Kunal Mukherjee
1/31/2018
CS 470
Dr. Deborah Hwang

Shell Program

Shell Program has one main function and three helper functions. A function called inputFromuser(), has the function that it just takes the input from the user and convert it to string and tokenized each arguments put in by the user. Then, the tokenized arguments are pushed into a queue. The second function called convert() takes in a queue by reference that has tokenized arguments in it and converts it into a C string, so that it can be used with execvp. The C string is dynamically allocated. So, that when the execvp command has been run and the process has been completed, the memory is again deallocated and returned to the system. The third function called printPast(), whose functionality is to print the past ten commands entered by the user, or a specific command as specified by the user when calling this function. It takes a vector by reference, which has the queues containing arguments, that were used by the program in previous runs and a int called curr, which points at the specific compartment in the vector that holds the queue of the past runs. The user can change the curr, to their desired instruction run from the past if they want to employ this functionality.

The program has a simple run, where there is a flag called should run and as long as it is true, the program runs, else it quits. While running, the program calls the inputFromuser(). Once it gets the input, it checks from some specific key words like history, !!, exit or cd. “cd” is important cause the function to execute thiscommand is chdir() and not execvp .If none of them are found, then it puts the command in history vector for future use and moves onto the next part of actual implementation. The program then checks if ‘&’ has been called, then it will turn on a flag called background, that will run a different loop, which creates the process sightly different cause it needs to run in the background. Once, the flag has been turned on, the ‘&’ is removed from the queue and the queue is passed to convert(), to generate the C string. Then, the fork logic is implemented with associated logic of what should be done if it is parent, child or grandchild. We have grand child cause we are using double fork technique.

The data structures that used are queue, to get the argument from the user and convert it to string. I sued queue, cause it was easy to implement and the program only cares about how it starts as well as how it ends, and we need to get access to the arguments in a definitive order FIFO order. I used vector for history because, as depending on the user’s decision the program may need to access any element of the vector. So, I needed a data structure that will give me the flexibility to get access to its elements without modifying it or the access function runs in constant time.

The history feature is depended on two things, the vector state and the current pointer. The vector is supposed to contain only ten elements. So, after the initial start, the vector is empty. Once, the user starts executing command the queues of arguments are add to the vector. Once, the vector gets ten elements and one more times the program ran, the vector will remove its first element and reset the pointer back to ten. So, the least recently used queue will be deleted. If the user enters a integer after “!”, then the a conditional statement is put in place will check if the element in the vector has been populated or not and see if it is bigger than the vector itself, then in both cases it will print out a message conveying the message that no such command exists. If ‘!!’ is entered, then the program looks at the curr, and runs the queue at that location of the vector. Then it enters the queue, in the next vector element and increments the curr element , so that the history is maintained. If ‘history’ command is being executed, then it runs through the vector and prints out all the queues in reverse order. The counter is decreased because we history should not be stored in the vector.

Test

- pwd- it will print the working directory
- !3 – should get an error saying that there is no command as you are trying to execute command that has not been entered yet
- emacs kunal – should open the emacs window with kunal as the name of the file;
- – Close kunal
- ps – will shows the processing running, emacs shouldn't be listed
- emacs kunal & – emacs should be opened in the background
- ps – emacs should be listed a process to be runnning
- – close emacs
- ps – the emacs process should not be listed any more
- !! – the ps should be again printed in the screen as you rerun the ps command
- history – it should show history from 7-1 with 7 being 'ps' and '1' being pwd
- cd 'name of directory you want to change' – this will show that the cd command works, for incorrect spelling or if no directory exists, it will give an error message
- pwd – you should be in the directory, if you spelled your directory name correctly and it exits
- date – this should give you the date of the current day
- history – now, you should see that pwd has been replaced by emacs kunal as you have enetered more than 10 commands in the past
- !2 – it should print the current directory you are in as the last second command is pwd
- !11 – this should give you an error message as youa re trying to access the 11th command and it will show that the history error conditional is working
- !5 – two emacs window should open in the background , this will show that the '&' functionality still works when called with !
- ps – should show emacs running as a process, now close emacs
- ps – should show that there is no emacs listed as the process has exited
- exit – the program should quit
- ps in the terminal – should show that no shell is running , to show that shelll has properly terminated its processes

Question-Answers

1> The creation of the grandchild or the double fork technique was a little difficult to understand. Also, which data structure to choose to help with the storage of the commands and then the history, was also a complicated processes. But, after the careful evaluation of the advantages and disadvantages of the data structure, I made the decision.

2> convert() function and the retrieval of queue from the vector based on user command was the easy as I did this manipulation in CS215 for editor project. The formulas were very similar.

3> I would add a clear history feature that would be easy to implement. Just clear out the vector and set the curr, to 0. This way we could delete the history that we didn't wanted someone else to find out. It is kind of browser history deletion.

4> The double fork method is itself very interesting. We learnt in class that if a process is terminated that its child processes would be adopted by the init. But, to use this design to make sure that there exist no zombie process for the duration of the new shelll execution was intriguing. Also, learning how our commands actually gets executed behind the covers of the terminal was very informative as well. Also, how cd has a different command that execvp, was also an interesting fact that I found out wile researching for this project.