

1. pthread_mutex_lock / pthread_mutex_unlock을 사용한 코드 부분과 이유

통상 클라이언트와 서버로 나뉘는 과정을 하나로 합쳐서 구현했기에 공유메모리와 멀티스레딩 기술이 필요했습니다. 사용자는 메시지를 씌고 동시에 다른 사용자로부터 메시지를 읽을 수도 있어야 하기에 읽기 전용 스레드 `usrthread`와 화면 출력 전용 스레드 `display_thread`를 사용했습니다. 이때 pthread_mutex_lock과 unlock을 통해 동기화 시켜주어야 상호통신에 문제가 생기지 않습니다. 여러 스레드가 동시에 메시지 배열에 접근해 write 동작을 수행한다면 충돌이 발생할 수 있기 때문입니다.

`pthread_mutex_lock(&mutex);` 함수는 뮤텁스를 획득하여 공유 데이터에 대한 접근을 막습니다. 그 후 해당 스레드는 내부의 작업을 수행하게 됩니다. 뮤텁스를 획득하지 못한 여타 스레드들은 해당 스레드가 `pthread_mutex_unlock(&mutex);`로 뮤텁스를 해제하기 까지 대기하게 됩니다. 이 과정을 통해 스레드 간, 배열에 대한 접근을 동기화하여 데이터 일관성을 유지할 수 있게 됩니다.

2. 전체 프로그램 소스 및 주석

(1) `chatshm.h` 헤더파일

```
#ifndef __CHAT_SHARE_MEMORY_H__
#define __CHAT_SHARE_MEMORY_H__
#include <pthread.h>
#include <ncurses.h>
typedef struct chatInfo{
    char userlist[3][10];    //유저목록
    int user_index;          //유저 인덱스용 변수
    char messages[10][100]; //메세지 저장
    int message_index;       //메세지 인덱스용 변수
} CHAT_INFO;

#endif//__CHAT_SHARE_MEMORY_H__
```

공유메모리에 연결될 구조체입니다. 주석의 내용들을 담고 있습니다.

(2) `csechat.c`의 전처리와 전역변수들

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#include <ncurses.h>
#include <pthread.h>
#include "chatshm.h"

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
WINDOW * sw1, *sw2, *sw3, *sw4;
CHAT_INFO *cf = NULL;
```

`chatshm.h`를 헤더파일로 삼고, 스레드 동기화에 필요한 mutex를 전역으로 선언했습니다.

(3) `csechat.c`의 main 함수

```
int main(int argc, char *argv[]) { //argv[1]에 유저id를 입력 받습니다.
    initscr();                      //ncurses 초기화 단계입니다.
    noecho();
    cbreak();
    keypad(stdscr, TRUE);
    int height, width;
    getmaxyx(stdscr, height, width);
    int sub_height = height / 2;
    int sub_width = width / 2;
    sw1 = newwin(sub_height, sub_width, 0, 0); //WINDOW *는 전역에 선언되었고
    sw2 = newwin(sub_height, sub_width, 0, sub_width); //서브윈도우는 4개로 분할 했습니다.
    sw3 = newwin(sub_height, sub_width, sub_height, 0);
    sw4 = newwin(sub_height, sub_width, sub_height, sub_width);
    box(sw1, 0, 0); //서브윈도우 간 테두리를 생성했습니다.
    box(sw2, 0, 0);
    box(sw3, 0, 0);
    box(sw4, 0, 0);
    mvprintw(sw1, 1, 1, "Chatting..\n");
    mvprintw(sw2, 1, 1, "Logged in Users\n");
    mvprintw(sw3, 1, 1, "input line\n");
    mvprintw(sw4, 1, 1, "Chat Info\n");
    mvprintw(sw4, 5, 5, "unlucky kakaotalk");

