보드게임 만들기

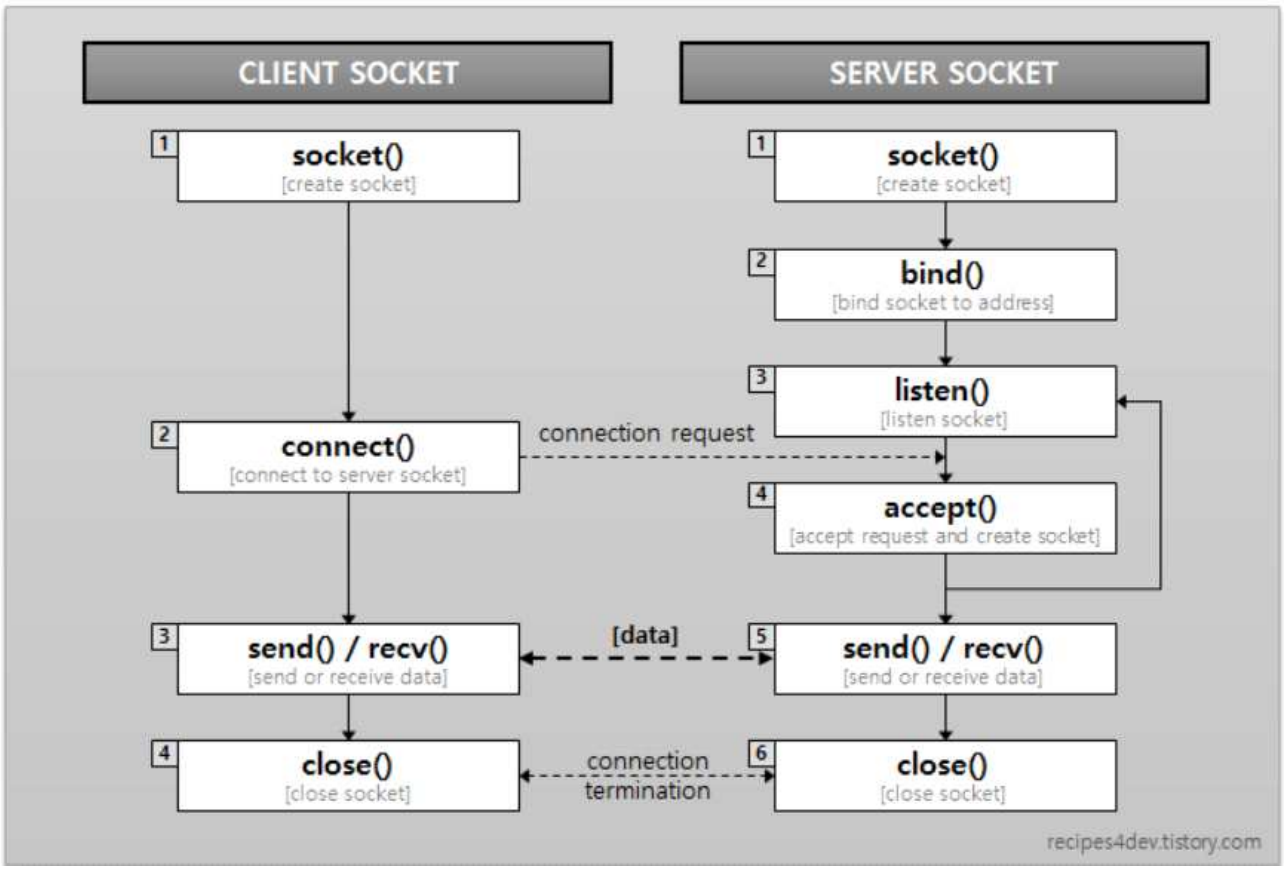
2019110514 김병재

1. 개요

Socket, IPC, thread, mutex를 이용한 보드게임 만들기 과제입니다. 그러나 IPC, thread를 구현하는 과제에서 에러가 발생하였고 해결하지 못하였습니다. 요구 조건을 만족시키고자 최대한 노력하였고, Socket통신과 mutex를 이용한 가위바위보 프로그램을 제작하였습니다.

