Telnet Assignment

Sebastian Schmidt

Contents

1. Problem Statement

The purpose for the C program is to create a simple server and client over TCP. The client can request, upload and list files via the command line. In addition the client can ask for the system information and any output can be stored to a local file. The server can accept multiple clients and is non blocking, that is a client can delay a request and still execute commands.

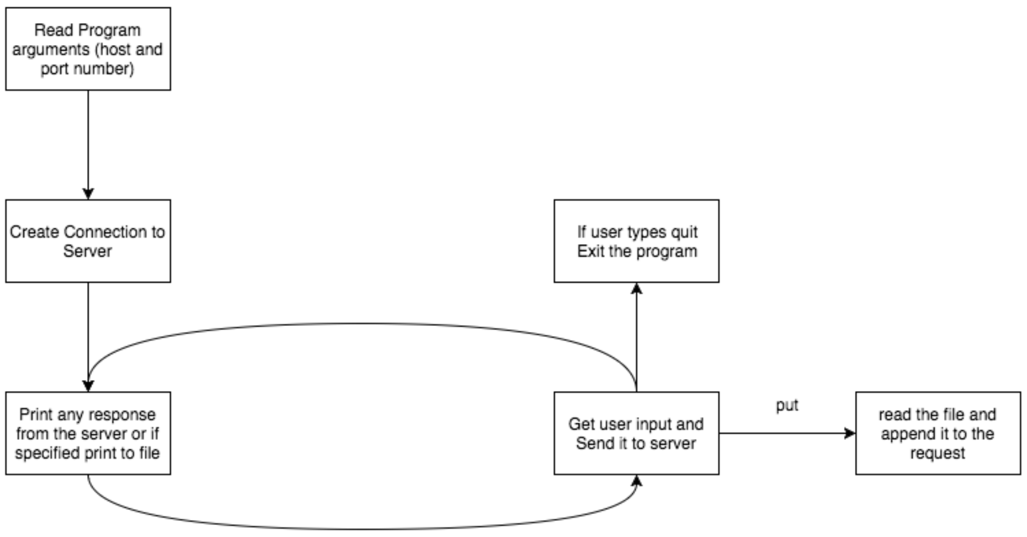
2. Software Requirements

1. Using a socket connection, a client program will query a server remote server listening on port 80.
2. The address of the server to be queried by the client shall be given as a command line argument.
3. The operation of the client once it is run is to wait for the user to enter queries, 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 Visual Studio.
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 /version of the Operating System and CPU type
   E. delay integer - returns the given integer after a delay of 'integer' seconds.
9. The long listing option of the list commands 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 or not a file was overwritten.
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 requestes being processed.
15. The client shall be able to redirect output to other processes using the '|' argument. What this does is that 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   > list | sort

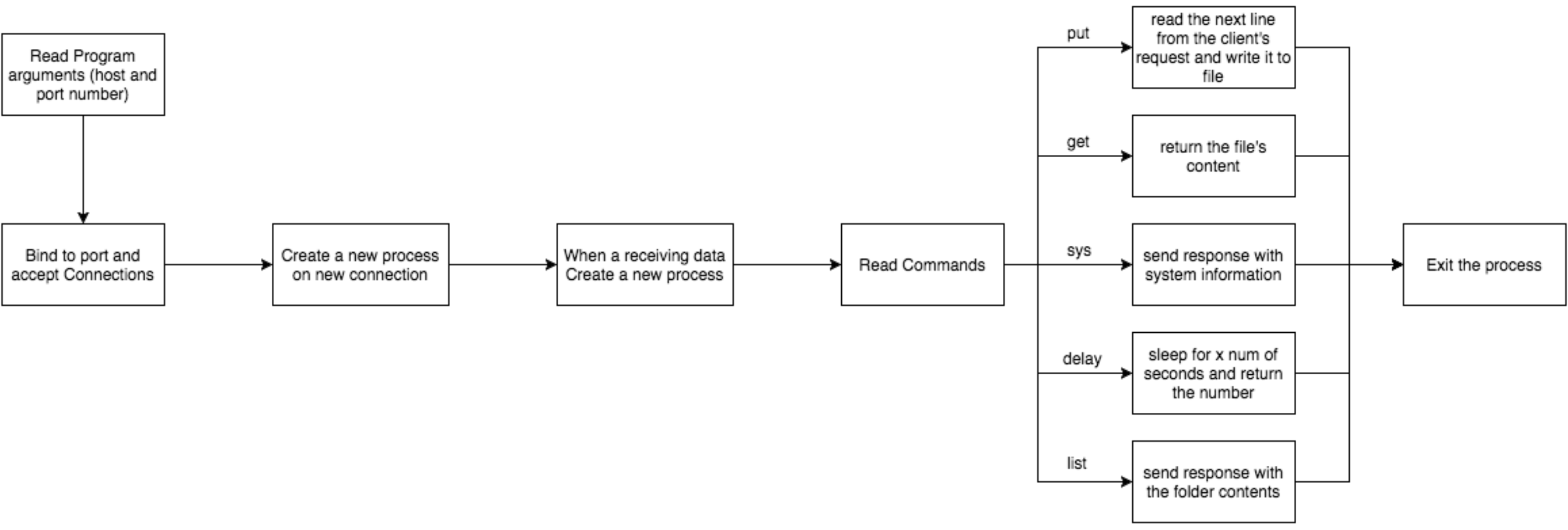
16. If the server receives an incorrectly specified command it will return an error. If the server is unable to execute a valid command the server will return the error string generated by the operating system to the client
17. All Zombies are removed when the server shuts down.

