Programming Assignment 3 Report

Simple Flexible Communication and Transfer Protocol

Christoph Bakshi
88988739
cbakshi@ufl.edu

CIS 4930 – Internet Networking Technologies

Abstract

The Simple Flexible Communication and Transfer Protocol (SFCTP) is an application-level protocol that enables simple communication and file transfer capabilities between two devices. The Protocol enables parallel operation between the two devices, including both file sending and file receiving functionality for both connected ends. The application utilizes TCP for communication. This document defines the current and future operational framework of SFCTP.

Status of This Memo

This document covers primarily the currently operational capabilities of SFCTP but also delves into future plans for the Protocol. There are ongoing changes to the protocol, and this document will reflect the latest operational version. However, it is quite possible for the described version to contain errors, software bugs, and potential unhandled error states not known to the protocol developer. Testing was done on lab computers lin114-04 and lin114-05.

Table of Contents

Basic Code Compilation and Execution Instructions (Linux)

Preparation Steps

1. Ensure the proper file structure on each machine.
    a. In a single folder, there should be "**FileIO.java**", "**makefile**", "**tcp_comm.java**", and any nonessential files that the program should have access to (such as an image to be sent to connecting devices).
    b. This same file structure should be on every device, though the nonessential files on each device may vary as desired.
2. Compile the code.
    a. On every device, navigate with terminal to the directory containing the files listed above and execute the "**make**" command; the code should now be compiled.

Execution Steps

1. On each device, run "**java tcp_comm**" to start the program.
2. Run commands as desired by typing in the command with any required arguments as specified.

Available Commands

- *Note that all commands listed do require that the current input source is the user; the program will not respond to user-entered commands will under the control of a client.*
- *Arguments should be separated by a space unless otherwise noted; arguments should not contain a space within them unless otherwise noted.*
- *File names should include the file extension.*

Command Structure	Description	Required Program State
q	Disconnects if connected and closes the program.	Any
pr_cmd	Displays a list of the available commands. Note that this list does not discriminate according to program state.	Any
cmd_help <command>	Provides more detailed information about a specified command. Does not provide information about all commands. Calling this command with no arguments provides information about *cmd_help* itself.	Any
ping	Sends a signal to the connected device and awaits a simple return signal; displays time between signal being sent and response being received.	Connected
copy <filename>	Creates a copy of the specified file; the copy will be placed in the same location as the original and will have the same name, preceded by "c_"	Any
w_f_c <port number>	Waits for a connection by another program. Opens a server with the specified port number. This will terminate any pre-existing connection. WARNING: This command will switch the current	Any

	input source to the connected device immediately. Thus, the program will not be locally controllable until a device connects and, eventually, disconnects, at which time control will be return to the local user.	
connect <hostname> <port number>	Connects to a waiting server with the specified hostname and port number created with the *w_f_c* command. This will terminate any pre-existing connection.	Any
upload <filename>	Sends a copy of the specified file to the connected device, if possible.	Connected
download <filename>	Retrieves a copy of the specified file from the connected device, if possible.	Connected

Unavailable, Nonfunctional, and Planned Commands

These commands are not currently usable but are under development or may be under development in the future. Note that not all of these commands will be implemented, and implementation and format is subject to change. These commands will follow the same basic sets of rules as do Available Commands.

Command Structure	Description	Required Program State
do <filename>	Executes a command file; executes a series of instructions stored in the specified file as if they were entered into the program. Each command should follow the previously established set of rules, and there should be one command per line, with no extra whitespace before or after each command. WARNING: This command will switch the current input source to the command file until the command file's instructions have been executed. Note that there is the possibility of an infinite loop/infinite recursion if the command file contains a *do* command.	Any
send <command>	Send a message or command directly to the connected device as if controlling that device locally through user input.	Connected
set <variable identifier> <new value>	Modify certain variables within the program, such as file transfer slice size (when receiving), which print statements to show, local device name within the program, and more.	Any
disconnect	Disconnect from a connected device without exiting the program.	Connected

Description of Code and Structure

Classes

The code employs two java classes: *FileIO* and *tcp_comm*. *FileIO* handles reading and writing to files and is not dependent on other created classes from the program. *tcp_comm* handles everything else, including taking and parsing user input, establishing network connections, and communication over the network using TCP.

Command Functionality

Commands are stored in and retrieved from a queue data structure, meaning that the order that commands are entered determine the order that they are executed. Currently, all commands are executed as soon as they are received, so this is not significant, but the usage of the queue is intended to assist with the implementation of the *do* command for a premade set of commands to be entered and executed in order.

Commands are parsed and acted upon (if recognized and situationally appropriate) immediately upon parsing. If the command is not recognized or does not make sense in the given context, an error message will inform the user of the issue. Parsing is done based on the usage of spaces, meaning that any individual argument cannot contain a space. Keep this in mind when naming files for usage with this program.

