2025-1 운영체제 Project 보고서

멀티스레드를 이용한
실시간 대화 및 게임 서비스

2025. 6. 13

스마트 IoT
20235177
박혜진

한림대학교
HALLYM UNIVERSITY

목 차

그 림 목 차

※ 본 장에서는, 보고서에 사용된 그림의 목차를 순차적으로 작성할 것
- 그림의 번호는 [장, 순서] 로 작성할 것
- 예시: 1장에서 사용되는 그림은 [그림 1-1]부터 [그림 1-n] 형식으로 작성하고, 2장에서 사용되는 그림들은 [그림 2-1]부터 순서대로 작성할 것

제 1 장 서 론

제 1 절 필요성 및 목적

1. 프로젝트의 필요성

비대면 소통의 중요성이 커지면서, 여러 사용자가 동시에 실시간으로 소통할 수 있는 채팅 서비스의 수요가 급증하고 있다. 하지만 기존의 채팅 서비스들은 사용자간 다양한 상호작용 기능이 부족하여 편리한 사용 환경을 제공하는 데 한계가 있다. 단순히 텍스트를 주고받는 기능에 머물러 사용자 참여도를 높이기 어렵다고 생각한다. 따라서 본 프로젝트는 이러한 한계를 극복하고자 다중 클라이언트가 동시에 접속해 자유롭게 채팅할 수 있는 환경을 구현하고, 채팅방 생성 및 이동, 사용자 이름 변경 등 사용자 편의 기능을 포함한다. 더 나아가, 다중 사용자 간에 실시간으로 참여할 수 있는 업다운 게임을 구현함으로써 단순한 채팅 서비스를 넘어서는 재미와 상호작용 요소를 제공한다. 이러한 기능들은 pthread를 활용한 멀티스레드 환경과 소켓 통신, IPC 기법을 통해 안정적이고 효율적으로 운영될 수 있어, 기존 서비스의 문제점을 보완할 수 있다.

2. 프로젝트의 목적

본 프로젝트의 목적은 다중 클라이언트가 동시에 접속하여 실시간으로 원활한 채팅을 주고받고, 게임을 할 수 있는 환경을 구축하는 데 있다. 사용자가 원하는 시점에 채팅방을 생성하거나 이동할 수 있도록 하여 소규모 그룹별 대화가 가능하도록 지원하며, 사용자 이름 변경 기능을 통해 개인화된 대화 환경을 제공하려고 한다. 또한, pthread를 활용한 멀티스레드 처리와 IPC 기법을 적용하여 서버와 클라이언트 간의 안정적이고 효율적인 통신을 보장하는 것을 목표로 한다. 다중 사용자가 함께 즐길 수 있는 업다운 게임을 추가하여, 단순한 텍스트 채팅을 넘어선 재미와 사용자 간 상호작용을 높이고자 한다. 이런 기능을 구현함으로써 사용자들이 더 편리한 소통 환경을 경험할 수 있도록 하는 것을 목표로 삼는다.

제 2 장 본 론

제 1 절 Server 구현

1. Server 역할 및 동작

본 채팅 서버는 다중 클라이언트 접속을 지원하며, 각 클라이언트와 독립적으로 통신하는 구조로 설계되었다. 서버는 지정된 포트에서 클라이언트의 접속을 대기하고, 새로운 클라이언트가 접속하면 accept() 함수를 통해 연결한다. 이후 각 클라이언트마다 별도의 스레드를 생성하여 동시에 여러 사용자가 채팅에 참여할 수 있도록 한다. 클라이언트가 접속하면 우선 사용자 이름을 등록하도록 하며, 기본 채팅방인 "Lobby"에 입장시킨다. 사용자는 명령어를 통해 새로운 채팅방을 생성하거나 기존 방으로 이동할 수 있으며, 이름 변경 또한 가능하다.

