REPORT

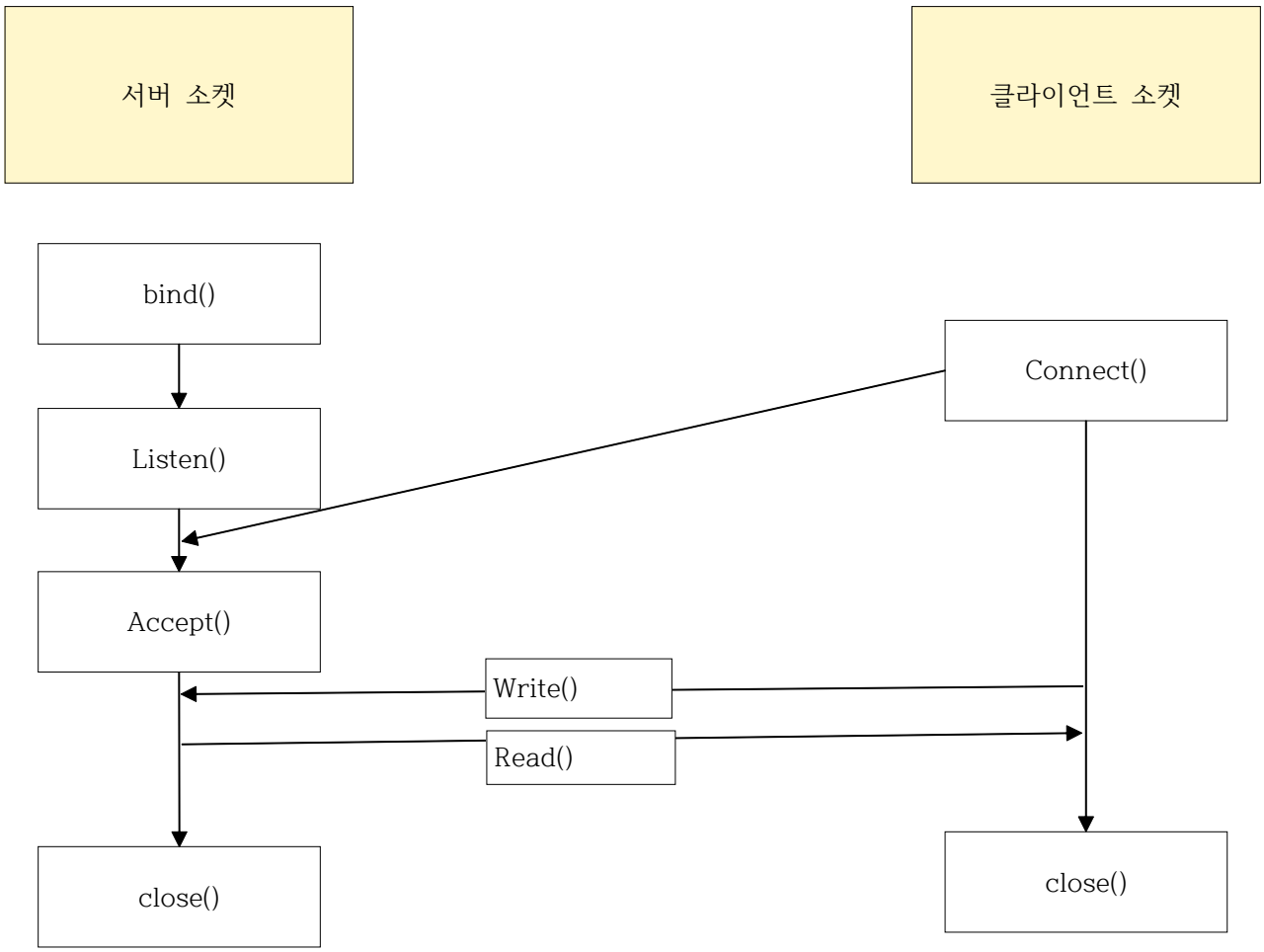
채팅서버 & 웹서버 구현

제출일	2020. 07. 03	전 공	소프트웨어과
과 목	사물인터넷	학 번	20170735
담당교수	전상준 교수님	이 름	이정아

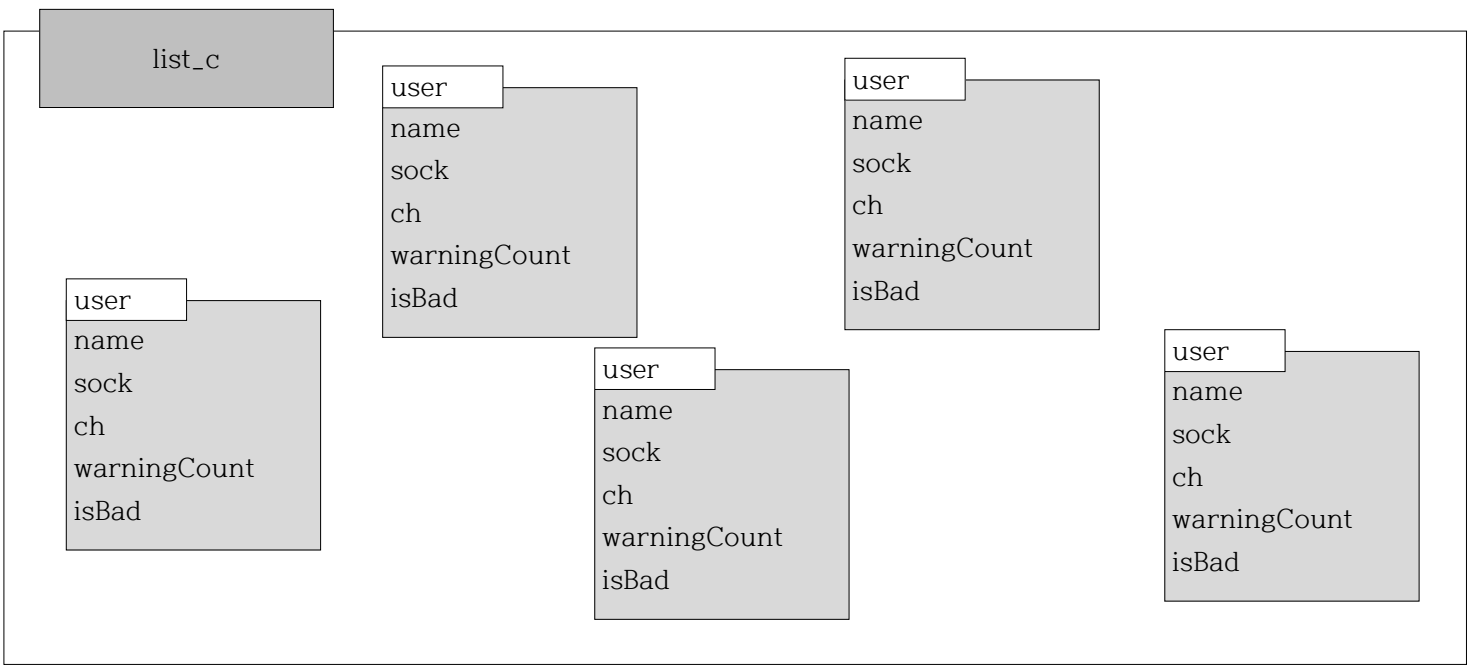
목 차

Ⅰ. 채팅 서버 구현

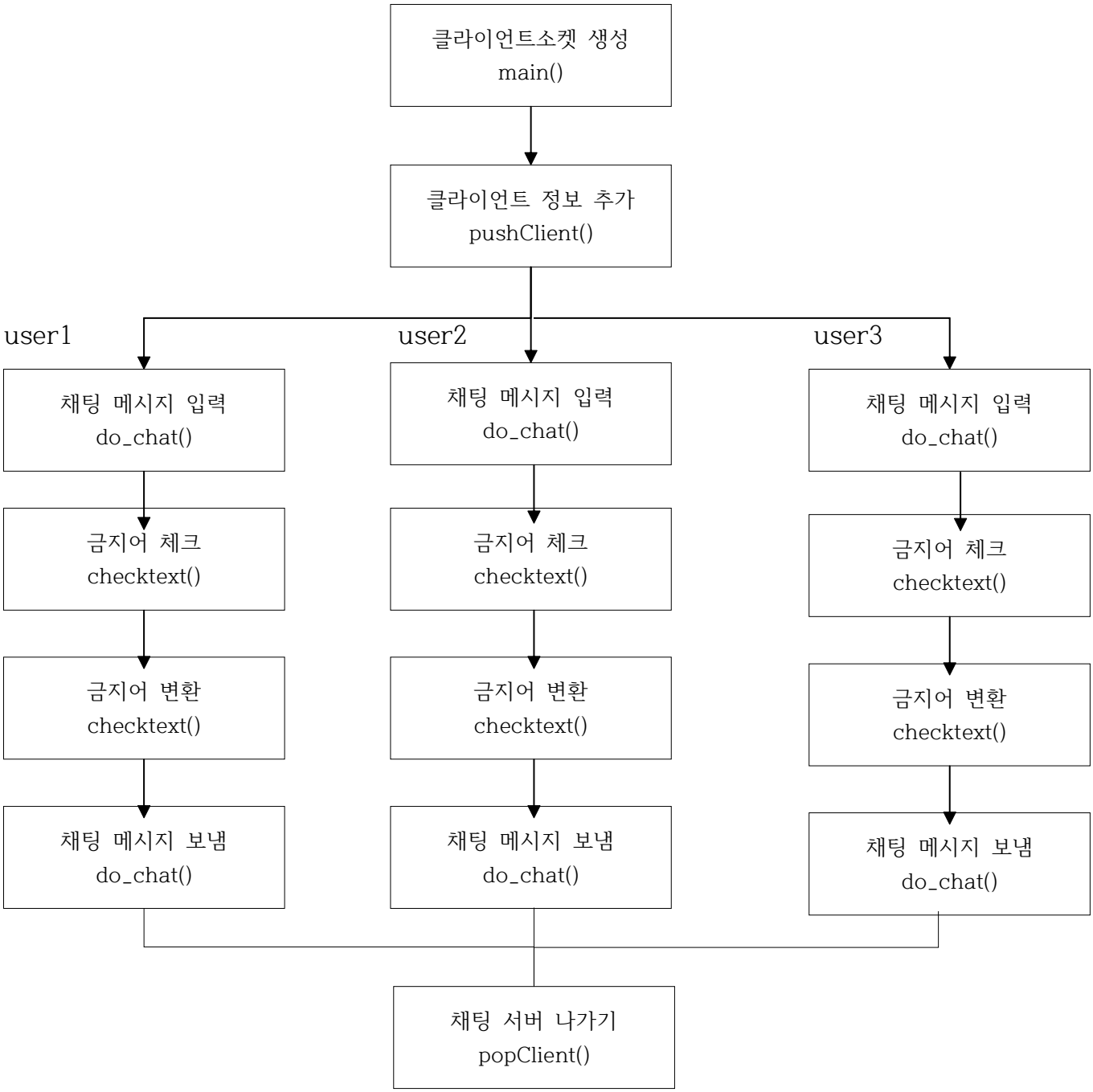
1. 기본 구조



1.1 클라이언트 저장 구조



1.2 채팅서버 구조


클라이언트소켓 생성
main()
클라이언트 정보 추가
pushClient()
user1
user2
user3
채팅 메시지 입력
do_chat()
금지어 체크
checktext()
금지어 변환
checktext()
채팅 메시지 보냄
do_chat()
채팅 메시지 입력
do_chat()
금지어 체크
checktext()
금지어 변환
checktext()
채팅 메시지 보냄
do_chat()
채팅 메시지 입력
do_chat()
금지어 체크
checktext()
금지어 변환
checktext()
채팅 메시지 보냄
do_chat()
채팅 서버 나가기
popClient()

2. 기능 설명

기능	설명.
chanel : [number]	채팅방을 선택하여 입장합니다.
/w [nickname] [message]	귓속말 기능으로 원하는 사용자에게만 메시지를 보냅니다.
/?	채팅방 기능과 사용법에 대해 알려줍니다.
****	욕설 입력 시 경고 횟수를 알려줍니다. 경고 3회 누적 시 채팅서버에서 퇴장당합니다.
/userlist	현재 채팅 채널에 접속중인 사용자들의 닉네임을 출력합니다.
/nickname	현재 접속중인 클라이언트의 닉네임을 알려줍니다.
exit	채팅방이 종료됩니다.

