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:

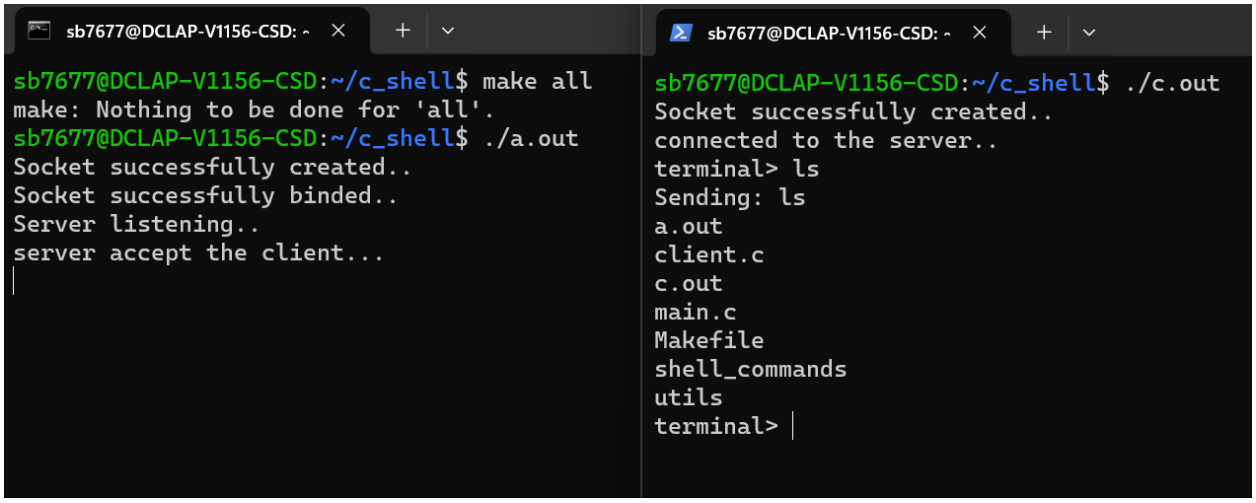
- On the server (main.c), execute the server program: ./a.out
- On the client (client.c), connect to the server: ./c.out

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.

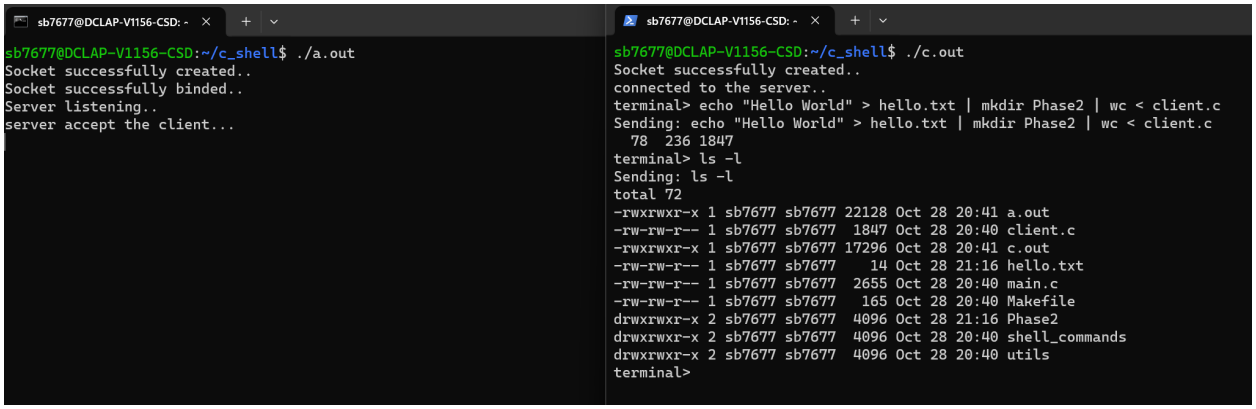


Redirection and Pipe Operators

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.