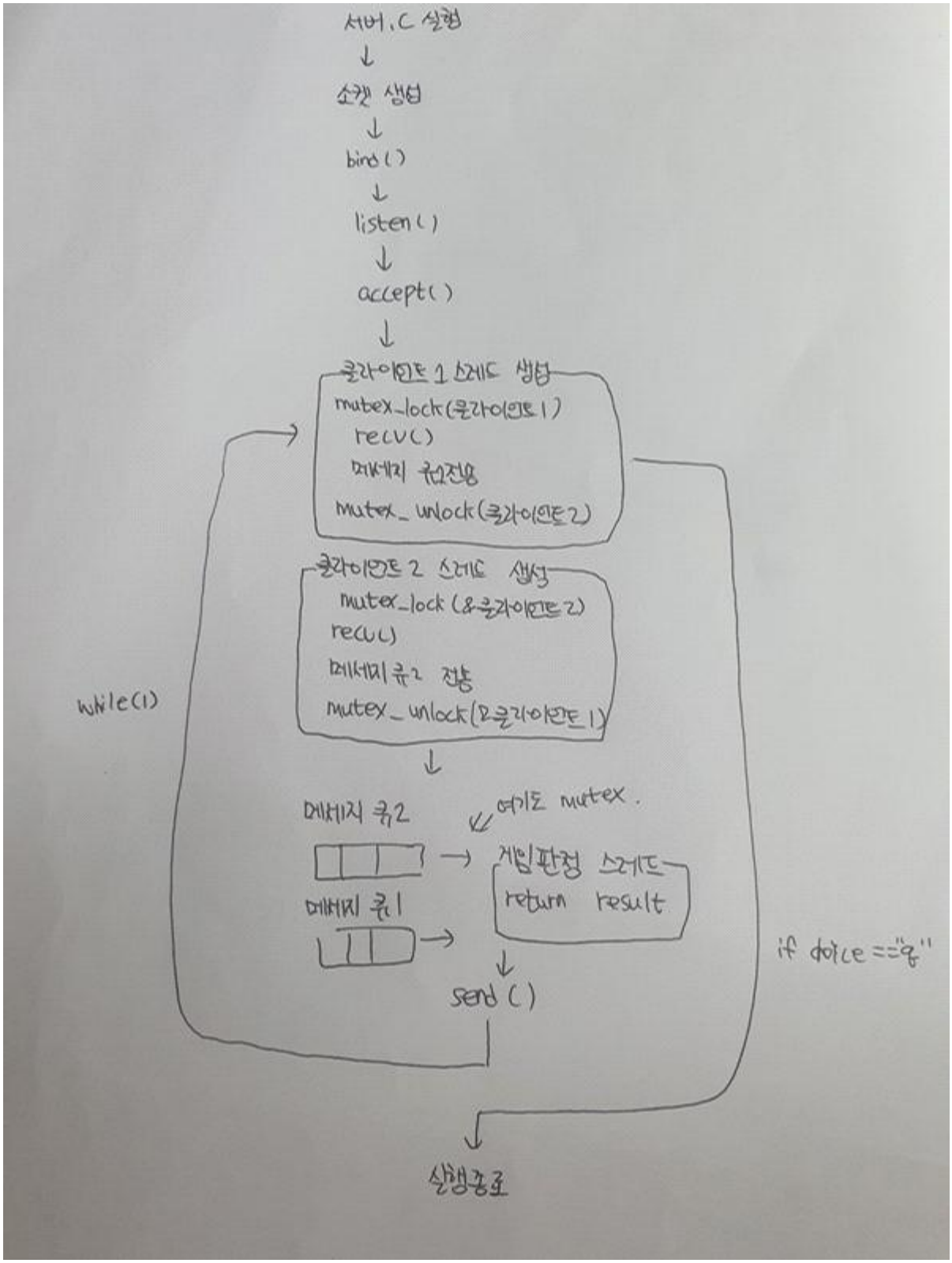
2. 프로그램의 구조

1) 순서도



위 사진은 일반적인 서버, 클라이언트 프로그램의 순서도이다. 나는 과제의 조건을 충족하기 위해 두 개의 스레드

를 이용하여 각 클라이언트와 메시지를 주고받고, 소켓 스레드와 게임 스레드가 입력 값고 결과값을 주고 받도록 코드를 짤 계획이었다.