서버는 클라이언트가 전송한 메시지를 읽고, 명령어인 경우 별도로 처리하며, 일반 메시지는 동일 채팅방의 다른 클라이언트에게 전달한다. 채팅방 내에서 숫자 맞추기 게임 기능을 제공한다. 게임은 /start 명령어로 시작되며, 방에 있는 클라이언트들이 차례로 숫자를 추측하는 방식이다. 게임 진행 중에는 차례가 아닌 사용자가 입력할 경우 "차례가 아니다"라는 오류 메시지를 전송하여 입력을 제한한다. 다중 스레드 환경에서 클라이언트 목록과 게임 상태와 같은 공유 데이터는 뮤텍스(pthread_mutex_t)를 사용해 동기화하여 데이터 무결성과 충돌 방지를 보장한다.

2. Sever 소스코드 설명

2.1 클라이언트 접속 관리

서버의 주요 기능은 크게 클라이언트 접속 관리, 메시지 처리, 그리고 게임 진행으로 나눌 수 있다. 우선 main 함수에서 accept()를 통해 새로운 클라이언트 접속을 받으면, 클라이언트 정보를 구조체에 저장하고 전역 클라이언트 목록에 추가한다. 이때 최대 클라이언트 수를 초과하지 않도록 검사하며, 클라이언트마다 독립적인 스레드를 생성해

handle_clnt 함수를 실행한다. 이 함수는 클라이언트와의 모든 입출력을 처리하며, 이름 등록, 명령어 처리, 메시지 전송, 클라이언트 종료를 담당한다.

```c
int main(int argc, char *argv[]) {
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;
    socklen_t clnt_adr_sz;
    pthread_t t_id;

    struct tm *t;
    time_t timer = time(NULL);
    t = localtime(&timer);

    if(argc != 2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    pthread_mutex_init(&mutx, NULL);
    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family = AF_INET;
    serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_adr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1)
        error_handling("bind() error");
    if(listen(serv_sock, 5) == -1)
        error_handling("listen() error");

    menu(argv[1]);
```

[그림 2-1] main 함수 1

```c
    while(1) {
        clnt_adr_sz = sizeof(clnt_adr);
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
        if(clnt_sock == -1) continue;

        pthread_mutex_lock(&mutx);
        if(clnt_cnt >= MAX_CLNT) {
            pthread_mutex_unlock(&mutx);
            close(clnt_sock);
            continue;
        }
        client_t new_client;
        new_client.sock = clnt_sock;
        new_client.name[0] = 0;
        strcpy(new_client.room, "Lobby");
        add_client(new_client);
        pthread_mutex_unlock(&mutx);

        pthread_create(&t_id, NULL, handle_clnt, (void*)&clnt_sock);
        pthread_detach(t_id);

        printf("Connected client IP: %s ", inet_ntoa(clnt_adr.sin_addr));
        printf("(%d-%d-%d %d:%d)\n", t->tm_year+1900, t->tm_mon+1, t->tm_mday,
               t->tm_hour, t->tm_min);
        printf(" chatter (%d/%d)\n", clnt_cnt, MAX_CLNT);
    }

    close(serv_sock);
    return 0;
}
```

[그림 2-2] main 함수 2

2.2 메시지 처리 (handle_clnt 함수 주요 부분)

