TCP Client/Server
Assignment Description

We implement two programs, a tserver and a tclient. The server (i.e., tserver) creates a stream socket (i.e., TCP) in the Internet domain bound to a port number (specified as a commandline argument when you start your server), receives requests from a client (i.e., tclient), produces results based on the content of requests, and sends the results back to the client on the same connection.

For this exercise, there are three types of messages you must implement:

- a request to get the file type of a file on the server (FILETYPE_REQ)
- a request to get the checksum of a file on the server (CHECKSUM_REQ)
- a request to download a file from the server (DOWNLOAD_REQ)

All messages have the same structure: a 1-byte MessageType field, a 4-byte (unsigned 32-bit integer) DataLength field, and a variable-length data field.

Byte Pos	Name	Description
0	MessageType	0xea (FILETYPE_REQ): file-type request 0xe9 (FILETYPE_RSP): successful file-type response 0xe8 (FILETYPE_ERR): failed file-type response 0xca (CHECKSUM_REQ): file checksum request 0xc9 (CHECKSUM_RSP): successful checksum response 0xc8 (CHECKSUM_ERR): failed checksum response 0xaa (DOWNLOAD_REQ): download file request 0xa9 (DOWNLOAD_RSP): successful download response 0xa8 (DOWNLOAD_ERR): failed download response 0x51 (UNKNOWN_FAIL): catch-all failure response
1-4	DataLength	Length of the Data field below (in network byte order).
5+	Data	Binary data byte stream. Meaning of this field depends on the MessageType field.

The server is normally in a sleeping state. When it receives a message from a client, it expects the message to be in the format described above. Here is the what the server does when it gets a message of a certain type:

FILETYPE_REQ : The Data field is a filename (not null-terminated). The server calls popen() to execute a "file" command with "filename" as the argument (i.e., as if the command, "/usr/ucb/file filename" is executed on the server machine), reads the output of the command, converts each tab character to a space character, and sends it back to the client in a FILETYPE_RSP message. If filename is invalid, the server sends back a 5-byte long FILETYPE_ERR message.

CHECKSUM_REQ : The Data field is a 4-byte offset, followed by a 4-byte length, followed by a filename (not null-terminated). The server computes a MD5 checksum for the data bytes of the specified file in the specified range and send the checksum value back to the client in a CHECKSUM_RSP message. If the message is too short, or the offset, the length, or the filename part of the Data is invalid or if it cannot open the specified file, the server sends back a 5-byte long CHECKSUM_ERR message.

DOWNLOAD_REQ : The Data field is a 4-byte offset, followed by a 4-byte length, followed by a filename (not null-terminated). The server sends the data bytes of the specified file in the specified range back to the client in a DOWNLOAD_RSP message. If the message is too short, or the offset, the length, or the filename part of the Data is invalid or if it cannot open the specified file, the server sends back a 5-byte long DOWNLOAD_ERR message.

? : If the server receives anything unrecognizable, it send back a 5-byte long UNKNOWN_FAIL message.

Here are some additional requirements:

filename : If a filename is specified in a message, it must not be empty and it must contain only valid characters. Valid characters in a filename include numbers, uppercase or lowercase letters, dashes and underscores, periods and commas, and the plus symbol. All other characters are invalid (including the space character, the tab character, international character, etc.)

offset : An offset is a 32-bit unsigned integer specifed in the network byte order format. The value of offset must be ≥ 0 and ≤ 2,147,483,647 (or 0x7fffffff in hex) . For a given file, if offset is larger or equal to the size of the file, it is considered an invalid offset.

length : A length is a 32-bit integer (signed) specifed in the network byte order format. If the value of length is negative, it means that you must send all the remaining data in the specified file starting with the specified offset. For a given file, if length is non-negative and offset+length is larger than the size of the file (equal is okay), it is considered an invalid length.

Commandline Syntax & Program Output

The commandline syntax for the tserver and tclient is given below. The syntax is:

tserver [-d] [-t seconds] port

tclient [hostname:]port filetype filename
tclient [hostname:]port checksum [-o offset] [-l length] filename
tclient [hostname:]port download [-o offset] [-l length] filename [saveasfilename]

Square bracketed items are optional. We follow the UNIX convention that commandline options can come in any order. Output of our program goes to stdout and error messages goes to stderr.

