1. “Named semaphore 를 사용하여 Shared Memory 에서 공유되는 자원을 보호한다” 가 사용된 코드를 기술하고 사용한 이유를 설명하시오

(read.message 함수)

```
while(1)
{
    /** for subwin 3 **/
    echo();
    mvwgetnstr(sw3, 5, 5, curmsg, 40);
    wclear(sw3);
    mvwprintw(sw3, 1, 1, "input line\n");
    box(sw3, 0, 0);
    wrefresh(sw3);

    sem_wait(sem);  // sem 값이 1일때 감소시키고 다음 코드 실행

    // 메세지가 10개 초과될 시 한 칸씩 앞으로 배치 후 새 메세지 추가
    if(cf->message_index >= 10)
    {
        for (int i = 0; i < 9; i++)
        {
            strcpy(cf->messages[i], cf->messages[i+1]);
        }
        snprintf(cf->messages[9], sizeof(char) * 60,
                "[%s]: %s", (char *)arg, curmsg);
    }
    else
    {
        snprintf(cf->messages[cf->message_index], sizeof(char) * 60,
                "[%s]: %s", (char *)arg, curmsg);
        cf->message_index++;
    }
    sem_post(sem);  // sem 값을 1 증가시켜 동기화
}
```

read_message() 함수에서 sem_wait(sem);을 통해 초기 세마포어값 1이 들어오면 0으로 감소 시키고 입력된 메시지를 메시지 리스트에 추가하는 작업을 수행한 뒤 sem_post(sem);으로 sem 값을 1로 증가시켜 동기화를 유지했습니다. 입력받은 메시지를 공유메모리에 적재된 메시지 배열에 넣을 때, 다른 프로세스가 이 메시지 배열에 접근하여 메시지 간 순서나 내용을 충돌시킬 우려가 있기에 세마포어로 공유되는 자원을 보호했습니다. sem_wait와

sem_post 사이의 코드 블록은 동기화되어 아토믹하게 동작됩니다.

(display_message 함수)

```
/** for subwin1 and subwin2 **/
while(1)
{
    sem_wait(sem);  // sem 값을 1 감소시킴

    wclear(sw1);
    box(sw1, 0, 0);
    mvwprintw(sw1, 1, 1, "Chatting..");
    for (int i = 0, h = 2; i < cf->message_index; i++, h++)
    {
        mvwprintw(sw1, h, 3, "%s", cf->messages[i]);
    }
    wrefresh(sw1);

    wclear(sw2);
    mvwprintw(sw2, 1, 1, "Loged in Users\n");
    box(sw2, 0, 0);
    // 로그인된 유저 목록 출력
    for (int i = 0, h = 2; i < 3; i++, h++)
    {
        if(strcmp(cf->userlist[i],"") == 0) {h--; continue;}
        mvwprintw(sw2, h, 5, "%s..", cf->userlist[i]);
    }
    wrefresh(sw2);

    sem_post(sem);  // sem 값을 1 증가시켜 동기화
    usleep(50000);  // 화면 반짝임 없앰
}
```

display_message() 함수에서 채팅 내역을 출력하는 서브윈도우 1과 로그인된 유저목록을 출력하는 서브윈도우 2를 화면에 반영하는 동작을 sem_wait와 sem_post 사이에 배치 시켜 동기화 했습니다. 이로써 메시지를 read하는 행위와 읽은 메시지를 display하는 행위는 원자성이 보장 되게 됩니다.

2. “채팅 사용자가 Ctrl+C 입력시 접속자 수를 1 개씩 감소시키고, 접속자 수가 0 이면 프로그램을 종료시킨다.“가 사용된 코드를 기술하고 사용한 이유를 설명하시오

