ASSIGNMENT - 2

SIMPLE SHELL: A UNIX SHELL IN C FROM SCRATCH

Link to Github: https://github.com/shrutya22487/Assignment-2

IMPLEMENTATION:
The simple shell program is a basic Unix shell that allows users to interactively execute commands, including both single commands and piped commands, and also run commands from script files.

1. User Input:

   The program provides a simple command-line interface where users can input commands. It displays a shell prompt, the current working directory, and waits for user input. Input validation is performed, which ensures that any empty or whitespace entries are being ignored.
Input() reads user input and returns as string. It flags whether valid input was received.
break_spaces() splits a command string into individual arguments based on spaces and newline characters.
Check_for_pipes() checks if a given command contains a pipe character ('|').

2. Command Execution:

   Users can enter single commands, which the shell will execute using execvp call. The shell can also handle piped commands, breaking them into individual commands and creating pipes to connect them.
executeCommand() executes a single command given an array of command arguments (argv) using fork and execvp. In the child process, it replaces its image with the specified command using execvp. In the parent process, waits for the child process to complete execution.
executePipe(), break_pipes_1(), and break_pipes_2() execute piped commands by creating a pipeline using pipe and fork. It splits a command line into individual commands based on the pipe character ('|') and into arrays of arguments, further returning a 2D array of argument strings suitable for executePipe.

3. ‘&’ implementation for background processes (bonus a):

Used to determine whether a command should run in the background.. check_and() function is used to check if the last character of a command string is an ampersand ('&'), removes the ampersand if found and returns true.

4. Text File Execution (bonus b):
   executeScript() function ensures that users can execute a text file by prefixing the script file's name with '@', indicating that the subsequent text is the name of a script file. The program opens and reads the script file, executing each line as a separate command. It can handle both single commands and piped commands within the script.

5. Signal Handling:
   The shell sets up a signal handler for Ctrl+C (SIGINT). This function ensures that the command history is being printed before the program's termination on pressing the Ctlr+C. This has been done using the signal_handler() and setup_signal_handler() functions.

6. Command History:
   The program also maintains a history of executed commands, including command text, process ID (PID), and start and end times.
   Users can view the commands history details on pressing Ctlr+C, the program will print all the details before terminating the program.
   get_time() function is used fir returning the time in microseconds.
   add_to_history() function maintains an array to store all the information, and display_history() function helps the users view the history.

LIMITATIONS:

1. Cannot handle complex command-line arguments or special characters correctly. For instance, the code may not handle quotes, escapes
2. might not work for commands that require specific paths or dependencies.
3. mechanism to manage or monitor background processes has not been implemented
4. scripting features like conditionals and loops are not supported.
5. doesn't provide options for custom signal handling or trapping.

6. Cannot handle cd , exit , export, unset, and env eg: cd is not an external program; it's a shell-specific command of the shell process itself. Since it's not an external executable, we cannot use execvp to execute it.
7. Max number of commands are 100 although we can change the max commands by simply changing the array size.

CONTRIBUTIONS:

Shrutya Chawla (2022487):

- Established the base code structure for the shell program.
- Implemented user input functionality for reading user input.
- Contributed to command execution with pipelines.
- Collaborated on developing the main function, coordinating program initialization and user input handling.
- Implementation of the '&' (background process) feature.

Swara Parekh(2022524):

- Implemented command execution for single commands.
- Designed signal handling mechanisms.
- Implemented the command history feature to display the commands, PID, and times.
- Developed text file execution functionality using the executeScript() function.
- Worked on the main function, integrating various program components, input validation, and function invocation based on user input.