시스템소프트웨어보안 프로젝트

사이버보안학과
201920663 이진근

<자가진단표>

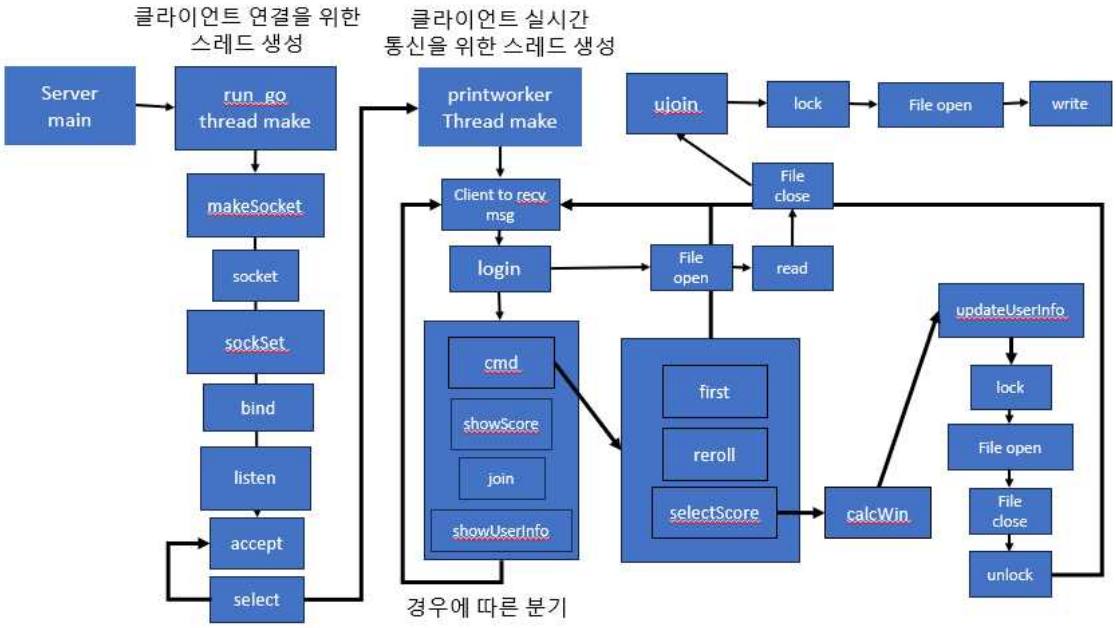
	창의 단계 (단계에 체크√)	단계 기준	주요 창의적 내용	
창의성	5■	전혀 새로운 창의적인 프로그램	야찌 주사위 게임 서버 개발 및 클라이언트 간의 1대1 실시간 멀티 플레이 게임	
	4□	예제 프로그램 또는 기존 프로그램을 참고로 새로운 프로그램 제작		
	3□	예제 프로그램 확장		
	2□	예제 프로그램 일부 수정		
	1□	단순한 예제 프로그램 수준		
난이도/복잡도	복잡 단계 (단계에 체크)	단계 기준	사용한 시스템호출과 라이브러리	고급 난이도 기술
	5■	소켓, concurrency, 동기화, 파일 억세스, 기타 다양한 고급 기능을 포함	stdio.h: 표준 입출력 함수 사용을 위해 포함 stdlib.h: 일반적인 유틸리티 함수 사용을 위해 포함 sys/time.h: 시간 관련 함수 사용을 위해 포함 sys/types.h: 시스템 타입 정의를 위해 포함 sys/socket.h: 소켓 프로그래밍 함수 사용을 위해 포함 string.h: 문자열 관련 함수 사용을 위해 포함 netinet/ip.h: IP 헤더 구조체 및 상수 정의를 위해 포함 netinet/in.h: 인터넷 프로토콜 관련 상수 및 구조체 정의를 위해 포함 pthread.h: POSIX 쓰레드 관련 함수 사용을 위해 포함 unistd.h: UNIX 표준 함수 사용을 위해 포함	select 함수 사용하여 socket fd 관리 kernel API를 이용해 linked list 구현 및 이를 이용한 socket fd 관리 파일 입출력을 이용한 유저의 정보 관리
	4□	소켓, concurrency, 동기화, 파일 억세스 포함		
	3□	네트워크 프로그래밍을 이용한 클라이언트-서버 모델 구현		
	2□	기본적인 클라이언트-서버 모델 구현		
	1□	부분적인 기능 구현		
완성도	완성 단계 (단계에 체크)	내용	주요 bug 및 프로그램의 문제점	
	5■	모든 기능의 안정적인 동작		
	4□	주요 기능을 포함한 대부분 동작 (일부 버그 존재)		
	3□	제한된 환경에서 일부 기능 동작		
	2□	컴파일 완성 후 주요 동작 불가		
	1□	프로그램 제작후 컴파일 불가		

2. 목적
과제 조건에 맞는 서버와 클라이언트가 통신하고, 쓰레드, 파일 입출력 등을 사용하고 concurrency 한 실시간 통신서비스를 생각하면 게임이 적합하겠다고 생각했다.
누구나 쉽게 할 수 있는 1대1로 진행가능한 야찌라는 주사위 게임을 만들었다.

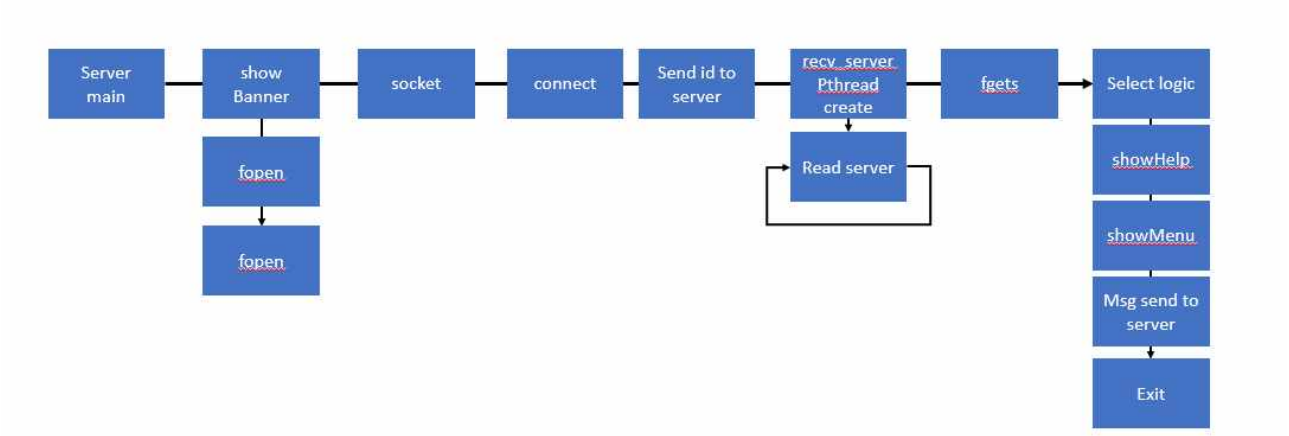
3. 기능
기술적 기능
클라이언트-서버 아키텍처
게임을 구현하기 위해 클라이언트와 서버 간의 통신을 수행하는 클라이언트-서버 아키텍처를 구성했다. 클라이언트는 서버에 접속하여 게임 데이터를 송수신하고, 서버는 클라이언트 간의 상호작용을 조율하고 게임 로직을 처리하도록 구현했다.
concurrency 한 통신
클라이언트와 서버 간의 통신에는 소켓 프로그래밍을 활용한다. 클라이언트는 소켓을 생성하여 서버에 연결하고, 서버는 소켓을 통해 클라이언트와 데이터를 주고받습니다. 이를 통해 실시간 게임 데이터를 전송하고 수신한다. 하지만 싱글스레드로 구현하면 하나의 클라이언트만 사용이 가능하기에 select 시스템콜을 이용해 소켓을 관리하고 클라이언트와의 통신을 별도의 쓰레드로 처리하여 동시에 여러 클라이언트와 상호작용할 수 있도록 했다.

