Lab 2: UNIX File I/O – ARVIK

Please read this entire assignment. Take some notes. Think about it some. Take some more notes. Look for answers to your questions in this document.

Due: As shown in Canvas. Submit a single tar.gz file.

Do not place ANY directories in your submitted tar file. I will not change into any sub-directories to hunt down your source files. When you create your `tar.gz` file to submit, do it within the directory where you created the source files, **NOT from a higher level directory. If I cannot find your source files in the same directory where I extract your submitted tar file, I will simply give you a zero on the assignment**. Submit a single `tar.gz` file to Canvas for this assignment. If you need help creating a tar.gz file, read the material about `Makefile` or look at the example `Makefile` in my `src` directory.

In this assignment, you will be working with UNIX file I/O system calls and library functions. **I urge you to not delay beginning it**.

1. **This homework is about *reading and writing files*** using the C Programming Language. **That's it, just read files and write files**. You will need to `fstat()` (or `stat()`, I used `fstat()`) files, check file permissions, check file time stamps, and perform seek functions through a file (using `lseek()`). My solution C file has 762 lightly commented lines. Some of those are preprocessor directives. You will probably have fewer lines than I have in my file.

   1.1. Write a **C program** on `babbage` called `arvik-md4`. This program will illustrate the use of file I/O on UNIX by managing a **file archive library**.

   1.2. For this assignment, the following is the generic command line syntax your program must support:

   ```
   arvik-md4 <options> [-f archive-file] [member [...]]
   ```

      1.2.1. The `archive-file` is the name of the archive file to be used, and *options* comes from the following table. Archive members are files you designate on the command line to copy into the archive file. Command line options can be given in any order and may be duplicated. If the `-f` command line option is not used on the command line, you must default to using `stdin`/`stdout` (based on the action you are performing).

Option	Action
-x	**Extract** members from `arvik` file. • extract all archive members to files. • When extracting files from the archive, the file timestamp (the `mtime` data member from `stat()`) must be restored as stored in the archive. • If the `-f` command line option is not used, default to reading the archive file from `stdin`. • When extracting files from the archive, the file permissions (mode) must be restored as stored in the archive. • You cannot restore file ownership as stored in the archive, so don't spend time on it. Extracted files will always be owned by you. • If the extracted file already exists, overwrite it. • If the `-v` option is also on the command line, print a line indicating each file member extracted from the archive.
-c	**Create** an `arvik` style archive file. • The names of the files to place in the archive are given on the command line. • If no file members are given on the command line, create an `arvik` file that only contains the `ARVIK_TAG` header line. • If the `-f` command line option is not used, default to writing to `stdout`. • If the archive file does not exist, put the following permissions on it `"-rw-rw-r"`. • If the archive file already exists, overwrite it, but leave the existing permissions on it. • If the `-v` option is also on the command line, print a line indicating each file member added to the archive.
-t	**Table of contents**. • Your table of contents **must match exactly** the one produced by my implementation of `arvik`. • If the `-f` command line option is not used, default to reading the archive file from `stdin`. • Look at the output of my solution for the format of the output. • If the `-v` option is also on the command line, give the **LONG** table of contents. • If the `-v` option is NOT on the command line, give the **short** table of contents.
-f filename	Specify the **name of the `arvik` file** on which to operate. • If there is not a **`-f filename`** on the command line, you must use `stdin`/`stdout` depending on which action you are performing (reading or creating).
-h	**Show the help text and exit**. Your help text must match mine.
-v	Verbose processing. • Output additional information for `-t`, `-c`, and `-x`. • Except for the `-t` option, I don't care what diagnostics you send to `stderr`.
-V	**Validate** the md4 value for the **header** and the **data** for each archive member. If the `-f` command line option is not used, default to reading the archive file from `stdin`.

1.3. If an invalid command line option is passed to `arvik-md4`, it should exit with a value `INVALID_CMD_OPTION`.

1.4. The archive files you create with your `arvik-md4` must **exactly** match the archive files created by my implementation of `arvik-md4`.

My `arvik-md4` program lives in the same directory with the header include file (`~rchaney/Classes/cs333/Labs/Lab2`).

1.4.1. The archive files created with my implementation of `arvik-md4` command and those created with your `arvik-md4` must be **interoperable**.

1.5. The following `.h` file specifies the `arvik-md4` header line (the macro `ARVIK_TAG`) and the header used for each file in the archive (the `typedef arvik_header_t` and `arvik_footer_t`).

`~rchaney/Classes/cs333/Labs/Lab2/arvik.h`

1.5.1. **Do not copy or in any way modify the `arvik.h` include file**. You can create a symbolic link to `arvik.h` or use the `-I` command line option to help `gcc` to find it. **When grading, I will delete the `arvik.h` file in your directory and create a new symbolic link before compiling your code**.

1.6. Notes (lots of them and they are all important):

1.6.1.1. For the `-c` command `arvik-md4` should create an archive file, if it doesn't exist, using permissions `"-rw-rw-r--"` (`0664`). Since your default `umask` will probably strip off some of those permissions, you'll either need to reset your `umask` or `fchmod()` the file afterwards.

