LINGI2251
Software Quality Assurance
Assignment 2
Spring 2020

Igor Zavalyshyn & Charles Pecheur

March 24, 2020

Due Mon Apr 20

The objective of this assignment is to learn how to use a set of Valgrind tools to analyze and debug a C program. The program to be tested, and the tools to be used, are available online and are referenced from the website of the course.

This assignment can be performed in **groups of two students**. Individual submissions are also accepted. Interaction between students and groups is allowed but plagiarism will not be tolerated.

Assignment results will be returned as an archive file (`.zip` or `.gz`) whose base name is the last names of the students (e.g. `Pecheur_Cailliau.zip`, no accents please). The archive will at least contain a **report** covering all tasks below. If more files are provided, their use will be explained at the beginning of the report. Make sure to **include proper identification** (course, year, assignment number, student names) at the beginning of all documents.

Submit your deliverables in the *Assignments* section of the Moodle website, before the deadline stated above. Late submissions will not be accepted.

Please post any question regarding this assignment on the corresponding forum on the course website. For personal issues you may contact *Charles Pecheur*.

The subject: Buggy Hash Table

The software to analyze and debug is a C program ***bht.c*** that implements a simple hash table that supports just two methods *put* and *get*. The *main* method of the program fills in the table with random keys and values and then performs a lookup of these keys. The program relies on *pthread* to parallelize execution of both put and get operations depending on the number of threads used (specified as input argument).

To run a program, execute `./bht N`, where N is the number of threads that execute put and get operations on the hash table. The program inserts `NKEYS` into a hash table in the put phase and then looks them all up in the get phase. You will likely observe that the code is incorrect. When run with N>1 threads the application will likely report that some keys are missing. There may be even zero keys missing in some runs. If you run with one thread, there will never be any keys missing.

The main reason for missing keys, is of course the fact that the program was intentionally modified to introduce bugs and memory leaks. One of your tasks would be to find and fix those bugs.

Tools

The following software tools will be used in this assignment.

gcc — You will have to compile the source code of the application first. Use standard C compiler available for all the common platforms. We recommend to compile with *-pthread* flag for pthread support, as well as *-g* flag to facilitate debugging.

Valgrind — You will use Valgrind's tool suite[1] for debugging and profiling. With it you can automatically detect many memory management and threading bugs. From all the Valgrind tools we recommend to use *memcheck* and *helgrind* for this assignment, but feel free to use others as well if you want to.

You would need to compile the latest release version of Valgrind from source and install it in your system.

[1] `http://valgrind.org/`

Tasks

Have the tools downloaded, compiled and/or installed in your system.

Download the source code of Buggy Hash Table `bht.c` (from Moodle) and compile it[2].

Memory Bugs

The program contains few bugs that cause invalid read & writes as well as memory leaks. Use Valgrind's *memcheck* tool to find those bugs and fix them. After fixing, make sure that *memcheck* does not report any memory errors, i.e. there should be 0 (zero) errors reported in the `ERROR SUMMARY` section of *memcheck*.

Question 1: What was causing the invalid read & writes? How did you fix it?

Question 2: Explain the reason for the memory leak. Specify the part of the code causing it. How did you fix it?

Race Condition

Analyze the program with *helgrind* tool. Study helgrind's output and the code of `bht.c` program.

Question 3: Why are there missing keys with 2 or more threads, but not with 1 thread? Your explanation should refer to the given code and make use of theory to justify your answer. Identify a sequence of events that can lead to keys missing for 2 threads.

To avoid this sequence of events, insert lock and unlock statements where needed so that the number of keys missing is always 0. The relevant pthread calls are (for more see the manual pages, man pthread):

```
pthread_mutex_t lock; // declare a lock
pthread_mutex_init(&lock, NULL); // initialize the lock
pthread_mutex_lock(&lock); // acquire lock
pthread_mutex_unlock(&lock); // release lock
```

Modify `bht.c` to make it correct and recompile with gcc. Test your code first with 1 thread, then test it with 2 threads. Is it correct (i.e. have you eliminated missing keys)? Check the correctness using helgrind. (Note that valgrind slows down `bht` a lot, so you may want to modify `NKEYS` to

[2]`gcc -pthread -g -o bht bht.c`

be some small number. Additionally, due to the slowdown introduced by valgrind, you may not see any missing keys. Make sure to check `bht` both with valgrind and on its own)

Question 4: Describe your changes and argue why these changes ensure correctness.

Question 5: Is the two-threaded version faster than the single-threaded version in the put phase? Report the running times for the two-threaded version and the single-threaded version. (Make sure to undo your `NKEYS` change.)

Question 6: Most likely (depending on your exact changes) `bht` won't achieve any speedup. Why is that?

Living Dangerously

Remove the locks from `get()` in `bht.c`, recompile, and rerun the program.

Question 7: You should observe speedup on several cores for the `get` phase of `bht`. Why is that?

Question 8: Why does valgrind report no errors for `get()`?

Going Further

Question 9: Can you think of a way of modifying `bht.c` so that you get speedup for both phases and *Valgrind* won't report race conditions? Implement your plan and test it. Describe your changes in the report.

Credits

The original version of this assignment was prepared and compiled by the Electrical Engineering & Computer Science (EECS) department of Massachusetts Institute of Technology (MIT) for the "6.033: Computer Systems Engineering" course.