2803ICT Assignment 1
Server and Client OPTION A

Carl Humphries

Carl Humphries
s5084150

Contents

Carl Humphries
s5084150

Problem Statement

The aim of assignment 1 option A was to create a server/client system that acts as a remote execution system. This enables clients to remotely compile and run their programs as well as look at and change code they have uploaded.

The server must be able to accept an infinite number of clients and must use another process to handle that request. All commands are sent to the server which it will run and return a response or error if the command is unknown.

User Requirements

Below are the user requirements for the assignment:

- The user should only be able to use the client.
- The user can enter as many requests as they like without being blocked. (except for list and get commands).
- The user can type quit to close the client.
- The user if free to type any command but only recognised requests are sent.
- Requests can take infinite time to respond.
- PUT:
    - The user can enter a program name followed by source files which will be uploaded to the server inside the program name directory.
    - Adding -f will specify if those source files exist overwrite them.
- GET:
    - The user can enter a program name then a source file. This will return the contents of the source file.
    - The user will be able to see up to 40 lines at a time and the program will wait for any input before showing the next 40 lines.
- RUN:
    - The user can specify a program name followed by an undefined number of arguments. This will compile (if needed) and run the program using the given arguments.
    - Adding -w followed by [-f <local file>] or [<local file>] will write the return values to the local file given. If the file exists, then -f is needed to overwrite it.
- LIST:
    - Will list the programs (by their name) stored on the server.
    - Adding -l will also list other details such as size (directories will be 0), file modification date and access permission (eg 0777 for read, write, execute for owner, group and others).
    - Adding a program name will instead list the files inside that program.
- SYS:
    - Will return details about the server hardware and software. (Name, OS version and CPU type/number of cores).

Carl Humphries
s5084150

Software Requirements

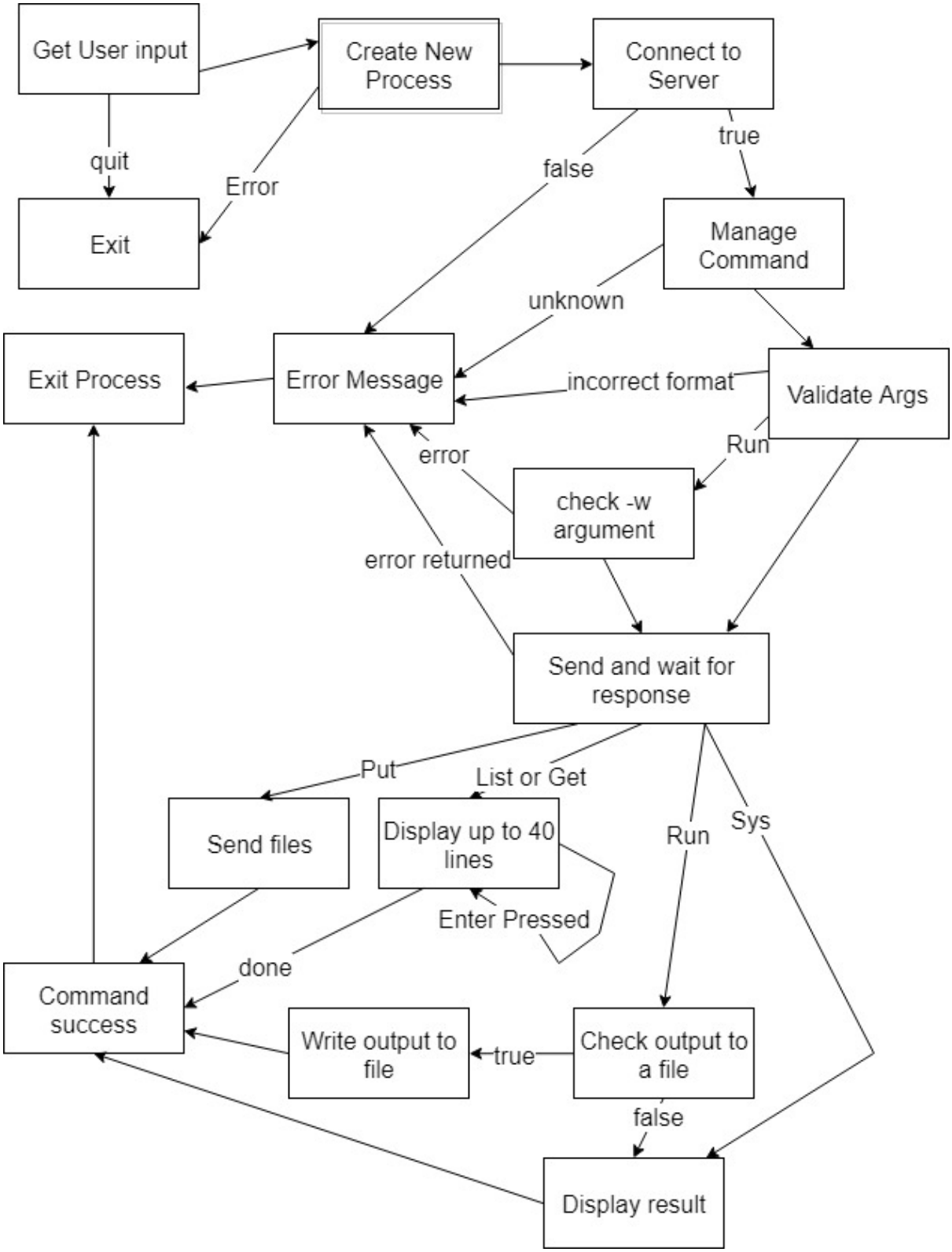
Below are the software requirements for the assignment:

