Isabel Lally

Test 2: Concurrency

1. True/False
    a. True
    b. True
    c. True
    d. False
    e. True
    f. False
    g. False
    h. True
    i. False
    j. True

2. Multiple Choice
    a. C
    b. A
    c. D
    d. C
    e. D
    f. D
    g. A
    h. B
    i. D
    j. A

3. Briefly Describe
    a. Sem_init initializes a semaphore with the parameters being (pointer to semaphore, 0, starting value).
    b. Pthread_create creates a thread with the parameters being (pointer to thread, 0, the function to run, and any arguments for function).
    c. Pthread_unlock allows another thread to access a critical section of code with the parameters being (a pointer to the mutex that is protecting that section).
    d. Int pthread_cond_signal tells the mutex that it is safe to unlock with the parameters being (a pointer to the conditional variable that is checking that it is safe to unlock that mutex).
    e. Sem_wait tells the semaphore to lock that area of code until it has completed with the parameters being (a pointer to that semaphore that is protecting that code).
4. Short Answer
    a. We did not need to lock the hash table because each entry was protected within the linked list associated. This allowed for multiple entries to be worked with, while still protecting the ones that were already being accessed.
    b. We use wait any if there isn't special data to signal the ending of a pipeline. Since we don't have that end, if the which vector is not locked around any changes to the which vector, it will not signal correctly when our pipeline is to end.
    c. Both the spin lock without the yield and spin lock with the yield will have the same amount of correctness. The yield lock with the yield will be fairer than the spin lock

without the yield. The spin lock without the yield will be more efficient than the spin lock with the yield.

d. You need to lock around the update of the counter because if multiple threads access the counter value at the same time, it will not increment correctly. For example, one thread may read the counter at 1 and before it has updated another thread may also read the counter at one. Both threads will then update the counter to 2 and will obviously not be counting correctly.

e. It is possible to build a queue that allows one thread to push items at the same time as another thread is popping items off the other end because as long as there is something in the queue to pop, it is able to pop and it isn't going to pop until something has been added completely. The push and pop functions are never going to be accessing the same item at one time.

5. Trace the Code

a. Thread runs first after it is created.

Line	Description	signal
26	Initialize Semaphore	0
29	Thread Create	
17	"Child: Start"	
18	Thr_signal	
9	Post Semaphore	1
19	"Child: After Signal"	
31	Thr_wait	
13	Semaphore Wait (doesn't wait)	0
31	"Parent: After waiting for child"	
35	Thread Join	
37	"Parent: End"	

b. Main runs first after thread is created.

Line	Description	signal
26	Initialize Semaphore	0
29	Thread Create	
31	Thr_wait	
13	Semaphore Wait (does wait)	-1
17	"Child: Start"	
18	Thr_signal	
9	Post Semaphore	0
19	"Child: After Signal"	
32	"Parent: After waiting for child" (stops waiting)	
35	Thread Join	
37	"Parent: End"	

6. Two Errors and How to Correct

a. There's no conditional variable to tell lock that it is okay for it to unlock. Just adding a pthread_cond_t to the list_t struct would fix that and putting it between line 33 and 34 and at line 54.

   b. If any of the threads stop working, we can end up in a deadlock. There needs to be something that will deal with releasing that lock for List_Insert.

7. Fill in Missing Code
   a. pthread_mutex_lock(&mutex);
   b. sem_wait(&empty);
   c. sem_post(&empty);
   d. pthread_mutex_unlock(&mutex);
   e. pthread_mutex_lock(&mutex);
   f. sem_wait(&full);
   g. sem_post(&full);
   h. pthread_mutex_unlock(&mutex);