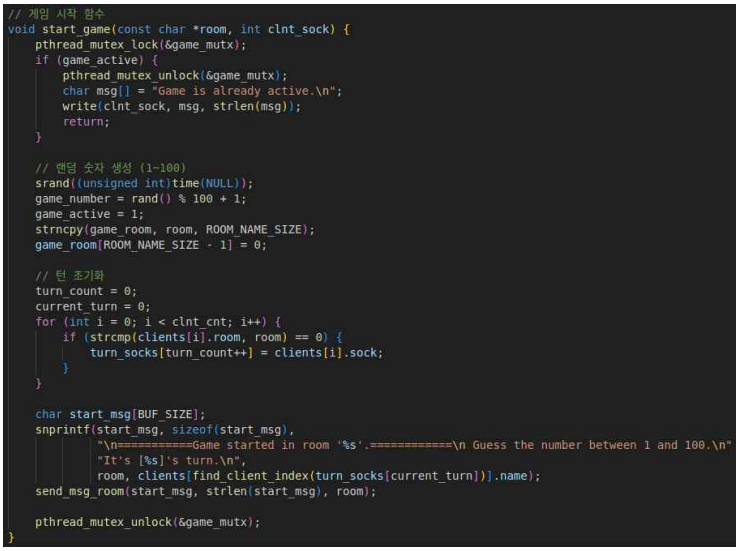
handle_clnt 함수는 각각의 클라이언트 스레드가 실행하는 함수로, 클라이언트와 서버 간의 입출력 작업을 담당한다. 처음에는 클라이언트가 자신의 이름을 서버에 등록할 때까지 대기한다. 이름이 등록되면 해당 사용자가 속한 채팅방에 입장 메시지를 전송한다. 이후 클라이언트가 보내는 메시지를 반복해서 읽으면서, 메시지 앞에 '/'가 붙으면 명령어로 인식해 별도의 명령어 처리 함수로 넘긴다. 그렇지 않은 일반 메시지는 같은 채팅방 내의 다른 클라이언트들에게 전달한다. 클라이언트가 연결을 종료하면 목록에서 제거하고 퇴장 메시지를 보내며 소켓을 닫는다.

```c
void *handle_clnt(void *arg) {
    int clnt_sock = *((int*)arg);
    int str_len;
    char msg[BUF_SIZE];
    char send_msg_buf[BUF_SIZE + NAME_SIZE + ROOM_NAME_SIZE];
    int idx;

    // 이름 등록 루프
    while (1) {
        str_len = read(clnt_sock, msg, sizeof(msg) - 1);
        if (str_len <= 0) {
            pthread_mutex_lock(&mutx);
            remove_client(clnt_sock);
            pthread_mutex_unlock(&mutx);
            close(clnt_sock);
            return NULL;
        }
        msg[str_len] = 0;
        strip_newline(msg);

        pthread_mutex_lock(&mutx);
        idx = find_client_index(clnt_sock);
        if(idx == -1) {
            pthread_mutex_unlock(&mutx);
            close(clnt_sock);
            return NULL;
        }
        if(strlen(clients[idx].name) == 0) {
            if(strlen(msg) < NAME_SIZE) {
                strcpy(clients[idx].name, msg);
                snprintf(send_msg_buf, sizeof(send_msg_buf), "\n----[%s] joined the %s----\n\n", clients[idx].name, clients[idx].room);
                send_msg_room(send_msg_buf, strlen(send_msg_buf), clients[idx].room);
                pthread_mutex_unlock(&mutx);
                break;
            } else {
                char err_msg[] = " Name too long.\n";
                write(clnt_sock, err_msg, strlen(err_msg));
                pthread_mutex_unlock(&mutx);
                continue;
            }
        } else {
            pthread_mutex_unlock(&mutx);
            break;
        }
    }

    // 채팅 및 명령 처리 루프
    while((str_len = read(clnt_sock, msg, sizeof(msg) - 1)) > 0) {
        msg[str_len] = 0;
        strip_newline(msg);

        if(msg[0] == '/') {
            pthread_mutex_lock(&mutx);
            idx = find_client_index(clnt_sock);

            if(strncmp(msg, "/name ", 6) == 0) {
                char *new_name = msg + 6;
                if(strlen(new_name) < NAME_SIZE) {
                    char old_name[NAME_SIZE];
                    strcpy(old_name, clients[idx].name);
                    strcpy(clients[idx].name, new_name);
                    snprintf(send_msg_buf, sizeof(send_msg_buf), " [%s] changed name to [%s]\n", old_name, new_name);
```

[그림 2-3] 메시지 처리 (handle_clnt 함수)

2.3 게임 진행 시작 (start_game 함수)

게임 시작은 start_game 함수에서 이루어진다. 해당 채팅방의 게임 상태를 활성화하고, 게임 참여 클라이언트들의 소켓 정보를 별도의 배열에 저장한다. 게임에서 맞춰야 할 숫자는 1부터 100 사이의 랜덤한 값으로 설정한다. 이후 턴 순서대로 클라이언트가 게임을 진행할 수 있도록 준비한다. 게임 상태를 변경하는 작업은 여러 스레드가 동시에 접근할 수 있으므로 뮤텍스를 이용해 안전하게 처리한다.