1. The server is to be listening on port 80.
2. The client program takes the server IP as a command line argument.
3. The client program will loop waiting for the user to type a command which will be forwarded to the server which can respond at any time. Any response is immediately displayed.
4. The client program will report the time taken for the server to respond in milliseconds together with the server's response.
5. The client is non-blocking all requests are put onto another process and dealt with there. The one exception is that get and list commands need to block so there isn't any commands sent while the display isn't finished. (This could have been changed but for user flow it seemed better).
6. The server will spawn new processes and needs to be able to accept multiple clients and requests at any given time.
7. The server will be able to accept a program given as a program name and source files which it will create a directory called program name and store the source files inside it. It will also compile the program (if needed) and run it returning the result to the client.
8. The source code should be able to compile on both Unix and Windows.
9. The following commands are what the server will be able to recognise (taken from assignment sheet):
    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] [-w [-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.
10. The ***list*** command's -l optional argument is to signify that more than just the name of the file should be returned. It should also display the file size (0 if directory) the modification date and access permissions (eg 0777 for read, write, execute for owner, group and others).
11. The ***get*** command will display the contents of the file to the clients screen, displaying 40 lines at a time pausing waiting for the user to enter to continue.
12. The ***put*** command will create a new directory if it doesn't exist, otherwise it will check the source files to see if they exists making sure -f is specified or it will return an error. This command can have multiple file or just one uploaded. If -f is specified only the files sent will be overwritten.
13. If the ***run*** commands -w is given it signifies that the output should be put into a local file rather than printed to screen. It will check for a local file argument as well as -f, if the local file exists -f is required to overwrite it. If the local file exists and -f isn't specified it will return an error before sending the command to the server.
14. The run command will check to see if the program has been compiled and that none of the source files are newer than the executable. If this is the case the program will be [re]compiled. It will then run the compiled program passing in any given arguments from the command. If the command cannot be run or compiled an appropriate error message is sent back to the client.
15. If the server receives an incorrectly specified command it will return an error. If the server cannot run a known command it will generate an error message to be returned to the client.
16. All Zombie processes are terminated as required. The should not be any zombies on either the client or the server.