서비스 관점의 기능
- 닉네임을 통한 회원관리
게임 전적을 저장하고 조회할 수 있도록 개발하였다. 파일 입출력을 통해 로그인 및 회원가입을 구현하였다.
- 게임방에 접속
기다리는 사람을 저장하여 게임을 원하는 사람끼리 매칭할 수 있도록 구현했다.
- 주사위 던지기
야찌 주사위 게임의 규칙에 따라 총 3번까지 주사위를 던질 수 있고 mutex와 turn을 명시하는 구조체를 선언하고 이를 점유하는 방식으로 게임에서의 턴을 점유할 수 있도록 구현했다.
- 점수계산
주사위 던지기 단계가 끝나면 원하는 족보에 해당하는 점수를 기입할 수 있는데 기입 시 점수를 계산한다.
- 전적 확인
게임에 매칭된 상대방 및 본인의 현재까지 전적을 확인할 수 있다.
4. 사용한 시스템 콜
본 서버 프로그램 및 클라이언트 프로그램에서
read -> 통신 시 받는 역할을 할 때 read를 사용한다. socket fd에서 read한다.
write -> 통신 시 데이터를 전송할 때 write를 사용한다. socket fd에 write하면 데이터가 전송된다.
select -> 다중 I/O 작업을 효율적으로 처리하기 위해 사용했다. client가 accept되면 이를 관리하기 위해서 select 시스템 콜을 사용했고 socket fd를 multiflex로 관리가 가능하다.
socket -> 네트워크 통신을 위한 시스템 콜입니다. 소켓은 서버에서 클라이언트 컴퓨터를 연결하기 위해 사용하고 클라이언트에서 서버에 연결하기 위해 사용했다.
connect -> TCP 소켓을 사용하기 위해서 사용했다. 서버는 bind, listen을 사용하기 위해 사용하고 클라이언트는 connect를 사용하기 위해 만들었다.
bind -> 포트번호를 소켓에 할당하기 위해 사용했다.
listen -> bind에서 만든 소켓을 실제 통신할 수 있도록 listen 시스템 콜을 통해서 연결을 기다린다.
accept -> 서버에서 bind 이후 연결이 된 클라이언트의 socket을 받아올 수 있도록 사용했다.
exit -> 프로세스를 종료하기 위해서 사용한 시스템 콜이다. 클라이언트와 서버 프로그램에서 특정 커맨드

를 실행시키면 프로그램을 종료하도록 설계하고 구현하였다.
time -> 야찌 주사위 게임에서 주사위를 던질 때 random하게 주사위가 나와야하기에 random의 인자를 시간으로 주었다.
5. 시스템구성도



Server Side 시스템 구성도



ClientSide 시스템 구성도

6. 구현 내용 및 방법

Server Side

```c
int main(int argc, char **argv){
    if(argc < 2){
        printf("%s PORT\n",argv[0]);
        return 0;
    }
    printf("initialized...\n");
    srand(time(NULL));
    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&fileMutex, NULL);
    pthread_t run_t;
    int p = atoi(argv[1]);
    pthread_create(&run_t, NULL, run_go,(void*)(&p));
    sleep(1);
    while(1){
        char buf[100];
        printf("$ ");
        scanf("%s",buf);
        if(exitCheck(buf))
            break;

    }
    printf("end\n");
    return 0;
}
```

main함수, mutex를 initialize 하고 rand 함수를 사용할 때 항상 random한 숫자를 가져오려고 srand 함수를 사용하고 안에 time()으로 시드를 넣어주었다.

또한 run_go 함수에 argv[1] 즉, 서버를 실행할 때 받는 첫 번째 인자를 포트로 주기 위해 인자로 주었다. 인자가 없다면 프로그램을 종료하고 올바른 실행 방법을 알려준다.

```c
void *run_go(void* p){
    int* port = (int*)(p);
    printf("server on %d\n",*port);
    play.id = -1;
    server_socket = makeSocket();
    sockSet(*port);
    if(-1 == bind(server_socket,(struct sockaddr*)&server_addr, sizeof(server_addr))){
        fprintf(stderr,"bind error\n");
        exit(1);
    }
    if(-1 == listen(server_socket, 5)){
        fprintf(stderr,"listen err\n");
        exit(1);
    }
    int client_addr_size;
    client_addr_size = sizeof(client_addr);
    fd_set reads, cpy_reads;
    int maxi,maxfd,fd_num;
    struct timeval timeout;
    int client;
    maxfd = server_socket;
    // socket linked list
    struct socketInfo* serverEntry = (struct socketInfo*) malloc(sizeof(struct socketInfo));
    INIT_LIST_HEAD(&serverEntry->list);
    serverEntry->fd = server_socket;
    list_add_tail(&serverEntry->list,&head);
    // socket linked list end
    FD_ZERO(&reads);
    FD_SET(server_socket, &reads);
    pthread_t threads[7];
    int threadIdx = 0;
    while(1){
        cpy_reads = reads;
        timeout.tv_sec=5;
        timeout.tv_usec=5000;
        struct socketInfo* last = list_last_entry(&head,struct socketInfo,list);
        fd_num = select(maxfd+1, &cpy_reads, 0, 0, &timeout);
        if(fd_num == -1){
            fprintf(stderr,"fd_errrrr\n");
            maxfd = server_socket;
            continue;
        }
        else if(fd_num == 0){
            continue;
        }
        int i=0;
        for(;i<maxfd+1;i++){
            if(FD_ISSET(i,&cpy_reads)){
                    printf("maxfd : %d\n",maxfd);
                    client = accept(server_socket, (struct sockaddr*)&client_addr,&client_addr_size);
                    if (client == -1){
                        fprintf(stderr,"client conn err\n");
                        break;
                    }
                    else{
                        FD_SET(client,&reads);
                        if(maxfd < client)
                            maxfd=client;
                        printf("client Connect : %d\n",client);
                        struct socketInfo* new = (struct socketInfo*) malloc(sizeof(struct socketInfo));
                        new->fd = client;
                        new->id = clientCnt;
                        new->vs = NULL;
                        new->dices[0] = -1;
                        for(int i=0;i<13;i++){
                            new->scores[i] = -1;
                        }
                        INIT_LIST_HEAD(&new->list);
                        list_add_tail(&new->list,&head);
                        printf("client %d\n",client);
                        printf("clints[?] = %d\n",new->fd);
                        pthread_create(&threads[threadIdx], NULL, printWorker, (void*)new);
                        clientCnt++;
                        threadIdx++;
                    }
            }
        }
    }
}
```

run_go는 스레드로 실행되는 함수이고 sockSet함수를 통해 소켓을 세팅하고 makeSocket 함수를 통해서 소켓을 생성한다.

