

C_Shell Implementation Report: Phase 2

Introduction

In this report, we discuss the implementation and upgrade of our C_Shell from Phase 1 to Phase 2, which introduces remote access capabilities through Socket communication. The C_Shell now consists of a client-server architecture, where the client takes user inputs and sends requests to the server for execution. The server executes the requested commands and returns the results to the client. We have created a client.c file for the client-side implementation and kept the original main.c as the server.

Usage

The upgraded C_Shell now supports remote access, allowing users to interact with the server through the client. The user can input commands, composed commands, or programs to execute via the client. The server processes these requests and sends back the output or result, which is displayed on the client's screen.

Test Cases and Results

We conducted test cases to validate the functionality of the C_Shell with remote access. Below are some key test cases and their results:

- On the server (main.c), execute the server program: ./a.out
- On the client (client.c), connect to the server: ./c.out

Basic Command Execution

Client Input: ls

Expected Result: The client should send the 'ls' command to the server, and the server should execute it. The client should receive and display the directory listing.

Actual Result: The command executed on the server, and the client received and displayed the directory listing.

```
sb7677@DCLAP-V1156-CSD: ~ × + ▾
sb7677@DCLAP-V1156-CSD:~/c_shell$ make all
make: Nothing to be done for 'all'.
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...

sb7677@DCLAP-V1156-CSD: ~ × + ▾
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./c.out
Socket successfully created..
connected to the server..
terminal> ls
Sending: ls
a.out
client.c
c.out
main.c
Makefile
shell_commands
utils
terminal> |
```

Redirection and Pipe Operators

Client Input: *composed commands*

Expected Result: The client should send the composed commands to the server, and the server should execute it. The client should receive and display the last command in a pipe.

Actual Result: The command executed on the server, and the client received and displayed the results.

```
sb7677@DCLAP-V1156-CSD: ~ × + ▾
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...

sb7677@DCLAP-V1156-CSD: ~ × + ▾
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./c.out
Socket successfully created..
connected to the server..
terminal> echo "Hello World" > hello.txt | mkdir Phase2 | wc < client.c
Sending: echo "Hello World" > hello.txt | mkdir Phase2 | wc < client.c
78 236 1847
terminal> ls -l
Sending: ls -l
total 72
-rwxrwxr-x 1 sb7677 sb7677 22128 Oct 28 20:41 a.out
-rw-rw-r-- 1 sb7677 sb7677 1847 Oct 28 20:40 client.c
-rwxrwxr-x 1 sb7677 sb7677 17296 Oct 28 20:41 c.out
-rw-rw-r-- 1 sb7677 sb7677 14 Oct 28 21:16 hello.txt
-rw-rw-r-- 1 sb7677 sb7677 2655 Oct 28 20:40 main.c
-rw-rw-r-- 1 sb7677 sb7677 165 Oct 28 20:40 Makefile
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 28 21:16 Phase2
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 28 20:40 shell_commands
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 28 20:40 utils
terminal>
```

Executable File

Client Input: execute executable/output file

Expected Result: Print the output of the file

Actual Result: The command executed on the server, and the client received and displayed the output of the file.

```
sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V1156-CSD: ~ X + v
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
|

sb7677@DCLAP-V1156-CSD:~/c_shell$ ./c.out
Socket successfully created..
connected to the server..
terminal> pwd | ./hello
Sending: pwd | ./hello
Hlo World
terminal> |
```

Error Fixation

```
sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V1156-CSD: ~ X + v
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
|

sb7677@DCLAP-V1156-CSD:~/c_shell$ ./c.out
Socket successfully created..
connected to the server..
terminal> cat client.c | grep a |
Invalid input
terminal> |
```

Stderr transmission

```
sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V11: X sb7677@DCLAP-V1156-CSD: ~ X + v
sb7677@DCLAP-V1156-CSD:~/c_shell$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
|

sb7677@DCLAP-V1156-CSD:~/c_shell$ ./c.out
Socket successfully created..
connected to the server..
terminal> ls fsrdsdhjgdsds
Sending: ls fsrdsdhjgdsds
ls: cannot access 'fsrdsdhjgdsds': No such file or directory
terminal> |
```

Implementation Details

The implementation of the C_Shell with remote access involves the following components and techniques:

- We have implemented socket communication using the `socket()`, `bind()`, `listen()`, and `accept()` functions on the server-side and `socket()`, `connect()`, and `send()/recv()` functions on the client-side.

- The server processes user commands using techniques similar to those in Phase 1. It uses `fork()`, `exec()`, `wait()`, and `dup2()` to execute commands and handle redirection.
- The client accepts user input and sends it to the server for execution. The server sends the results back to the client, which then displays them on the screen. A command 'exit' is implemented for termination.

C_Shell Implementation Report: Phase 1

Introduction

In this report, we describe the implementation of a custom C_Shell program. The C_Shell is designed to take Linux commands as input from the user and execute them. We have employed various system calls and techniques, such as `fork()`, `exec()`, `wait()`, `dup2()`, and `pipe()`, to create a functional shell. This report provides details about the program's usage, test cases, and an overview of the implementation.

Usage

The C_Shell provides a command-line interface that allows users to enter Linux commands. The shell supports basic command execution and redirection, as well as pipes for connecting multiple commands in a pipeline. To execute a command, users simply input it and press Enter. The shell will then execute the command and return the result.

Test Cases and Results

Below are some test cases to verify the functionality of our C_Shell.