3. Software Design

Client


Read Program arguments (host and port number)
Create Connection to Server
If user types quit Exit the program
Print any response from the server or if specified print to file
Get user input and Send it to server
put
read the file and append it to the request

Server


Read Program arguments (host and port number)
Bind to port and accept Connections
Create a new process on new connection
When a receiving data Create a new process
Read Commands
put
read the next line from the client's request and write it to file
get
return the file's content
sys
send response with system information
delay
sleep for x num of seconds and return the number
list
send response with the folder contents
Exit the process

List of all functions in the software.

lib.c

- int strLen(char *string);
  my own strlen() function. It returns the length of string.

- void removeNewLine(char* inStr);
  removes the '\n' character in inStr if '\n' is the last character in the string.

- char* nextLine(char* inStr);
  returns a pointer to the next line in a string by searching for '\n'.

- char* splitNextLine(char* inStr);
  similar to nextLine except it puts in a null character where the '\n' was.

- void strToLower(char *string);
  converts the string to lowercase using tolower(); on every character.
- int splitStr(char* srcString, char* tokens[], int maxTokens);
  returns the number of found tokens. The functions places a null character at every found space and adds a pointer to the tokens array. This can be uses to read the srcString like an array of arguments. maxTokens limits how many tokens (spaces) should be searched for.

- struct timespec getTime();
  gets the current time from the system clock and returns it as a timespec struct.

- float getTimeLapsed(struct timespec start);
  returns the amount of time that has passed between start and now in nanoseconds.

- float getTimeNsec();
  gets the current time from the system clock in nanoseconds

- void printClient(char *message);
  adds blue colour to the message and prints it to stdout.

- void printServer(char *message);
  adds cyan colour to the message and prints it to stdout.

- int saveToFile(char* to, char* bytes, int force, String* out);
  saves bytes to disk. 'to' is the filename. The force flag determines if the file can be overwritten without asking. the out parameter is optional, it is used to relay any messages back to the client. returns status code if there was an error or 0 when successful

- int readFile(char* filepath, String* buffer);

read a file specified in the filepath variable from disk. The read data is returned in the buffer. returns a status code if there was an error or 0 when successful

- int copy(char *from, char *to);
  copy a file from a file path to a file path. returns a status code if there was an error or 0 when successful

../../01StringAndList/String.c

Is a class full of string functions…

- void strInit(String* self);
  initialise String and allocates memory.

- void strCInit(String* self, char* init);
  initialises the String, allocates memory and copies ‘init’ into String

- int strLength(String* self);
  returns the length of the String

- void strAddChar(String* self, char c);
  concatenates ‘c’ to the String

- void strConcatC(String* self, char c);
  alias of strAddChar();
- void strConcat(String* self, String* input);
  concatenates ‘input to ‘self’

- void strConcatCS(String* self, char* input);
  concatenates ‘input to ‘self’

- void strOverrideCS(String* self, char* input);
  overrides String with the contents of ‘input’

- void strConcatI(String* self, int input);
  concatenates ‘input to ‘self’

- void strConcatF(String* self, float input);
  concatenates ‘input to ‘self’

- void strPrint(String* self);
  prints the String to standard output

- int strParseInt(String* self);
  returns AKSII converted String.

- void strClean(String* self);
  fills the String with null charters and sets the length to 0

- void strFree(String* self);
  frees the memory

- void strResize(String* self, int newSize);
  increases the buffer size of the String

client/main.c

- int getCommand(char *request, int *fflag, char *filepath, String* filedata);
  This function checks the user input if for example a file needs to be read and added to the request that is send to the server. The function also checks if, in the next iteration of the main() loop, the output should be saved to a file. In addition the function checks and sets the fflag argument if the user typed -f as an argument. the request is the original buffer from the user input. The remaining (fflag, filepath and filedata) arguments are return values. The file path can contain the path to save the output from the server. The file data can contain the data of a file that was read. Basically the function does some preprocessing. getCommand returns a status code if there was an error or 0 when successful

- int main(int argc, char *argv[]);
  argc and argv are used to read in the port number and the hostname where the client should connect to. Once a connection is established a loop reads data for the server and prints the output to stout or a file. The loop is non blocking and switches between reading from standard input and the socket about every half a second. main returns a status code if there was an error or 0 when successful

server/main.c

- void childProcessExit (int sig);
  When a child process exits this function waits and prints the process id with a message to standard out. sig is not used.

