**Computer Networks**

Homework #2

2015318622 노미래 Noh Mirae

**Problem 1: List 3 advantages of IMAP in comparison with POP3**

IMAP is similar to POP3, but it provides more powerful functions.

First, in IMAP, a user can check the e-mail header prior to downloading.

Second, a user can search the contents of the e-mail for a specific string of characters prior to downloading, and also can partially download e-mail.

Lastly, a user can create, delete, rename, and create a hierarchy of mailboxes.

**Problem 2: In a Chord network, we have node N10 and key k10.**

1. Is N10 the successor of k10? Why?

N10 is the successor of k10.

The meaning of successor is the closest peer with N>=k.

N10 is the closest peer with k, N10>=k10. Therefore N10 is the successor of k10.

1. Is N10 the predecessor of k10? Why?

No, N10 is not the predecessor of k10.

In find\_successor function of Chord interface, we find the successor by finding the predecessor first and returning predecessor.fingertable[1]. N10 can be the successor of k10, then predecessor of k10 must be located before k10, because its fingertable[1] have to point the successor.

**Problem 3: Why do HTTP, FTP, SMTP, and POP3 run on top of TCP rather than on UDP?**

TCP provides connection-oriented service, so it is more reliable than UDP. HTTP, FTP, SMTP, and POP3 need to send long messages and require reliability. Therefore, they run on top of TCP rather than on UDP.

Problem 4:

* Introduction

I developed simple TCP server and client programs for file transfering.

The server program uses port number 1398 and sends a file named “send”.

The client program gets the port number and the name of the file to save from the user, recieves the data of file from the server, and save it to the file.

* Development Environment

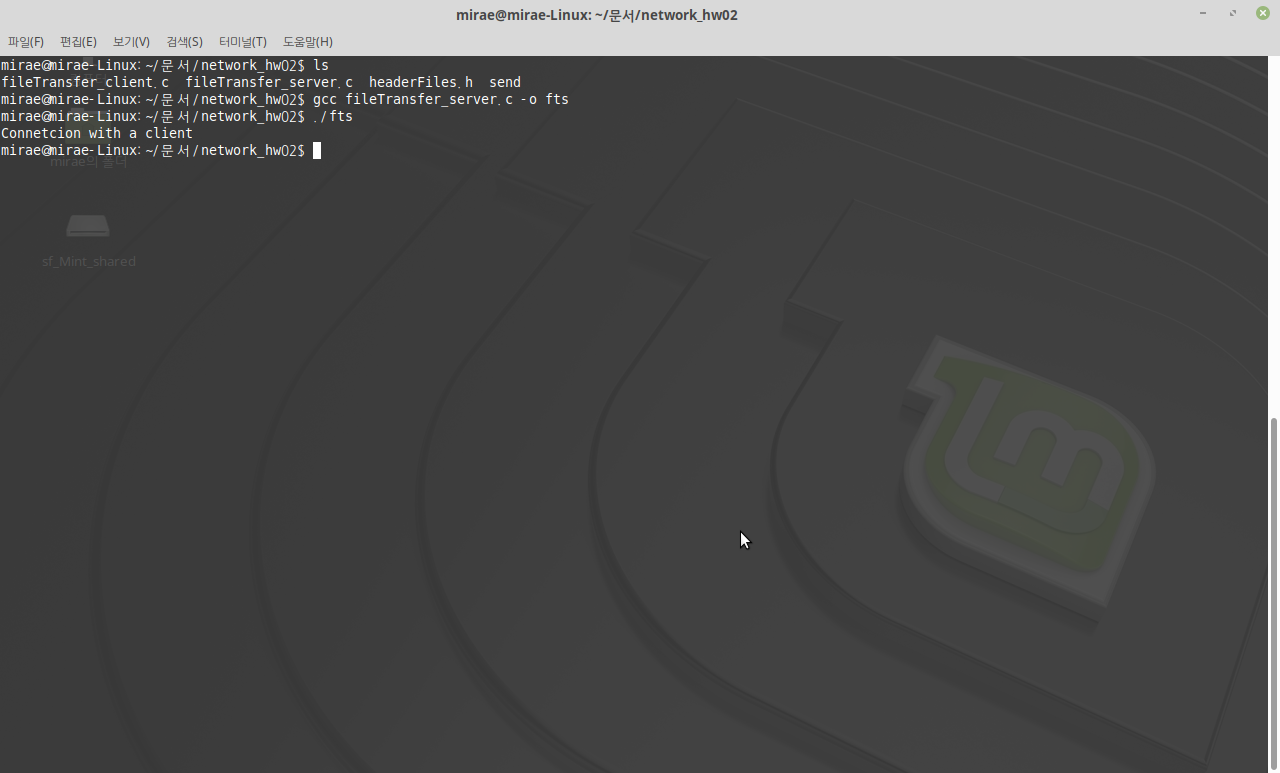
The application works in LINUX environment.

