Dwight Thomas

@02765585

Data Communication

Socket Programming Assignment UDP

**The Application Protocol:**

**Architecture**: Client/Server

**Protocol:**

* Message Types:

Request, Data, ACK, Confirmation Response, Failure Response

* Message Syntax:

**Request** – [Head],

**[Head] 🡪**

file path <= Max 30 bytes of ASCII ended by a space character

format to = 1 byte of ASCII ended by a space character

target <= 8 bytes of ASCII ended but the null character

**Data - [**Body]

**[Body] 🡪**

**Type 0:**

Type = 1 byte of binary characters (0 in this case) ended by space character

Amount = 1 byte of binary characters ended by space character

N number of Numbers = N 2-byte binary characters, each delimited by a space and ended with n/ character.

**Type 1:**

Type = 1 byte of binary characters (1 in this case) ended by space character

Amount = 3 bytes of ASCII characters

N number of Numbers = N unsigned integers (16bit unsigned binary) separated by a comma followed by a space and ended with a space with no preceding comma.

**ACK –** [Body]

**[Body] 🡪**

6 bytes of ASCII character ended by a null character.

**Confirmation Response** – [Body]

**[Body] 🡪**

6 bytes of ASCII character ended by a null character.

**Failure Response** – [Body]

**[Body] 🡪** = 12 bytes of ASCII characters with a space after 6 and ended by a null character

* Message Semantics:

**Request** – [Head]

**[Head]** 🡪

file path: the string that represents the path to the file being sent, and name of file

format to: a number that represents which format changes need to be done

target: the string that represents the name of the file to save the newly formatted units

**Data** - [Body]

**[Body] 🡪**

A line of strings separated by spaces or spaces and commas that are the type, the amount of numbers, and the numbers

**ACK – [**Body]

**[Body] 🡪**

A string

**Confirmation Response** – [Body]

**[Body] 🡪**

A string with the message success

**Failure Response** – [Body]

**[Body] 🡪**

A string with the message format error.

* Rules:

Requests are sent by the client only.

Ack are sent by the server.

Confirmation Responses are sent by the server only.

Failure Responses are sent by the server only.

Multiple Data messages can be sent by client.

One ack should be send for each data message by Server.

One confirmation response can be sent for one request.

One failure response can be sent for one request.

One request can be sent per TCP session.

**Test Cases:**

**Case 1**

**The Rational:** This test case was to test that everything works and does what it should.

**Content:**

**Input:**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Expected Output:**

**Commands:**

**Format = 0**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Format = 1**

00000001 484851 5256, 554848, 504848

00000001 484850 5152, 554848

00000001 484851 504848, 5053484848, 565455

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Format = 2**

00000000 00000011 0000000000110000 0000001010111100 0000000011001000

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000000 00000100 0000001010001110 0000000110000110 0000000001011010 0000000000001000

00000000 00000001 0000000000101000

**Format = 3**

00000000 00000011 0000000000110000 0000001010111100 0000000011001000

00000001 484850 5152, 554848

00000001 484851 504848, 5053484848, 565455

00000000 00000100 0000001010001110 0000000110000110 0000000001011010 0000000000001000

00000000 00000001 0000000000101000

**Actual Output:**

**Commands:**

**Format = 0**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Format = 1**

00000001 484851 5256, 554848, 504848

00000001 484850 5152, 554848

00000001 484851 504848, 5053484848, 565455

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Format = 2**

00000000 00000011 0000000000110000 0000001010111100 0000000011001000

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000000 00000100 0000001010001110 0000000110000110 0000000001011010 0000000000001000

00000000 00000001 0000000000101000

**Format = 3**

00000000 00000011 0000000000110000 0000001010111100 0000000011001000

00000001 484850 5152, 554848

00000001 484851 504848, 5053484848, 565455

00000000 00000100 0000001010001110 0000000110000110 0000000001011010 0000000000001000

00000000 00000001 0000000000101000

**Case 2**

**The Rational:** This test case was to test that it can handle errors in format, meaning if an invalid format was passed to client what the output would be.