- void terminate (int sig);
  When a the parent process exits this function waits for all child processes and prints the process id with a message to standard out. sig is not used.

- void console(char *request, String *response);
  This function splits the request with splitStr and determines which command the client wants to execute by comparing strings. The relevant function is executed and variables are passed along, or if none match an error is returned by concatenating a string to the response.

- int newProcess (Socket com);
  This function create a new process and reads data from the client in an inflate loop. If dat from the client is read second fork creates another process to hand the request. the request is passed onto the console function. The function keeps track of child process and waits before exiting it self;

com is the Socket the client is connected to. newProcess returns a status code (-1 on error) process id or 0 on sucesfull exit.

- int main(int argc, char *argv[]);

  argc and argv are used to read in the port number where the server should listen for connections. Ports below 1024 require sudo. once the port number is known main tries to bind to it and start accepting connections in an infinite loop. As soon as a new connection is established it is passed onto newProcess(). if newProcess returns a error status code and a message is returned to the client saying something is wrong.

server/cat.c

- int __get(const char* filePath, String *out);

  reads the file from disk at filePath and sends the data to the client by concatenating to the out variable. __get returns a error status code and a message is returned to the client if the status code is not 0. 0 indicates successful execution.

- int get(int argc, char *argv[], String* out);

  The function reads in the arguments and passes them to __get(); If invalid arguments are found than a message is send back to the client using the out variable. get() returns the return value from __get();

server/cp.c

- int put(int argc, char *argv[], char* fileData, String* response);

  The put function writes the fileData to hard disk. put returns a error status code and a message is returned to the client if the status code is not 0. 0 indicates successful execution. saveToFile() is used.

server/ls.c

- int __list(const char path[], int lflag, String* out);

  This function lists the files for the specified path. The lflag indicates if the long list should be returned. the out variable is used to return the output of the function back to the client. __list() returns a error status code and a message is returned to the client if the status code is not 0. 0 indicates successful execution.

- int list(int argc, char *argv[], String* out);

  this function gets the arguments and executes __list(). out is used to relay any error messages back to the client. list() returns the return value from __list();

server/sleep.c

- int delay (int seconds, String *out)

  this function sleeps ‘seconds’ seconds and one awake the number of seconds is rather retired via concatenation of out if it is not NULL or via standard output. the return value for the function is the number of ‘seconds’ slept.

server/uname.c

- int sys(String* out);

  sys returns the system information from uname(); out is a optional argument and if it defined it is used to pass the system information to the client as a string. If out is NULL the information is printed to standard output. the function returns errno on an error or 0 on successful execution.

4. Requirement Acceptance Tests

(You will need to fill out the column on the left with the requirements listed in sections 2 and 3 and the columns on the right with the results of your own testing)

No	Test	Implemented (Full / Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	Server address as a command line argument	Full	Pass	
2	Client reports time taken for server responses	Full	Pass	
3	Client runs continuously in a loop but is non-blocking	Full	Pass	
4	Server spawns new process for new requests one Win32	None		program can't compile on windows
5	Server spawns new process for new requests on Unix	Full	Pass	
7	Correct operation of list -l command on Windows	None		program can't compile on windows
8	Correct operation of get command file data correctly copied to screen or file as needed	Full	Pass	
9	Correct operation of put command file data correctly copied	Full	Pass	
10	Correct operation of sys command on Unix	Full	Pass	
11	Correct operation of sys command on Windows	None		program can't compile on windows
12	Scrolling list & get that pauses after 40 lines	None		Ran out of time
13	Put command - create remote file with same name or specified name or returns error	Full	Pass	
14	Correct operation of the -f option on list, get and put commands	Full	Pass	

No	Test	Implemented (Full / Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
15	Commands accept relative and absolute path names for local and remote paths	Full	Pass	
16	Commands accept relative and absolute path names for local and remote paths	Full	Pass	
17	Correct operation of output redirection on Win32	None		program can’t compile on windows
18	Correct operation of output redirection on Unix	None		Ran out of time
19	Server error handling on Win32 and Unix	Partial	Pass	only on Unix
20	Zombie removal	Full	Pass	

5. User Instructions

Please see the Readme.md file