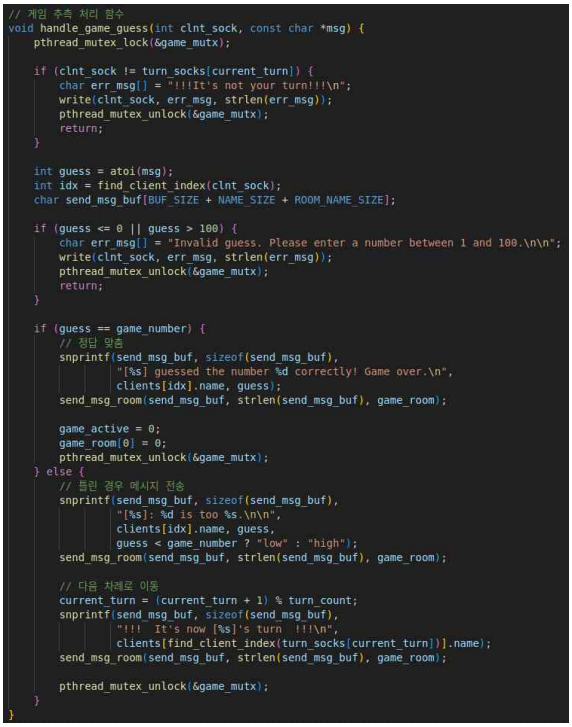


```c
// 게임 시작 함수
void start_game(const char *room, int clnt_sock) {
    pthread_mutex_lock(&game_mutx);
    if (game_active) {
        pthread_mutex_unlock(&game_mutx);
        char msg[] = "Game is already active.\n";
        write(clnt_sock, msg, strlen(msg));
        return;
    }

    // 랜덤 숫자 생성 (1~100)
    srand((unsigned int)time(NULL));
    game_number = rand() % 100 + 1;
    game_active = 1;
    strncpy(game_room, room, ROOM_NAME_SIZE);
    game_room[ROOM_NAME_SIZE - 1] = 0;

    // 턴 초기화
    turn_count = 0;
    current_turn = 0;
    for (int i = 0; i < clnt_cnt; i++) {
        if (strcmp(clients[i].room, room) == 0) {
            turn_socks[turn_count++] = clients[i].sock;
        }
    }

    char start_msg[BUF_SIZE];
    snprintf(start_msg, sizeof(start_msg),
             "\n==========Game started in room '%s'.==========\n Guess the number between 1 and 100.\n"
             "It's [%s]'s turn.\n",
             room, clients[find_client_index(turn_socks[current_turn])].name);
    send_msg_room(start_msg, strlen(start_msg), room);

    pthread_mutex_unlock(&game_mutx);
}
```

[그림 2-4]

