Final Project #2

2014146034 이정욱

2014146019 석상우

1. 동작설명

1. Server 는 초기에 Client 들의 접속을 대기한다.

```
Server-socket() sockfd is OK...
Server-bind() is OK...
Server-listen() is OK...
```

2. 새로운 Client1 이 Server 에 접속을 한다.

```
Client_socket() sockfd is OK...
Client_connect() is OK...

======================
Hello! I`m P2p File Sharing Server..
 Please, Log-In!
======================
ID: user1
PW: passwd1
Log-in success!! [user1] *^^*
-------Chatting Room------
```

<- ID 와 Password 가 일치한다면 Log-in success [user(n)]!! 이라는 문구와 함께 Server 와 접속한 Client 에 채팅방이 생성된다.

2-1) 동일하게 Client2 가 Server 에 접속해 채팅창이 생성되는 모습

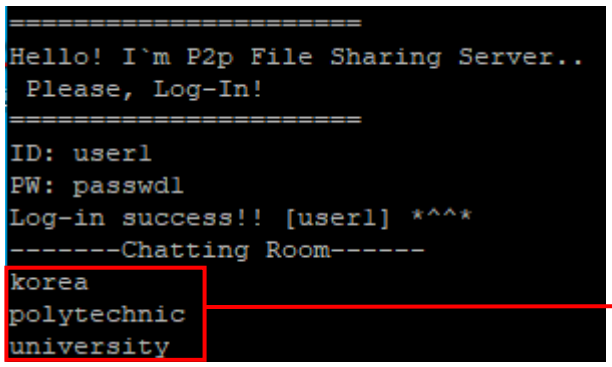
```
Client_socket() sockfd is OK...
Client_connect() is OK...

======================
Hello! I`m P2p File Sharing Server..
 Please, Log-In!
======================
ID: user2
PW: passwd2
Log-in success!! [user2] *^^*
-------Chatting Room------
```

3. Client 들이 접속을 하면 서버에서는 어떤 Client 들이 접속했는지 알 수 있고, 동시에 Client 가 접속 할 때마다 Server 에서도 채팅방이 생성되는 것을 볼 수 있다.

```
========================
User Information
ID : user1, PW : passwd1
========================
Log-in success!! [user1] *^^*
-------Chatting Room-------
[user1] : <4121> , <220.149.128.101>
========================
User Information
ID : user2, PW : passwd2
========================
Log-in success!! [user2] *^^*
-------Chatting Room-------
[user2] : <4122> , <220.149.128.102>
```

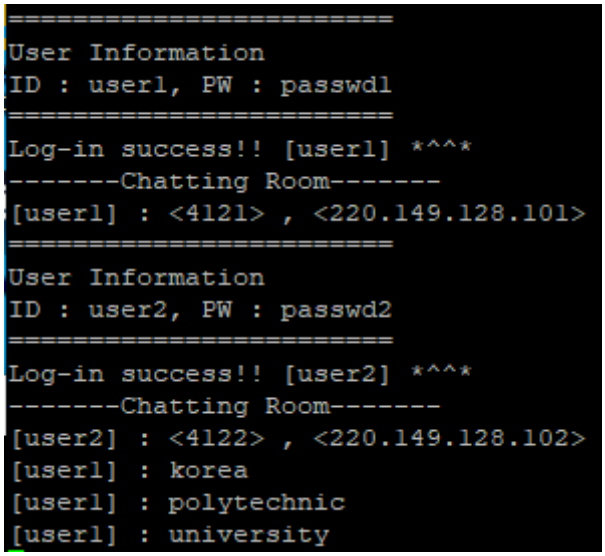
User1 과 User2 가 접속에 성공하여 채팅방이 생성되고 각 Client 의 Port 와 IP 를 알려주며 채팅 프로그램이 실행된다.