(main 함수)

```
// 시그널 핸들러 함수 등록
signal(SIGINT, sigint_handler);
```

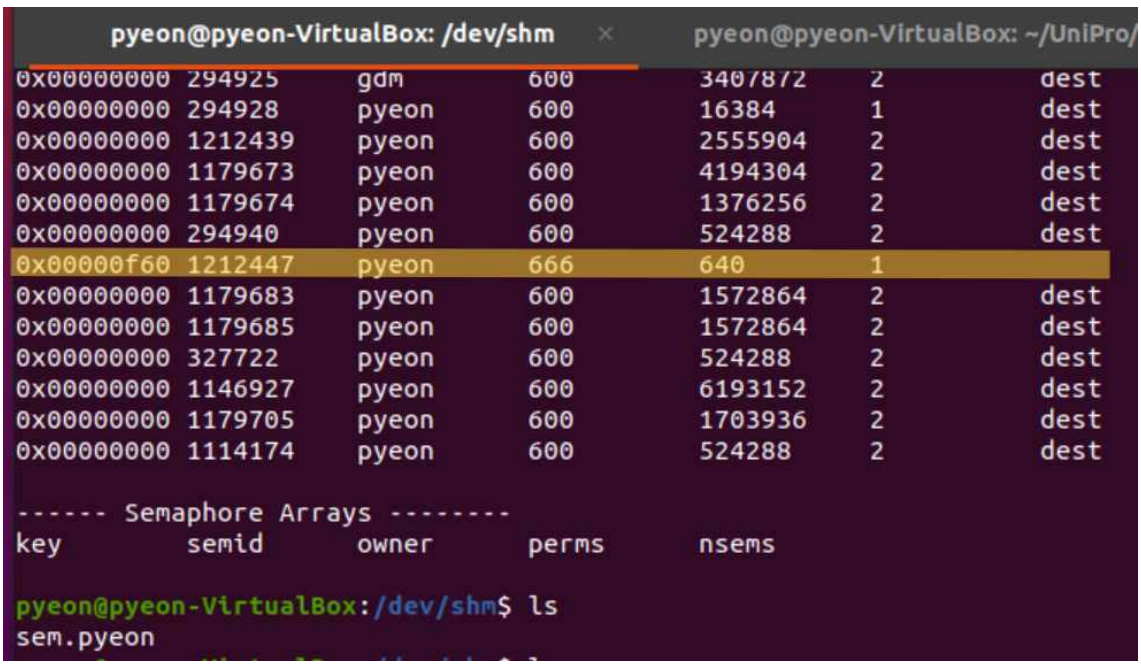
```
// 메인 함수는 스레드들을 기다려 먼저 종료되지 않게 함
pthread_join(usrthread, NULL);
pthread_join(display_thread, NULL);

// 시그널 발생 후 스레드들이 종료되면 sem을 unlink시킴
sem_unlink(argv[1]);
sem_close(sem);
// 프로세스 내 유저수가 0명이 되면 공유메모리 해제를 진행
if(cf->user_no == 0) system("./shmremove");
endwin();

return 0;
```

(sigint_handler 함수)

