Project 3, CSCI 1730

Differences in Files using C
Due date/time are on the Breakout Labs and
Projects webpage

Learning Objectives

- Design and implement C programs that employ the UNIX file access primitives (open, close, read, write) or their C system library counterparts (fopen, fclose, fread, fwrite).
- Demonstrate knowledge of UNIX kernel and process data structures by sketching file descriptor table, file table, and inode table structures and the updates that correspond with the execution of your code.
- Use malloc (or calloc) and free for dynamic memory allocation and deallocation in C.
- Design and implement programs that use both static objects and dynamic memory management and demonstrate knowledge of state and behavior by constructing memory maps and predicting program output.

Problem / Exercise

**You are encouraged to work with one partner on this project,
but you and your partner must both be registered for the same lab section
(note: only one submission on nike is needed for a group of two students).
Also, both students must contribute equally to their project.**

The goal of this project is to give you system programming experience with file I/O and malloc in UNIX using the C programming language. In this project, you will implement a C program that reads in the contents of two input files. The program will then compare the two files byte by byte and write differences into two output files and output timings for the program's two main steps as indicted in the project's requirements.

Project Requirements

Your program must adhere to all requirements stated below.

1. Write a C program that compares two input text files and writes every byte in file one that is different from file two into a third file named `differencesFoundInFile1.txt` and every byte that is different in file two from file one into a fourth file named `differencesFoundInFile2.txt`. If `differencesFoundInFile1.txt` or `differencesFoundInFile2.txt` exists, then your program should overwrite the file(s) with new output (do not append to these files). The C source code must be implemented in a single file called `proj3.c` and it must compile with our provided Makefile to create an executable called `proj3.out`. You cannot alter the Makefile provided. Your program must only use the C programming language, and it CANNOT use any aspects of the C++ programming language that aren't part of the C language. The use of C++ for this programming project will result in a failing grade on this project.

2. Your program must complete the project requirement 1 in two different steps as indicated below.

Step 1: Compare input files one and two. This will be done by grabbing 1 byte at a time from the hard disk (buffering turned off on the input streams and the output stream) comparing them and then writing any byte from file one that is not equal to the corresponding byte in file two into the file called `differencesFoundInFile1.txt`. This entire step should be implemented as a function called step1, and step1 should be called by your program's main function.

Step 2: Compare input files one and two. You will read both of the files into **two dynamically allocated arrays**. The input arrays should be allocated to be the **exact size needed** since we don't want to waste RAM. You will then compare the two arrays and write any byte from file two that is not equal to the corresponding byte in file one into a third array (also dynamically allocated), and then copy that third array into a file called `differencesFoundInFile2.txt`. This entire step should be implemented as a function called step2, and step2 should be called by your program's main function.

3. For each of the two steps in requirement 2, you will time how long the entire step takes to complete inside your program, and then output the two times to the command line when your program is finished (use `gettimeofday()`). In your README file include a short, plausible theory on why the two times are different (if they are different). If they are not different, then explain why the two different steps results in the same times. Note: you will have to run larger input files to see a significant increase in the time the steps will take.

4. All arrays must be dynamically allocated and freed (with malloc/calloc and free). Programs that fail to do this will be penalized. Run valgrind to check for memory leaks. Your program should not contain any memory leaks.

5. Your program should be robust and include appropriate error checks. If there is an error reading a file, then your program should print (to standard output) "There was an error reading a file.", and if there is an error writing to a file, then your program should print (to standard output) "There was an error writing to a file". If there aren't enough command line arguments (or too many), then print the "Usage" message shown in the Examples section. After an error message is printed, then your program should terminate normally.

6. You may ONLY use the four following #include statements in your `proj3.c`. The use of any additional #include statements or source code copied/pasted (or included in your source code in any other way) from other libraries than those mentioned below will result in a failing grade on this project.

   (a) #include <stdio.h>

   (b) #include <sys/stat.h>

   (c) #include <sys/time.h>

   (d) #include <stdlib.h>

7. You may implement additional C functions and call upon these C functions that you implemented as needed to complete this project.

