Homework
Part 2: Concurrency –
6) Threads and Locks

master@323240b (20230907-115823)

P. Mainini / E. Benoist / C. Fuhrer / L. Ith

BTI1341 / Fall 2023/24

1 Threads and Concurrency Issues

This exercise uses the source code provided in the course git or on Moodle. It also requires the ***`helgrind`*** *tool from the Valgrind suite ([val]). If it is not installed on your system, install it using "* ***`sudo apt install valgrind`****"!*

1.1 Understanding the Problem

1. Study `counter/counter.c` and try to fully understand it.
2. Compile the program using the provided makefile and run it with different parameters. Example:

```
$ make
<output omitted>
$ ./counter 1 10
$ ./counter 1 100
[...]
$ ./counter 1 100000

$ ./counter 4 10
$ ./counter 4 100
[...]
$ ./counter 4 100000
[etc.]
```

What can you observe? Are the results correct? If not, why?
Have you tried compiling without optimization (i.e. "`make debug`")? Does that change anything? If so, why?
Can you run more threads than cores in your CPU? What is the order of the threads?

1.2 Analyze Using Valgrind

Valgrind is a suite of tools for finding memory leaks, concurrency issues etc. We will now use it's `helgrind` tool to see if we can identify the problem automatically.

👉 *When using tools from Valgrind, it is best to compile the binary with debug symbols. This way, it can display the line numbers in your program where an error has occurred. Also, start with few threads and loops per thread, as it may generate a lot of output! Example:*

```
$ make clean && make debug
<output omitted>
$ valgrind --tool=helgrind ./counter 2 5
```

Read Valgrind's output *from top to bottom.* Are there any concurrency issues? Where do they occur and why?

👉 *Some versions of `helgrind` may generate false positives in `printf()`! Only look out for symbols (variables) found in `counter.c`.*

1.3 Fixing Shared Thread Parameters

One of the problems is the sharing of parameters between the counting threads and `main()`. Create some data structure which is *allocated* per thread to pass the parameters. At the same time, also eliminate any unneeded global variables.

👉 *When you are allocating data on the heap, it is a good idea to use Valgrind to check for memory leaks. Run the following command to see if there are any memory leaks in your implementation:*

```
$ valgrind --leak-check=yes --show-reachable=yes ./counter [...]
```

1.4 Locking the Counter

The counter must be incremented by all threads, thus it needs to be shared. To fix the data race on it, find all critical sections and add a mutex to prevent concurrent access by more than one thread.

When you are done, re-run `helgrind` to verify that the problem has been fixed!

1.5 Visualize Progress

Add a new thread that displays the current counter value periodically. You can use the `sleep()` function for this, although it is not generally safe to be used in multi-threaded applications.

References

[val] *Valgrind*, https://valgrind.org/.