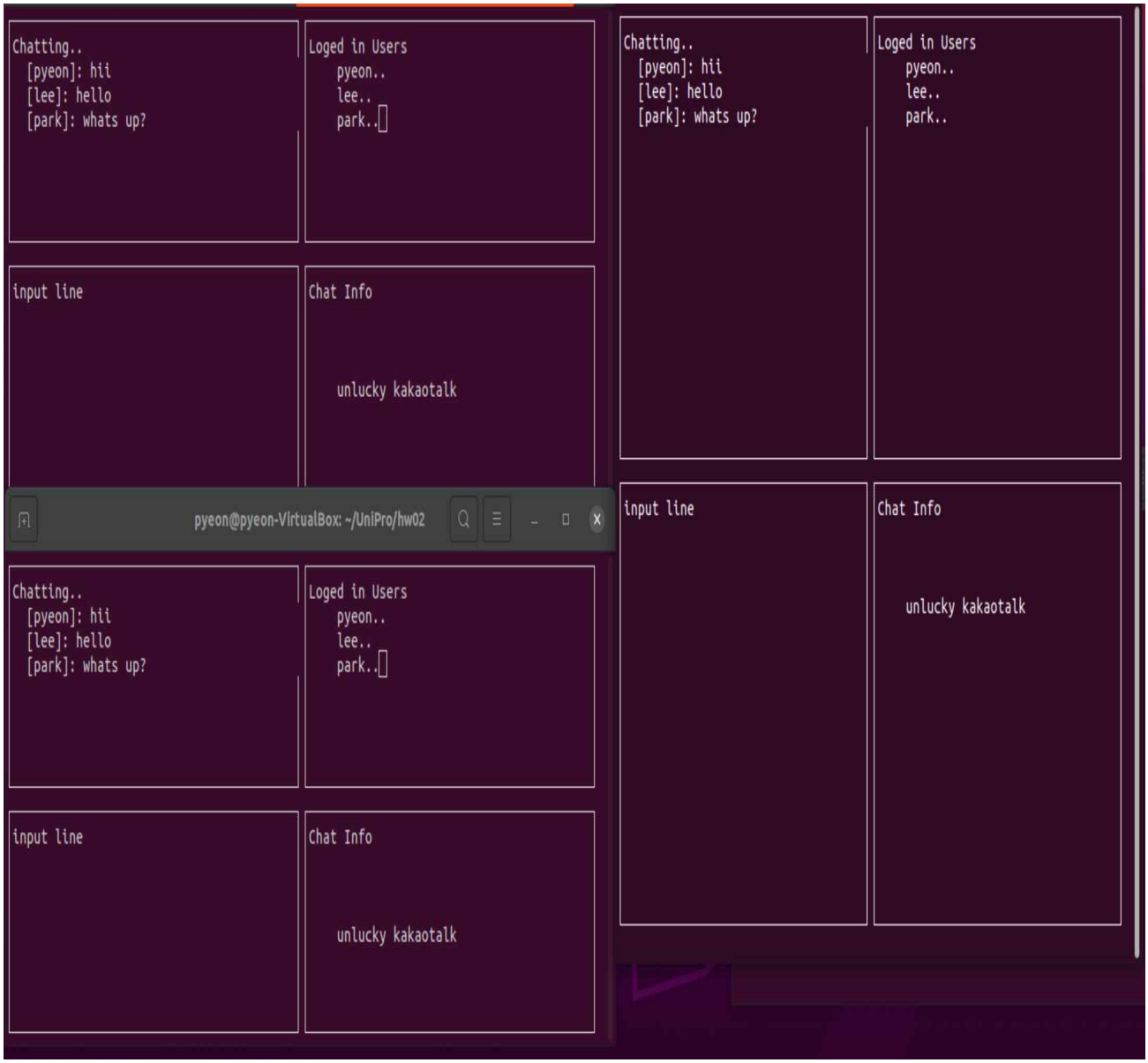
```
void sigint_handler(int signum) {

    // 컨트롤 c 입력시 실행 중인 스레드에게 취소요청 보냄
    pthread_cancel(usrthread);
    pthread_cancel(display_thread);
    cf->user_no--;  // 유저 1명 감소
    // 후에 입장할 유저를 위해 퇴장한 유저 자리를 빈칸으로 둠
    for (int i = 0; i < 3; i++)
    {
        if(strcmp(cf->userlist[i], cname) == 0)
        {
            strcpy(cf->userlist[i], "");
            break;
        }
    }
}
```

main 함수에서 signal(SIGINT, sigint_handler); 코드를 통해 SIGINT 시그널이 발생했을 때 sigint_handler 함수를 호출하도록 설정했습니다. SIGINT 시그널은 사용자 인터럽트 시그널로, 사용자가 Ctrl+C 키를 누를 때 발생합니다. sigint_handler 함수에서는 동작 중인 userthread(메세지 입력 받음), display_thread(화면을 출력함)에게 pthread_cancel로 메시지 종료 요청을 보냅니다. 두 스레드는 pthread_setcancelstate(), pthread_setcanceltype() 부분에서 각각 enable, Asynchronous 하게 동작하기에 취소 요청이 들어오면 즉각적으로 종료됩니다. 시그널 핸들러 함수는 두 스레드를 종료시키고 난 뒤 유저를 1명 감소시키고 나중에 새로운 유저가 들어왔을 때 유저목록에 추가 시켜주기 위해 퇴장한 유저 자리를 공백으로 만듭니다. 시그널 핸들러 함수가 종료되면 pthread_join으로 대기 중이던 메인 스레드가 sem_unlink로 해당 프로세스의 argv[1]값인 유저 아이디를 갖는 네임드 세마포어를 제거합니다. 만약 프로세스 내 유저수가 0명이 되면 shmremove 파일을 실행시켜 공유메모리 세그먼트를 자동으로 제거하도록 구현했습니다. 이로서 접속자가 컨트롤 c를 눌러 채팅을 빠져나가면 해당 유저의 이름으로 생성된 네임드 세마포어가 자동으로 제거되고 모든 접속자가 빠져나가면 채팅 프로그램이 종료되도록 유도할 수 있었습니다.