    wrefresh(sw1); wrefresh(sw2); wrefresh(sw3); wrefresh(sw4); //위 내용들을 반영하기위해 refresh
```

`gcc -o csechat csechat.c -pthread -lncurses`로 컴파일 후 `./csechat user1`를 입력하면 main 함수의 argv[1]에 user1이라는 유저아이디를 받습니다.

ncurses를 사용해 채팅프로그램의 초기화면을 구성하는 과정입니다. 4개의 서브윈도우를 생성하고 2X2 형태로 구성했습니다. 각 서브윈도우 간 테두리를 생성하고 첫 서브윈도우는 채팅 내역, 두 번째 서브윈도우는 유저 목록, 세 번째 서브윈도우는 입력창, 마지막 윈도우는 채팅프로그램의 이름을 보이도록 구성했습니다.

(3) `csechat.c`의 main 함수 (계속)

```
pthread_t usrthread, display_thread;
int shmid;
bool init_flag = true;
void *shmaddr = (void *)0;
shmid = shmget((key_t)3936, sizeof(CHAT_INFO), 0666|IPC_CREAT|IPC_EXCL); //공유메모리를 연결합니다
if (shmid < 0)
{
    init_flag = false; //최초 연결자가 아니면 init_flag를 false로 바꿔줍니다
    shmid = shmget((key_t)3936, sizeof(CHAT_INFO), 0666);
    shmaddr = shmat(shmid, (void *)0, 0666);
    if (shmaddr < 0)
    {
        perror("shmat attach is failed : ");
        exit(0);
    }
}

shmaddr = shmat(shmid, (void *)0, 0666);
cf = (CHAT_INFO *)shmaddr; //구조체 cf를 shamaddr과 연결 시킵니다

if (init_flag) //공유메모리를 attach하는 최초의 양수 shmid가
{ //예상치 못한 sig fault를 방지하기 위해 헤더 변수들을 초기화를 시켜줍니다
    cf->user_index = 0;
    cf->message_index = 0;
    for (int i = 0; i < 3; i++) strcpy(cf->userlist[i], "");
    for (int i = 0; i < 10; i++) strcpy(cf->messages[i], "");
}

//각 프로세스는 usrthread로서 메시지를 읽고
pthread_create(&usrthread, NULL, read_messages, (void *) argv[1]);
pthread_create(&display_thread, NULL, display_messages, (void *) argv[1]);
//display_thread로서 메시지를 출력하도록 멀티스레딩합니다

pthread_join(usrthread, NULL);
pthread_join(display_thread, NULL);

return 0;
}
```

`chatshm.h`에 선언된 구조체를 공유메모리로 연결하는 작업입니다. 최초의 연결자는 init_flag를 true로 설정하고 이후의 연결(shmid<0)에서는 false로 설정해 공유메모리의 구조체를 초기화하여서 쓰레기값이 들어가지 않도록 했습니다. 예상치 못한 segmentation fault를 막아줍니다.

(4) `csechat.c`의 `read_messages()` 함수

```
void *read_messages(void *arg)           //유저의 입력 스레드를 관리하는 함수
{
    char curmsg[40];
    /** for subwin 2 */
    if(cf->user_index > 2) {mvwprintw(sw4, 9, 9, "too many users"); exit(0);}
    strcpy(cf->userlist[cf->user_index], (char*) arg); //입력받은 arg를 유저목록에 추가시킵니다
    cf->user_index++;                                //다음 유저를 저장하기 위해 인덱스를 증가시킵니다

    while (1)
    {
        /** for subwin 3 */
        echo();
        mvwgetnstr(sw3, 5, 5, curmsg, 40); //사용자 입력이 서브윈도우3에 보이도록 echo를 설정합니다
        if(strcmp(curmsg, "quit\n") == 0) break;
        wclear(sw3); //입력이 끝난 내용은 채팅프로그램 특성상
        mvwprintw(sw3, 1, 1, "input line\n"); //지워줘야 하기에 clear시킨후 서브윈도우를 다시 그려줍니다
        box(sw3, 0, 0);
        wrefresh(sw3);

