CSSE 332 -- Operating Systems

Introduction to Mutual Exclusion

Name: Solution Key

Question 1. (5 points) In your own words, describe the main problem leading the sample code in `simple_example.c` to produce incorrect results.

> **Solution:** The main issue is that we have a race condition that gets triggered on the `counter` variable. Multiple threads are reading **and writing** to that variable at the same time, which introduces a concurrency bug.

Question 2. Consider the following code snippet:

```
1  struct metadata {
2    unsigned int id, length;
3    int *array;
4  };
5
6  void *thread_function(void *arg) {
7    struct metadata *ptr = (struct metadata*)arg;
8    printf("Thread %d started, processing array of length %d...", ptr->id,
9           ptr->length);
10
11   // swap_max_with_last is a function defined elsewhere that finds the
12   // maximum element in an array and swaps it with the last element in that
13   // array (the last element is specified by the argument end below).
14   // The signature for this function is the following:
15   //   void swap_max_with_last(int *array, int start, int end);
16   // where:
17   //   array: An array of integer.
18   //   start: The starting index to access the array from.
19   //   end:   The ending index to access the array.
20   swap_max_with_last(ptr->array, 0, ptr->length - ptr->id + 1)
21
22   printf("Thread %d done.\n", ptr->id);
23   return 0;
24 }
25
26 int main(int argc, char **argv) {
```

```
27    int *array = malloc((2<<20) * sizeof(int));
28    struct metadata all_meta[TOTAL_THREADS];
29    pthread_t threads[TOTAL_THREADS];
30    int i;
31
32    // defined elsewhere
33    initialize_array(array);
34    for(i = 0; i < TOTAL_THREADS; i++) {
35      all_meta[i].id = i + 1;
36      all_meta[i].length = 2<<20;
37      all_meta[i].array = array;
38      pthread_create(&threads[i], NULL, thread_function, &all_meta[i]);
39    }
40
41    for(i = 0; i < TOTAL_THREADS; i++) {
42      pthread_join(threads[i], NULL);
43    }
44    exit(0);
45  }
```

(a) (5 points) What do you think this piece of code is attempting to do?

Solution: It is attempting to sort the array by successively trying to swap the maximum element with the last element.

(b) (10 points) In the code listing above, identify any critical sections and possible race conditions. Feel free to add your notes to the code listing itself.

Solution: The entirety of `swap_max_with_last` is a critical section since it modifies elements of the array. We cannot know beforehand which elements will be modified, so it is likely that threads will overwrite each other if we are careful.

(c) (5 points) At the end of the `main`, what do you expect the contents of `array` to be?

Solution: We cannot tell because of the damned scheduler.

Question 3. (5 points) In the space below, write down the main API functions used to create and use mutex lock in the `pthreads` library.

Solution:

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER; // init a mutex lock
pthread_mutex_lock(&m); // lock the mutex, block if already locked (by me
                        //  or someone else).
pthread_mutex_unlock(&m); // unlock the mutex, make sure you have it!
```

Question 4. (10 points) The code listing below contains a major bug, identify it and suggest a way to fix it.

```
void *thread1(void *ignored) {
  // some code
  pthread_mutex_lock(&lock);

  // do some stuff

  pthread_mutex_unlock(&lock);
  return 0;
}

void *thread2(void *ignored) {
  // some initialization code

  pthread_mutex_unlock(&lock);
  pthread_mutex_lock(&lock);

  // do some stuff

  pthread_mutex_unlock(&lock);
  return 0;
}
```

Solution: We unlock the mutex from `thread2` even though we do not have it, so we might end up releasing `thread1`'s lock. The behavior is undefined if that happens.