3. 코드

chatServer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>
#include <stdbool.h>

#define MAX_CLIENT 10
#define CHATDATA 1024
#define INVALID_SOCK -1
#define PORT 9000
#define USERNAME 256
#define MAX_CHANNEL 3
#define MAX_WARNING_COUNT 3

struct user{
        char name[USERNAME];
        int sock;
        int ch;
        int warningCnt;
        bool isBad;
};

void *do_chat(void *); //채팅 메세지를 보내는 함수
int pushClient(int); //새로운 클라이언트가 접속했을 때 클라이언트 정보 추가
int popClient(int); //클라이언트가 종료했을 때 클라이언트 정보 삭제
void checktext(char *word, int);
void checkword(char *word, const char* find, int);

//int     list_c[MAX_CLIENT]; //접속한 클라이언트를 관리하는 배열
```

```
struct user list_c[MAX_CLIENT];
char    escape[ ] = "exit";
char    userlist[ ] = "/userlist";
char    nickname[ ] = "/nickname";
char    greeting[ ] = "Welcome to chatting room\n help command > /? \n";
char    CODE200[ ] = "Sorry No More Connection\n";
char    warning[ ] = "비속어가 포함된 문장입니다.\n";
char    help[ ] = "/?";
char    helpline[ ] = "/w : 귓속말 대화 기능\n/userlist : 채널 접속중인 사용자 보기\n/nickname : 내 아이디 보기
\nexit : 나가기\n**욕설누적 3회시 강퇴입니다.\n채널을 바꾸시려면 채팅방 종료 후 재접속 해주세요\n\n";
char    retreat[ ] = "욕설 누적 3회 강퇴입니다.\n";


pthread_t thread;
pthread_mutex_t mutex;

int chList[MAX_CHANNEL];

int main(int argc, char *argv[ ])
{
    int c_socket, s_socket;
    struct sockaddr_in s_addr, c_addr;
    int     len;
    int     i, j, n;
    int     res;

    if(pthread_mutex_init(&mutex, NULL) != 0) {
        printf("Can not create mutex\n");
        return -1;
    }

    s_socket = socket(PF_INET, SOCK_STREAM, 0);
    memset(&s_addr, 0, sizeof(s_addr));
    s_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    s_addr.sin_family = AF_INET;
    s_addr.sin_port = htons(PORT);

        // 소켓 생성 후 LED1 blink

    if(bind(s_socket, (struct sockaddr *)&s_addr, sizeof(s_addr)) == -1) {
        printf("Can not Bind\n");
        return -1;
    }

    if(listen(s_socket, MAX_CLIENT) == -1) {
        printf("listen Fail\n");
        return -1;
```

```
    }

        // listen 상태임을 LED2 blink로 표시

    for(i = 0; i < MAX_CLIENT; i++)
        list_c[i].sock = INVALID_SOCK;

    //channel setting
    for(i = 0; i < MAX_CHANNEL; i++)
        chList[i] = i+1;

    while(1) {
        len = sizeof(c_addr);
        c_socket = accept(s_socket, (struct sockaddr *) &c_addr, &len);
        res = pushClient(c_socket); //접속한 클라이언트를 list_c에 추가
        if(res < 0) { //MAX_CLIENT만큼 이미 클라이언트가 접속해 있다면,
            write(c_socket, CODE200, strlen(CODE200));
            close(c_socket);
        } else {
            write(c_socket, greeting, strlen(greeting));
            //pthread_create with do_chat function.
            pthread_mutex_lock(&mutex);
            pthread_create(&thread, NULL, do_chat, (void*)&c_socket);
            pthread_mutex_unlock(&mutex);
        }
    }
}

void *do_chat(void *arg)
{
    int c_socket = *((int *)arg);
    char chatData[CHATDATA], sendData[CHATDATA], tempData[CHATDATA], warningData[CHATDATA];
    char userChk[USERNAME];
    int i, n, ch;
    char *chk, *uname, *msg;
        char yourname[30];

    while(1) {
        memset(chatData, 0, sizeof(chatData));
        if((n = read(c_socket, chatData, sizeof(chatData))) > 0) {
            strcpy(tempData, chatData);
            chk = strtok(tempData, " ");
            chk = strtok(NULL, " ");

            checktext(chatData, c_socket);

                for(i=0; i<MAX_CLIENT; i++){
```

```
                    if(list_c[i].sock == c_socket & list_c[i].isBad == true){
                                        ++list_c[i].warningCnt;
                                        list_c[i].isBad = false;

                                        sprintf(warningData, "욕설 누적 횟수 : %d회!!\n ",
list_c[i].warningCnt);

                                                write(c_socket, warning, strlen(warning));
                                                write(c_socket, warningData, strlen(warningData));

                                                if(list_c[i].warningCnt >= MAX_WARNING_COUNT){
//욕설 누적 강퇴
                                                        write(c_socket, retreat,
strlen(retreat));
                                                        popClient(c_socket);
                                        }
                                }
                } // 욕설 경고


            if(strncmp(chk, "/w", 2)==0){   //귓속말
                uname = strtok(NULL, " ");
                msg = strtok(NULL, "\0");

                //송신
                for(i=0; i<MAX_CLIENT; i++){
                        if(list_c[i].sock == c_socket){
                                strcpy(userChk, list_c[i].name);
                                break;
                        }
                }

                //수신자
                for(i=0; i<MAX_CLIENT; i++){
                        if(strncmp(list_c[i].name, uname, strlen(uname))==0){
                                sprintf(sendData, "[%s] %s", userChk, msg);
                                write(list_c[i].sock, sendData, n);
                        }
                }
            }
                else if(strstr(chatData, escape) != NULL) { //채팅방 나가기
                    popClient(c_socket);
                    break;
        }else if(strstr(chatData, userlist) != NULL){

                for(i=0; i<MAX_CLIENT; i++){
                        if(list_c[i].sock == -1) break;
```

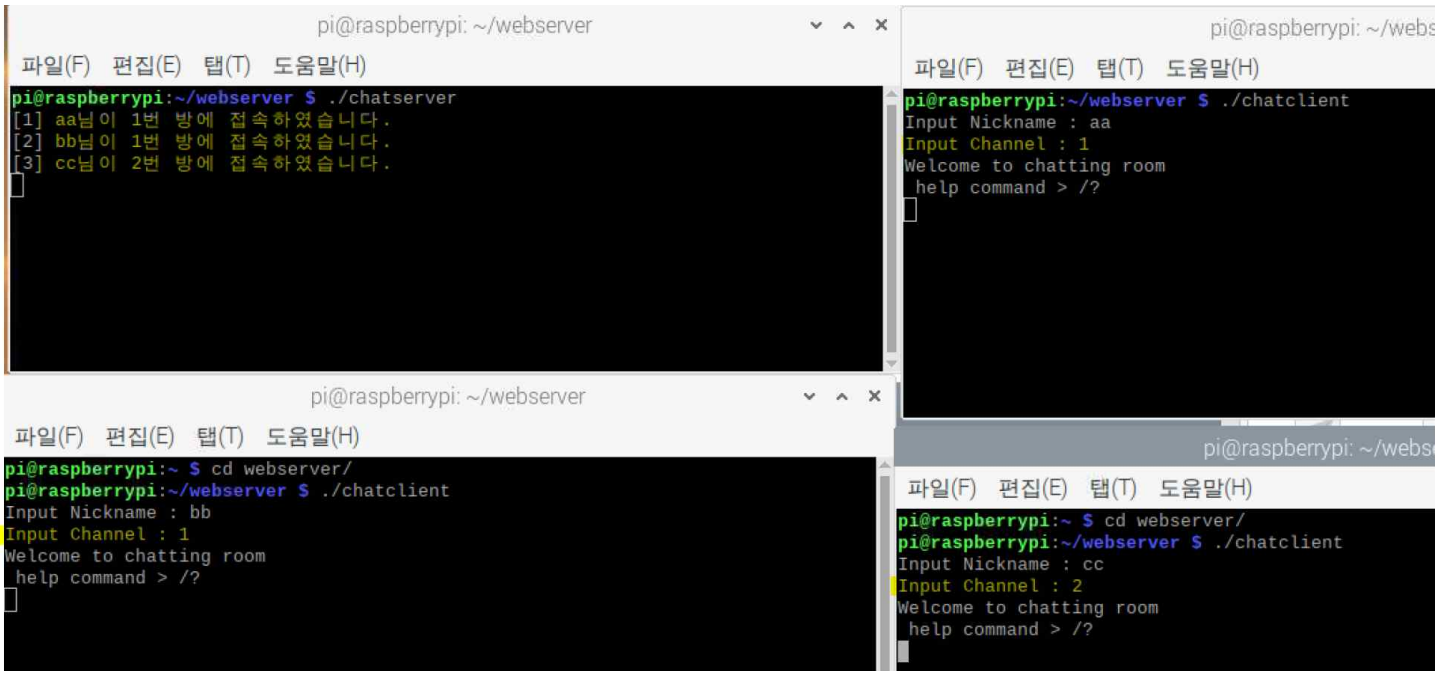
```
                        sprintf(sendData, "[%s] \n", list_c[i].name);
                        write(c_socket, sendData, n);
                }
                }else if(strstr(chatData, help) != NULL){
                        write(c_socket, helpline, strlen(helpline));
                }else if(strstr(chatData, nickname) != NULL){
                                        for(i=0; i<MAX_CLIENT; i++){
                                if(list_c[i].sock == c_socket){
                                                        sprintf(yourname, "your nickname : %s\n",
list_c[i].name);
                                                }
                                        }
                                                write(c_socket, yourname, strlen(yourname));
                }
                else{
                                for(i=0; i<MAX_CLIENT; i++){
                                if(list_c[i].sock==c_socket){
                                ch = list_c[i].ch;
                                break;
                        }
                }

                for(i=0; i<MAX_CLIENT; i++){
                        if(list_c[i].sock==INVALID_SOCK)  break;
                        if(list_c[i].ch==ch)            write(list_c[i].sock, chatData, n);
                }
            }
        }

