2803ICT Assignment 1

Due Thursday 30th August,
Worth 20% of final grade

You must complete either option A OR option B

OPTION A

Objectives:

The objective of this assignment is to write a simple remote execution system. This requires putting into practice what has been taught about systems programming.

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 source code shall be portable so that it can be compiled and run on both Unix and Windows.
9. 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.
10. 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.
11. 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.
12. 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

13. 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.
14. 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'.
15. 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.
16. All **Zombie** processes are terminated as required. There is to be no unwanted Zombie processes on either the client, or the server.

Submission:

- Source code plus statically linked Win32 and Cygwin executables uploaded to L@G (include all makefiles, project files, project subdirectories; except the object files).
- Software documentation must be submitted according to the sample documentation that is available on learning@griffith. Please use the sample as a template and change the contents of each section to reflect your assignment.
- Software documentation must include compilation instructions for both Unix and Win platforms (compilers used, dependencies, Visual Studio Version, etc).

Marking Scheme:

Requirement	Marks
2. Server IP address as a command line argument	4
4. Client reports time taken for server responses	4
5. Client runs continuously in a loop but is non-blocking - can have outstanding queries	8
6a. Server spawns new process for new requests on Win32 and Unix	8
6b. Server accepts multiple simultaneous requests and from multiple clients	8
8. Compiles and runs on both Win32 and Unix	4
9a+12.Correct operation of put command with -f option	4
9b. Correct server operation of get command	4
9c1+14. Correct server operation of run command - file compilation on windows and unix	4
9c2+14. Correct server operation of run command - execute with params	4
9c3+14. Correct operation of run command - returns results to client on windows and unix	8
9d+10. Correct operation of list command with -l option	4
9e. Correct operation of sys command on Windows and Unix	4
11. Client: scrolling list that pauses after 40 lines on list and get command	4
16. Server error handling on Win32 and Unix	4
18. Zombie removal	4
19. Documentation	10
20. Code Style – Completeness – Robust - Quality	10

OPTION B

Objectives:

The objective of this assignment is to write a simple client server system for querying the status of remote servers. This requires putting into practice what has been taught about systems programming.

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 handle each new request and be able to accept multiple clients.
7. The source code shall be portable so that it can be compiled and run on Unix and Windows.
8. The following query commands with options are recognised by the server (anything within [] is optional):
   A. list [-l] [-f] [pathname] [localfile] : list the files in the current or given directory to the screen or print them to a file. Options: –l = long list, -f force overwrite
   B. get filepath [-f] [localfile] : print the content of a nominated file to screen or given file.
   C. put localfile [-f] [newname] : create a remote copy of a local file with same (or other) name.
   D. sys : return the name and version of the Operating System and CPU type.
   E. delay integer : returns the given integer after a delay of 'integer' seconds.
9. The long list option of the ***list*** command will also return the file size, owner, creation date and access permissions. If no pathname is given, then the contents of the current directly will be returned. The list command will either dump the directory listing to the screen 40 lines at a time and pause waiting for a key to be pressed before displaying the next 40 lines etc, or it will save the listing to a local file if given. If local file already exists an error will result.
10. The ***put*** command will create a remote file with the same name or a new name is one is specified. If the remote filename exists, the server will return an error.
11. The –f option will force an existing file to be overwritten if it exists and advise the user whether a file was overwritten or not.
12. If no filename argument is given, the get command will dump the contents of the remote file to the screen 40 lines at a time and pause waiting for a key to be pressed before displaying the next 40 lines etc. Otherwise it will create a new local file with the given filename. If a file with that name already exists, the client will return an error before sending the get request to the server.
13. All commands shall accept both relative and absolute path names for the local and remote paths.
14. The ***delay*** command will cause the server to sleep for the specified interval for the given request without affecting any other requests being processed.
15. The client shall be able to redirect output to other processes using the '|' argument. The client calls the nominated process passing to it the data returned from the server and then prints out the result to the screen. You must not use the system() call.
    - get remotefile | grep name

- get remotefile | findstr name
- list | sort

16. 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

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

Submission:

- Source code plus statically linked Win32 and Cygwin executables uploaded to L@G (include all makefiles, project files, project subdirectories; except the object files).
- Software documentation must be submitted according to the sample documentation that is available on learning@griffith. Please use the sample as a template and change the contents of each section to reflect your assignment.
- Software documentation must include compilation instructions for both Unix and Win platforms (compilers used, dependencies, Visual Studio Version, etc).

Marking Scheme:

Requirement	Marks
2. Server address as a command line argument	4
4. Client reports time taken for server responses	4
5. Client runs continuously in a loop but is non-blocking	4
6a. Server spawns new process for new requests on Win32	4
6b. Server spawns new process for new requests on Unix	4
7. Compiles and runs on both Win32 and Unix	4
8a1. Correct operation of list –l command ox Unix	4
8a2. Correct operation of list –l command on Windows	4
8b. Correct operation of get command file data correctly copied to screen or file as needed	4
8c. Correct operation of put command file data correctly copied	4
8d1. Correct operation of sys command on Unix	4
8d2. Correct operation of sys command on Windows	4
9+12. scrolling list & get that pauses after 40 lines	4
10. Put command - create remote file with same name or specified name or returns error	4
11. Correct operation of the –f option on list, get and put commands	4
13. Commands accept relative and absolute path names for local and remote paths	4
14. Correct operation of delay command	2
15a. Correct operation of output redirection on Win32	4
15b. Correct operation of output redirection on Unix	4
16. Server error handling on Win32 and Unix	4
18. Zombie removal	2
19. Documentation	10
20. Code Style – Completeness – Robust - Quality	10