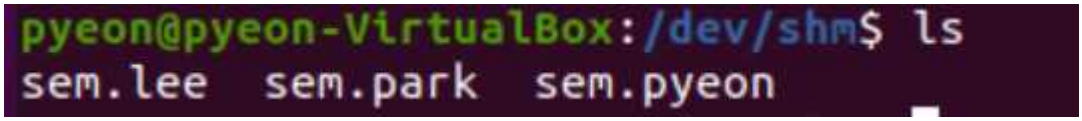
3. 구현한 결과물의 실행 과정을 스크린 샷과 함께 단계별로 기술하시오.



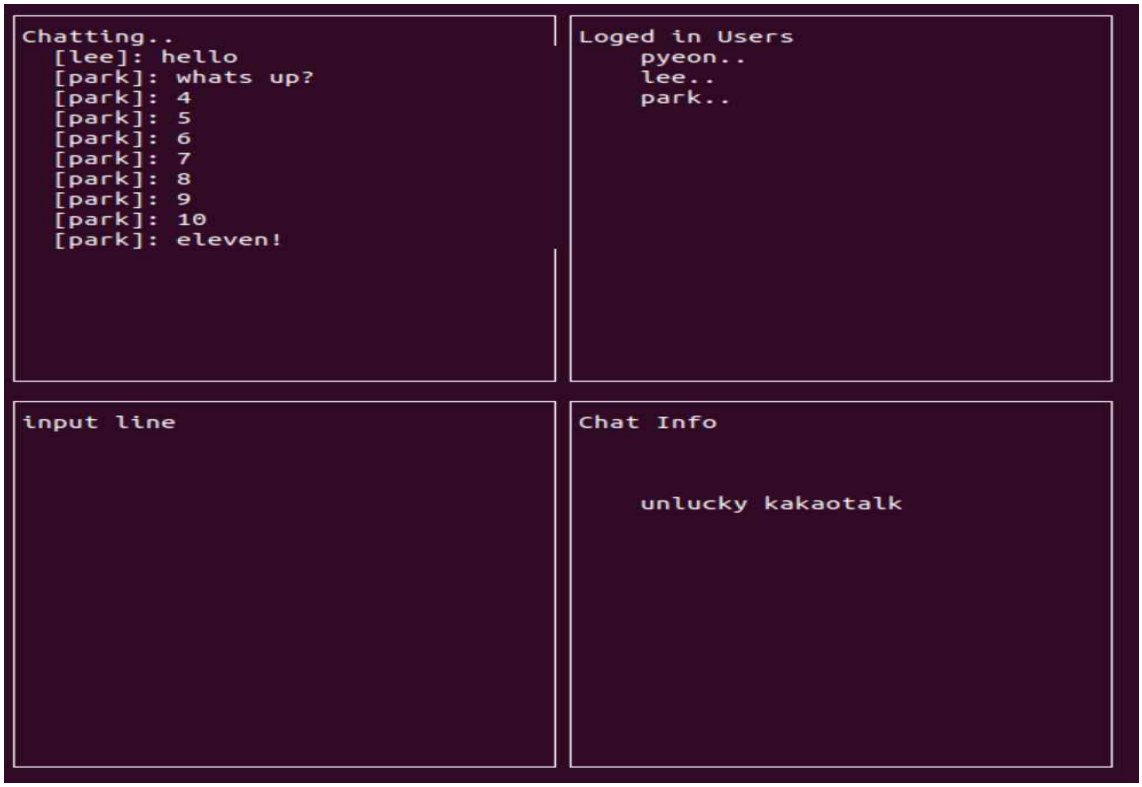
'./csechat pyeon'을 통해 첫 유저가 접속하고 ipcs 명령어를 입력하면 0x00000f60의 키 값을 갖는 공유메모리 세그먼트가 생성 된 것을 확인할 수 있습니다. 이때 key값이 0인 공유메모리들은 시스템에서 예약된 공유 메모리 세그먼트를 나타내며, 사용자가 직접 만든 것이 아닙니다. /dev/shm 경로에 들어가 ls를 입력하면 프로세스에 생성된 네임드 세마포어를 볼 수 있습니다. 현재 한 명의 접속자(sem.pyeon)의 네임드 세마포어가 생성된 모습을 확인할 수 있습니다.