4. User1 이 채팅방 접속 후 Message 를 보낸다.



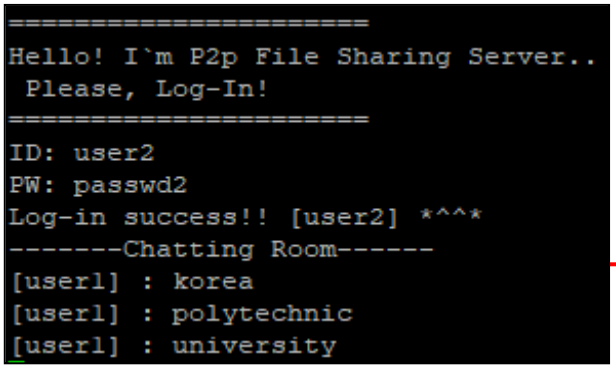
보낸 Message

5. User1 이 보낸 Message 는 Server 와 User2 에게 전달된다.



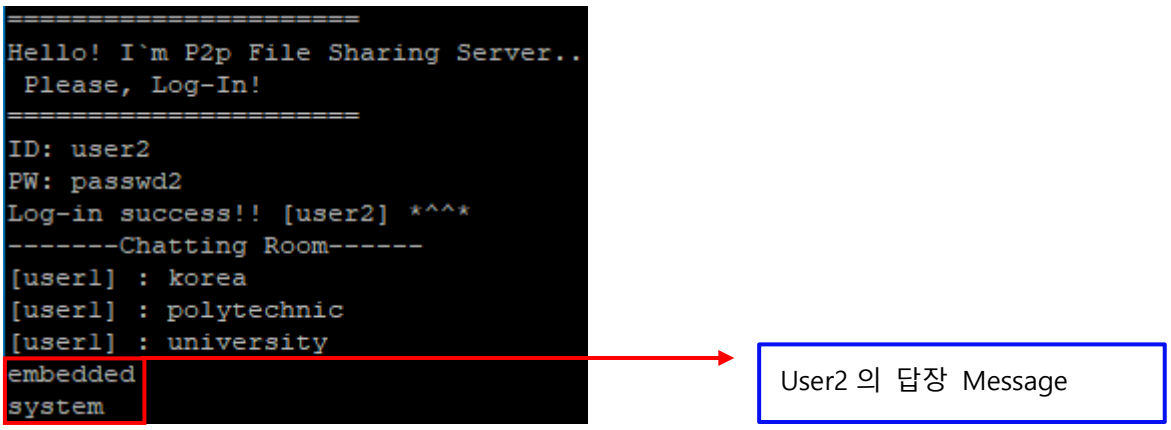
<-- 서버에서는 User1 이 보낸 메시지를 확인

6. User1 이 보낸 Message 를 User2 도 받고, 어디로부터 오는 메시지인지 메시지 앞에서 확인시켜 준다.

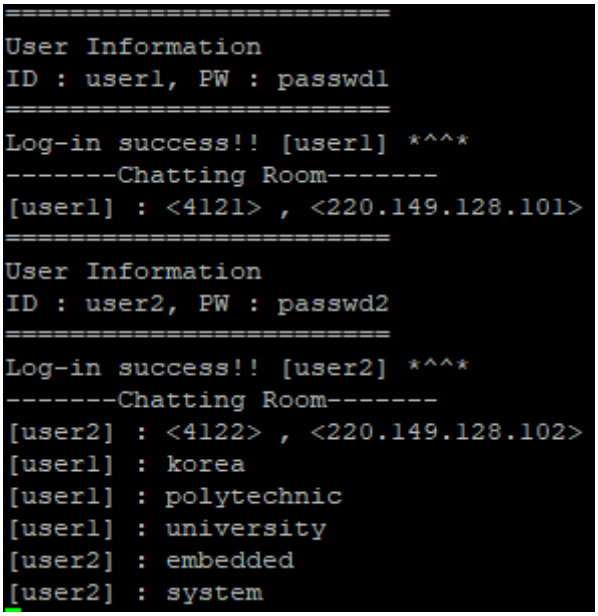


User1 으로부터 온 메시지임을 확인할 수 있다.

