Alice Jiang, Sonali Singh
OS_HW2

Part 4 Summary Questions

Q1

a. **How do we implement the following list of commands successively?**

We can use the AND operator or use semicolons to chain them up. Under the hood, we could implement AND and ';' by having the shell call fork on each sub-command separated by '&&' or ';' We may also require adding functionalities in our parse so that it can recognize which command ends with '&&' or ';'.

cmd-1 && cmd-2 && cmd-3 //will execute the next command only if the previous succeeds
cmd-1 ; cmd-2 ; cmd-3 //will execute the next command regardless of whether previous succeeds

b. **How do you ensure that commands are executed regardless of whether each previous command succeeds, and how do you implement them?**
In this case, use semicolons to chain them up.

cmd-1 ; cmd-2 ; cmd-3
cmd-2 and cmd-3 will execute even if cmd-1 fails.

We can implement it by letting the shell call fork on each sub-command, waiting for the child to finish, and moving on to the next sub-command whether the previous command succeeds or not.

c. **What is another method to warrant that a command executes only if the previous command succeeds?**

I can think of two ways:

(1) Use AND operator:

cmd-1 && cmd-2 && cmd-3

Based on the logic of the AND operator, we know that the next command will be executed only if its previous command succeeds.

(2) We could output the exit status/exit code of the previous command to a variable in the corresponding structure. Every time we execute a command, we first check if the previous process's exit variable is 0 or 1. If 0, we execute the current command; otherwise, abort the program.

Alice Jiang, Sonali Singh
OS_HW2

Q2 How would you implement sub-shells by implementing “(” and “)”?

First, there will be an extra step when parsing the command. If ‘(‘ and ‘)’ is found in the command line, then it indicates that there is a subshell to be executed. The parse or lexer parses the subshell to be executed and stores it in some structure or variable.

Second, the shell (or parent) will spawn a child process to run the subshell program. fork() duplicates the current working environment of the shell so that the child can access whatever resources the shell currently has. While the child runs, the parent will wait for it to finish. A pipe is required to open up a communication channel between the parent and the child. The parent can write the to-be-executed command to the child, and the child and write the output back to the parent.

Lastly, the parent reads the output of the subshell from the read end of the pipe. It then closes the pipe and reaps the child. Now it will continue executing the rest of the program.

Q3 How would you implement running commands in the background by supporting “&” and “wait”?

To support running a program in the background, we could **fork** a new child every time we encounter ‘&.’ There will be an extra step when parsing the command. If ‘&’ is found in the command line, then it indicates that there is a background process to be executed.

For example, if the command to be executed is cmd1 & cmd2 & cmd3 & wait, the control flow will look like this:

```
Parent
        Child – execute cmd1
        Child – execute cmd2
        Child – execute cmd3
WAIT();
WAIT();
WAIT();
exit(0);
```

‘wait’ waits for all background programs to finish, while ‘wait $bg_pid’ wait for some background process to finish. If we want to wait for all background programs to finish, we need to call WAIT() on each child it has spawned before moving on to execute what goes after ‘wait’ in the command line. Wait can be implemented by storing the exit code in the corresponding structure of the process and then checking the exit code of some or all previously running processes, making sure they are successfully terminated before moving on to other jobs.

If we only need to wait for some specific process *p* to finish before moving on, we call waitpid() on *p* to specifically wait for it to finish. For example, if the script is

```
cmd1 &
p1=$!   #store the pid of the process that runs cmd1
wait $p1
cmd2 &
cmd3 &
wait
…
```

Then the shell will spawn the first child to handle cmd1 and then wait to receive its exit status. After the shell reaps the first child, it will then fork twice to let the first child handle cmd2 and the second child handle cmd3. Then it will wait for both of them to finish before it moves on to the next command.