Program 2 - Unix Tools and Bits

Due Date: 5:00 p.m., September 23

All programs will be tested on the machines in the Q22 lab. It is required that you complete this program on the computers in lab (or via ssh). The binary provided was compiled on the lab machines and is unlikely to work on your machine.

Part 1 - Using Unix Tools

In Part 1 of the program we will be using gdb to defuse a bomb.

- ### Part A
  - The nefarious Dr. Evil has planted a “binary bomb” on our machines. A binary bomb is a program that consists of a sequence of phases. Each phase expects you to type a particular string on the standard input (stdin). If you type the correct string, then the phase is defused and the bomb proceeds to the next phase. Otherwise, the bomb explodes by printing "BOOM!!!" and then terminating. The bomb is defused when every phase has been defused. Your mission, which you have no choice but to accept, is to defuse your bomb before the due date. Good luck.
  - Below I have given you two source code files and an object file. You must write a makefile that will link them into a binary executable.
    - Download source files
      - bomb.c
      - defuseMe.c
      - hidden.o
    - In your makefile, create a target called defuseTheBomb the produces a binary executable called defuseTheBomb.
    - **DO NOT** point the ‘all’ target to the defuseTheBomb target.
  - Once you start execution of the program, you have ***300 seconds*** to defuse the bomb. Using GDB, step through the code, inspecting each passcode as you go.
    - The passcodes are random and will change each time you execute, so you have to learn how to use GDB.
    - Each phase’s passcode is the same length as the phase, so phase 1 only has a single letter passcode
  - Once you complete all 6 phases, and file called bomb_defused.txt will be generated. **Make sure you include this file, *unaltered*, with your submission.**

- ## Part B
  - Write a bash script called ‘trash’ that takes a single argument, which should be the name of an existing file in the current directory.
    - The script should move the given file, if it exists, to a directory named TRASH that is located within your home directory (~/TRASH).
    - If the TRASH directory does not exist, the script should create it. If the given file does not exist, an appropriate error message should be printed.
    - The script should also check all files currently in the TRASH folder to see if they have been there for more than 1 hour. If they have, they should be deleted.
      - Hint: There are several "environment" variables that automatically exist within a bash script. These include the following. You may want to explore what values these hold by echoing them to stdout.
        - $HOME
        - $USER
        - $PWD
        - $PATH
    - You should be able to run your script as such:

      ```
      ./trash <filename>
      ```
  - Once your trash script is working well, modify it so that it accepts a list of files and moves all of them to the trash. The script should print an error message for each file that does not exist.
    - When testing your script, try invoking it with a command like the following:
      - ./trash *.o

Part 2 - Bits and Bytes

In Part 2 you are going to write a looping binary interpreter that create an array representation of a binary value.

- ## Part A - Binary Printer
  - Write a function that uses bit shifting to store the binary representation of an integer in an array passed as a parameter. You will be given the size of the array and should make sure you fill out all elements in the array. To generate the array of values, you will need to use a bit mask and bitwise right shift.
    - 8 would store the following value in your array
      - 0000 0000 0000 0000 0000 0000 0000 1000
    - The provided driver code (main) which calls your binaryArray function with each of the below values and an array buffer to print the binary representation (from right to left) for each one:

            - 2
            - 255
            - -1
            - INT_MAX
            - INT_MIN
                - Using the included library <limits.h> at the top of the main source code file so you can use the global constant INT_MAX and INT_MIN

    - You can use the following website to check your results: http://www.binaryhexconverter.com/binary-to-decimal-converter

- ## Part B - Extending your Makefile and Checking your Code
    - Using the makefile you created for Part 1, add the following:
        - An 'all' target that only compiles your Binary Printer program to a binary executable called project2
        - A 'checkmem' target that recompiles your binary printer code (if necessary), then runs your program2 with valgrind
            - You must not have any errors or memory leaks. If valgrind reports errors or memory leaks, fix them and recompile.
                - You do not need to worry about suppressed errors
        - A 'clean' target the deletes all your object files (except the hidden.o) and your binary executables

Part 3 - Submission

You must use any driver code as I have given you and only make alterations where stated. DO NOT change it to take user input, different output, etc.

- Required code organization:
    - program2.c //Driver Code
    - bomb.c
    - defuseMe.c
    - hidden.o
    - makefile
    - bomb_defused.txt
    - trash.sh
    - Any additional files should be included in your project2.zip submission
- While inside your project 2 folder, create a zip archive with the following command
    - zip -r project2 *
        - This creates an archive of all file and folders in the current directory called project2.zip

        - **Do not zip the folder itself, only the files required for the project**
- Upload the archive to Blackboard under Project 2.

Grading Guidelines

Total: 20 points

- **Part 1:**
    - Part A
        - Successfully defuses the bomb (4 points)
    - Part B
        - Bash Script takes command line arguments (1 point)
        - Moves file to trash folder as described (1 point)
        - deletes files older than 1 hour (2 points)
        - accepts a list of files (2 points)
        - prints error message for missing files (2 point)
- **Part 2:**
    - Part A:
        - Passes Binary Tests (5 points total / 1 point each)
    - Part B:
        - An ‘all’ target as described (1 point)
        - A ‘clean’ target as described (1 point)
        - A ‘checkmem’ target as described (1 point)
- **Style Guidelines**
    - Your submission will be rejected if it does not…
        - Follow requested program structure and submission format
        - Follow formatting guidelines