Linux Shell Project

Objective:

-C program that copies the work of the Linux terminal.

Story:

-program that receives an input, analyze it, then depending on the analysis, the program print the result of the input. Actually the input is in the form of a Linux command, so this program should give the same results of the command as if it is wrote in the Linux terminal.

The input of the user is divided into the following types:

1. the command contains 1 and only 1 pipe ( ex: ls | wc )
2. the command either contains more than 1 pipe (ex : ps –ef |grep root | wc)..
3. the command begin with “cd” move between directories
4. the command contains ‘>’ or “>>” to write command results to file
5. User presses ctrl D (stop the shell)
6. Command ends with ampercand “&” (run command in background)
7. User enters “history” (view this history of commands written)
8. User enters “jobs” (view running process in background and stopped ones)
9. User requests parallel process (command contains “&&”)
10. User runs a script (C program or bash).
11. Pressing cntrl+C or cntrl+D
12. Other

The way we deal with each of the above types is as following:

1. When the user enter a command containing one pipe, the program divide it into 2 inner commands, then make a pipe between 2 children, first one execute the first inner command and write the output in the pipe, while the other one reads the input from the pipe and perform the second inner command based on the founded input.
2. When the command contains more than 1 pipe we wrote a function the split the command into sub commands by ( | ) and forked children on the number of sub commands where each time the parent creates a pipe and if it is the first command the child closes the stdout only and write into the pipe where the parent closes the write end of the pipe only and in a command between the child closes both the stdout and the stdin and reads from the read end of the pipe before and executes then writes in the new created pipe.. This process repeats until the last command where the child closes the stdin and reads from the pipe and executes then show the result of all the executed before.
3. The command that begins with cd means to change the current directory, so the predefined function “chdir” is responsible for the task.
4. When command contain ‘>’ character a child is forked and redirects the pipe inherited from the parent to the file requested and if not found it creates and the result is written in that file.
5. Ctrl D will exit the shell.
6. When user enter a background command the shell will execute the entered command and check if the entered command ended or not using a function which checks a file contains the result of ps command which is always updated every time the user enters a command, if command found it adds it to the linked list of jobs as a running process.
7. Every time user enters a command this command will be stored in a linked list. When user writes “history”, this list is printed.
8. When user enter jobs it prints the linked list of jobs.
9. When user requests parallel commands n children are born depending on number of commands, then each child will execute a command.
10. Running a script was solved by forking a child which executes it in a system call.
11. The parent and any child is ready for two signals SIGINT and SIGTSTP, if the parent is waiting and it receives any of these it do nothing but if a running child is doing something it dies if it received a SIGINT(cntr+c) it kills it self directly and if it received a SIGTSTP (cntrl+z) it returns to default behavior (SIGDFL) but the parent also receives the signal so it adds it to the jobs linked list as stopped process.
12. The other commands are executed using “execvp” function

Problems and solutions:

* The “fgets” function take the pressed “Enter key” with the string entered so a function was made in order to deal with all the string written before the “Enter key”. This function removes the ‘\n’ from the end of the string entered(removeEndl).
* Splitting the command using space “ “ as a delimeter in order to make an array that is given for execvp so a function is used (splitBySpace).
* Splitting the command using pipe “|” as a delemiter and counting the pipes in the command is solved using functions (splitByPipe).
* Function of many pipes had a problem in retrieving data where the parent wasn’t waiting for all the children to finish so a loop function on the number of forked children with a wait call solved the problem.
* Getting the current directory is solved by the function “getcwd”.
* A command that needs time to be executed leads to miss-order in printing between parent and child so the parent waits all the childs to exit or stop.
* Check if written command is in ps file. This problem is solved by using a function that go to the file and split each and check the last word if it is equal to the written command (checkPs).
* One problem that is not solved is when user press arrow ‘up or down’ key. This key should load the previous written commands. This can be made by using the function getch() “ncurses library” which can detect if user pressed any key on the keyborad without pressing enter key. However to use ncurses you should start a curses by using the function initsrc() which will open a new window. In this window, I can’t control the printing of execvp and system. The output is randomly disributed on the window.