

# 네트워크 프로그래밍

---

다자간 야구게임

8조 ★정하림, 정호길, 김준기, 정주리

# INDEX

---

1

설계 목표

설계 환경

설계 내용

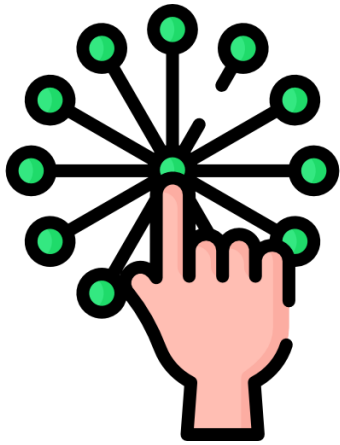
변동 내용

설계 시연

교훈

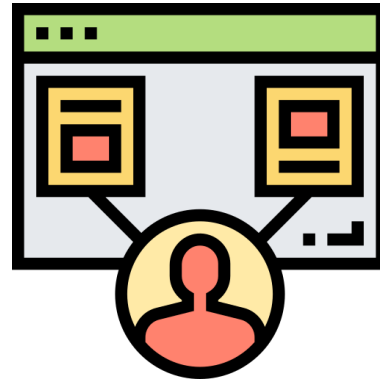
# • DESIGN Objectives

2



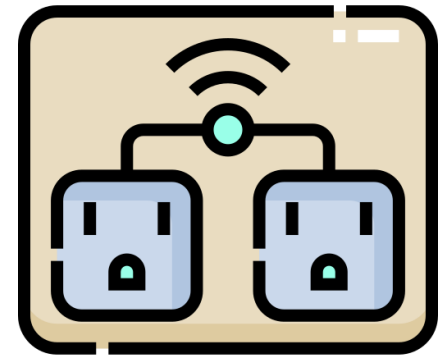
## 멀티 쓰레드를 활용한 다중 접속

멀티 쓰레드 설계를 통해  
다중 사용자처리 서비스를 제공



## DB를 활용한 사용자 정보 관리

사용자 회원 정보, 전적 기록 등을 저장하여  
DB로 관리한다