Additional Features and Notes

The program automatically creates a log of events and saves the log upon program exit. The log is saved as a text file and is uniquely identified by the session ID, which is based on the time that the program was started and is displayed at the start of the program execution.

File transfer has been tested up to 300 KB and was tested with files with a ".png" file extension.

Future Plans

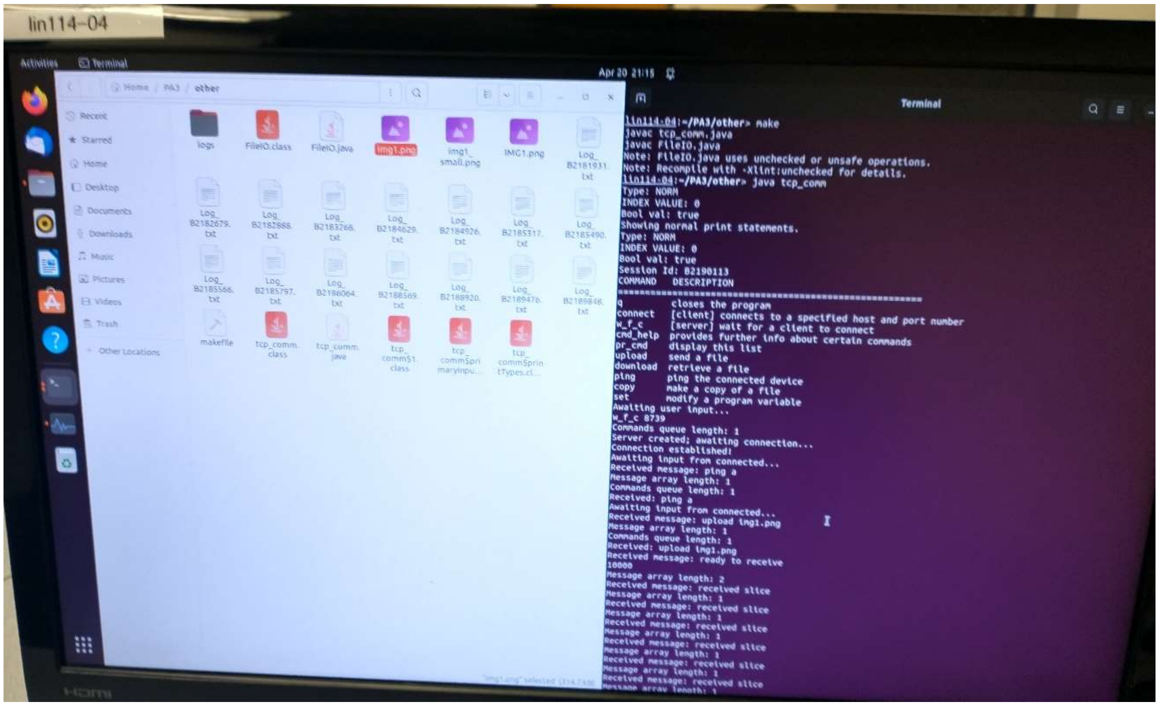
In later versions of this protocol, some form of the Unavailable Commands will be implemented, most likely in the way described in that section. *println*-type functions will be replaced in usage with *output* functions, which will decide whether to print output to the local device, send the output string over the connection, or both. Future versions of the protocol will be built around a multithreaded framework, allowing for handling of very rare hang events on the server program and also for overriding processes on the server side without waiting for clients to return control to the server. This will also allow for each device to form multiple simultaneous connections. The program log will also be improved for clarity and conciseness, and it will be split into two logs: one which only records major events, and one which records many more details.

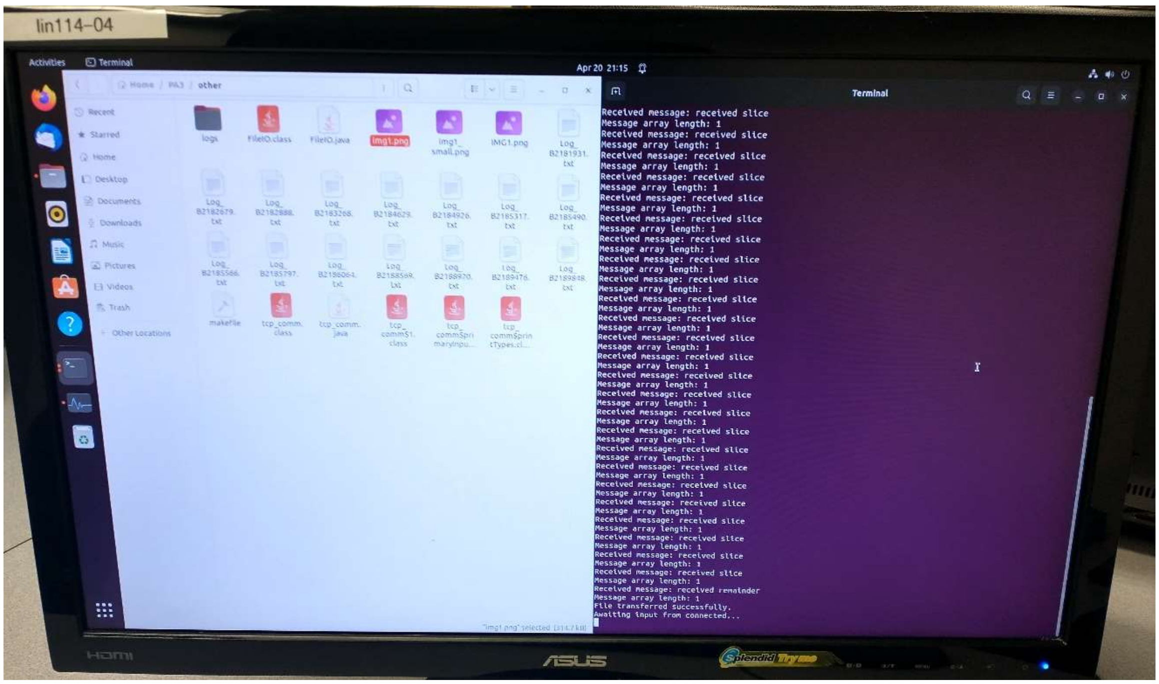
In addition, more flexibility will be added, and the difference between the functioning of server and client will be further diminished, as the primary input source will not be based solely on who awaited a connection from the other.

