2803ICT Systems and Distributed Computing
Assessment 1 - Sockets

Thomas Carney
s5130828

Contents

Problem Statement

The goal of this project was to develop a client and a server capable of completing small tasks. The client will issue instructions to the server and the server will execute these instructions and return the results back to the client. These instructions include: sending files from client to server, reading files from the server to the client, compiling and executing c files on the server from the client and listing the directories and files on the server. To connect to the server, the client will take the server's IP address as a command line argument.

User Requirements

1. The user will connect to the server using the client program. When the client program is run, the server IP address will be given as a command line argument.

2. The user can enter as many commands as desired until the quit command is entered, this will disconnect from the server and exit the program.

3. Any file types can be moved from the client to the server. They can also be read 40 lines at a time from the server.

4. The commands that can be entered by the user can be seen below. (bold are commands, [] are optional parameters.
    a. ***put progname sourcefile[s] [-f]*** : upload source files to progname dir, -f overwrite if exists.
    b. **get progname sourcefile** : download source file from progname dir to client screen.
    c. **run progname [args] [-f localfile]** : compile (if req.) and run the executable (with args) and either print the return results to screen or given local file.
    d. **list [-l] [progname]** : list the prognames on the server or files in the given progname directory to the screen, –l = long list
    e. **sys** : return the name and version of the Operating System and CPU type.

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 the 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 either print the return results to screen or given local file.
    d. **list [-l] [progname]** : list the prognames on the server or files in the given progname directory to the screen, –l = long list
    e. **sys** : return the name and version of the Operating System and CPU type.

9. 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.

10. 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.

11. 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

12. 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.

13. 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’.

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

Software Design

Functions

void ensure_compiled(char *arg0, struct stat st)

This function ensures the .c files in the desired directory are compiled. arg0 is the directory to be checked, st is to check the actual directory. First this function checks which file was the last edited. It then checks if there is already an executable made and if there is it checks if it is the last edited file (ensuring the source files have not been updated). If either of

these are false, all .c files in the current directory are compiled into an executable. If the executable is already up to date, the function prints that it is up to date.

int give_forty(int client_socket, FILE* fp)

Send 40 lines of a given file to the client. If the end of file (EOF) is reached 1 is returned, otherwise 0 is returned once 40 lines have been sent. client_socket is used so the lines of the file can be sent to the client. fp is a file pointer to the file that is being sent.
If 1 is returned the main function will alert the client that the file has ended, if 0 is returned the main function will wait for the client's input to send another 40 lines.

void kill_zombie(int sig)

Removes all zombie processes.

Data Structures

The main data structure used was an array of char* called args. This was inspired by argv in the main function. Each command/parameter entered by the client is broken up into each arg allowing the input to be used more easily.

Throughout the program many arrays of chars are used, these all have dynamically allocated memory that is freed once they are no longer needed.

Requirements Acceptance Tests

Software Requirement No.	Test	Implemented (Full/Partial/None)	Test Results (Pass/Fail)	Comments(for parietal implementation or fail test)
1.	Using a socket connection, the client program will query the remote server program that is listening on port 80.	Full	Pass	
2.	IP address of the server given to the client as command line argument	Full	Pass	
3.	Client waits for the user to enter queries, which it then forwards to the server in a loop until the user types ‘quit’. All server responses are displayed to the user.	Full	Pass	
4.	Client reports the time taken for the server to respond with the server’s response.	Full	Pass	
5.	Client is non-blocking	Full	Pass	
6.	The server spawns new processes to execute each new request and can accept multiple clients.	Full	Pass	
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 the result to the clients	Full	Pass	

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

	before sending the get request to the server.			
13.	The run command will check to see if a ‘progname’ has been compiled, and if not will compile the relevant files as required. 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’.	Full	Pass	
14.	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. 15. All Zombie processes are terminated as required. There is to be no unwanted Zombie processes on either the client, or the server.	Full	Pass	

Detailed Software Testing

No.	Test	Expected Results	Actual Results
1.0	**Client**		
1.1	Commands sent with less arguments then necessary	Client does not send commands, prints that not enough parameters where given and continues to wait for input.	As expected
1.2 1.3	Directory entered does not exist File entered does not exists	For 1.2 - 1.3 Prints it does not exist, continues to wait for input.	As expected
1.4	Run command is given a local file but no directory to execute.	Prints error message, continues to wait for input	Exits program with EXIT_FAILURE.
1.5 1.6 1.7 1.8	Single file is moved with put. Multiple files moved with put. Files replace directory with -f flag. -f flag is entered when directory does not exist.	1.5-1.8 results in the files ending up in the correct directory.	As expected
1.9 1.10	Get is called on file with less than 40 lines Get is called on file with more than 40 lines	1.9 client receive all lines of file 1.10 client receives 40 lines of file and no more until a key is pressed for the next 40.	As expected
1.11	Run command is called on program with no arguments	1.11-1.15 output of program is displayed as expected	As expected

1.12 1.13 1.14 1.15 1.16 1.17	Run command is called on a program with multiple arguments. Run command is called to print output into a local file on client. Run command is called to print output to stdout. Run command is called to replace a local file (with -f flag) Run command is called to replace a local file (without -f flag.) Run command is called with -f flag when not needed.	1.16 Sends an error message, does not request the server to execute the program. Continues to wait for input 1.17 program runs as if -f was not entered.	
1.18 1.19 1.20 1.21	List is called to list directories List is called to list directories with -l flag List is called on a directory to show its contents List is called on a directory to show its contents with -l flag.	1.18-1.21 output list of information at the expected level of detail.	As expected
1.22	Sys is called.	1.22 displays cpu and OS info	As expected.
2.0	**Server**		
2.1	Run is called on a file with no executable	File is compiled and ran.	As expected

	Run is called on a file with an executable older than a source file.		
2.2	Run is called on a file with an up to date executable.	File is not compiled, just ran.	As expected.
2.3	Multiple client connecting at once Multiple clients requesting at once.	Multiple clients and requests are handled, server works as usual.	As expected.

User Instruction

- The server directory is to be open, server.c is compiled and ran.
- Find the IP address of the system running the server.
- The client directory is to be open, client.c is to be compiled and ran with the server IP address as an argument.
- The client will be prompted with a '>' symbol when it is connected and ready for input.
- Multiple clients may connect and use the server at once.
- Commands stated in the requirements can be entered with or without the optional ([square bracket]) parameters.