OS Lab 7 Assignment

23.10.2020

1 Introduction

Threads are execution entities like processes but are extremely light weight. They are used extensively for parallelization of large problems. Threads share the same code segments and data segments, hence switching between threads has much lesser overhead than switching between processes. Each Thread has separate stacks and registers.
Thread synchronization is used for concurrent execution of two or more threads that share critical resources. Threads should be synchronized to avoid conflicts while using critical resources. Otherwise, conflicts may arise when parallel-running threads attempt to modify a common variable at the same time, leading to unwanted results.
A mutex is a lockable object that is designed to signal when critical sections of code need exclusive access, preventing other threads with the same protection from executing concurrently and access the same memory locations.
In parallel computing, a barrier is a type of synchronization method. A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.
The basic barrier has mainly two variables, one of which records the pass/stop state of the barrier, the other of which keeps the total number of threads that have entered in the barrier. The barrier state was initialized to be "stop" by the first threads coming into the barrier. Whenever a thread enters, based on the number of threads already in the barrier, only if it is the last one, the thread sets the barrier state to be "pass" so that all the threads can get out of the barrier. On the other hand, when the incoming thread is not the last one, it is trapped in the barrier and keeps testing if the barrier state has changed from "stop" to "pass", and it gets out only when the barrier state changes to "pass".

In this week's first assignment we will be mainly focusing on the implementation of threads in xv6 and implement switching between user threads. In the second and third assignments we will see some common difficulties that can arise due to improper implementation of threads, and we try to resolve it using mutex and barriers.

2 Problem 1

This assignment is about thread implementation is xv6. Your task is to implement a context-switch mechanism for user-level threads. Before starting the assignment do the following:

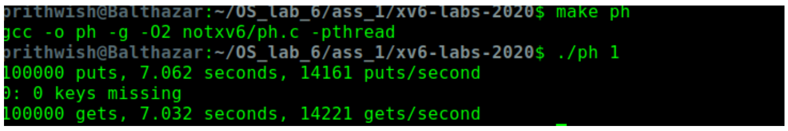
```
$ git clone git://g.csail.mit.edu/xv6-labs-2020
$ cd xv6-labs-2020
$ git fetch
$ git checkout thread
$ make clean
```

After completing the steps look into the user/uthread.c and user/uthread_switch.S files. In uthread.c most of the threading package has already been implemented and code for three simple test threads has been given.

2.1 Task

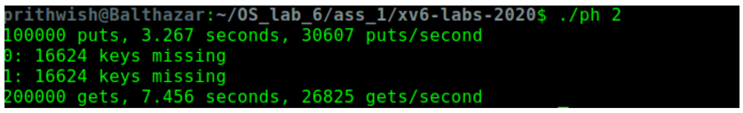
You have to complete the thread_create() and thread_schedule() methods. In uthread_switch.S you have to add the code for switching the thread context. This will be called from the thread_scheduler() in uthread.c.
Tentative Output :(Might not be in the same order)

```
$ uthread
thread_a started
thread_b started
```



```
prithwish@Balthazar:~/OS_lab_6/ass_1/xv6-labs-2020$ make ph
gcc -o ph -g -O2 notxv6/ph.c -pthread
prithwish@Balthazar:~/OS_lab_6/ass_1/xv6-labs-2020$ ./ph 1
100000 puts, 7.062 seconds, 14161 puts/second
0: 0 keys missing
100000 gets, 7.032 seconds, 14221 gets/second
```

Figure 1: No Key missing when running on single thread



```
prithwish@Balthazar:~/OS_lab_6/ass_1/xv6-labs-2020$ ./ph 2
100000 puts, 3.267 seconds, 30607 puts/second
0: 16624 keys missing
1: 16624 keys missing
200000 gets, 7.456 seconds, 26825 gets/second
```

Figure 2: Key missing when running with 2 thread

```
thread_c started
thread_c 0
thread_a 0
thread_b 0
..........
thread_c 99
thread_a 99
thread_b 99
thread_c: exit after 100
thread_a: exit after 100
thread_b: exit after 100
thread_schedule: no runnable threads
$
```

2.2 Questions

1. While between two threads what is saved as a part of context?
2. Explain the changes that you made in uthread.c and uthread_switch.S?
3. Can an user level thread exist after termination of the containing process?

3 Problem 2

In this problem we explore parallel programming with locks in Linux or MacOS(not xv6). To do this assignment you have to go through the different pthread operations in the pthread library.
In the notxv6/ph.c file, threads are used to insert and retrieve keys from a hashtable, when run with single thread($ ./ph number of threads) Fig 1, we see that no key is missing. But when run with 2 threads we get output similar to Fig 2. We see that there is a 2.2x speedup in the puts/second, but there is a lot of key that is missing while retrieving(gets). The puts/sec,gets/sec and time changes from system to system based on number of cores, existing load and speed.

3.1 Tasks

1. Make sure that there is no key missing while using 2 threads
Hint: use pthread_mutex_t and the init,lock,unlock operations associated with it
2. After completing 1 ensure that the speedup in puts/sec is at least 1.25x the puts/sec rate with single thread.

3.2 Questions

1. What are the different operations you used on mutex for this assignment?
2. Explain briefly why keys were missing initially when running 2 threads?
3. What changes did you make to avoid missing any keys?

4. Position in which you apply lock and unlock on mutex impacts the puts/sec, justify where you used them to maximize the puts/sec.

4 Problem 3

Barrier is a point in the execution of application where each participant threads must wait until all the other threads have reached that point too. The barrier function in notxv6/barrier.c is incomplete. Hence when you execute it thread it fails the assertion test.

```
$ make barrier
$ ./barrier 2
barrier: notxv6/barrier.c:42: thread: Assertion ‘i == t’ failed.
```

4.1 Task

1. Implement the barrier function properly so that no thread leaves before the other.
Hint :

```
pthread_cond_wait(&cond, &mutex); // go to sleep on cond, releasing lock mutex, acquiring upon wake up
pthread_cond_broadcast(&cond); // wake up every thread sleeping on cond
```

4.2 Question

1. Give an example of a scenario where you have to use barrier. 2. How did you implement barrier for the current problem?

5 Submission

Compress the whole xv6-labs-2020 folder as xv6-labs-2020.tar.gz. Create a file answers_threads.txt containing the answers to the questions asked. Compress the xv6-labs-2020.tar.gz and answers_threads.txt into {Roll_No}_LAB7.tar.gz.