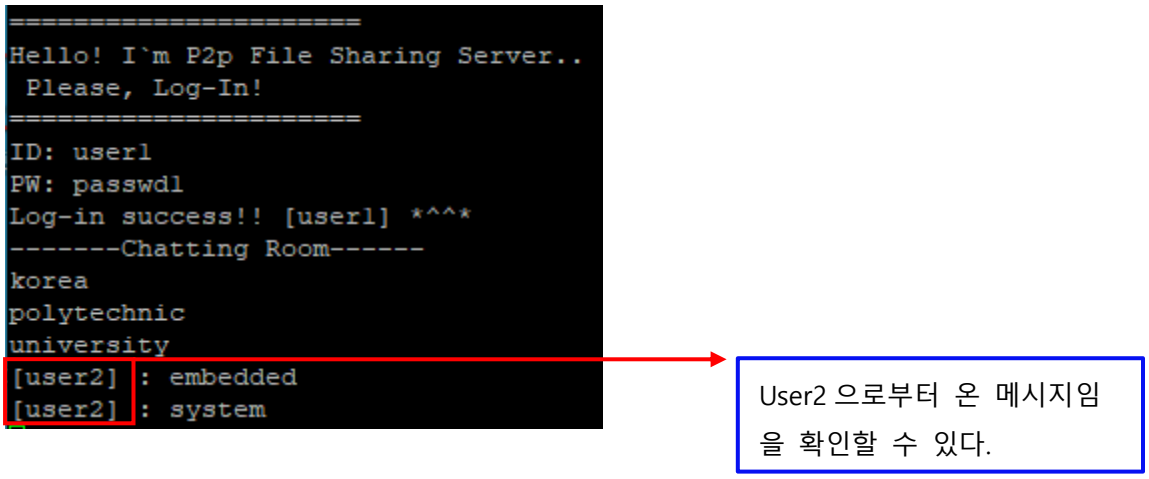
7. User2 가 User1 의 메시지를 받고 답장을 한다.



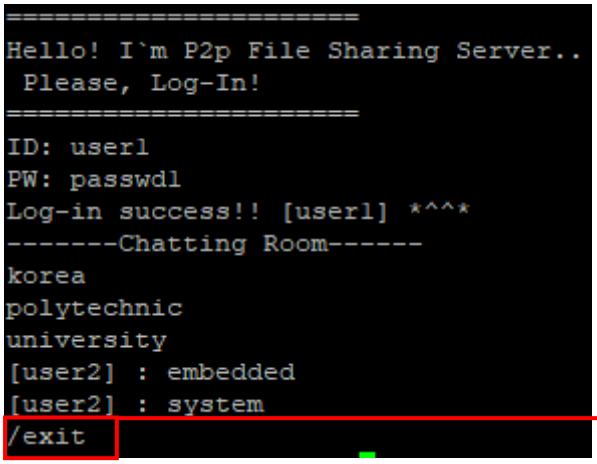
8. Server 에서는 User1 과 마찬가지로 User2 의 메시지를 받는다.



9. User1 도 User2 의 메시지를 받고 어디로부터 온 메시지인지 확인시켜준다.

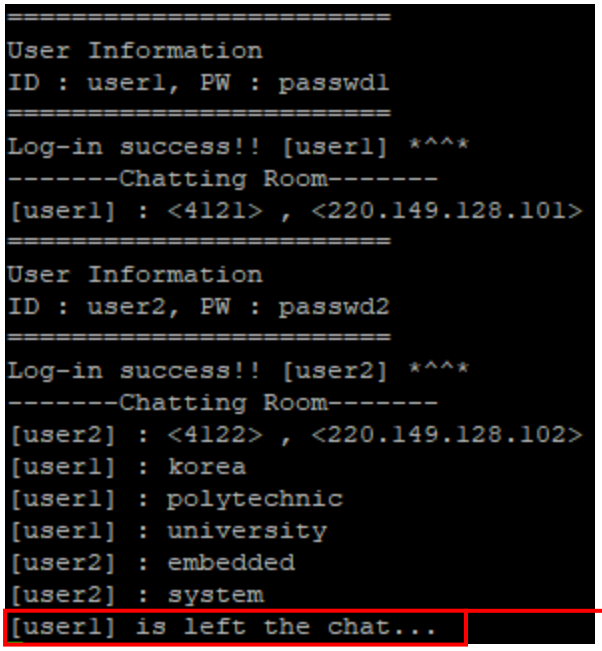


10. User1 이 채팅방에서 나가려고 시도한다.



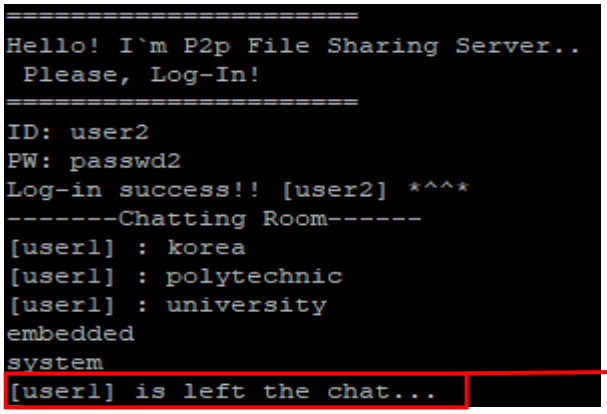
나가기 위해서는 /exit 라는 명령을 치면 채팅방에서 나가도록 설계했다.