1.6.1.2. When creating an `arvik-md4` file, the first thing you must do is write the `ARVIK_TAG` macro into the archive file.

1.6.1.3. When reading an archive file, the first thing you must do is validate that the file begins with the exact `ARVIK_TAG` macro. A bad file format causes an error statement to be printed (to `stderr`) and your program to exit. **Don't try and do anything with a file that has a bad format, just issue an error statement and exit, with an exit value of `BAD_TAG`.** One and done.

1.6.1.4. For the other commands `arvik-md4` reports an error if the archive specified on the command line does not exist or is in the wrong format. Notice the string that is used at the beginning of the `arvik-md4` archive files. The `ARVIK_TAG` macro.

1.6.2. You will have to use the system call `stat()` to properly deal with extracting timestamps.

1.6.2.1. The archive file stores the time associated time value from the `inode` with the member files: modify time (`st_mtim`).

1.6.3. You are **required** to handle multiple file names as members on the command line when creating an archive file.

1.6.4. I have created some sample files that you can use for testing your application.

1.6.4.1. Some are just plain text, so you can actually just `cat` the archive file after you've created it to see how it looks. **Try it, it can make the mystery of this assignment completely go away**.

1.6.4.2. Some text sample files include: `[0-6]-s.txt` . They are in the `Lab2` directory. There are more example files.

1.6.4.3. I have also created several sample archive files. Look at them, `cat` them to the screen. It will help.

1.6.4.4. The sample files can be found on the Unix server in:

`~rchaney/Classes/cs333/Labs/Lab2/`

1.6.5. Because you are *potentially* working with binary files, you **must** use the `open()`, `close()`, `read()`, and `write()` system calls. **Do not use** `fopen()`, `fscanf()`, `fprintf()` , or `fgets()` for the archive file I/O. **You can use `printf()`/`fprintf()` for terminal output**, such as with the verbose switch or the table of contents of the archive file.

1.6.6. When working on the `-t` option, think about how you can visualize whitespace in your output when combined with -v.

1.6.6.1. When the assignment says "**must match exactly**" that actually means must match exactly, including spaces and tabs. I use tabs for indentation and spaces for alignment.

1.7. You must have a single `Makefile` for this lab. Your `Makefile` must contain (at least) the following targets:

Target	Action
`all`	Builds all dependent C code modules for your application in the directory. This should be the first target in your `Makefile`.
`arvik-md4`	Builds all dependent C code modules for the `arvik-md4` application in the directory. The dependency of `arvik-md4` should be `arvik-md4.o`.
`arvik-md4.o`	Compiles the `arvik-md4.c` module for the arvik-md4 application in the directory, based off of any changed

	dependent modules. The dependency for `arvik-md4.o` should be `arvik-md4.c`.
`clean`	Deletes all executable programs, object files (files ending in .o produced by `gcc`), and any editor chaff (# files from vi and ~ files from **emacs**). Make sure you use this before you bundle all your files together for submission.

1.7.1. When I build your assignment, I should be able to just type

```
make clean
make clean all
```

to have it completely clean the directory and build `arvik-md4`.

1.7.2. If you are struggling with how to create a `Makefile`, I strongly recommend you look over the class material about `Makefile`s and then review the examples I have in the `src` tree for the class. I'm sure you have ample time in your schedule to spend 5-15 hours **wading through the effluent** on Stack Overflow or spend 20 minutes looking through the class material. You can also ask questions. I can help you get a good `Makefile` going in about 10 minutes. **And, your TA is amazing!**

1.7.3. I also recommend that you **use some form of revision control on your source files**. Not only does this reduce the possibility of catastrophic file loss, but it is a LOT better than making `.BAK1`, `.BAK17`, `.BAK4c` copies of your code.

1.7.3.1. Putting this into your `Makefile`, as described in the notes about `make` would make your life better.

1.7.4. You must compile your program using the following flags for `gcc`

1.7.4.1. Putting these flags into the `Makefile` with the `CFLAGS` variable will make your life better.

1.7.4.2. When compiled with the flags, the `gcc` compiler should emit no errors or warnings.

1.7.4.3. Any warning from the compiler is an automatic 20% deduction.

```
-Wall -Wextra -Wshadow -Wunreachable-code -Wredundant-decls
-Wmissing-declarations -Wold-style-definition
-Wmissing-prototypes -Wdeclaration-after-statement
-Wno-return-local-addr -Wunsafe-loop-optimizations
-Wuninitialized -Werror
```

1.8. **Remember**, the archive files you create with your `arvik-md4` must be interoperable with those created with my `arvik-md4`. You can easily test one from the other.

1.9. Your code must pass through `valgrind` without any memory leaks (lost memory) or any messages from `valgrind` about unsafe memory accesses.

1.9.1. **Having memory leaks or other warnings messages from `valgrind` will be an automatic 20% deduction**.