Software Design

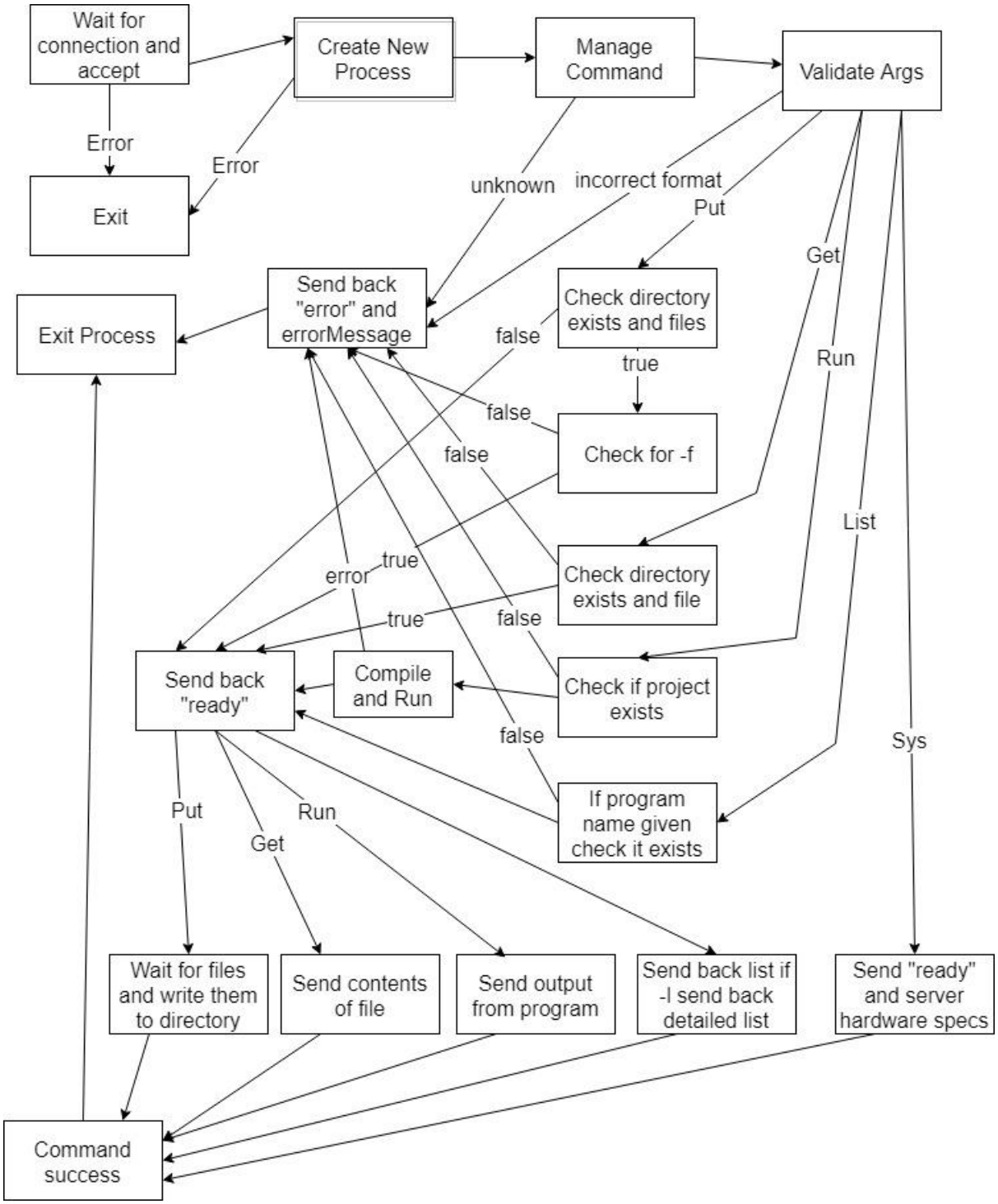
High Level Design - Logical Block Diagram

Client



Carl Humphries
s5084150

Server



NOTE: Refer to Server.pdf and Client.pdf for detailed documentation of programs.