이후 select 시스템 콜로 소켓을 관리할 것이기 때문에 서버의 소켓을 미리 저장하기 위해 미리 정의해둔 구조체를 통해 커널 API를 통해서 서버의 소켓을 링크드리스트 0번째 요소로 미리 등록하고 이후 accept를 통해서 연결된 클라이언트의 소켓들을 차례차례 링크드리스트로 등록해서 관리한다. 또한 연결된 커넥션을 따로따로 실시간으로 관리하기 위해서 클라이언트에서 주는 입력을 관리하기 위해서 printWorker 함수를 스레드를 사용했다. 따라서 연결된 클라이언트 수 만큼 스레드가 생성된다.

```
void sockSet(int port){
    memset(&server_addr, 0, sizeof(server_addr));
    memset(&client_addr, 0, sizeof(client_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
}
```

socket을 셋팅하는 함수.

```
1  int makeSocket(){
2      int socket_fd;
3      socket_fd= socket(PF_INET, SOCK_STREAM, 0);
4      if (socket_fd == -1){
5          fprintf(stderr, "socket fd err\n");
6          exit(1);
7      }
8      return socket_fd;
9  }
```

서버에서 사용할 tcp socket을 만드는 함수

```
void join(struct socketInfo* c){
    pthread_mutex_lock(&mutex);
    if(ready == NULL){
        if(play.id != -1){
            clientSend(c, "played game new\n");
        }
        else{
            ready = c;
            clientSend(c, "[!] join complate..\n");
        }
    }
    else{
        if(c->id != ready->id){
            ready->vs = c;
            c->vs = ready;
            clientSend(ready, "[!] your 1p your turn!\n");
            clientSend(c, "[!] your 2p\n");
            play.id = ready->id;
            ready = NULL;
            for(int i=0;i<13;i++){
                    c->scores[i] = -1;
                    c->vs->scores[i] = -1;
            }//sori
        }
        else{
            clientSend(c, "[!] waitting plz.\n");
        }
    }
    pthread_mutex_unlock(&mutex);
}
```

게임 방에 들어가는 로직이다.

클라이언트의 세션을 스레드로 관리하기 때문에 turn을 관리하는 것을 mutex로 제한하여 한번에 한명만 들어갈 수 있도록 사용했다.

play라는 구조체로 id를 등록해 먼저 들어온 사람이 선을 가지게 되고 ready라는 구조체 포인터를 통해 대기열을 구성한다. 대기열이 있다면 후순위로 턴을 가진다.

```c
void ujoin(struct socketInfo* c,char* userid){
    pthread_mutex_lock(&fileMutex);
    printf("uuuu\n");
    FILE* fp = fopen("users.txt","a+");
    printf("ujoin\n");
    fprintf(fp,"%s 0 0 0\n",userid);
    fclose(fp);
    strcpy(c->infos.name,userid);
    c->infos.win = 0;
    c->infos.draw = 0;
    c->infos.lose = 0;
    printf("joinend\n");
    pthread_mutex_unlock(&fileMutex);
}
```

사용자 회원가입 로직
파일을 lock해서 회원가입 시 동시 접근을 막아 읽을 때 무결성을 보장해 준다

```c
void clientSend(struct socketInfo* c,char* str){
    char buf[1025];
    strcpy(buf,str);
    buf[1024] = '\0';
    write(c->fd, buf, strlen(buf));
}
```

```c
void clientRecv(struct socketInfo* c){
    char buf[1025];
    read(c->fd,buf,1024);
    buf[1024] = '\0';
    printf("id[%d] : %s\n",c->id, buf);
}
```

socket을 이용한 recv와 send를 편리하게 하기 위해 함수를 구현하였다.

```c
int rollDice(int w){
    int dice = -1;
    int ran = rand()+w;
    dice = ran%6+1;
    return dice;
}
```

```c
void printClient(){
    struct socketInfo* p = NULL;
    printf("====================\n[");
    list_for_each_entry(p,&head,list){
        if(p->fd != 3)
            printf(" | %d | ",p->fd);
    }
    printf("]\n====================\n");
}
```

주사위를 굴릴 때 random 하게 1~6이 나오도록 구현하였다. 원래는 주사위를 던질 때 사용자가 가중치를 더할 수 있도록 구현하려고 했는데 실제 프로그램에서는 그냥 모든 값을 3으로 인자로 주었다.

```c
void first(struct socketInfo* c, char *buf){
    char msg[100] = "";
    c->dices[0] = rollDice(3);
    c->dices[1] = rollDice(3);
    c->dices[2] = rollDice(3);
    c->dices[3] = rollDice(3);
    c->dices[4] = rollDice(3);
    sprintf(msg,"first dice :%d %d %d %d %d",c->dices[0], c->dices[1], c->dices[2], c->dices[3],c->dices[4]);
    printf("%s\n",msg);
    clientSend(c,msg);
}
```

첫 턴에는 무조건 주사위를 굴려야해서 랜덤하게 주사위를 굴리도록 했다.
이 결과를 클라이언트에게 전달해준다.

```c
int reroll(struct socketInfo* c, char *buf){
    printf("reroll \n");
    char* flag[5] = {"",};
    int flagIdx = 0;
    char msg[100] = "";
    char nullStr[2] = "1";
    if(strcmp(buf,"end\n") == 0){
        printf("return\n");
        sprintf(msg,"dice :%d %d %d %d %d\n",c->dices[0], c->dices[1], c->dices[2], c->dices[3],c->dices[4]);
        clientSend(c,msg);
        return 1;
    }
    else{
    parse_command(buf, &flagIdx, flag);
    printf("buf : %s\n",buf);
    printf("%d %s %s %s %s %s\n",flagIdx,flag[0],flag[1],flag[2],flag[3],flag[4]);
    printf("rerolllll\n");
    for(int step = 0;step<5;step++){
        printf("[@] catch %s @\n",flag[step]);
        if(flag[step] == NULL){
            printf("NULL check\n");
            //strcpy(flag[step],"0");
            flag[step] = nullStr;
        }
        if(strcmp(flag[step],"0") != 0){
            printf("change change change\n");
            c->dices[step] = rollDice(3);
        }
        else{
            printf("%s %s %d\n",flag[step],"0",strcmp(flag[step],"0") != 0);
            printf("hehehehehehh\n");
        }
    }
    sprintf(msg,"reroll dice :%d %d %d %d %d\n",c->dices[0], c->dices[1], c->dices[2], c->dices[3],c->dices[4]);
    printf("%s\n",msg);
    clientSend(c,msg);
    }
    return 0;
}
```

