ICT1012
OPERATING SYSTEMS

LABORATORY INSTRUCTIONS

Quiz 2 – Question 2

Contents

Lab extension task (4 marks) 2

Prerequisite: 2

Task: Ubuntu “*pthread*” Deadlock & Race Condition (Add Missing Mutex Locks/Unlocks) 2

Task Requirements 2

Task Testing 3

NOTE: The grade script does not cover all the possibilities and it is your responsibility to carry out further testing 4

Quiz Submission 4

Lab extension task (4 marks)

Prerequisite:

Download the “**quiz2-q2.zip**” file from “LMS/xSiTe” folder:

“Lab Quizzes (Prerequisites) / Quiz 2 / Question 2”

Task: Ubuntu “*pthread*” Deadlock & Race Condition (Add Missing Mutex Locks/Unlocks)

Task Requirements

1. You are given a buggy (without synchronization) “*pthread*” based program “**notxv6/pthread_locks.c**”.
2. The program simulates access to **two shared global resources**:

   - resourceA (protected by lockA).
   - resourceB (protected by lockB).

   There are **three threads**:

   - worker_AB: updates A then B.
   - worker_BA: updates B then A.
   - worker_monitor: periodically reads *resourceA* and *resourceB* and sometimes adjusts them, then signals stop via “*done*”.

3. Threads receive parameters from “main()” through a struct pointer and return a heap-allocated result struct back to “main()”.
4. The program currently has:

   - **Race conditions** (shared variables accessed without synchronization).
   - If students add locks incorrectly (inconsistent order), it can still **deadlock intermittently.**

     Your job is to **add all missing mutex lock/unlock calls** and enforce a safe locking discipline.

5. Task — Add **missing mutex lock/unlock calls**

You must fix the program by adding "*pthread_mutex_lock()*" / "*pthread_mutex_unlock()*" calls.

Constraints

- You may only use the **two provided mutexes**:
    - lockA (for resource A)
    - lockB (for resource B)
- Do not add new mutexes, semaphores, or condition variables.

Correctness requirements

Your fixed program must:

1. Have **no data races** on:
    - resourceA, resourceB, total_ops, done
2. **Never deadlock** and always terminate.
3. Ensure the monitor's printed snapshots are taken safely (consistent read).

Task Testing

1. This program runs in Unix and not in xv6

   Executing "*pthread_locks*" from **Unix** command line:

```
$ cd notxv6/
$ make pthread_locks
cc     pthread_locks.c   -o pthread_locks
$ ./pthread_locks
[monitor] A=29943 B=29943 total_ops=40000 done=0
[monitor] A=60000 B=60000 total_ops=80000 done=0
[monitor] A=60000 B=60000 total_ops=80000 done=1

--- Results (returned to main) ---
Thread 1 ops=40000 lastA=59248 lastB=59248 lastTotalOps=79248
lastDone=0
Thread 2 ops=40000 lastA=60000 lastB=60000 lastTotalOps=80000
lastDone=0
Thread 3 ops=0 lastA=60000 lastB=60000 lastTotalOps=80000 lastDone=1

Final Shared: A=60000 B=60000 total_ops=80000 done=1
Expected: total_ops=80000 done=1
```

2. Testing with optional Unix shell script "**notxv6/optional_test**":
   Note: This will take few seconds as the "*optional_test*" script runs the binary multiple times (20) to catch intermittent issues.

```
$ ./optional_test
PASS
$
```

3. Testing with the grading script "**grade-quiz**":

   Note: This will take few seconds as the grade script runs the binary multiple times (20) to catch intermittent issues.

```
$ cd ..
$ ./grade-quiz pthread_locks
== Test pthread_locks_test == gcc -o pthread_locks -g -O2 -DSOL_THREAD
-DLAB_THREAD notxv6/pthread_locks.c -pthread
pthread_locks_test: OK (28.1s)
$
```

4. Testing with the "**make grade**"
   Note: This will take few seconds as the grade script runs the binary multiple times (20) to catch intermittent issues.

```
$ make clean && make grade
== Test pthread_locks_test == make[1]: Entering directory
'/mnt/c/ICT1012/xv6labs-w8-quiz2/q2_instructor'
gcc -o pthread_locks -g -O2 -DSOL_THREAD -DLAB_THREAD
notxv6/pthread_locks.c -pthread
pthread_locks_test: OK (28.1s)
Score: 1/1
$
```

NOTE: The grade script does not cover all the possibilities and it is your responsibility to carry out further testing

Quiz Submission

1. After "***make grade***" and "***make zipball***", submit the "***lab.zip***" file to Gradescope Assignment "***xv6labs-quiz2-q2-pthread***" for autograding.
   Note: Create the zip file with "***make zipball***" and not with Windows zip program or iOS Files App to avoid zipping multiple hierarchical directories.
2. Autograder will execute the same "*make grade*" when you submit your "*lab.zip*" file. This is for submission verification only. The score will not be included in the total score.
3. Test case script may not cover all the possibilities and it is your responsibility to carry out further testing,
4. After the quiz due date and time, all submissions will be regraded again with different set of test cases.

5. To ensure your submission goes through without technical issues, please submit at the earliest.