2.4 차례 제어 및 숫자 입력 처리 함수 설명

이 함수는 업다운 게임 중 사용자가 입력한 메시지를 처리한다. 가장 먼저, 현재 입력한 사용자가 차례인지 확인하며, 차례가 아닌 경우에는 오류 메시지를 보내고 함수 실행을 종료한다. 차례인 경우에는 해당 메시지를 정수로 변환하여 유효한 숫자인지 검사한다. 정답을 맞춘 경우에는 전체 방에 정답자 알림 메시지를 전송하고 게임을 종료한다. 맞추지 못한 경우에는 "업" 또는 "다운" 메시지를 해당 채팅방 모든 사용자에게 전송한 후, 다음 차례 사용자로 턴을 넘긴다. 이 함수는 여러 클라이언트 스레드가 동시에 접근할 수 있기 때문에 pthread_mutex_lock() / unlock()을 통해 공유 자원인 게임 상태를 보호한다.



```
// 게임 추측 처리 함수
void handle_game_guess(int clnt_sock, const char *msg) {
    pthread_mutex_lock(&game_mutx);

    if (clnt_sock != turn_socks[current_turn]) {
        char err_msg[] = "!!!It's not your turn!!!\n";
        write(clnt_sock, err_msg, strlen(err_msg));
        pthread_mutex_unlock(&game_mutx);
        return;
    }

    int guess = atoi(msg);
    int idx = find_client_index(clnt_sock);
    char send_msg_buf[BUF_SIZE + NAME_SIZE + ROOM_NAME_SIZE];

    if (guess <= 0 || guess > 100) {
        char err_msg[] = "Invalid guess. Please enter a number between 1 and 100.\n\n";
        write(clnt_sock, err_msg, strlen(err_msg));
        pthread_mutex_unlock(&game_mutx);
        return;
    }

    if (guess == game_number) {
        // 정답 맞춤
        snprintf(send_msg_buf, sizeof(send_msg_buf),
                 "[%s] guessed the number %d correctly! Game over.\n",
                 clients[idx].name, guess);
        send_msg_room(send_msg_buf, strlen(send_msg_buf), game_room);

        game_active = 0;
        game_room[0] = 0;
        pthread_mutex_unlock(&game_mutx);
    } else {
        // 틀린 경우 메시지 전송
        snprintf(send_msg_buf, sizeof(send_msg_buf),
                 "[%s]: %d is too %s.\n\n",
                 clients[idx].name, guess,
                 guess < game_number ? "low" : "high");
        send_msg_room(send_msg_buf, strlen(send_msg_buf), game_room);

        // 다음 차례로 이동
        current_turn = (current_turn + 1) % turn_count;
        snprintf(send_msg_buf, sizeof(send_msg_buf),
                 "!!!  It's now [%s]'s turn  !!!\n",
                 clients[find_client_index(turn_socks[current_turn])].name);
        send_msg_room(send_msg_buf, strlen(send_msg_buf), game_room);

        pthread_mutex_unlock(&game_mutx);
    }
}
```

[그림 2-5] 게임 추측 처리 함수

제 2 절 Client 구현

1. Client 역할 및 동작

클라이언트 프로그램은 사용자와 서버 간의 인터페이스 역할을 수행한다. 사용자가 입력한 메시지나 명령어를 서버로 전송하고, 서버로부터 전달된 메시지를 실시간으로 출력한다. 클라이언트는 최초 실행 시 사용자의 닉네임과 서버 IP 및 포트를 입력받아 서버에 접속하며, 접속 후 자동으로 자신의 이름을 서버에 전송한다. 이후 서버와의 송수신 처리를 위한 두 개의 스레드(send_msg, recv_msg)를 생성하여 사용자 입력을 실시간으로 처리한다. 클라이언트는 서버로부터 전송된 메시지를 그대로 출력하고, 사용자가 입력한 명령어(/create, /join, /name, /start)에 따라 서버에서 적절한 처리를 수행하도록

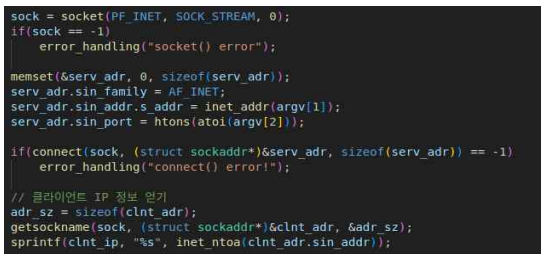
명령을 전달한다.

클라이언트는 q 또는 Q를 입력하면 접속을 종료하며, 종료 시 소켓을 닫고 프로그램이 종료된다. 또한 클라이언트는 서버 접속 시점의 시간과 자신의 IP, 포트, 닉네임 등의 정보를 출력해 사용자가 접속 정보를 확인할 수 있도록 한다.

