Homework 5

Ally Smith (Section B)

March 28, 2022

Question 1.

Consider the following code:

```
P1: {                                 P2: {
    shared int x;                         shared int x;
    x = 10;                               x = 10;
    while (1) {                           while (1) {
        x = x - 1;                            x = x - 1;
        x = x + 1;                            x = x + 1;
        if (x != 10) {                        if (x != 10) {
            printf("x is %d", x)                  printf("x is %d", x)
        }                                     }
    }                                     }
}                                     }
```

Note that the scheduler in a uni-processor system would implement pseudo-parallel execution of these two concurrent processes by interleaving their instructions, without restriction on the order of the interleaving.

Show a sequence (i.e., trace the sequence of interleavings of statements) such that the statement “x is 10” is printed.

```
x = 10;
                      x = 10;
                      x = x - 1;

x = x - 1;
x = x + 1;
if (x != 10)
printf("x is %d", x)
```

Question 2.

Answer the following questions on deadlock:

(a) What are the necessary conditions for a deadlock?

Circular wait, no preemption, hold & wait, and mutual exclusion.

(b) Describe the method to address the condition of hold and wait, and explain why it works.

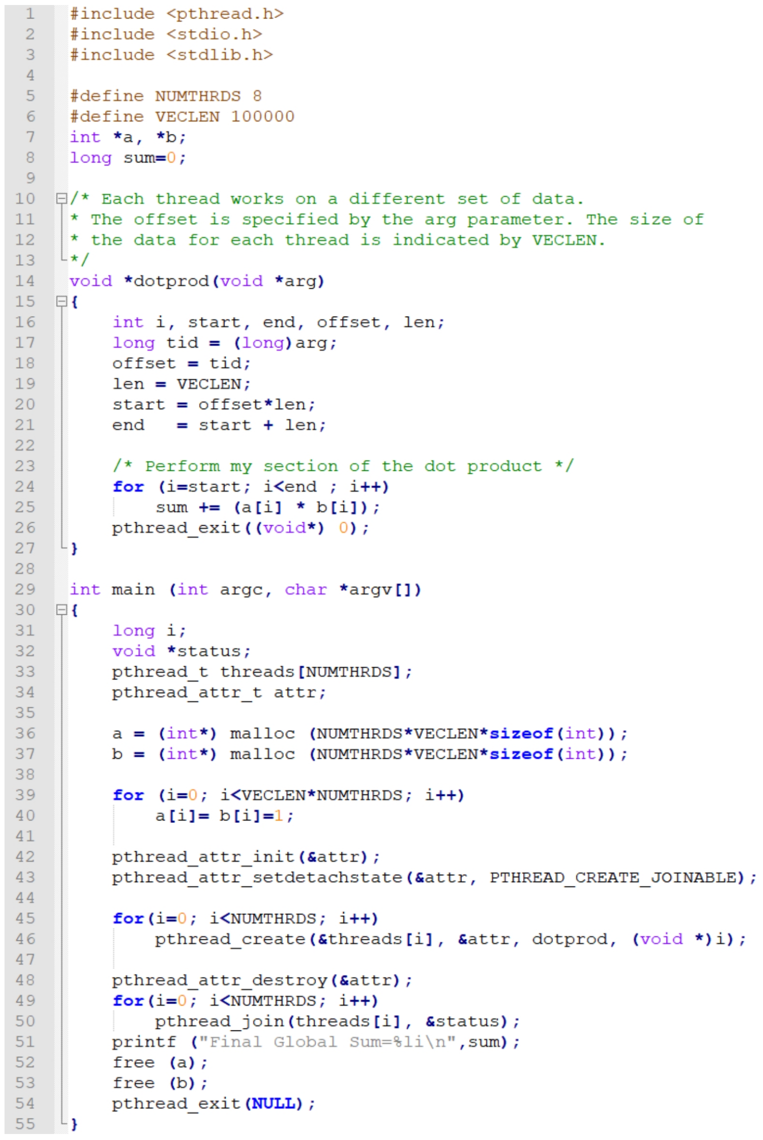
To address this, you can require that a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously. This ensures that the program will not be waiting on anything before it actually starts running.

(c) Describe the approach to addressing circular wait, and explain why it works.

If you were to define a linear set order in which resources can be used, ensuring that no cycles can form, you can avoid this problem entirely.

Question 3.

There is a bug in the code below. Explain what it is clearly and re-write the portion of the program that will fix the bug. You don't need to re-write the entire code. (i.e. Your solution should state: "replace lines X-Y of the code with this my piece of code")



```c
1   #include <pthread.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   #define NUMTHRDS 8
6   #define VECLEN 100000
7   int *a, *b;
8   long sum=0;
9
10  /* Each thread works on a different set of data.
11  * The offset is specified by the arg parameter. The size of
12  * the data for each thread is indicated by VECLEN.
13  */
14  void *dotprod(void *arg)
15  {
16      int i, start, end, offset, len;
17      long tid = (long)arg;
18      offset = tid;
19      len = VECLEN;
20      start = offset*len;
21      end   = start + len;
22
23      /* Perform my section of the dot product */
24      for (i=start; i<end ; i++)
25          sum += (a[i] * b[i]);
26      pthread_exit((void*) 0);
27  }
28
29  int main (int argc, char *argv[])
30  {
31      long i;
32      void *status;
33      pthread_t threads[NUMTHRDS];
34      pthread_attr_t attr;
35
36      a = (int*) malloc (NUMTHRDS*VECLEN*sizeof(int));
37      b = (int*) malloc (NUMTHRDS*VECLEN*sizeof(int));
38
39      for (i=0; i<VECLEN*NUMTHRDS; i++)
40          a[i]= b[i]=1;
41
42      pthread_attr_init(&attr);
43      pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
44
45      for(i=0; i<NUMTHRDS; i++)
46          pthread_create(&threads[i], &attr, dotprod, (void *)i);
47
48      pthread_attr_destroy(&attr);
49      for(i=0; i<NUMTHRDS; i++)
50          pthread_join(threads[i], &status);
51      printf ("Final Global Sum=%li\n",sum);
52      free (a);
53      free (b);
54      pthread_exit(NULL);
55  }
```

The code can be fixed by making the following changes. Assume all system calls succeed.

```
// Insert this code after line 8
pthread_mutex_t mutex;

// Replaced lines 24 with the following code:
pthread_mutex_lock(&mutex);
sum += (a[i] * b[i]);
pthread_mutex_unlock(&mutex);

// Insert this code after like 34
pthread_mutex_init(&mutex, NULL);
```