        pthread_mutex_lock(&mutex); //입력받은 메시지를 배열에 넣을 때 뮤텍스 락을 겁니다
        if(cf->message_index >= 10) //메세지 10개가 모두 차면
        {
            for(int i = 0; i < 9; i++) //뒤 원소를 한칸씩 앞으로 당기고
            {
                strcpy(cf->messages[i], cf->messages[i+1]);
            } //배열의 마지막에 현재 메세지를 저장합니다
            snprintf(cf->messages[9], sizeof(char) * 100,
                "[%s]: %s\n", (char *)arg, curmsg);
        }
        else
        {
            //메세지가 10개 이하이면 message_index를 증가시키며 순차적으로 저장합니다
            snprintf(cf->messages[cf->message_index], sizeof(char) * 100,
                "[%s]: %s\n", (char *)arg, curmsg);
            cf->message_index++;
        }
        pthread_mutex_unlock(&mutex); //저장 과정이 끝나면 락을 풉니다
    }
    return NULL;
}
```

curmsg[40]는 사용자가 입력한 메시지를 저장합니다. 채팅에 참여하는 유저가 3명 이하로 유지되도록 했습니다. 인자 arg는 유저 아이디이며 이를 userlist에 대응되는 user_index 순번에 차례로 저장했습니다. mvwgetnstr을 통해 특정 위치에서 입력을 받도록 했고 사용자가 타이핑 하는 내용을 echo()를 통해 보이도록 했습니다. 메시지 10개가 모두 차면 스크롤 업처럼 보이도록 기능을 구현했고 이 과정들은 모두 mutex를 획득한 스레드만 수행될 수 있게 하여 데이터 일관성을 유지했습니다.

(5) `csechat.c`의 `display_message()` 함수

```
void *display_messages(void *arg)           //출력 스레드를 관리하는 함수
{
    /** for subwin1 and subwin2 **/
    while(1)
    {
        pthread_mutex_lock(&mutex);        //출력시 mutex 락을 겁니다
        for (int i = 0, h = 2; i < cf->message_index; i++, h++)
        {
            mvwprintw(sw1, h, 1, "%s\n", cf->messages[i]); //서브윈도우1에 메세지 배열을 출력합니다
        }
        wrefresh(sw1);
        for (int i = 0, h = 2; i < 3; i++, h++)
        {
            if(strcmp(cf->userlist[i], "") == 0) continue; //서브윈도우2에 유저 목록을 출력합니다
            mvwprintw(sw2, h, 5, "%s..\n", cf->userlist[i]);
        }
        wrefresh(sw2);
        usleep(100000);
        pthread_mutex_unlock(&mutex);      //출력이 끝나면 unlock 합니다
    }
    return NULL;
}
```

화면을 출력하는 함수입니다. 메시지 배열에 저장된 내용을 ncurses 위치(h)에 맞게 출력하고 wrefresh해서 서브윈도우1 화면에 반영해줍니다. 서브윈도우 2에서는 유저 목록을 순차적으로 출력하고 wrefresh 해줍니다. 이 과정 역시 pthread_mutex_lock을 사용해 동기화 시켜 주었습니다.

(6) `shmremove.c` 소스파일

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#include "chatshm.h"

int main()
{
    int shmid;

    // 공유메모리 공간 만들
    shmid = shmget((key_t)3936, sizeof(CHAT_INFO), 0666);

    if (shmid == -1)
    {
        perror("shmget failed : ");
        exit(0);
    }

    if (shmctl( shmid, IPC_RMID, 0) < 0)
    {
        printf( "Failed to delete shared memory\n");
        return -1;
    }
    else
    {
        printf( "Successfully delete shared memory\n");
    }
    return 0;
}
```

이전에 연결된 공유메모리 주소를 `./shmremove`를 통해 해제하는 용도의 파일입니다.