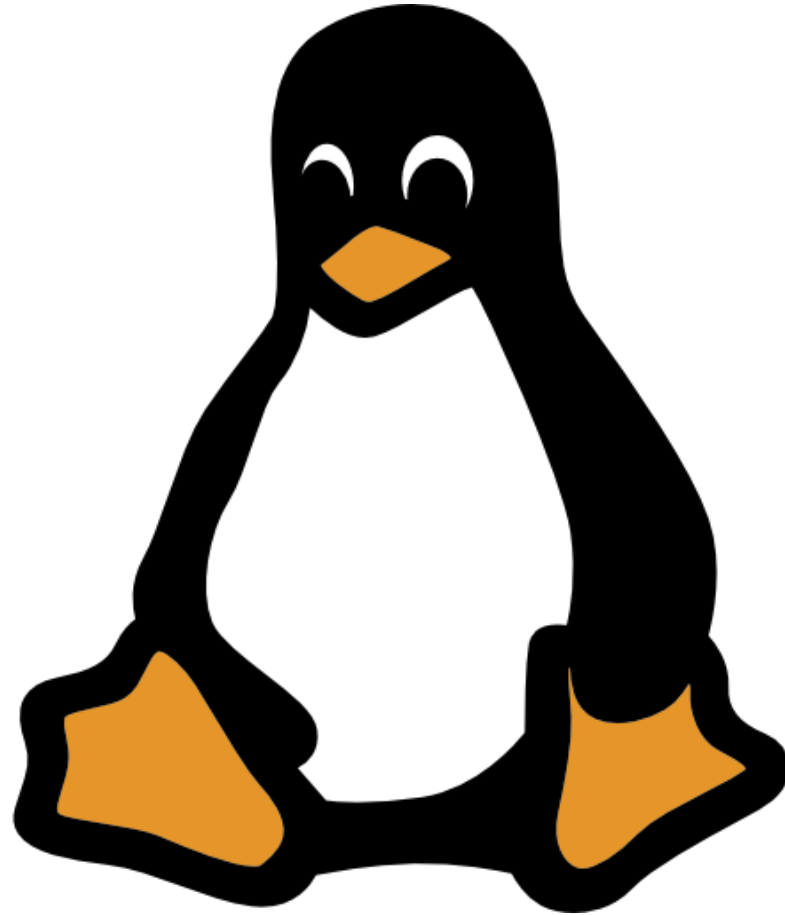


## 소켓을 활용한 네트워크 프로그램

TCP를 이용해서 사용자 통신을 통하여  
사용자간의 숫자 야구 게임을 진행한다

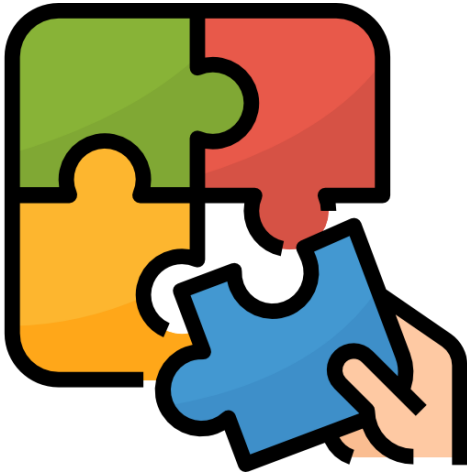
# • DESIGN ENVIORMENT

3



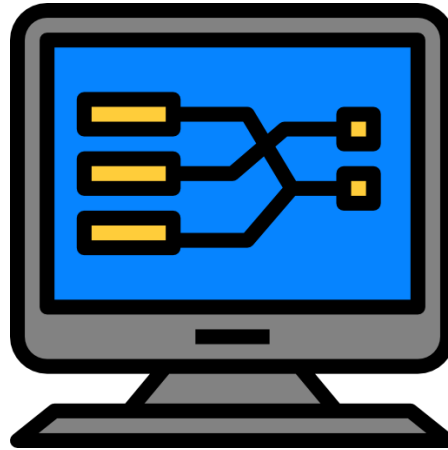
# • DESIGN CONTENTS

4



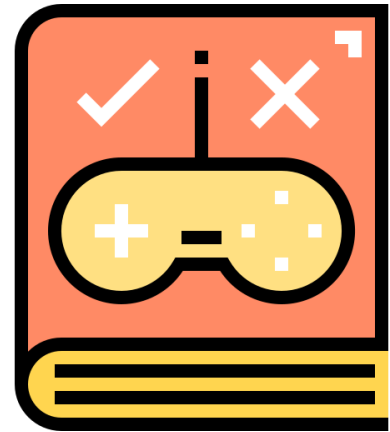
## 1) 시스템

멀티 쓰레드 설계를 통해  
다중 사용자처리 서비스를 제공



## 2) 기능

사용자는 회원가입과 로그인을 통해  
게임에 접속 할 수 있다.



## 3) 진행 방식

유저는 게임을  
어떻게 진행하는가?