    }
}

void checktext(char *word, int c_socket){
        char* badword[4] = {"fuck","Fuck", "suck", "asshole"};
        int cnt;
        int checksize = 4;

        for(cnt=0; cnt < checksize; cnt++){
                checkword(word, badword[cnt], c_socket);

        }
}


void checkword(char *word, const char *find, int c_socket){
        int cnt, find_size;
```

```
        char *position;

        find_size = strlen(find);
        while((position = strstr(word, find)) != NULL){
                for(cnt = 0; cnt < find_size; cnt++){
                        *(word + (strlen(word) - strlen(position)) + cnt) = '*';
                        }

                for(int i = 0; i <MAX_CLIENT; i++){
                if(list_c[i].sock==c_socket){
                      list_c[i].isBad = true;
                 }
             }
        }
}


int pushClient(int c_socket) {
        int i, j, n, ch;
        char nickname[USERNAME];

        n = read(c_socket, nickname, sizeof(nickname));
        nickname[n] = '\0';
        n = read(c_socket, &ch, sizeof(int));
        for(i=0; i<MAX_CLIENT; i++){
                if(list_c[i].sock==INVALID_SOCK){
                        list_c[i].sock = c_socket;
                        strcpy(list_c[i].name, nickname);

                        for(j=0; j<MAX_CHANNEL; j++){
                                if(ch==chList[j]){
                                     list_c[i].ch = ch;
                                     break;
                                }
                        }

                        if(j==MAX_CHANNEL)      return -1;

                        printf("[%d] %s님이 %d번 방에 접속하였습니다.\n", i+1, list_c[i].name, list_c[i].ch);
                        return i;
                }
        }

        if(i==MAX_CLIENT)         return -1;
}

int popClient(int c_socket)
```

```
{
        int i, j;

        close(c_socket);

        for(i=0; i<MAX_CLIENT; i++){
                if(list_c[i].sock==c_socket){
                        printf("%s님이 퇴장하셨습니다.\n", list_c[i].name);
                        for(j=i; j<MAX_CLIENT; j++){
                                list_c[j].sock=list_c[j+1].sock;
                                if(list_c[j].sock==INVALID_SOCK)    break;
                        }
                }
        }