클라이언트로부터 입력을 받고, 그 입력받은 값을 메시지 큐로 전달하는 스레드를 클라이언트 별로 만든 다음, 게임

의 결과를 판정하는 스레드가 메시지 큐로부터 입력받아서 출력된 값을 클라이언트 서버에 send하도록 프로그램을 구상하였다

.

critical section은 클라이언트로부터 입력받는 부분이다. 만약 하나의 메시지 큐를 공유한다는 가정하에, 한 클라이언트가 계속하여 입력한다면 결과가 이상하게 출력 되기 때문이다. (여기서 각 클라이언트마다 입력받은 값을 저장하는 메시지큐를 따로 만들려고 하였다.) 따라서 하나의 클라이언트 스레드를 실행하면 mutex_lock이 걸리고, 다른 클라이언트를 실행 완료하면 mutex_unlock이 되도록 프로그램을 구상하였다. 그러나 스레드를 이용하여 프로그램을 짜는데 실패하였기에 strlen 함수를 사용하여 choice1과 choice2가 모두 입력되어야지만 코드가 동작하도록 설계하였다.

여기서 IPC는 메시지 큐를 이용하여 스레드간 정보를 주고 받는 과정이다. 근데 IPC는 Inter Process Communication 이므로 프로세스 간 메시지를 주고받는 방법을 의미하는데, 나는 스레드간 정보를 주고 받기위해 message queue 방법을 이용하여 하였다. 그래서 아마 실패하지 않았나하는 생각이 든다.

2) 프로그램 설명

다음은 server.c 파일에 대한 설명이다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <sys/types.h>

#define PORT 12345


pthread_mutex_t player1, player2;
int client_socket1, client_socket2; // 클라이언트 소켓 변수
char choice1[100]; // 클라이언트1 입력값
char choice2[100]; //클라이언트 2 입력값
char* intro1 = "\n\n당신은 player1입니다.";
char* intro2 = "\n\n당신은 player2입니다.";

// 클라이언트1에게 문자열을 보내는 함수
void msgsend1(char* msg) {
    if (send(client_socket1, msg, strlen(msg), 0) == -1) {
        perror("데이터 전송 실패");
        exit(1);
    }
}

// 클라이언트2에게 문자열을 보내는 함수

void msgsend2(char* msg) {
    if (send(client_socket2, msg, strlen(msg), 0) == -1) {
        perror("데이터 전송 실패");
        exit(1);
    }
}
```

클라이언트에게 데이터를 전송할때마다 if문을 사용하는 것이 번거로워서 msgsend 함수를 생성하였다. msgsend의 인자로 전송할 데이터(문자열)을 입력하면 된다.

```
// 가위바위보 결과를 계산하는 함수
char* rsp(char* choice1, char* choice2) {
    if ( (strcmp(choice1,"q") == 0) || (strcmp(choice2,"q") ==0) ) {
       printf("게임이 종료되었습니다.\n");
       exit(0);
    } else if (strcmp(choice1, choice2) == 0) {
        return "무승부";
    } else if (
        (strcmp(choice1, "가위") == 0 && strcmp(choice2, "보") == 0)
        || (strcmp(choice1, "바위") == 0 && strcmp(choice2, "가위") == 0)
        || (strcmp(choice1, "보") == 0 && strcmp(choice2, "바위") == 0)
    ) {
        return "Player 1 승리";
    } else {
        return "Player 2 승리";
    }
}
```

rsp 함수는 각 클라이언트로부터 choice값을 받아서 결과를 계산하는 함수이다. 이때 q를 입력받으면 프로그램이 종료되도록 설계하였다.

```
int main() {
    int server_socket;
    struct sockaddr_in server_address, client_address;
    socklen_t client_address_length;

    // 소켓 생성
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("소켓 생성 실패");
        exit(1);
    }

    // 주소 설정
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(PORT);

    // 바인딩
    if (bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        perror("바인딩 실패");
        exit(1);
    }

    printf("가위바위보 게임 서버입니다.\n");
    printf("클라이언트를 기다리는 중...\n");

    // 클라이언트 대기열 설정
    if (listen(server_socket, 2) == -1) {
        perror("대기열 설정 실패");
        exit(1);
    }
```

다음은 메인함수이다. 메인함수에서는 소켓을 생성하며, 주소를 설정하고, 포트와 소켓을 바인딩한다.

그리고 listen 함수를 이용하여 대기열을 설정하였다. 이때 backlog를 2로 두었으며 포트는 그냥 12345로 설정하였다.