It can be compiled by gcc compiler and runned in the terminal.

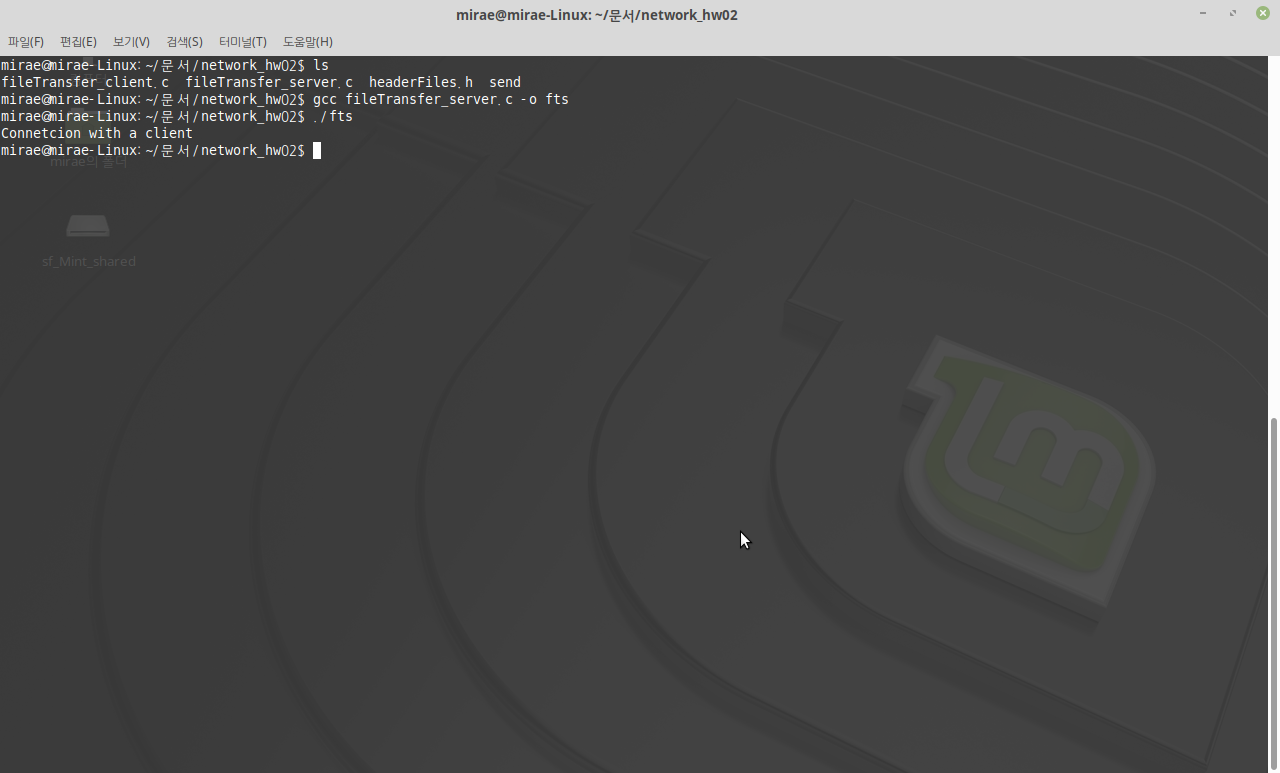
* How to complile and run the application

We need to run terminal in LINUX environment to compile and run the program.