8. All functions should be commented properly.

9. Your final program's executable, `proj3.out`, must have I/O that matches the examples exactly except for the timing output, which will vary each time your program is executed. Also, your program must correctly create (or overwrite) the two difference files as aforementioned. Failure to follow the I/O provided in the Examples section may result in a failing grade for this project.

Examples

The I/O of your final program must match the examples except for the timings, which will vary each time your program is executed. The input files can be downloaded from the labs and projects webpage to your account using the `wget` Unix command (note: downloading these text files to another operating system and then uploading them to your Unix account may result additional characters being added to the files which may cause issues with your programs). The first line of each example contains one or more commands (separated by a ; for multiple commands), and your program should produce the correct output along with the two output files mentioned in the project's requirements. You must check that all characters and character counts matches these examples exactly (otherwise, you may fail this project).

```
./proj3.out
Usage: proj3.out <file1> <file2>

./proj3.out input1.txt
Usage: proj3.out <file1> <file2>

./proj3.out a b
There was an error reading a file.
```

```
./proj3.out input1.txt input2.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt;
Step 1 took 0.038000 milliseconds
Step 2 took 0.018000 milliseconds
cat!dog.

./proj3.out input1.txt input2.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt | wc -c;
Step 1 took 0.066000 milliseconds
Step 2 took 0.040000 milliseconds
8

./proj3.out input3.txt input4.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt;
Step 1 took 0.034000 milliseconds
Step 2 took 0.016000 milliseconds
aaaaa
b

./proj3.out input3.txt input4.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt | wc -c;
Step 1 took 0.056000 milliseconds
Step 2 took 0.017000 milliseconds
8

./proj3.out input5.txt input6.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt;
Step 1 took 0.027000 milliseconds
Step 2 took 0.039000 milliseconds
a
bbbbbb

./proj3.out input5.txt input6.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt | wc -c;
Step 1 took 0.026000 milliseconds
Step 2 took 0.016000 milliseconds
9

./proj3.out alice1.txt alice2.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt;
Step 1 took 113.053000 milliseconds
Step 2 took 0.724000 milliseconds
showingAliceBender*Zoey!

./proj3.out alice1.txt alice2.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt | wc -c;
Step 1 took 113.401000 milliseconds
Step 2 took 0.784000 milliseconds
24

./proj3.out alice1.txt alice1.txt; cat differencesFoundInFile1.txt differencesFoundInFile2.txt | wc -c;
Step 1 took 114.963000 milliseconds
Step 2 took 0.646000 milliseconds
0
```

1 Submission

Before the date/time stated on the Breakout Labs and Projects webpage, you need to submit your code on nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-proj3` if you are working alone. If you are working with a partner, then place all of your group work in a directory called `LastName1-FirstName1-LastName2-FirstName2-proj3`. These directories must include all of the following.

1. All source files required for this project (proj3.c)
2. A copy of Makefile for this project

3. A README file filled out correctly (fill out a copy as you did in a previous lab, but replace the brainstorm sections with your short theory on why the timings differ). If you work with a partner, then put your first & last name and your partner's first & last name in the README file.

To submit from within the parent directory, execute the one of the following commands:

```
$ submit LastName-FirstName-proj3 cs1730
$ submit LastName1-FirstName1-LastName2-FirstName2-proj3 cs1730
```

If the submission is successful, then you'll see that a file that starts with `rec` (your receipt file) was created in the directory that you submitted. It is also a good idea to email a copy to yourself (to do this, modify the command found in Lab 01 for this lab).

2 Grading (50 points)

If your program does not compile on nike or a cf node with an unmodified copy of our Makefile, then you'll receive a grade of 0 on this assignment. Otherwise, your program will will be graded using the criteria below.

Makefile (working as aforementioned)	1 points
README file (filled out correctly with timing difference theory)	5 points
All functions/methods are commented properly	2 points
Program uses malloc (or calloc) and has no memory leaks	2 points
Program runs correctly on various test cases on nike or a cf node	40 points
Late Penalty	-10 points for each 24 hour period late
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on nike or a cf node with any given set of inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section exception for the timings, which will vary each time your program is executed. You may not assume inputs will be valid.