parse_command 함수를 통해 스페이스바를 기준으로 tokenize한다. 사용자의 입력이 end라면 바로 점수 입력 단계로 진입하도록 하기위해 1을 리턴하고, 이후 입력값에 따라 0이면 해당하는 위치에 주사위를 변경하지 않고 0이 아니라면 다시 주사위를 던진다.

```c
int exitCheck(char* buf){
    char* exitstr="exit";
    if(strcmp(buf,exitstr) == 0 ){
        struct socketInfo* pos;
        list_for_each_entry_reverse(pos, &head,list){
            strcpy(buf,"byebye\n");
                    write(pos->fd, buf, strlen(buf));
                    close(pos->fd);
        }
        return 1;
    }
    return 0;
}
```

서버에서 exit 문자열을 치면 byebye라는 문자열을 연결된 모든 클라이언트에게 보낸다.
클라이언트는 byebye 문자열을 받으면 소켓을 close하도록 만들었고 서버도 클라이언트의 fd를 close한다. 저 링크드리스트에는 서버 자기 자신의 소켓도 존재하기 때문에 list_for_each_entry_reverse 함수에서 가장 마지막 요소로 삭제가 된다.

```c
void *printWorker(void* p){
    struct socketInfo* clientEntry = (struct socketInfo*)p;
    struct socketInfo* vs = NULL;
    printf("print worker %d\n",clientEntry->fd);
    char msg[100] = "";
    char userid[16]="";
    read(clientEntry->fd,userid,sizeof(userid));
    printf("uid : %s\n",userid);
    login(clientEntry,userid);
    sprintf(msg,"%s %d %d %d\n",clientEntry->infos.name,clientEntry->infos.win,clientEntry->infos.draw,clientEntry->infos.lose);
    write(clientEntry->fd, msg, strlen(msg));

    while(1){
        char buf[1025] = {0};
        read(clientEntry->fd,buf,1024);
        buf[1024] = '\0';
        if(buf[0] == 'e' && buf[1] == 'x' && buf[2]=='i' && buf[3]=='t' && buf[4]=='\n'){
            printf("[!] %d disconnected\n",clientEntry->fd);
            strcpy(buf,"byebye\n");
            write(clientEntry->fd, buf, strlen(buf));
            close(clientEntry->fd);
            list_del(&clientEntry->list);
            clientCnt--;
            free(clientEntry);
            if(clientEntry->vs == NULL){
                ready = NULL;
            }
            printClient();
            break;
        }
        if(buf[0] !='\0'){
            if(strcmp(buf,"show\n") == 0 && play.id != -1){
                showScore(clientEntry);
            }
            else if(strcmp(buf,"join\n") == 0){
                join(clientEntry);
            }
            else if(strcmp(buf,"userinfo\n") == 0){
                showUserInfo(clientEntry);
            }
            else {
                cmd(clientEntry,buf);
            }

        }
        else{
            printf("%c",buf[0]);
        }
    }
}
```

printWorker는 accept 이후 각각의 클라이언트의 입력을 받는 것을 처리하는 함수로 각각의 세션이 생성되면 스레드를 생성해서 동작한다.
클라이언트에서 exit 함수를 보내면 그 세션을 끊도록 하고 즉시 매칭을 종료한다.
게임이 진행중인 상태에서 show 명령을 치면 showScore 함수가 작동해 현재 매칭되는 게임의 상태를 보여주고, join 명령을 치면 게임 방을 만들거나 만들어져 있다면 들어간다.
userinfo는 현재 자신의 전적을 보여주고 매칭 중이라면 상대방의 점수도 보여준다.
cmd는 게임이 매칭됐을 때 로직을 처리하는 부분이다.

```
void cmd(struct socketInfo* c, char *buf){
    int endflag=1;
    if(play.id == c->id && (c->vs != NULL)){
        pthread_mutex_lock(&play.turn);
        if(c->step  == 0){
            first(c,buf);
            c->step++;
        }
        else{
            if(c->step == 1){
                int res = reroll(c,buf);
                printf("res : %d\n",res);
                if(res){
                    c->step = 3;
                }
                else{
                    c->step++;
                }
            }
            else if(c->step == 2){
                reroll(c,buf);
                c->step++;
            }
            else if(c->step >= 3){
                int res = selectScore(c,buf);
                if(res){
                    c->step = 0;
                    play.id = c->vs->id;
                    clientSend(c, "[!] turn end\n");
                    clientSend(c->vs, "[!] your turn\n");
                    c->dices[0] = -1;
                    for(int u=0;u<13;u++){//step 13
                        if(c->scores[u] == -1 || c->vs->scores[u] == -1){
                            endflag = 0;
                        }
                    }
                    if(endflag){
                        printf("@@@@calc win\n");
                        calcWin(c);
                    }
                }
                else{
                    clientSend(c, "[!] re try!\n");
                    printf("re select\n");
                }

            }
            else{
                printf("c->step : %d\n",c->step);
                printf("err\n");
            }
        }
        printf("id [%d]: %s\n",c->id,buf);
        pthread_mutex_unlock(&play.turn);
    }
    else{
            printf("[%d] no your turn \n",c->id);
            clientSend(c,"[%d] no your turn \n");
    }
}
```

play라는 구조체로 현재 턴을 누가 가지고 있는지 기록하기 때문에 선공이 무조건 play.id의 값과 같을 것이고 매칭이 되지 않았다면 c->vs는 NULL일 것이다.