        return j;
}
```

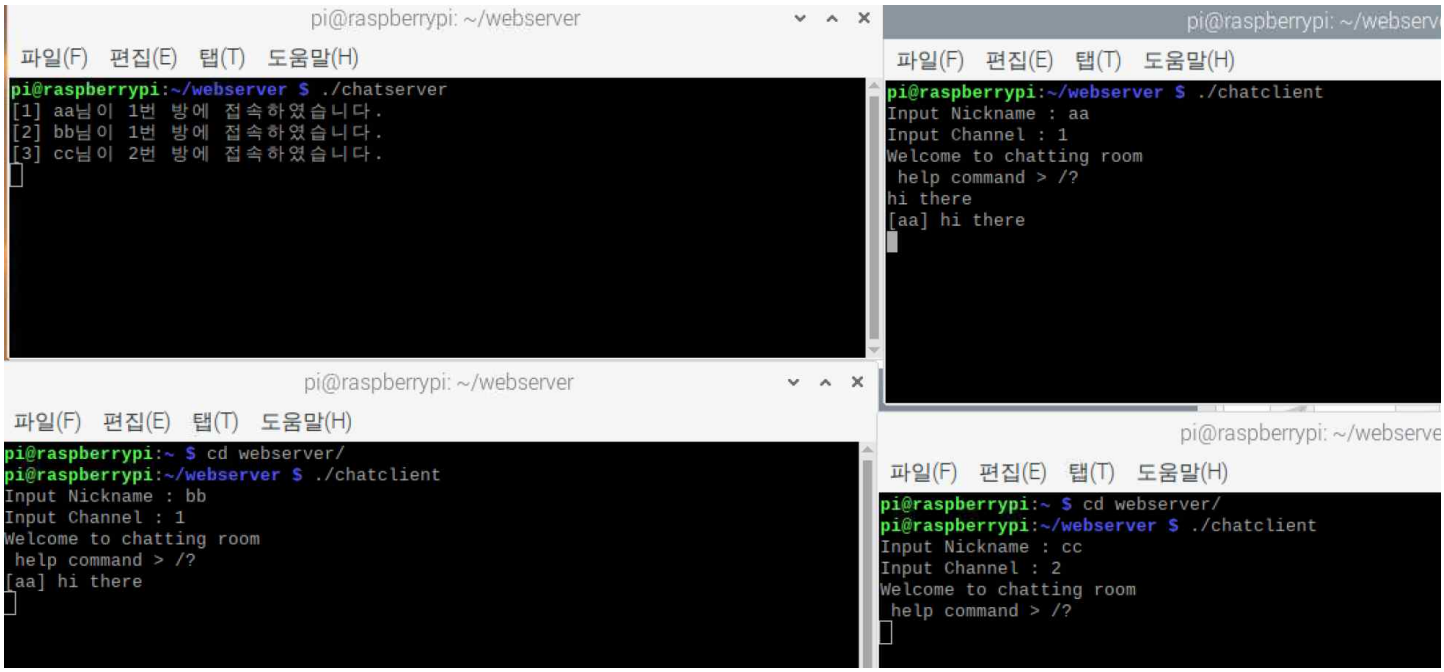
4. 시현 장면

4.1 입장



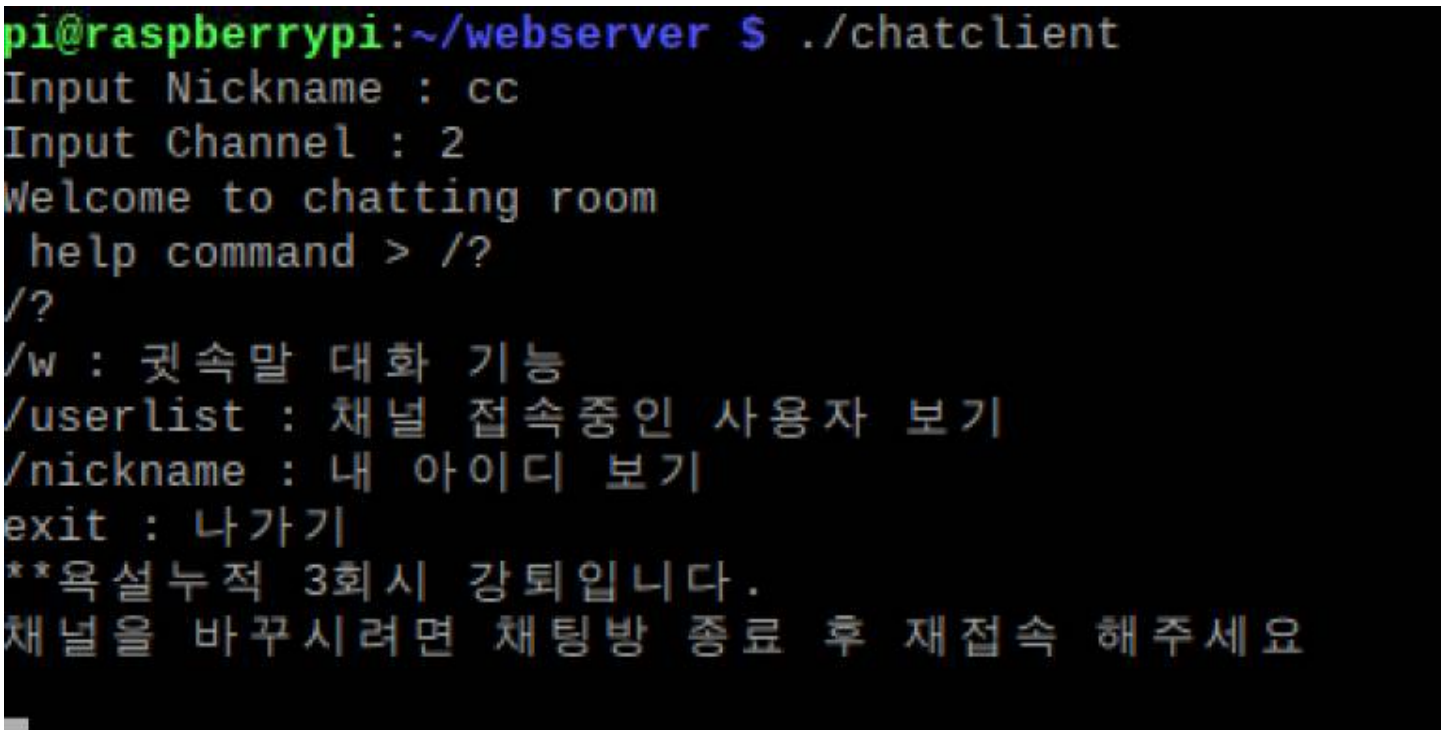
서버에서는 몇 번 채널에 누가 입장했는지 확인 할 수 있습니다.

4.2 채널 별 채팅



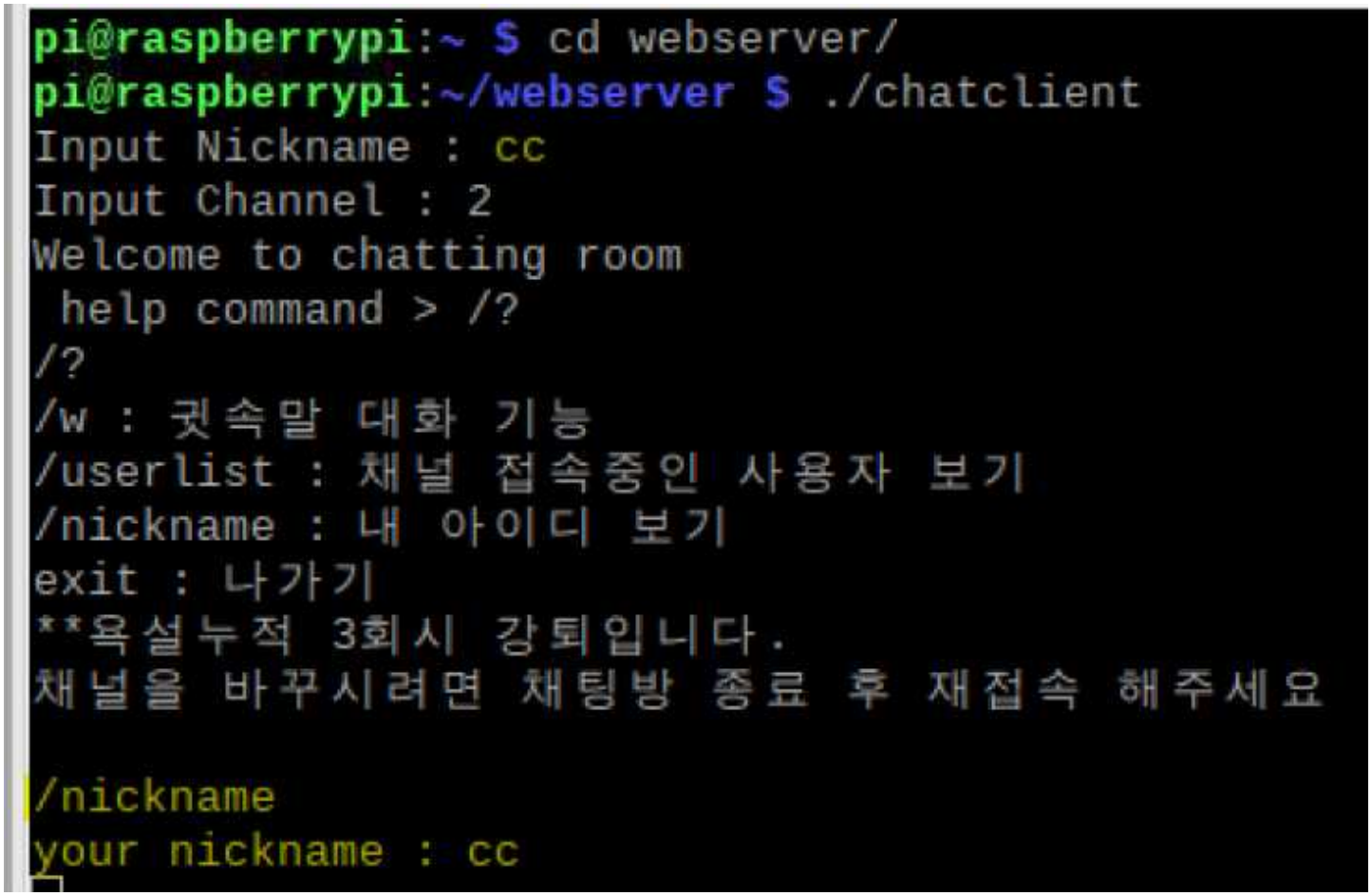
동일한 채널끼리만 통신이 가능합니다.

4.3 도움말 요청



/?를 입력하면 사용 가능한 기능과 주의할 점을 알려줍니다.

4.4 내 닉네임 보기

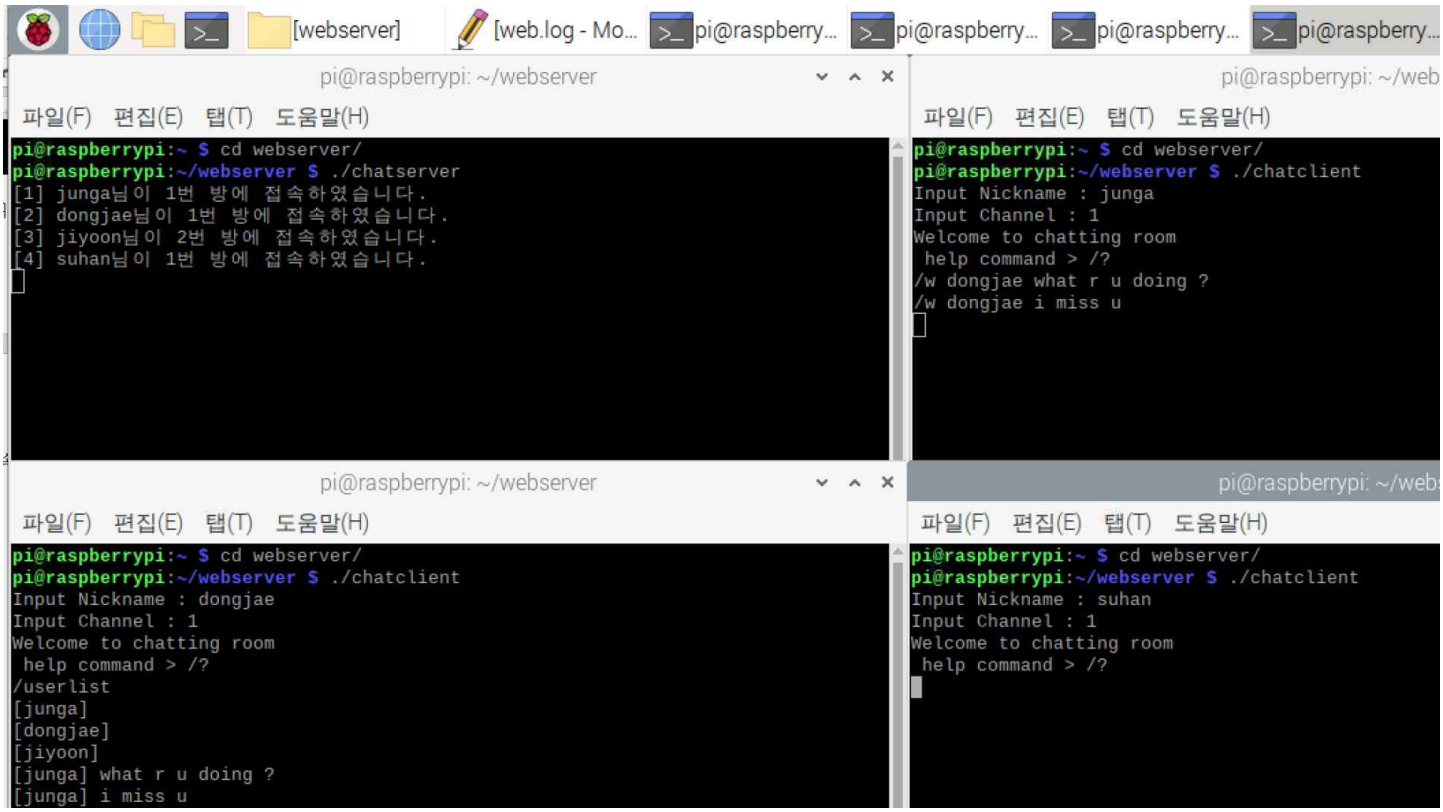


사용자가 입장할 때 설정한 자신의 닉네임이 출력됩니다.

4.5 접속중인 사용자 보기

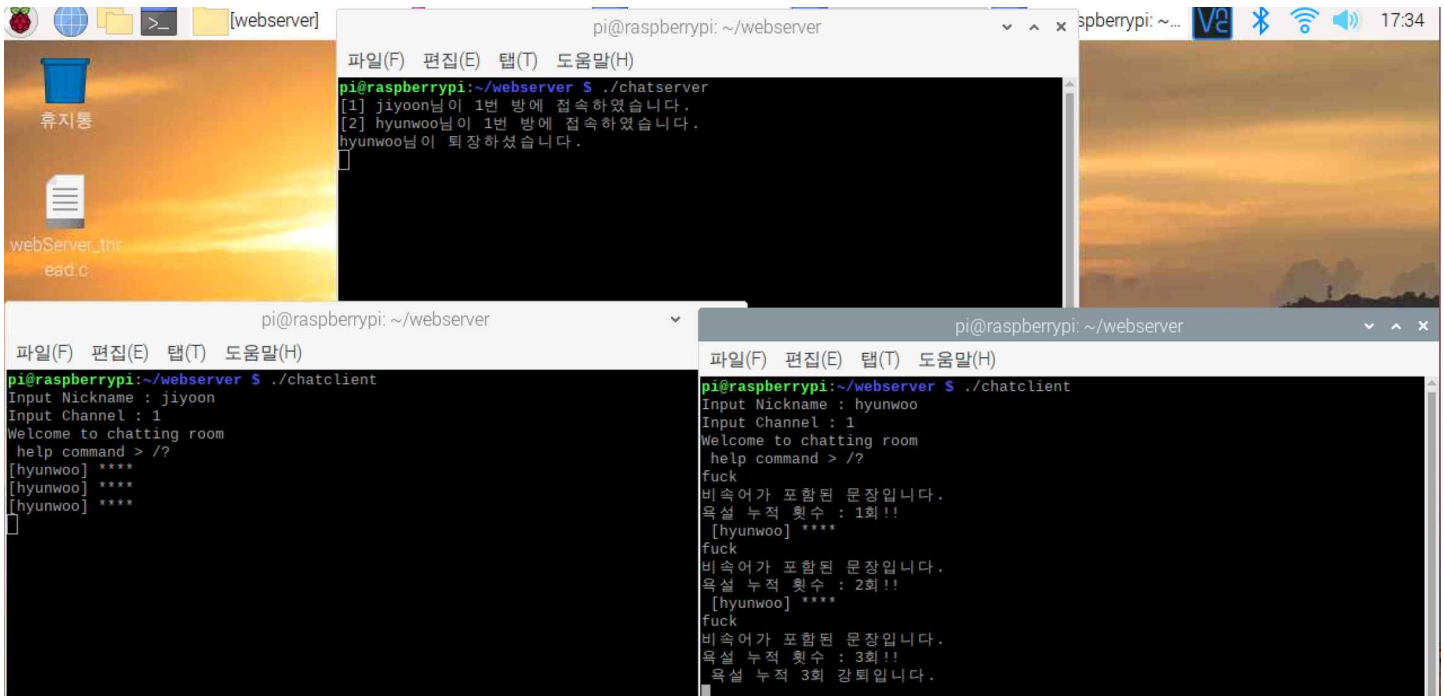
현재 채팅방에 접속중인 사용자들의 리스트를 출력합니다.

4.6 귓속말 기능



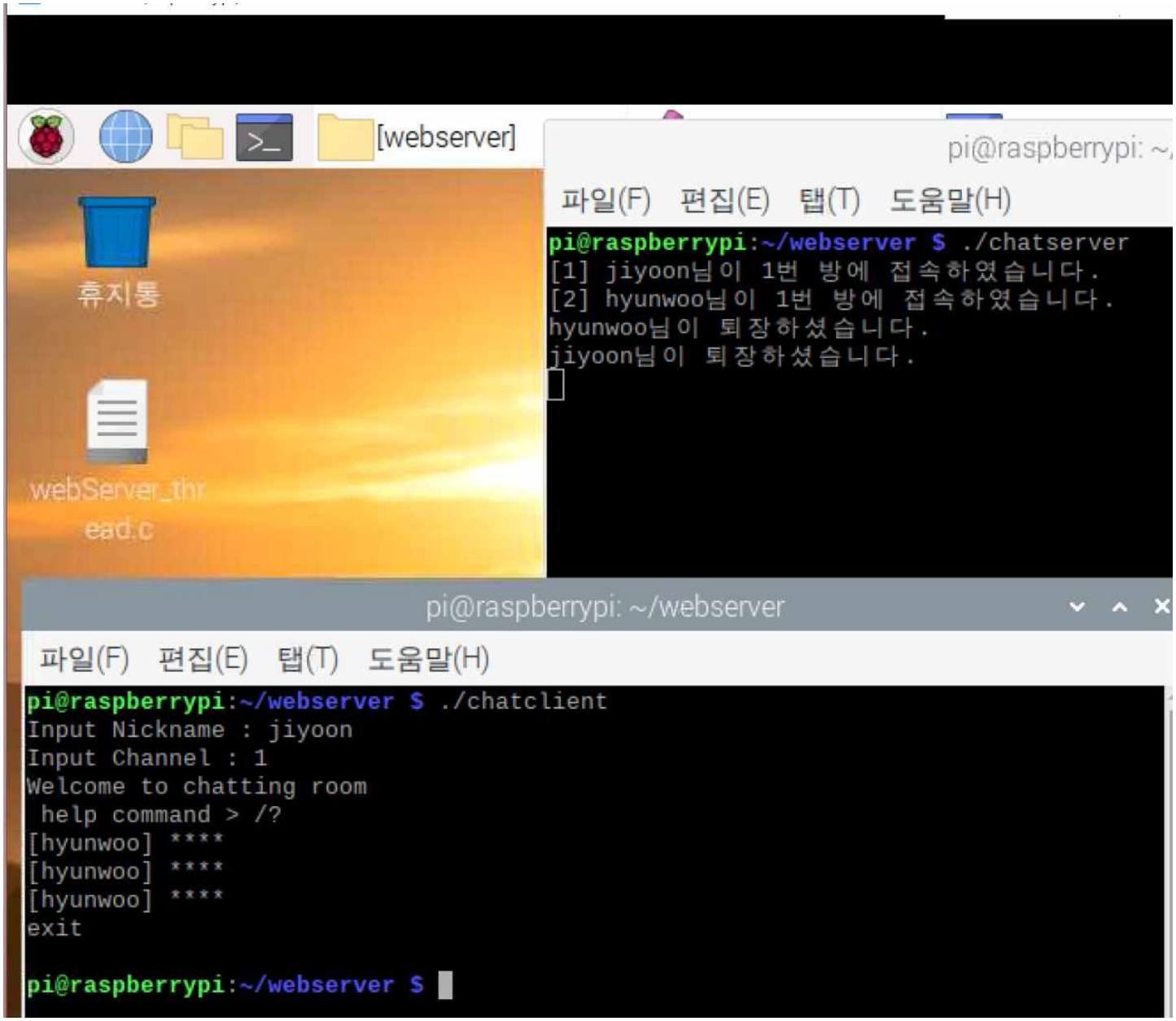
원하는 사용자에게만 메시지를 보낼 수 있습니다.

4.7 금지어 변환과 경고 누적 강퇴 기능



사용자가 금지어를 입력하면 금지어를 *로 변환하고 경고를 누적합니다.
경고 누적이 3회가 되면 서버에서 자동으로 퇴장시킵니다.

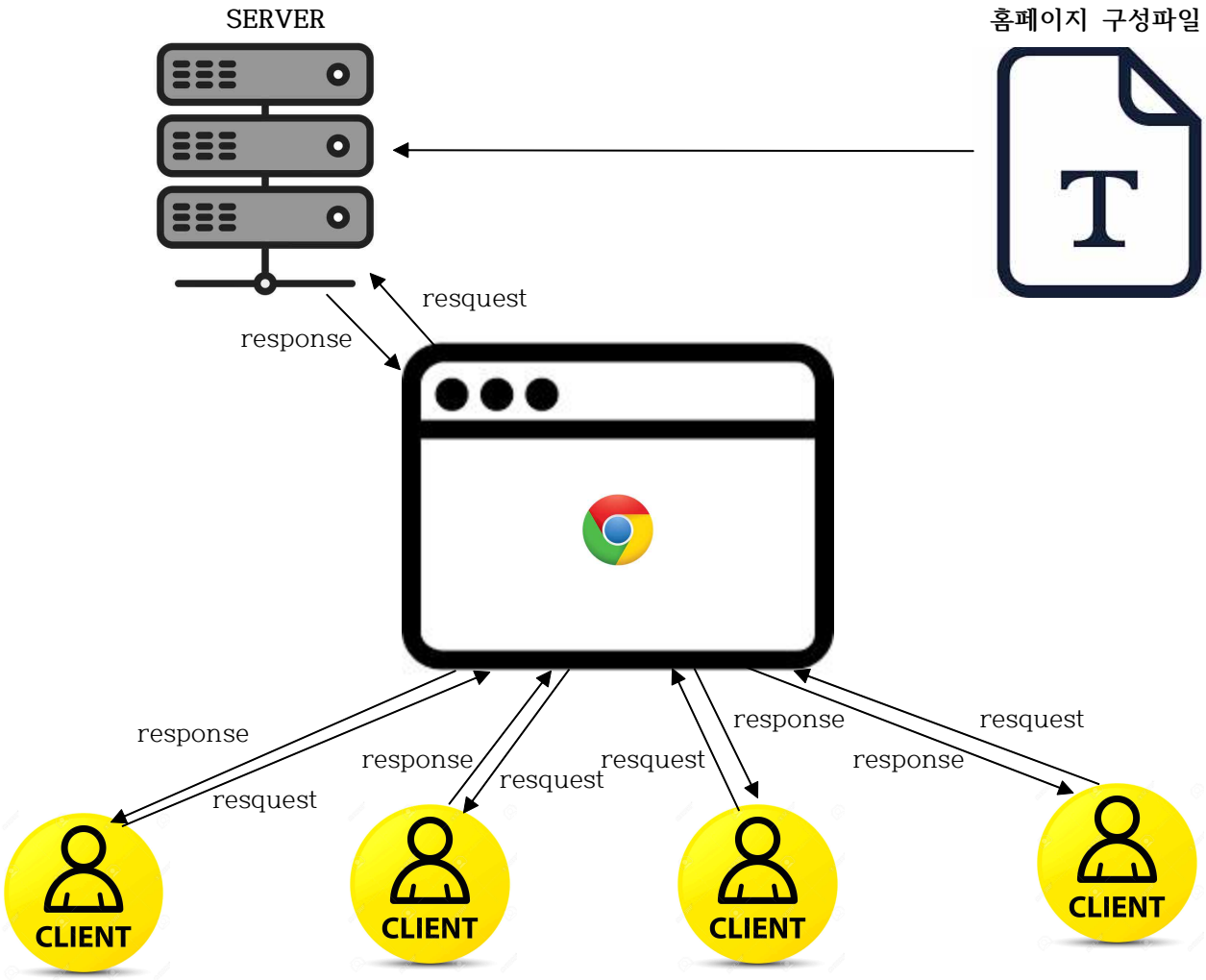
4.7 퇴장 기능



특정 명령어 exit를 입력하면 채팅방에서 퇴장합니다.

Ⅱ. 웹 서버 구현

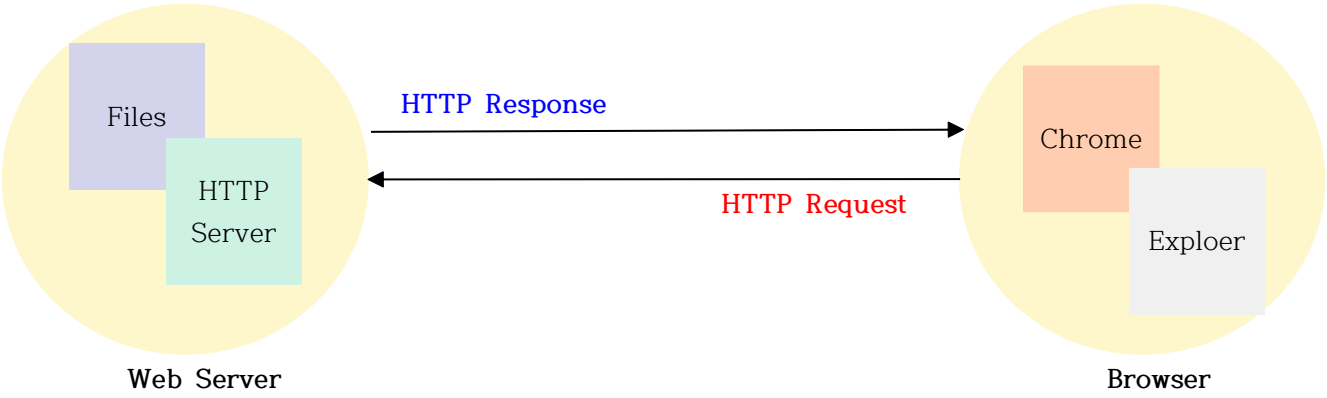
1.1 기본 구조



웹서버는 기본적으로 웹 사용자가 어떻게 호스트 파일들에 접근하는지를 관리합니다.
이 문서에서 web servers는 HTTP로 국한합니다. HTTP서버는 URL과 HTTP의 소프트웨어 일부입니다.

가장 기본적인 단계에서, 브라우저가 웹 서버에 불려진 파일을 필요로 할 때 브라우저는 HTTP를 통해 파일을 요청합니다. 요청이 올바른 웹 서버에 도달하였을 때, HTTP 서버는 요청된 문서를 HTTP를 이용해 보내줍니다.

1.2 HTTP의 구조



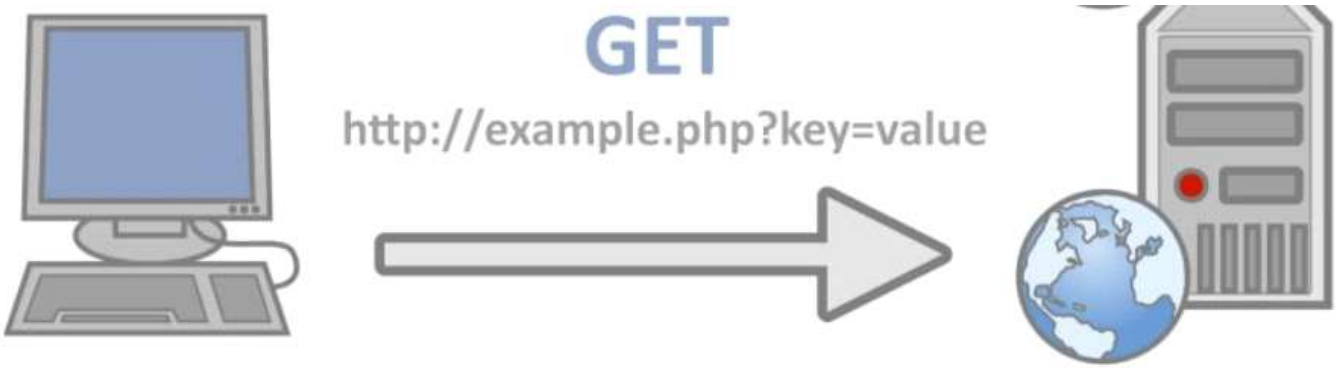
1.3 Request의 방식 Method

메서드는 요청의 종류를 서버에게 알려주기 위해 사용한다. 아래의 표는 요청에 사용할 수 있는 메서드들이다.

GET	정보를 요청하기 위해 사용한다. (SELECT)
POST	정보를 밀어넣기 위해 사용한다. (INSERT)
PUT	정보를 업데이트하기 위해 사용한다. (UPDATE)
DELETE	정보를 삭제하기 위해서 사용한다.
HEAD	(HTTP)헤더 정보만 요청한다. 해당 자원이 존재하는지 혹은 서버에 문제가 없는지를 확인하기 위해서 사용한다.
OPTIONS	웹서버가 지원하는 메서드의 종류를 요청한다
TRACE	클라이언트의 요청을 그대로 반환한다. 예컨대 ECHO 서비스로 서버 상태를 확인하기 위한 목적으로 주로 사용한다.

다양한 메소드들이 존재하지만 그 중 가장 많이 사용되는 GET과 POST에 대해 실습해봤다.

1.3.1 GET



GET 메소드는 주로 데이터를 읽거나(Read) 검색(Retrieve)할 때에 사용되는 메소드이다.