2. Client 소스코드 설명

2.1 클라이언트 초기화 및 서버 연결

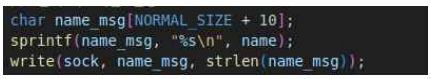
클라이언트가 서버에 접속하기 위해 소켓을 생성하고 connect() 함수를 통해 서버와 연결을 시도하는 부분이다. 접속에 성공하면 자신의 IP 주소를 확인하고, 입력받은 이름을 서버로 전송한다. 자신의 IP 주소와 접속 정보를 출력한다.



[그림 2-6] 클라이언트 초기화 및 서버 연결

2.2 사용자 정보 전송 및 인터페이스 출력

클라이언트는 서버 접속 직후 자신의 이름을 서버에 전송한다. 이 정보는 서버에서 클라이언트 식별에 사용된다.



[그림 2-7] 사용자 정보 전송 및 출력

접속 후 사용자에게 채팅 명령어와 정보를 안내하기 위해 아래의 menu() 함수가 실행된다. 사용자가 명령어를 직관적으로 이해할 수 있도록 돕는다.

```c
void menu() {
    printf(" <<<< Chat Client >>>>\n");
    printf(" Server Port : %s \n", serv_port);
    printf(" Client IP   : %s \n", clnt_ip);
    printf(" Chat Name   : %s \n", name);
    printf(" Server Time : %s \n", serv_time);
    printf(" ============= Mode ============\n");
    printf(" /create <room> : create new room\n");
    printf(" /join <room>   : join/move room\n");
    printf(" /name <name>   : change name\n");
    printf(" /start         : start up and down game\n");
    printf(" ================================\n");
    printf(" Exit -> q\n\n");
}
```

[그림 2-8] menu() 함수

2.3 메시지 송수신을 위한 스레드 생성

클라이언트는 서버와의 비동기 송수신을 위해 두 개의 스레드를 생성한다. 하나는 사용자 입력을 서버로 전송하는 역할을 하며(send_msg), 다른 하나는 서버에서 오는 메시지를 읽어 출력한다(recv_msg).

```c
pthread_create(&snd_thread, NULL, send_msg, NULL);
pthread_create(&rcv_thread, NULL, recv_msg, NULL);
pthread_detach(snd_thread);
pthread_detach(rcv_thread);
```

[그림 2-9] 스레드 생성

send_msg 함수는 사용자 입력을 읽어 서버로 전송하며, q 또는 Q를 입력하면 소켓을 닫고 클라이언트를 종료한다. recv_msg 함수는 서버에서 오는 메시지를 계속 수신하여 화면에 출력한다. 수신 길이가 0 이하인 경우 연결 종료로 판단하고 스레드를 종료한다.

```c
void* send_msg(void* arg) {
    char send_buf[BUF_SIZE];
    while(1) {
        fgets(send_buf, sizeof(send_buf), stdin);
        strip_newline(send_buf);

        if(!strcmp(send_buf, "q") || !strcmp(send_buf, "Q")) {
            close(sock);
            exit(0);
        }

        write(sock, send_buf, strlen(send_buf));
    }
    return NULL;
}
```

[그림 2-10] send_msg () 함수

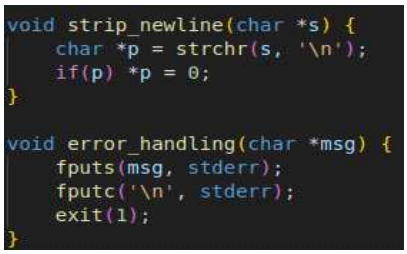
```c
void* recv_msg(void* arg) {
    int str_len;
    while(1) {
        str_len = read(sock, msg, sizeof(msg) - 1);
        if(str_len <= 0) {
            return NULL;
        }
        msg[str_len] = 0;
        printf("%s", msg);
    }
    return NULL;
}
```

[그림 2-11] recv_msg () 함수

2.4 기타 함수

사용자가 입력한 메시지에서 줄바꿈 문자(₩n)를 제거하는 strip_newline() 함수와, 오류

발생 시 메시지를 출력하고 프로그램을 종료하는 error_handling() 함수가 정의되어 있다.



