Gebze Technical University

Computer Engineering Faculty

Homework 2 Report

System Programming

Student: Burak Ersoy

No: 1901042703

What is the my program?

This is a C program that simulates a command-line interface. It allows the user to enter commands and arguments and execute them using the system's execvp function. It also supports input/output redirection and pipes.

The main function sets up signal handlers for interrupt, termination, and quit signals, initializes variables, and enters a loop to read user input and execute commands. It prompts the user with a $ symbol, reads input using fgets, removes the newline character from the input, and tokenizes the input by spaces using strtok.

For each token, the program checks if it is a redirection or pipe character and sets the input or output file descriptor accordingly. If it is a pipe character, the program executes the previous commands using execute_command and sets the input and output for the next command. If it is a regular token, the program adds it to the command array and increments the command count.

The program also checks for the exit command :q. If it is entered, the program creates a log file with a timestamp and writes the PID and command for each child process to the file. It then waits for all child processes to terminate and exits the program.

The execute_command function forks a child process and redirects the input and output file descriptors if necessary. It then executes the command using execvp. If execvp returns, an error has occurred, and the function prints an error message and exits. If the child process exits with a non-zero status, the parent process prints an error message.

Overall, this program provides a simple implementation of a command-line interface with basic features.

Function Explanation

1- void execute_command(char **command_tokens, int input_fd, int output_fd)

This function takes three parameters:

command_tokens: A pointer to an array of strings, where each string represents a token (i.e., a word or symbol) in the command to be executed. For example, if the command is "ls -l", command_tokens might be an array containing the strings "ls" and "-l".

input_fd: An integer representing the file descriptor for the input file. If the command to be executed reads input from a file (e.g., "cat < file.txt"), input_fd will be set to the file descriptor for that file. Otherwise, it will be set to STDIN_FILENO (i.e., 0).

output_fd: An integer representing the file descriptor for the output file. If the command to be executed writes output to a file (e.g., "ls > output.txt"), output_fd will be set to the file descriptor for that file. Otherwise, it will be set to STDOUT_FILENO (i.e., 1).

The execute_command function is responsible for actually executing the command specified by command_tokens, using the input and output file descriptors provided. It may use functions such as execvp() or dup2() to set up the appropriate file descriptors for the child process that will execute the command.

2-void signal_handler(int signal);

This function takes a single parameter, signal, which is an integer representing the signal that was received. Signals are a way for the operating system to notify a process of certain events, such as when a user hits Ctrl-C to send an interrupt signal.

The signal_handler function is responsible for handling the specified signal. Depending on the signal received, it may perform various actions such as terminating the current process, cleaning up resources, or setting a flag to

indicate that a certain condition has occurred. The exact behavior of the signal handler will depend on the specific needs of the program being written.

Test Cases