Shell

T.H.Arjun 2019111012 CSD

How to run

1. Open the folder containing files in terminal
2. Run make
3. Run ./shell

The Shell will open up!

The shell takes the directory it is present in as the home of the shell as per requirements.

Built-In-Commands

Assignment 2 Commands

cd

It changes the directory. It takes ~ as the folder that contains the executable. It takes relative paths also. Usage : cd < path to change to >

ls

It works like the OG ls command. It takes flags -al -a -la -l -aaal -laa -lllaa etc. Usage: ls < option > < list of dir/files >. Options can come between arguments also. Works like the OG Terminal mostly.

pwd

It displays the present working directory

pinfo

It executes the pinfo command mentioned in Assignment pdf. Usage: pinfo < pid1 > < pid2 > ...

If no pinfo is given it shows the pinfo of shell

echo

It executes echo command. It does not implement "" '' $ etc. But handles whitespaces as required.

Assignment 3 Commands :

setenv

It sets value for an environment variable. Usage : setenv var value . Where var is the variable and value is the value to assign. If value is not given it is set to empty string. Error is also handled. Creating the environment variable if it doesn't already exist.

unsetenv var

Destroy the environment variable var,if it exists. It is an error for the unsetenvcommand to be invoked with zero or more than one command-line arguments.

jobs

This command prints a list of all currently running background processes spawned by the shell in order of their creation times, along with their job number, process ID and their state, which can either be running or stopped.

kjob

Takes the job number of a job and sends the signal corresponding to signal number to that process.

fg

Brings the running or stopped background job corresponding to job number to the foreground, and changes its state to running.

bg

Changes the state of a stopped background job to running (in the background). Does nothing if the job is already running.

overkill

This command kills all background processes at once.

quit

This command exits the shell. CTRL + D also exits the shell.

CTRL- Z and CTRL - C are handled as required by requirements.

Assignment 3 Bonuses

cd -

BONUS 1 ! : The - argument for the cd command. This switches to the previous working directory and prints its path.

Exit Codes :') and :'(

BONUS 2 ! : Exit codes indicate whether a process exited normally, or encountered an error. It is displayed for all commands. It follows the rules mentioned in assignment requirements.

Command Chaining with $, @ and short-circuiting.

BONUS 3 ! : The logical AND and OR operators can be used to chain commands, such that the exit code of the entire chain is the logical AND or OR of the individual exit codes. Uses the symbols @ to denote AND and $ to denote OR. These operators bind looser than |, >,<, >>, &, but tighter than ;. These operators short-circuit, executing their second operands only if needed to evaluate their truth value. The chain is evaluated from left to right, because @ and $ have the same precedence. These follow the rules mentioned in requirements.

Assignment 2 Bonuses

nightswatch

BONUS 1 ! : Usage : nightswatch < -n << seconds >> > < command >

It executes the command every n seconds mentioned by the option. If no option is given 2s is taken as default. The commands can be:

1. interrupt : It shows the CPU interrupts as said in Assignment PDF
2. newborn : It shows the latest pid that was made on the system

Press q to exit the loop.

history

BONUS 2 !: Usage : history < num >

It executes the history command similar to the one in OG Terminal.

By default it shows max 10. But if num is given it shows the last num commands if num commands are available or last available. If num is > 20, It defaults to 20. It takes " ls" "ls" as same. But takes "ls -al" and "ls -al" as different commands as in real terminal.

It also includes the current command that executed like in real terminal. The history is stored in a file called history.txt in the folder that the files exist.

Files Included

main.c

It has the main function and the loop that executes the shell. It also has the prompt printing. It also sets up all the variables and calls functions to initialise the shell. It also gets the command as input and calls the input handlers in inputhandlers.c to handle the input.

err.c

It has the functions related to exit codes. It has setters and getters for exit codes.

jobs.c

It is the home to the job family of commands, eg : kjobs, jobs, overkill, bg, fg etc. It has a linked list implementation for job queue.

pipe.c

It has the function that implements pipes. The shell supportes any number of pipes. pipe() is used for implementing this.

redirection.c

It is the home to the redirection logic and it's functions. My shell supports > , >> , < . (with their appropriate meanings). If multiple of them are given then it will take the

last one. It works as per requirements. The file uses dup(), dup2(), open(), close() etc for implementing this.

setenv.c

It is home to the setenv command. It uses the setenv () system call for implementing this.

unsetenv.c

It is home to the unsetenv command. It uses the unsetenv () system call for implementing this.

Pipes, Redirection are implemented according to assignment instructions

runCommand.c

It contains the implementation of forking and executing other commands that are not built-in using execvp(). It can run commands in Background and Foreground. It uses fork(),signal(), setpgid(), waitpid(), tcsetpgrp() etc to implement the same.

To run in Background

End the command with an &

Formats supported:

- vim&
- vim&vim&
- vim &
- vim & vim 1.c (vim 1.c executes in foreground as it did not end with &)
- vim &
- vim 1.c &vim& vim& etc

&& is not supported.

Built-in commands do not run in background. And this is not valid vim&ls;

But vim&; ls

or

~> vim&

~> ls

are valid.

To run in foreground

Just execute normally. eg: - vim 1.c

It also implements asynchronous signal handler to report the exit status of a process once it finishes. It reports if a process exited normally or not. It prints to stderr. It also kills zombie processes in case of background processes by doing the same. It also implements a linked list to store the command associated with a pid to use in status reporting.

inputhandlers.c

It has all the input handlers that handles the inputs. It tokenises, executes a command, and adds to history by calling appropriate functions. It has functions to translate shell path to system path and viceversa. It goes through the given command and starts it appropriate function call.

cd.c

It contains the implementation of cd command. It uses chdir() syscall to function. It uses the input handler function to get system path from shell path given.

echo.c

It contains the implementation of echo command.

global.c

It contains global variables that are required in different files and need to be shared.

char * usr, pwd[1024], opwd[1024], host [1024];

usr has username, pwd has present working directory, opwd has ~, and host has host name.

history.c

It has the implementation of bonus question of history. It uses a queue to implement it. The max size is 20 for the queue. It adds the command to the queue and takes commands from front if it becomes greater than 20 after adding. It also has functions to initialise the queue from previous session if history.txt exists, by reading from it. It also updates history.txt after every update. It uses "FILE" and fopen, fclose etc to do the same.

ls.c

It has the implementation of ls command. It checks for flags and prints appropriately. Uses stat, operdir, DIR, dirent, getpwuid, getgrgid etc to get the required info for ls. It uses a function to get the permissions in correct format.

nightswatch.c

It has the implementation of the bonus nightswatch command. It executes the nightswatch command appropriately. It uses /proc/interrupts, /proc/loadavg file structures for the required data.

pinfo.c

It contains the implementation of pinfo command. It uses /proc/pid/stat, /proc/pid/exe file structures to get appropriate data.