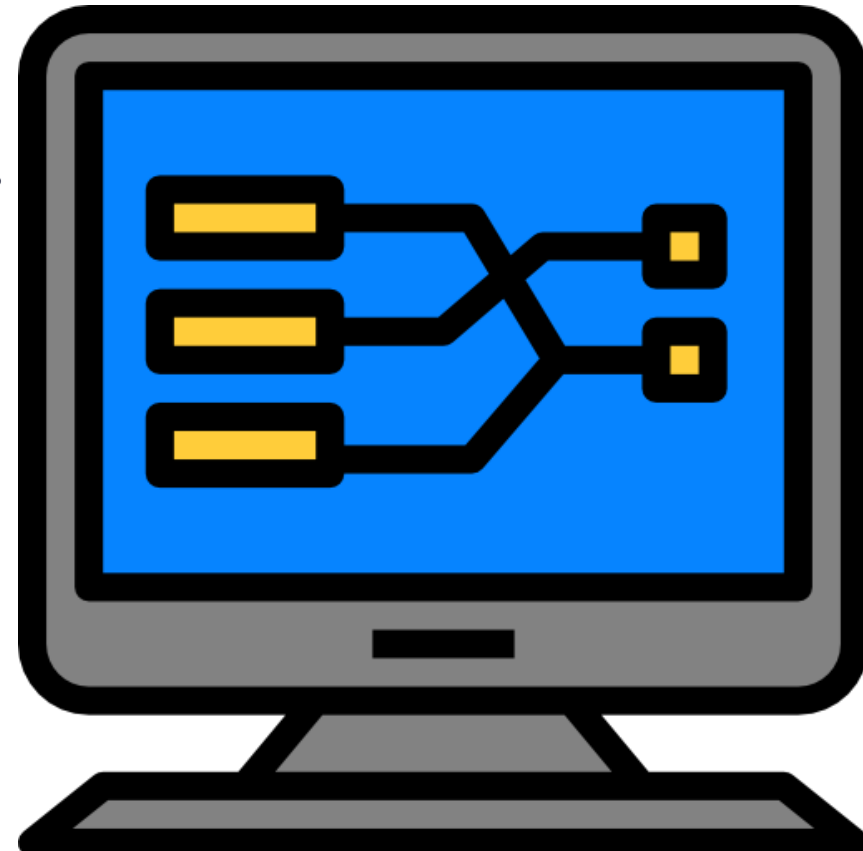
# • 1. SYSTEM

- 멀티쓰레드를 이용하여 다중 접속을 가능하게 한다.
- 유저는 최초 접속 시 회원가입을 해야 한다.
- 유저는 로그인을 해야만 게임에 접속이 가능하다.
- 유저의 아이디는 중 복 될 수 없다.
- 유저의 아이디와 전적은 DB로 관리한다.
- 게임이 완료되면 유저의 전적을 갱신한다.



## • 2. FUNCTION

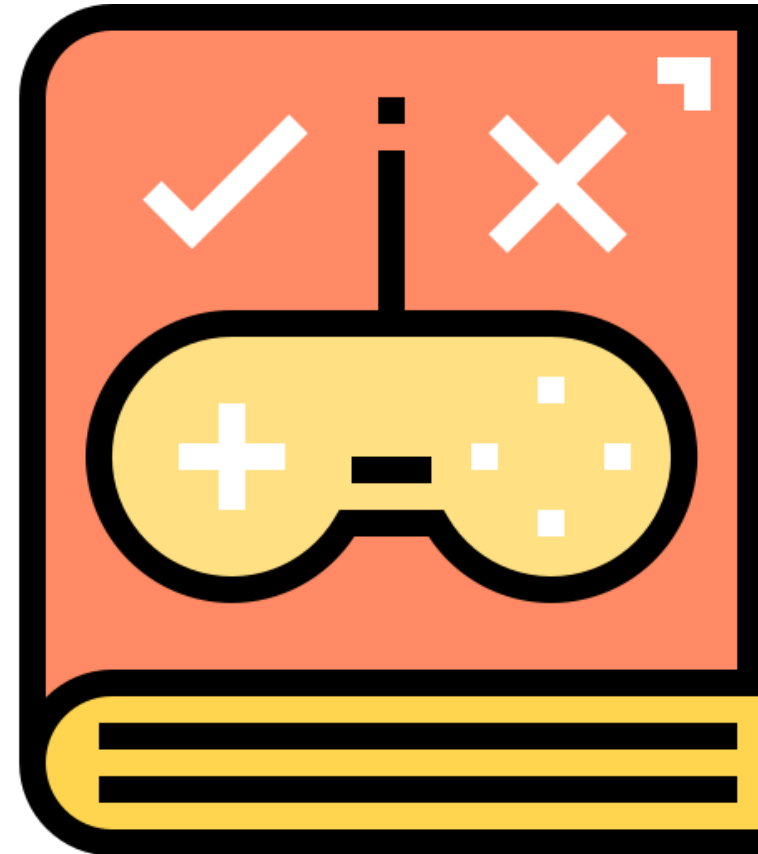
- 유저는 게임 방을 생성 할 수 있다.
- 유저는 방을 확인 할 수 있다.
- 유저는 원하는 방에 입장이 가능하다.
- 유저는 자신 또는 상대방의 전적을 확인 할 수 있다.
- 접속자 명단을 확인 할 수 있다.
- 방에서 다른 유저를 초대할 수 있다.



