2803ICT – Assignment 1

Jesse Schneider S5087288

Contents

Contents

1. Problem Statement..... 1
2. User Requirements..... 2
3. Software Requirements..... 2
4. Software Design..... 4
   - Logical Block Diagram..... 4
   - Functions..... 4
   - Client:..... 4
   - Server:..... 5
   - Data Structures..... 6
5. Requirement Acceptance Tests..... 7
6. Detailed Software Testing..... 9
7. User Instructions..... 10

1. Problem Statement

To complete this assignment, a Client Server program needed to be created, with the ability to execute commands on the Client that utilized the Server. The required commands were:

Put – to store a program from the Client on the Server
Get – to retrieve a previously stored program from the Server to the Client
Run – to compile (if required) and execute a C program on the Server and send results to the Client
List – to list either all programs on the server, or the files in a program (long list possible)
Sys – to retrieve information about the Server to the Client
Quit – Close and exit the Client

All of the above have been completed.

2. User Requirements

A user of the Client needs to:

Enter any of the commands at any time
Receive feedback on completion of processes/commands
Use any local file for any of their file inputs
Have the Client and Server running in the same or different directories
They do not need to know about the structure of the Server for it to work
For ease of socket use, socket buffers are limited to 1024 bytes
Should the user ever do something incorrectly, they will be provided with feedback as to what went wrong

3. Software Requirements

1. Using a socket connection, the client program will query the remote server program that is listening on port 80.

2. The IP address of the server to be queried by the client shall be given to the client as a command line argument.

3. The client shall wait for the user to enter queries (via stdin), which it then forwards to the server in a loop until the user types ‘quit’. Any responses from the server are immediately displayed to the user.

4. The client will report the time taken for the server to respond to each query together with the server’s response.

5. The client is non-blocking. An infinite number of server queries may be outstanding.

6. The server will spawn a new process to execute each new request and must be able to accept multiple clients.

7. The server will be able to accept one or more source files and a ‘progname’ and place the files in a directory called ‘progname’. It will be able to compile the source files (if not previously compiled), run the executable with command line arguments provided from the client and return result to the clients.

8. The following query commands (and options) are to be recognised by the server (anything within [] is optional):

    A. put progname sourcefile[s] [-f] : upload sourcefiles to progname dir, -f overwrite if exists.

    B. get progname sourcefile : download sourcefile from progname dir to client screen.

C. run progname [args] [-f localfile] : compile (if req.) and run the executable (with args) and

9. either print the return results to screen or given local file.

10. D. list [-l] [progname] : list the prognames on the server or files in the given progname directory to the screen, –l = long list

11. E. sys : return the name and version of the Operating System and CPU type.

12. The long list (-l) option of the list command will also return the file size, creation date and access permissions. If no progname is given, then the list of all available progname directories will be returned.

13. The get command will dump the file contents to the screen 40 lines at a time and pause, waiting for a key to be pressed before displaying the next 40 lines etc.

14. The put command will create a new directory on the server called ‘progname’ If the remote progname exists the server will return an error, unless -f has been specified, in which case the directory will be completely overwritten (old content is deleted). This command allows you to upload one or more files from the client to the server

15. If a localfile option is given to the run command a new file on the client will be created. If the localfile name exists the client will return an error, unless -f has been specified in which case the file will be overwritten. If a file with that name already exists the client will return an error before sending the get request to the server.

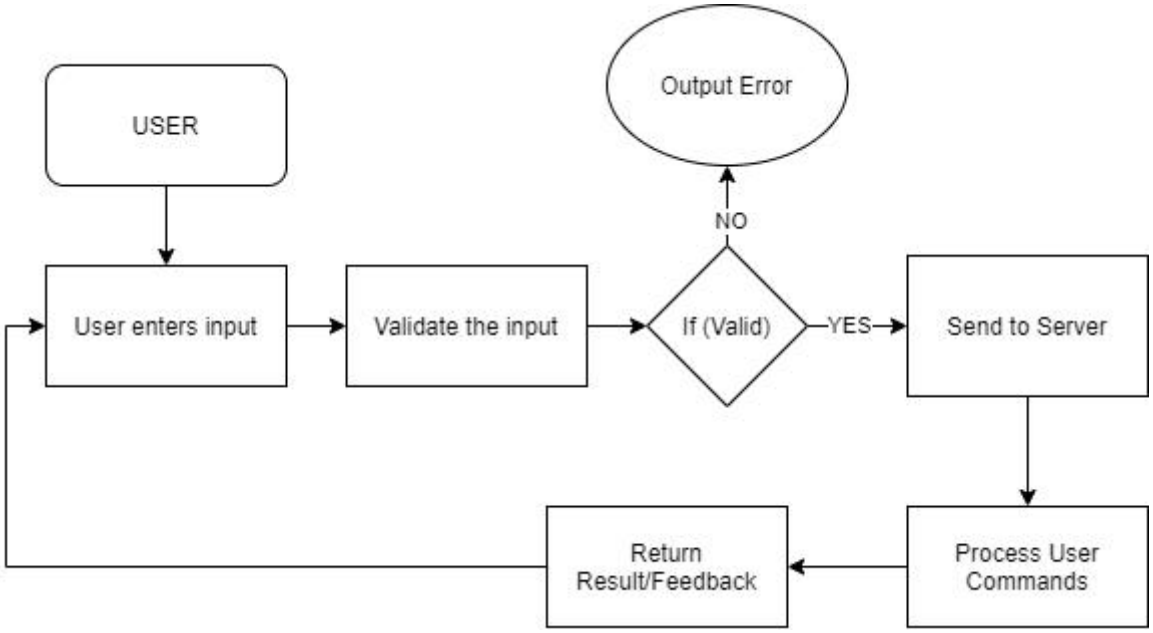
16. The run command will check to see if a ‘progname’ has been compiled, and if not will compile the relevant files as require. run will initiate a compile if there is no executable in the folder, or its creation date is older than the last modified date of a source file. It will then run the executable, passing to it any specified command line arguments, and the server will redirect output from the executed program to the client. If the program can’t be run (or compiled) an appropriate error will be returned to the client. You must not use the system() call to compile or run the ‘progname’.

