

0/3 Questions Answered

Check-in Quiz07, mutex and condition variables

Q1 Mutex

8 Points

Consider the following code that is a modified version of some of the lecture sample code.

We are writing a program that supports 1 producer thread running `producer_thread()` that reads strings from `stdin`, and pushes them onto a double ended queue `deque`. There is at least one consumer thread running `consumer_thread` that will wait till there is a string on the `deque` before removing it and printing the value. There can be multiple consumer threads.

One big difference though is that we also have a boolean variable that the producer thread can set, which will let the consumer threads know they should exit.

We already have the mutex for the deque handled. Your job is to make sure the `bool_lock` is properly acquired and released so that no two threads access the boolean `eof_read` variable at the same time.

Again, this is C++ code but only for the `deque` data structure so that we do not need to implement our own `deque`. We went over the data structure in lecture briefly. I hope it is mostly self explanatory, but **please post on Ed if you have questions about anything regarding the premise of this question.**

```
1 pthread_mutex_t deque_lock;
2 pthread_mutex_t bool_lock;
3
4 bool eof_read = false
5 deque<char*> strings;
6
7 void* producer_thread(void* arg) {
8
9     while (true) {
10         char* line = NULL;
11         size_t n = 0;
12
13         ssize_t val = getline(&line, &n, stdin);
14
15         // eof
```

```
16     if (val < 0) {
17
18         eof_read = true;
19
20         pthread_exit(NULL);
21
22     }
23     pthread_mutex_lock(&deque_lock);
24     strings.push_back(line);
25     pthread_mutex_unlock(&deque_lock);
26 }
27 return NULL;
28 }
29
30 void* consumer_thread(void* arg) {
31
32     while(eof_read == false) {
33
34         pthread_mutex_lock(&deque_lock);
35         while (strings.size() == 0) {
36             pthread_mutex_unlock(&deque_lock);
37
38             if (eof_read == false) {
39
40                 pthread_exit(NULL);
41
42             }
43
44             pthread_mutex_lock(&deque_lock);
45         }
46
47         char* to_print = strings.at(0);
48
49         strings.pop_front();
50         pthread_mutex_unlock(&deque_lock);
51         printf("%s\n", to_print);
52
53         free(to_print);
54
55     }
56
57 }
```

Q1.1 lock**4 Points**

On which lines, should we add the code

```
pthread_mutex_lock(&bool_lock);?
```

 14 17 19 21 31 33 37 39 41 43 46 54 56**Save Answer**

Q1.2 unlock**4 Points**

On which lines, should we add the code

```
pthread_mutex_unlock(&bool_lock);?
```

 14 17 19 21 31 33 37 39 41 43 46 54 56**Save Answer**

Q2 Condition Variables

2 Points

The following code creates two threads which each attempt to increment a global integer 100 times per thread. `main()` prints the global variable after joining both threads, which should be 200.

To make sure that there aren't any data races (bad thread interleavings). Travis decides to use a condition variable and mutex to make this work.

When this code is run, what happens?

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

const int LOOP_NUM = 100;

int sum_total = 0;
pthread_mutex_t lock;
pthread_cond_t cond;

void* thread_func(void* arg) {
    pthread_mutex_lock(&lock);
    pthread_cond_wait(&cond, &lock);
    for (int i = 0; i < LOOP_NUM; i++) {
        sum_total++;
    }
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&lock);

    return NULL;
}

int main(int argc, char** argv) {
    pthread_t thd1, thd2;

    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&cond, NULL);

    pthread_create(&thd1, NULL, thread_func, NULL);
    pthread_create(&thd2, NULL, thread_func, NULL);

    pthread_join(thd1, NULL);
    pthread_join(thd2, NULL);
}
```

```
printf("sum_total: %d\n", sum_total);

pthread_mutex_destroy(&lock);
pthread_cond_destroy(&cond);

return EXIT_SUCCESS;
}
```

Does not compile

Deadlocks/does not terminate

prints 200

Still has a data race, can't guarantee that the result is 200

Segmentation Fault

Save Answer

Save All Answers

Submit & View Submission ➤