Tutorial 6

CS3103 Operating Systems

Student Name: _____

Student No. : | | | | | | | | |

Day: ☐Tuesday ☐Wednesday ☐Friday

Time: ☐ 10:00 - 10:50 ☐ 11:00 - 11:50 ☐ 16:00 - 16:50 ☐ 18:00 - 18:50

Deadlocks

Introduction

Topics to be covered in this tutorial include:

- Explore some real code that deadlocks (or avoids deadlock).
- The different versions of code correspond to different approaches to avoiding deadlock in a simplified `vector_add()` routine.

Acknowledgement

This tutorial was adapted from OSTEP book written by Remzi and Andrea Arpaci-Dusseau at the University of Wisconsin. This free OS book is available at http://www.ostep.org.

Getting Started

1. Logging in to the Linux server

- Start the SSH client, e.g., MobaXterm or Xshell.
- Log in to the Linux server using the following details:

  Host Name: gateway.cs.cityu.edu.hk
  User Name: your EID (e.g., cctom2)
  Password: your password

Your password will not be shown on the screen as you type it, not even as a row of stars (******).

NOTE: The shell will always give you a prompt if it is ready to accept commands. The shell prompt normally ends in a `$` sign as we use in this tutorial. Some shell prompts end in `%` or `>` instead. Never copy/type the shell prompt used in this tutorial.

NOTE: Please don't forget to log out (use the `exit` command) after you finish your work.

2. Getting the real code

This tutorial lets you play around with a number of ways to implement a small, deadlock-free vector object in C. The vector object is quite limited (e.g., it only has `add()` and `init()` functions) but is just used to illustrate different approaches to avoiding deadlock.

Start by copying the codes to a directory in which you plan to do your work. For example, to copy *tutorial6* directory and its contents (10 files) to your current directory and change to it, enter:

```
$ cp -rf /public/cs3103/tutorial6 .
$ cd tutorial6
```

The last dot/period (.) indicates the current directory as destination.

Introduction of the code

Some files that you should pay attention to are as follows. They, in particular, are used by all the variants in this tutorial.

- `mythreads.h`

  The usual wrappers around many different pthread (and other) library calls, so as to ensure they are not failing silently.

- `vector-header.h`

  A simple header for the vector routines, mostly defining a fixed vector size and then a struct that is used per vector (`vector_t`).

- `main-header.h`

  A number of global variables common to each different program.

- `main-common.c`

  Contains the `main()` routine (with arguments parsing) that initializes two vectors, starts some threads to access them (via a `worker()` routine), and then waits for the many `vector_add()`'s to complete.

The variants of this tutorial are found in the following files. Each takes a different approach to dealing with concurrency inside a "vector addition" routine called `vector_add()`; examine the code in these files to get a sense of what is going on. They all use the files above to make a complete runnable program.

- `vector-deadlock.c`

  This version blithely grabs the locks in a particular order (dst then src). By doing so, it creates an "invitation to deadlock", as one thread might call `vector_add(v1, v2)` while another concurrently calls `vector_add(v2, v1)`.

- `vector-global-order.c`

  This version of `vector_add()` grabs the locks in a total order, based on address of the vector.

- `vector-try-wait.c`

  This version of `vector_add()` uses `pthread_mutex_trylock()` to attempt to grab locks; when the try fails, the code releases any locks it may hold and goes back to the top and tries it all over again.

- `vector-avoid-hold-and-wait.c`

  This version of `vector_add()` ensures it can't get stuck in a hold and wait pattern by using a single lock around lock acquisition.

- `vector-nolock.c`

This version of `vector_add()` doesn't even use locks; rather, it uses an atomic fetch-and-add to implement the `vector_add()` routine. Its semantics (as a result) are slightly different.

Type "make" (and read the Makefile) to build each of five executables.

```
$ make
```

Then you can run a program by simply typing its name:

```
$ ./vector-deadlock
```

Each program takes the same set of arguments (see `main-common.c` for details):

```
-d
   This flag turns on the ability for threads to deadlock.
   When you pass -d to the program, every other thread calls vector_add()
   with the vectors in a different order, e.g., with two threads, and -d
    enabled, Thread 0 calls vector_add(v1, v2) and Thread 1 calls
    vector_add(v2, v1)
-p
   This flag gives each thread a different set of vectors to call add()
   upon, instead of just two vectors. Use this to see how things perform
   when there isn't contention for the same set of vectors.
-n num_threads
   Creates some number of threads; you need more than one to deadlock.
-l loops
   How many times should each thread call vector_add()?
-t
   Turns on timing and shows how long everything took.
```

Questions

See the answer sheet file. All questions should be answered on the separate answer sheet provided.