11. 서버에서는 User1 이 종료하면 그에 대한 메시지를 받도록 하였다.

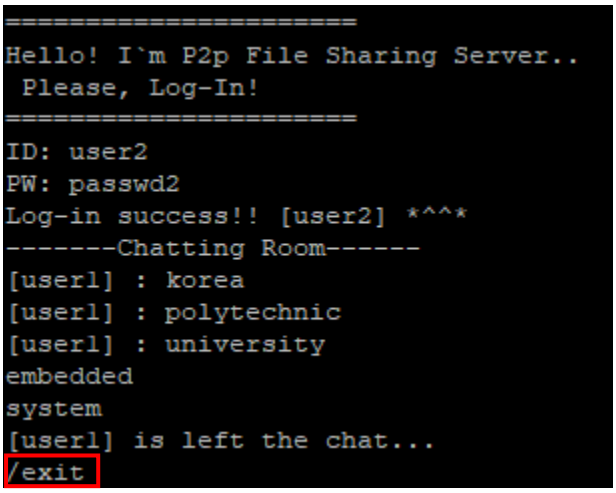


[서버] : User1 이 나간 것을 확인 할 수 있다.

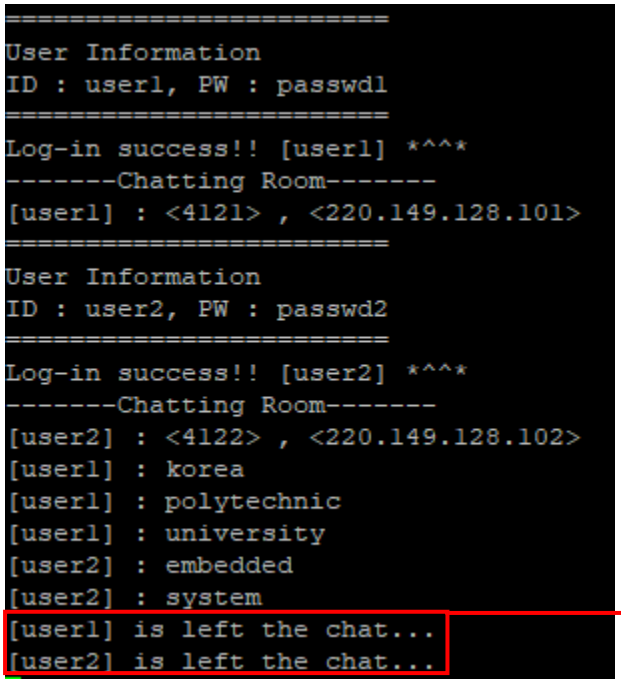
12. User1 이 종료하면 User2 에서도 User1 이 종료한 것을 확인할 수 있다.



13. User2 가 채팅프로그램을 종료하고자 한다. [/exit 사용]



14 . 서버에서는 User1 과 User2 가 채팅프로그램에서 나가는 것을 확인한다.



[서버] : User1 과 User2 둘다 나간 것을 확인할 수 있다

2. 코드분석

1. [Server.c]

