# Simple Shell SiSH

Rafael
32215130
Global SW Convergence
Freedays : 2

**Introductions**

The provided code is a simple version of a custom shell program called "SiSH" (Simple Shell). It functions as a command-line interface enabling users to run commands like they would in a regular terminal or shell environment. The software takes input from the user, breaks it down into individual command arguments, and searches for and executes the command in directories defined in the system's PATH environment variable. Once the command is located and run successfully, the shell will wait for the child process to finish before asking the user for more input.

This shell can run basic commands and manage various command-line tools in the system like ls, pwd, or cat using the execv system call. The program continues to ask the user for input until the quit command is entered, which ends the program smoothly. Nevertheless, the shell does not have more advanced functions like command history, I/O redirection, or background process support, which makes it perfect for gaining a grasp on basic shell and process management principles in C.

**Features**

The code possesses several exceptional and distinctive characteristics that are noticeable, even with its straightforward nature.

Manual PATH Searching is different from regular shell implementations that utilize execvp or similar functions; this code parses the PATH environment variable manually and looks for the executable in every directory. This unique method of resolving commands provides a more interactive way to learn how a shell finds commands in the operating system.

After looking in the specified directories in PATH, the shell tries to run commands from the current working directory by adding ./ before the command. This backup feature enables users to execute programs from their current directory, a common feature in simple shells.

These characteristics offer a more in-depth understanding of the execution of shell commands, providing a chance to learn about the functioning of processes and environment variables in UNIX-like operating systems.

**Build Environment**

This code is written in C language using Visual Studio 2022 and used for trial in PuTTY with assam server.

**ChatGPT**

- **Where to start** : I didn't quite understand the assignment as I was absent in class, therefore I was pretty clueless even though I read the README file. With this situation I asked ChatGPT to help explain the assignment using game terms so that I can understand better. I had ChatGPT explain the usage of fork, getenv, etc. in game terms for better understanding.
- **Libraries** : As I was starting the project, I had issues with what various libraries that I needed to use, and asked ChatGPT for recommendations on what libraries are particularly required for a shell program such as this assignment. And therefore, I learned about these libraries.

  **#include <unistd.h>**
  **#include <sys/types.h>**
  **#include <sys/wait.h>**

- **Pid_t :** I learned that pid_t is for representing process identifiers (PIDs), and it can be used in managing processes in UNIX-like operating systems. It can be used in situations where processes need to be created, controlled, or monitored, such as in process creation with fork(), signalling with kill(), or waiting for a process to finish with wait(). Particularly in this code, pid_t is used to store the return value of the fork() function, allowing the program to determine whether it is executing in the parent or child process and to manage the execution flow accordingly.
- **Fflush :** After I finish the code, I asked ChatGPT if there are any extra functions that can be added to boost the program. Adding fflush(stdout); is suggested to ensure the prompt displays immediately after printing. Initially, I was confused about its significance, but after ChatGPT clarified the concept of buffered standard output, it made sense. I made the adjustment, and upon re-testing, the program exhibited a much more responsive feel. A minor adjustment had a big impact on user interaction.

**Personal Opinion**

I honestly didn’t really know what to do at the start of this assignment, as mentioned above, there are many things that I just knew from doing this assignment. I am also pretty concerned about the result of this code.