Carl Humphries
s5084150

Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	The server is to be listening on port 80.	Partial	Pass	Windows cannot for security reasons. Uses 3000.
2	The client program takes the server IP as a command line argument.	Full	Pass	Note that if firewall blocks connection it will take a little bit of time to report this.
3	The client program will loop waiting for the user to type a command which will be forwarded to the server which can respond at any time. Any response is immediately displayed.	Full	Pass	
4	The client program will report the time taken for the server to respond in milliseconds together with the server's response.	Full	Pass	
5	The client is non-blocking all requests are put onto another process and dealt with there. The one exception is that get and list commands need to block so there isn't any commands sent while the display isn't finished. (This could have been changed but for user flow it seemed better).	Full	Pass	
6	The server will spawn new processes and needs to be able to accept multiple clients and requests at any given time.	Full	Pass	
7	The server will be able to accept a program given as a program name and source files which it will create a directory called program name and store the source files inside it. It will also compile the	Full	Pass	

	program (if needed) and run it returning the result to the client.			
8	The source code should be able to compile on both Unix and Windows.	Full	Pass	With the aid of winchild the windows compiled program works in the same way.
9	The following commands are what the server will be able to recognise (taken from assignment sheet): 1. ***put progname sourcefile[s] [-f]***: upload sourcefiles to progname dir, -f overwrite if exists. 2. ***get progname sourcefile***: download sourcefile from progname dir to client screen. 3. ***run progname [args] [-w [-f] [localfile]]***: compile (if req.) and run the executable (with args) and either print the return results to screen or given local file. 4. ***list [-l] [progname]***: list the prognames on the server or files in the given progname directory to the screen, –l = long list 5. ***sys***: return the name and version of the Operating System and CPU type.	1. Full 2. Full 3. Full 4. Full 5. Full	1. Pass 2. Pass 3. Pass 4. Pass 5. Pass	Refer to detailed testing to show what was tested.
10	The ***list*** command's -l optional argument is to signify that more than just the name of the file should be returned. It should also display the file size (0 if directory) the modification date and access permissions (eg 0777 for read, write, execute for owner, group and others).	Full	Pass	
11	The ***get*** command will display the contents of the file to the clients screen, displaying 40 lines at a time pausing waiting for the user to enter to continue.	Full	Pass	
12	The ***put*** command will create a new directory if it doesn't exist, otherwise it will check the source	Full	Pass	

	files to see if they exists making sure -f is specified or it will return an error. This command can have multiple file or just one uploaded. If -f is specified only the files sent will be overwritten.			
13	If the ***run*** commands -w is given it signifies that the output should be put into a local file rather than printed to screen. It will check for a local file argument as well as -f, if the local file exists -f is required to overwrite it. If the local file exists and -f isn't specified it will return an error before sending the command to the server.	Full	Pass	
14	The run command will check to see if the program has been compiled and that none of the source files are newer than the executable. If this is the case the program will be [re]compiled. It will then run the compiled program passing in any given arguments from the command. If the command cannot be run or compiled an appropriate error message is sent back to the client.	Full	Pass	
15	If the server receives an incorrectly specified command it will return an error. If the server cannot run a known command it will generate an error message to be returned to the client.	Full	~Pass	Note: since my client only send correct commands it was hard to check invalid, but the code does validate and make sure before it runs.
16	All Zombie processes are terminated as required. The should not be any zombies on either the client or the server.	Full	~Pass	Zombies were killed on unix with the given zombie kill function. But I wasn't sure if windows kept Zombies and online stated that once they exit they are removed anyway.

Detailed Software Testing

All tests were done with Windows server and two clients (one on mac one on windows) as well as Mac server and two clients on either.