2. Since there are quite a few parts to completing this lab, I encourage you to start small.

2.1. Make a plan for how you will break down the assignment into smaller pieces. Don't just start hacking away.

2.2. Plan the work. Work the plan.

2.3. My *recommendation* on how to proceed with this assignment is:

2.3.1. Create the `Makefile` and keep it current as the project is developed.

2.3.1.1. Have a target in the `Makefile` to check your code into revision control. You know it's easy, you know how to do it, so **just do it**.

2.3.2. Spend a little time looking at the `arvik.h` file. You are going to need to understand the contents of these files very well. This is a good time to start.

```
typedef struct arvik_header_s {
    char arvik_name[ARVIK_NAME_LEN];
    char arvik_date[ARVIK_DATE_LEN];
    char arvik_uid[ARVIK_UID_LEN];
    char arvik_gid[ARVIK_GID_LEN];
    char arvik_mode[ARVIK_MODE_LEN];
    char arvik_size[ARVIK_SIZE_LEN];
    char arvik_term[ARVIK_TERM_LEN]; .
} arvik_header_t;

typedef struct arvik_footer_s {
    char md4sum_header[MD4_DIGEST_LENGTH * 2];
    char md4sum_data[MD4_DIGEST_LENGTH * 2];
    char arvik_term[ARVIK_TERM_LEN];
} arvik_footer_t;
```

2.3.3. Write your C code that chews up the command line, using `getopt()`, of course.

2.3.3.1. This should help you plan how to storm the castle and, at a minimum, put you over the top on the `-h` command line option.

2.3.3.2. No file processing should occur within the `getopt()` loop. The `getopt()` loop should configure your process for what to do next.

2.3.4. Create an archive file using my `arvik-md4` command.

2.3.4.1. Use all the small sample files provided, `?-s.txt` files.

2.3.4.2. **`cat` the archive file to the screen**.

2.3.4.3. This will take a lot of the mystery out of this assignment!!!

2.3.4.4. Create a couple more archive files like this and `cat` them to the screen to improve your comfort with the file format.

2.3.4.5. Look over the sample archive files along with the sample input files.

2.3.4.6. It's not scary. Really. 😊

2.3.5. On to your C code, process all command line options using `getopt()`. Just use empty stubs for command line options you've not yet completed. You'll fill in the stub code as your work progresses. Just get `getopt()` working for you.

2.3.6. Process the `-h` and `-v` options. They are very easy and it gives you a good start. You are filling in that stubbed code.

2.3.7. Get that `-f filename` command line option and argument going. It should be straight forward and is necessary to get the other portions working correctly. **Remember, if the `-f` command line option is not used, you must default to using `stdin/stdout`.**

2.3.8. When processing the data for an archive member, you'll need to start and update the md4 value for the footer. The md4 checksum we use to make sure the header and data for an archive member have not been corrupted someplace. If you need to see some sample code for calculating the md4 value, look at the `md4_file.c` file in the source tree (`~rchaney/Classes/cs333/src/md5`). **The md4 functions we use are from the `md` library, so you'll need to link with `-lmd` at end of you link line in your `Makefile`.**

2.3.9. The md5 value is stored in the footer as hex characters, which does not have a leaving `0x`.

2.3.10. Create code that performs the `-t` option on an archive file you created with the my `arvik-md4` program (or a some of the samples I placed in the `Lab2` directory). Once you have stepped through a couple archive files for the `-t` option (and `-tv`), you have a good idea how to read and manage the archive files. Check the provided slide set for some ideas of how to step through an archive file.

2.3.11. Use the `diff` or `md5sum` commands. The `diff` command works better for text files. The `md5sum` command gives you a MD5 digest for the file (great for binary files).

3. **Remember**, the archive files you create with your `arvik-md4` need to be interoperable with those created with my `arvik-md4`. You can easily test one from the other.
4. System/Library functions you may find interesting (in no particular order):
   4.1. `close()`
   4.2. `exit()`
   4.3. `fchmod()`
   4.4. `fprintf()`
   4.5. `fstat()`
   4.6. `futimens()`
   4.7. `getopt()`
   4.8. `localtime()`
   4.9. `lseek()`
   4.10. `memcmp()`
   4.11. `memcpy()`
   4.12. `memset()`
   4.13. `open()`
   4.14. `perror()`
   4.15. `read()`
   4.16. `sprintf()`
   4.17. `strchr()`
   4.18. `strlen()`
   4.19. `strncpy()`
   4.20. `strtol()`
   4.21. `umask()`
   4.22. `write()`
   4.23. `strftime()` I use the following format string with `strftime()`
   `"%b %e %R %Y"`
5. When you are ready to submit your code, bundle it all up in a single `gzip`-ed `tar.gz` file and submit to Canvas. Your `tar.gz` file should contain exactly 2 files: `arvik-md4.c` and `Makefile`.

6. Final note

The labs in this course are intended to give you basic skills. In later labs, we assume that you have mastered the skills introduced in earlier labs. **If you don't understand, ask questions.**