3. 실행과정 단계별 기술

(1) 터미널 준비

```

pyeon@pyeon-VirtualBox: ~/UniPro/hw02
pyeon@pyeon-VirtualBox:~$ cd UniPro/hw02
pyeon@pyeon-VirtualBox:~/UniPro/hw02$ ls
201824607-pyeon.zip  csechat.c          csechatremove.o  ncurses          shmwrite
chat                 csechatread.c     csechatshm.h    ncurses.c       shmwrite.c
chat.c              csechatread.o     csechatwrite.c  shmread.c       shmwrite
chatshm.h           csechatremove.c   csechatwrite.o  shmread.c       temp
chatshm.h.gch       csechatremove.o   csechatwrite.o  shmremove.c     temp.c
csechat             csechatremove.c   Makefile        shmremove.c
pyeon@pyeon-VirtualBox:~/UniPro/hw02$ gcc -o csechat csechat.c -lncurses -pthread
pyeon@pyeon-VirtualBox:~/UniPro/hw02$ ./csechat pyeon

pyeon@pyeon-VirtualBox:~$ cd UniPro/hw02/
pyeon@pyeon-VirtualBox:~/UniPro/hw02$ ./cse chat lee

```

3개의 터미널을 띄워 3명의 채팅 사용자를 준비합니다. `./csechat [username]`으로 유저 아이디를 입력해 프로그램에 들어갈 수 있습니다.

(2) 첫 번째 유저와 두 번째 유저가 프로그램 입장

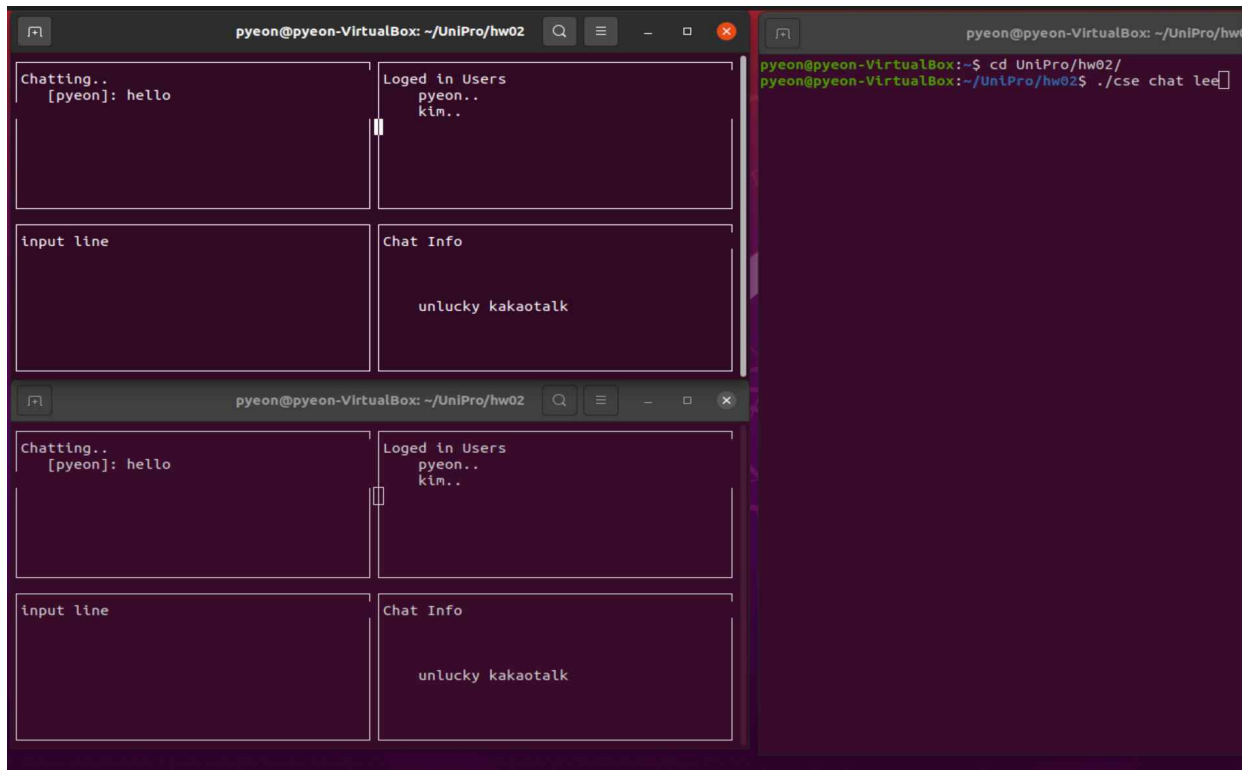
```

pyeon@pyeon-VirtualBox: ~/UniPro/hw02
pyeon@pyeon-VirtualBox:~$ cd UniPro/hw02/
pyeon@pyeon-VirtualBox:~/UniPro/hw02$ ./cse chat lee

