Spring 2019 Operating Systems
Lab 12: Chatting Server

1. 목표

우리는 지난 실습까지 리눅스에서 전반적으로 사용할 수 있는 명령과 이론에 대해서 프로그램을 통하여 구현하고 테스트하였다. 이번 LAB12는 한 학기를 마무리하는 차원에서 C언어 작성능력을 전반적으로 증진시킬 수 있는 소켓 프로그래밍에 대하여 실습해본다.

2. 실습

[chatclient.c]

```
//
// A client that uses select() to enable asynchronous user input.
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAXDATASIZE 1000

int
main(int argc, char *argv[])
{
    fd_set master, read_fds;
    struct addrinfo hints, *servinfo;
    int rv;
    int sockfd;
    int numbytes;
    int quit;
    char buf[MAXDATASIZE];
    char tmpbuf[MAXDATASIZE];
    char s[INET_ADDRSTRLEN];

    // If number of command-line arguments is not 2, print usage and exit.
    if (argc != 3) {
        fprintf(stderr, "usage: %s hostname portnum\n", argv[0]);
        exit(1);
    }

    // Prepare the data structure "servinfo".
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    if ((rv = getaddrinfo(argv[1], argv[2], &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }

    // Create a socket using "servinfo".
    if ((sockfd = socket(servinfo->ai_family, servinfo->ai_socktype, \
            servinfo->ai_protocol)) == -1) {
        perror("client: socket");
        return 2;
    }

    // Connect to the server.
    if (connect(sockfd, servinfo->ai_addr, servinfo->ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect");
        return 2;
    }
```

```
    // Print debug message.
    inet_ntop(servinfo->ai_family, \
        &((struct sockaddr_in *)servinfo->ai_addr)->sin_addr, s, sizeof(s));
    printf("client: connecting to %s\n", s);

    // The "servinfo" data structure is no longer needed, free the data.
    freeaddrinfo(servinfo);

    // Initialize fd_set structure.  Only stdin(0) and the socket connected
    // to the server is active.
    FD_ZERO(&master);
    FD_SET(0, &master);
    FD_SET(sockfd, &master);

    // If "quit" is 1, then we break out of the while loop.  Initially "quit"
    // is 0.
    quit = 0;

    // The main loop
    while (1) {
        // Make a copy of "master" and use that, because the function select()
        // will modify the data.
        read_fds = master;
        if (select(sockfd+1, &read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            exit(1);
        }

        for (int i = 0; i <= sockfd+1; i++) {
            if (FD_ISSET(i, &read_fds)) {
                if (i == 0) {
                    // Message entered by user (stdin).
                    fgets(buf, sizeof(buf), stdin);  // read the message from stdin.
                    buf[strlen(buf)-1] = '\0';  // need to attach a null character at the end.

                    // If the user enters "/quit", set the "quit" flag.
                    if (strcmp(buf, "/quit") == 0) {
                        quit = 1;
                        break;
                    }

                    // Otherwise, send the message to the server.
                    if (send(sockfd, buf, strlen(buf), 0) == -1) {
                        perror("send");
                        close(sockfd);
                        exit(1);
                    }
                } else if (i == sockfd) {
                    // Message received from server.  Receive message from the socket.
                    if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
                        perror("recv");
                        close(sockfd);
                        exit(1);
                    }

                    // If numbytes is 0, it means the server is disconnected.
                    if (numbytes == 0) {
                        printf("server disconnected.\n");
                        quit = 1;
                        break;
                    }

                    // Attach the null character at the end and print message
                    // on the display.
                    buf[numbytes] = '\0';
                    printf("%s\n", buf);
                }
            }
        }

        if (quit) {
            break;
        }
    }

    printf("Connection closed.\n");
    close(sockfd);

    return 0;
}
```

