**· 채팅 프로그램 설명**

- 서버: 임의의 클라이언트로부터 채팅 참여 요청이 오면 참여리스트에   
추가합니다. 그리고 참여한 클라이언트로부터 온 메시지를 채팅에  
참여한 클라이언트(들)에게 전송하는 기능을 수행합니다.

- 클라이언트: 서버에게 채팅 참여 요청을 하여 채팅에 참가한 후 다른   
클라이언트와 채팅합니다.

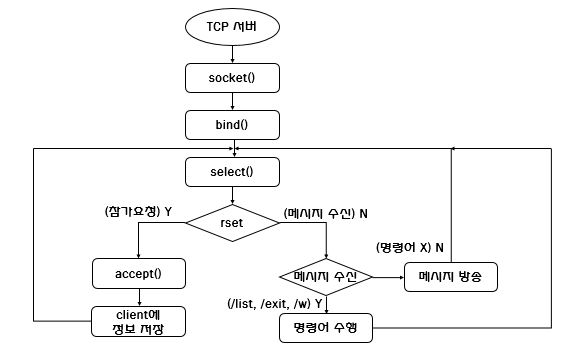
**· 채팅 프로그램 구조 설명(서버)**

- 채팅 서버는 임의의 클라이언트로부터 오는 채팅 참여요구 처리와 동시에,   
채팅에 참여한 임의의 클라이언트의 메시지를 방송해주는 역할을 수행합니다.

- 채팅 서버는 참여요청과 메시지 방송을 동시에 처리해야하므로 어떤 일이   
만족되어야 동작하는 blocking 모드로 하면 안됩니다.

- 그러므로, select() 함수를 이용하여 참가요청과 각 클라이언트로부터의   
메시지 전송을 동시에 관리, 처리합니다.

- 다음 그림은 채팅 프로그램이 동작하는 과정을 나타냅니다.



- 채팅 서버는 rset에 저장된 감시대상에서 데이터가 전송되면   
해당 소켓의 업무를 처리합니다.

- 참가요청이 온 경우 다음 구조체의 client에 클라이언트 정보를 저장합니다.  
struct connection {  
 int sock;  
 char name[20];  
 char ipaddr[20];  
 int port;  
} client[FD\_SETSIZE];

- 메시지 방송이라면, 명령어인지 단순히 메시지인지 확인한 후   
해당 역할을 수행합니다.

- 주어진 역할을 수행 후에는 다시 select를 호출하여 대기합니다.

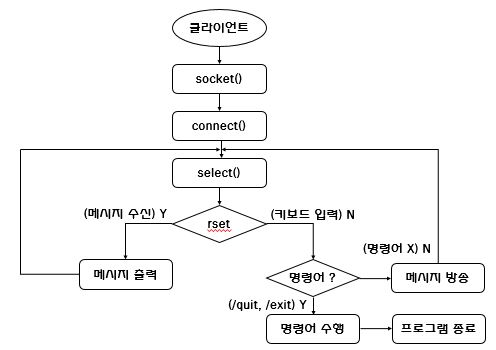
**· 채팅 프로그램 구조 설명(클라이언트)**

- 클라이언트 채팅은 키보드로부터 오는 입력 처리와 동시에  
 서버로부터 오는 방송 메시지를 수신하는 역할을 수행합니다.

- 즉, 클라이언트 역시 2개 이상의 동작을 동시에 수행하므로,  
 select() 함수를 이용하여 동시에 관리하도록 되어있습니다.

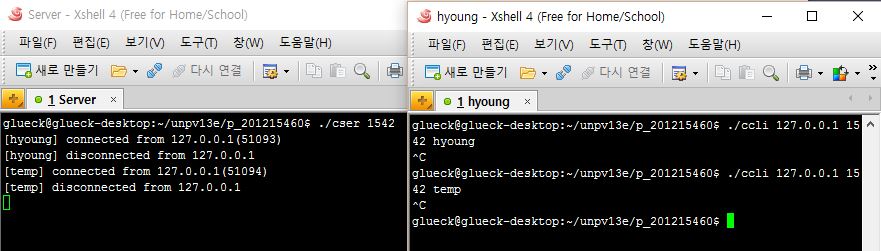
- ‘/quit’, ‘/exit’을 입력받으면 채팅 서버와 연결을 종료합니다.

- 클라이언트 구조는 다음 그림과 같이 되어있습니다.

