CS 2110 Project 4: C Programming

Camille Bossut, Sean Crowley, Charlie Gunn, Christopher Turko, Justin Hu, Manley Roberts

Section A, Fall 2020

Contents

1 General

1.1 Introduction

Hello and welcome to Project 4. In this project, you'll write a C program to interact with a toy file-system of documents.

The project is due November 9th, 12:00PM EST

Please read through the entire document before starting. Often times we elaborate on instructions later on in the document. **Start early** and if you get stuck, there's always Piazza or office hours.

1.2 Background

For this project, you are given the header file `project4.h`, and a shell of the C file `project4.c`. The header file contains the backbone of the toy file-system you will be implementing commands for, as well as three macros which will greatly aid in your implementation for several functions. Keep in mind that you will have to implement these macros.

```
doc_t doc_system[MAX_NUM_DOCS];
char data[MAX_DOCSIZE * MAX_NUM_DOCS];
uint64_t doc_valid_vector;

// Implement this macro; it should evaluate to a non-zero (true) result if and only if
// the "index"th bit is set in doc_valid_vector, which would imply that a doc
// is already present at index.
#define GET_DOC_PRESENT(doc_valid_vector, index)

// Implement this macro; it should evaluate to a copy of doc_valid_vector with the
// "index"th bit set, showing that a doc is now present at index.
#define SET_DOC_PRESENT(doc_valid_vector, index)

// Implement this macro; it should evaluate to a copy of doc_valid_vector with the
// "index"th bit cleared, showing that no doc is now present at index.
#define CLEAR_DOC_PRESENT(doc_valid_vector, index)
```

Firstly, we have an array of `doc_t` structs which represent all of the documents that currently reside in the file-system. However, the `doc_t` struct does not actually contain the data for a document – all document data is stored in the `char` array `data`. This character array is thought of as partitioned into many sections of length `MAX_DOCSIZE`, where each section may or may not contain a document. The 64-bit integer `doc_valid_vector` is a bitvector whose nth bit (starting from the LSB) indicates whether or not there is a document at the nth slot of the `data` array. You will need to update this bit vector as your document system changes.

The header file also defines various constants, the `ext_t` enum, and the `doc_t` struct.

```
typedef enum extension {
    DOC = 0,
    ASCII = 1
} ext_t;

typedef struct document_t {
    char name[MAX_NAMESIZE];
    ext_t extension;
    uint8_t permissions;
    char *data;
```

```
    int size;
    int index;
} doc_t;
```

The extension enum is largely self-explanatory: it indicates that the only file extensions that are valid for documents in our file-system are `.doc` and `.ascii`. These extensions do not have any inherent meaning, since all of our documents are simply represented by character data.

The document struct is more involved. It contains its own name and extension (e.g., `example.ascii` would have the name "example" and the extension `ASCII`). Furthermore, it contains an 8-bit integer `permissions`, the first 3 bits of which represent whether or not the user has read, write, and execute permissions for the file. Executing a document has no meaning in this project, but you will have to consider read and write permissions when implementing various commands. The struct also contains a `char` pointer, which points to the first character of the document in the `data` array. Finally, a document also contains its own size (i.e. the number of chars in the document) and its position in the `data` array (note that this index could be calculated from the `char` pointer, but it is convenient to store it in the struct).

2 File-system Commands

The bulk of your work for this project will be implementing various functions that act on the file-system. The specifications for these functions are listed briefly in the shell code itself, but are explained with more detail in the sections below.

Many of these functions return an integer flag which indicates success or failure. We have provided the macros `#define ERROR -1` and `#define SUCCESS 0` to help you. Also, any function that receives a pointer as input should error if the pointer is `NULL`.

Furthermore, we have provided you with some helper functions in `helper.c`. Feel free to take a look at them to determine their functionality. In the specifications below, we mention any helper functions that will be useful when coding that command. Finally, please do not create your own global variables! Everything you need is given to you.

2.1 New

```
int new_doc(char* docname);
```

This function creates a new, empty, document and adds it to the filesystem. It should return -1 if there is no space left in the filesystem for a new document, or if a document with the given name already exists. Otherwise, it should add the new document at the first available slot in the file-system. Files created with this function should have all permissions (i.e. the read, write, and execute bits should all be set to 1). To get the name and extension of the new document, you should use the `get_name_ext` function that has been implemented for you.