자기가 턴을 가지고 있지않고 상대가 없다면 no match를 출력하고 상대가 있고 자기가 턴을 가지고 있다면 클라이언트가 몇 번 스텝인지 확인해서 각 단계에 맞게 행동한다. 첫 턴이라면 first함수를 통해 먼저 주사위를 던지고 두 번째 이상이라면 클라이언트가 전송한 메시지가 end라면 점수 기입단계로, 정수 데이터가 온다면 그걸 기반으로 주사위를 새로 던진다. 점수기입 단계 마지막에 모든 점수가 기입이 되었는지

확인하고 기입이 되었다면 누가 이겼는지 검사하는 함수인 calcWin함수를 호출한다.
세 번째 스텝이라면 주사위를 마지막으로 던지고 selectScore 함수를 통해 족보별로 점수를 기입할 수 있다.
모든 단계별로 단계 시작시 lock을 잡고 단계가 끝나면 unlock한다.

```c
int selectScore(struct socketInfo* c, char *buf){
    char* s[10];
    int cnt = 0;
    int ret = 0;
    int tmp = 0;
    printf("select score\n");
    parse_command(buf,&cnt,s);
    if(cnt == 2){
        printf("set score : %d\n",c->scores[0]);
        printf("s[0] = %s\n",s[0]);
        if(strcmp(s[0], "set") == 0){
            int num = atoi(s[1]);
            printf("num : %d\n",num);
            if(num <= 13 && num>= 1){
                if(c->scores[num-1] == -1){
                    c->scores[num-1] = calcScore(c,num);
                    c->scores[14] += calcScore(c,num);
                    for(int i=0;i<6;i++){//bonus calc
                        if(c->scores[i] == -1){
                            tmp += 0;
                        }
                        else{
                            tmp += c->scores[i];
                        }
                    }
                    if(c->scores[13] == -1 && tmp >= 63){
                        c->scores[13] = 35;
                        c->scores[14] += 35;
                    }
                    ret = 1;
                }
                else{
                    ret = 0;
                }
            }
            else{
                ret = 0;
            }
        }
        else{
            ret = 0;
        }
    }
    else{
        ret = 0;
    }
    return ret;
}
```

selectScore함수는 클라이언트가 전송한 메시지를 스페이스바 단위로 tokenize해서 0번째 단어가 set이라면 족보에 따라 기입해주는 기능의 분기를 한다. set 뒤의 인자가 1~13이라면 이전에 해당 족보에 점수를 기입한 적이 있는지 확인하고(기입한 적이 없으면 초기화 값인 -1이 기입되어 있음) 기입한 적 없다면

족보에 따라 점수를 계산하도록 calcScore 함수를 실행한뒤 결과를 실제 점수를 저장하는 구조체내의 정수 배열에 저장한다. score[14]는 점수의 총 합이기 때문에 모든 결과를 더한다. 점수 기입에 성공한다면 1을 리턴하고 실패한다면 0을 리턴한다.
점수 기입 이후 1,2,3,4,5,6 족보에 해당하는 숫자가 63 이상이라면 보너스로 35점을 더 주도록 계산한다

```
int calcScore(struct socketInfo* c, int num){
    int s = 0;
    if(num == 1){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 1){
                s += 1;
            }
        }
    }
    if(num == 2){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 2){
                s += 2;
            }
        }
    }
    if(num == 3){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 3){
                s += 3;
            }
        }
    }
    if(num == 4){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 4){
                s += 4;
            }
        }
    }
    if(num == 5){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 5){
                s += 5;
            }
        }
    }
    if(num == 6){
        for(int i=0;i<5;i++){
            if(c->dices[i] == 6){
                s += 6;
            }
        }
    }
```

이제부터는 야찌 주사위게임 족보를 기반으로 점수를 계산하는 로직이다. 1~6은 각각 주사위의 개수에 따라 점수를 계산한다. 예를 들어 1을 선택하면 던진 5개의 주사위 중 1이 나온 주사위의 합이 점수가 되

고, 5를 선택하면 5개의 주사위 중 5가 나온 주사위의 합이 점수가 된다.

```
if(num == 7){//three of a kind
    int cnt[6] = {0};
    int flag = 0;
    for(int i=0;i<5;i++){
        for(int j=1;j<=6;j++){
            if(c->dices[i] == j){
                cnt[j-1] += 1;
            }
        }
    }
    for(int i=0;i<6;i++){
        if(cnt[i] == 3){
            flag = 1;
        }
    }
    if(flag){
        for(int i=0;i<5;i++){
            s += c->dices[i];
        }
    }
    else{
        s = 0;
    }
}
```

three of a kind라는 족보를 계산한다. 같은 수가 3개 이상 나오면 모든 주사위의 합을 기입한다.

```
if(num == 8){//four of a kind
    int cnt[6] = {0};
    int flag = 0;
    for(int i=0;i<5;i++){
        for(int j=1;j<=6;j++){
            if(c->dices[i] == j){
                cnt[j-1] += 1;
            }
        }
    }
    for(int i=0;i<6;i++){
        if(cnt[i] == 4){
            flag = 1;
        }
    }
    if(flag){
        for(int i=0;i<5;i++){
            s += c->dices[i];
        }
    }
    else{
        s = 0;
    }
}
```

같은 수가 나온 주사위가 4개 이상 나오면 모든 주사위의 합을 기입한다.

```
if(num == 9){//full house
    int cnt[5] = {6};
    int flag1 = 0;
    int flag2 = 0;
    for(int i=0;i<5;i++){
        for(int j=1;j<=6;j++){
            if(c->dices[i] == j){
                cnt[j-1] += 1;
            }
        }
    }
    for(int i=0;i<5;i++){
        if(cnt[i] == 3){
            flag1 = 1;
        }
        if(cnt[i] == 2){
            flag2 = 1;
        }
    }
    if(flag1 == 1 && flag2 == 1){
        s = 25;
    }
    else{
        s = 0;
    }
}
```

같은 숫자의 주사위가 각각 3개와 2개이면 25점을 고정적으로 획득한다.

```
if(num == 10){//small straight
    int cnt[6] = {0};
    for(int i=0;i<5;i++){
        for(int j=1;j<=6;j++){
            if(c->dices[i] == j){
                cnt[j-1] += 1;
            }
        }
    }
    if(cnt[0]==1 && cnt[1]==1 &&cnt[2]==1 &&cnt[3]==1){//1234
        s = 30;
    }
    if(cnt[1]==1 && cnt[2]==1 &&cnt[3]==1 &&cnt[4]==1){//2345
        s = 30;
    }
    if(cnt[2]==1 && cnt[3]==1 &&cnt[4]==1 &&cnt[5]==1){//,3456
        s = 30;
    }
}
```

4개가 연속되는 숫자면 30점으로 고정적으로 획득한다.

```
 if(num == 11){//large straight
     int cnt[6] = {0};
     for(int i=0;i<5;i++){
         for(int j=1;j<=6;j++){
             if(c->dices[i] == j){
                 cnt[j-1] += 1;
             }
         }
     }
     if(cnt[0]==1 && cnt[1]==1 &&cnt[2]==1 &&cnt[3]==1 && cnt[4] ==1){//12345
         s = 40;
     }
     if(cnt[1]==1 && cnt[2]==1 &&cnt[3]==1 &&cnt[4]==1 && cnt[5] ==1){//23456
         s = 40;
     }
 }
```

5개가 연속되는 숫자면 40점을 고정적으로 획득한다.

```
if(num == 12){//chance
    for(int i=0;i<5;i++){
        s += c->dices[i];
    }
}
```

