Andres X Lopez

Project1

Simple Shell

1. Why is it necessary to implement a change directory 'cd' command in

the shell? Could it be implemented by an external program instead?

It is necessary to implement a change of directory or ‘cd’ command in the shell because then you would have to call cd outside of the shell which would be an inconvenience since you can do it from the shell.

This would only be done by the parent process because only the parent can change directory any other processes may not control since the would likely create security issues, or a user could even lose control to child process if a cd command is ran.

2. Explain how our sample shell implements the change directory command.

In the sample shell the change directory command is implemented by an else if its checks that the argument is cd and if it is then it also checks that exec\_argc is greater than 1, if those conditions are meant cd is executed by parent in shell. If any other condition it is to fail the cd and output “failed to chdir” and shows the directory with the arg (exec\_argv[1]);

3. What would happen if this program did not use the fork function, but just used execv directly? (Try it!)

Try temporarily changing the code 'pid\_from\_fork = fork();'

to 'pid\_from\_fork = 0;'

By not using the fork function in the program it doesn’t allow any output with the feedback, and also gets kicked out of the shell every time. This causes to only run an instance then exit, therefore not staying in the shell. The execv command replaces the process then exits.

4. Explain what the return value of fork() means and how this program uses it.

The fork() is used to return values that if 0 will create a child process, if a negative 1 is returned a child process would not be created because of kernel running out of memory or process descriptors. If fork() returns a 1 then the pid is outputted.

5. What would happen if fork() were called prior to chdir(), and chdir() invoked within the forked child process? (Try it!)

Try temporarily changing the code for 'cd' to use fork:

if (fork() == 0) {

if (chdir(exec\_argv[1]))

/\* Error: change directory failed \*/

fprintf(stderr, "cd: failed to chdir %s\n", exec\_argv[1]);

exit(EXIT\_SUCCESS);

}

When we change the code to use the fork instead of cd the process gets forked to a child process. This happens when we call cd and fork it then gives a failure with code 255. When I run fork() I get a child process but the change directory doesn’t happen since its inside a child process. It just exits out of the shell without success.

6. Can you run multiple versions of ./b.sh in the background? What happens to their output?

It is possible to run multiple versions of ./b.sh in the background but when I do it I get crazy output with output from the unique processes. I get output of random numbers and different pid from the different instances of the processes being run, and counted down.

7. Can you execute a second instance of our shell from within our shell program (use './shell')? Which shell receives your input?

Executing a second instance of our shell from within a shell is possible but what happens is when ./shell is ran the second time a child process then gets forked the process, so it becomes a childs process. This happens every time after its called after the initial call. So each shell becomes a new child process.

8. What happens if you type CTRL-C while the countdown script ./b.sh is running? What if ./b.sh is running in the background?

When running the ./b.sh countdown script and CTRL-C is pressed ./b.sh in the foreground immediately exits whether in the shell or just running the ./b.sh by itself. When in the background and ran as a command it will continue counting down but will hang at the end of the script, if ran in a shell it immediately exits.

9. Can a shell kill itself? Can a shell within a shell kill the parent shell?

./shell

./shell

/bin/kill -s KILL NNN (Where NNN is the the parent's PID.)

When a shell is ran inside as a shell and the kill command is sent to kill the parents pid it does kill the parent process by forking to a child that then sends the exit or kill command to the parent. So it does kill the parent process through a child process.

10. What happens to background processes when you exit from the shell? Do they continue to run? Can you see them with the 'ps' command?

./shell

./b.sh&

exit

ps

Apparently when the script is ran and an exit is ran it will keep running in the background until the script finishes then will terminate and not appear in the process list once complete. So it doesn’t kill the process since it still runs.