After successfully creating a document, return its index.

2.2 Find

```
int find_doc(const char* docname);
```

This function returns the index of the document in the file-system with the given name, returning -1 if such a document does not exist. Once again, you should use the `get_name_ext` function that has been implemented for you. Once you split the given docname into a name and extension, make sure to compare each of them to each document in the docsystem.

2.3 List

```
void list(void);
```

When this function is called, you should print out all valid documents in the file-system (in index order). You should use the `print_doc_name` function that has been implemented for you.

2.4 Append

```
int append(doc_t *doc, char *str);
```

The `append` function takes a pointer to a document and a string, and expects the string to be appended onto the document. This action is only allowed if the document has write-permissions, and there is enough space for the entire given string to be appended (-1 should be returned if these conditions are not met, and 0 should be returned on success). You should consider using the C functions `strlen` and `strcpy` in your implementation.

(Hint: The data for each document in the `data` array is expected to be zero-terminated – take that into account when deciding whether there is enough space to append the given string. Since `MAX_DOCSIZE` is the maximum total length of a document's data, `MAX_DOCSIZE - 1` is the maximum size of *actual* contents, since the zero-terminator will always take up one byte)

2.5 Import

```
int import(char* docname, FILE *file);
```

This function takes in a document name (which should be passed to `new_doc`) and a file-pointer. It should import the contents of the given file into the new document. If the contents of the file are too large to fit into a document, the function should fail (the newly created file should be removed, and you should return -1). The helper function `get_file_size` will be useful, and you should also use the C function `fread` in your implementation.

2.6 Print and Export

```
int print_doc_data(const doc_t *doc);
int export(const doc_t *doc, FILE *file);
```

The `print_doc_data` function should print the entire contents of the given document, followed by a newline. You should implement this command in the project4.c file. You will also have to implement `export`, which should print the contents to the given file-pointer with `fprintf`. The function should fail if the given document does not have read access, returning -1. Return 0 on success.

2.7 Remove

```
int remove_doc(doc_t* doc);
```

In this function, you should remove the document given as input. Note that you do not need to clear any memory or alter the `doc_system` array (although you will not be penalized for doing so). You only need to modify the `doc_valid_vector`.

2.8 Change Mode

```
int change_mode(doc_t *doc, char* mode_changes);
```

This function should change the permissions of the given document based on the `mode_changes` string. This string will have as a first character either `+` or `-`, indicating whether we are removing or adding permissions. After this sign, there will be a sequence of at most three characters from the set (r, w, x) indicating which permissions need to be added/removed.

3 The Interaction Loop

After you have completed all of the required functions, you can move onto the interaction loop, where you will implement the interface between the user of your file-system and the commands you wrote. Essentially, the interaction loop repeatedly takes text commands from the user, and performs the requested action. For example, if the user typed `new test.doc`, you would create a new document called `test.doc`. The user is repeatedly prompted for these commands until they type `exit`.

Inside the `my_main` function, we have already implemented the shell of how this will work, along with the specific implementation of the `new` and `exit` commands. To complete this function, you will have to create `else if` branches for each user command we require, the specifications of which are listed below:

1. `ls`: lists all valid documents
2. `import [external_file] [doc_name]`: imports a file
3. `export [doc_name] [external_file]`: exports to a file
4. `print [doc_name]`: prints the contents of a document
5. `append [doc_name] [string]`: appends a string to a document
6. `chmod [doc_name] [mode_changes]`: changes the permissions of a document
7. `rm [doc_name]`: removes a document with the specified name from the `doc_system`

(Note that we have included the code to parse the user input for you – all you need to do to access the command arguments is use the `arguments` array, and pass them into functions you defined earlier. Also, you will have to make use of the C functions `fopen` and `fclose` for dealing with file input)

4 Building & Testing

All of the commands below should be executed in your Docker container terminal, in the same directory as your project files.

4.1 Unit Tests

To run the autograder locally (without GDB):

```
// To clean your working directory
make clean

// Compile all the required files
make tests

// Run the tester object
./tests
```

This will run all the test cases and print out a percentage, along with details of the **failed test cases**.