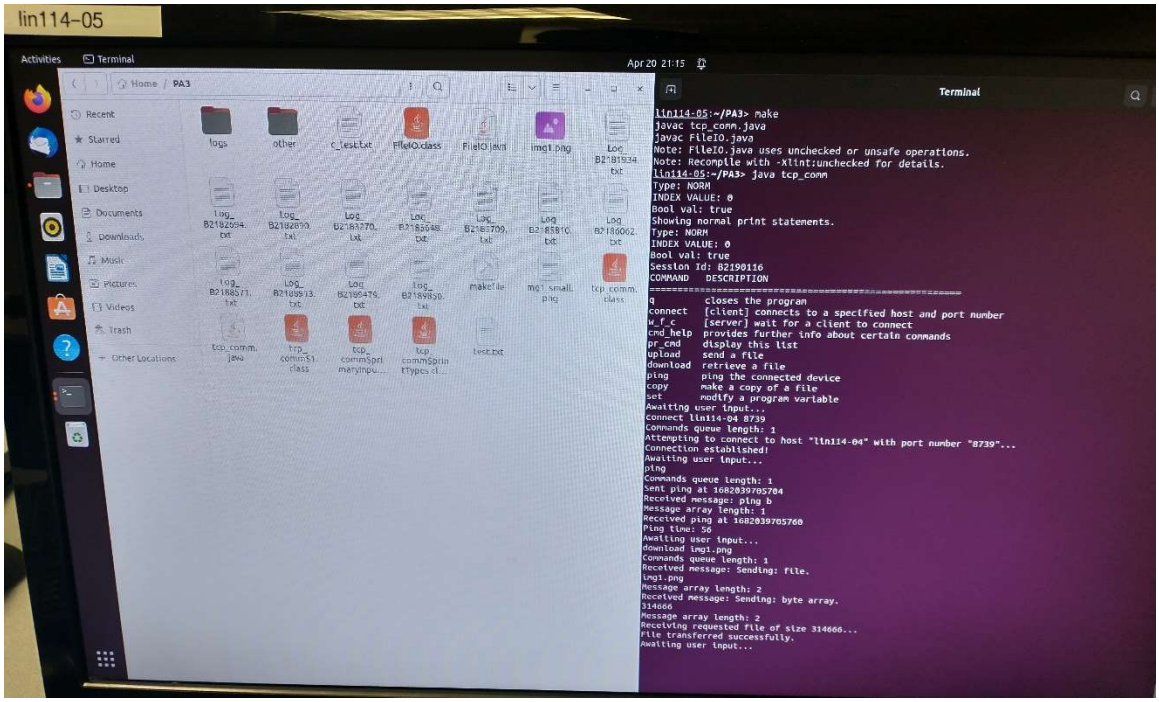
Program Execution Examples

Visual Example

Screenshots of program execution on Linux machines:







Single Instance Example

img0.png is duplicated, the commands list is shown, and additional info for the *connect* command is shown.

Single Instance
copy img0.png
pr_cmd
cmd_help connect
q

Dual Instances Example

A client device connects to a host with hostname serverHostName. A ping is executed, and then *img1.png* is copied from the server to the client, and *img2.png* is copied from the client to the server. Note that no local user input will be accepted or used on the grayed-out lines, as the server device is, during those points, taking commands only from the connected client.

Instance 1 (Executed first for each line)	Instance 2 (Executed after Instance 1 in each line)
w_f_c 8739	connect serverHostName 8739
	ping
	download img1.png
	upload img2.png
	q
q	

Video Demonstration

https://youtu.be/1iYCZ3tVeJQ