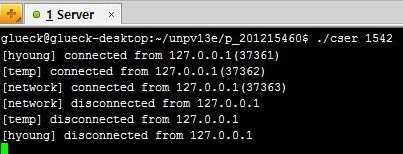


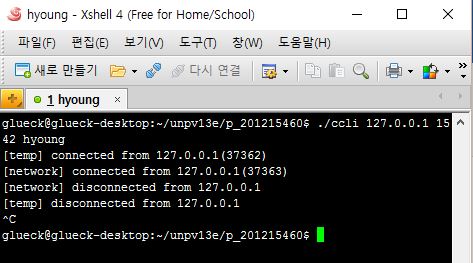
**· 채팅 프로그램 시연 사진.**

**- 연결/연결종료 확인(서버)**



**- 연결/연결종료 확인(클라이언트 ‘hyoung’, ‘temp’, ‘network’)**

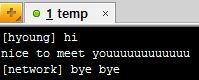
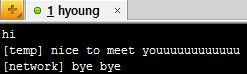


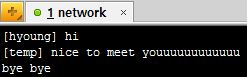


- ‘hyoung’ 컴퓨터는 처음으로 들어왔으므로 다음 참가자들의 접속여부를   
 받아볼 수 있습니다.

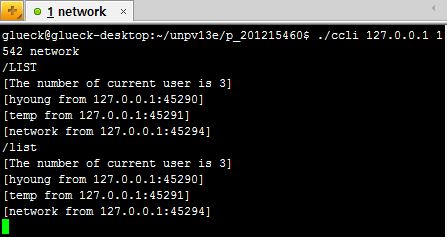
- ‘network’ 컴퓨터는 마지막으로 들어왔으므로 먼저 참가한 접속자들을   
 하려면 ‘/list’ 명령어를 이용하면 됩니다.

**- 채팅 시연 정상작동.**

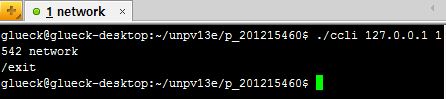




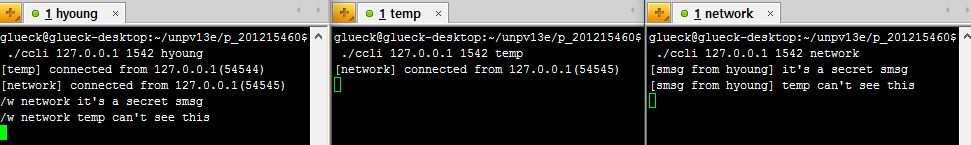
**- ‘/list’ 명령어: 현재 채팅 참가자들을 확인합니다. (‘/LIST’ 동일)**



**- ‘/exit’ 명령어: 채팅프로그램을 종료합니다. (‘/EXIT’, ‘/quit’, ‘/QUIT’ 동일)**



**- ‘/w’ 명령어: 특정 대상에게 귓속말을 보냅니다. (‘/w’, ‘/smsg’ 동일)**

****

- 특정 대상에게만 전송되며, 그 외의 채팅 참여자에게는 전송되지 않습니다.

**· 코드 디렉토리 및 컴파일**

- 코드 및 실행파일 압축명: p\_201215460.tar.gz  
- unpv13e 디렉토리 안에 p\_201215460.tar.gz를 넣어줍니다.  
- 압축 해제: tar xvf p\_201215460.tar.gz  
- (서버 코드) cser.c, (클라이언트 코드) ccli.c 를 컴파일 합니다.  
gcc -o cser -I ../lib cser.c ../libunp.a  
gcc -o ccli -I ../lib ccli.c ../libunp.a  
- 실행합니다.

**· 코드 설명(서버)**

#include "unp.h"

**// 클라이언트 정보를 저장할 구조체 공간**

struct connection {

int sock;

char name[20];

char ipaddr[20];

int port;

};

int main(int argc, char \*\*argv) {

int i, j, cnt, nready;

int maxi, maxfd, listenfd, connfd, sockfd;

char buf[MAXLINE], sendbuf[MAXLINE];

ssize\_t n;

fd\_set rset, allset;

socklen\_t clen;

struct connection client[FD\_SETSIZE];

struct sockaddr\_in caddr, saddr;

**// 서버 실행 시 필요한 정보 누락 시 사용법 출력 후 종료.**

if(argc != 2)

err\_quit("usage: ./cser <PORT>");

listenfd = Socket(AF\_INET, SOCK\_STREAM, 0);

bzero(&saddr, sizeof(saddr));

saddr.sin\_family = AF\_INET;

saddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

saddr.sin\_port = htons(atoi(argv[1]));