# • 3. HOW TO PROCEED

7

- 게임 진행 시 상대방 선택 방법은 랜덤 매칭과 사용자가 직접 수동으로 선택하는 2가지 방식이 존재
- 다른 유저 입장 시까지 대기해야 한다.
- 모든 유저가 게임 방 입장 후 준비 상태가 되어야 한다.
- 게임 시작 시 각자 3자리 숫자를 입력하고 상대방의 번호를 맞춘다.(항상 세 자리 숫자로 진행)
- 먼저 입장한 사람부터 숫자를 맞추기를 시작한다.
- 서로 동일한 턴에 숫자를 맞출 시에는 무승부로 기록한다.





```

User* login_view(void* arg) {
    int cInt_sock = *((int*)arg);
    User* user = (User*)malloc(sizeof(User));
    char msg[BUF_SIZE];
    char answer[ANSWER_SIZE];
    int login_result = 0;
    int str_len = 0;

    memset(user, 0, sizeof(user)); // user init

    while (1) {
        str_len = read(cInt_sock, answer, sizeof(answer));
        if (str_len == -1) // read() error
            return 0;
        answer[str_len - 1] = '\0';

        switch (answer[0]) {
            case '1':
                login_result = login(&cInt_sock, user);

                if (login_result == 1)
                    return user; // user 정보를 반환
                else
                    break;
            case '2':
                sign_up(&cInt_sock);
                break;
            case '3':
                printf("클라이언트 종료\n");
                return NULL;
            default:
                break;
        }
    }

    return NULL;
}

```

## 로그인 뷰 " 서버 "

```
// 접속 시 첫 화면으로 로그인을 위한 뷰
User* login_view(void* arg) {
    int sock = *((int*)arg);
    User* user = (User*)malloc(sizeof(User));
    char msg[BUF_SIZE];
    char answer[ANSWER_SIZE];
    int login_result = 0;

    while (1) {
        fputs("\n* * * * * \n", stdout);
        fputs("*                               *\n", stdout);
        fputs("      다자간 숫자야구          *\n", stdout);
        fputs("*                               *\n", stdout);
        fputs("*****\n", stdout);
        fputs("\n\n== 메뉴 ==\n", stdout);
        fputs("\n1. 로그인\n", stdout);
        fputs("\n2. 계정 생성\n", stdout);
        fputs("\n3. 종 료\n", stdout);
        fputs("\n메뉴를 선택해주세요. (1-3). ? ", stdout);
        fflush(stdout);

        fgets(answer, sizeof(answer), stdin);
        write(sock, answer, sizeof(answer));
        fflush(stdin);

        switch (answer[0]) {
            case '1': // 로그인
                login_result = login(&sock, user); // 로그인 결과

                if (login_result == 1) { // 로그인 성공
                    return user;
                } else // 로그인 실패
                    break;
            case '2': // 계정 생성
                sign_up(&sock);
                break;
            case '3':
                fputs("\n연결이 종료됩니다.\n", stdout);
                return NULL; // 종료 요청
            default:
                fputs("\n보기 중 입력해 주세요. (1-3)? \n", stdout);
                break;
        }
    }

    return NULL;
}
```

