Operating Systems

Assignment-1

Dockerizing Linux Shell Scripts: Introduction to Docker Containerization and Script Execution

Submission Date 23rd February 2024

Instructions:

- *This is an individual Assignment.*
- ***All parties involved in any kind of plagiarism/cheating (even in a single line of code) will be given zero marks in all the assignments.***
- *Assignment deadline won't be extended*
- *Late submissions will be discarded so submit your assignment on-time*
- *You must follow below submission guidelines, otherwise your submission won't be accepted*
    - *create a new directory*
    - *change its name to the following format*
      YOURSECTION_ROLLNUMBER_NAME.
      E.g. A_22I-0012_Muhammad Ahamd
    - *put all your files (.c & .sh & .txt ōlonly) into this newly created directory*
    - *Compress the directory into a compressed .zip file*
    - *Submit it on google classroom*
- *Use good programming practices (well commented and indented code; meaningful variable names, readable code etc.).*
- *Your shell scripts must be compatible with docker containers*
- *Understanding the problem is also part of the assignment.*

You are tasked with practicing shell scripting and Docker containerization in preparation for your Operating Systems course. You will create different shell scripts to solve different tasks, Dockerize these scripts, and implement a menu-driven script to execute them inside a Docker container.

1. **Shell Script Development:**
    - Develop the four shell scripts (detailed requirements of the scripts are provided at the end)
    - Ensure each script should follow coding guidelines and best practices and solve its respective task accurately.
2. **Dockerization:**
    - Create a Dockerfile to define the environment and dependencies required for running the shell scripts.
    - Copy all the shell scripts into the Docker image's working directory.
    - Set the default command for the Docker container to start the Bash shell and call the **master** menu script to exec.
    - Build the Docker image locally using the Docker CLI.

3. **Menu-Driven (master script) Script Executor:**
    - Develop a shell script named **menu.sh** that presents a menu to the user, allowing them to choose one of the six tasks to execute.
    - Implement error handling to ensure valid input from the user.
    - When the user selects a task, execute the corresponding Dockerized shell script inside the Docker container.
    - Display the output of the selected task to the user.
    - Output of **main.sh** should be like this

```
Menu:
1. Get Processor Information
2. Create a Testing Directory
3. Safe Delete
4. MinEdit: A Minimalist CLI-Based C Program Editor
5. Exit
Enter your choice: 1
You selected Option 1
```

DockerHub:

- Create your account on DockerHub using the official university email account and the username must be your registration number **(**i.e., **22i-1234)** . If you already have created an account with a different username, deactivate that account and create a new account with your registration number.
- Create your repository with name **"Assignment_01_Section_X_SP2024_OS"**
- Submit the image on Docker
- Follow the naming convention, otherwise you will be marked zero

Submission:

- A zip folder containing all your files named as **YOURSECTION_ROLLNUMBER_NAME**

Requirements for Scripts

Script 1. Get Processor Information

/proc/cpuinfo is a read-only, plain text file that contains information about the CPUs (central processing units) on a computer. https://www.linfo.org/proc_cpuinfo.html

Your task is to write a shell script that extracts the following information from the **/proc/cpuinfo** file:

1. Total Number of Processors
2. Total Number of Physical Cores
3. Is Hyperthreading Enabled (Yes/No)
4. Total Number of Cores with Hyperthreading
5. Processor Model

Store the extracted information in a file called "**CPU_info.txt**"

Script 2. Create a Testing Directory

Create a directory "**TestData**" and create **N** files in the directory with random text content containing multiple sentences. The file name should be generated in the following pattern f1, f2, f3 …, fN for **N**. The value of **N** will be provided by the user.

Script 3. Safe Delete

The default **rm** command does not confirm before it deletes any regular files. Write a short script called **SafeRemove**, such that it will make a copy before deleting a single file (that is, we do not use wildcard expressions for this problem) by do the following:

- Take one and only one argument at the command line (hint: search for an expression representing the number of arguments in the shell scripts). Print out an error message if no argument or more than one argument are provided (hint: use echo).
- Create a directory "trash" in the current one if it is not already created. And set its permissions to make it writable.
- Copy the file indicated by the first argument to a "**MyTrash**" folder.
    - Remove this file in the current working directory.

Script 4. MinEdit: A Minimalist CLI-Based C Program Editor

Develop a minimalist Command Line Interface (CLI)-based Linux C Program Editor named "MinEdit". You must write the program in C language and then make the program known to the terminal so that it can be executed as a known command.

To register a C program with the terminal so that it can be executed like a known terminal command, you need to add the executable file to a directory that is included in your system's PATH environment variable. This allows the terminal to locate and execute the program without specifying its full path.

The program should offer essential functionalities for creating, editing, saving, compiling, and running C code files.

Create New File: Users execute MinEdit without any options to create a new C code file within the editor. They will be prompted to enter the desired filename for the new C code file.

Edit C Code File: Users execute MinEdit followed by the filename to open an existing C code file for editing. If the file doesn't exist, a new file with the specified name will be created. Users can write the code directly on the console, which will be stored in a file using the Save command described below. Other operations such as **Save, Close, Compile, and Run** can also be performed on the file.

Save File: During editing, users can save modifications made to the C code file by pressing escape and providing the **:s** command.

Close File: To close the opened file, users can press Escape key and provide the **:x** command. They will be prompted to confirm before closing.

Compile C Code: Users can compile the C code file currently opened in the editor by pressing Escape and providing the **:!c** command. Use GCC compiler for compiling the C program. If the compiler is not installed, install it and then compile it.

Run Compiled File: After compiling the C code, users can execute the compiled program by pressing Escape key and providing **:e** command.

Note: Ensure that MinEdit provides clear and concise error messages to assist users in case of any issues or errors encountered during operation.

_____Good Luck □_____