Homework #2

Karmanya Mendiratta (km6296), Aya Oshima(ao2368)

Part 4 - Summary Questions (30 Points)

1. **How do we implement the following list of commands successively? How do you ensure that commands are executed regardless of whether each previous command succeeds, and how do you implement them? Also, what is another method to warrant that a command executes only if the previous command succeeds?**
    a. **(cd /XXXdirectory)**
    b. **(find . -name 'a*' -exec rm {} \ )[removes all files starting with filename a in XXXdirectory]**
    c. **(ls -la) [lists files in the current directory "XXXdirectory" after removing]**

To execute commands successively, semicolons (;) can be utilized. This allows for commands to run sequentially irrespective of the success or failure of any individual command. For instance, executing commands in the shell in this manner:
cd /XXXdirectory; find . -name 'a' -exec rm {} \; ls -la*
means that even if the action of moving to the directory (cd /XXXdirectory) fails, the shell would still attempt to execute the find command, followed by the ls -la command.

Conversely, to ensure a command only runs upon the successful execution of the preceding command, the logical 'AND' operator (&&) is employed as follows:
cd /XXXdirectory && find . -name 'a' -type f -exec rm {} \; && ls -la*
Here, the find command will only execute if cd /XXXdirectory is successful. Similarly, ls -la will only run if both the cd and find commands successfully execute.

To implement commands successively, a mechanism similar to the subshell concept is beneficial. The processes of switching to a directory, deleting specific files, and then listing the contents can be conceptualized as isolated environments or "subshells". When a command initiates, it's essentially starting a subshell. Once this subshell concludes, the next command's subshell begins, each maintaining its distinct environment and operations. For the shell design, the runcmd function can be tailored to recognize and handle these sequential command structures. Here are the steps:

- Creating a structure to represent each specific command.
- During command parsing, detecting each sequential operation and forming a dedicated subshell for it.
- As each command or subshell is invoked, the main process waits for the child process to complete its execution before moving on to the next one.

- In the case of the “&&” operator, as each command or subshell is invoked, the main process waits for the child process to complete its execution **successfully** before moving on to the next one.

2. **How would you implement sub shells by implementing “(” and “)”. For example,**
    a. **echo "-$(sed 's/cat/lion/g' a.txt)" > hello.txt**
        i. **Purpose: First, replace all occurrences of cat with lion in file “a.txt” and echo into hello.txt file.**

A subshell is a separate instance of the shell that is created within an existing shell Process. It operates as a child process of the parent shell. When a subshell is created, it has its own environment, variables and set of commands to execute, which are different from the parent shell. A sub shell can be created in the shell by enclosing a group of commands inside parenthesis “(“ and “)”.
Thus, in the given example when the parenthesis are started, a subshell is created. The shell then waits for the subshell to complete the commands present in the parenthesis “**(sed 's/cat/lion/g' a.txt)**”. Once the command is successful and execution of the subshell is completed, its output is substituted into the “**-$(...)**” block and then the shell can make use of its output to complete the rest of the command.

To implement subshells in our implementation of the shell, we can extend the code by adding a case in the runcmd to handle parenthesis “(“ . Here’s how we can modify the “runcmd” function to accommodate sub shells:

- We would have to define a new structure for subshell commands which contains a pointer to the command to be executed within the subshell.
- When parsing the command, we would identify the “(“ case and construct a subshell structure for the content within parentheses.
- When executing a subshell command, we will fork and create a child process, and execute the command within the subshell in the child process. The parent process will wait for the child process i.e. the subshell to complete and then proceed with its execution.

3. **How would you implement running commands in the background by supporting “&” and “wait”?**

We can run commands in the background by making use of ampersand(&). To do the same, we have to append “&” at the end of the command to indicate to the shell that we want to run this command in the background. This tells the shell to start the command in a different process and continue with the next command without waiting for the background process to finish.
If we want to wait for all background processes to finish before proceeding, we make use

of the “wait” command.

To implement running commands in the background by supporting “& and “wait”, we would have to make some changes to our implementation of the shell. The summary of those changes would be as follows:

- We would have to parse the command and check if the command has a ‘&’ at the end to indicate background execution. If it does, we would set the flag for background execution to true.
- When a command is marked for background execution, we will fork the process to create a child process and run it independently.
- A list of background processes would also have to be maintained and the process ID for the child process created above would be added to the list of background process.
- We would also implement the wait command to wait for background processes to finish. The wait command should either wait for a specific background process to finish or wait for all background processes to finish if no process ID is provided.
- Once the background processes have been finished, the process IDs for finished background processes would be removed from the background process list.