Basic Command Execution

Input: `ls -l`

Expected Result: The shell should execute the '`ls -l`' command and display the directory listing.

Actual Result: The command executed successfully, and the directory listing was displayed.

```
sb7677@DCLAP-V1156-CSD: ~ X + v
sb7677@DCLAP-V1156-CSD:~/c_shell$ make && ./a.out
gcc -Wall -o a.out main.c utils/parser.c shell_commands/commands.c
terminal> ls -l
total 36
-rwxrwxr-x 1 sb7677 sb7677 17776 Oct  9 22:00 a.out
-rw-rw-r-- 1 sb7677 sb7677 1054 Oct  9 22:00 main.c
-rw-rw-r-- 1 sb7677 sb7677 110 Oct  9 22:00 Makefile
drwxrwxr-x 2 sb7677 sb7677 4096 Oct  9 22:00 shell_commands
drwxrwxr-x 2 sb7677 sb7677 4096 Oct  9 22:00 utils
terminal> pwd
/home/sb7677/c_shell
terminal> echo hello
hello
terminal> |
```

Redirection

Input: `cat file.txt > output.txt`

Expected Result: The shell should redirect the contents of 'file.txt' to 'output.txt'.

Actual Result: Redirection worked as expected, and the contents were copied to 'output.txt'.

```
sb7677@DCLAP-V1156-CSD: ~ X Windows PowerShell
sb7677@DCLAP-V1156-CSD:~/c_shell$ make && ./a.out
gcc -Wall -o a.out main.c utils/parser.c shell_commands/commands.c
terminal> echo hello > new-file
terminal> cat < new-file
hello
terminal> ls frthtrhrjy
ls: cannot access 'frthtrhrjy': No such file or directory
terminal> ls frthtrhrjy 2> errout-file
terminal> cat errout-file
ls: cannot access 'frthtrhrjy': No such file or directory
terminal> |
```

Pipe Operator

Input: `cat output.txt | grep hello | grep there`

Expected Result: The shell should pipe the output of 'cat output.txt' to 'grep hello'.

Actual Result: The pipe operator correctly connected the two commands, and 'grep' filtered the lines containing the specified keyword.

```
sb7677@DCLAP-V1156-CSD: ~ X + v
sb7677@DCLAP-V1156-CSD:~/c_shell$ make && ./a.out
make: 'a.out' is up to date.
terminal> echo hello there | grep hello | grep there > output.txt
terminal> cat < output.txt
hello there
terminal> cat < output.txt | grep hello
hello there
terminal> pwd | ls -l | wc < output.txt
1 2 12
terminal> pwd | ls | ls -l | ps
      PID TTY          TIME CMD
3175263 pts/19    00:00:00 bash
3175381 pts/19    00:00:00 a.out
3175860 pts/19    00:00:00 ps
terminal> |
```

Exit Command

Input: exit

Expected Result: The shell should exit gracefully.

Actual Result: The shell terminated as expected, returning control to the system.

```
sb7677@DCLAP-V1156-CSD: ~ X + v
terminal> pwd
/home/sb7677/c_shell
terminal> ls
a.out main.c Makefile new-file output.txt shell_commands test-file utils
terminal> ls -l
total 48
-rwxrwxr-x 1 sb7677 sb7677 17776 Oct 9 22:00 a.out
-rw-rw-r-- 1 sb7677 sb7677 1054 Oct 9 22:00 main.c
-rw-rw-r-- 1 sb7677 sb7677 110 Oct 9 22:00 Makefile
-rw-rw-r-- 1 sb7677 sb7677 6 Oct 9 22:11 new-file
-rw-rw-r-- 1 sb7677 sb7677 12 Oct 9 22:30 output.txt
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 9 22:00 shell_commands
-rw-rw-r-- 1 sb7677 sb7677 6 Oct 9 22:12 test-file
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 9 22:00 utils
terminal> rm new-file
terminal> ls -l
total 44
-rwxrwxr-x 1 sb7677 sb7677 17776 Oct 9 22:00 a.out
-rw-rw-r-- 1 sb7677 sb7677 1054 Oct 9 22:00 main.c
-rw-rw-r-- 1 sb7677 sb7677 110 Oct 9 22:00 Makefile
-rw-rw-r-- 1 sb7677 sb7677 12 Oct 9 22:30 output.txt
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 9 22:00 shell_commands
-rw-rw-r-- 1 sb7677 sb7677 6 Oct 9 22:12 test-file
drwxrwxr-x 2 sb7677 sb7677 4096 Oct 9 22:00 utils
terminal> exit
sb7677@DCLAP-V1156-CSD:~/c_shell$ |
```

Implementation Details

The C_Shell is implemented as follows:

- Each command is executed in an individual child process spawned from the main process using the fork() system call.
- The execvp() system call is used to execute the user-entered command with its arguments.
- Input and output are redirected to files using the < and > operators, achieved through the dup2() system call.
- The pipe() system call is used to connect the standard output of one command to the standard input of another when the pipe symbol | is used. The shell ensures that the next process/command waits for the last process in the pipeline to terminate before proceeding.
- The shell employs a while loop to continuously prompt the user for input until the 'exit' command is entered.

List of fifteen commands:

- pwd
- ls (all arguments)
- ps (all arguments)
- clear
- cat
- touch
- rm (all arguments)
- echo
- mkdir
- grep
- wc
- curl (www.google.com)
- nslookup (www.google.com)
- chmod
- env
- whoami
- tty