## 로그인 뷰 "클라이언트"

8

```
// DB에서 아이디 중복 확인 및 추가
sign_up_result = rw_login_db(mode, uid, upw);

if (sign_up_result == 1) {
    answer[0] = '1';
    printf("계정 생성: %s\n", uid);
    sprintf(log_msg, "계정 생성 성공: %s\n", uid);
    write(log_fds[1], log_msg, strlen(log_msg));
}
else {
    answer[0] = '0';
    printf("계정 생성 실패\n");
    sprintf(log_msg, "계정 생성 실패\n");
    write(log_fds[1], log_msg, strlen(log_msg));
}

str_len = write(clnt_sock, answer, strlen(answer));
```

```
pthread_mutex_lock(&login_db_mutex); // login_db.txt mutex lock
if ((fp = fopen("login_db.txt", mode)) == NULL) {

    error_handling("fopen() error");
}
printf("\ndb접속 성공\n");
if (strcmp(mode, "r", strlen(mode)) == 0) { // 로그인에 해당하므로 mode가 read 전용
    // login_db.txt 에서 한줄씩 가져옴 "id pw\n"
    while (fscanf(fp, "%s %s\n", get_id, get_pw) != EOF) {
        if (strcmp(uid, get_id, strlen(get_id)) == 0) { // 아이디 일치
            printf("아이디 일치\n");
            if (strcmp(upw, get_pw, strlen(get_pw)) == 0) { // 비밀번호 일치
                printf("비밀번호 일치\n");
                result = 1; // true
            }
        }
    }
}
else if (strcmp(mode, "r+", strlen(mode)) == 0) { // 회원가입에 해당하므로 mode가 rw
    // login_db.txt 에서 한줄씩 가져옴
    while (fscanf(fp, "%s %s\n", get_id, get_pw) != EOF) {
        if (strcmp(uid, get_id, strlen(get_id)) == 0) { // 아이디 일치
            printf("아이디 중복\n");
            is_duplicated_id = 1; // 아이디 중복 체크
            break;
        }
    }
    // 아이디가 중복되지 않으면 login_db.txt에 새로운 계정 등록
    if (!(is_duplicated_id)) {
        printf("계정 등록\n");
        fprintf(fp, "%s %s\n", uid, upw); // login_db.txt에 계정 등록
        if ((rank_fp = fopen("ranking_db.txt", "a+")) == NULL) { // ranking_db.txt open
            error_handling("fopen() error");
        }
        fprintf(rank_fp, "%s 0 0 0\n", uid); // 랭킹 초기값 등록
        fclose(rank_fp); // ranking_db.txt close
        result = 1; // true
    }
}
fclose(fp); // login_db.txt close
pthread_mutex_unlock(&login_db_mutex); // login_db.txt mutex unlock
```

메인 뷰  
"서버"

메인 뷰  
"클라이언트"

10

```
typedef struct GameController {
    char answer_list[2][3];
    int ready_state[2];
    int turn;
} GameController;
```

```
typedef struct Room {
    int id;
    char user_id[2][USER_ID_SIZE];
    char name[ROOM_NAME_SIZE];
    int cnt_sock[2];
    int cnt;
    GameController gameController;
} Room;
```

```
typedef struct MatchingRoomController {
    Room* room_list[MAX_ROOM];
    int cnt; // matching room count
} MatchingRoomController;
```

```
typedef struct CreatedRoomController {
    Room* room_list[MAX_ROOM];
    int cnt; // created room count
} CreatedRoomController;
```

```
typedef struct User {
    char id[USER_ID_SIZE];
    int sock;
} User;
```

```
typedef struct UserController {
    User* user_list[MAX_CLNT];
    int cnt;
} UserController;
```

```
pthread_mutex_lock(&room_mutex);
room->id = cRoomController.cnt++;
cRoomController.room_list[room->id] = room;
pthread_mutex_unlock(&room_mutex);
```

```
printf("%s : 방 생성 -> %d %s", user->id, room->id, room->name);
sprintf(log_msg, "%s : 방 생성 -> %d %s", user->id, room->id, room->name);
write(log_fds[1], log_msg, strlen(log_msg));
```

유저  
컨트롤러

```
typedef struct User {  
    char id[USER_ID_SIZE];  
    int sock;  
} User;  
  
typedef struct UserController {  
    User* user_list[MAX_CLNT];  
    int cnt;  
} UserController;
```