```

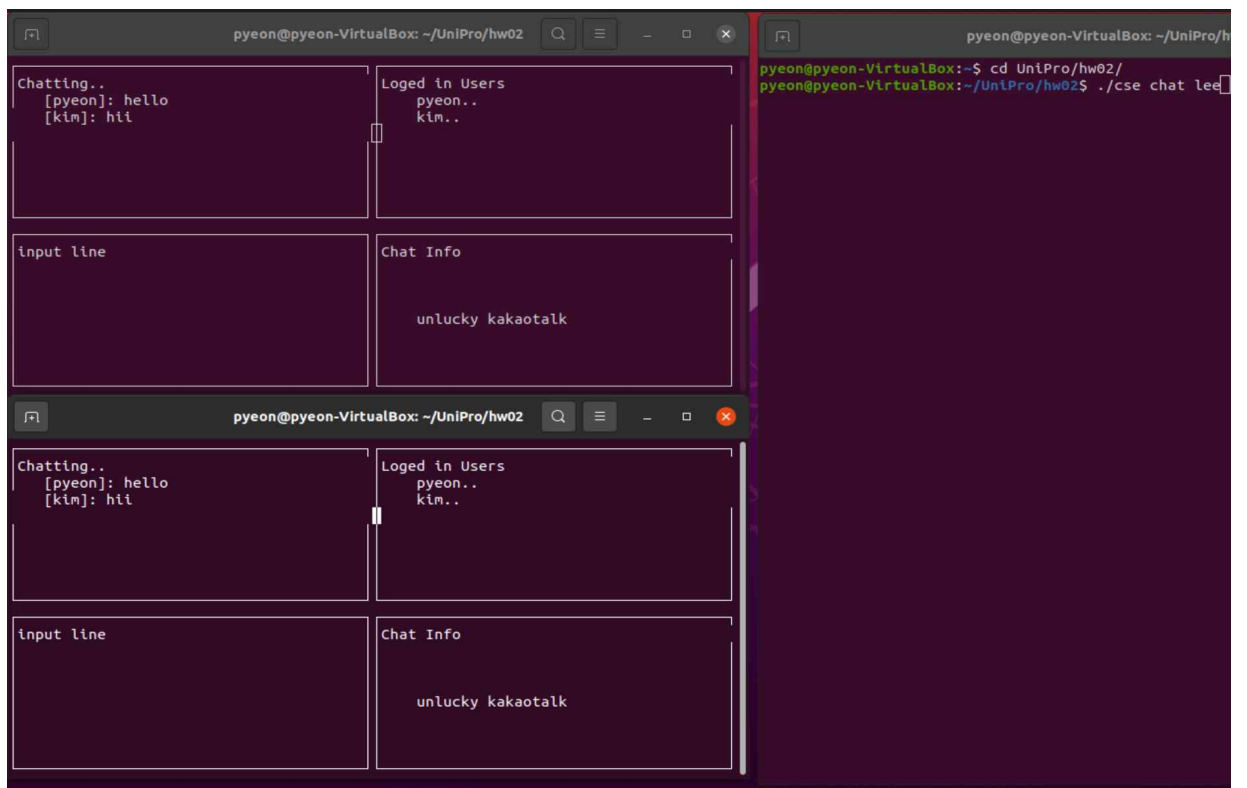
첫 유저가 'hello'를 입력하고 엔터를 치기 전 모습입니다.