In the application directory, there are the files below.



First, let’s compile and run the server program.

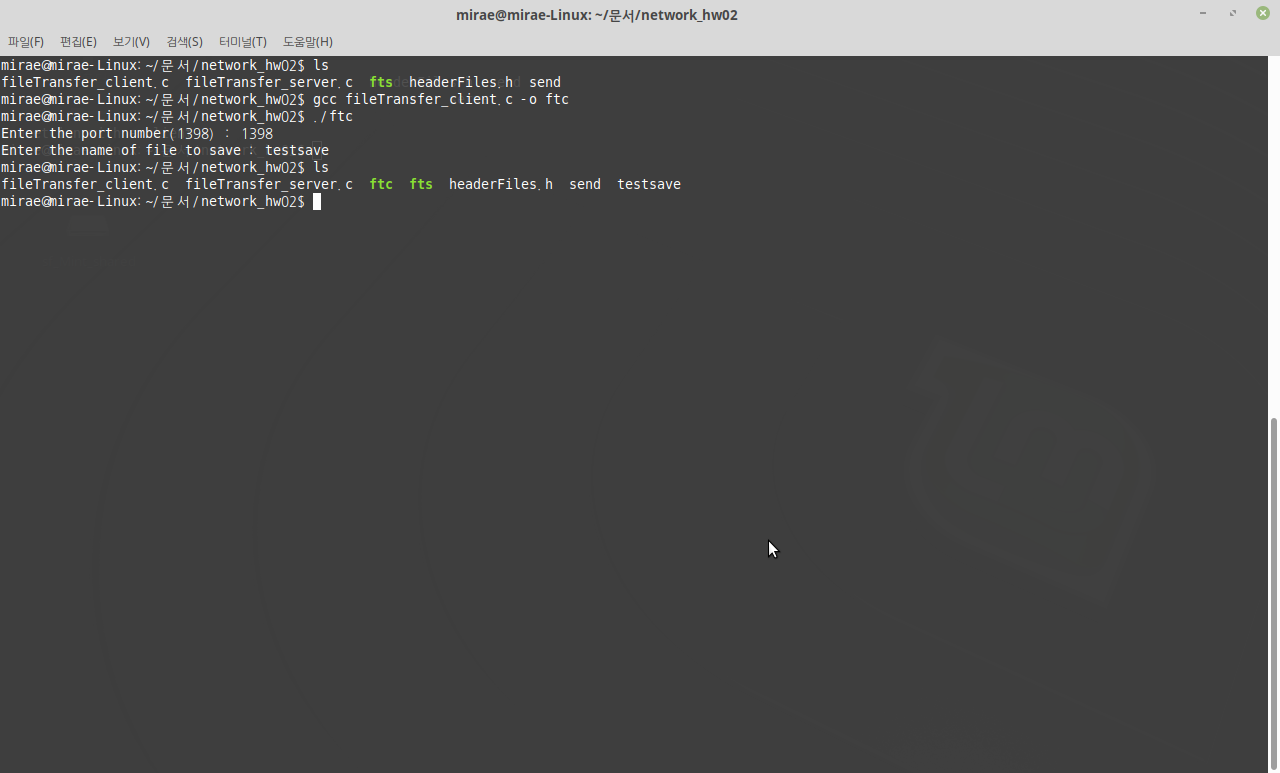


We can compile “fileTransfer\_server.c” into the executable file named “fts” via gcc command.

Then, we can run the server program by typing “./fts”.

The server program prints out “Connection with a client” when it is connected to the client program. It automatically terminates when it finishes transfering the file.

