Program 3 - Unix Tools

CS 580U - Fall 2017

Due Date: 5:00 p.m., October 12, 2017

Any changes/updates to the project description will be in red - ***Last Updated 9/18 8:30 a.m.***

All programs will be tested on the machines in the Q22 lab. If your code does not run on the system in this lab, it is considered non-functioning EVEN IF IT RUNS ON YOUR PERSONAL COMPUTER. You can write your code anywhere, but always check that your code runs on the lab machines before submitting.

Driver Code and Test Input Files

- **Provided Files**
  - program3B.c //Driver Code for Part 2
  - hidden.o //object file for part 1

Grading Rubric

TOTAL: 20 points

- **Part 1: (4 points)**
  - contains bomb_defused.txt with unique identifying key showing the bomb was defused (2 points)
  - Uses separate compilation, compiling defuseMe.o and bomb.o to object files, then links all 3 into an executable (2 points)
- **Part 2: (10 points)**
  - Passes Valgrind Tests without memory leak or error (10 points)
- **Part 3: (6 points)**
  - Makefile has all target that runs the defuseTheBomb, checkmem, and clean targets (1 point)
  - Makefile has target that compiles the defuseTheBomb executable (1 point)
  - Makefile has target for checkmem that compiles course to an executable and launches valgrind (2 points)
  - Makefile has target clean that uses the trash.sh script to clean all non-essential files (object files other than hidden.o and executables) (2 points)
- **Submission Guidelines**
  - Does not follow requested program structure and submission format (-5 points)

Guidelines

This is an individual assignment. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically *to* solve the problem you have been given, and you may not have anyone else help you write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

By submitting this assignment, you agree that you have followed the above guidelines regarding collaboration and research.

Part 1: GDB

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

- 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 ***200 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.**

  NOTE: Part A includes an object file, hidden.o, that was compiled on the machine in the lab. You must complete part A on the Q22 lab machines. The remaining portions of the lab can be completed on any machine with the valgrind and make utilities.

Part 2: Valgrind

- I have provided you with code for Part B above called that contains 3 functions riddled with memory errors, including a segmentation fault.
- Run the provided code with Valgrind to inspect the memory errors.
- Fix all memory errors so that Valgrind reports no memory leaks and errors.
    - You should see the following once you have fixed all of the errors:
        - All heap blocks were freed -- no leaks are possible

          ...

          ERROR SUMMARY: 0 errors from 0 contexts

- *CONSTRAINT: You may alter the code in any line in the 3 functions, but you can not simply delete lines of code.*

Part 3: Bash Script

- 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 the same directory as the script (./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.sh <filename>
    - *This means you will need to use the 'chmod' command to mark it as executable*
- 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.sh *.o

Part 4: Makefile

- Create a makefile with the following targets
    - all
        - all should have all 3 targets below as dependencies to allow us to run your code
    - defuseTheBomb
        - Should compile the defuseMe.c and bomb.c to separate object files, then link them with the provided hidden.o to create an executable. This means you will need at *minimum 3 targets* for this part of the program.
    - checkmem
        - Should compile your corrected program3B.c for part 2 and run with Valgrind

    - clean
        - The clean target uses your bash script from part 3, trash.sh, to remove any executable and object files you created when testing.
        - Be careful you do not accidentally delete the provided object file hidden.o

Part 5: Submission

- While inside your program 3 folder, create a zip archive with the following command
    - zip -r <yourid>_program3 <files to include>
        - This creates an archive of all file and folders in the current directory called <yourid>_program3.zip
        - **Do not zip the folder itself, only the files required for the lab**
        - **Do not include your object files or executables**
- Upload the archive to Blackboard under Program 3.