Duration: **30 minutes**
Aids Allowed: None

---

Do **not** turn this page until you have received the signal to start.
(In the meantime, please fill out the identification section above, and read the instructions below carefully.)

---

This midterm test consists of ?? question on ?? page (including this one). When you receive the signal to start, please make sure that your copy of the test is complete. Extra space was left for each of the programming questions. Please indicate clearly the part of your work that should be marked.

IMPORTANT: You may use C-like pseudo code in answering the coding question. We will be grading the correctness and clarity of the algorithm, not syntax.

Marks will be awarded for precise answers. Vague answers will not receive full marks.

**Marking Guide**

??   # 0: _____/??

TOTAL: _____/??

---

**Good Luck!**

Total Pages = ??
Page 1
Question 1.  

NOTE: These are short answer questions. We are expecting a sentence or two. Please do not fill up all the available space!

Part (a)  
Consider the following code.

```c
int turn = 0; // a global variable

my_work(id_t id) { // id_t can be 0 or 1
    while (turn != id)
    
    execute_critical_region();
    turn = 1 - id;
}
```

If `my_work` is executed by 2 threads, one that passes in the value 0, and the other that passes in the value 1, is mutual exclusion on `execute_critical_region` satisfied? If yes, explain why. If no, give a sequence of operations that would lead to both threads calling `execute_critical_region` at the same time.

Yes, mutual exclusion is satisfied. The variable `turn` can only hold one value at a time, so only one of the threads will see the condition in the while loop become false and enter the critical region.

Part (b)  

Explain the difference between a semaphore wait and a condition variable wait.

A semaphore wait will return when the value of the semaphore variable becomes greater than 0 (or non-zero in Posix). The condition variable wait will return when it receives a signal message. They are similar in that more than one thread may be waiting, but a semaphore wait may return immediately if signal has been called before the wait call, whereas the condition variable wait will alway block until a later signal is received.

Any answer that shows a clear understanding of semaphores and condition variables is sufficient.
Part (c) [4 MARKS]
One possible implementation of a mutex is that all threads waiting on the mutex will be awakened, and only one thread will be able to acquire the mutex. The remaining threads will go back to waiting. Explain two major drawbacks of this implementation.

- system overhead in restarting and executing all the threads just to have them go back to sleep
- starvation - If the thread that “wins” is chosen randomly, a thread might never succeed at acquiring the mutex.
Question 2. [10 marks]
The Hungry Birds. There are several baby birds and one parent bird. The baby birds eat out of a dish that initially contains $F$ worms. Each baby repeatedly eats one worm, sleeps for a while, and then comes back to eat.

When the dish becomes empty, the baby bird who empties the dish notifies the parent bird. The parent refills the dish with $F$ worms, and then waits for the dish to become empty again. This pattern repeats forever.

The basic algorithm for the baby bird and parent are given below. Complete the `get_food` and `fill_dish` functions with appropriate synchronization using a monitor implemented with condition variables ($cv\_wait$, $cv\_signal$) and their associated mutexes. The functions `get_food` and `fill_dish` will be inside the monitor.

Remember to handle the case where baby birds want to get food but find the dish empty.

Declare all shared variables including the synchronization variables in the GLOBALS section, but assume all condition variables and mutexes are correctly initialized in some other part of the code.

```c
baby_bird() {
    while(1) {
        get_food();
        sleep(sometime);
    }
}
parent() {
    while(1) {
        fill_dish();
    }
}
// GLOBALS
int dish = F;
// More globals for solution
pthread_mutex_t mutex;
pthread_cond_t empty;
pthread_cond_t mama_cv;
```
// Consume one worm from the dish (dish--;
// If this causes the dish to go to 0, then notify the parent
void get_food() {
    mutex_lock(mutex)
    while(dish == 0) {
        cv_wait(empty, mutex)
    }
    dish--
    if(dish == 0) {
        cv_signal(mama_cv)
    }
    mutex_unlock(mutex)
}

// Wait for the dish to be empty, then fill it by setting dish = F
void fill_dish() {
    mutex_lock(mutex);
    while(dish != 0) {
        cv_wait(mama_cv, mutex);
    }
    dish = FULL;
    foreach baby bird {
        cv_signal(empty);
    }
    pthread_mutex_unlock(&mutex);
}

Marking:
• 2 using mutex correctly (lock and unlock)
• 2 using cv correctly
• 2 when dish goes to 0 signal
• 2 when bird finds dish empty wait
• 2 signal all birds when dish is full