

File Choice Calc

Cyber Solutions Development - Georgia

September 19, 2020

Abstract

Your task is to use the same parser you built for FileCalc, but this time the grader will ask for only a subset of the files in the `input_dir` using unix domain sockets.

1 Requirements

In this assignment, you will build an application that will perform the same operations as FileCalc, but will also take input from the grader over a unix domain socket.

1.1 Basic Requirements

1. Written in C
2. Everything FileCalc does
3. Uses socket communications to a unix domain socket
4. Uses a linked-list to hold information pertaining to each file
5. Uses a stack to compute each equation in different formats
6. Single-threaded (No multi-threading)

1.2 Specific Requirements

1.2.1 Data Structures

Use a linked-list data structure that will hold each file's data in a series of nodes. The choice of linked-list is up to you, but you must read in each file and store it using a linked-list. The follow procedure is what you should expect:

1. Read in each file as in FileCalc.
2. Store each file into a node in your linked-list.

3. Wait for a file id from the grader. If the id does not exist, print error but do not exit. If the id does exist, remove the item from the linked-list, solve the equations, and output file to the output directory. If you get the id: 0xDEADBEEF, that is the signal to cleanup and exit.

You will also have to handle equations in other formats from standard notation:

1. Prefix: OPERATOR OPERAND1 OPERAND2
2. Postfix: OPERAND1 OPERAND2 OPERATOR
3. Standard: OPERAND1 OPERATOR OPERAND2

Your solution to implementing each notation should use a stack data structure implemented with either a linked-list or an array. Think about why you don't need to specify in the file header what notation the equation is in.

1.2.2 Socket Communications

The grader will be listening on a unix domain socket, and when it receives a connection, it will go ahead and supply you a formatted list, containing the file IDs it wants solved. Your program should look for a socket file called *grader-socket* located in the */tmp* directory. If the socket file is not present, report an error, released resources and exit. Otherwise go ahead and establish a connection to it. If the provided ID does not exist, report ID_NOT_FOUND, otherwise report STATUS_GOOD. Use a single send at the end using the REPORT_STATUS command. See **protocol-specification** for more information.

BONUS: Ensure the socket file in the */tmp* is actually a socket file. Research what system call will be useful for determining this.

BONUS: Include an option "-s" that will allow the binary to function as the server that the grader will connect to. Create the socket file, bind to the socket file as stated above, and wait for the grader to connect. Do not worry about timeouts or other socket options. This will include sections 3.1.12 (f, h)

1.2.3 Memory Management

Your program should not be leaking memory. Your program should show no memory leaks with:

```
valgrind --leak-check=full ./filecalc input_dir output_dir
```

1.2.4 Assumptions

1. All numbers are little endian
2. All numbers are 64-bits in size
3. All numbers should be treated as signed (int64_t)

2 Deliverables

Your code should have the following file structure:

```
FileCalc
├── src
│   └── source-files
├── hdrs
│   ├── file-calc.h
│   └── file-calc-protocol.h
├── docs
│   └── documentation
└── CMakeLists.txt
```

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

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

3 Notes to grader

The purpose of this assignment is to start using simple data structures in C. Use this assignment to achieve the following objectives:

1. Perform file input and output operations using system calls
2. Building on CMake knowledge. Your mentee should create a linked-list implementation before this project. Have them build their implementation as a static library and link it using cmake to the project.
3. Code organization. This is complex project, and code organization will be key. Ensure your mentee is leveraging good practices for code organization.
4. This project is a step towards building to network communications. The grader acts as an listening post (LP) that sends commands and the C program responds to these tasks just as an implant would.
5. This project introduces data structures. Please ensure your mentee is prepared with the necessary theoretical background and has implemented a working linked-list. Consider making successful completion of 3.3.1 a pre-requisite for this project.

4 JQR Sections Covered

- All sections in FileCalc
- 3.1.12 (a, b, c, d, e, g, p)
- 3.3.2 / 3.3.3 / 3.3.4 (Depending on which linked-list is chosen)
- 3.3.9 (all)