CUT - cpu usage tracker

Michał Kukowski

May 2022

1 Requirements

- The project should be written in C (C99 or higher). In addition, code should perfectly follow the standard in this way that we could compile the project using gcc and clang (without any warnings, please use -Wall -Wextra in case of gcc and -Weverything in case of clang). Remember there are some obsolete warnings like manual padding. You can skip those kinds of warnings, but you need explain yourself.
- The build system should be created for this project. You can use Makefile or CMake. Build system should support both gcc and clang compilation mode (you can use CC variable from a shell).
- Use git or svn to create a development history. Please be aware of big commits without description. You should follow the rule: 1 commit = 1 functionality. Use free hosting to put there your code. I suggest to use github. You need to send a link to your repo to our HR team.
- The application has to work correctly on every Linux distribution. You can use Ubuntu, Arch, Fedora, or Debian to test your app.
- Use Valgrind, or another similar program to deal with memory leaks. The final version of your app cannot have any memory leaks.
- Your app need to have at least 1 automatic test. It can be an unit test written in C (build by command make test) or any other test written in C or any other script language which execute your whole program and check something using some rule.
- Please note that this is a multi thread app. This kind of app required a proper synchronization.
- During your technical interview you will be asked to present the app and the code. Maybe you will be doing some modification. **BE PREPARE FOR THAT**.

2 Description

The goal is to write a simple console application in C to track CPU percentage usage.

1. The first thread (Reader) reads /proc/stat and sends the string (as raw data) or a structure (fields from file needed for calculation) to the second thread (Analyzer),
2. The second thread (Analyzer) works on raw data and produces CPU usage (in percentages) for each CPU core from /proc/stat. Then thread sends CPU usage to the third thread (Printer),
3. The third thread (Printer) prints to the console (in format way, the format is up to you) average CPU usage in every one second. (You cannot print in per received data, you need to aggregate data and print only avg from 1s),
4. The fourth thread (Watchdog) tries to save the program from any deadlocks an other errors. If threads do not send any message to Watchdog in 2s, Watchdog should finish the application with the proper error message,

5. The fifth thread (Logger) receives messages from all threads and writes them into a file. Logger is useful to write down debug prints in a synchronized way,

6. You should implement also a SIGTERM handler to close the application gently. Close descriptors, finish threads, free memory, etc.

Mandatory functionality: Reader, Analyzer, and Printer. Optional functionality: Watchdog, Logger, and SIGTERM handler. I encourage you to develop all the functionality to make this project powerful as it was designed.

3 Hints and Tips

1. To calculate the CPU usage you can use this formula,

2. procfs is a virtual file system. It means that always the size of files under procfs have size equals 0. On top of that every single read can produce different output (even with different length). Be aware of this issue. Please read /proc/stat file several times, analyze the structure of the file and prepare safe way of reading data from this file,

3. Sending the data between threads can be tricky. You don't need a socket or something. You can use a global variable / structure,

4. Reading and sending the data in a concurrent world is a well-known problem. You can model this as a Consumer - producer problem. PCP solution with spin locks (i.e. reading queue size without idle time) will not be accepted,

5. Consider data buffering. What if the producer writing the data faster than the reader reading the data? Please solve this problem using a data structure like RingBuffer or Queue,

6. To implement concurrency in C you can use pthread library or an overlay embedded in language C11,

7. You can use this example to implement a signal handler,

8. Please note that we will evaluate not only correctness but also the programming style. Read carefully the Modern C and implement tips founded there. You can of course create your style. Remember that there is no best style in a programming world. Try to play with some C99 features like VLA, FAM, Compund literals. Moreover, try to find weak points strong points of each feature,

9. Try to design code using OOP principles like KISS, DRY and SOLID,

10. To write an unit test you don't need a third part library. You can use assert(),

11. To be able to write a tests, you need a proper design. Split the app into several modules. Each module should be tested. The app layer should run a thread and use tested functionality from modules,

12. If you are implementing logger, please write some useful messages. Example: when watchdog killed the program, you should have a message with the root cause to make debug simpler. Of course you are deciding what kind of message is useful for you.