GET 방식으로 데이터를 보낼땐 클라이언트의 데이터를 URL뒤에 붙여서 보낸다. 아래는 그 예시이다.

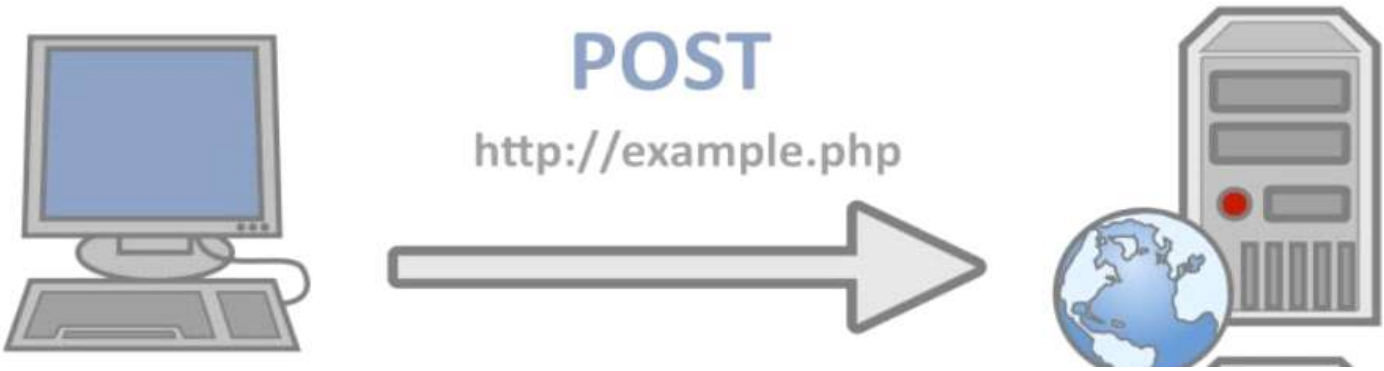
```
www.junga.com?id=20170735&pass=jajjag
```

URL뒤에 “?” 마크를 통해 URL의 끝을 알리면서, 데이터 표현의 시작점을 알린다. 데이터는 key와 value를 쌍으로 입력한다. 또한 두 개 이상의 key-value쌍을 보낼 때는 & 마크로 구분한다.
URL에 붙여서 데이터를 전송하는 방식으로 당연히 HTTP패킷의 헤더에 포함되여 서버에 요청한다. 따라서 BODY는 빈 상태로 보내진다.

GET요청이 성공적으로 이루어진다면 XML이나 JSON과 함께 200 (Ok) HTTP 응답 코드를 리턴한다. 에러가 발생하면 주로 404 (Not found) 에러나 400 (Bad request) 에러가 발생한다.

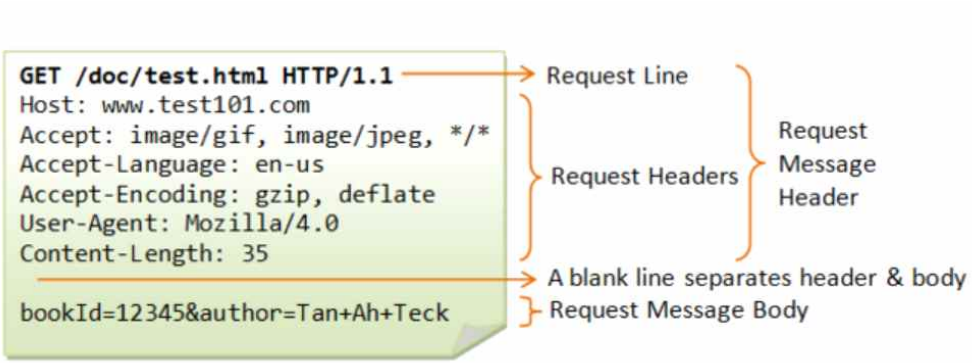
GET은 URL형태로 표현되므로, 특정 페이지를 다른사람 에게 접속하게 할 수 있다.
또한 간단한 데이터를 넣도록 설계되어, 데이터를 보내는 양의 한계가 있다.

1.3.2 POST



POST방식은 GET방식과 달리, 데이터 전송을 기반으로 한 요청 메서드이다.
GET 방식은 URL에 데이터를 붙여서 보내는 반면, POST 방식은 URL에 붙여서 보내지 않고 BODY에 데이터를 넣어서 보낸다. 헤더필드 중 BODY의 데이터를 설명하는 Content-Type이라는 헤더가 존재해야하고 데이터 타입을 반드시 명시해야한다.

1.4 HTTP 요청 데이터 포멧



요청 데이터는 “HEADER”와 “BODY”로 구성된다.
헤더에는 요청과 요청데이터에 대한 메타 정보들이 들어있다. 즉 GET 방식으로 전송하면 헤더 부분에서 볼 수 있고, POST 방식으로 전송하면 BODY에 숨어 데이터가 들어온다.

1.5 HTTP 연결상태

모든 HTTP 응답 코드는 5개의 클래스(분류)로 구분된다. 상태 코드의 첫 번째 숫자는 응답의 클래스를 정의한다. 마지막 두 자리는 클래스나 분류 역할을 하지 않는다. 첫자리에 대한 5가지 값들은 다음과 같다.

- 1xx (정보): 요청을 받았으며 프로세스를 계속한다.
  HTTP/1.0이래로 어떤 1XX 상태 코드들도 정의 되지 않았다. 서버들은 1XX 응답을 실험적인 상태를 제외하고 HTTP/1.0 클라이언트(서버에 연결된 컴퓨터)로 보내면 안 된다.

- 2xx (성공): 요청을 성공적으로 받았으며 인식했고 수용하였다.
  이 클래스의 상태 코드는 클라이언트가 요청한 동작을 수신하여 이해했고 승낙했으며 성공적으로 처리했음을 가리킨다.

- 3xx (리다이렉션): 요청 완료를 위해 추가 작업 조치가 필요하다.
  클라이언트는 요청을 마치기 위해 추가 동작을 취해야 한다.

- 4xx (클라이언트 오류): 요청의 문법이 잘못되었거나 요청을 처리할 수 없다.
  4xx 클래스의 상태 코드는 클라이언트에 오류가 있음을 나타낸다.

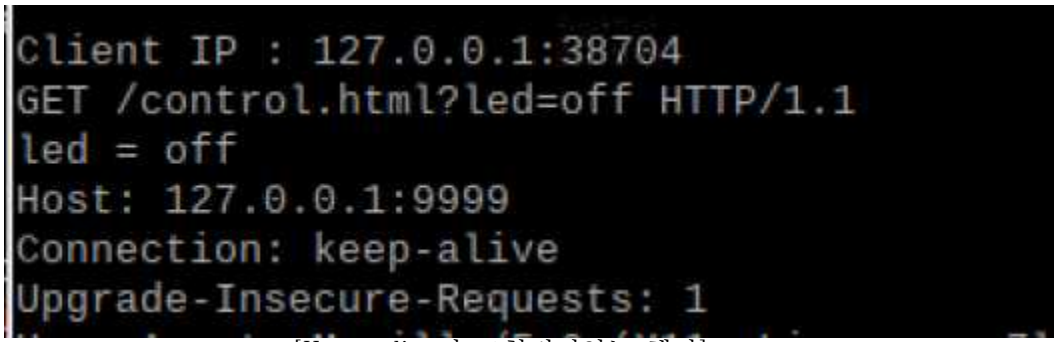
· 5xx (서버 오류): 서버가 명백히 유효한 요청에 대해 충족을 실패했다.
서버가 유효한 요청을 명백하게 수행하지 못했음을 나타낸다.

다음은 많은 응답코드 중 주요 응답코드이다.

