Quiz 4, CS 202(-001), Spring 2026

Please write your name and netid below, and answer all of the questions. The quiz has two sides.

Name: Net ID (@nyu.edu):

1. Recall this code from the last class, and note that `p1()` and `p2()` will execute concurrently, in separate threads (one executes in the main thread, the other on a newly created thread).

```
int data = 0, ready = 0;

int main () {
    tid id = thread_create (p1, NULL);
    p2 ();
    thread_join (id);
}

void p1 () {
    data = 2000;
    ready = 1;
}

int p2 () {
    while (!ready) {} // loop until we read ready != 0
    use(data);
}
```

   If we assume a single CPU (and the compiler doesn't reorder instructions), can the function `use()` be called with 0? Justify your answer in two sentences or less.

2. Consider a linked list defined as in lab 1:

```
struct list_node {    // from lab1
        int  value;
        struct list_node *next;
};
```

   On the next page, write a function, `count_nodes`, that given a pointer to the head of the list, returns the number of nodes in the list. Your code must handle the case that the list is empty, which is indicated by a NULL (or 0) value of the parameter `head`.

```
int count_nodes(struct list_node* head)
{















}
```

3. Assume a setup where multiple threads are operating on shared state, and there is a flag called `ready`; the requirement is that shared state is accessed only when `ready` is non-zero. There is also a declaration of a mutex and condition variable:

   ```
   mutex m;
   Cond cv;
   int ready = 0;
   // other shared state
   ```

   Now consider a function executed by a thread:

   ```
   acquire(&m);

   if (!ready)
       cond_wait(&cv, &m);

   /* use shared state */

   release(&m);
   ```

   Which of the following are reasons that the above function is incorrect?

   Circle ALL that apply:

   a The mutex is released during `cond_wait`, leading to a race condition (accessing shared state without a mutex).

   b `cond_wait` may return even if no thread signaled `cv` and `ready` is still 0.

   c Another thread may set `ready` back to 0 after a signal but before this thread resumes.

   d None; this code is correct.