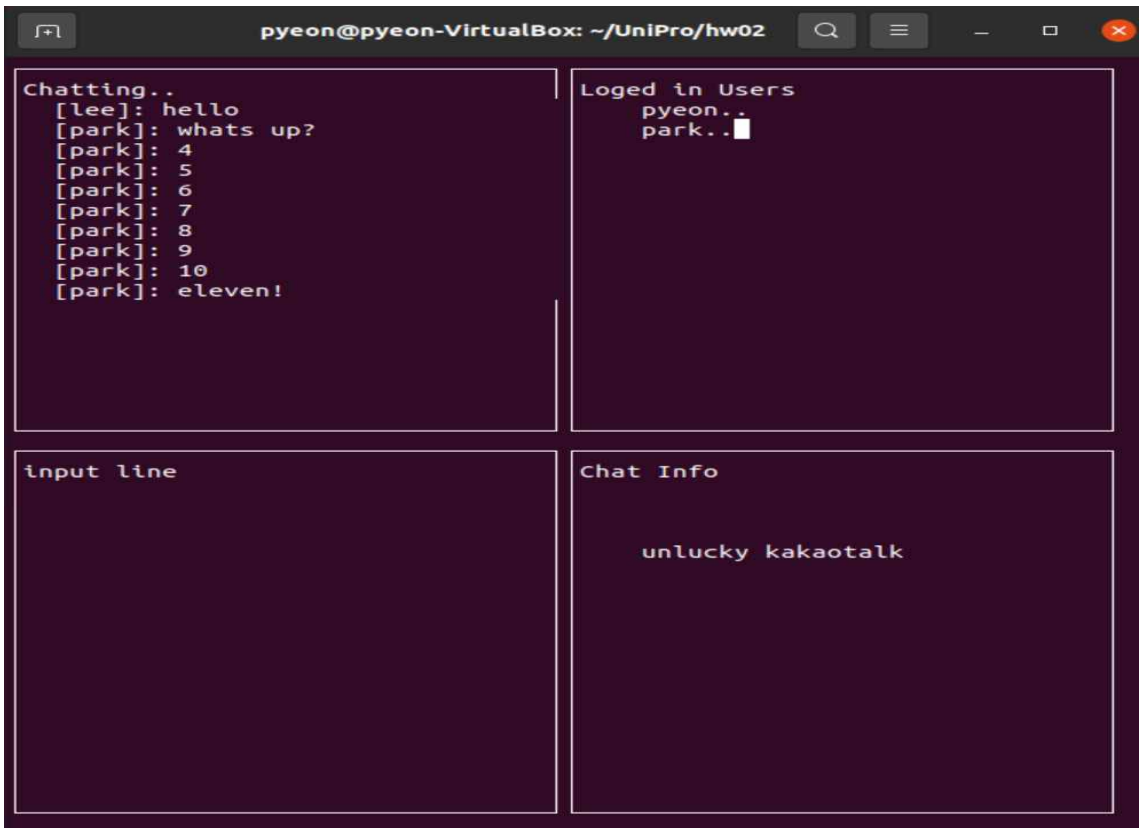
세 명의 접속자가 채팅프로그램에 들어와 메시지를 주고 받는 모습입니다. 서브윈도우2인 접속된 유저 리스트 목록에 사용자가 추가되고 모든 사용자의 채팅프로그램에 메시지가 실시간으로 공유되는 것을 볼 수 있습니다.



/dev/shm의 세마포어 목록에서도 3명의 세마포어가 각각 생성된 걸 확인할 수 있습니다.