Bind(listenfd, (SA \*) &saddr, sizeof(saddr));

Listen(listenfd, LISTENQ);

**/\* 초기화 \*/**

maxfd = listenfd;

maxi = -1;

cnt = 0;

for(i=0; i<FD\_SETSIZE; i++) client[i].sock = -1;

FD\_ZERO(&allset);

FD\_SET(listenfd, &allset);

for( ; ; ) {

rset = allset;

nready = Select(maxfd+1, &rset, NULL, NULL, NULL);

**/\* 새로운 접속 요청 시 \*/**

if(FD\_ISSET(listenfd, &rset)) {

clen = sizeof(caddr);

connfd = Accept(listenfd, (SA \*) &caddr, &clen);

for(i=0; i<FD\_SETSIZE; i++)

if(client[i].sock < 0) {  
 cnt++;

client[i].sock = connfd;

if((n = Read(connfd, client[i].name, sizeof(client[i].name))) == 0)

err\_quit("Connection's nickname missed");

snprintf(client[i].ipaddr, sizeof(client[i].ipaddr), "%s",   
 Inet\_ntop(AF\_INET, &caddr.sin\_addr, buf, sizeof(buf)));  
 client[i].port = ntohs(caddr.sin\_port);

**// 기존 접속자에게 새로운 접속자 알림**

sprintf(buf, "[%s] connected from %s(%d)\n",

client[i].name, client[i].ipaddr, client[i].port);

printf("%s", buf);

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

if(client[j].sock == connfd) continue;

Writen(client[j].sock, buf, strlen(buf));

}

break;

}

if(i == FD\_SETSIZE) err\_quit("Too many clients");

FD\_SET(connfd, &allset);

if(connfd > maxfd) maxfd = connfd;

if(i > maxi) maxi = i;

}

**/\* 클라이언트로부터 메시지 수신 시 \*/**

for(i=0; i<=maxi; i++) {

memset(sendbuf, 0, sizeof(sendbuf));

if((sockfd = client[i].sock) < 0) continue;

if(FD\_ISSET(sockfd, &rset)) {

**// 클라이언트로부터 수신 데이터 크기가 0인 경우 접속 해제**

if((n = Read(sockfd, buf, MAXLINE)) == 0) {

sprintf(buf, "[%s] disconnected from %s\n",

client[i].name, client[i].ipaddr);

printf("%s", buf);

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

if(client[j].sock == sockfd) continue;

Writen(client[j].sock, buf, strlen(buf));

}

Close(sockfd);

FD\_CLR(sockfd, &allset);

client[i].sock = -1;

cnt--;

memset(client[i].name, 0, sizeof(client[i].name));

memset(client[i].ipaddr, 0, sizeof(client[i].ipaddr));

}

**/\* 접속 종료 명령어 \*/**

**// '/quit', '/QUIT', '/exit', '/EXIT'**

**// 클라이언트로부터 위 명령어를 입력받을 시 접속을 해제합니다.**  
 else if(n == 6 && (!strncmp(buf, "/exit", 5) || !strncmp(buf, "/quit", 5) ||

!strncmp(buf, "/EXIT", 5) || !strncmp(buf, "/QUIT", 5))) {  
 sprintf(buf, "[%s] disconnected from %s\n",   
 client[i].name, client[i].ipaddr);

printf("%s", buf);

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

if(client[j].sock == sockfd) continue;

Writen(client[j].sock, buf, strlen(buf));

}

Close(sockfd);

FD\_CLR(sockfd, &allset);

cnt--;

client[i].sock = -1;

memset(client[i].name, 0, sizeof(client[i].name));

memset(client[i].ipaddr, 0, sizeof(client[i].ipaddr));

}

**/\* 접속자 확인 명령어 \*/**

**// '/list', '/LIST'**

**// 클라이언트로부터 위 명령어를 입력받을 시 현재 접속자 명단 송신.**

else if(n == 6 && (!strncmp(buf, "/list", 5) || !strncmp(buf, "/LIST", 5))) {

sprintf(buf, "[The number of current user is %d]\n", cnt);

Writen(sockfd, buf, strlen(buf));

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

sprintf(buf, "[%s from %s:%d]\n",   
 client[j].name, client[j].ipaddr, client[j].port);

Writen(sockfd, buf, strlen(buf));

}

}

**/\* 귓속말 명령어 \*/**

**// '/w', '/smsg'**

**// '/w <사용자명> <내용>'**

**// 특정 사용자에게만 메시지를 전송.**

else if(!strncmp(buf, "/w", 2) || !strncmp(buf, "/smsg", 5)) {

char receiver[20];

char tmp[MAXLINE];

char \*temp;

int chk=0;

temp = strtok(buf, " ");