찬스로 그냥 모든 주사위의 합을 기입한다.

```
    if(num == 13){//yahtzee
        int cnt[5] = {0};
        int flag = 0;
        for(int i=0;i<5;i++){
            for(int j=1;j<=6;j++){
                if(c->dices[i] == j){
                    cnt[j-1] += 1;
                }
            }
        }
        for(int i=0;i<5;i++){
            if(cnt[i] == 5){
                flag = 1;
            }
        }
        if(flag){
            s = 50;
        }
        else{
            s = 0;
        }
    }
    return s;
}
```

같은 수의 주사위가 5개 나오면 고정적으로 50점을 획득한다.

```
void calcWin(struct socketInfo* c){
    printf("calcWin func\n");
    showScore(c);
    showScore(c->vs);
    if(c->scores[14] > c->vs->scores[14]){
        clientSend(c, "[!] you win!\n");
        clientSend(c->vs, "[!] you lose\n");
        c->infos.win += 1;
        c->vs->infos.lose +=1;
    }
    if(c->scores[14] < c->vs->scores[14]){
        clientSend(c->vs, "[!] you win!\n");
        clientSend(c, "[!] you lose\n");
        c->infos.lose += 1;
        c->vs->infos.win += 1;
    }
    if(c->scores[14] == c->vs->scores[14]){
        clientSend(c->vs, "[!] Draw!\n");
        clientSend(c, "[!] Draw\n");
        c->infos.draw += 1;
        c->vs->infos.draw += 1;
    }
    printf("calcwin end\n");
    printf("c : %s %d %d %d\n",c->infos.name, c->infos.win, c->infos.draw, c->infos.lose);
    printf("v : %s %d %d %d\n",c->vs->infos.name, c->vs->infos.win, c->vs->infos.draw, c->vs->infos.lose);
    updateUserInfo(c);
    c->vs->vs = NULL;
    c->vs = NULL;
    play.id = -1;
}
```

클라이언트 구조체의 scores라는 배열 14번째에는 모든 점수의 합이 저장되기 때문에 이 합을 기준으로 누가 이겼는지 판단하고 각각 전적을 기록한 뒤 updateUserInfo 함수를 통해 전적을 저장한다.
이후 player1 의 상대방의 상대(player1)을 NULL로 바꾸고 player1의 상대를 NULL로 설정한 뒤 play의 id를 -1로 바꾸는 방식으로 게임의 방을 해제한다.

```
1  void updateUserInfo(struct socketInfo* user){
2      pthread_mutex_lock(&fileMutex);
3      FILE* fp = fopen("users.txt","r+");
4      char update[2000] = "";
5      char *tokens[30];
6      int nr=0;
7      char line[100];
8      if(fp != NULL){
9          while (fgets(line, sizeof(line), fp) != NULL ) {
10             parse_command(line,&nr,tokens);
11             if(strcmp(tokens[0],user->infos.name) == 0){
12                 sprintf(update, "%s%s %d %d %d\n",update,user->infos.name,user->infos.win,user->infos.draw,user->infos.lose);
13             }
14             else if(strcmp(tokens[0],user->vs->infos.name) == 0){
15                 sprintf(update, "%s%s %d %d %d\n",update,user->vs->infos.name,user->vs->infos.win,user->vs->infos.draw,user->vs->infos.lose);
16             }
17
18             else{
19                 sprintf(update, "%s%s %s %s %s\n",update,tokens[0],tokens[1],tokens[2],tokens[3]);
20             }
21         }
22         fclose(fp);
23     }
24     printf("update : %s",update);
25     fp = fopen("users.txt", "w+");
26     fputs(update, fp);
27     fclose(fp);
28     pthread_mutex_unlock(&fileMutex);
29 }
```

updateUserInfo 함수를 통해 전적을 기록하는데 file기록 시 read하는 사람과 충돌이 날 것을 예상하여 lock을 걸어주었다. 각각의 행을 찾아서 자기와 상대가가 속한 행을 찾으면 변경된 전적을 update라는 배열에 저장하고 그 외라면 그대로 update에 저장한다.
이후 파일을 clsoe하고 write mode로 open한 다음에 다시 파일을 쓴 뒤 변경된 내용을 파일에 쓰고 저장한 뒤 unlock한다.

========
클라이언트 부분

```
char playerID[16] = "";
char userid[16]="";
int rate[3] = {0,};
int connEnd = 0;
```

클라이언트 정보를 저장하는 부분, 각각 userID, 승무패를 기록하는 배열, 커넥션이 끊어졌는지 확인하는 정수

```
void showBanner(){
    char banner[600]="";
    FILE* fp = fopen("banner.txt","r");
    if(fp != NULL){
        fread(banner, 1, 600, fp);
        printf("%s\n",banner);
        fclose(fp);
    }
    else{
        printf("cannot banner file..\n");
    }
}
```

야찌 게임 배너를 출력하는 함수, 배너가 없다면 배너가 없다고 출력한다.

```
void showMenu(){
    printf("\n==================\n");
    printf("$ join\t -> join the game..\n");
    printf("$ help\t -> show help\n");
    printf("$ menu\t -> show menu\n");
    printf("$ set [number]\t->write scores..\n");
    printf("$ [dicenum] [dicenum] [dicenum] [dicenum] [dicenum] -> re roll dice\n 0 != reroll\n");
}
```

메뉴를 출력하는 함수

```c
void *recv_server(void* p){
    int* c = (int*)p;
    int client = *c;
    char bye[8] = "byebye\n";
    while(1){
        char recv[1025]="";
        read(client,recv,1024);
        recv[1024] = '\0';

        if(strcmp(recv,bye) == 0){
            printf("socket end@@ \n");
            close(client);
            break;
        }
        if(strcmp(recv,"[!] your turn\n") == 0){
            printf("my turnturnturnturn\n");
        }
        printf("\n=============\n");
        printf("\nserver-> \n%s\n",recv);
        printf("=============@\n");
        printf("%s:$ ",playerID);
    }
    printf("thread end\n");
    connEnd = 1;
}
```

서버에서 보낸 메시지를 출력하는 함수 스레드로 실행이 된다.

```c
int main(int argc,char **argv){
    showBanner();
    struct sockaddr_in server;
    memset(&server,0,sizeof(server));
    printf("[@] %s start!\n",argv[0]);
    int client = socket(PF_INET, SOCK_STREAM, 0);
    if(client == -1){
        fprintf(stderr,"client socket err\n");
        exit(1);
    }
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(7777);
    if(connect(client,(struct sockaddr*)&server, sizeof(server))== -1){
        fprintf(stderr, "conn err\n");
    }
    else{
        char buf[1025];
        char* userinfos[16];
        printf("id : ");
        scanf("%s",userid);
        write(client,userid,strlen(userid));
        read(client,buf,1024);
        printf("recv\n");
        int nr =0;
        parse_command(buf,&nr,userinfos);
        strcpy(playerID,userinfos[0]);
        rate[0] = atoi(userinfos[1]);
        rate[1] = atoi(userinfos[2]);
        rate[2] = atoi(userinfos[3]);
        //playerID = atoi(idtok);
        printf("playerID : %s\n",playerID);
        sleep(1);
        pthread_t recv_t;
        pthread_create(&recv_t, NULL, recv_server, (void*)&client);
        while(1){
            char input[1000];
            sleep(1);
            fgets(input,999,stdin);
            fflush(stdin);
            fflush(stdout);
            input[999] = '\0';
            if(connEnd){
                break;
            }
            if(strcmp(input,"help\n") == 0){
                showHelp();
            }
            else if(strcmp(input,"menu\n") == 0){
                showMenu();
            }
            else if(!exitCheck(input, client)){
                write(client,input,strlen(input));
            }
            else{
                break;
            }
        }
    }
    return 0;
}
```

