Operating Systems. Home Work #1.

Due to November 18, 2017, 23:55.

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).

Remark

If you put your work in your TAU home directory, set file and directory permissions to be accessible only by you.

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.

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.
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. Print the produced result to the standard output; do not change the input file.
   - Use only system calls to open the file and read it. You can use library functions like `fwrite()` to print it.
   - Don't read in a one-by-one-byte manner.
5. Close the file

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

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 an error messages, you may use a library function.

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 as `HW1DIR` defines from the working directory. (Hint: the relevant flags are "`-rf`").
2. Create a subdirectory in the working directory, named as `HW1DIR` defines. You can
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.

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

We check

1. Programs implement the specified behavior.
2. C code passes auto-checking code quality checks.
3. Error handling in the C code.
4. If a buffer is allocated dynamically then it should be released correctly.
5. The script shall be a Bash script
6. Every step in the script (Part II) is done as one command.