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/select.h>
#define BUFF_SIZE 1024          //buff size
#define SEV_IP "220.149.128.100" //
#define SERV_PORT 4120
#define INIT_MSG "=====================\nHello! I`m P2p File Sharing Server..\n Please, Log-In!\n=====================\n" // hello msg
#define User1_ID "user1"
#define User1_PW "passwd1"
#define User2_ID "user2"
#define User2_PW "passwd2"
#define User3_ID "user3"
#define User3_PW "passwd3"
#define Wrong_PW "Log-in fail: Incorrect password\n\n"
#define FALSE 0
#define TRUE 1
int* mutex = NULL;
int* sockets = NULL;
int* cnt_socket = NULL;
int* clients_port = NULL;
char** clients_IP = NULL;
char** clients_PORT = NULL;
void mystop(int signo)
{
    munmap(mutex, sizeof *mutex);
    munmap(cnt_socket, sizeof *cnt_socket);
    munmap(sockets, sizeof(*sockets) * 5);
    exit(1);
}
int main(void)
{
    int flag = FALSE;
    int server_socket;  //server socket
    int client_socket;  //client socket
    int client_addr_size; // addr size
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    int rcv_byte;
    char buff[BUFF_SIZE];
    char id[20];
    char pw[20];
    char msg[BUFF_SIZE];
    int option;

    fd_set reads, cpy_reads;
    struct timeval timeout;
    int fd_max, fd_num;
    int i;

    sockets = mmap(NULL, sizeof(*sockets) * 5, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    cnt_socket = mmap(NULL, sizeof *cnt_socket, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    mutex = mmap(NULL, sizeof *mutex, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    *mutex = 1;
    signal(SIGINT, (void*)mystop);
```

1 ~ 24 줄 : 소켓통신을 하기 위한 함수를 쓰기 위해 include 하고, 서버 Port 와 IP 주소 그리고 User 들의 id 와 password 를 define 한다. 또한 인터페이스를 위한 문자열을 생성한다

25 ~ 30 줄 : 공유메모리와 서버소켓, 클라이언트 소켓 디스크립트 번호의 주소들을 저장하기 위한 변수들을 생성한다.

Sockets : user 들의 FD 를 저장하는 메모리

cnt_socket : FD 의 총 개수를 저장하는 메모리

clients_port (CHAR): user 의 port 번호를 저장하는 메모리

clients_ID : user 들의 ID 를 저장하는 메모리

clients_IP : user 들의 IP 를 저장하는 메모리

clients_PORT (INT): user 들의 ID 를 저장하는 메모리

---

31 ~ 37 줄 : 인터럽트 신호처리를 하기위한 함수(ctrl + C 로 종료시 함수가 실행되고 공유 메모리를 해제하는 역할을 한다.)

40 ~ 51 줄 : 필요한 변수들을 선언한다.

53 ~ 56 줄 : SELECT(다중 입출력 함수)를 사용하기 위한 변수 선언

58 ~ 60 줄 : 공유메모리 변수에 메모리를 할당한다. [소켓의 메모리를 5 개로 잡는다 (최대 5 명 수용가능)]

61 줄 : 공유메모리에 들어가는 프로세스들을 관리,제어하기 위해서 Mutex 라는 변수를 선언한다.

62 줄 : 인터럽트 처리 함수를 SET 한다(Ctrl + C)