```
// 첫 번째 클라이언트 접속
client_address_length = sizeof(client_address);
client_socket1 = accept(server_socket, (struct sockaddr*)&client_address, &client_address_length);
if (client_socket1 == -1) {
    perror("첫 번째 클라이언트 접속 실패");
    exit(1);
}
printf("첫 번째 클라이언트가 접속하였습니다.\n");

// 두 번째 클라이언트 접속
client_socket2 = accept(server_socket, (struct sockaddr*)&client_address, &client_address_length);
if (client_socket2 == -1) {
    perror("두 번째 클라이언트 접속 실패");
    exit(1);
}
printf("두 번째 클라이언트가 접속하였습니다.\n");
```

accept 함수를 이용하여 클라이언트의 연결을 수락하고, 클라이언트와 연결된 소켓을 cleint_socket에 저장한다. 이때 client_address 구조체를 이용하였다. 만약 accept 함수가 성공하면 client_socket에 소켓 파일 디스크립터가 리턴되며, accept의 반환값이 -1이면 클라이언트의 접속이 실패한 것이기 때문에 프로그램이 종료되도록 설정하였다.

```
// 클라이언트1의 수신
memset(choice1, 0, sizeof(choice1));
pthread_mutex_lock(&player1);
if (recv(client_socket1, choice1, sizeof(choice1), 0) <= 0) {
    perror("수신 실패");
    exit(1);
}
printf("Player1 : %s\n", choice1);
pthread_mutex_unlock(&player1);


// 클라이언트2의 수신
memset(choice2, 0, sizeof(choice2));
pthread_mutex_lock(&player1);
if (recv(client_socket2, choice2, sizeof(choice2), 0) <= 0) {
   perror("수신 실패");
   exit(1);
}
printf("Player2 : %s\n", choice2);
pthread_mutex_unlock(&player1);
```

클라이언트로부터 데이터를 입력받는 코드이다. memset으로 choice변수를 초기화 시킨 후, recv 함수를 이용하여 클라이언트가 입력한 값을 choice에 저장하였다. 그리고 한 클라이언트가 데이터를 여러번 입력하는 것을 막기 위

해 mutex를 사용하였다. (Critical Section)

```
// 선택이 모두 완료되면
if (strlen(choice1) > 0 && strlen(choice2) > 0) {
    // 게임 결과 계산
    char* result = rsp(choice1, choice2);
    printf("게임 결과: %s\n", result);

    // 결과 전송
    msgsend1(result);
    msgsend2(result);


    // 선택 초기화
    memset(choice1, 0, sizeof(choice1));
    memset(choice2, 0, sizeof(choice2));

    //메시지 전송
    msgsend1(intro1);
    msgsend2(intro2);
```

choice1과 choice2가 모두 입력되면 게임 결과를 계산한 후, msgsend함수를 사용하여 클라이언트에게 게임 결과를 전송하도록 만들었다. 그 다음 게임을 위하여 memset 변수를 사용하여 choice를 초기화 한 후, 클라이언트가 player1인지 player2인지 구분할 수 있도록 알려주는 메시지를 전송시켰다.

```
    // 소켓 종료
    close(client_socket1);
    close(client_socket2);
    close(server_socket);

    return 0;
}
```

그리고 소켓을 모두 종료한 후 프로그램은 끝이 난다. 지금 생각해보니 rsp함수에 exit를 넣지 말고 while문 안에 choice1과 choice2중 하나가 q이면 break; 되도록 코드를 만드는게 더 나았을 수도 있다는 생각이 든다.

이번에는 client.c 파일이다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1"
#define PORT 12345

int main() {
    int client_socket;
    struct sockaddr_in server_address;
    char result[100];
    char choice[100];
```

