Do not turn this page until you have received the signal to start.

(Please fill out the identification section above, write your name on the back of the test, and read the instructions below.)

Good Luck!

This midterm consists of 3 questions on 6 pages (including this one). When you receive the signal to start, please make sure that your copy is complete. Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can’t figure out how to write the code.

If you use any space for rough work, indicate clearly what you want marked.

# 1: _____/ 7
# 2: _____/ 8
# 3: _____/ 6

TOTAL: _____/21
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] Malloc is a system call.

Part (b)  [1 mark] An advantage of threads is that all threads can access the entire address space.

Part (c)  [1 mark] Locks and semaphores are identical structures.

Part (d)  [1 mark] A system which uses a preemptive scheduling algorithm is just as efficient (ratio of useful work to operating system overhead) as a system which uses non-preemptive scheduling.

Part (e)  [1 mark] Round-robin scheduling is a good policy for minimizing turnaround time.

Part (f)  [1 mark] Limited direct execution means that once the OS has prepared a process for execution, the process has full control of the CPU.

Part (g)  [1 mark] The producer/consumer problem is a special case of the readers/writers problem with one reader and one writer.
Question 2. [8 marks]

Consider adding synchronization to the following working implementation of linked list `lookup()` and `insert()` functions using a single lock for the whole list.

```c
1 struct node *create_node(int value) {
2     struct node *n;
3     n = malloc(sizeof(struct node));
4     n->val = value;
5     n->next = NULL;
6     return n;
7 }
8
18 void insert(struct list *L, int value) {
19     struct node *newnode = create_node(value);
20     LOCK()
21     struct node *cur = L->head;
22     if(L->head == NULL || L->head->val > value) {
23         newnode->next = L->head;
24         L->head = newnode;
25         UNLOCK
26         return;
27     }
28     while(cur->next != NULL && cur->next->val <= value) {
29         cur = cur->next;
30     }
31     newnode->next = cur->next;
32     cur->next = newnode;
33     UNLOCK()
34     return;
35 }
```

Part (a) [2 marks] Is it necessary to lock `create_node`? Explain.

Part (b) [2 marks] Add lock and unlock calls to `insert` so that the lock is held for the minimum time.

Part (c) [2 marks]

If there are multiple threads calling `insert` and `lookup` concurrently, is locking `lookup` required? Explain carefully, referring to the code.

Part (d) [2 marks]

Suppose we now add a delete function. Is locking `lookup` required, when threads can be calling `insert`, `delete`, and `lookup` concurrently? Explain.
Question 3.  [6 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 scheduled 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.
- Every 20 time steps, the kernel swaps the two queues. Queue 1 becomes queue 0 and queue 0 becomes queue 1.

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

Part (b)  [2 marks] Does this solve the starvation problem? Explain your answer with an example.

Part (c)  [2 marks] Does this algorithm give priority to interactive processes? Explain.