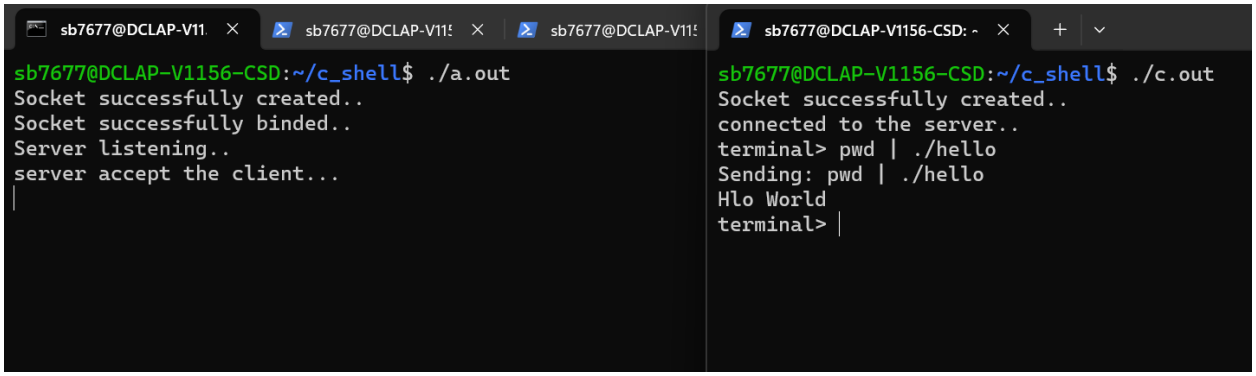


Executable File

Client Input: execute executable/output file

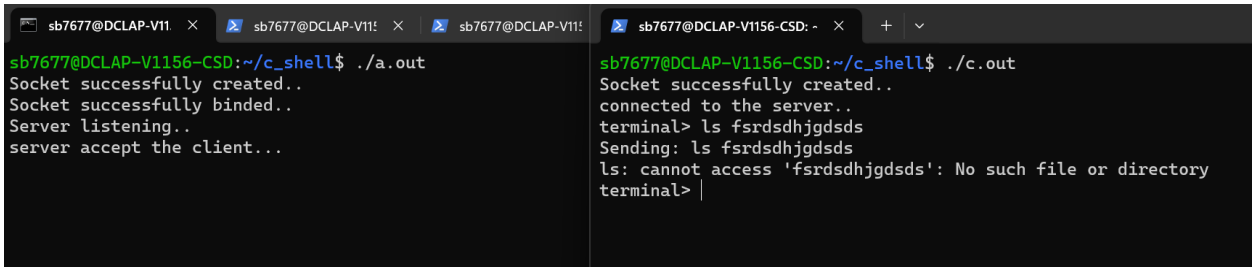
Expected Result: Print the output of the file

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



Error Fixation

Stderr transmission



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

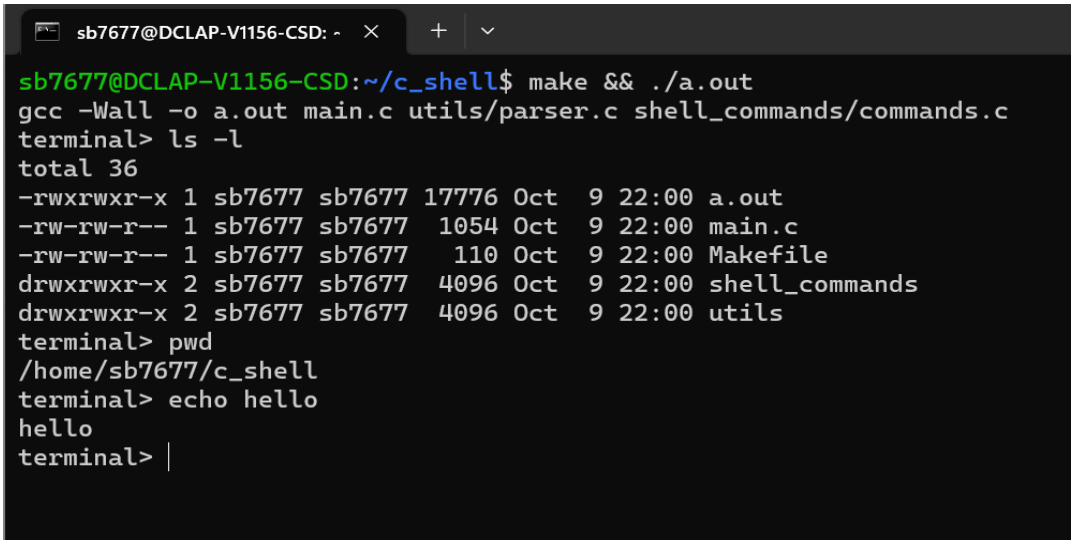
Below are some test cases to verify the functionality of our C_Shell.

