Quiz 2

Your Name: ________________________

Note: For all the questions asking you to show the problem, we only consider the following situations as problems: deadlock, data loss (writing to a buffer when it's already full or data overwritten), consuming data from a buffer when there is no data, all threads go to sleep forever.

1. Term explanation: binary semaphore. (5 points)

2. Term explanation: race condition. (5 points)

3. Given the code snippet below what is the maximum number of threads that can simultaneously enter the code protected by the semaphore below. (10 points)

```
sem_t s;
void *threadStuff(void *d)
{
    sem_wait(&s);
    /* code */
    sem_post(&s);
}

void main()
{
    thread_t thread1, thread2, thread 3;
    sem_init(&s, 0, 2);
    pthread_create(&thread1, NULL, threadStuff, NULL);
    pthread_create(&thread2, NULL, threadStuff, NULL);
    pthread_create(&thread3, NULL, threadStuff, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    sem_destroy(&s);
}

Answer: 2.
```

4. Given the following trace, in which each register %dx is initialized to 3, what values will %dx see after the run? Is there a race in this code? (10 points)

```
   dx          Thread 0                Thread 1
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1000 sub  $1,%dx
    ?                            1001 test $0,%dx
    ?                            1002 jgte .top
    ?                            1003 halt
```

```
Answer: No race condition - threads do not share registers.
dx will be -1, for both thread 0 and thread 1.
```

5. Given the following trace, in which each register %dx is initialized to 3, what value would %dx have in the end? Is there a race in this code? (10 points)

```
  dx          Thread 0                Thread 1
   ?
   ?   1000 sub  $1,%dx
   ?   1001 test $0,%dx
   ?   1002 jgte .top
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1000 sub  $1,%dx
   ?                            1001 test $0,%dx
   ?                            1002 jgte .top
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?   1000 sub  $1,%dx
   ?   1001 test $0,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1000 sub  $1,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?   1002 jgte .top
   ?   1000 sub  $1,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1001 test $0,%dx
   ?                            1002 jgte .top
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?   1001 test $0,%dx
   ?   1002 jgte .top
   ?   1000 sub  $1,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1000 sub  $1,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?   1001 test $0,%dx
   ?   1002 jgte .top
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1001 test $0,%dx
   ?                            1002 jgte .top
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?   1003 halt
   ?   ----- Halt;Switch -----  ----- Halt;Switch -----
   ?                            1000 sub  $1,%dx
   ?                            1001 test $0,%dx
   ?   ------ Interrupt ------  ------ Interrupt ------
   ?                            1002 jgte .top
   ?                            1003 halt
```

```
Answer: No race condition - threads do not share registers.
dx will be -1, for both thread 0 and thread 1.
```

6. **The Producer/Consumer Problem - Solution 1**. - Will not be in the final exam.

Below is a possible solution for the producer and consumer problem; we assume the buffer size is 1, meaning only 1 item can be stored in the buffer.

```
int buffer;
int count = 0; // initially, empty

void put(int value) {
assert(count == 0); // throw an error and terminate the program if count isn't 0.
count = 1;
buffer = value;
}

int get() {
assert(count == 1); // throw an error and terminate the program if count isn't 1.
count = 0;
return buffer;
}
```

The put and get routines.

```
int loops; // must initialize somewhere...
cond_t cond;
mutex_t mutex;

void *producer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        pthread_mutex_lock(&mutex);              // p1
        if (count == 1)                          // p2
            pthread_cond_wait(&cond, &mutex);    // p3
        put(i);                                  // p4
        pthread_cond_signal(&cond);              // p5
        pthread_mutex_unlock(&mutex);            // p6
    }
}

void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        pthread_mutex_lock(&mutex);              // c1
        if (count == 0)                          // c2
            pthread_cond_wait(&cond, &mutex);    // c3
        int tmp = get();                         // c4
        pthread_cond_signal(&cond);              // c5
        pthread_mutex_unlock(&mutex);            // c6
        printf("%d\n", tmp);
    }
}
```

The producer and consumer threads.

Let's say we have one producer thread (Tp), two consumer threads (Tc1 and Tc2). Above is a broken solution, please draw a trace below to show the problem. Assume there is only one CPU (one core), and each thread can have 3 states: running, ready, sleep. (15 points)

Tp	State	Tc1	State	Tc2	State	Count	Comment
	Ready		Ready		Ready	0	This is the initial state for the entire program.

7. **The Producer/Consumer Problem - Solution 2**. - Will not be in the final exam.

   Below is another possible solution for the producer and consumer problem; we assume the buffer size is 1, meaning only 1 item can be stored in the buffer.

```
int buffer;
int count = 0; // initially, empty

void put(int value) {
assert(count == 0); // throw an error and terminate the program if count isn't 0.
count = 1;
buffer = value;
}

int get() {
assert(count == 1); // throw an error and terminate the program if count isn't 1.
count = 0;
return buffer;
}
```

The put and get routines.

```
int loops;
cond_t cond;
mutex_t mutex;

void *producer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread_mutex_lock(&mutex);           // p1
        while (count == 1)                    // p2
            Pthread_cond_wait(&cond, &mutex); // p3
        put(i);                               // p4
        Pthread_cond_signal(&cond);           // p5
        Pthread_mutex_unlock(&mutex);         // p6
}
}

void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread_mutex_lock(&mutex);           // c1
        while (count == 0)                    // c2
            Pthread_cond_wait(&cond, &mutex); // c3
        int tmp = get();                      // c4
        Pthread_cond_signal(&cond);           // c5
        Pthread_mutex_unlock(&mutex);         // c6
        printf("%d\n", tmp);
}
}
```

The producer and consumer threads.