룸  
컨트롤러

```
typedef struct GameController {  
    char answer_list[2][3];  
    int ready_state[2];  
    int turn;  
} GameController;  
  
typedef struct Room {  
    int id;  
    char user_id[2][USER_ID_SIZE];  
    char name[ROOM_NAME_SIZE];  
    int clnt_sock[2];  
    int cnt;  
    GameController gameController;  
} Room;  
  
typedef struct MatchingRoomController {  
    Room* room_list[MAX_ROOM];  
    int cnt; // matching room count  
} MatchingRoomController;  
  
typedef struct CreatedRoomController {  
    Room* room_list[MAX_ROOM];  
    int cnt; // created room count  
} CreatedRoomController;
```

11

```
Room* get_create_room(User * user) {
    Room* room = (Room*)malloc(sizeof(Room));    //방 생성
    char name[21];
    char str_len;
    room->cnt = 1;
    room->clnt_sock[0] = user->sock;
    memcpy(room->user_id[0], user->id, sizeof(user->id));

    str_len = read(user->sock, name, sizeof(name));
    if (str_len == -1)
        return 0;
    sprintf(room->name, "%s", name);

    pthread_mutex_lock(&room_mutex);
    room->id = cRoomController.cnt++;
    cRoomController.room_list[room->id] = room;
    pthread_mutex_unlock(&room_mutex);

    printf("%s : 방 생성 -> %d %s", user->id, room->id, room->name);
    sprintf(log_msg, "%s : 방 생성 -> %d %s", user->id, room->id, room->name);
    write(log_fds[1], log_msg, strlen(log_msg));

    return room;
}
```

```
// mutex lock...
pthread_mutex_lock(&matching_mutex);
for (int i = 0; i < mRoomController.cnt; i++) {
    in_room_cnt = mRoomController.room_list[i]->cnt;
    if (in_room_cnt < 2) { // empty space in room
        room = mRoomController.room_list[i];
        room->client_sock[in_room_cnt] = user->sock;
        memcpy(room->user_id[in_room_cnt], user->id, sizeof(user->id));
        room->cnt++;
        entered = 1;
        printf("유저 방에 입장\n");

        break;
    }
}

pthread_mutex_unlock(&matching_mutex);
// mutex unlock

if (entered == 1) {
    printf("%s : 매칭 성공\n", user->id);
    sprintf(log_msg, "%s : 매칭 성공\n", user->id);
    write(log_fds[1], log_msg, strlen(log_msg));
    return room;
}
```

```
pthread_mutex_lock(&room_mutex);
for (i = 0; i < cRoomController.room_list[i]->cnt; i++) {
    if (strcmp(msg, cRoomController.room_list[i]->name) == 0
        || (num == cRoomController.room_list[i]->id)) { // 방을 찾으면 접속
        in_room_cnt = cRoomController.room_list[i]->cnt;
        if (in_room_cnt >= 2) {
            printf("방이 가득참\n");
            break;
        }
        else {
            printf("입장 가능\n");
            room = cRoomController.room_list[i];
            room->clnt_sock[in_room_cnt] = user->sock;
            memcpy(room->user_id[1], user->id, sizeof(user->id));
            room->cnt++;
            entered = 1;
            break;
        }
    }
}

pthread_mutex_unlock(&room_mutex);

if (entered == 1) {
    printf("입장성공\n");
