**CS2106 Operating Systems**

**Lab 4**

**Introduction to Mutexes (ANSWER BOOK)**

|  |  |
| --- | --- |
| **STUDENT NUMBER 1** | **STUDENT NAME 1** |
| **STUDENT NUMBER 2** | **STUDENT NAME 2** |
| **STUDENT NUMBER 3** | **STUDENT NAME 3** |

**Question 1** (1 mark)

The value of glob at the end of main is:

20

**Question 2** (3 marks)

My changes are:

pthread\_create(&thread[i] ,NULL ,child, (void \*) i);

**Question 3** (3 marks)

The value of glob is/is not correct. This is why:

Not correct. Race conditions cause threads to overwrite each other’s updates

**Question 4** (3 marks)

The value of glob is now correct/not correct. This is why:

Correct. The mutex enforces blocking during the update, causing the race condition to be removed.

**Question 5** (4 marks)

Here are my modifications:

for(i=0; i<10; i++)

pthread\_join(thread[i], NULL);

**Question 6** (2 marks)

I see that the system prints are close to being in order, with iterations only incrementing when all threads have been printed.

**Question 7** (2 marks)

I see that the system prints are in the mess, with iterations varying and all the prints interweaving.

**Question 8** (4 marks)

lab4p3.c prints incorrectly because..

The child processes do not know that they are being context switched, which results in messy prints due to the buffer being updated halfway. Furthermore, they are reading from a volatile global variable which results in race conditions in updating and reading.

The mutexes fix the problem because..

They enforce a queuing process. As a result, the threadCode thread blocks the other threadCode threads from executing and updating the count and iteration. As a result, no other process is able to write to the buffer. The only way the mutex is unlocked is after the writeSerialThread thread writes out the buffer. (which is the correct content). This makes the processes behave synchronously as

* Write to buffer, increment count, take len of count
* Print all the characters in the buffer

**Question 9** (3 marks)

My modifications are:

Add global variable:

**pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;**

In WriteLog fn:

pthread\_mutex\_lock(&mutex);

before writing to buffer

in loggerThread:

pthread\_mutex\_unlock(&mutex);

after writing to log.txt

They work because:

The threads will be blocked from updating the log buffer while the buffer already has content, and the mutex is only unblocked after the buffer contents are written to log.txt and flushed.

**TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / 25**