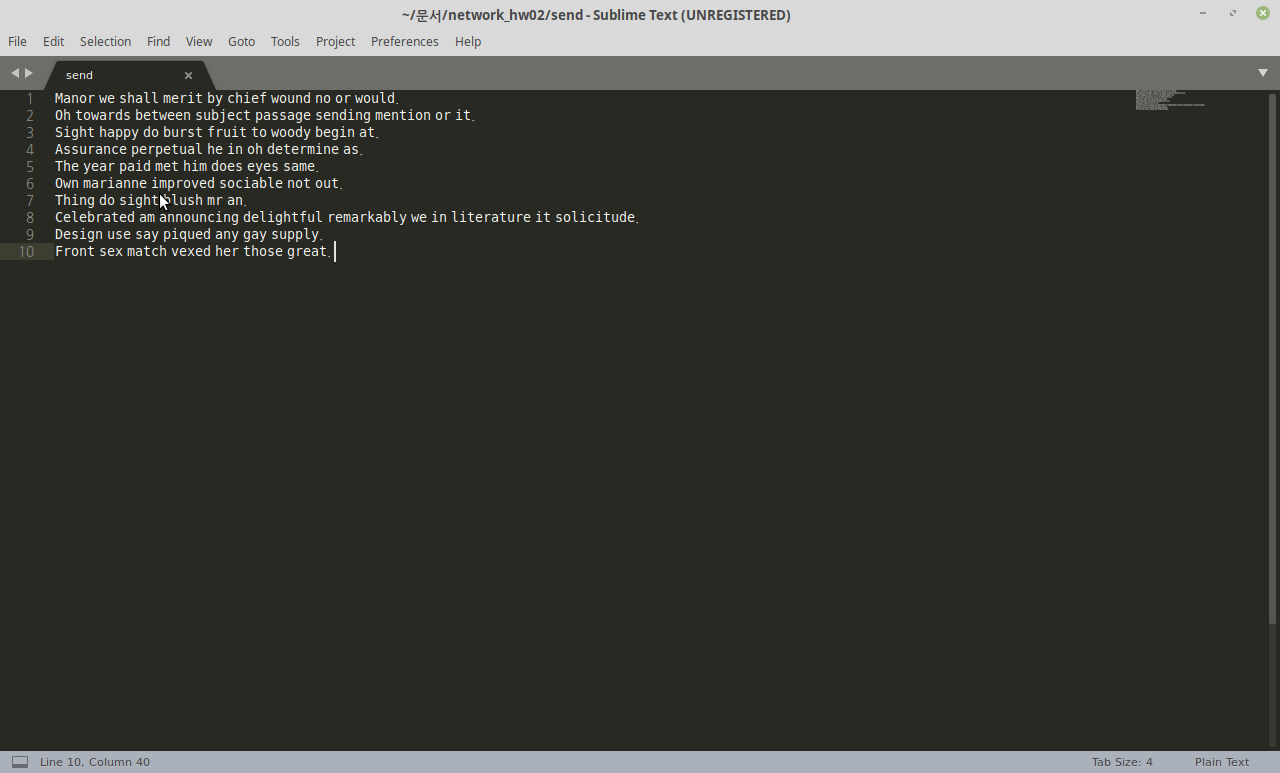
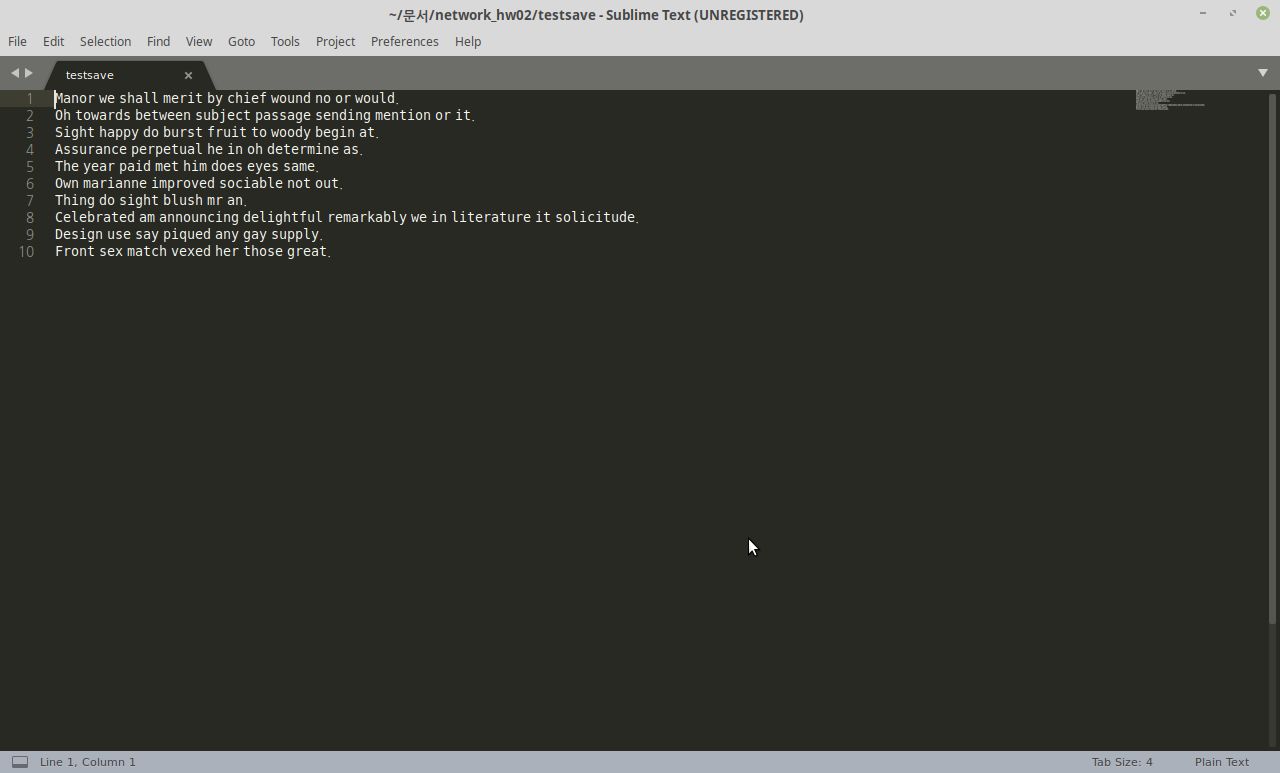
Then let’s compile and run the client program.