```
63
64      server_socket = socket(AF_INET, SOCK_STREAM, 0);
65      option = 1;
66      setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, (char*)&option, sizeof(option));
67      if (-1 == server_socket) // create a server socket
68      {
69          printf("server socket fail\n");
70          exit(1);
71      }
72      else printf("Server-socket() sockfd is OK...\n");
73
74      server_addr.sin_family = AF_INET;
75      server_addr.sin_port = htons(SERV_PORT);
76      server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //32bit IPv4 address
77
78      memset(&(server_addr.sin_zero), 0, 8);
79      if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option)) < 0) // set a socket option
80      {
81          perror("setsockopt");
82          close(server_socket);
83          return -1;
84      }
85      if (-1 == bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr))) //Associate a socket with a p
    ort and address
86      {
87          perror("bind() fail!!");
88          exit(1);
89      }
90      else printf("Server-bind() is OK...\n");
91
92      if (-1 == listen(server_socket, 10)) { // listen for connection
93          perror("listen fail \n");
94          exit(1);
95      }
96      else printf("Server-listen() is OK...\n");
97
98      client_addr_size = sizeof(client_addr);
99      FD_ZERO(&reads);
100     FD_SET(server_socket, &reads);
101     fd_max = server_socket;
102     while (1)
103     {
104         cpy_reads = reads;
105         timeout.tv_sec = 5;
106         timeout.tv_usec = 50000;
107         if ((fd_num = select(fd_max + 1, &cpy_reads, 0, 0, &timeout)) == -1)
108         {
109             perror("select() error");
110             break;
111         }
112         if (fd_num == 0) continue;
113
114         for (i = 0; i < fd_max + 1; i++)
115         {
116             if (FD_ISSET(i, &cpy_reads))
117             {
118                 if (server_socket == i)
119                 {
120                     client_socket = accept(server_socket, (struct sockaddr*)&client_addr, &client_addr_size);
121
122                     if (client_socket == -1) perror("accept() error!");
123                     FD_SET(client_socket, &reads);
124                     if (fd_max < client_socket)
125                         fd_max = client_socket;
```

64 ~ 96 줄 : 클라이언트와 연결 하기 위한 소켓을 만들고 Binding 과 Listen 을 하는 구간.

99 줄 : reads 라는 fd_set 구조체를 초기화한다.

100 줄 : reads 에 서버 FD 를 SET 시킨다. FD = File Descriptor

101 줄 : fd_max 에 서버 FD 를 넣는다.

104 줄 : reads 를 cpy_reads 에 복사

107 줄 : select 는 system block 함수이므로 cpy_reads 중에 입출력 하고자 하는 FD 가 SET 될 때까지 Block 처리한다.

112 줄 : Block 대기시간이 지나면 fd_num 에 0 이 들어간후 다시 대기한다.

116 줄 : 특정 FD bit 가 set 되어있는 FD 번호를 찾는다.

118 줄 : set 되어있는 FD 와 서버 FD 가 동일하다면 이하 코드들을 실행시킨다.

120 줄 : Client 를 accept 하는 코드

123 줄 : fd_set 구조체에 accept 한 client FD 를 SET 시킨다.

124 ~ 125 줄 : fd_max 에 accept 한 FD 번호를 최신화

```
126                 send(client_socket, INIT_MSG, strlen(INIT_MSG) + 1, 0); // send hello msg
127                 rcv_byte = recv(client_socket, id, sizeof(id), 0); // recv_id
128                 rcv_byte = recv(client_socket, pw, sizeof(pw), 0); //recv_pw
129                 printf("========================\n");   //print information
130                 printf("User Information\n");
131                 printf("ID : %s, PW : %s\n", id, pw);
132                 printf("========================\n");
133
134                 /*-----------¾ÆÀÌµð °Ñ¹Ð¹øÈ£ °ñ±³-----------*/
135                 if ((strcmp(id, User1_ID) == 0) || (strcmp(id, User2_ID) == 0) || (strcmp(id, User3_ID) == 0))
136                 {
137                     if ((strcmp(pw, User1_PW) == 0) && (strcmp(id, User1_ID) == 0))
138                     {
139                         printf("Log-in success!! [%s] *^^*\n", id);
140                         printf("-------Chatting Room-------\n");
141                         send(client_socket, "o", strlen("o") + 1, 0);
142                         flag = TRUE;
143                     }
144                     else if ((strcmp(pw, User2_PW) == 0) && (strcmp(id, User2_ID) == 0))
145                     {
146                         printf("Log-in success!! [%s] *^^*\n", id);
147                         printf("-------Chatting Room-------\n");
148                         send(client_socket, "o", strlen("o") + 1, 0);
149                         flag = TRUE;
150                     }
151                     else if ((strcmp(pw, User3_PW) == 0) && (strcmp(id, User3_ID) == 0))
152                     {
153                         printf("Log-in success!! [%s] *^^*\n", id);
154                         printf("-------Chatting Room-------\n");
155                         send(client_socket, "o", strlen("o") + 1, 0);
156                         flag = TRUE;
157                     }
158                     else
159                     {
160                         printf("%s\n", Wrong_PW);
161                         send(client_socket, "p", strlen("p") + 1, 0);
162                         FD_CLR(client_socket, &reads);
163                         fd_max--;
164                         close(client_socket);
165                     }
166                 }
```

129 ~ 132 줄 : Client 로부터 ID 와 PW 를 입력 받고 그에 해당하는 메시지를 보낸다.

158 ~ 165 줄 : PW 입력이 잘못된 경우 다시 fd_set 구조체의 Client FD 변수를 clear 시킨 후 close 한다.

