**COMPUTER NETWORKING**

**HW 4**

**FTP IMPLEMTATION WITH TCP**

**14조**

1. **Abstract**

|  |
| --- |
| 1. Client can set it’s get rate and put rate. 2. Server can deal with multiple clients. 3. A client can do multiple file transfer in it’s process. |

1. **interface**

Overall system and interface are same as HW3(FTP using UDP). When we run client program, we will encounter 1st interface.

|  |  |
| --- | --- |
| **1st interface : connect to server or quit** | |
| **Connect[IP/HOST] [PORT]** | Connect to server using given IP or PORT.  Client can either type 127.0.0.1(IP ADDR)  or hostname(HOSTNAME) |
| **[STUDENT NO.]** | Print out given student’s contributions. |
| **quit** | Terminate client program. |

After connection to server, 2nd interface are as follows.

|  |  |
| --- | --- |
| **2nd interface : file transfer** | |
| **get [FILENAME]** | Get file from server program’s current directory. |
| **put [FILENAME]** | Put file into server program’s current directory. |
| **sendrate [n]K** | Set his own send rate to n KB (A block) |
| **recvrate [n]K** | Set his own recv rate to n KB (A block) |
| **ratecurr** | Print out his rates of send and recv. |
| **close** | Break connection. |

1. **Transfer rate**

Client can set it’s own transfer speed. Server mangaes large array of struct about each clients’ speed. Indexing will be implemented by each clients’ fd number.

|  |  |
| --- | --- |
|  |  |

Whenever receive command “sendrate” or “recvrate” from a ‘client 6’, ‘client 6’ changes own transfer rates with the values he typed. And after that, ‘client 6’ send its values to server to let server modify his(client 6) transfer rate by accessing speeds[6] struct.

In this case, client’s receiving rate and server’s sending rate will be set as same value. (This applied equally to client’s sending rate and server’s receiving rate.)

1. **Multiple clients**

As for UDP. We do not need any connections between server and client. Any clients could send messages as long as it has information about server socket. While clients’ multiplexing can be implemented easily(but may require several algorithms for reliable transfer). TCP needs a little bit different way due to it’s mandatory-connection characteristic. A socket is allocated to a clear connection.  
 Multiplexing can be implemented by means of either multi-processing(Using fork()/exec()) or select() function. We Originally used multi-processing program but challenged difficulty about managing each socket and clients’ transfer rates. For this reason, we decided to apply select(). Each clients would be identified by fd.

|  |
| --- |
|  |
| Part of select() function |

1. **Multiple threads for a client**

Multiple clients are binded in one socket. But their each jobs will be run on a thread with another socker. In this case, a single client can run multiple transfer.

This code uses pthread for implementation of multithread.

|  |
| --- |
|  |

1. **Results**

|  |
| --- |
| **Multiple Clients’ single file transfer** |
|  |
| Client A requested **‘put centos2\_.vdi’(1029.70MB)**  And Client B requested **‘get centos\_vdi’(1029.70MB)** simultaneously. |
|  |
| Two files were Successfully transferred. |

|  |
| --- |
| **Single Clients’ multiple file transfer** |
|  |
| A client requested **‘put centos\_.vdi’(1029.70MB)** and **‘get test8.zip’(106.55MB)** simultaneously. |
|  |
| Two files were Successfully transferred. |

|  |
| --- |
| **Multiple Clients’ multiple file transfer** |
|  |
| Client A requested **‘get test8.zip’(106.55MB)** and  Client B requested **‘put centos\_.vdi’(1029.70MB)** and  **‘get centos2\_vdi’(1029.70MB)** simultaneously. |
|  |
| Three files were Successfully transferred. |