서버의 정보를 기입하는 소켓을 만들고 이를 이용해 connect 함수를 실행한다.
서버 주소는 로컬에서 테스트 해서 루프백 IP를 전달하고 7777포트로 접속하도록 했다.
서버에 연결이 되면 맨 처음으로 로그인 할 id를 입력하고 이를 서버에 전달한다. 서버에 전달하면 로그인

된 아이디와 전적을 받는데 이를 모두 클라이언트에 저장한다.
쉘처럼 출력해주기 위해서 입력받기 전 1초를 기다린다.
이후 서버의 통신을 recv하는 스레드를 만들고 쉘처럼 로그인된 아이디와 커맨드를 입력할 수 있도록 한다. 각각 help, menu 등의 로컬 기능을 수행하거나 서버에게 명령어를 보낸다.

```
1  int parse_command(char *command, int *nr_tokens, char *tokens[]){
2      const char *delimiters = " \t\r\f\r\n\v";
3      char *curr;
4      *nr_tokens = 0;
5      while ((curr = strtok(command, delimiters))) {
6          *tokens++ = strdup(curr);
7          (*nr_tokens)++;
8          command = NULL;
9      }
10     *tokens = NULL;
11     return (*nr_tokens > 0);
12 }
13
```

버퍼에서 스페이스바와 엔터 등 escape하는 문자열들을 기준으로 tokenize하는 함수이다. 소프트웨어학과의 김상훈 교수님의 코드를 사용했다.
7. 동작 시연

```
sori@noe:~/sss/pj2$ ./server
./server PORT
```

포트 미 기입시 화면

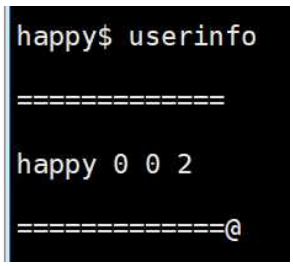
```
sori@noe:~/sss/pj2$ ./server 7777
initialized...
server on 7777
$
```

포트 기입 시 정상 실행 화면

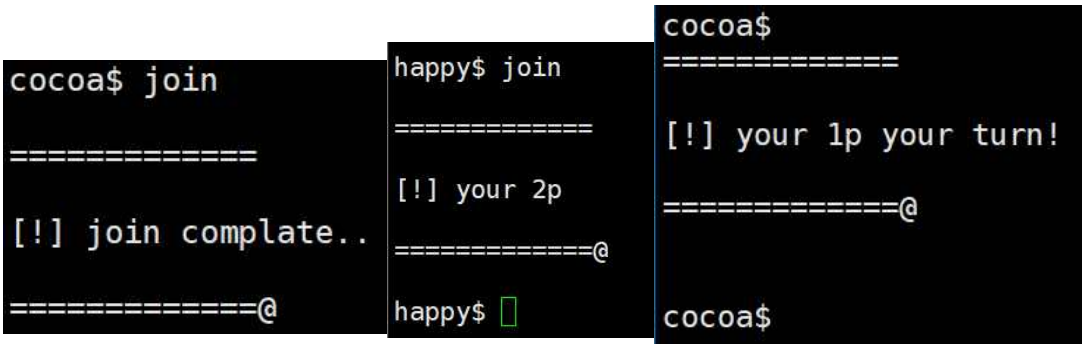
```
sori@noe:~/sss/pj2$ ./client
'##:::'##:::'###::::'##::::'##:'########:'########:'########:'########:
. ##:'##::::'## ##::: ##:::: ##:... ##...:::... ##::: ##.....:: ##.....::
:. ####::::'##:. ##:: ##:::: ##::::: ##:::::::: ##::: ##::::::: ##::::::::
::. ##:::::'##:::. ##: #########::::: ##:::::::: ##::::: ######::: ######:::
::: ##:::::: #########: ##.... ##::::: ##:::::::: ##::::: ##...:::: ##...::::
::: ##:::::: ##.... ##: ##:::: ##::::: ##:::::::: ##::::: ##::::::: ##:::::::
::: ##:::::: ##:::: ##: ##:::: ##::::: ##:::::::: ##::::: ########: ########:
:::..:::::::..:::::..::..:::::..::::::..:::::::::..::::::........::........::

[@] ./client start!
id : sori
recv
playerID : sori
buf :


=============

server->
[%d] no match..

=============@

sori$
```

클라이언트 실행 화면 (정상적 연결 시)

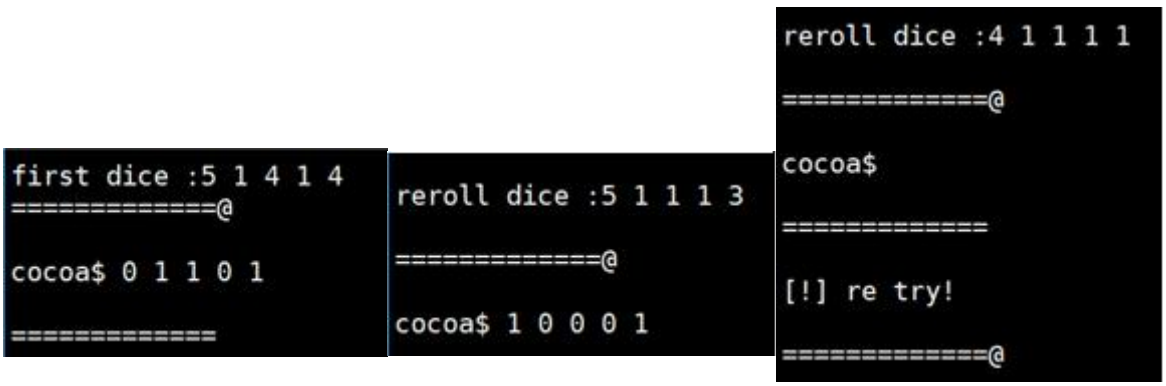
서버에 연결되지 않았을 때 화면



로그인 후 전적확인



방 만들기 및 방 입장



플레이어 1의 첫 번째 턴 주사위가 잘 굴러가는 것을 확인할 수 있다.

```
cocoa$ set 1

=============

[!] turn end

=============@

cocoa$ show

=============

player           num      |       cocoa    |       happy
one              1        |       4        |       -1
two              2        |       -1       |       -1
three            3        |       -1       |       -1
four             4        |       -1       |       -1
five             5        |       -1       |       -1
six              6        |       -1       |       -1
three_of_a_kind  7        |       -1       |       -1
four_of_a_kind   8        |       -1       |       -1
full_house       9        |       -1       |       -1
small_straight   10       |       -1       |       -1
large_straight   11       |       -1       |       -1
chance           12       |       -1       |       -1
yahtzee!         13       |       -1       |       -1
bonus                     |       0        |       0
sum                       |       4        |       0

=============@
```