서버 IP는 127.0.0.1로 설정하였는데, 이는 내 컴퓨터에서 루프백 통신을 하기 위해 이렇게 설정하였다.
sockaddr_in 는 소켓 통신을 하기 위해 사용한 구조체이다.

```c
// 클라이언트 소켓 생성
client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (client_socket == -1) {
    perror("소켓 생성 실패");
    exit(1);
}

// 서버 주소 설정
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr(SERVER_IP);
server_address.sin_port = htons(PORT);

// 서버에 연결
if (connect(client_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
    perror("서버 연결 실패");
    exit(1);
}

printf("가위바위보 게임입니다.\n");
```

socket 함수를 사용하여 클라이언트 소켓을 생성하고, sockaddr 구조체의 멤버 변수들을 사용하여 연결할 서버의 주소를 설정한 다음 connect 함수를 이용하여 서버에 연결하였다. 서버와 연결이 성공하면 “가위바위보 게임입니다” 라는 문장이 출력되도록 하였다.

```c
    // 가위바위보 입력
    while (1) {
        printf("가위, 바위, 보 중 하나를 선택하세요(종료 : q) : ");
        fgets(choice, 100, stdin);
        choice[strcspn(choice, "\n")] = '\0'; // 개행 문자 제거


        // 전송
        if (send(client_socket, choice, strlen(choice), 0) == -1) {
            perror("데이터 전송 실패");
            exit(1);
        }

        // 입력이 q라면 프로그램 종료
        if (strcmp(choice, "q") == 0) {
            printf("게임을 종료합니다.\n");
            break;
        }

        // 결과 수신
        memset(result, 0, sizeof(result));
        if (recv(client_socket, result, sizeof(result), 0) <= 0) {
            perror("데이터 수신 실패");
            exit(1);
        }
        printf("게임 결과: %s\n", result);
    }

    // 소켓 종료
    close(client_socket);

    return 0;
}
```

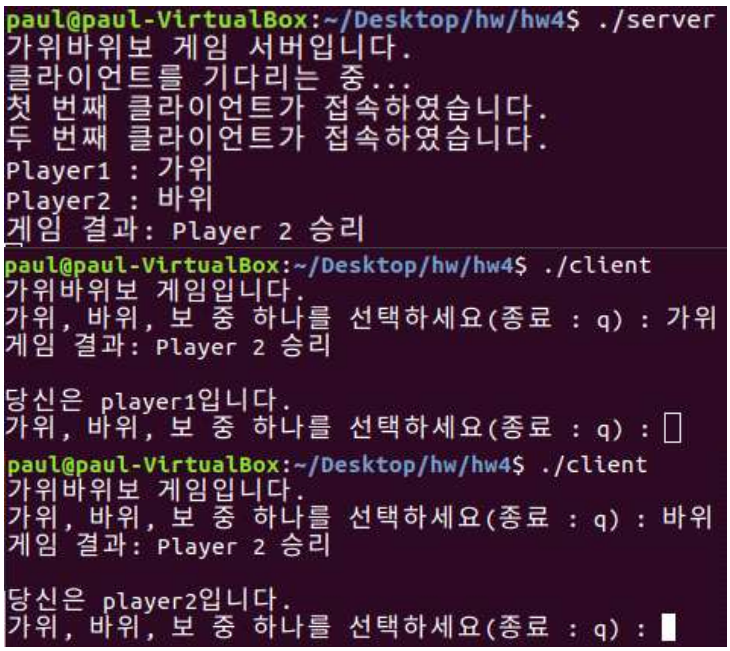
choice함수에 사용자가 입력한 값을 저장한 다음, send함수를 이용하여 choice의 값을 서버로 보냈다. 서버에서 클라이언트가 보낸 데이터를 바탕으로 게임 결과를 계산하면 result 변수에 데이터를 저장한 다음 게임 결과가 출력되도록 설정하였다. 이때 입력한 값이 q라면 while문을 탈출하고 소켓을 닫고 프로그램이 종료되도록 코드를 짰다.

