Project 1 Report

Emre Kılıç

The myshell implementation consists of different c files according to their tasks. Alias.c has functions about saving, retrieving and creating aliases. Builtins.c has functions about the shell builtins (alias, bello, -cd and pwd are implemented too-) and necessary functions to detect if the command is a builtin. Parse.c has functions regarding input parsing (replacing aliases, tokenizing, creating command struct, freeing the command allocations). Print.c has some functions about print statements (printing prompt and generic error messages, custom printf function for builtins to redirect). Process.c controls the process flow of the program, creates child processes, handles redirections, handles interrupts and finished background processes. Shell.c has functions for initializing and exiting the shell, checking the path and running the commands. Strlib.c has necessary string helper functions. Types.h is a header for defining the necessary data structures for the program.

The program flow is the following: Firstly, the shell is initialized (the path is retrieved, signals are handled, aliases are loaded, hello message is printed). Then, in the main loop, a prompt is printed and asks the user for an entry. For each entry, the first word is replaced with alias if possible, then parser tokenizes the line and creates the command struct which has fields command, args, redirection, background info. Then the command is sent to execution. Here first there is a check for the command to be a builtin, if yes, the corresponding builtin function is called, if not, the path is checked for the program name. When the path is found for the program, the full path and command is sent to the process controller. The process controller forks the new process. If there is a redirection, the stdout descriptor is changed with the file by dup2(2) function. If the redirection is '>>>' then a pipe is created and the child dumps its output to the pipe. The parent process waits for the child if there is no background and handles the reverse redirection from the pipe if needed. If there is a background, a process struct is created for the child and added to a linked list with all background processes and the program flow goes on. When a child process dies, since child processes send SIGCHLD signal to their parents and in the initialization part there is a handler defined for SIGCHLD, this handler iterates through the linked list and finds the terminated child, reaps it and handles the reverse redirection if needed. This mechanism is similar to the polling interrupt system where the master iterates over its descendants and finds the correct messenger to handle its concern. There is an enum defined in the program named SIGNAL that is meant to be the output status of the commands. So that when a command is finished with a successful manner, its command is stored in a global variable for bello's last executed command line.

The Bello command is executed like the following: The username is fetched with geteuid(2) and getpwuid(3) functions. The hostname is fetched by the gethostname(3) function. The last executed command is stored in a global variable. Tty name is fetched by the ttyname(3) function. The current shell is found by executing a ps program with args "ps -o args= -p <ppid>". The command execution is done by fork-exec manner and the output is written to a pipe. The home directory is fetched by getenv(3) with the argument "HOME". The date is fetched with time and localtime functions in <time.h> and some operations to convert the datetime to roman numerals. The number of processes can be found the size of the background processes linked list and plus 1 for myshell process.

The main concern in the project was redirection in a reverse order. It was solved by creating a pipe and using pipe as a target file to dump the output. After that it was very easy to reverse. The other difficulty was determining if background processes are terminated. It was solved by some research about signals and handlers. Another difficulty was redirecting the bello command, it was solved by defining a custom printf function which is called in bello. It prints the output to a file or stdout inside the printf function.