점수 등록 이후 점수판 확인, 이전에 1이 4개 였기 때문에 4점이 되었다.

```
first dice :1 4 5 1 3
=============@

happy$ 0 1 1 0 1

=============
```

```
reroll dice :1 1 5 1 3

=============@

happy$ end

=============
```

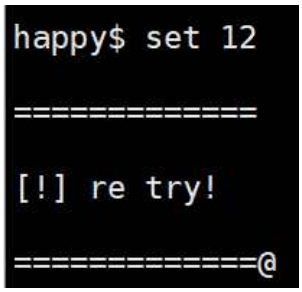
```
dice :1 1 5 1 3

=============@

happy$ set 1

=============

[!] turn end
```

```
happy$ show

============

player           num      |       happy    |       cocoa
one              1        |       3        |       4
two              2        |       -1       |       -1
three            3        |       -1       |       -1
four             4        |       -1       |       -1
five             5        |       -1       |       -1
six              6        |       -1       |       -1
three_of_a_kind  7        |       -1       |       -1
four_of_a_kind   8        |       -1       |       -1
full_house       9        |       -1       |       -1
small_straight   10       |       -1       |       -1
large_straight   11       |       -1       |       -1
chance           12       |       -1       |       -1
yahtzee!         13       |       -1       |       -1
bonus                     |       0        |       0
sum                       |       3        |       4

============@
```

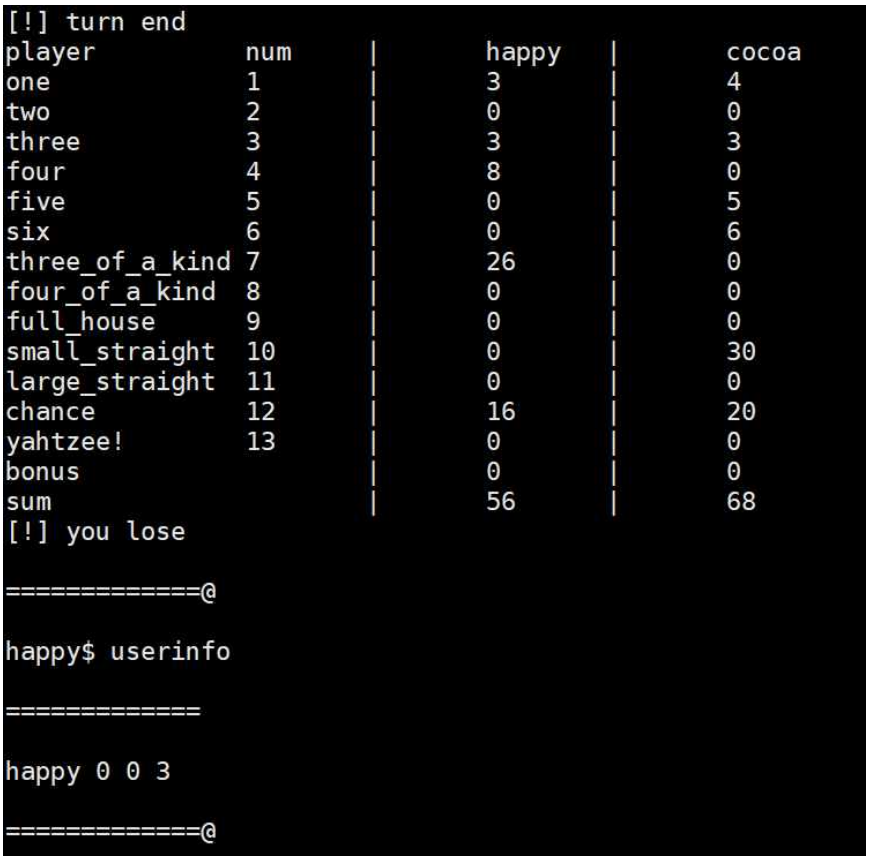
```
player           num      |       cocoa    |       happy
one              1        |       4        |       3
two              2        |       -1       |       -1
three            3        |       -1       |       -1
four             4        |       -1       |       -1
five             5        |       -1       |       -1
six              6        |       -1       |       -1
three_of_a_kind  7        |       -1       |       -1
four_of_a_kind   8        |       -1       |       -1
full_house       9        |       -1       |       -1
small_straight   10       |       -1       |       -1
large_straight   11       |       -1       |       -1
chance           12       |       -1       |       -1
yahtzee!         13       |       -1       |       -1
bonus                     |       0        |       0
sum                       |       4        |       3
```

턴 종료 이후 show 명령어를 통한 클라이언트 동기화

플레이어2의 턴, 주사위 굴린 이후 end로 바로 점수 기입단계로 진입 점수 기입이 되었다.



이미 기입한 턴에 다시 기입할 때의 화면



게임종료 후 전적 갱신 화면

8. 느낀점

실시간 게임을 만들기가 굉장히 어려웠다. 여러 클라이언트의 커넥션을 동시에 관리하는것도 굉장히 힘들었다. 스레드가 아닌 select를 이용해서 socket fd를 관리할 수 있다는게 신기했고 이를 이용해서 fd를 listked list로 만들어 플레이어를 올바른 로직으로 조작할 수 있었다.

또한 방을 만들고, 턴을 올바르게 계산하고 게임의 규칙을 코드로 녹여내는 것이 굉장히 힘들었다. mutex lock을 사용하면서 lock에 대한 이해와 적절하게 사용하는 방법을 학습할 수 있었다. 코드를 짜면서 서버와 1:1로 통신하는 프로그램을 짜는 것 보다, 서버를 경유해 다른 클라이언트와 통신하도록 계산하는게 너무 어려워서 버그가 굉장히 많고 플레이어의 행동을 제한하는 것이 너무 힘들었다.

QA가 조금 미흡한 것 같지만 그래도 생각한 기능들이 다 동작했기에 프로젝트를 완료했다고 생각한다.

지금은 mutex가 하나밖에 없어 방을 하나밖에 만들 수 없지만 추후에 linkedlist를 이용해 방을 여러개 생성 할 수 있도록 만들고 싶다.

9. 참고자료

리눅스 시스템 콜 목록

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md

parse_command함수가 속한 parser.c 파일
https://git.ajou.ac.kr/sslab/os-pa1/-/blob/main/parser.c
리눅스 토발즈가 개발한 list 커널 API
https://github.com/torvalds/linux/blob/master/include/linux/list.h
야찌 룰
https://namu.wiki/w/야찌