Introduction

A shell in unix-like systems is a program that enables users to run other programs in the system via terminal (command line interface).

This assignment is an attempt to replicate functioning of bash shell (a very common shell) using c language.

Logic

Steps

1. Display prompt and take command.
2. Extract and seperate commands and arguments (technically called parsing).
3. Search for executable of the commands.
4. execute commands.
5. Display pid and status.
6. Loopback until user presses ctrl+c or ctr+z.

Data Structure

A command structure contains 5 attributes;

1. Path: String
2. Command String (the actual command entered): String
3. Arguments (arguments entered with command eg '-a' in 'ls -a'): Array of strings
4. PID (a unique number assigned to each process by kernel): Integer
5. Status (a number returned by process after terminating to tell how it terminated): Integer

Since the shell supports piping (execute commands in sequence), a command entered by user may contain several commands. Hence an array of commands is used.

Here is th structure of Command.

```
typedef struct{
    char path[PATH_SIZE];
    char cmd_str[CMD_STR_SIZE];
    char **args;
    int pid;
    int stat;
} Command;

Here PATH_SIZE and CMD_STR_SIZE are simply numbers defined using #define
```

Finer Logic

The main function initialises *cmds* array of commands structure and a string *cmd_str* that will contain command entered by user.

take_cmd

Displays current working directory as prompt and accepts command from user into *cmd_str*. I inverted the prompt color using escape sequece \e[7m[to make it more visible.

It gets current working directory by using getcwd function in unistd.h header file.

parse_cmds

It takes the *cmd_str* (which contains command entered by user) and *cmds* array of command structure. It parses cmd_str and updates cmds accordingly.

It first replaces all '|' symbols with \0. In c end of a string is defined using \0 character. This fact will be used to seperate commands to be executed in sequence.

Then the string updates cmd_str and args attribute in cmds array.

search_path_cmds

It takes cmds and searches for paths for command/s entered by user. Returns -1 if path for some command is not found in PATH environment variable. (Search for environment variable if confused).

For each command in cmds array, the following procedure is repeated.

It gets PATH variable using getenv("PATH") function in unistd.h.

It loops through every path listed in PATH variable and searches for existence of executable of the command entered.

It updates path attribute of command structure if path is found.

execute_cmds

It takes the cmds array and executes each command as seperate process using fork function to create process and execv function to run the executable in that process.

It uses pipe and dup2 function to redirect output of one command as input of other.

It also updates the pid and status of the process in command structure.

print_cmds

It takes commands array and prints the pids and status of each command.