Software Specification
&
Assignment Instructions

Miniature File Transfer Program (mftp)

Version 1.8

CS 360

Spring 2012
WSU Vancouver

1. General Requirements
    a. The miniature ftp system consists of two programs (executables), mftpserve and mftp, the server and client respectively.
    b. Both programs will be written in C, to execute in the Linux OS environment.
    c. The programs will utilize the Unix TCP/IP sockets interface (library) to communicate with each other across a TCP/IP network. Only IPv4 addressing will be supported by these programs.
2. User Interface
    a. Command line
        i. The user initiates the client program using its name (mftp) with exactly one command line argument (in addition to the name of the program). This argument will be interpreted by the client program as the name of the host on which the mftpserve process is running and to which the client will connect. The hostname may be either a symbolic host name or IPv4 dotted-decimal notation. Optional debug related flags may be defined for the command line but must not interfere with the user's ability to specify the name of the remote host.
    b. Login
        i. The server process will <u>not</u> require a login dialog from the client. The server process will retain the identity and privileges of the user that initiated it and these privileges, identity and initial working directory are the context in which client commands are executed by the server.
    c. User commands
        i. The client program will prompt the user for commands from standard input. Commands are regarded as a series of tokens separated by one or more white-space characters and terminated by a new line character. The first token is the name of the command. Successive tokens are command arguments, if any. Note: the first token may be preceded by spaces. White-space characters are

defined to be any character for which `isspace()` returns true. Commands and their arguments are case-sensitive.

ii. The user commands are:

1. exit
    a. terminate the client program after instructing the server's child process to do the same.
2. cd <pathname>
    a. change the current directory of the client process to <pathname>. It is an error if <pathname> does not exist, or is not a readable directory.
3. rcd <pathname>
    a. change the current directory of the server process to <pathname>. It is an error if <pathname> does not exist, or is not a readable directory.
4. ls
    a. execute the "ls –l" command locally and display the output to standard output, 20 lines at a time, waiting for a carriage return on standard input before displaying the next 20 lines. See "rls" and "show".
5. rls
    a. execute the ls command remotely and display the output to the client's standard output, 20 lines at a time.
6. get <pathname>
    a. retrieve <pathname> from the server and store it locally in the client's current working directory using the last component of the pathname as the file name. It is an error if <pathname> does not exist or is anything except a regular file.
7. show <pathname>
    a. retrieve the contents of the indicated remote <pathname> and write it to the client's standard output, 20 lines at a time. Wait for the user to type a carriage return, before typing out the next 20 lines. The "more" command can be used to implement this feature. It is an error if <pathname> does not exist or is anything except a regular file.
8. put <pathname>
    a. transmit the contents of the local file <pathname> to the server and store the contents in the server process' current working directory using the last component of the pathname as the file name. It is an error if <pathname> does not exist locally or is anything except a regular file.

d. The client program should report command errors and operational errors in a manner that is clear and understandable to the user. If the error is recoverable, the command is aborted and a new command prompt is issued. If the error is not recoverable, the client should terminate.

3. Server
    a. The server process, mftpserv, will be initiated with no command line arguments other than flags related to debugging. The server will log its activity to standard output.
4. Server/Client Interface Protocol
    a. Initiation
        i. The mftpserve process shall listen on ephemeral TCP port number 49999 for client connections. It shall permit a queue of 4 connection requests simultaneously.
        ii. The client program (mftp), prior to prompting the user for commands shall establish a connection to the server process on TCP port 49999. If the connection fails, the client will issue and error message and abort.
        iii. When a connection is established, an appropriate success message is displayed to the user by the client and logged by the server. This connection is regarded as the control connection. A second connection, the data connection, is required by certain other commands. The data connection will be established when needed and will be closed at the end of the associated data transfer for each command that establishes it.
        iv. Once the control connection is established, the server process (child) will attempt to read commands from the control connection, execute those commands as described below, and then read another command, until a "Q" command is received, or until an EOF is received, when the server child will terminate.
        v. Server commands have the following syntax:
            1. No leading spaces.
            2. A single character command specifier.
            3. An optional parameter beginning with the second character of the command (the character following the single character command specifier) and ending with the character before the terminator.
            4. The command terminator is either a new line character or EOF.
        vi. Server response syntax:
            1. The server responds once to each command it reads from the control connection.
            2. The response is either an error response or an acknowledgement.
            3. All responses are terminated by a new line character.
            4. An error response begins with the character "E". The intervening characters between the "E" and the new line character comprise a text error message for display to the client's user (written to standard output).