메시지 10개가 채워지면 스크롤 업 되는 기능은 여전히 구현되어 있습니다.



유저 한명이 컨트롤 c를 누르고 프로그램에서 빠져나간 모습입니다. 로그인된 유저 목록에 유저 lee가 빠져 있는 걸 확인 할 수 있습니다.

```
pyeon@pyeon-VirtualBox:/dev/shm$ ls
sem.lee  sem.park  sem.pyeon
pyeon@pyeon-VirtualBox:/dev/shm$ ls
sem.park  sem.pyeon
```

/dev/shm에도 sem.lee가 제거되어 있습니다.

```
0x00000000 1146880    pyeon      600        1916928    2          dest
0x00000000 1212417    pyeon      600        1916928    2          dest
0x00000000 294922     gdm        600        16384      1          dest
0x00000000 294925     gdm        600        3407872    2          dest
0x00000000 294928     pyeon      600        16384      1          dest
0x00000000 1179673    pyeon      600        4194304    2          dest
0x00000000 1179674    pyeon      600        1376256    2          dest
0x00000000 294940     pyeon      600        524288     2          dest
0x00000000 1179683    pyeon      600        1572864    2          dest
0x00000000 1179685    pyeon      600        1572864    2          dest
0x00000000 327722     pyeon      600        524288     2          dest
0x00000000 1146927    pyeon      600        6193152    2          dest
0x00000000 1212464    pyeon      600        2555904    2          dest
0x00000000 1179705    pyeon      600        1703936    2          dest
0x00000000 1114174    pyeon      600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

pyeon@pyeon-VirtualBox:/dev/shm$ ls
pyeon@pyeon-VirtualBox:/dev/shm$
```

남아있는 모든 유저가 컨트롤 c를 눌러 빠져나가면 공유메모리 세그먼트는 자동으로 해제되고 /dev/shm의 리스트에도 네임드 세마포어가 모두 제거된 것을 확인할 수 있습니다.

4. 전체적인 코드 설명

(chartshm.h)

```
#ifndef __CHAT_SHARE_MEMORY_H__
#define __CHAT_SHARE_MEMORY_H__
typedef struct chatInfo{
    char userlist[3][10];    //유저목록
    int user_no;             //유저 인덱스용 변수
    char messages[10][60]; //메세지 저장
    int message_index;       //메세지 인덱스용 변수
} CHAT_INFO;

#endif//__CHAT_SHARE_MEMORY_H__
```

공유메모리에 연결될 구조체입니다.

(csechat.c)

```c
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#include <ncurses.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <sys/sem.h>
#include <unistd.h>
#include <fcntl.h>
#include "chatshm.h"

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_t usrthread, display_thread;    // 스레드는 전역으로 배치
WINDOW * sw1, *sw2, *sw3, *sw4;         // 서브윈도우
CHAT_INFO *cf = NULL;                   // 공유메모리에 연결될 구조체

sem_t *sem;                             // 세마포어
char cname[10];                         // argv[1] 값을 저장할 배열

void init_screen(); // 초기 화면 구성 프로토타입
```

전역으로 선언된 변수들과 init_screen() 함수의 프로토타입입니다.

(read_messages 함수 - 1)

```
void *read_messages(void *arg)
{
    // 시그널 핸들러 함수에서 스레드 캔슬 발생시 즉각 처리
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,NULL);

    char curmsg[40];

    /** for subwin 2 **/
    // 중복된 이름 있다면 종료시킴
    for (int i = 0; i < 3; i++)
        if(strcmp(cf->userlist[i], (char*)arg) == 0) {endwin(); exit(0);}

    // 유저수가 3명인데 입장한 경우 unlink후 종료
    if(cf->user_no > 2) {sem_unlink((char*)arg); endwin(); exit(0);}

    // 유저 목록의 빈 곳에 새로운 유저 추가
    for (int i = 0; i < 3; i++)
    {
        if(strcmp(cf->userlist[i], "") == 0)
        {
            strcpy(cf->userlist[i], (char*) arg);
            cf->user_no++;
            break;
        }
    }
```

스레드 캔슬 요청 받았을 시 즉각적으로 스레드를 종료시키는 코드를 넣었습니다. 중복된 이름을 체크하고 최대 접속 가능한 유저 수를 유지합니다. 새로운 유저가 입장하면 빈 유저 목록에 이름을 추가합니다.