[프로그램 실행 매뉴얼]

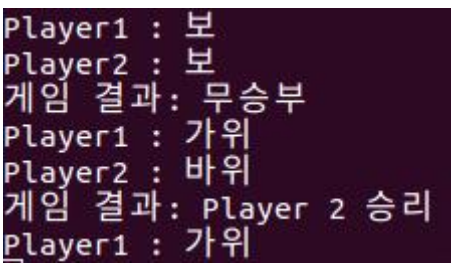
1. 터미널을 총 3개를 연다.
2. ./server 를 입력하여 서버프로그램 실행.
3. 나머지 두 터미널에 /client를 각각 입력 후 클라이언트 프로그램을 총 두 개 연다.
4. 입력할때는 정확히 “가위”, “바위”, “보”, “q”중 하나만 입력한다.
5. 한 플레이어가 여러번 입력한 경우, 다음 라운드에 그 입력된 값이 반영된다.
6. 플레이어2가 먼저 입력한 경우, 플레이어1이 입력되어야 서버 프로그램이 반응(출력)한다.

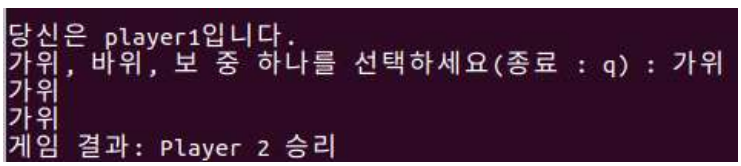
3. 결과

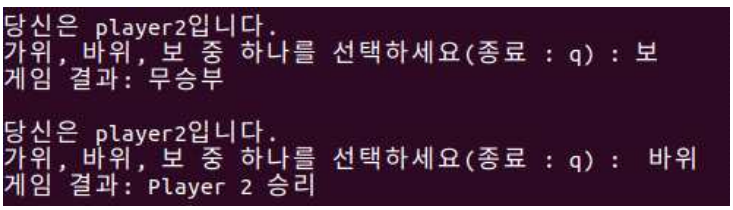
(1) 실행 결과



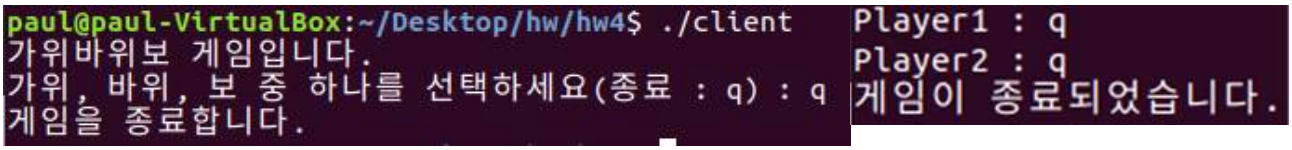
서버 프로그램과 클라이언트 간 통신이 잘 되는 것을 확인할 수 있다.







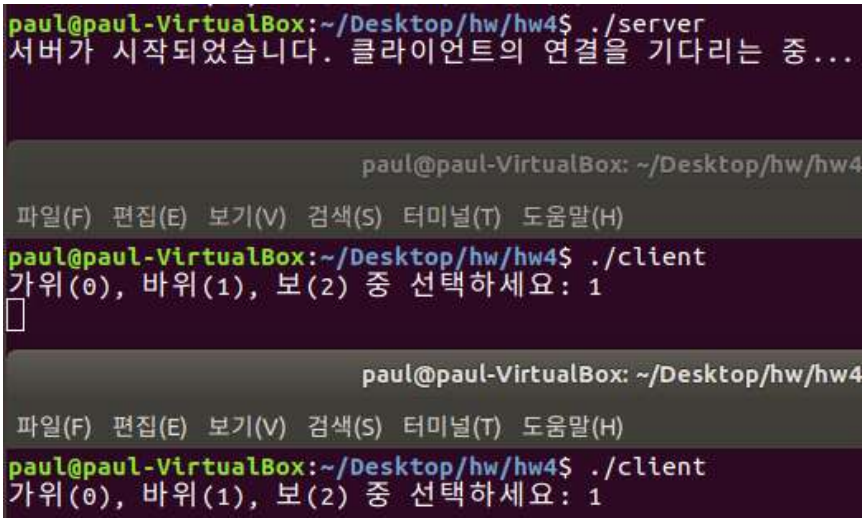
무승부일 경우에도 잘 동작하며, player1이 입력을 여러번 할 경우, player2가 입력을 해야지만 결과가 계산되어 출력된다. 그리고 이전에 입력된 내용은 저장되는 것을 확인 할 수 있다.



q를 입력한 경우 프로그램이 정상적으로 종료된다.