**Content:**

**Input:**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100011

00000001 484852 545352, 515748, 5748, 56

00000001 484849 5248

**Expected Output:**

**Commands:**

**Format = 4**

(The following message should be printed from client)

Format error.

And an empty output file

**Actual Output:**

**Format = 4**

(The following message was printed from client)

Format error.

Output file empty.

**Case 3**

**The Rational:** This test case was to test that it can handle errors in format of the actual numbers in if they are in the wrong format

**Content:**

**Input:**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100019

00000001 484852 545352, 515748, 5745, 56

00000001 484849 5248

**Expected Output**

**Format = 0**

Format error. (empty output file).

**Format = 1**

Format error. (empty output file).

**Format = 2**

Format error. (empty output file).

**Format = 3**

Format error. (empty output file).

**Actual Output**

**Format = 0**

00000001 484851 5256, 554848, 504848

00000000 00000010 0000000000100010 0000001010111100

00000000 00000011 0000000011001000 0110000110101000 0000001101100019

00000001 484852 545352, 515748, 5745, 56

00000001 484849 5248

**Format = 1**

Format error. (empty output file).

**Format = 2**

Format error. (empty output file).

**Format = 3**

Format error. (empty output file).

**Errors:**  With format = 0 the file just printed instead of reporting the format error.

**Case 4**

**The Rational:** This test case was to test the extreme values can be handled properly but programs

**Content:**

**Input:**

00000001 484850 48, 5453535153

00000000 00000010 0000000000000000 1111111111111111

**Expected Output**

**Format = 0**

00000001 484850 48, 5453535153

00000000 00000010 0000000000000000 1111111111111111

**Format = 1**

00000001 484850 48, 5453535153

00000000 00000010 1111111111111111 0000000000000000

**Format = 2**

00000000 00000010 0000000000000000 1111111111111111

00000000 00000010 1111111111111111 0000000000000000

**Format = 3**

00000000 00000010 0000000000000000 1111111111111111

00000001 484850 5453535153, 48

**Actual Output**

**Format = 0**

00000001 484850 48, 5453535153

00000000 00000010 0000000000000000 1111111111111111

**Format = 1**

00000001 484850 48, 5453535153

00000000 00000010 1111111111111111 0000000000000000

**Format = 2**

00000000 00000010 0000000000000000 1111111111111111

00000000 00000010 1111111111111111 0000000000000000

**Format = 3**

00000000 00000010 0000000000000000 1111111111111111

00000001 484850 5453535153, 48

**The usage of your client and server program:**

The Server calls command was: ./Server 3353 0.6 8

The Client calls command was: ./Client 127.0.0.1 input\_file\_path 3 output 0.6 8

The first number represents the format, second loss probability and third random seed.

**Instruction to compile:**

When the file is on zipped open the folder UDP\_Project then open the folder Final\_Folder\_Client\_Server\_Program\_UDP and from there open two terminals: in the first terminal run the command gcc Server\_main.c -o Server and then run ./Server <port number> <loss probability> <random seed> ( the other files are there if need as well).

In the second terminal run the command gcc Client\_main.c -o Client and the run ,/Client 127.0.0.1 <input file path> <format> <output name> <loss probability> <random seed>

The server must be running first or the program won’t run.

After the running of the program there should be a file in the folder of the server program that holds the output of the units, unless it failed.

NOTE!!!!!

DR. Li I know my work is late it is partly due to the fact that me pc crashed today, and I had not saved a big bit of my work wasn’t using github as I should. Anyway, this note is to let you know that I researched how to use the Alarm() signal that was on the slides to help implement my stop and wait and I discovered after about 2 days of searching that the alarm interrupt doesn’t not work on windows. I didn’t have a vm of a Linux box to test it on. So, I left all the code snippets that should work for the stop and wait commented in my code. As my code is not it works, and I was able to test it however I was not able to test the stop and wait stuff. As such I was just hoping that you could still take it into consideration. So that my grade doesn’t be too bad (especially since I’m already late). I believe that I generally grasped the concept and the implementation to a point, so I hope that the implementation I out in even though commented can earn me some marks