This midterm consists of 5 questions on 8 pages (including this one and blank pages). When you receive the signal to start, please make sure that your copy is complete.

Pseudo-code is acceptable where code is required. Answer the questions concisely and legibly. Answers that include both correct and incorrect or irrelevant statements will not receive full marks.

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

Q1: _____/7
Q2: _____/6
Q3: _____/13
Q4: _____/6
Q5: _____/4
Total: _____/36
Q1. (1 mark each) True/False Indicate below, for each statement, whether it is (T)rue or (F)alse. Circle the correct answer.

T / F: Scheduling processes using FCFS minimizes average wait time.

T / F: Threads share the same address space, but have their own stack, SP and PC.

T / F: The properties of a monitor ensure that only one thread can be active inside a monitor at any given moment.

T / F: If no thread is blocked on a particular semaphore, then a signal on that semaphore will be recorded and let the first arriving thread go through a wait() operation on that semaphore.

T / F: Most modern operating systems are single-threaded.

T / F: User-level processes use the system calls interface to request services from the operating system.

T / F: A context-switch is handled entirely in hardware, while the OS just makes sure to change the protection mode.
Q2. (2 marks each) (Conceptual) Explain the following concepts or terms in the context of this course:

a) Race condition

b) Non-preemptive scheduling

c) Priority inversion

Q3. (13 marks) Short questions (Conceptual) Answer the following short questions.

a) (1 mark) Which monitor semantics requires using a while loop to test a condition.

b) (2 marks) Which critical section requirements does Peterson’s algorithm satisfy?
c) (2 marks) What sort of events drive the execution of an operating system? Give a couple of examples of such events.

d) (2 marks) Explain some of the limitations of semaphores in terms of usability. In what type of synchronization scenarios are condition variables preferred over semaphores?

e) (4 marks) Explain how the Bakery algorithm works. Considering the critical section requirements, then: Are we guaranteed mutual exclusion? Can starvation happen? Explain your rationale for both.

f) (2 marks) Which of the following process scheduling algorithms: FCFS, SJF, Round-Robin, can cause starvation? Provide supporting arguments for your answer.
Q4. (6 marks) Synchronization (Reasoning)

At Hogwarts school of magic, all wizards (students and professors) use a social network to establish groups of friends. The code behind this application uses some synchronization, as shown below.

You can assume that a wizard_list is a data type that represents a basic linked list, where each node contains a wizard structure and a next pointer. The list supports the following operations:

- void list_add (wizard_list *l, wizard *w);
- void list_remove (wizard_list *l, wizard *w);
- int wizard_is_in_list (wizard_list *l, wizard *w);

The first operation adds a wizard to a given list. The second removes a wizard from a given list (if the wizard is already in the list). The last operation checks if a wizard is in a given list (returns an integer: 1=yes, 0=no).

The function friend_wizard() is called whenever a wizard wants to befriend another wizard (you can assume the request is always granted). The function defriend_wizard() is called when a wizard wishes to break a friendship with another wizard.

Your task is to decide whether the two functions are correct, or whether one or several problems may arise. Indicate below what your conclusions are, explaining in detail your reasoning. Give examples if necessary.

typedef struct wiz {
    char *name;
    wizard_list *friends;
    pthread_mutex_t *lock;
} wizard;

void friend_wizard(wizard *me, wizard *newfriend) {
    pthread_mutex_lock(me->lock);
    if ( ! wizard_is_in_list(me->friends, newfriend)) {
        list_add(me->friends, newfriend);
        pthread_mutex_lock(newfriend->lock);
        list_add(newfriend->friends, me);
        pthread_mutex_unlock(newfriend->lock);
        printf("%s is now connected to %s\n", me->name, newfriend->name);
        return;
    }
    pthread_mutex_unlock(me->lock);
}

void defriend_wizard(wizard *me, wizard *oldfriend) {
    pthread_mutex_lock(me->lock);
    if ( wizard_is_in_list(me->friends, oldfriend) ) {
        list_remove(me->friends, oldfriend);
        pthread_mutex_lock(oldfriend->lock);
        list_remove(oldfriend->friends, me);
        pthread_mutex_unlock(oldfriend->lock);
        printf("%s is no longer connected to %s\n", me->name, oldfriend->name);
        return;
    }
    pthread_mutex_unlock(me->lock);
}
Q5. (4 marks) Scheduling (Reasoning)
Assume that we have a multi-level queue scheduler, with 2 queues Q0 and Q1, where Q0 has the highest priority. Both queues use a round-robin scheduling algorithm, with a time quantum of 100 nanoseconds. New processes and processes returning from I/O start at the back of Q0 (among these two categories, new processes go first). When a process finishes its time quantum, it gets preempted and placed at the back of the lower queue (unless it is already in Q1, in which case it stays in Q1).

This scheduler is used on a typical Unix machine. A context switch on this machine is typically 2-3 microseconds. Most processes running on this system typically finish after 5 seconds of runtime.

After having a look over this scheduler, Linus Torvalds states that this scheduler will be awful.
a) Why would Linus state this? Do you agree? Explain your answer.
b) If you agree with Linus, how would you improve your scheduler, such that the system remains interactive (all processes make some progress)?
Print your name and student number in this box.