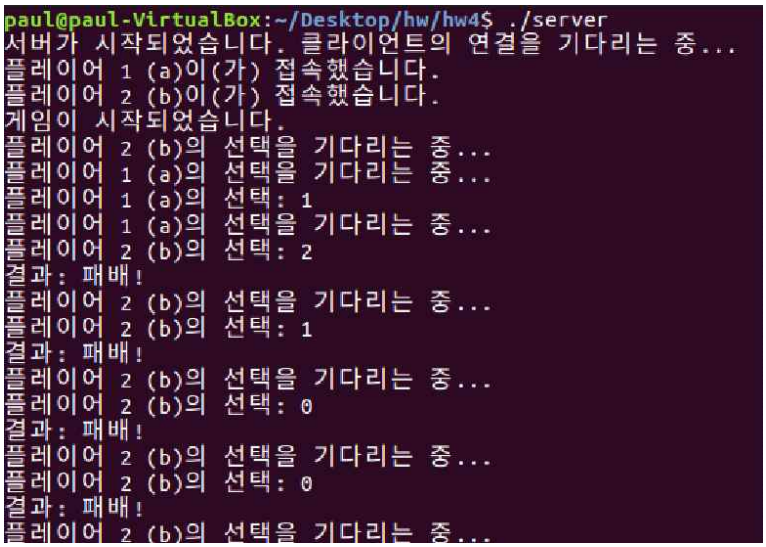
(2) 실패 사례

(1) 에러1



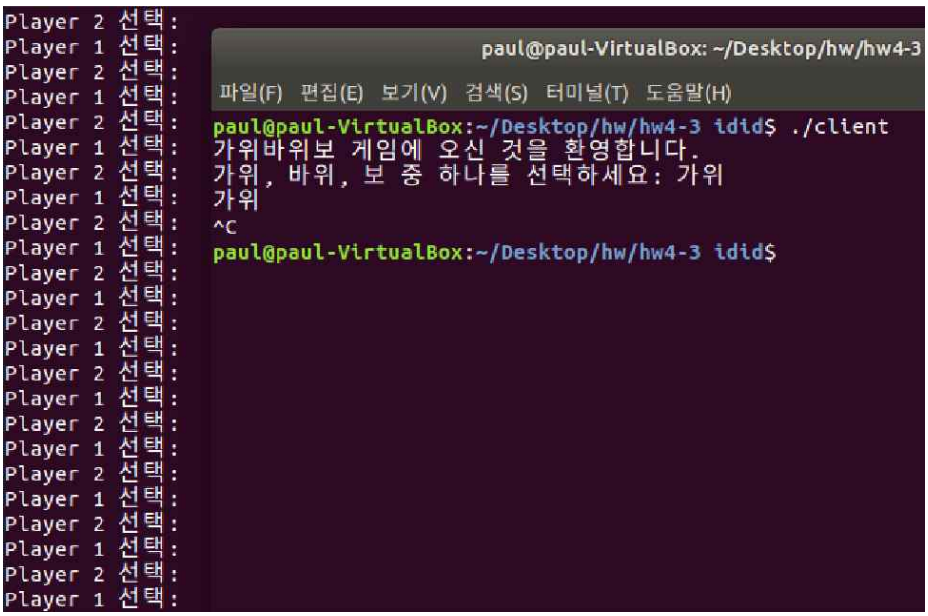
클라이언트와 서버가 연결은 되었으나 데이터를 전송해도 아무런 반응이 없다.

(2) 에러2



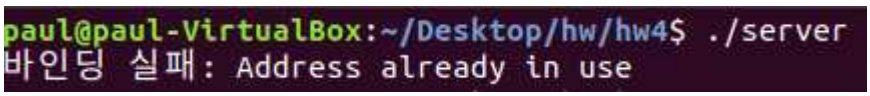
플레이어1의 입력을 받지 못한다. 플레이어 2의 입력만 받아들인다.

