Programming Assignment 1 - Mutual Exclusion and Locking Basics

SOIC B524 - Parallelism in Programming Languages and Systems
Indiana University

Spring 2017

1 Introduction

In this assignment you will learn to use Clang's *thread sanitizer*, implement two basic locking strategies, enforce mutual exclusion in an example program, and benchmark the results.

2 Setup

Begin by unpacking the archive `pa1-src.tar.xz` and extracting the code handout directory using the command `tar -xJf pa1-src.tar.xz`. This will create the directory `pa1-src`. Once you have entered the directory you will find several `C/C++` source files, a file called `CMakeLists.txt`, and the shell script `setup.sh`.

CMake is a configuration system that will generate a build system customized for the environment you are building in. Normally the `cmake` tool would be invoked manually, but we have provided the `setup.sh` file to automate this process for you. Once you run the script from the `pa1-src` directory you will find a new directory called `build`, with several sub-directories for different build types (e.g. debug, release, et cetera). Inside each of these sub-directories you can use the `make` tool to build several executables from the file `pa1.cpp`. The only difference between the produced executables is what the type `lock_t` is defined as:

1. `pa1-dummy` - A dummy lock that does nothing.
2. `pa1-std` - A wrapper around the `std::mutex` lock.
3. `pa1-petersons` - Your implementation of Peterson's Filter Lock.
4. `pa1-bakery` - Your implementation of the Bakery Lock.

The programs take three arguments: the number of threads to spawn, the number of rounds to perform, and the size of the data vector. The default values for these parameters are 4, $4 * \text{num\_threads}$, and 1024 respectively. To run the `pa1-std` program with 4 threads, for 100 rounds, with a data size of 2048 you would use the command '`./pa1-std 4 100 2048`'.

3 Instructions

As you follow the instructions bellow you will be asked to answer several questions. Please enter your answers (with appropriate numbering) in a file called `pa1-answers.txt` and then follow the instructions bellow to submit it.

1. Open the file `pa1.cpp` and familiarize yourself with its contents.

   Q1) What common parallelism bug does this code appear to exhibit?

2. Enter into the `build/debug` build directory and compile the executables. Run the program `pa1-dummy` several times.

   Q2) Describe at least two different observed behaviors from the program. Does the bug identified in Question 1 explain the observed behavior? Explain your answer.

3. Read the documentation for Clang's Thread Sanitizer:
   `https://clang.llvm.org/docs/ThreadSanitizer.html`

   Q3) What common parallelism issue(s) does the Thread Sanitizer help to detect?

   Q4) What is the typical slowdown of an application when it is compiled with the Thread Sanitizer?

   Q5) What command line option needs to be passed to Clang to compile an executable with the Thread Sanitizer?

   Q6) What command line option is implied by the use of the Thread Sanitizer?

4. Switch to the `build/tsan` directory, compile the executables, and run `pa1-dummy` several times.

   Q7) What bug(s) was detected, and what line(s) of code does the tool blame for the bug(s)?

5. Open the files `locks.cpp` and `locks.h`. Several locations in these files are marked with `FIXME` comments. Implement Peterson's Filter Lock (pg. 28) and the Bakery Lock (pg. 30) by filling these locations in with function definitions and class members.

6. Notice that all of our locks conform to the same interface. This means that we can test multiple lock types using the same source program simply by re-defining `lock_t`, which is exactly what `pa1.cpp` does. Use the type `lock_t` to enforce mutual exclusion in `pa1.cpp`. The `pa1-dummy` program should still exhibit any previously present bugs, but the other executables should now be properly synchronized.

   Q8) Could the critical sections you identified be made larger? If so, what would be the advantages and disadvantages?

   Q9) Could the critical sections you identified be made smaller? If so, what would be the advantages and disadvantages?

7. Enter the `build/tsan` directory, compile the executables, and test your locks to ensure that the *thread sanitizer* no longer detects any errors.

8. Enter the `build/release` directory and compile the executables. Next, benchmark `pa1-petersons`, `pa1-bakery`, and `pa1-std` with the following parameters:

   (a) A fixed round count of 100

   (b) Data sizes 2^{10} (1024), 2^{18} (262144), and 2^{20} (1048576)

   (c) Number of threads ranging from 2 to 128, by powers of two

When benchmarking, run each program 6 times, drop the first time, and average the remaining times. Once you have collected your data create three line graphs, one for each data size, comparing the total elapsed time versus the number of threads.

Q10) On average, which lock performed best? Were there any situations where it was not the best lock?

Q11) Which lock had better performance: Peterson's Filter Lock or the Bakery Lock?

Q12) Was the relative performance of the locks the same for all data sizes? Explain why or why not.

4 Submitting the Assignment

To submit the assignment rename the directory `pa1-src` to `<username>-pa1-submission`, remove the `<username>-pa1-submission/build` directory, and compress the directory (with your pa1-answers.txt file included) into an archive. Next, upload the archive file as your submission on the assignment's page on Canvas.