5. An acknowledgement begins with the character "A". It may optionally contain an ASCII coded decimal integer between the "A" and the new line character if required by the server command (only the "D" command below requires an integer in the acknowledgement).

vii. The server commands are:

1. "D" Establish the data connection. The server acknowledges the command on the control connection by sending an ASCII coded decimal integer, representing the port number it will listen on, followed by a new line character.
2. "C<pathname>" Change directory. The server responds by executing a chdir() on the specified path and transmitting an acknowledgement to the client. If an error occurs, an error message is transmitted to the client.
3. "L" List directory. The server responds by initiating a child process to execute the "ls" command. It is an error to for the client to issue this command unless a data connection has been established. Any output associated with this command is transmitted along the data connection to the client. The server child process waits for its child (ls) to exit, at which point the data connection is closed. Following the termination of the child (ls) process, the server reads the control connection for additional commands.
4. "G<pathname>" Get a file. It is an error to for the client to issue this command unless a data connection has been established. If the file cannot be opened or read by the server, an error response is generated. If there is no error, the server transmits the contents of <pathname> to the client over the data connection and closes the data connection when the last byte has been sent.
5. "P<pathname>" Put a file. It is an error to for the client to issue this command unless a data connection has been established. The server attempts to open the last component of <pathname> for writing. It is an error if the file already exists or cannot be opened. The server then reads data from the data connection and writes it into the opened file. The file is closed and the command is complete when the server reads an EOF from the data connection, implying that the client has completed the transfer and closed the data connection.
6. "Q" Quit. This command causes the server (child) process to exit normally. Before exiting, the server will acknowledge the command to the client.

        viii. Each command is logged by the server, along with the arguments and its success or failure, to standard output.

5. Suggestions
    a. Establishing a data connection.
        i. In response to a "D" command, the server child process will create a new socket using `socket()` and then use `bind()` to "give it a name". The family, port, address structure given to bind, should use a port number of zero, which is a wildcard for any port. Following the `bind()` call, call `getsockname()` to get back a structure in which the ephemeral port assigned by the kernel will be found. You will need to pass `getsockname()` an empty (zeroed) servaddr structure and a pointer to an integer containing its length. Upon return from `getsockname()`, servaddr.sin_port will contain the port number assigned by bind(). However, you will need to use ntohs() when storing it into an int so that its byte order is correct. This is the port number you will send in the command acknowledgement to the client. The client will then make a connection using `socket()` and `connect()` to that port on the server while the server listens (calls `accept()`) for waiting for the connection to be made.
    b. Consider using `strtok()` to parse the client's user commands.
    c. Consider defining a debug flag for both your programs such that when in debug mode, your programs print out all the details of what they are doing. If you want to be elegant about it, look for argv[1] to be "-d" to turn on debug mode (of course, in that case, the client will find the hostname as argv[2]).
    d. Test for errors in user input, command input and from system calls and give sensible error messages to the user.
    e. Test your client and server against the client and server programs of your classmates, they should all be interoperable if everyone follows this specification. Be careful not to clobber each other's files when you do so, these programs offer no protections.
    f. You may also run my implementations of these programs (which reside in the directory `/encs/cs/class/cs360/lab/net2`) and test your programs against them. Both of my programs will produce extensive debug output if you make the first command line argument "-d".
6. Completion of this assignment
    a. Organize your code into (at least) three files:
        i. mftp.c – client code
        ii. mftpserve.c – server code
        iii. mftp.h – header file with declarations common to both programs
    b. When you believe your programs are completed, email them to me as an attached tarball.
    c. Schedule time with me to demonstrate your programs in the lab. Give me at least 3 hours advance notice that you are ready to demonstrate.

d. Submission <u>and</u> demonstration of the programs are due by 5:00PM PDT, Thursday, May 3, 2012.