(3) 에러3

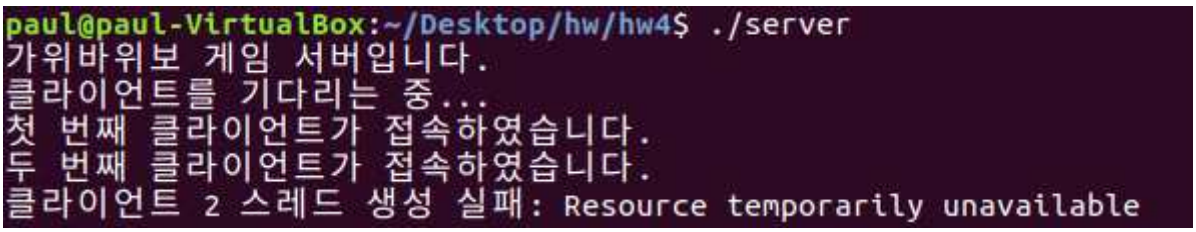


자꾸 플레이어 선택이 뜬다. 아마 while문 안에서 문제가 발생한 듯 하다.

(4) 에러4



주소가 이미 사용중이라서 바인딩에 실패하였다고 뜬다.



이외에도 스레드를 만들어서 할려고 했는데 자원 관리 문제 때문에 실패한 듯 하다.

4. 에러수정 및 고찰

1) 에러 해결 과정

(1) 에러1
이전에 메시지 큐를 사용하여 인수를 전달했는데 그 과정에서 오류가 발생한 것 같다. 지금처럼 전역변수를 사용하니까 문제가 해결되었다. 그러나 과제에서 요구하는 조건은 충족하지 못했다.

(2) 에러2,3
mutex관련 문제인거 같다. 그전에 오류가 발생하였을 때 코드를 따로 찍어 놓지 못하였다. mutex인자를 여러개 꼬아놓았던걸 없앴다. 그래서 그냥 지금처럼 strlen을 사용하여 둘다 입력된 경우에 rsp 함수가 실행되도록 코드를 수정하였다.

(3) 에러4
에러4는 프로그램을 실행한 후, 수정하였다가 재실행하니까 발생하였다. 아마 기존 프로그램이 제대로 종료되지 않아서 이러한 오류가 발생된 것 같다. 이 경우 netstat -lntp를 입력한 후, 해당 어드레스를 사용중인 프로그램을 죽이면 된다고 하는데

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
```

pid가 나오지 않아서 kill을 사용할 수가 없었다. 그냥 기다리니까 알아서 종료되고 문제가 해결되었다.

마지막 스레드 에러역시 제대로 해결하지 못하였다. 이전에 했던것처럼 for문을 써서 스레드를 돌려야하는건가, 그냥 다른 스레드 두 개를 사용해서 오류가 발생한건지 갈피가 안잡혔다.

2) 고찰 및 느낀점

이번 과제는 진짜 어떻게 시작해야할지 감이 안잡혔다. 교수님께서 수업시간에 소켓통신이 뭔지, ipc와 mutex의 개념은 설명해주셨으나 mutex를 제외하고는 코드를 어떤식으로 적용해야하는지 말씀해주시지 않아서 멘땅에 헤딩하면서 했다. 소켓, ipc, 뮤텍스, 스레드를 각각은 사용할 줄 알겠는데 이를 한 프로그램에 같이 적용하려고 하니까 오류도 엄청 발생하고 머리가 터지는 줄 알았다. 그래서 우선 프로그램을 동작하는걸 목표로 하였다. 그 다음 하나씩 살을 붙여나갈려고 했는데 메시지 큐, 스레드에서 오류가 많이 발생하고 원인 및 해결법도 알기 어려워서 과제에서 요구하는 조건을 다 만족시키지 못하였다. 공부가 더 필요하다고 느낀 과제였다.

5. 참고자료

https://aronglife.tistory.com/entry/NetworkTCPIP-소켓-프로그래밍2-서버-구현
https://kingko.tistory.com/entry/Message-Queue를-이용한-Thread-간-메시지-전달
https://reakwon.tistory.com/98 뮤텍스 적용 예시
https://velog.io/@woojin/address-already-in-use-port-포트-실행-중-에러가-생길-때
https://reakwon.tistory.com/117
https://doitnow-man.tistory.com/119
https://mintnlatte.tistory.com/280 -listen 함수
https://recipes4dev.tistory.com/153
https://blockdmask.tistory.com/441 등등
수업자료