Operating Systems. Home Work #1.

Due to November 15, 2018, 23:59

In this exercise we practice some basic Bash scripting and interaction between a Bash script and a C program. You will need to implement a simple C program (Part I) and a script that runs it (Part II). The goal of the C assignment is not to solve an algorithmic problem but to write safe and high-quality C code—code without memory-related bugs, with error checking, that frees allocated resources, etc.

Cheating Policy

Part of the course requirements is to program assignments by yourself. You can discuss concepts and ideas with others, but looking at other people's code, looking at code from previous semesters, sharing your code with others, coding together, etc. are all not allowed. Students caught violating the requirement to program individually will receive a "250" grade in the course and will have to repeat it. *Even if you're under pressure, ask for help or even don't submit, rather than cheat and risk having to repeat the course.*

Late Submission Policy

You have 5 grace days throughout the semester, you can use all of them, some of them or none of them for this exercise. A 10 point deduction will be applied for every late day past the grace days.

Submission Guidelines

1. Submit one ZIP file that contains two source files: `hw1_script.sh, hw1_subs.c.`
2. Name the ZIP file `hw1_012345678.zip`, where 012345678 is your ID number.
   For example, in BASH: `zip hw1_012345678.zip hw1_script.sh hw1_subs.c`.

Attention for Mac users: Mac pushes some hidden subfolders into zip archives. Remove them.

We check

1. Programs implement the specified behavior.
2. C program compiles without warning with "`gcc -O3 -Wall -std=c99`".
3. C code passes auto-checking code quality checks.
4. Error handling and resource management in the C code.
5. If a buffer is allocated dynamically then it should be released correctly.
6. The script shall be a Bash script
7. Every step in the script (Part II) is done as one command.

Part I

Write a C program named `hw1_subs.c`. The program gets two strings as command line arguments (assume that the 1st argument is not empty, but the 2nd argument can be the empty string "").

The program performs the following flow:

1. Check whether the two environment variables `HW1DIR` and `HW1TF` are defined and read them. If either is not defined, it's an error.
2. Combine input file path. The combined input file path starts with the value of `HW1DIR`, then "/", and finally, the value of `HW1TF`.
3. Open the input file for reading.
4. Iterate through the input file content, substituting every occurrence of the string given by the 1st command line argument with the string given by 2nd command line argument.
    - To read from the file, you may use only the `read()` system call.
5. Print the produced result to the standard output; do not change the input file.
    - You can use library functions like `fwrite()` to print the output.

String operations: You can use any library function you want for string searches, comparisons, and so on.

Using system calls: Read the manual of the system calls you use carefully. Your code should handle their possible return values and errors.

Error handling: If the flow completes successfully, the program should return 0. If the program encounters an error, it should return 1. If you want to print error messages, you may use a library function.

Resource management: Be sure to `free()` every `malloc()`ed memory and to `close()` any `open()`ed file, in every exit flow of the program.

Self checking: We recommend using the program `~oscourse/checkme.py` on the server nova to check the quality of your C program. This program takes as input either a directory with your submission or a ZIP file containing your submission.

If there's a problem in your code, the auto-checker will tell you what problem exists but not exactly where it is. It will be up to you to find and fix these problems.

Assumptions about Execution

1. You **may not** assume a bound on the lengths of the strings passed (either the arguments or the environment variables).
2. You **may not** assume a bound on the size of the file. However, you may assume that you can `malloc()` enough memory to hold the file. That is, your program can treat a failure of such a `malloc()` as an error and exit.

Part II

Write a bash script named `hw1_script.sh` that gets one command line argument and performs the flow described below. The command line argument is an *absolute* path to some text file (i.e. starts with '/', for example "`/some/path/INPUTDATA.TXT`"). Check the auxiliary document (Moodle, "Sample BASH Session") for an example how to access the command line argument. (Hint: "${1}")

1. Silently and recursively remove the subdirectory named `HW1DIR` (relative to the working directory). (Hint: the relevant flags are "`-rf`").
2. Create a subdirectory in the working directory, named `HW1DIR` ("replacing" the one you've just removed).
3. Set the permissions for the subdirectory: full access to the owner, read/execute to the group/others.
4. Change current directory to the new subdirectory that was just created.
5. Copy the text file specified by the command line argument to the current directory under the name defined by the `HW1TF` environment variable.
6. Set the permissions for the file: full access to the owner, read-only to the group/others.
7. Change current directory back to be the working directory.
8. Invoke the `hw1_subs` executable (see Part I) on the copy of the file you just created. You choose the arguments supplied to the `hw1_subs` executable.

Important:

- Every step in the above flow should be performed with ONE command.
- To simplify your life, in this part only, you can assume that environment variables `HW1DIR` and `HW1TF` are defined when your script runs.

Assumptions about Execution

1. The script and the executable are in one directory, referred to as the *working directory.*
2. The script will be run from the working directory.
3. The executable is named `hw1_subs`, and the script `hw1_script.sh`.
4. If the following 2 environment variables are defined, this is their contents:
   - `HW1DIR` – non-empty string, a name of a sub-directory. Example: "`my_hw1_folder`"
   - `HW1TF` – non-empty string, a name of a temporary file. Example "`some_temp.file.data`"

   The strings don't contain "/" symbols.
5. The correct number of command line arguments is supplied.