Thread Calc

Cyber Solutions Development - Tactical

October 4, 2021

Abstract

This is a mini project that is designed to extend the functionality of your FileCalc submission. It will operate in exactly the same way as FileCalc, but will utilize a threadpool for parallel job processing.

1 Requirements

In this assignment, you will extend the functionality of FileCalc with a threadpool. This should require little to no modification of a file parser, but will force you to use a Queue data structure.

1.1 Basic Requirements

1. Written in C
2. Everything FileCalc does
3. Utilize one of the previously created Data Structures to store read and store work
4. Utilize a threadpool to process work in the queue
5. Output files in the same way FileCalc does

1.2 Specific Requirements

1. Utilize the Queue from DataStructures1
2. Utilize a Threadpool to process work from the queue
3. Utilize getopt to process the following arguments

   -n NUM (number of threads in the pool; defaults to 4)

1.2.1 Data Structures

Trainee dependent. At this point, you should have all the tools you need to solve this in a logical and straightforward way.

1.2.2 Memory Management

However you chose to solve it, your program should not be leaking memory. Your program should show no memory leaks with:

```
valgrind --leak-check=full ./threadcalc <in_dir> <out_dir> <-n optional
    thread num>
```

1.2.3 Thread Management

You must implement a well designed and fully functional threadpool. Your program should not have deadlocks, race conditions, or odd failure states. Your program should close appropriately if the user stops it with CTRL-C, and not hang waiting for IO Your program should exit cleanly after calculating all files in the directory

1.2.4 Assumptions

1. All numbers in EQU spec are little endian
2. All valid files will have a valid Magic Number
3. All Equation IDs are unique (they are unique enough)

1.2.5 FileFormat

1. No Changes from FileCalc - Refer to FileSpec

1.3 File Reading Guidelines

1. If you encounter Errors in the Equ Header (ie invalid magic):

   Log an Error to stderr

   Move the file to a BadFiles directory

   Continue Parsing other files

2. If you encounter Errors in Individual Equations:

   Use the Flags field to indicate it was unsolved

   Continue Parsing other Equations

2 Deliverables

Your code should have the following file structure:

```
4_ThreadCalc
└── src
    └── source-files
```

```
├── hdrs
│   └── header-files
├── references
│   └── documentation
└── CMakeLists.txt
```

Your code should build and compile with the following shell script ran from the ThreadCalc directory:

```
// build.sh
mkdir build
cd build
cmake ..
make
```

The build script has the same requirements as the SimpleCalc build script.

3 Notes to grader

The purpose of this project is to bridge the gap between FileCalc and NetCalc. Throwing in threadpools and networking in one project was a little much.

Although you ultimately control the trainees JQR completion status, allow them to complete the projects using whichever means they want (within reason). Engineers need to be able to break down problems, and iterate on solutions. If we say "you will do X and Y and Z to solve this" we risk putting the trainees in a box where they can't fully flex their problem solving muscles.

/pagebreak

4 JQR Sections Potentially Covered - Mentor/Trainee dependent

NOTE: These projects are designed to give each trainee room to think creatively. Although the reference spec may target specific JQR requirements, it is up to the trainee to pick which JQR items they want to solve, and it should be indicated somewhere in the trainees documentation. It is up to the mentor to determine which JQRs have been satisfied and which need more work.

By this point, your trainee should absolutely be able to point to specific JQR line items, and where they are in the codebase