    sprintf(log_msg, "%s : 방 입장 성공 -> %s", user->id, room->name);
    write(log_fds[1], log_msg, strlen(log_msg));
    str_len = write(user->sock, "1", sizeof("1"));
    room_view(user, room);
}
else {
    printf("입장 실패\n");
    sprintf(log_msg, "%s : 입장 실패 -> %s", user->id, room->name);
    write(log_fds[1], log_msg, strlen(log_msg));
    str_len = write(user->sock, "0", sizeof("0"));
}
```

```
int input_rank(char id[USER_ID_SIZE], int outcome) {
    FILE* fp = fopen("ranking_db.txt", "rt+");
    if (fp == NULL) error_handling("fopen() error");
    else
    {
        long seek;
        char temp_id[USER_ID_SIZE] = { 0 };
        int w = 0, d = 0, l = 0;
        while (1)
        {
            seek = ftell(fp);
            if (fscanf(fp, "%s %d %d %d\n", temp_id, &w, &d, &l) == EOF) break;
            if (!strcmp(temp_id, id)) {
                switch (outcome) {
                    case 1: w++; break;
                    case 2: d++; break;
                    case 3: l++; break;
                    default: break;
                }
                fseek(fp, seek, SEEK_SET);
                fprintf(fp, "%s %d %d %d\n", temp_id, w, d, l);
                fflush(fp);
            }
        }
        rewind(fp);
        fclose(fp);
    }
}
```



```

void print_ranking(User* user) {
    Stat stat[MAX_CLNT] = { 0 };
    Stat temp;
    char rankingme[100] = { 0 };
    char rankingall[1000] = { 0 };
    int record = 0;
    int i, j;
    FILE* fp = NULL;

    if ((fp = fopen("ranking_db.txt", "r")) == NULL) {
        error_handling("fopen() error");
    }
    else {
        while ((fscanf(fp, "%s %d %d %d\n", stat[record].id, &stat[record].win, &stat[record].draw, &stat[record].lose)) != EOF){
            record++;
        }
    }
    fclose(fp);

    for (i = record - 1; i > 0; i--) {
        for (j = 0; j < i; j++) {
            if (stat[j].win < stat[j + 1].win) { // 승수를 비교하여 자리이동
                temp = stat[j + 1];
                stat[j + 1] = stat[j];
                stat[j] = temp;
            }
            // 승수가 같다면 패배수가 작은 순
            else if (stat[j].win == stat[j + 1].win && stat[j].lose > stat[j + 1].lose) {
                temp = stat[j + 1];
                stat[j + 1] = stat[j];
                stat[j] = temp;
            }
            // 패배수마저 같다면 무승부가 많은 순
            else if (stat[j].win == stat[j + 1].win && stat[j].lose == stat[j + 1].lose && stat[j].draw < stat[j + 1].draw) {
                temp = stat[j + 1];
                stat[j + 1] = stat[j];
                stat[j] = temp;
            }
        }
    }

    for (i = 0; i < record; i++) { // 유저 리스트 복사
        sprintf(rankingme, "%3d등 %15s %5d %6d %6d", i + 1, stat[i].id, stat[i].win, stat[i].draw, stat[i].lose);
        strcat(rankingall, rankingme);
        strcat(rankingall, "\n");
    }

    write(user->sock, rankingall, strlen(rankingall)); // 클라이언트에게 전송
}

