On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work.

Print name and sign: _____

Question:	1	2	3	4	Total
Points:	5	5	15	5	30
Score:					

1. (5 points) What is a spinlock? Give one advantage to using a spinlock implementation of a lock primitive and one disadvantage.

2. (5 points) Compare and contrast **threads** and **process**.

3. Dr. Summet is working on a library of data structures which will be **thread-safe**. That is, her data structures will be safe to be used in concurrent, multi-threaded programs. She is working on a `vector` class which is really an array and a lock to protect the array, like this:

```
typedef struct vector {
    int size;               /* length of vector */
    int* v;                 /* pointer to array of ints */
    pthread_mutex_t lock;   /* lock to protect array */
} vector_t;
```

Then she wrote a function to add two vectors together in (she thinks...) a thread-safe way:

```
1  void add_vector(vector *v1, vector *v2) {
2      mutex_lock(v1->lock);
3      mutex_lock(v2->lock);
4      for(int i = 0; i < v1->size; i++) {
5          v1->v[i] = v1->v[i] + v2->v[i];
6      }
7      mutex_unlock(v1->lock);
9      mutex_unlock(v2->lock);
10 }
```

You are using Dr. Summet's library and have written code which does the following:

Thread 1 calls `add_vector(&vectorA, &vectorB)` Thread 2 calls `add_vector(&vectorB, &vectorA)`

However, the code hangs, and you think you might be observing deadlock. Dr. Summet may have made some mistakes!

(a) (5 points) Why did the deadlock happen? Describe or draw a picture which clearly shows how deadlock could happen in the given scenario.

(b) (5 points) How could you write `add_vector` so that this deadlock never happens?

(c) (5 points) We know that for deadlock to occur we must have the following: hold-and-wait, mutual exclusion, no preemption, and circular dependencies.
In your proposed solution in part b, which of these 4 conditions did you remove? Describe how your solution removes that condition and thus makes deadlock impossible.

4. Consider the following code which implements a counter.

```
int counter = 0;

mythread(void *arg) {
    int i;
    for (i = 0; i < 10000; i++) {
        counter = counter + 1;
    }
    return NULL;
}
```

Assume two threads, A and B, are running the above `mythread` function at the same time.

(a) (3 points) Circle **all** of the following which is/are possible values of `counter` after both threads have completed.

A. 5000
B. 10000
C. 15000
D. 20000
E. 25000

(b) (2 points) Do we need a lock to protect the variable `i`? Why or why not?