```
167                     else
168                     {
169                         printf("Log-in fail: ID Does not exist..\n\n");
170                         send(client_socket, "i", strlen("i") + 1, 0);
171                         FD_CLR(client_socket, &reads);
172                         fd_max--;
173                         close(client_socket);
174                     }
175                     if (flag == TRUE)
176                     {
177                         while (mutex <= 0);
178                         *mutex = 0;
179                         *(sockets + *cnt_socket) = client_socket;
180                         *cnt_socket += 1;
181                         *mutex = 1;
182                         flag == FALSE;
183                     }
184                 }
185                 else
186                 {
187                     rcv_byte = recv(i, msg, sizeof(msg), 0);
188                     rcv_byte = recv(i, id, sizeof(id), 0);
189                     if (strcmp(msg, "/exit") == 0)
```

167 ~ 173 줄 : ID 입력이 잘못된 경우 다시 fd_set 구조체의 Client FD 변수를 clear 시킨 후 close 한다.

175 ~ 183 줄 : ID 와 PW 가 정상 입력되면 mutex 를 통해 접근하는 프로세스들을 제어하고 client_socket 의 정보들을 공유메모리에 저장한다.

187 ~ 188 줄 : Client 로부터 id 와 pw 를 recv 한다.

```
190                     {
191                         while (*mutex <= 0);
192                         *mutex = 0;
193                         for (int j = 0; j < 5; j++)
194                         {
195                             if (*(sockets + j) == i)
196                             {
197                                 send(*(sockets + j), msg, strlen(msg) + 1, 0);
198                                 *(sockets + j) = 0;
199                                 *cnt_socket--;
200                             }
201                         }
202                         *mutex = 1;
203                         FD_CLR(i, &reads);
204                         close(i);
205                         sprintf(buff, "[%s] is left the chat...", id);
206                     }
207                     else
208                     {
209                         sprintf(buff, "[%s] : %s", id, msg);
210                     }
211                     printf("%s\n", buff);
212                     while (*mutex <= 0);
213                     *mutex = 0;
214                     for (int j = 0; j < 5; j++)
215                     {
216                         if (i != *(sockets + j))
217                         {
218                             send(*(sockets + j), buff, strlen(buff) + 1, 0);
219                         }
220                     }
221                     *mutex = 1;
222                 }
223             }
224         }
225     }
226
227     munmap(mutex, sizeof *mutex);
228     munmap(cnt_socket, sizeof *cnt_socket);
229     munmap(sockets, sizeof(*sockets) * 5);
230     close(server_socket);
231     return 0;
232 }
```

190 ~ 206 줄 : exit 를 했을 경우의 동작을 나타내는 코드.

191 ~ 205 줄 : [exit 했을경우] 공유 메모리에 있는 해당 클라이언트의 정보를 없애고 소켓을 닫아준다. 또한 어떤 user 가 나갔는지 sprintf 를 통해 알려준다.

207 ~ 221 줄 : Client 로부터 들어오는 메시지를 출력하고, 다른 Client 들에게 받은 메시지를 뿌려주는 코드.

216 ~ 219 줄 : 자기 자신을 제외한 다른 Client 에게 메시지를 send 한다.

227 ~ 230 줄 : 공유메모리를 사용하는 변수들이 메모리를 지속적으로 차지하면 안되므로 해제시켜주고 서버소켓을 close 해줌으로써 프로그램을 종료한다.

2.[Client.c]

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/mman.h>
#define MAXLINE 1024              //def buffsize
#define SERV_IP "220.149.128.100"
#define SERV_PORT 4120
#define MY_IP "220.149.128.102"
#define MY_PORT "4122"
#define OK "o" // def connect success
#define WP "p" // def passwd error
#define WI "i" // def id error
int main(int argc, char **argv) {
    struct sockaddr_in clientaddr;
    int client_sockfd;
    int client_len;
    int rcv_byte;
    char buf[MAXLINE];   //buff, buffer size
    char id[20];         //  id recieve buff
    char pw[20];         // passwd recieve bff
    char last[MAXLINE];  // error or success buff
    char msg[MAXLINE];
    pid_t pid;
    if ((client_sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {  // create server socket
        perror("error : ");
        return 1;
    }
    else printf("Client_socket() sockfd is OK...\n");

    clientaddr.sin_family = AF_INET;
    clientaddr.sin_addr.s_addr = inet_addr(SERV_IP); //init server ip
    clientaddr.sin_port = htons(SERV_PORT); // init server port
    memset(&(clientaddr.sin_zero), 0, 8);

    client_len = sizeof(clientaddr);

    if (connect(client_sockfd, (struct sockaddr*)&clientaddr, client_len) == -1) { //connect part
        perror("connect error : ");
        exit(1);
    }
    else printf("Client_connect() is OK...\n\n");

    rcv_byte = recv(client_sockfd, buf, sizeof(buf), 0);
    printf("%s", buf);

    printf("ID: ");
    scanf("%s", id);
    send(client_sockfd, id, strlen(id) + 1, 0);    // send id

    printf("PW: ");
    scanf("%s", pw);
    send(client_sockfd, pw, strlen(pw) + 1, 0);    // send pw
```