Other available commands (after running make tests):

- To run a specific test case (to avoid all printing output/debug messages for all test cases):

  ```
  make run-case TEST=testCaseName
  ```

- To run a test case with gdb:

  ```
  make run-gdb TEST=testCaseName (or no testCase to run all in gdb)
  ```

4.2 Testing my_main

There are two test cases for the main function that exist in the `my_main_Tests` directory. Each consists of a subdirectory with:

- An input file (`caseX_in.txt`). This will consist of a list of commands that will be given to your main function, as if you were typing them one after the other on the command line.
- An expected console output (`caseX_console_expected_out.txt`), consisting of exactly what the program is expected to print out when given the input file. You must match this exactly.
- For case 2 specifically, an expected export file (`case2_export_expected_out.txt`). This should exactly match the export file produced during the test case.

The full program (executable from command line with a working shell) can be used like so:

```
// To clean your working directory
make clean

// Compile all the required files and build the executable
make project4
```

```
// Run the tester object
./project4
```

You can then type in commands on the shell to see their effect on the document system.

The autograder for this portion of the assignment is a shell script that will run the test cases against your executable and compare the output text against the expected output. You can run it with the following command. It will show you the differences between your output and the expected output.

```
make run-main-tests
```

5 Deliverables

Turn in the files `collaborators.txt`, `project4.c`, and `project4.h` to Gradescope by the due date.

Note: Please do not wait until the last minute to run/test your project, history has proved that last minute turn-ins will result in long queue times for grading on Gradescope. You have been warned.

6 Demos

This project will be demoed. Demos are designed to make sure that you understand the content of the project and related topics. They may include technical and/or conceptual questions.

- Sign up for a demo time slot via Canvas **before** the beginning of the first demo slot. This is the only way you can ensure you will have a slot.
- If you cannot attend any of the predetermined demo time slots, e-mail the Head TA **before** the beginning of the first demo slot.
- If you know you are going to miss your demo, you can cancel your slot on Canvas with no penalty. However, you are **not** guaranteed another time slot. You cannot cancel your demo within 24 hours or else it will be counted as a missed demo.
- Your overall project score will be `((autograder_score * 0.5) + (demo_score * 0.5))`, meaning if you received a 90% on your autograder, but a 30% on the demo you would receive an overall score of 60%. **If you miss your demo you will not receive any of these points and the maximum you can receive on the project is 50%.**
- You will be able to makeup one of your demos at the end of the semester for half credit.

7 Rules and Regulations

7.1 General Rules

1. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

2. Please read the assignment in its entirety before asking questions.

3. Please start assignments early, and ask for help early. Do not email us a few hours before the assignment is due with questions.

4. If you find any problems with the assignment, it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

7.2 Submission Conventions

1. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

2. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

7.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. You are additionally responsible for ensuring that the collaborators list you have provided in your submission is accurate.

4. Projects turned in late receive partial credit within the first 48 hours. We will take off 30% of the points for a project submitted between 0 and 24 hours late, and we will take off 50% of the points for a project submitted between 24 and 48 hours late. We will not accept projects turned in over 48 hours late. This late policy is also in the syllabus. The late policy is applied to both portions of the project; turning the code in late means that you will also lose points on the demo.

5. You alone are responsible for submitting your project before the assignment is due; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until the deadline.

7.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class.

1. Students are expected to have read and agreed to the Georgia Tech Honor Code, see http://osi.gatech.edu/content/honor-code.
2. Suspected plagiarism will be reported to the Division of Student Life office. It will be prosecuted to the full extent of Institute policies.
3. A student must submit an assignment or project as his/her own work (this is what is expected of the students).
4. Using code from GitHub, via Googling, from Stack Overflow, etc., is plagiarism and is not permitted. Do not publish your assignments on public repositories (i.e., accessible to other students). This is also a punishable offense.
5. Although discussion among the students through piazza and other means are encouraged, the sharing of work is plagiarism. If you are not sure about it, please ask a TA or stop by the instructor's office during the office hours.
6. You must list any student with whom you have collaborated in your submission. Failure to list collaborators is a punishable offense.
7. TAs and Instructor determine whether the project is plagiarized. Trust us, it is really easy to determine this....

7.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you should not be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as Bluejeans, to help someone with debugging if you're not in the same room.

Any of your peers with whom you collaborate in the above fashion must be properly added to your **collaborators.txt** file. Collaborating with another student without listing that student as a collaborator is considered plagiarism.