Due: Midnight, March 26, 2020

For each of the problems below, the solution will be a table walking through the code statement by statement showing:

1. The line number in the program
2. A short description of what the line is doing
3. The values of each of the variables in the program
4. The state (or value) of each Lock, Control Variable or Semaphore in use.

As an example, the excel spreadsheet we created in class on Thursday, March 12 is available in Oaks. You don't have to use a spreadsheet; you can do it on paper and scan it or whatever other approach you would like.

1. Control Variables

In the program below, there is a simple use of a control variable to signal the main process. There are two possible paths through the code:

a) The thread runs first after it is created, before the main process runs.
b) The main process runs first after the thread is created, before the thread runs.

For **each** path through the code, trace the state of the program at least far enough to show that the control variable is working correctly.

```
1  #include <assert.h>
2  #include <pthread.h>
3  #include <stdio.h>
4
5  int state_var = 0;
6  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
7  pthread_cond_t c = PTHREAD_COND_INITIALIZER;
8
9  void thr_signal() {
10     assert(pthread_mutex_lock(&m) == 0);
11     state_var = 1;
12     assert(pthread_cond_signal(&c) == 0);
13     assert(pthread_mutex_unlock(&m) == 0);
14 }
15
16 void thr_wait() {
17     assert(pthread_mutex_lock(&m) == 0);
18     while (state_var == 0)
19         assert(pthread_cond_wait(&c, &m) == 0);
20     assert(pthread_mutex_unlock(&m) == 0);
21 }
22
23 void *child(void *arg) {
24     printf("Child: Start\n");
25     thr_signal();
26     printf("Child: After Signal\n");
27     return NULL;
28 }
29
30 int main(int argc, char *argv[]) {
31     printf("Parent: Start\n");
32
33     pthread_t p;
34     assert(pthread_create(&p, NULL, child, NULL) == 0);
35
36     thr_wait();
37     printf("Parent: After waiting for child\n");
38
39     void * rc;
40     assert(pthread_join(p, &rc) == 0);
41
42     printf("Parent: End\n");
43     return 0;
44 }
end
```

2. Semaphores - Locking

In the program below, there is a simple use of a semaphore as a lock. Trace through the code at least far enough to provide the lock is working correctly. Assume both threads will start immediately after they are created.

```
1  #include <assert.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6
7  int max = 5;
8  int counter = 0;
9
10 sem_t lock;
11
12 void *mythread(void *arg) {
13
14     char *letter = arg;
15
16     for (int i = 0; i < max; i++) {
17
18         assert(sem_wait(&lock) == 0);
19         counter = counter + 1;
20         assert(sem_post(&lock) == 0);
21
22     }
23
24     printf("%s: done\n", letter);
25     return NULL;
26 }
27
28 int main(int argc, char *argv[]) {
29
30     if (argc != 2) {
31         fprintf(stderr, "usage: SemaphoreLocking <loopcount>\n");
32         exit(1);
33       }
34
35     max = atoi(argv[1]);
36
37     printf("main: begin [counter = %d] [%p]\n", counter, &counter);
38
39     // Initilize the lock
40     assert(sem_init(&lock, 0, 1) == 0);
41
42     // Create the threads
43     pthread_t p1;
44     assert(pthread_create(&p1, NULL, mythread, "A") == 0);
45
46     pthread_t p2;
47     assert(pthread_create(&p2, NULL, mythread, "B") == 0);
48
49     // join waits for the threads to finish
50
51     assert(pthread_join(p1, NULL) == 0);
52     assert(pthread_join(p2, NULL) == 0);
53
54     printf("main: done\n [counter: %d]\n [should: %d]\n",
55            counter, max*2);
56
57     return 0;
58 }
end
```

3. Semaphores - Signaling

In the program below, there is a simple use of a semaphore to signal the main process. As with the control variable example, there are two possible paths though the code:

a) The thread runs first after it is created, before the main process runs.
b) The main process runs first after the thread is created, before the thread runs.

For **each** path through the code, trace the state of the program at least far enough to show that the control variable is working correctly.

```
1  #include <assert.h>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <semaphore.h>
5
6  sem_t signal;
7
8  void thr_signal() {
9      assert(sem_post(&signal) == 0);
10 }
11
12 void thr_wait() {
13     assert(sem_wait(&signal) == 0);
14 }
15
16 void *child(void *arg) {
17     printf("Child: Start\n");
18     thr_signal();
19     printf("Child: After Signal\n");
20     return NULL;
21 }
22
23 int main(int argc, char *argv[]) {
24     printf("Parent: Start\n");
25
26     assert(sem_init(&signal, 0, 1) == 0);
27
28     pthread_t p;
29     assert(pthread_create(&p, NULL, child, NULL) == 0);
30
31     thr_wait();
32     printf("Parent: After waiting for child\n");
33
34     void * rc;
35     assert(pthread_join(p, &rc) == 0);
36
37     printf("Parent: End\n");
38     return 0;
39 }
end
```