```
HTTP/1.1 200 OK\r\n
HTTP/1.1 404 Not Found\r\n
HTTP/1.1 500 Internal error\r\n
```

1.6 Keep alive 설정

Keep-alive는 HTTP/1.1 부터 사용할 수 있으며 웹 브라우저는 요청 헤더에 keep-alive 방식의 연결을 할 것을 명시한다.



[Keep alive가 포함되어있는 헤더]

2. 기능 설명

2.1 GET

RED LED on/off/blink	빨간색 LED를 켜고 끄고 반짝 일 수 있습니다.
BLUE LED on/off/blink	파란색 LED를 켜고 끄고 반짝 일 수 있습니다.
MUSIC PLAY	부저를 통한 음악을 재생할 수 있습니다.
Moter	모터를 회전할 수 있습니다.
submit	위의 상태를 설정하고 GET 방식으로 요청합니다.

2.2 POST

LED on/off	빨간색 LED를 켜고 끄고 반짝일 수 있습니다.
submit	위의 상태를 설정하고 POST 방식으로 요청합니다.

3. 코드

3.1 C

webcontrol.c

```
// webcontrol_2.c
// Control the raspberry Pi from webserver.
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <wiringPi.h>
#include <softTone.h>
#include <errno.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define CDS     0       /* GPIO17 */
#define LED1    1       /* GPIO18 */
#define LED2    26      // GPIO12
#define SW      5       /* GPIO24 */
#define SPKR    6       /* GPIO25 */
#define MOTOR 2         /* GPIO13 */
#define TOTAL  32       /* 학교종의 전체 계이름의 수 */
int notes[] = {         /* 학교종을 연주하기 위한 계이름 */
    391, 391, 440, 440, 391, 391, 329.63, 329.63, ₩
    391, 391, 329.63, 329.63, 293.66, 293.66, 293.66, 0, ₩
    391, 391, 440, 440, 391, 391, 329.63, 329.63, ₩
    391, 329.63, 293.66, 329.63, 261.63, 261.63, 261.63, 0
};
pthread_mutex_t music_lock;
pthread_mutex_t motor_lock;
static int is_run =1; /* 스레드 종료를 위한 플래그 */
void *musicfunction(void *arg)
{
    if(pthread_mutex_trylock(&music_lock) != EBUSY) { /* 임계 구역 설정 */
        musicPlay();
        pthread_mutex_unlock(&music_lock); /* 임계 구역 해제 */
    }
    return NULL;
}
void *motorfunction(void *arg)
{
    if(pthread_mutex_trylock(&motor_lock) != EBUSY) { /* 임계 구역 설정 */
        motorControl(MOTOR);
        pthread_mutex_unlock(&motor_lock); /* 임계 구역 해제 */
    }
    return NULL;
}
int ledControl(int gpio, int onoff)
{
    pinMode(gpio, OUTPUT) ;                 /* 핀(Pin)의 모드 설정 */
    digitalWrite(gpio, (onoff)?HIGH:LOW);   /* LED 켜고 끄기 */
    return 0;
```