[chatserver.c]

```c
//
// chat server
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <time.h>

#define MAXDATASIZE 1000
#define BACKLOG 10
#define MAXUSERS 20
#define MAXIDSIZE 40

// Global variables.
char userid[MAXUSERS][MAXIDSIZE];
unsigned int temp_usernum;

int
main(int argc, char *argv[])
{
    fd_set master, read_fds;
    socklen_t sin_size;
    struct sockaddr_storage their_addr;
    struct addrinfo hints, *servinfo;
    int rv;
    int yes = 1;
    int sockfd, newfd, fdmax;
    int numbytes;
    char s[INET_ADDRSTRLEN];
    char buf[MAXDATASIZE], tmpbuf[MAXDATASIZE], msg[MAXDATASIZE];

    // For user information.
    temp_usernum = 1;

    // If number of command-line arguments is not 2, show usage and abort.
    if (argc != 2) {
        printf("usage: server portnum\n");
        exit(1);
    }

    // Set random number generator seed randomly (using current time).
    srand(time(NULL));

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;  // use my IP

    // Prepare "servinfo" data structure.
    if ((rv = getaddrinfo(NULL, argv[1], &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        exit(1);
    }

    // Create the "receptionist" socket.
    if ((sockfd = socket(servinfo->ai_family, servinfo->ai_socktype, \
            servinfo->ai_protocol)) == -1) {
        perror("server: socket");
        exit(1);
    }

    // Allow immediate reuse of ports.
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

    // Bind the socket with the addresses in "servinfo".
    if (bind(sockfd, servinfo->ai_addr, servinfo->ai_addrlen) == -1) {
        close(sockfd);
        perror("server: bind");
        exit(1);
    }
```

```
// "servinfo" is no longer needed.
freeaddrinfo(servinfo);

// Begin listening on the port.
if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

// Prepare data structure for using select().  Initially, only the
// receptionist socket are active.
FD_ZERO(&master);
FD_SET(sockfd, &master);
fdmax = sockfd;

// The main loop.
while (1) {
    read_fds = master;
    if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
        perror("select");
        exit(1);
    }

    for (int i = 0; i <= fdmax; i++) {
        if (FD_ISSET(i, &read_fds)) {
            if (i == sockfd) {
                // A new user is trying to connect to the server.
                sin_size = sizeof(their_addr);
                newfd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);

                if (newfd == -1) {
                    perror("accept");
                } else {
                    // Make the new socket file descriptor active in the "master" set.
                    FD_SET(newfd, &master);
                    if (newfd > fdmax) {
                        fdmax = newfd;  // update "fdmax" if necessary
                    }
                    printf("chatserver: new connection from %s on socket %d\n", \
                        inet_ntop(their_addr.ss_family, \
                            &((struct sockaddr_in *)&their_addr)->sin_addr, s, sizeof(s)), newfd);
                }

                // Assign the new user a temporary ID.
                sprintf(userid[newfd], "user%02d", temp_usernum);
                temp_usernum++;

                // Notify the user his/her ID.
                sprintf(buf, "[SERVER] Welcome! your ID is %s", userid[newfd]);
                if (send(newfd, buf, strlen(buf), 0) == -1) {
                    perror("send");
                }

                // Tell other clients that the new user has joined.
                sprintf(buf, "[SERVER] New user %s has joined.", userid[newfd]);
                for (int j = 3; j <= fdmax; j++) {
                    if (j == sockfd) continue;  // skip the receptionist socket
                    if (j == newfd) continue;  // skip the client who just joined
                    if (!FD_ISSET(j, &master)) continue;  // skip the inactive sockets
                    if (send(j, buf, strlen(buf), 0) == -1) {
                        perror("send");
                    }
                }
            } else {
                // A message has arrived from a connected user.
                if ((numbytes = recv(i, buf, sizeof(buf), 0)) <= 0) {
                    // If numbytes is less than or equal to 0, it means
                    // the client is disconnected.
                    if (numbytes == 0) {
                        printf("chatserver: socket %d hung up\n", i);
                    } else {
                        perror("recv");
                    }
                    close(i);
                    FD_CLR(i, &master);

                    // Tell other clients that a user has left.
                    sprintf(buf, "[SERVER] %s has left.\n", userid[i]);
                    for (int j = 3; j <= fdmax; j++) {
                        if (j == sockfd) continue;  // skip the receptionist socket
                        if (j == i) continue;  // skip the client who sent the message
                        if (!FD_ISSET(j, &master)) continue;  // skip the inactive sockets
                        if (send(j, buf, strlen(buf), 0) == -1) {
                            perror("send");
                        }
                    }
```

```
                    } else {
                        // A message has arrived.
                        buf[numbytes] = '\0';
                        strcpy(msg, buf);
                        printf("server received from %s \"%s\"\n", userid[i], msg);

                        if (strncasecmp(buf, "/id ", 4) == 0) {
                            strcpy(tmpbuf, userid[i]);  // save the old ID in "tmpbuf"
                            strcpy(userid[i], buf+4);  // update the user ID

                            // Send a message to client i.
                            sprintf(buf, "[SERVER] Changed ID to %s", userid[i]);
                            if (send(i, buf, strlen(buf), 0) == -1) {
                                perror("send");
                            }

                            // Send a message to other clients.
                            sprintf(buf, "[SERVER] %s changed ID to %s", tmpbuf, userid[i]);
                            for (int j = 3; j <= fdmax; j++) {
                                if (j == sockfd) continue;
                                if (j == i) continue;
                                if (!FD_ISSET(j, &master)) continue;
                                if (send(j, buf, strlen(buf), 0) == -1) {
                                    perror("send");
                                }
                            }
                        } else if (strncasecmp(buf, "/user", 5) == 0) {
                            // Prepare a message to send to client i.
                            strcpy(buf, "");
                            for (int j = 3; j <= fdmax; j++) {
                                if (j == sockfd) continue;
                                if (!FD_ISSET(j, &master)) continue;
                                strcat(buf, userid[j]);
                                strcat(buf, "\n");
                            }
                            if (send(i, buf, strlen(buf), 0) == -1) {
                                perror("send");
                            }
                        } else {
                            strcpy(tmpbuf, buf);
                            sprintf(buf, "%s: %s", userid[i], tmpbuf);
                            for (int j = 3; j <= fdmax; j++) {
                                if (j == sockfd) continue;
                                if (j == i) continue;
                                if (!FD_ISSET(j, &master)) continue;
                                if (send(j, buf, strlen(buf), 0) == -1) {
                                    perror("send");
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    return 0;
}
```

