

Md Sarwar Hossain

Id: 161044121

This terminal emulator program handles command execution, input/output redirection, piping, and signal handling.

## Implementation Details:

**command:** A structure named Command defined to represent a command.. It consists of 3 fields. One is the command to execute and other two are the input output file descriptor. This way it's easily manageable as the input output descriptor might change as there is piping involved.

```
typedef struct {  
    char *command;  
    char *input_file;  
    char *output_file;  
} Command;
```

Fig: Command

**signal:** The signal\_handler() function is implemented to handle signals such as SIGINT and SIGTERM. When a signal is caught, the program terminates all child processes and displays information about the signal on the screen, then returns to the prompt to receive new commands. I used sigaction to handle these two signals.

```

void register_signals(){
    struct sigaction sa;
    sa.sa_handler = signal_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGINT, &sa, NULL) == -1) {
        perror("sigaction: SIGINT");
        exit(1);
    }

    if (sigaction(SIGTERM, &sa, NULL) == -1) {
        perror("sigaction: SIGTERM");
        exit(1);
    }
}

```

Fig: Signal with sigaction

Helper Functions: Various helper functions are implemented, such as `astrim()` to remove leading and trailing whitespaces from a string, `create_log_entry()` to create log files for executed commands, and `execute_command()` to execute a given command with input/output redirection and piping.

**main program:** The main function runs an infinite loop, reading user input and parsing it into commands, input and output redirections, and pipes. It executes each command sequentially and manages file descriptors for input and output redirection and piping.

The program exits when the user inputs ':q'.

Using `strtok` I parsed the program. If there is any `>` `<` redirection then the commands file descriptor are set accordingly. If it is part of a pipe (`|`) chain, then pipes input and output descriptors are set. Using `dup2` I change the standard input, output descriptor to commands descriptor.

Based on the number of commands it creates child processes, and waits till the child processes finish executing. I used `execl` to execute the command.

### Helpers:

```

void trim(char *str);
void create_log_entry(char *command, pid_t pid);
void execute_command(Command *cmd, int input_fd, int output_fd);
void signal_handler(int sig);
void register_signals();

```

Trim method removes extra white spaces from the beginning and end of a command.

Create\_log\_entry method creates a log file of each executed command.

Signal handler handles a signal on arrival.

Register\_signal method creates proper signaling mechanisms with sigaction.

Execute\_command method executes a given command. This function bears the heavy lifting of the program.

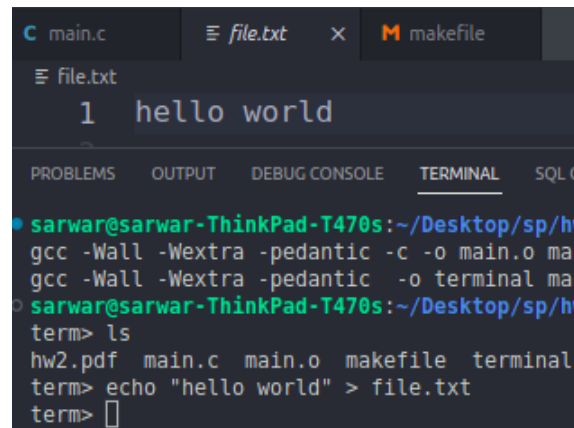
#### Test cases:

-> ls

```
• sarwar@sarwar-ThinkPad-T470s:~/Desktop/sp/hws/hw2$ make
gcc -Wall -Wextra -pedantic -c -o main.o main.c
gcc -Wall -Wextra -pedantic -o terminal main.o
• sarwar@sarwar-ThinkPad-T470s:~/Desktop/sp/hws/hw2$ ./terminal
term> ls
hw2.pdf main.c main.o makefile terminal
term> █
```

Output: all directories listed

-> echo "hello world" > file.txt



The screenshot shows a code editor with a tab for 'file.txt' containing the text '1 hello world'. Below the editor is a terminal window showing the following commands and output:

```
• sarwar@sarwar-ThinkPad-T470s:~/Desktop/sp/hw
gcc -Wall -Wextra -pedantic -c -o main.o ma
gcc -Wall -Wextra -pedantic -o terminal ma
• sarwar@sarwar-ThinkPad-T470s:~/Desktop/sp/hw
term> ls
hw2.pdf main.c main.o makefile terminal
term> echo "hello world" > file.txt
term> █
```

Output : file.txt created with "hello world" in it.

-> ls | grep "f"

```
term> ls | grep "f"
file.txt
hw2.pdf
makefile
term> █
```

Output: listed all files containing f

-> sort < file.txt

```
term> sort < file.txt
hello world
term> █
```

Output: contents of file.txt

-> ctrl + c

```
term> ^C
Caught signal SIGINT. Use ':q' to exit
term> ^C
Caught signal SIGINT. Use ':q' to exit
term> ^C
Caught signal SIGINT. Use ':q' to exit
term> ^C
Caught signal SIGINT. Use ':q' to exit
term> 
```

Output: caught sigint signal