(read_messages 함수 - 2)

```
    while(1)
    {
        /** for subwin 3 **/
        echo();
        mvwgetnstr(sw3, 5, 5, curmsg, 40);
        wclear(sw3);
        mvwprintw(sw3, 1, 1, "input line\n");
        box(sw3, 0, 0);
        wrefresh(sw3);

        sem_wait(sem);  // sem 값이 1일때 감소시키고 다음 코드 실행

        // 메세지가 10개 초과될 시 한 칸씩 앞으로 배치 후 새 메세지 추가
        if(cf->message_index >= 10)
        {
            for (int i = 0; i < 9; i++)
            {
                strcpy(cf->messages[i], cf->messages[i+1]);
            }
            snprintf(cf->messages[9], sizeof(char) * 60,
                    "[%s]: %s", (char *)arg, curmsg);
        }
        else
        {
            snprintf(cf->messages[cf->message_index], sizeof(char) * 60,
                    "[%s]: %s", (char *)arg, curmsg);
            cf->message_index++;
        }
        sem_post(sem);  // sem 값을 1 증가시켜 동기화
    }

    return NULL;
}
```

서브윈도우3 화면을 구성하고 메시지가 10개 이하일 때와 초과일 때를 구분해서 메시지 배열에 저장합니다. 이 과정은 sem_wait와 sem_post로 크리티컬 섹션을 만들어 다른 프로세스가 접근할 수 없도록 동기화 했습니다.

(display_message 함수)

```c
void *display_messages(void *arg)
{
    // 시그널 핸들러 함수에서 스레드 캔슬 발생시 즉각 처리
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,NULL);
    /** for subwin1 and subwin2 **/
    while(1)
    {
        sem_wait(sem);  // sem 값을 1 감소시킴

        wclear(sw1);
        box(sw1, 0, 0);
        mvwprintw(sw1, 1, 1, "Chatting..");
        for (int i = 0, h = 2; i < cf->message_index; i++, h++)
        {
            mvwprintw(sw1, h, 3, "%s", cf->messages[i]);
        }
        wrefresh(sw1);

        wclear(sw2);
        mvwprintw(sw2, 1, 1, "Loged in Users\n");
        box(sw2, 0, 0);
        // 로그인된 유저 목록 출력
        for (int i = 0, h = 2; i < 3; i++, h++)
        {
            if(strcmp(cf->userlist[i],"") == 0) {h--; continue;}
            mvwprintw(sw2, h, 5, "%s..", cf->userlist[i]);
        }
        wrefresh(sw2);

        sem_post(sem);  // sem 값을 1 증가시켜 동기화
        usleep(50000);  // 화면 반짝임 없앰
    }

    return NULL;
}
```

시그널 핸들러 함수에서 스레드 캔슬 요청시 스레드가 즉각적으로 종료되도록 처리했습니다. 서브윈도우 1과 2에 각각 메시지 목록, 접속 중인 유저 리스트를 출력하며 그 과정은 sem_wait와 sem_post로 감싸 아토믹하게 동작되도록 유지했습니다. usleep을 통해 지나친 화면 깜빡임을 제거했습니다.

(sigint_handler 함수)

```
void sigint_handler(int signum) {

    // 컨트롤 c 입력시 실행 중인 스레드에게 취소요청 보냄
    pthread_cancel(usrthread);
    pthread_cancel(display_thread);
    cf->user_no--;  // 유저 1명 감소
    // 후에 입장할 유저를 위해 퇴장한 유저 자리를 빈칸으로 둠
    for (int i = 0; i < 3; i++)
    {
        if(strcmp(cf->userlist[i], cname) == 0)
        {
            strcpy(cf->userlist[i], "");
            break;
        }
    }
}
```

컨트롤 c 입력시 usrthread와 disaply_thread에 종료 요청을 보내고 유저 수를 한명 감소시킴과 동시에 퇴장한 유저 자리를 빈칸으로 두어 나중에 입장할 유저를 추가할 공간을 마련했습니다.