```
}
void *switchfunction(void *arg)
{
    int i;

    pinMode(SW, INPUT);                         /* 핀의 모드를 입력으로 설정 */
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    while(is_run) {
        if(digitalRead(SW) == LOW) { /* Push 버튼이 눌러지면(LOW) */
            digitalWrite(LED1, HIGH); /* LED 켜기(On) */
            digitalWrite(LED2, LOW);
        } else {
            digitalWrite(LED1, LOW); /* LED 끄기(Off) */
            digitalWrite(LED2, HIGH);
        }
    };
    return 0;
}
int musicPlay()
{
    int i ;
    softToneCreate(SPKR) ;                      /* 톤 출력을 위한 GPIO 설정 */
    for (i =0; i < TOTAL; ++i) {
        softToneWrite(SPKR, notes[i]);          /* 톤 출력 : 학교종 연주 */
        delay(280);                             /* 음의 전체 길이만큼 출력되도록 대기 */
    }
    return 0;
}
void *cdsfunction(void *arg)
{
    int i;
    pinMode(SW, INPUT);                         /* 핀의 모드를 입력으로 설정 */
    pinMode(CDS, INPUT);                        /* 핀의 모드를 입력으로 설정 */
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);     /* 핀의 모드를 출력으로 설정 */
    while(is_run) {
        if(digitalRead(CDS) == HIGH) {          /* 빛이 감지되면(HIGH) */
            digitalWrite(LED1, HIGH);           /* LED 켜기(On) */
            delay(1000);
        } else {
            digitalWrite(LED1, LOW);            /* LED 끄기(Off) */
        }
    };
    return 0;
}
int motorControl(int gpio)
{
    int i;
    pinMode(gpio, OUTPUT) ;                     /* 핀(Pin)의 모드 설정 */
    for (i =0; i <3; i++) {
        digitalWrite(gpio, HIGH);               /* HIGH(1) 값을 출력 : 모터 켜기 */
        delay(1000);                            /* 1초(1000ms) 동안 대기 */
        digitalWrite(gpio, LOW);          /* LOW(0) 값을 출력 : 모터 끄기 */
        delay(1000);
    };
    return 0;
}
int kbhit()
{
```

```
    struct termios oldt, newt;                  /* 터미널에 대한 구조체 */
    int ch, oldf;
    tcgetattr(0, &oldt);                /* 현재 터미널에 설정된 정보를 가져온다. */
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);                   /* 정규 모드 입력과 에코를 해제한다. */
    tcsetattr(0, TCSANOW, &newt);               /* 새로 값으로 터미널을 설정한다. */
    oldf = fcntl(0, F_GETFL, 0);
    fcntl(0, F_SETFL, oldf | O_NONBLOCK);       /* 입력을 논블로킹 모드로 설정한다. */
    ch = getchar();
    tcsetattr(0, TCSANOW, &oldt);               /* 기존의 값으로 터미널의 속성을 바로 적용한다. */
    fcntl(0, F_SETFL, oldf);
    if(ch != EOF) {
        ungetc(ch, stdin);                      /* 앞에서 읽은 위치로 이전으로 포인터를 돌려준다. */
        return 1;
    }
    return 0;
}
/* 스레드 처리를 위한 함수 */
static void *clnt_connection(void *arg);
int sendData(int fd, FILE* fp, char *ct, char *file_name);
void sendOk(FILE* fp);
void sendError(FILE* fp);
static pthread_t ptSwitch, ptCds, ptMusic, ptMotor;
int main(int argc, char** argv)
{
    int serv_sock;
    pthread_t thread;
    struct sockaddr_in serv_addr, clnt_addr;
    unsigned int clnt_addr_size;
    pthread_t ptSwitch, ptCds, ptMusic, ptMotor, ptchat;
    pthread_mutex_init(&music_lock, NULL); /* 뮤텍스 초기화 */
    pthread_mutex_init(&motor_lock, NULL);
    printf("l : LED On, L : LED Off, m : Music, d : DC Motor, q : Quit₩n");
    wiringPiSetup();
    pthread_create(&ptSwitch, NULL, switchfunction, NULL);
    pthread_create(&ptCds, NULL, cdsfunction, NULL);
    /* 프로그램을 시작할 때 서버를 위한 포트 번호를 입력받는다. */
    if(argc!=2) {
        printf("usage: %s <port>₩n", argv[0]);
        return -1;
    }
    /* 서버를 위한 소켓을 생성한다. */
    serv_sock = socket(AF_INET, SOCK_STREAM, 0);
    if(serv_sock ==-1) {
        perror("socket( )");
        return -1;
    }
    /* 입력받는 포트 번호를 이용해서 서비스를 운영체제에 등록한다. */
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = (argc !=2)?htons(8000):htons(atoi(argv[1]));
    if(bind(serv_sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))==-1) {
        perror("bind( )");
        return -1;
    }
    /* 최대 10대의 클라이언트의 동시접속을 처리할 수 있도록 큐를 생성한다. */
    if(listen(serv_sock, 10) ==-1) {
        perror("listen( )");
        return -1;
```

```
    }
    while(1) {
        int clnt_sock;
        /* 클라이언트의 요청을 기다린다. */
        clnt_addr_size =sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
        printf("Client IP : %s:%d₩n", inet_ntoa(clnt_addr.sin_addr), ntohs(clnt_addr.sin_port));
        /* 클라이언트의 요청이 들어오면 스레드를 생성하고 클라이언트의 요청을 처리한다. */
        pthread_create(&thread, NULL, clnt_connection, &clnt_sock);
        pthread_join(thread, 0);
    };
END:
    printf("Good Bye!₩n");
    return 0;
}
void *clnt_connection(void *arg)
{
     /* 스레드를 통해서 넘어온 arg를 int 형의 파일 디스크립터로 변환한다. */
    int clnt_sock =*((int*)arg), clnt_fd;
    FILE *clnt_read, *clnt_write;
    char reg_line[BUFSIZ], reg_buf[BUFSIZ];
    char method[10], ct[BUFSIZ], type[BUFSIZ];
    char file_name[256], file_buf[256];
    char* type_buf;
    int i =0, j =0, len =0;
    /* 파일 디스크립터를 FILE 스트림으로 변환한다. */
    clnt_read = fdopen(clnt_sock, "r");
    clnt_write = fdopen(dup(clnt_sock), "w");
    clnt_fd = clnt_sock;
    /* 한 줄의 문자열을 읽어서 reg_line 변수에 저장한다. */
    fgets(reg_line, BUFSIZ, clnt_read);
    /* reg_line 변수에 문자열을 화면에 출력한다. */
    fputs(reg_line, stdout);
    /* ‘ ’ 문자로 reg_line을 구분해서 요청 라인의 내용(메소드)를 분리한다. */
    strcpy(method, strtok(reg_line, " "));
    if(strcmp(method, "POST") ==0) { /* POST 메소드일 경우를 처리한다. */
        sendOk(clnt_write); /* 단순히 OK 메시지를 클라이언트로 보낸다. */
        fclose(clnt_read);
        fclose(clnt_write);
        return (void*)NULL;
    } else if(strcmp(method, "GET") !=0) { /* GET 메소드가 아닐 경우를 처리한다. */
        sendError(clnt_write); /* 에러 메시지를 클라이언트로 보낸다. */
        fclose(clnt_read);
        fclose(clnt_write);
        return (void*)NULL;
    }
    strcpy(file_name, strtok(NULL, " ")); /* 요청 라인에서 경로(path)를 가져온다. */
    if(file_name[0] =='/') { /* 경로가 ‘/’로 시작될 경우 /를 제거한다. */
        for(i =0, j =0; i < BUFSIZ; i++) {
            if(file_name[0] =='/') j++;
            file_name[i] = file_name[j++];
            if(file_name[i+1] =='₩0') break;
        };
    }
    /* 라즈베리 파이를 제어하기 위한 HTML 코드를 분석해서 처리한다. */
    if(strstr(file_name, "?") !=NULL) {
        char optLine[32];
        char optStr[4][16];
        char opt[8], var[8];
        char* tok;
```

```
        int i, count =0;
        strcpy(file_name, strtok(file_name, "?"));
        strcpy(optLine, strtok(NULL, "?"));
        /* 옵션을 분석한다. */
        tok = strtok(optLine, "&");
        while(tok !=NULL) {
            strcpy(optStr[count++], tok);
            tok = strtok(NULL, "&");
        };
        /* 분석한 옵션을 처리한다. */
        for(i =0; i < count; i++) {
            strcpy(opt, strtok(optStr[i], "="));
            strcpy(var, strtok(NULL, "="));
            printf("%s = %s₩n", opt, var);
            if(!strcmp(opt, "led") &&!strcmp(var, "On")) { /* LED를 켠다. */
                ledControl(LED1, 1);
            } else if(!strcmp(opt, "led") &&!strcmp(var, "Off")) { /* LED를 끈다. */
                ledControl(LED1, 0);
            }
            else if(!strcmp(opt, "led2") &&!strcmp(var, "On")){
                ledControl(LED2, 1);
            } else if(!strcmp(opt, "led2") &&!strcmp(var, "Off")){
                ledControl(LED2, 0);
            }
            else if(!strcmp(opt, "music") &&!strcmp(var, "Play")) { /* 음악을 연주한다. */
                pthread_create(&ptMusic, NULL, musicfunction, NULL);
            } else if(!strcmp(opt, "motor") &&!strcmp(var, "On")) { /* 모터를 켠다. */
                pthread_create(&ptMotor, NULL, motorfunction, NULL);
            }
        };
    }
    /* 메시지 헤더를 읽어서 화면에 출력하고 나머지는 무시한다. */
    do {
        fgets(reg_line, BUFSIZ, clnt_read);
        fputs(reg_line, stdout);
        strcpy(reg_buf, reg_line);
        type_buf = strchr(reg_buf, ':');
    } while(strncmp(reg_line, "₩r₩n", 2)); /* 요청 헤더는 '₩r₩n'으로 끝난다. */
    /* 파일의 이름을 이용해서 클라이언트로 파일의 내용을 보낸다. */
    strcpy(file_buf, file_name);
    sendData(clnt_fd, clnt_write, ct, file_name);
    fclose(clnt_read); /* 파일의 스트림을 닫는다. */
    fclose(clnt_write);
    pthread_exit(0); /* 스레드를 종료시킨다. */
    return (void*)NULL;
}
int sendData(int fd, FILE* fp, char *ct, char *file_name)
{
    /* 클라이언트로 보낼 성공에 대한 응답 메시지 */
    char protocol[ ] ="HTTP/1.1 200 OK₩r₩n";
    char server[ ] ="Server:Netscape-Enterprise/6.0₩r₩n";
    char cnt_type[ ] ="Content-Type:text/html₩r₩n";
    char end[ ] ="₩r₩n"; /* 응답 헤더의 끝은 항상 ₩r₩n */
    char buf[BUFSIZ];
    int len;
    fputs(protocol, fp);
    fputs(server, fp);
    fputs(cnt_type, fp);
    fputs(end, fp);
    fflush(fp);
```

```
    fd = open(file_name, O_RDWR); /* 파일을 연다. */
    do {
        len = read(fd, buf, BUFSIZ); /* 파일을 읽어서 클라이언트로 보낸다. */
        fwrite(buf, len, sizeof(char), fp);
    } while(len == BUFSIZ);
    fflush(fp);
    close(fd); /* 파일을 닫는다. */
    return 0;
}
void sendOk(FILE* fp)
{
    /* 클라이언트에 보낼 성공에 대한 HTTP 응답 메시지 */
    char protocol[ ] ="HTTP/1.1 200 OK₩r₩n";
    char server[ ] ="Server: Netscape-Enterprise/6.0₩r₩n₩r₩n";
    fputs(protocol, fp);
    fputs(server, fp);
    fflush(fp);
}
void sendError(FILE* fp)
{
    /* 클라이언트로 보낼 실패에 대한 HTTP 응답 메시지 */
    char protocol[ ] ="HTTP/1.1 400 Bad Request₩r₩n";
    char server[ ] ="Server: Netscape-Enterprise/6.0₩r₩n";
    char cnt_len[ ] ="Content-Length:1024₩r₩n";
    char cnt_type[ ] ="Content-Type:text/html₩r₩n₩r₩n";
    /* 화면에 표시될 HTML의 내용 */
    char content1[ ] ="<html><head><title>BAD Connection</tiitle></head>";
    char content2[ ] ="<body><font size=+5>Bad Request</font></body></html>";
    printf("send_error₩n");
    fputs(protocol, fp);
    fputs(server, fp);
    fputs(cnt_len, fp);
    fputs(cnt_type, fp);
    fputs(content1, fp);
    fputs(content2, fp);
    fflush(fp);
}
```

3.2 HTML

webpost.html