3. 컴파일

1) chatclient: **gcc chatclient.c -o client**
2) chatserver: **gcc chatserver.c -o server**

4. 확인

컴파일을 수행한 후 다음과 같이 파일이 나타나면 된다.

```
s_guest@A1409-OSLAB-01:~/os/net$ ls
chatclient.c  chatserver.c  client  server
```

5. 테스트

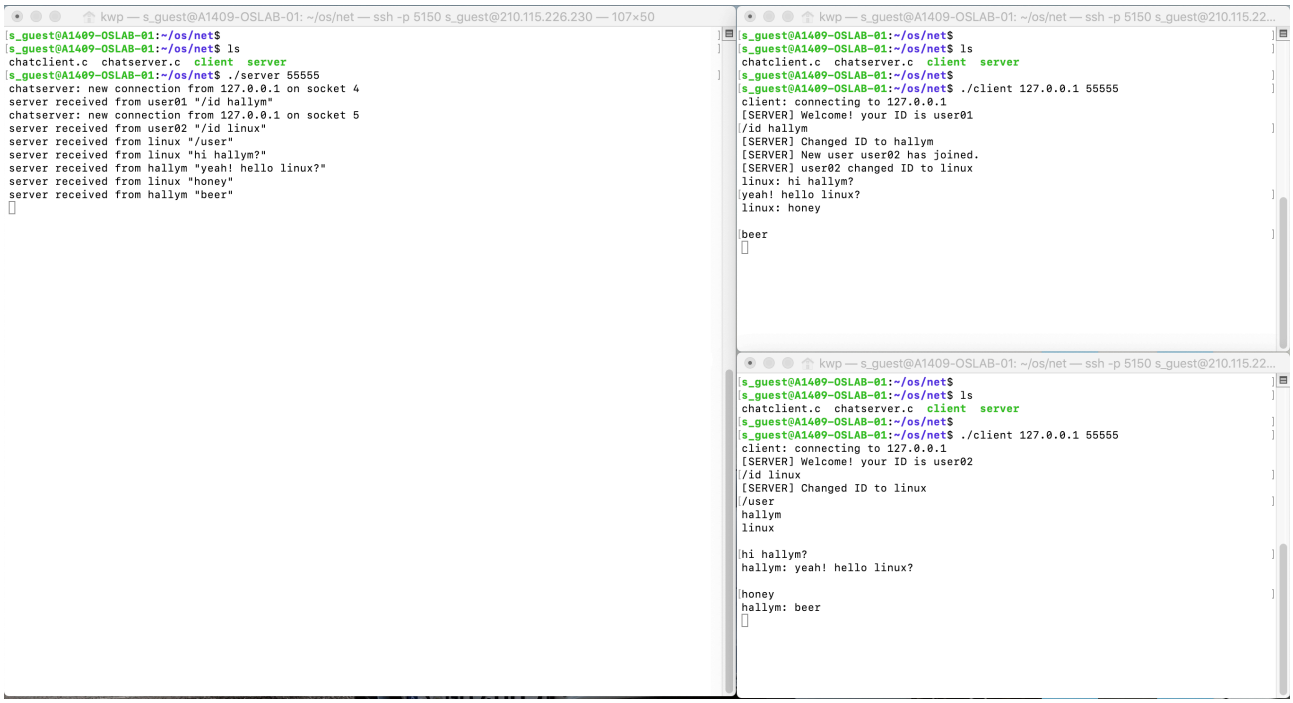
1) 서버 프로그램을 실행시킨다. **./server 55555**
(**55555는 포트번호이므로 각자 개별설정한다. 1025~65535**)

2) 추가 쉘을 하나 더 실행해서 클라이언트를 실행한다. **./client 127.0.0.1 55555**
여기서 127.0.0.1은 localhost를 의미한다.

3) 클라이언트 프로그램에서 아이디를 수정한다. **/id hallym**

4) 클라이언트 프로그램에서 /user, /quit 명령어를 각각 실행해본다. **/user**, **/quit**

6. 실행 예



쉘 3개를 만든 후 먼저 서버 프로그램을 실행시킨다.

나머지 2개의 쉘에서는 ./client 127.0.0.1 55555 로 서버 프로그램에 접속한다.

클라이언트끼리 서로 대화를 주고 받을 수 있다.