(main 함수 - 1)

```
int main(int argc, char *argv[]) {

    init_screen();
    // argv[1]값을 cname에 복사시켜 시그널핸들러 함수가 cname을 사용하게 함
    strcpy(cname, argv[1]);
    // argv[1]의 이름을 갖는 네임드 세마포어, 초기 sem 값을 1로 줌
    sem = sem_open(argv[1], O_CREAT, 0666, 1);
    // 시그널 핸들러 함수 등록
    signal(SIGINT, sigint_handler);
    if (sem == SEM_FAILED)
    {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    int shmid;
    bool init_flag = true;
    void *shmaddr = (void *)0;
    // 헤더파일의 구조체와 공유 메모리를 연결
    shmid = shmget((key_t)3936, sizeof(CHAT_INFO), 0666|IPC_CREAT|IPC_EXCL);
    if (shmid < 0)
    {
        init_flag = false;
        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;
```

signal 핸들러 함수를 등록하고 chatshm.h에 정의된 구조체를 공유메모리와 연결합니다. sem_open으로 argv[1]값을 갖는 다른 프로세스와 구분되는 네임드 세마포어를 생성하였습니다. 초기값을 1로 설정했기에 read_message나 display_messages 함수의 sem_wait(sem); 에 진입 할 수 있게 됩니다.

(main 함수 - 2)

```
    if (init_flag)
    {
        cf->user_no = 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], "");
    }

    // 메세지를 읽는 스레드와, 메세지를 화면에 출력하는 스레드 생성
    pthread_create(&usrthread, NULL, read_messages, (void *) argv[1]);
    pthread_create(&display_thread, NULL, display_messages, (void *) argv[1]);

    // 메인 함수는 스레드들을 기다려 먼저 종료되지 않게 함
    pthread_join(usrthread, NULL);
    pthread_join(display_thread, NULL);

    // 시그널 발생 후 스레드들이 종료되면 sem을 unlink시킴
    sem_unlink(argv[1]);
    sem_close(sem);
    // 프로세스 내 유저수가 0명이 되면 공유메모리 해제를 진행
    if(cf->user_no == 0) system("./shmremove");
    endwin();

    return 0;
}
```

변수들에 쓰레기값이 들어가지 않도록 초기화 하고 pthread_create로 메시지를 읽는 스레드, 메시지를 화면에 출력하는 스레드를 생성하고 메인 스레드가 먼저 종료되지 않도록 join을 걸어두었습니다. 시그널이 발생해 두 스레드가 종료되면 메인 스레드는 생성한 네임드 세마포어를 제거하고 세마포어를 close 합니다. 채팅 프로그램에 남아있는 유저가 0명이 되면 shmremove 파일을 호출한 뒤 ncurses 화면을 끝내고 종료하게 됩니다.

(init_screen 함수)

```
void init_screen() {
    // 서브윈도우 4개로 분할 후 테두리 생성
    initscr();
    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);
    sw2 = newwin(sub_height, sub_width, 0, sub_width);
    sw3 = newwin(sub_height, sub_width, sub_height, 0);
    sw4 = newwin(sub_height, sub_width, sub_height, sub_width);
    box(sw1, 0, 0);
    box(sw2, 0, 0);
    box(sw3, 0, 0);
    box(sw4,0, 0);
    mvwprintw(sw1, 1, 1, "Chatting..");
    mvwprintw(sw2, 1, 1, "Loged in Users");
    mvwprintw(sw3, 1, 1, "input line");
    mvwprintw(sw4, 1, 1, "Chat Info");
    mvwprintw(sw4, 5, 5, "unlucky kakaotalk");

    wrefresh(sw1); wrefresh(sw2);wrefresh(sw3);wrefresh(sw4);
}
```

4개의 서브윈도우를 생성하고 각각 채팅 목록, 접속한 유저, 메시지 입력란, 채팅 정보를 출력하도록 구현했습니다.

(shmremove.c)

```c
#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 만듦
    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;
}
```

csechat.c의 메인함수에서 system() 코드로 호출 받으면 공유메모리를 제거하는 역할을 합니다.