We also can compile and run the server program by using gcc command and “./ftc”.

When the client program starts, we have to enter the port number and the name of file to save. Then it connects to the server program which has the exact port number, recieve the file data from the server, and save the data on the file.

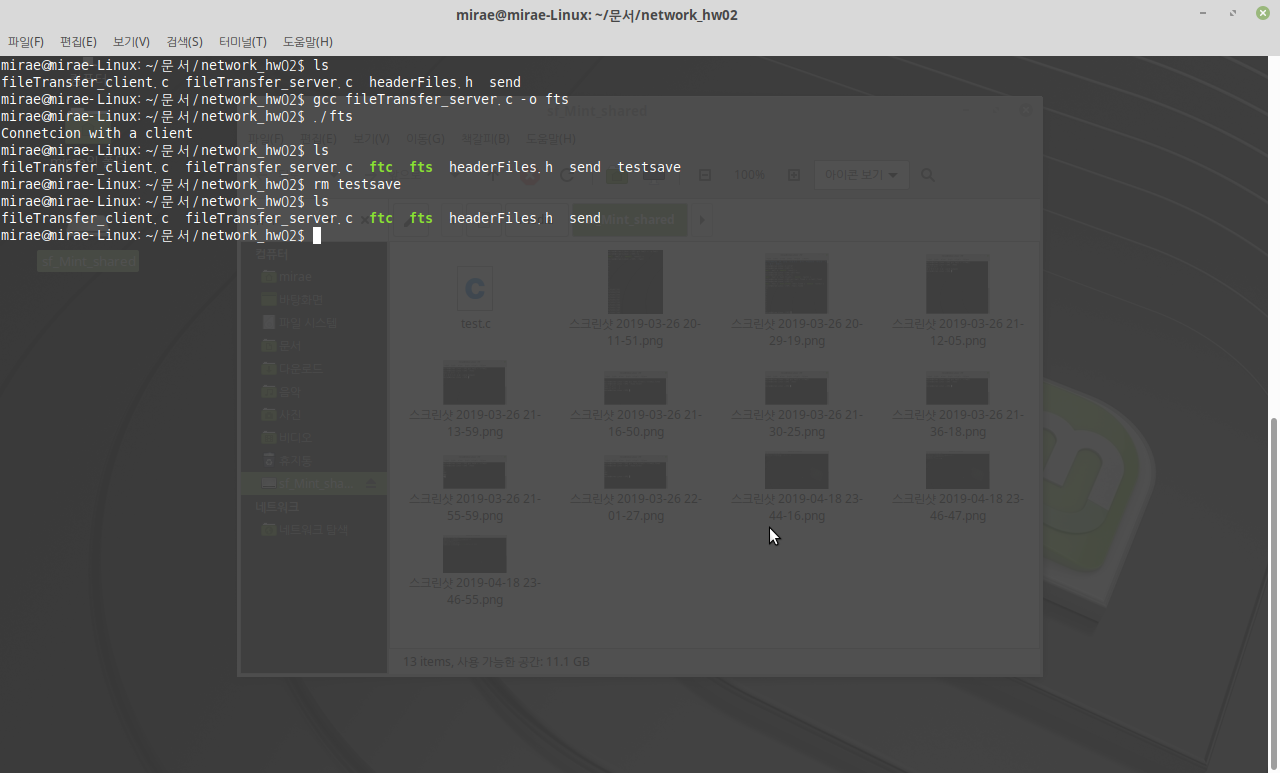
Lastly, let’s compare the sent file and the recieved file.

