In-class Activity 2: Multithreading & Pthreads

Questions

Q1. Which of the following is a feasible schedule for the snippet of code given below, if the time points are ordered as t1 < t2 < t3 < t4 < t6 and t1 < t2 < t3 < t5 < t6.

a) Main thread starts = t1, Thread 1 starts = t1, Thread 2 starts = t1
Main thread terminates = t2, Thread 1 terminates = t2, Thread 2 terminates = t2

b) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t2
Main thread terminates = t3, Thread 1 terminates = t3, Thread 2 terminates = t3

c) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t4, Thread 1 terminates = t4, Thread 2 terminates = t4

d) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t6, Thread 1 terminates = t4, Thread 2 terminates = t5

Main thread

```
pthread_create(&pt1, NULL, sr1, NULL); // Thread 1

pthread_create(&pt2, NULL, sr2, NULL); // Thread 2

pthread_join(pt1 ,NULL);

pthread_join(pt2, NULL);

return 0;
```

void * sr1(void *arg)

```
printf(“Hello \n“);

return 0;
```

void * sr2(void *arg)

```
printf(“Greetings \n“);

return 0;
```

Q2. Which of the following is a feasible schedule for the snippet of code given below, if the time points are ordered as t1 < t2 < t3 < t4 < t5 < t7 and t1 < t2 < t3 < t4 < t6 < t7.

a) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t4, Thread 1 terminates = t5, Thread 2 terminates = t6

b) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t7, Thread 1 terminates = t5, Thread 2 terminates = t4

c) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t7, Thread 1 terminates = t4, Thread 2 terminates = t5

d) Main thread start = t1, Thread 1 start = t2, Thread 2 start = t3
Main thread terminates = t6, Thread 1 terminates = t4, Thread 2 terminates = t5

Main thread

```
pthread_create(&pt1, NULL, sr1, NULL); // Thread 1

pthreade_create(&pt2, NULL, sr2, NULL); // Thread 2

pthread_join(pt2, NULL);

return 0;
```

void * sr1(void *arg)

```
printf(“Hello \n“);

return 0;
```

void * sr2(void *arg)

```
printf(“Greetings \n“);

pthread_join(pt1, NULL);

return 0;
```

Q3. Assume that Thread 1 and Thread 2 may run in parallel/concurrently and that the shared variable done is initialized to 0. Please answer the following two questions considering the code snippet below:

a) Is there a data race? Explain.

b) What may go wrong? Explain.

Thread 1

```
do_the_work(); // accesses private data only
pthread_mutex_lock(&mtx);
done = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mtx);

pthread_exit(0);
```

Thread 2

```
while (done == 0) {
   pthread_mutex_lock(&mtx);
   pthread_cond_wait(&cond, &mtx);
   pthread_mutex_unlock(&mtx);
}

pthread_exit(0);
```

Q4. Which of the following may cause a thread to terminate prematurely, i.e., to terminate without executing a terminating statement. Please check all correct answers.

a) pthread_join
b) return (by the main function)
c) pthread_detach
d) pthread_exit
e) exit (by the main function)
f) exit (by a function that is executed by the main thread)

Q5. Which of the following is a correct (and not necessarily efficient) implementation of the producer consumer problem? Please check all correct answers.

a)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
          pthread_cond_wait(&cond, &mtx);
  // insert item and update
  pthread_cond_signal(&cond);
  pthread_mutex_unlock(&mtx);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
          pthread_cond_wait(&cond, &mtx);
  // remove item and update
  pthread_cond_signal(&cond);
  pthread_mutex_unlock(&mtx);
}
```

b)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
          pthread_cond_wait(&cond, &mtx);
  // insert item and update
  pthread_cond_broadcast(&cond);
  pthread_mutex_unlock(&mtx);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
          pthread_cond_wait(&cond, &mtx);
  // remove item and update
  pthread_cond_broadcast(&cond);
  pthread_mutex_unlock(&mtx);
}
```

c)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
          pthread_cond_wait(&full, &mtx);
  // insert item and update
  pthread_cond_signal(&empty);
  pthread_mutex_unlock(&mtx);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
          pthread_cond_wait(&empty, &mtx);
  // remove item and update
  pthread_cond_signal(&full);
  pthread_mutex_unlock(&mtx);
}
```

d)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
          pthread_cond_wait(&full, &mtx);
  // insert item and update
  if (numItems == 1)
    pthread_cond_signal(&empty);
  pthread_mutex_unlock(&mtx);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
          pthread_cond_wait(&empty, &mtx);
  // remove item and update
  if (numItems == N-1)
     pthread_cond_signal(&full);
  pthread_mutex_unlock(&mtx);
}
```

e)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
          pthread_cond_wait(&full, &mtx);
  // insert item and update
  if (numItems == 1)
    pthread_cond_signal(&empty);
  pthread_mutex_unlock(&mtx);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
          pthread_cond_wait(&empty, &mtx);
  // remove item and update
  pthread_mutex_unlock(&mtx);
  if (numItems == N-1)
     pthread_cond_signal(&full);
}
```

f)

Producer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == N)
           pthread_cond_wait(&full, &mtx);
  // insert item and update
  pthread_mutex_unlock(&mtx);
  pthread_cond_signal(&empty);
}
```

Consumer

```
while (true) {
  pthread_mutex_lock(&mtx);
  while (numItems == 0)
           pthread_cond_wait(&empty, &mtx);
  // remove item and update
 pthread_mutex_unlock(&mtx);
 pthread_cond_signal(&full);
}
```