```
<html>
        <h3> POST </h3>
<head>
  <title> Raspberry Pi Controller </title>
</head>
<body>
  <form action="control.html"method="POST"onSubmit="document.reload()">
    <table>
      <tr><td>LED</td>
            <td><input type="radio"name="led"value="on"/>on </td>
            <td><input type="radio"name="led"value="off"/>off</td>
      </tr>
      <tr>
            <td> submit </td>
            <td><input type="submit"value="submit"/></td>
     </table>
  </form>
```

```
</body>
</html>
```

webget.html

```
<html>
<head>
  <title> Raspberry Pi Controller </title>
</head>
<body>
  <form action="control.html"method="GET"onSubmit="document.reload()">
    <table>
      <tr><td>RED LED</td>
            <td><input type="radio"name="led"value="on"/>on </td>
            <td><input type="radio"name="led"value="off"/>off</td>
        <td><input type="radio"name="led"value="blink"/>blink</td>
        </tr>
        <tr>
        <td>BLUE LED </td>
        <td><input type="radio"name="led2"value="on"/>on</td>
        <td><input type="radio"name="led2"value="off"/>off</td>
        <td><input type="radio"name="led2"value="blink"/>blink</td>
        </td>
        <tr>
        <td> MUSIC PLAY </td>
        <td><input type="radio"name="music"value="Play"/>
        </tr>
       <tr>
        <td> Moter </td>
        <td><input type="radio"name="moter"value="on"/>
       </tr>
      <tr>
            <td> submit </td>
            <td><input type="submit"value="submit"/></td>
     </table>
  </form>
</body>
</html>
```

4. 시현 장면

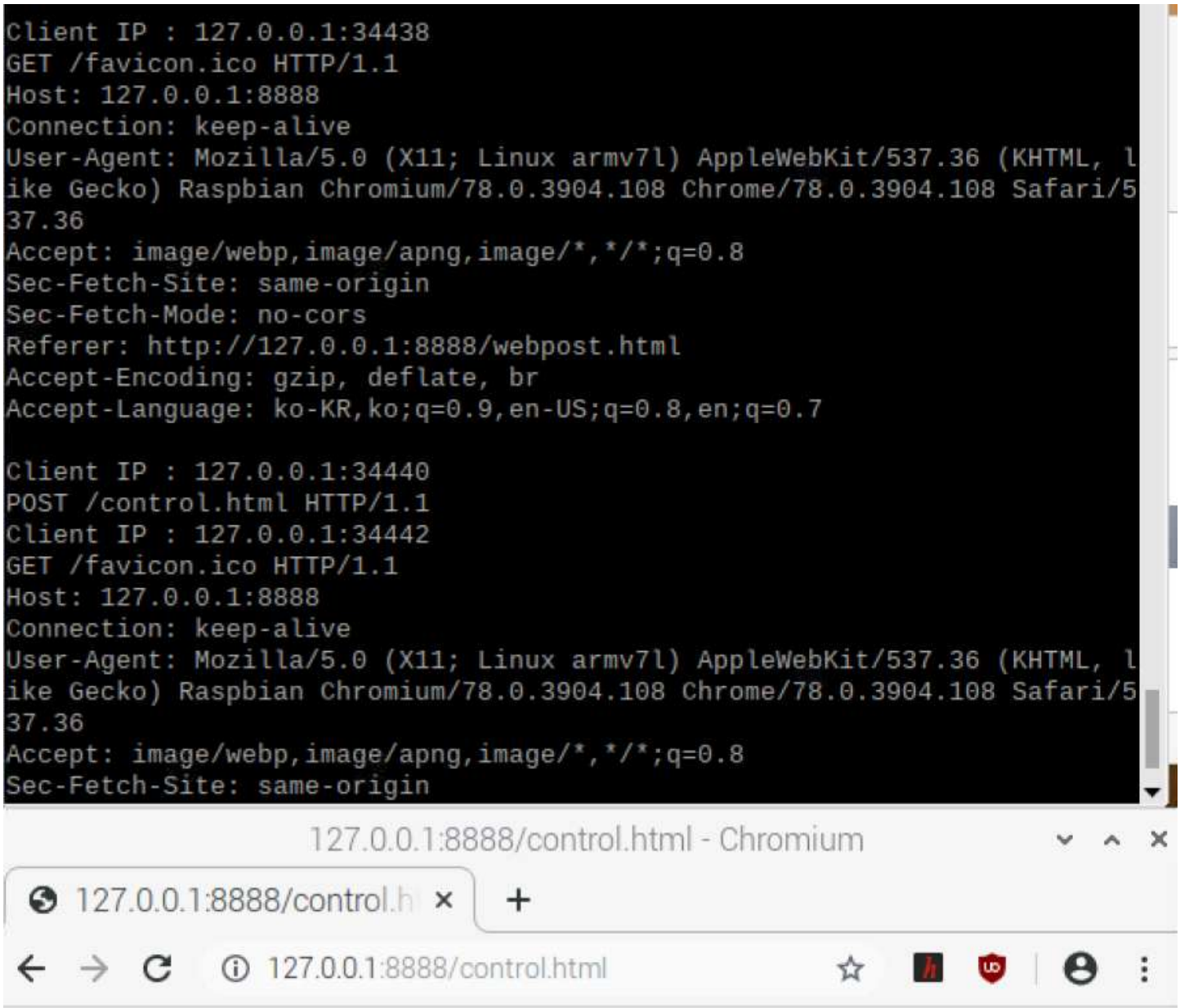
4.1 GET 방식으로 value-key 한 쌍만 전송

```
Client IP : 127.0.0.1:38768
GET /control.html?led=on HTTP/1.1
led = on
Host: 127.0.0.1:9999
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, l
ike Gecko) Raspbian Chromium/78.0.3904.108 Chrome/78.0.3904.108 Safari/5
37.36
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Referer: http://127.0.0.1:9999/control.html
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
```

4.2 GET 방식으로 value-key 여러 쌍 전송

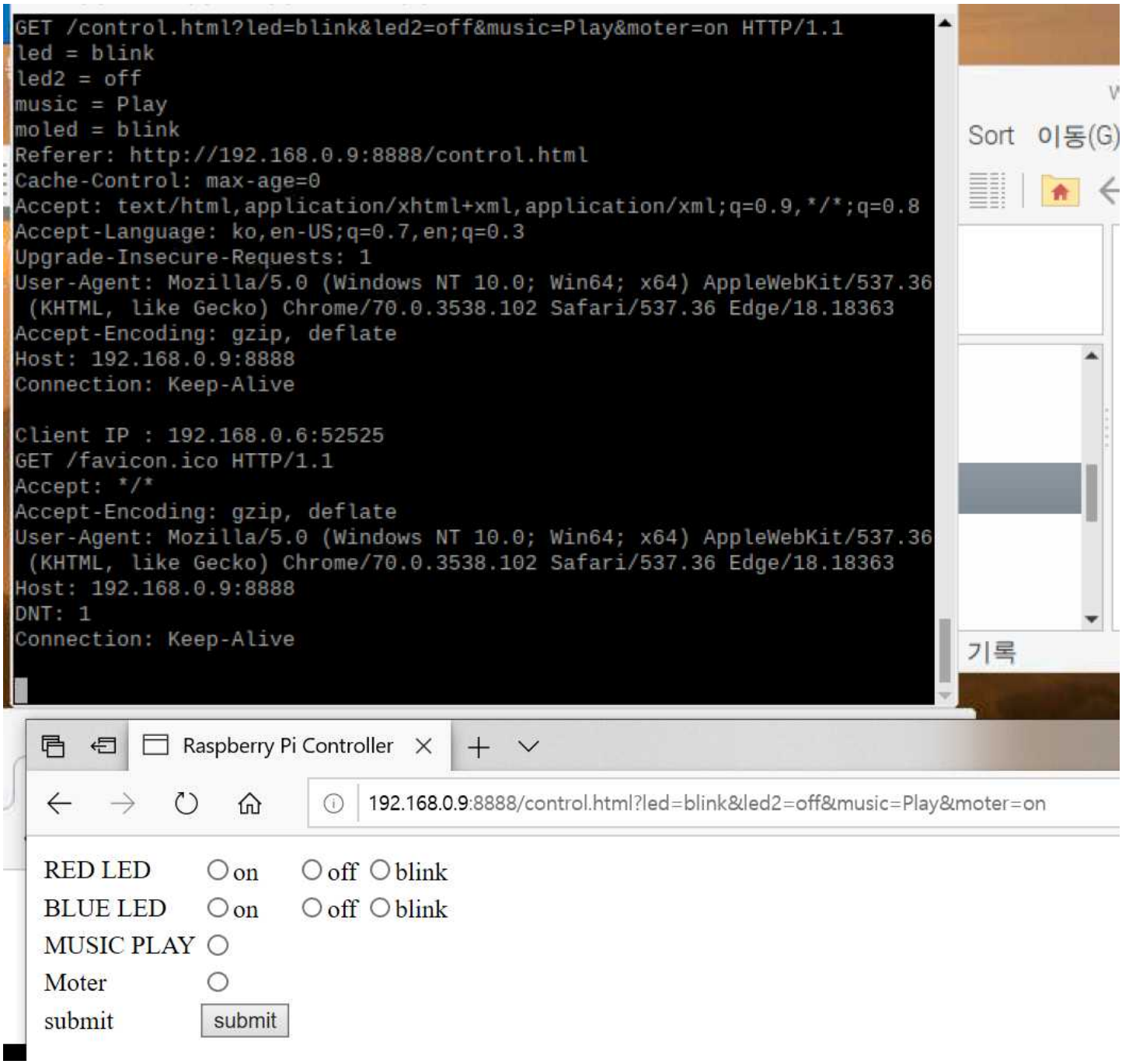
```
Client IP : 127.0.0.1:38772
GET /control.html?led=on&led2=off&music=Play&moter=on HTTP/1.1
led = on
led2 = off
music = Play
moterled = on
Host: 127.0.0.1:9999
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML,
ike Gecko) Raspbian Chromium/78.0.3904.108 Chrome/78.0.3904.108 Safari/5
37.36
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Referer: http://127.0.0.1:9999/control.html?led=on
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
```

4.3 POST 방식으로 값 전송



POST 방식인 webpost.html로 접속한 장면입니다.
헤더에 아무 값도 들어가지 않아 파싱된 값이 나오지 않습니다.

4.3 외부 브라우저에서 접속했을 때의 화면



참고자료 및 참고사이트

HTTP
https://ko.wikipedia.org/wiki/HTTP

HTTP 메서드
https://www.zerocho.com/category/HTTP/post/5b3723477b58fc001b8f6385

외 수업자료