Question 1.  [7 marks]

Each of the following statements is false. In one sentence, explain why. (If you disagree that the statement is false, provide your reasoning.)

Part (a)  [1 mark]  A system call is invoked in the same manner as a user function.

A system call traps into the kernel so that privileged instructions can be run. A function call just jumps to another instruction in user-mode.
(Might say, switch to kernel-mode, software interrupt, syscall instruction.)

Part (b)  [1 mark]  Malloc and free must maintain data structures that track all free and allocated blocks of memory. Otherwise, memory could be leaked (lost).

It isn't necessary to keep a list of allocated blocks, since there is no reason to traverse them. It is up to the user-level program to keep track of allocated pointers so that memory isn't “lost”.
(I could imagine an argument that leads toward GC, but that isn't really the point here.)

Part (c)  [1 mark]  If two threads access a shared variable at the same time, there will be a concurrency error.

It is fine for threads to load a shared variable at the same time. They will both see the right value. It may also be fine if only one thread is guaranteed to be updating the shared variable, and the timing of when the second thread looks at the value doesn't matter.

Part (d)  [1 mark]  As seen in Exercise 6, hand-over-hand locking will always be slower than a single lock for the entire data structure.

We need a scenario where the overhead of acquiring and releasing locks is small relative to the amount of computation done holding the lock, and where multiple threads are operating on different parts of the list.

Part (e)  [1 mark]  Interactive systems should use non-preemptive scheduling algorithms.

A process with short cpu bursts (interactive process) might get stuck behind a process with a long cpu burst, leading to poorer response times relative to a preemptive scheduling algorithm.

Part (f)  [1 mark]  If we can reasonably predict how long a job will run, then Shortest Job First (or Shortest Time to Completion First) is a good policy for minimizing response time.

SJF is best for minimizing turnaround time, but jobs with short cpu bursts can still get stuck behind jobs with longer cpu bursts.

Part (g)  [1 mark]  A trap instruction is a privileged operation.

A user-level process needs to be able to call trap to switch into kernel-mode.
Question 2.  [3 marks]
Suppose you have implemented a doubly-linked list data structure. Threads begin traversing the list either from the head node or the tail node. Is it possible to enforce mutual exclusion by using one lock for the head and one lock for the tail? Explain.

No. Two processes might have access to the list at the same time, one holding the head lock and one holding the tail lock. They could potentially be modifying the same data.

Question 3.  [4 marks]
Consider an implementation of a doubly-linked list insert function where synchronization is implemented using a lock for each node. To change any pointer value, the lock must be held for the node containing the pointer, and the node the pointer points to. For example, to change A->next in the example below, we would need to hold A->lock and B->lock.

```
A      B      C
prev   prev   prev
next   next   next
lock   lock   lock
```

A remove function takes a valid pointer to a node in the doubly linked list, and removes that node from the list. To remove node B in the above example we would need to hold the locks for A, B, and C.

Part (a)  [2 marks]
Does this implementation of remove satisfy mutual exclusion? In other words, could another thread modify any next or prev pointers so that the list is not correctly linked together? Explain your thinking.

Yes, if all three locks are held, and the code is written to ensure that only the process holding the node lock can modify the next or prev pointers, then mutual exclusion is achieved. To delete B, A->next and C->prev must be modified.

A->next and C->prev must be modified as an atomic operation.

Part (b)  [2 marks]
Explain what can go wrong with this approach and why.

Deadlock. For example, if process X tries to remove B at the same time as process Y tries to remove C, then process X might acquire B->lock and A->lock, and process Y acquires C->lock. Now when process X tries to acquire C->lock it blocks, and process Y blocks waiting to acquire B->lock.
**Question 4.** [7 marks]

Consider the following multi-level queue algorithm:

- Processes in queue 0 are scheduled using round robin and a time quantum of 2 time units.
- Processes in queue 1 are scheduling using round robin and a time quantum of 4 time units.
- Processes that use their full time quantum move to (or stay in) queue 1.
- Processes that do not use their full time quantum move to (or stay in) queue 0.
- The scheduler only chooses processes from queue 1 when queue 0 is empty.

**Part (a) [2 marks]** When processes arrive, should they begin in queue 0 or queue 1? Explain your answer with an example.

*They should begin in queue 0. If a process with a short CPU burst begins in queue 1 it might have a long time to wait before being scheduled.*

**Part (b) [2 marks]**

Explain what benefit, if any, is gained by giving queue 1 a longer time quantum than queue 0.

*Lower overhead due to less frequent context switches. It shouldn’t impact responsiveness much because there aren’t any I/O bound processes executing if we are running a process in queue 1.*

**Part (c) [1 mark]** Explain how starvation could occur with an example.

*If there are enough long running but highly interactive processes or frequently arriving new processes so that queue 0 is never empty, any process that gets relegated to queue 1 will starve.*

**Part (d) [2 marks]** Propose a modification to this algorithm that would prevent starvation.

*Every n timesteps it moves all the processes from queue 1 to queue 0. Or, add a variable to each process to keep track of how long it has been in queue 1, and when it reaches some limit, move it to queue 0.

Students must describe a plausible algorithm rather than just say “aging” or something like that. It must be clear and complete for full marks.*