```

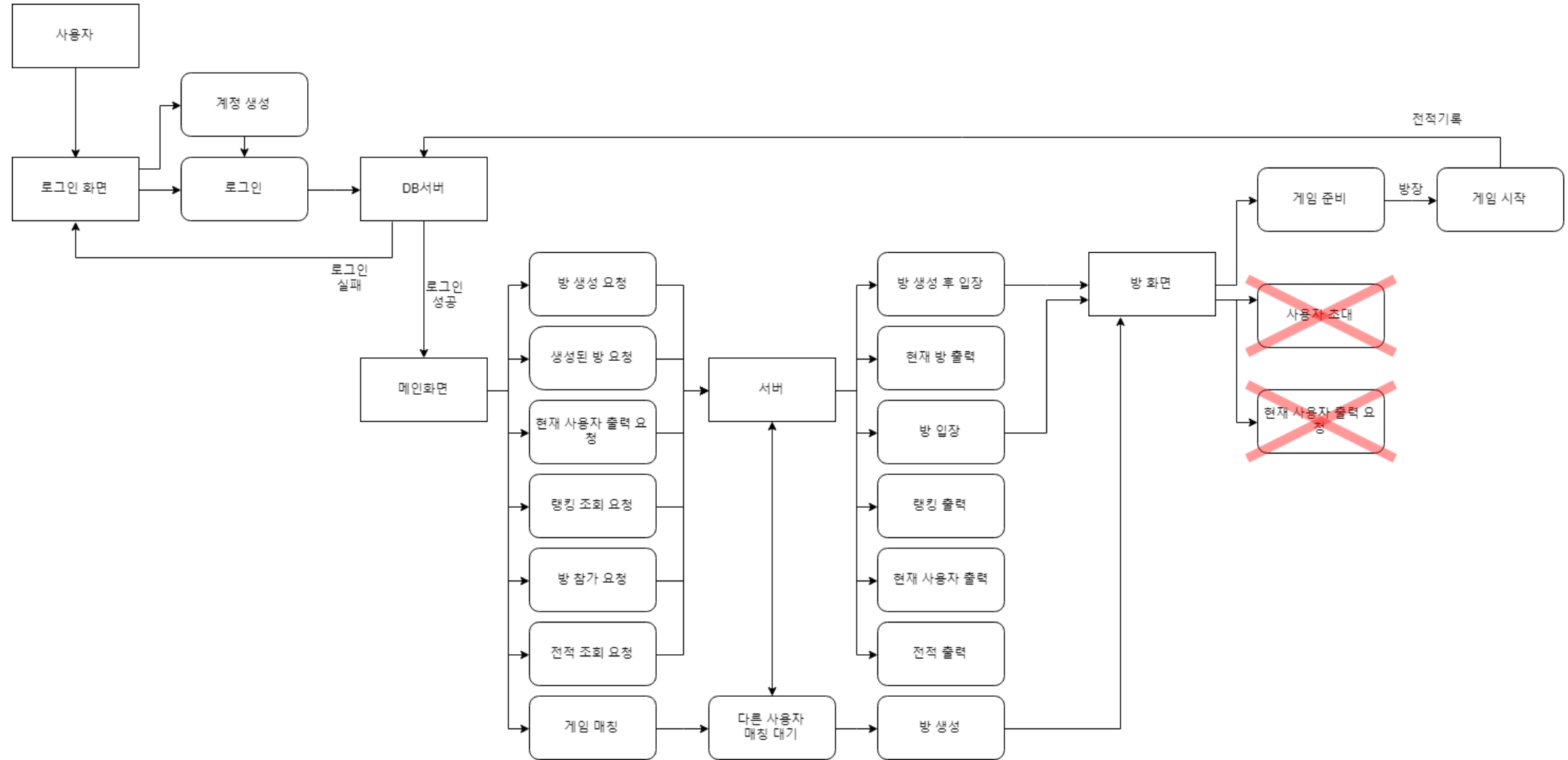
```
pipe(log_fds);

void* write_log() {          // 로그 작성 파이프함수
    int len;
    FILE* fp = NULL;
    while (1) {
        char buf[100] = { 0, };
        len = read(log_fds[0], buf, sizeof(buf));
        if ((fp = fopen("log.txt", "a")) != NULL) {
            fprintf(fp, "%s", buf);
            fclose(fp);
        }
        else {
            printf("파일 열기 오류\n");
        }
    }
    return NULL;
}
```

```
if (sign_up_result == 1) {
    answer[0] = '1';
    printf("계정 생성: %s\n", uid);
    sprintf(log_msg, "계정 생성 성공: %s\n", uid);
    write(log_fds[1], log_msg, strlen(log_msg));
}
else {
    answer[0] = '0';
    printf("계정 생성 실패\n");
    sprintf(log_msg, "계정 생성 실패\n");
    write(log_fds[1], log_msg, strlen(log_msg));
}
```

# • CHANGES

18



# • RESULT

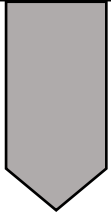
19




# LESSONS LEARNED

---

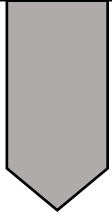
20



read, write가  
예상과 다르게  
작동하는 것을 보며  
tcp가 데이터의 경계가  
없다는 것을  
확실히 알게 되었다.



서버와 클라이언트를  
따로 구현하는 것이  
생각보다 어려웠다.  
설계 단계의  
중요성을 깨닫게 되었다.



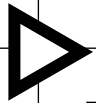
다중 접속을 생각하여  
구현해본 것은 처음인데  
동기화 문제가 중요하  
다는 것을 알게 되었다.

질문  
받습니다.

다자간 야구게임

8조 ★정하림, 정호길, 김준기, 정주리

감사합니다  
:-)



다자간 야구게임

8조 ★정하림, 정호길, 김준기, 정주리