**// 이름 공백 시**

if((temp = strtok(NULL, " ")) == NULL) {

chk = 1;

sprintf(sendbuf, "No receiver name\n");

Writen(sockfd, sendbuf, strlen(sendbuf));

} else sprintf(receiver, "%s", temp);

**// 내용 공백 시**

if(chk == 0) {

if((temp = strtok(NULL, "\n")) == NULL) {

chk = 1;

sprintf(sendbuf, "No mesg\n");

Writen(sockfd, sendbuf, strlen(sendbuf));

} else sprintf(tmp, "%s", temp);

}

**// 제대로 된 귓속말 수신 시**

if(chk == 0) {

**// 13 + name + tmp + ENTER + NULL**

n = 13 + strlen(client[i].name) + strlen(tmp) + 1 + 1;

snprintf(sendbuf, n, "[smsg from %s] %s\n", client[i].name, tmp);

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

if(client[j].sock == sockfd) continue;

if(!strncmp(receiver, client[j].name, strlen(receiver))) {

Writen(client[j].sock, sendbuf, n);

}

}

}

}

else {

**// buf + name + [] + " " + NULL**

n = n + strlen(client[i].name) + 2 + 1 + 1;

snprintf(sendbuf, n, "[%s] %s\n", client[i].name, buf);

for(j=0; j<=maxi; j++) {

if(client[j].sock < 0) continue;

if(client[j].sock == sockfd) continue;

Writen(client[j].sock, sendbuf, n);

}

}

if(--nready <= 0) break;

}

}

}

}

**· 코드 설명(클라이언트)**

#include "unp.h"

void str\_cli(FILE \*fp, int sockfd) {

int n;

int maxfdp1, stdineof;

char buf[MAXLINE], sendbuf[MAXLINE];

fd\_set rset;

stdineof = 0;

FD\_ZERO(&rset);

for( ; ; ) {

**/\* 초기화 \*/**

**// 키보드와 (서버와 연결된)소켓을 select 관리명단에 추가**

if(stdineof == 0) FD\_SET(fileno(fp), &rset);

FD\_SET(sockfd, &rset);

maxfdp1 = max(fileno(fp), sockfd) + 1;

Select(maxfdp1, &rset, NULL, NULL, NULL);

**/\* 서버로부터 메시지 수신 시 \*/**

**// 화면에 출력**

if(FD\_ISSET(sockfd, &rset)) {

if((n = Read(sockfd, buf, MAXLINE)) == 0) {

if(stdineof == 1) return;

else err\_quit("SERVER Terminated\n");

}

Write(fileno(stdout), buf, n);

}

**/\* 키보드로부터 메시지 수신 시 \*/**

**// 종료명령어일 시 소켓 쓰기권한을 종료 후 키보드를 명단에서 제거**

**// 평범한 메시지일 경우 서버로 전송**

if(FD\_ISSET(fileno(fp), &rset)) {

if((n = Read(fileno(fp), buf, MAXLINE)) == 0) {

stdineof = 1;

Shutdown(sockfd, SHUT\_WR);

FD\_CLR(fileno(fp), &rset);

continue;

}

else if(n == 6 && (!strncmp(buf, "/exit", 5) || !strncmp(buf, "/quit", 5) ||

!strncmp(buf, "/EXIT", 5) || !strncmp(buf, "/QUIT", 5))) {

stdineof = 1;

Shutdown(sockfd, SHUT\_WR);

FD\_CLR(fileno(fp), &rset);

continue;

}

Writen(sockfd, buf, n);

}

}

}

int main(int argc, char \*\*argv) {

int sockfd;

struct sockaddr\_in saddr;

**// 프로그램 실행에 필요한 정보 누락 시 사용법 출력 후 프로그램 종료**

if(argc != 4)

err\_quit("usage: a.out <IPaddress> <PORT> <Nickname>");

if((sockfd = Socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

err\_sys("Socket Error");

bzero(&saddr, sizeof(saddr));

saddr.sin\_family = AF\_INET;

saddr.sin\_port = htons(atoi(argv[2]));

Inet\_pton(AF\_INET, argv[1], &saddr.sin\_addr);

Connect(sockfd, (SA \*) &saddr, sizeof(saddr));

Write(sockfd, argv[3], strlen(argv[3])); **// 서버에 클라이언트 닉네임 전송**

str\_cli(stdin, sockfd);

exit(0);

}