1 ~ 17 줄 : 서버와 소켓 통신을 하기 위한 헤더파일 선언과 클라이언트 자신의 Port 와 IP 를 저장하기 위한 define 문

19 ~ 27 줄 : 소켓번호와 id, pw 등을 받기 위해 변수들을 선언한다.

28 줄 : 자식프로세스를 생성하기 위해 pid 변수를 선언한다.

29 ~ 46 줄: 서버와 연결하기 위한 connect 과정.

48 ~ 57 줄 : 48 줄에서 메인 인터페이스에 대한 문자열을 받고 출력 후 ID 와 PW 를 입력하고, 입력한 ID 와 PW 문자열을 서버로 send 한다.

```
58
59      rcv_byte = recv(client_sockfd, last, sizeof(last), 0);   //recieve errors
60
61      if (strcmp(last, OK) == 0)    //connect success
62      {
63          printf("Log-in success!! [%s] *^^*\n", id);
64          printf("-------Chatting Room------\n");
65      }
66      else if (strcmp(last, WP) == 0)    //wrong passwd
67      {
68          printf("Log-in fail: Incorrect password...\n");
69          close(client_sockfd);
70          exit(1);
71      }
72      else if (strcmp(last, WI) == 0)    //worng id
73      {
74          printf("Log-in fail: ID does not exist...\n");
75          close(client_sockfd);
76          exit(1);
77      }
78      send(client_sockfd, MY_IP, strlen(MY_IP) + 1, 0);
79      send(client_sockfd, MY_PORT, strlen(MY_PORT) + 1, 0);
80      pid = fork();
81      // buf : 1024 msg : 1024
82      if (pid == 0)
83      {
84          while (1)
85          {
86              fgets(buf,MAXLINE,stdin);
87              buf[strlen(buf)-1] = '\0';
88              send(client_sockfd, buf, strlen(buf) + 1, 0);
89              send(client_sockfd, id, strlen(id) + 1, 0);
90              if (strcmp(buf, "/exit") == 0)
91              {
92                  close(client_sockfd);
93                  exit(1);
94              }
95          }
96      }
97      else if (pid > 0)
98      {
99          while (1)
100         {
101             rcv_byte = recv(client_sockfd, msg, sizeof(msg), 0);
102             if (strcmp(msg, "/exit") == 0)
103             {
104                 close(client_sockfd);
105                 exit(1);
106             }
107             else
108                 printf("%s\n", msg);
109         }
110     }
111     return 0;
112 }
```

59 줄 : 서버로부터 error 메시지를 받는다.

61 ~ 77 줄: 입력한 ID 와 PW 가 서버에서 지정한 ID/PW 와 일치하거나 불일치 할 때의 상태를 보여주는 코드.

78 ~ 89 줄 : 서버에게 Client 본인의 Port 와 IP 를 보낸다.

80 줄 : 자식프로세스 생성을 위한 Fork 문

82 줄 : [자식프로세스 시작] ----→ send 프로세스

86 ~ 87 줄 : 메시지를 입력하고 메시지의 가장 끝 문자를 '₩0'으로 만든다.

88 ~ 94 줄 : /exit 를 입력하기 전까지는 입력한 메시지들을 계속 send 한다

97 줄 : [부모프로세스 시작] ----→ recv 프로세스

101 줄 : 서버로부터 오는 메시지를 계속 받는 recv 동작을 한다.

102 ~ 108 줄 : 오는 메시지가 /exit 가 아니라면 계속 받는동작을 하고 /exit 가 들어오게 되면 클라이언트 소켓을 닫고 exit 한다. ----→ [종료]