For tserver, if the -t commandline option is specified, seconds is the number of seconds it takes for the tserver to auto- shutdown and it must be ≥ 5. If -t is not specified, your server must auto-shutdown 300 seconds after it starts. If the -d commandline option is specified, it puts the server in debug mode. In the debug mode, the server must print the content of every message it received and sent to stdout.

For tclient, if a hostname is not specified, we connect to "localhost". If -o is not specified, we use an offset of zero. If -l is not specified, we use a length of (-1), which is 0xffffffff for a 32-bit integer. If -o is specified, offset must be > 0. If -l is specified, length must be > 0.

The requests should have the message structure described above. tclient sends one request to the server, waits for a response, prints the result, and terminates itself.

For every response tclient receives, it prints one line of output. Here are the information about the required output:

FILETYPE_RSP : Check every byte in the Data field and make sure that every byte is ≤ 0x7f. If it is, you must print the entire Data field in one line to stdout. If it is not, you must print "Invalid characters detected in a FILETYPE_RSP message.\n" to stdout.

CHECKSUM_RSP : Since an MD5 checksum must be exactly 16 bytes long, if the DataLength field of the message is not 16, you must print "Invalid DataLength detected in a CHECKSUM_RSP message.\n" to stdout. Otherwise, you must print the hexstring representation of the Data field in one line to stdout. In this case, your output must contain exactly 32 hex characters followed by a "\n".

DOWNLOAD_RSP : After we have read the first 5 of this message and DataLength is > 0, you must save the Data part of the message into a file. If saveasfilename is specified in the commandline, we check to see if the file already exists. If the file does not exists, we write the Data part of the message into the file specified by saveasfilename.

If the file specified by saveasfilename already exists, you must print "File saveasfilename already exists, would you like to overwrite it? [yes/no](n) " (and replace saveasfilename with the last commandline argument). We will not print a "\n" and put the cursor right after your prompt to the user. The "(n)" in the message means that the default answer is "no". If the first letter of the
user's response is "y", we overwrite the existing file. If the first letter of the
user's response is anything else, we print "Download canceled per user's request.\n".

If saveasfilename is not specified in the commandline, the file we need to create will have the same name as what we are downloading, i.e., you need to get it from filename. Since filename may be a file system path, you must search for the last "/" character and what comes after that is
the file name you need to use. In this case, the file we need to create must be created in the current working directory (which is the output of the Unix command "pwd"). The rest of the logic is the same as the logic above for saving a file into a user-specified file name.

When we have successfully write Data into FILE, you must print "Downloaded data have been successfully written into 'FILE' (MD5=...)\n" where FILE is either saveasfilename (when applicable) or the actual file name you wrote into and ... is the MD5 checksum of Data in hexstring format.

? : If the client receives anything else, it must print "Unexpected message of type MSGTYPE received.\n", and you must replace MSGTYPE with the name of the MessageType if it's recognized. If the MessageType is not recognized, you must replace MSGTYPE with the hexstring representation of the MessageType you received.

Server Debug Mode

When we put the server in the debug mode, we must print the content of every message it received and sent to stdout. Below is a table of what the server should print to stdout when it recevied a particular type of message:

FILETYPE_REQ	: FILETYPE_REQ received with DataLength = ???, Data =	'...'\n
CHECKSUM_REQ	: CHECKSUM_REQ received with DataLength = ???, offset	= ???, length = ???, filename = '...'\n
DOWNLOAD_REQ	: DOWNLOAD_REQ received with DataLength = ???, offset	= ???, length = ???, filename = '...'\n
?	: Message with MessageType = 0x?? received. Ignored.\n	

Please replace "???" and "..." with the information related to the message you sent.

Below is a table of what the server should print to stdout when it sends a particular type of message:

FILETYPE_RSP	:	FILETYPE_RSP sent with DataLength = ???, Data = '...'\n
FILETYPE_ERR	:	FILETYPE_ERR sent with DataLength = ???\n
CHECKSUM_RSP	:	CHECKSUM_RSP sent with DataLength = ???, checksum = ...\n
CHECKSUM_ERR	:	CHECKSUM_ERR sent with DataLength = ???\n
DOWNLOAD_RSP	:	DOWNLOAD_RSP sent with DataLength = ???\n
DOWNLOAD_ERR	:	DOWNLOAD_ERR sent with DataLength = ???\n
UNKNOWN_FAIL	:	UNKNOWN_FAIL sent with DataLength = ???\n