(3) 엔터 후

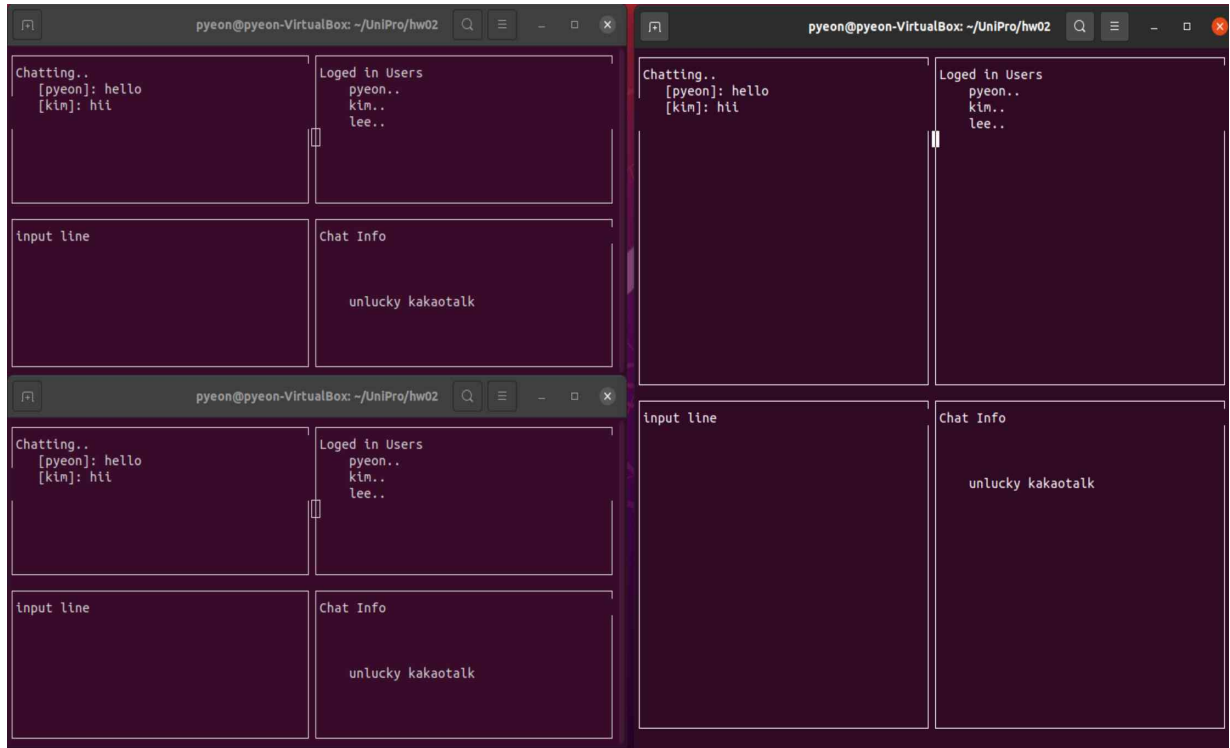


입장한 두 유저의 화면에 실시간으로 메시지가 출력된 모습입니다.

