c. Upgrade from a single-channel disk controller to a three-channel disk controller, and spread the filestore over three disk arrays.
   6d. Have root increase the priorities of all the I/O bound processes.

7. If a process is blocked in a slow system call, (for example `read()` from a network socket) and the process receives a signal, its state will be changed from blocked to ready.

   7a. What code will run first when the process is restarted?
   7b. What will the return on `read()` be and what will `errno` be?
   7c. In view of what signals are used for, why do you think the designers of Unix made the above decisions?

8. Suppose a systems programmer writes a kernel task which has a critical region starting with `CLI`. He makes a mistake and forgets the `STI` at the end of the critical region. What is the most likely consequence?

9. The system manager knows his virtual memory system has sufficient ram for all of its processes to run simultaneously, so he has turned off swapping (this can be done, but it’s dangerous if you miscalculate). The ram itself has an access time of 50 nSec. Can one therefore conclude that when a user program accesses a memory variable it will never take longer than 50 nSec to get the data into a register? Why or why not?

10. For a system of threads and resources we can create a directed graph called a resource allocation
graph which shows the current state of the system in terms of who is holding resources, who is (blocked and) requesting resources, and who produces resources (consumable resources only). Suppose we have threads 1, 2, and 3 and resources A and B. Suppose that resource B is consumable and produced only by thread 2. Currently, thread 1 is holding the only unit of resource A and is requesting resource B, thread 2 is requesting resource A, and thread 3 is requesting resource A. Is there a deadlock situation here? Why or why not?

11. Suppose that no resource is consumable and that all resources are numbered: R₁, R₂, . . . , Rₙ. Suppose that whenever a thread requires more than one resource to complete a subtask, it must request the resources in order. For example, if R₃, R₁₁, and R₆ are needed to accomplish a sub-task (after which they are released), they must be requested in the order:
   
   down(R₃)
   down(R₆)
   down(R₁₁)
   ... do subtask ...
   up(R₁₁)
   up(R₆)
   up(R₃)

   Is deadlock possible in this situation with these rules? Why or why not?

12. Kernel tasks/threads run between stopping points. If the resource which the thread needs at the stopping point is unavailable, the thread will usually call _task_yield() or the equivalent. What does this kernel function/macro do?