They’re exactly same, so the file transferring was successfully done.

* Folders and Files

Whole files for the program are located in a single directory.



headerFiles.h : Header file which contains all required headerfiles for the server and the clien

fileTransfer\_server.c : C source file for the server program

fileTransfer\_client.c : C source file for the client program

fts : The executable server program which is compiled from “fileTransfer\_server.c”

ftc : The executable client program which is compiled from “fileTransfer\_client.c”

send : The pre-determined file to be sent.

* Implementation

To implement the application, I basically followed the source codes in the text book. But I edited them properly and added function for reading, writing, and transfering files.

1. Server program

<reading file>

|  |
| --- |
| char\* readFile(char \*string, int \*size) {    FILE \*fptr;  fptr = fopen(FILENAME, "r");  // Find out file's size and allocate string pointer  fseek(fptr, 0L, SEEK\_END);  \*size = ftell(fptr);  fseek(fptr, 0L, SEEK\_SET);  string = (char\*)calloc(\*size, sizeof(char));  for (int i=0; i<\*size; ++i)  fscanf(fptr, "%c", &(string[i]));  fclose(fptr);  return string;  } |

Reads pre-determined file, “send”, by using file pointer.

Gets the size of the file, using fseek function.

Returns the pointer of given string.

The server’s structure of creating and binding listen socket, listening to requests, and accepting the connection from client is same as in the textbook. But I deleted things related buffer because the server doesn’t have to receive data through it.

<sending data>

|  |
| --- |
| send(s, contents, len, 0); |

Simply sends the contents of given file by using send function and the data socket.

2. Client program

<data transferring section>

|  |
| --- |
| // Data transfer section  while((n = recv(s, ptr, maxLen, 0)) > 0) {  ptr += n;  maxLen -= n;  len += n;  }    buffer[len] = '\0';  writeFile(buffer, len); |

I changed data transferring section a lot.

The client program doesn’t send data, but just recieved data from the server.

So it simply recieves the data through “recv” function and save it using “writeFile()”

<writing file>

|  |
| --- |
| void writeFile(char\* string, int size) {  char fName[100];  printf("Enter the name of file to save : ");  scanf("%s", fName);  FILE \*fptr;  fptr = fopen(fName, "w");  for (int i=0; i<size; ++i)  fprintf(fptr, "%c", string[i]);  fclose(fptr);  return;  } |

Gets the file name to save on by the user.

Write given data on the file by using file pointer and “fprintf” function.