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.

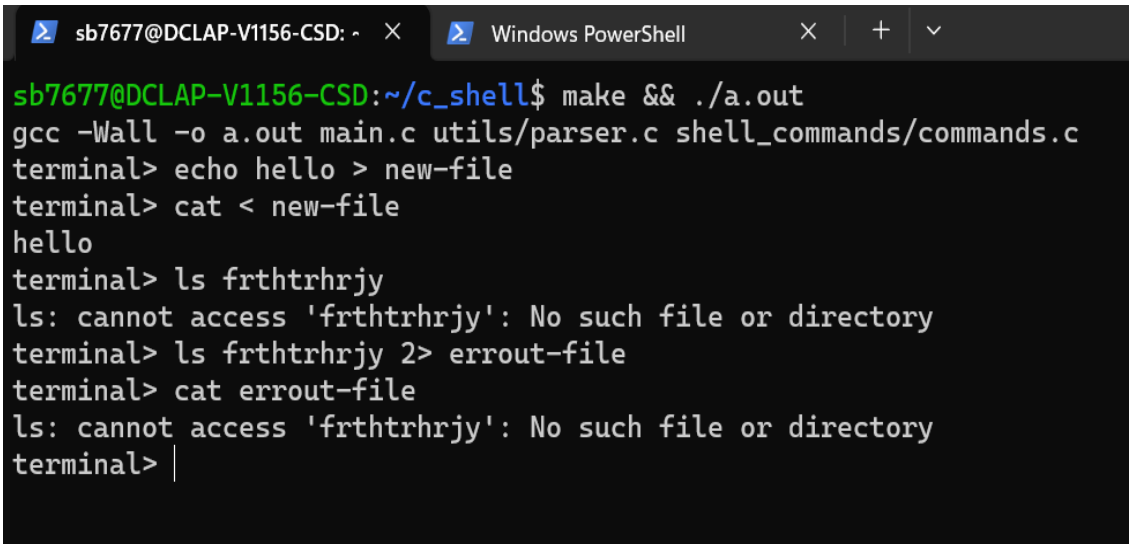


Redirection

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'.