(4) 두 번째 유저의 메시지 입력

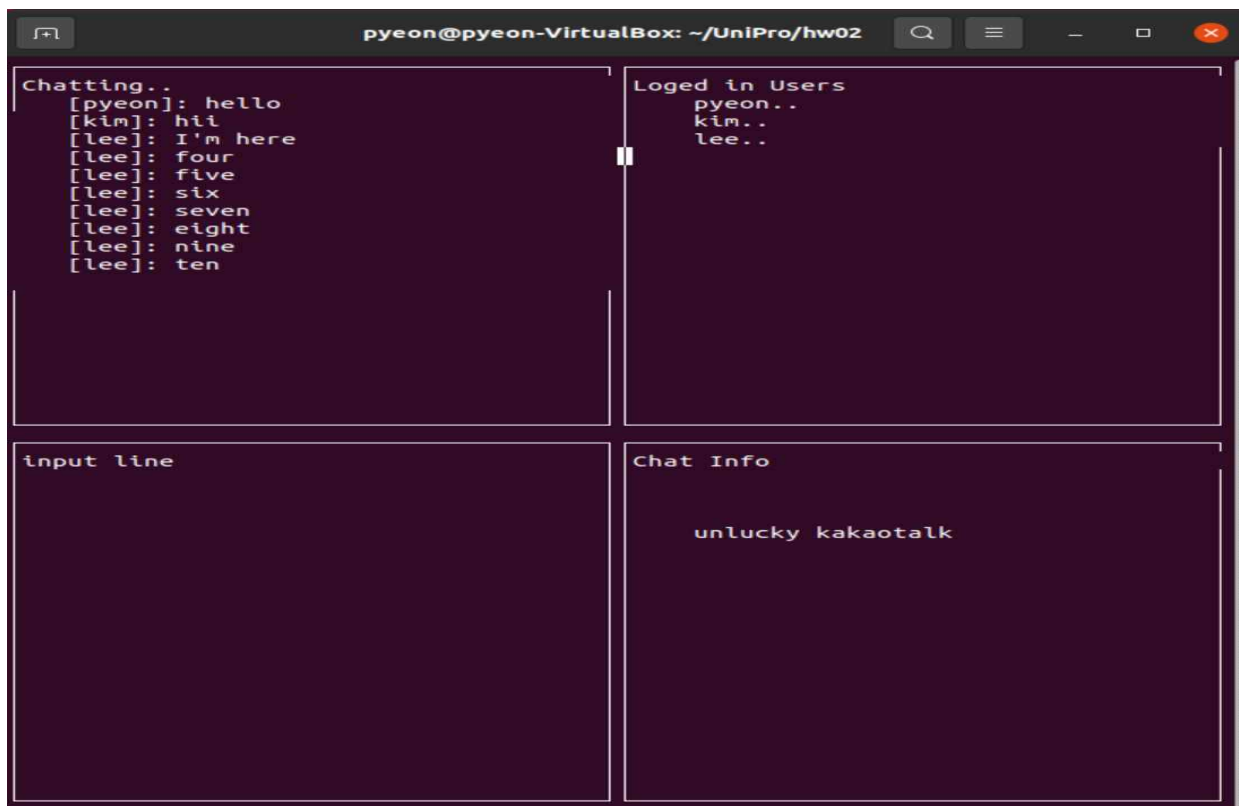


(5) 세 번째 유저의 입장

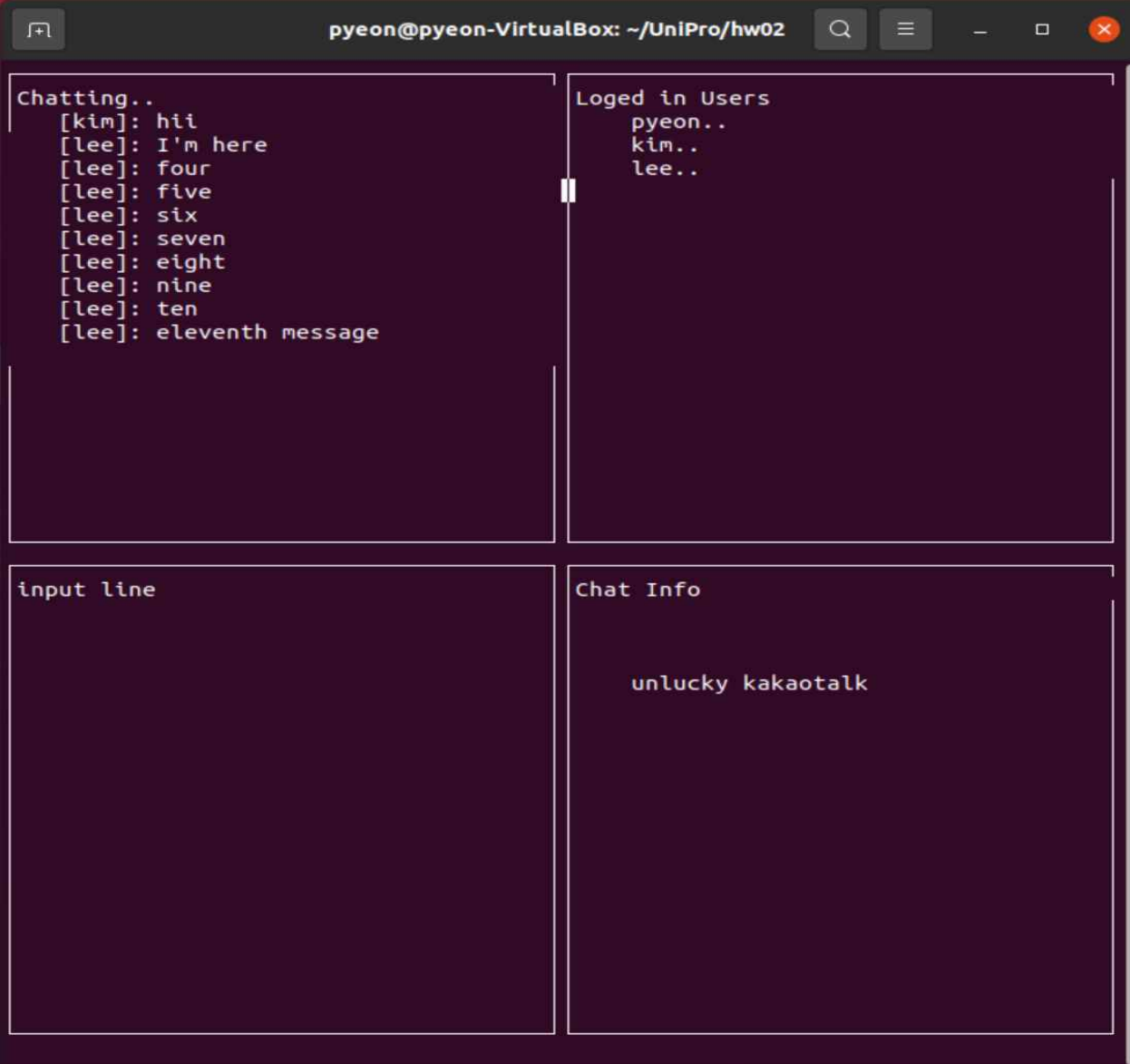


입장 후 두 번째 서버윈도우 유저목록에 이용자가 추가된 것을 확인할 수 있습니다.

(6) 메시지를 10개 까지 입력



(7) 11번째 메시지 입력



```
pyeon@pyeon-VirtualBox: ~/UniPro/hw02
```

Chatting..

[kim]: hii
[lee]: I'm here
[lee]: four
[lee]: five
[lee]: six
[lee]: seven
[lee]: eight
[lee]: nine
[lee]: ten
[lee]: eleventh message

Logged in Users

pyeon..
kim..
lee..

input line

Chat Info

unlucky kakaotalk

11번째 메시지가 들어오고 첫 메시지가 스크롤 업 되어 사라진 모습을 확인할 수 있습니다.