Let's say we have two producer threads (Tp1 and Tp2), one consumer thread (Tc). Once again this is a broken solution, please draw a trace below to show what problem could happen. Assume there is only one CPU (one core), and each thread can have 3 states: running, ready, sleep. (15 points)

Tp1	State	Tp2	State	Tc	State	Count	Comment
	Ready		Ready		Ready	0	This is the initial state for the entire program.

8. **The Producer/Consumer Problem - Semaphore Solution**. - Will not be in the final exam.

```
int buffer[MAX];
int fill = 0;
int use = 0;

void put(int value) {
buffer[fill] = value;     // Line F1
fill = (fill + 1) % MAX; // Line F2
}

int get() {
int tmp = buffer[use];   // Line G1
use = (use + 1) % MAX;   // Line G2
return tmp;
}
```

The put and get routines

```
sem_t empty;
sem_t full;

void *producer(void *arg) {
int i;
for (i = 0; i < loops; i++) {
sem_wait(&empty);         // Line P1
put(i);                   // Line P2
sem_post(&full);          // Line P3
}
}

void *consumer(void *arg) {
int i, tmp = 0;
while (tmp != -1) {
sem_wait(&full);          // Line C1
tmp = get();              // Line C2
sem_post(&empty);         // Line C3
printf("%d\n", tmp);
}
}

int main(int argc, char *argv[]) {
// ...
sem_init(&empty, 0, MAX); // MAX buffers are empty to begin with...
sem_init(&full, 0, 0); // ... and 0 are full
// ...
}
```

The producer and consumer threads.

Assume there is only 1 single CPU with 1 core.

1. Assume we have 1 producer thread, 1 consumer thread, and MAX is 1, does this program work? (Please say Yes or No first, if, and only if, you say No, then justify it with a detailed example). (10 points)

2. Assume we have 2 producer threads, 2 consumer threads, and MAX is equal to 3, what problem could happen? Please demonstrate the problem with an example. Draw a trace if you want. (10 points)

9. **Deadlock**. Below are two solutions to the producer/consumer problem: one is correct, one is broken - the broken solution could cause a deadlock situation. Which one is the broken solution? Explain how would it cause a deadlock situation. Draw a trace if you want. (10 points)

   The put and get routine for both solutions:

```
int buffer[MAX];
int fill = 0;
int use = 0;

void put(int value) {
buffer[fill] = value;    // Line F1
fill = (fill + 1) % MAX; // Line F2
}

int get() {
int tmp = buffer[use];  // Line G1
use = (use + 1) % MAX;  // Line G2
return tmp;
}
```

   Solution 1:

```
sem_t empty;
sem_t full;
sem_t mutex;

void *producer(void *arg) {
int i;
for (i = 0; i < loops; i++) {
sem_wait(&mutex);     // Line P0
sem_wait(&empty);     // Line P1
put(i);               // Line P2
sem_post(&full);      // Line P3
sem_post(&mutex);     // Line P4
}
}

void *consumer(void *arg) {
int i;
for (i = 0; i < loops; i++) {
sem_wait(&mutex);    // Line C0
sem_wait(&full);     // Line C1
int tmp = get();     // Line C2
sem_post(&empty);    // Line C3
sem_post(&mutex);    // Line C4
printf("%d\n", tmp);
}
}

int main(int argc, char *argv[]) {
// ...
sem_init(&empty, 0, MAX); // MAX buffers are empty to begin with...
sem_init(&full, 0, 0); // ... and 0 are full
sem_init(&mutex, 0, 1); // mutex=1 because it is a lock
// ...
}
```

   Solution 2:

```
sem_t empty;
sem_t full;
sem_t mutex;

void *producer(void *arg) {
int i;
for (i = 0; i < loops; i++) {
sem_wait(&empty);      // Line P0
sem_wait(&mutex);      // Line P1
put(i);                // Line P2
sem_post(&mutex);      // Line P3
sem_post(&full);       // Line P4
}
}

void *consumer(void *arg) {
int i;
for (i = 0; i < loops; i++) {
sem_wait(&full);       // Line C0
sem_wait(&mutex);      // Line C1
int tmp = get();       // Line C2
sem_post(&mutex);      // Line C3
sem_post(&empty);      // Line C4
printf("%d\n", tmp);
}
}

int main(int argc, char *argv[]) {
// ...
sem_init(&empty, 0, MAX); // MAX buffers are empty to begin with...
sem_init(&full, 0, 0); // ... and 0 are full
sem_init(&mutex, 0, 1); // mutex=1 because it is a lock
// ...
}
```

Answer: solution 1 is broken - locking too much. Only lock what you really really need to lock.