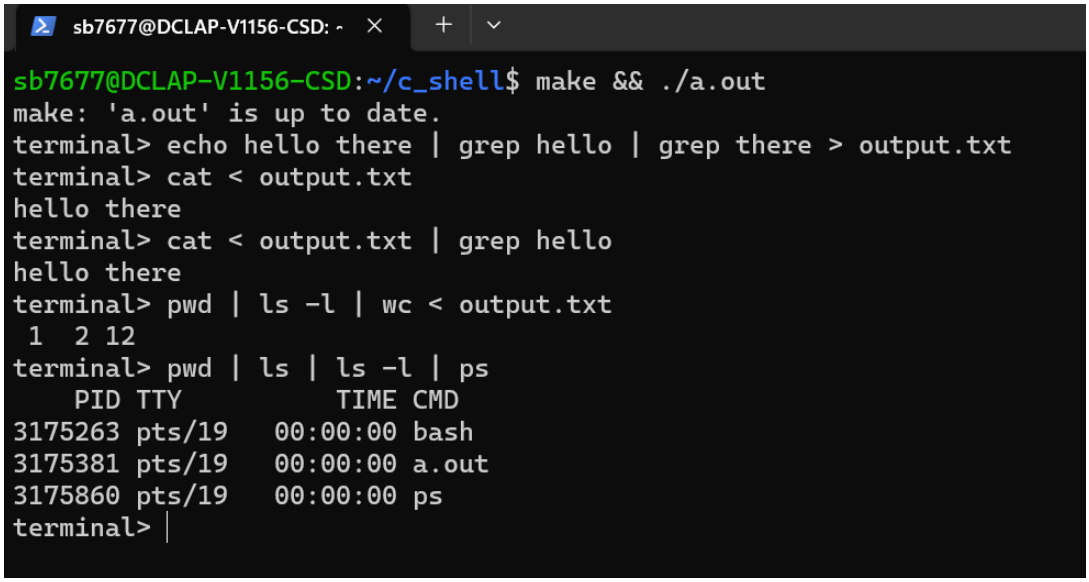


Pipe Operator

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.

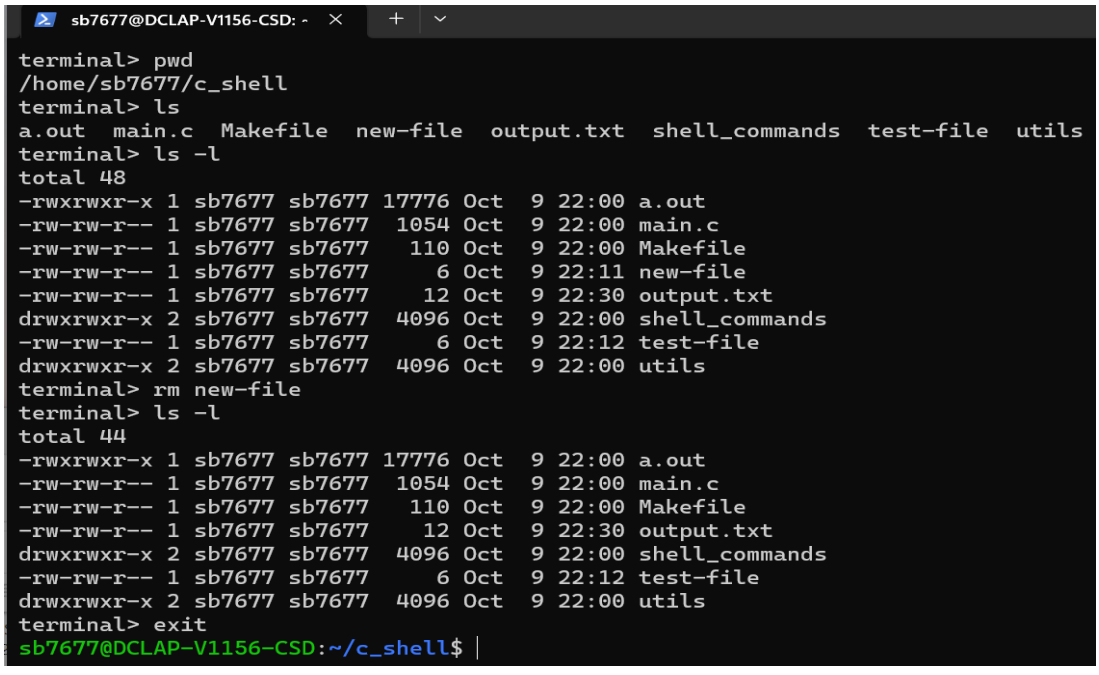


Exit Command

Input: exit

Expected Result: The shell should exit gracefully.

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



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