Assignment #1 Design

CMPS 111

Team

Forrest Kerslager, David Taylor, Connie Yu, Nick Noto, Kevin Yeap

Purpose

The purpose of assignment 1 is to create a basic shell in MINIX 3.1.8. The shell will be able to start foreground processes, background processes, and handle I/O redirection.

Available System Calls

exit(), fork(), execvp(), wait(), waitpid(), signal(), dup2(), open(), close()

Functions

void handler(int sig)

Handles the SIGCHLD signal sent from background processes by calling wait().

int main(int argc, char *argv[])

The main function for the shell. It runs the main while loop and contains all the code.

Testing

The shell will be tested by entering various commands: an empty line, exit, malformed exit, command with no arguments, command with one too many arguments, malformed command, nonexistent command, combinations of all the previous tests with input/output forwarding, combinations of all the previous tests running in the background.

Errors

All functions that return error values will be checked. In the event an error was generated the program will print a message and exit with an error.

Functionality

- A prompt is printed when the shell's main loop begins.
- Entering a blank command at the prompt will print the prompt again.

- Entering *exit* will break out of the loop and exit the shell.
- Entering a command with a process and arguments to run (eg. ls or ls -a -l or ls -al) will be executed using execvp(). Execvp will handle any errors such as a non-existent process, and the process itself will handle malformed arguments
- Appending '&' to a command will run it in the background. It is assumed nothing will exist in a command past the '&' character. This will bring you back to the prompt immediately and allow new commands to be executed. The background process will print its output upon finishing, and the user must press enter to continue. Exiting the shell will not stall to close background processes.
- Appending a command with '< file' will direct the command to use file as input instead of stdin. If a file is not provided an error message will be printed, and the command will be aborted as well.
- Appending a command with '> file' will direct the command to use file as output instead of stdout. If a file is not provided an error message will be printed, and the command will be aborted as well.
- A command can contain both input and output redirection in any order as long as it is placed at the end and before any '&'.
- A command can contain both output redirection and background processing.

Specific Issues

Zombie Processes

Wait must eventually be called on a process that was run in the background. If this does not happen the process will terminate and stay in the process table. If too many of these zombie processes are spawned they can fill up the entire process table.

At the beginning of the main function a signal handler will be set to pass SIGCHLD signals to a user defined handler function. This function will call wait() to clear the terminated child process from the process table. The purpose of this handler is to wait on zombie processes once we know they have terminated.

If a process is meant to run in the foreground the shell calls waitpid(pid, &status, 0) in the parent's code. This guarantees that the parent will only wait on the foreground child process spawned by fork().

If a process is meant to run in the background the parent's code will skip running waitpid and return immediately to the prompt. When the background process finishes, its SIGCHLD signal will be caught by the user defined handler.

Redirection

Dup2() is used to redirect a program's input and output from stdin or stdout to user specified files. The file specified is first opened by open() and then passed into dup2() along with the stream it is meant to replace (stdin or stdout). Before calling the process with execvp(), the user specified file is closed using close().

Main Function Flowchart