No	Test	Expected Results	Actual Results.
1.0	**PUT**		
1.1 1.2	- put: without any arguments - put progname: without source files	Client doesn't send command and logs error "No source file specified"	As expected
1.3 1.4	- put progname sourcefile: - put programme sourcefile[s]:	a. File exists on client: I. File exists on server: ERROR from server cannot create no -f and file exists. II. File doesn't exist on server: puts file[s] onto the server in progname directory. b. File doesn't exist: File not found.	As expected
1.5 1.6	- put progname sourcefile -f: - put programme sourcefile[s] -f:	a. File exists on client: I. File exists on server: puts file[s] onto the server in progname directory. II. File doesn't exist on server: puts file[s] onto the server in progname directory. b. File doesn't exist: File not found.	As expected
1.7 1.8	- put -f progname sourcefile: with -f as any argument	Interprets -f as program name if in arg[0] and source file if any position other than the end.	As expected
1.9	- put progname -f	Sends no files	As expected

2.0	**GET**		
2.1 2.2 2.3	- get: no args - get sourcefile|progname: only one arg - get progname sourcefile blah: > 2 args	Client doesn't send command and shows ERROR: get [progname] [sourcefile]	As expected
2.4	- get sourcefile progname:	a. Program exists on server: I. File exists on server: display 40 lines at a time of files contents II. File doesn't exist on server: ERROR: Could not open file: sourcefile b. Program doesn't exist on server: ERROR: Could not open directory progname	As expected
3.0	**RUN**		
3.1	- run: no args	Command should not be sent and reported as error no prog to compile and run.	As expected
3.2	- run progname:	a. Program exists on server: I. Valid c program that can compile: Server should compile and i. Program will error: Server should return an error failed to run. ii. Program is valid: Server will return valid output from program II. Program can't compile: Server will return why it didn't compile including warnings.	As expected

		b. Program doesn't exist on server: Server will return ERRO: no project: progname	
3.3	- run progname arg[s]: upto ARG_MAX	Same flow from 3.2 but the program now should run with args if it compiles.	As expected
3.4	- run progname [args] -w	Will interpret -w as a program arg.	As expected
3.5	- run progname [args] -w localfile	a. Localfile exists: Wont send command to server "No -f cannot overwrite file." b. Localfile doesn't exists: Creates file locally and stores server return (if successful) into it.	As expected
3.6	- run progname [args] -w -f localfile	Creates file locally and stores server return (if successful) into it. Will overwrite any previous file.	As expected
3.7	- run progname -w -f [args] localfile: -w -f anywhere but [-w -f localfile] at the end	Will treat -w and -f as arguments for progname unless they are in the correct format.	As expected
4.0	**LIST**		
4.1	- list:	Server returns a list of all directories (program) on the server.	As expected
4.2	- list -l:	Server will return a list with more details described in requirements.	As expected
4.3	- list progname	a. Program exists on server: Server will return a list of files inside program b. No Program on server: Sever will return error No such program on the server. Progname.	As expected
4.4	- list -l progname: - list progname -l:	Combination of 4.2 and 4.3.	As expected
4.5	- list -l progname blah:	Client won't send command. Too many args	As expected

5.0	**SYS**		
5.1	- sys [blah[s]]: will ignore args anyway.	Will return specs of the server as described in requirements.	As expected
6.0	**OTHER**		
6.1	Nothing entered	Client won't do anything.	As expected
6.2	Multiple client using get and put	Should handle as per each command.	Tried on all machines to force an issue with heaps of request but nothing failed. This could be tested more but I though spamming different commands on different clients/servers would fail if anything was majorly wrong.
6.3	Disconnection in get	Client stops receiving and reports there was a disconnection.	As expected
6.4	- quit entered	Quits the client.	As expected
6.6	- unknown command entered	Client doesn't send command and reports unknown command	As expected

User Instructions

- Run $ Client <ip of server> <PORT: 80 (unix) 3000 (windows)>
- Enter one of the five commands.
    - **put progname sourcefile[s] [-f]**
    - **get progname sourcefile**
    - **run progname [args] [-f localfile]**
    - **list [-l] [progname]**
    - **sys**
- **Put:** will upload files in current directory to the server into progname directory.
- **Get:** will return file contents 40 lines at a time.
- **Run:** will compile and run a program uploaded to the server. Displaying results or writing them to file.
- **List:** will display details of program or files inside a program.
- **Sys:** will display details of the server hardwar

Carl Humphries
s5084150