17. If the server receives an incorrectly specified command it will return an error. If the server is unable to execute a valid command the server will return the error string generated by the operating system to the client.

18. All Zombie processes are terminated as required. There is to be no unwanted Zombie processes on either the client, or the server.

4. Software Design

Logical Block Diagram



Functions

Client:

Put

Void readFile(FILE*, int);
- Read local file, send to Server
- Inputs: File Pointer to read (FILE *), socket to send file to (int)

Char* parseFFlag(char*, char*);
- Check if overwrite is forced
- Inputs: input command to search (char*), delimiter character (char*)
- Return: found or not found

Void sendFiles(int, char*, char*, FILE*);
- Faciltates using readFile() to send all files to the Server
- Inputs:socket to send to (int), the entered filenames (char*), delimiter character (char*), File Pointer to use to open files (FILE*)

Run

Void writeToFile(char*, char*);
- Function to write run output to given file
- Inputs: buffer to write to file (char*), filename to write (char*)

Int countCommands(char*, char*);
- Count and return the number of arguments
- Inputs: the command string (char*), delimiter character (char*)
- Return: number of command arguments

Void allocateArgs(char***, int);

- Allocate dynamic memory for the arguments 2D array
- Inputs: pointer to dynamic 2D array (char ***), number of arguments (int)

Server:

Server functions

Void killZombie(int);

- Kills zombie Processes on the socket int
- Inputs: process signal (int)

Void sendError(char*, char*, int);

- Generic function for returning errors over sockets to the Client
- Inputs: dir/file name for error (char*), error text (char*), socket (int)

Char * getPath(char*);

- Custom get CWD function
- Inputs: directory to add to path (char*)
- Return: new directory path

Char * getSourcePath(char*, char*);

- Custom get CWD function
- Inputs: file to add to path (char*), current path (char*)
- Return: new file path

Put

Void writeFile(char*, char*, char*);

- Write a buffer to file
- Inputs: filename to write (char*), buffer to write to file (char*), path to write to (char*)

Void putFile(int, char*);

- Facliates using writeFile() to put whatever files needed into a given directory
- Inputs: socket to receive from (int), current path (char *)

Get

Void sendFile(int, char*);

- Function for facilitating sending a file to the Client
- Inputs: socket to send to (int), file to be sending (char *)

Void readFile(FILE *, int);

- Function to Read a local file into memory
- Inputs: File Pointer to open (FILE *), socket to send to (int)

Run

Int checkIfCompile(time_t, char*);

- Function to compare modified dates to check whether or not to compile

- Inputs: time of binary modification (time_t), directory to check in (char*)
- Return: 1 if needing to recompile, -1 otherwise

Void compile(char*, char*);

- Function to compile all C files in a directory into object files and then link to a single binary
- Inputs: directory path to compile (char*), directory name to compile (char*)

Char* executeBin(char*, char*, char*);

- Function to execute the compiled binary
- Inputs: directory path (char*), program name to execute (char*), optional arguments (char*)
- Return: the output of the program execution

System

Void systemInfo(int);

- Find and send system information to the Client
- Inputs: socket to send information to

List

Char* performLs(char*);

- Different faciliations of the “ls” command
- Inputs: the directory to perform ls in (char*)
- Return: the result of ls as a string

Data Structures

Only 1 data structure was used : Args[][]

It was used to store a variable list of arguments for the run command on the Client.

Other than that, only ints, chars, char[] and pointers were used (a lot of char*).

5. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	Server IP address as a command line argument	Full		
2	Client reports time taken for server responses	Full		
3	Client runs continuously in a loop but is non-blocking - can have outstanding queries	Full		
4	Server spawns new process for new requests	Full		
5	Server accepts multiple simultaneous requests from multiple clients	Full		
6	Correct operation of put command with -f option	Full		
7	Correct server operation of get command	Full		
8	Correct server operation of run command - file compilation	Full		
9	Correct server operation of run command - execute with params	Full		
10	Correct operation of run command - returns results to client	Full		
11	Correct operation of list command with -l option	Full		
12	Correct operation of sys command	Full		

13	Client: scrolling list that pauses after 40 lines on list and get command	Full		
14	Server error handling	Full		
15	Zombie removal	Full		

6. Detailed Software Testing

No	Test	Expected Results	Actual Results
1.0	**Files**		
1.1 1.2 1.3 1.4	- Don’t exist/misspelt - Output with different extensions - Files in different Directories - Correct Directory, wrong file name	For all test cases, a relevant (specific) error is shown.	As expected.
2.0	**Directories**		
2.1 2.2	- Incorrect names - Directorys that already exist without the ‘-f’	2.1 – Error message displayed 2.2 – Don’t let the User overwrite the data	As expected.

7. User Instructions

Navigate to the Client directory
Navigate to the Server directory in a separate terminal
Run make in both directories if compiling is required
Run ./server to start the Server
Run ./client <ip address> where <ip address> = the Server's address (local machine), if using the same machine, you can enter "127.0.0.1" for localhost
Now both processes are running, the user can enter any of the defined commands (put, get, run, sys, list)
Enter quit to exit