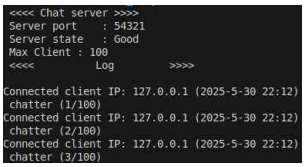
```
void strip_newline(char *s) {
    char *p = strchr(s, '\n');
    if(p) *p = 0;
}

void error_handling(char *msg) {
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

[그림 2-12] error_handling() 함수

제 3 절 실행 결과

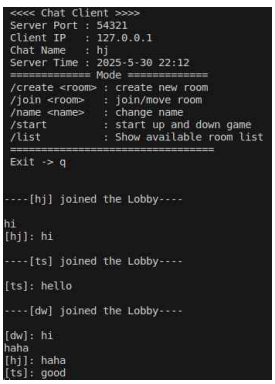
1. 실행 결과

Server 출력 화면



[그림 2-13] Server

Client #1 출력 화면

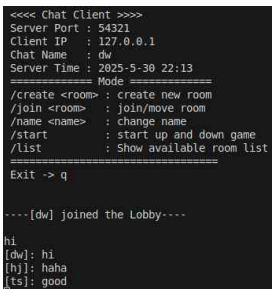


[그림 2-14] Client #1

Client #2 출력 화면

[그림 2-15] Client #2

Client #3 출력 화면



[그림 2-16] Client #3

/create : 새로운 채팅방을 생성하고, 해당 채팅방으로 자동 입장한다.

[그림 2-17] 채팅방 생성

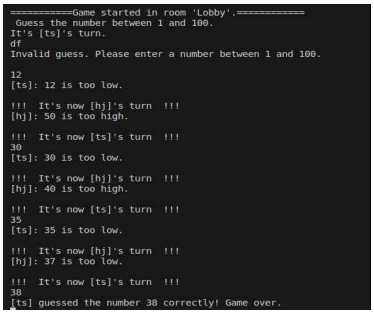
/join : 생성된 채팅방이 있을 시 해당 채팅방으로 이동하고, 없다면 채팅방이 없다는 출력문을 출력한다.

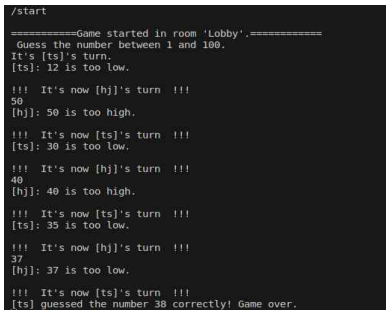
[그림 2-18] 생성된 채팅방 존재 [그림 2-19] 채팅방 존재 하지 않음

/name : 자신의 닉네임을 새로운 이름으로 변경한다.

[그림 2-20] /name 결과

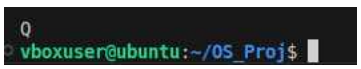
/start : 현재 채팅방에서 숫자 맞추기(Up & Down) 게임을 시작한다. 서버는 참가자 차례를 관리하며, 차례가 된 사용자에게 입력을 요구한다.
(ts, hj)





[그림 2-21] /start 결과 (ts) [그림 2-22] /start 결과 2 (hj)

Q or q : 사용자가 q 또는 Q를 입력하면 클라이언트 프로그램은 서버와의 연결을 종료하고 프로그램을 종료한다.



[그림 2-23] 프로그램 종료

제 3 장 결 론

본 프로젝트는 다중 클라이언트를 지원하는 채팅 서버와 클라이언트를 구현하였다. 채팅방 생성, 이동, 이름 변경, 숫자 맞추기 게임 등의 기능을 제공한다. 서버는 스레드 기반 구조와 뮤텍스 동기화를 통해 안정적인 통신과 게임 진행을 보장하고, 클라이언트는 사용자 친화적인 명령어 인터페이스와 비동기 송수신 기능을 갖추어 실시간 채팅 환경을 구현하였다.

참 고 자 료

- 본 장에서는, 프로젝트 구현을 위해 참고한 자료를 기술할 것

[1] https://study-program.tistory.com/6

[2] https://codingboycc.tistory.com/35