tcp_client.c

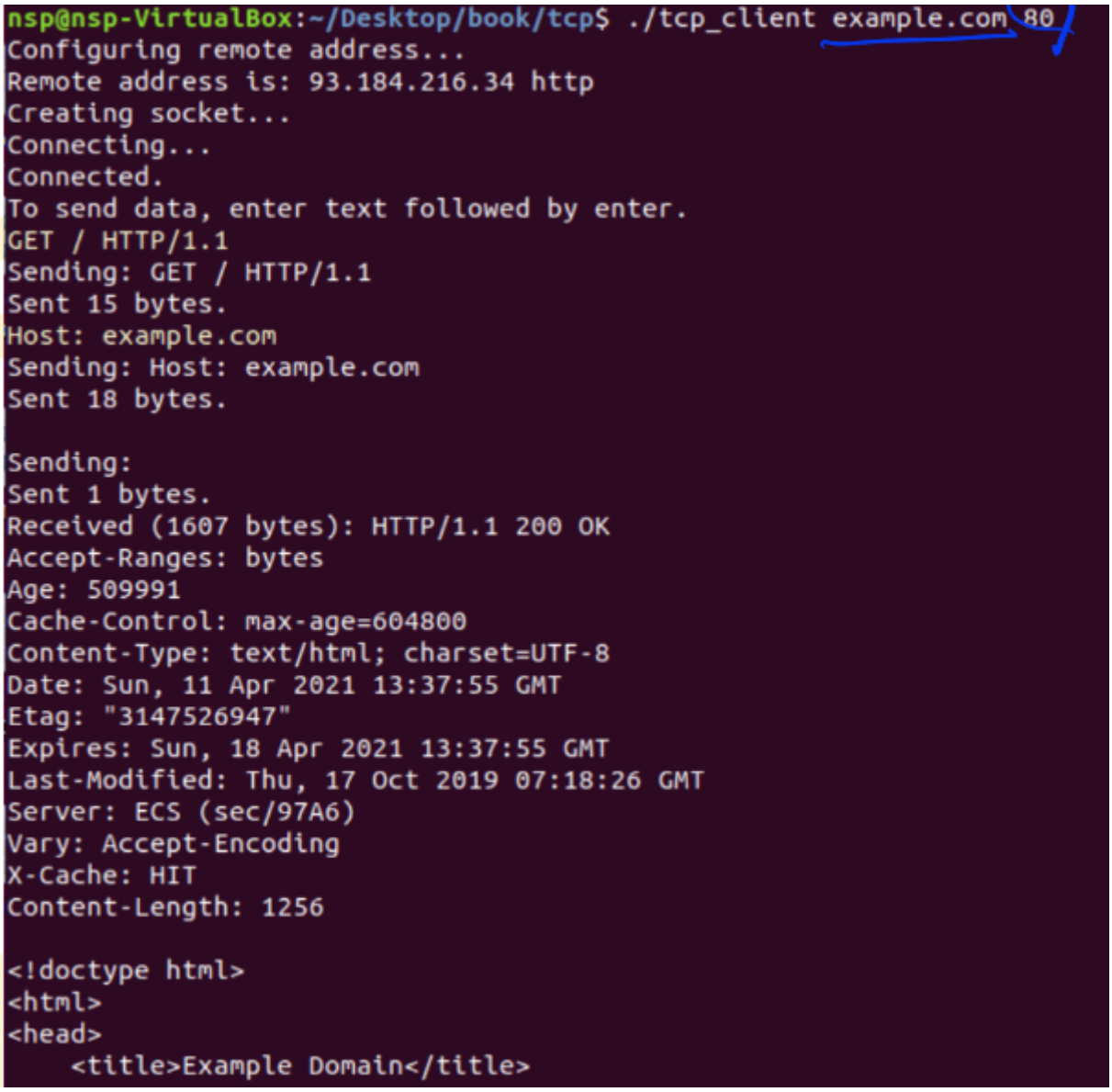
```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

#define ISVALIDSOCKET(s) ((s) >= 0)
#define CLOSESOCKET(s) close(s)
#define SOCKET int
#define GETSOCKETERRNO() (errno)

int main(int argc, char *argv[]) {


    if (argc < 3) {
        fprintf(stderr, "usage: tcp_client hostname port\n");
        return 1;
    }

    printf("Configuring remote address...\n");
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_socktype = SOCK_STREAM;
    struct addrinfo *peer_address;

    //hostname, port를 통해 도메인의 IP를 알아내 hints 형태로 peer_address에 저장
    if (getaddrinfo(argv[1], argv[2], &hints, &peer_address)) {
        fprintf(stderr, "getaddrinfo() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }


    printf("Remote address is: ");
    char address_buffer[100];
    char service_buffer[100];
    //IP로 도메인 이름 찾기
    getnameinfo(peer_address->ai_addr, peer_address->ai_addrlen,
            address_buffer, sizeof(address_buffer),
            service_buffer, sizeof(service_buffer),
            NI_NUMERICHOST);
    printf("%s %s\n", address_buffer, service_buffer);


    printf("Creating socket...\n");
    SOCKET socket_peer;
```

```
    socket_peer = socket(peer_address->ai_family,
            peer_address->ai_socktype, peer_address->ai_protocol);
    //소켓 잘 만들어졌는지 확인
    if (!ISVALIDSOCKET(socket_peer)) {
        fprintf(stderr, "socket() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }


    printf("Connecting...\n");
    if (connect(socket_peer,
                peer_address->ai_addr, peer_address->ai_addrlen)) {
        fprintf(stderr, "connect() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }
    freeaddrinfo(peer_address);

    printf("Connected.\n");
    printf("To send data, enter text followed by enter.\n");

    while(1) {

        fd_set reads;
        FD_ZERO(&reads);
        //서버소켓
        FD_SET(socket_peer, &reads);
        //stdin을 관심있게 설정
        FD_SET(0, &reads);

        //timeout 설정
        struct timeval timeout;
        timeout.tv_sec = 0;
        timeout.tv_usec = 100000;

        //관심있는 소켓 관찰
        if (select(socket_peer+1, &reads, 0, 0, &timeout) < 0) {
            fprintf(stderr, "select() failed. (%d)\n", GETSOCKETERRNO());
            return 1;
        }

        //서버소켓으로부터 요청 오면
        if (FD_ISSET(socket_peer, &reads)) {
            char read[4096];
            //데이터 받기
            int bytes_received = recv(socket_peer, read, 4096, 0);
            if (bytes_received < 1) {
                printf("Connection closed by peer.\n");
                break;
            }
            printf("Received (%d bytes): %.*s",
                    bytes_received, bytes_received, read);
        }

        //stdin의 요청이면
```

```
        if(FD_ISSET(0, &reads)) {
            char read[4096];
            if (!fgets(read, 4096, stdin)) break;
            printf("Sending: %s", read);
            //사용자 입력 받아서 보내기
            int bytes_sent = send(socket_peer, read, strlen(read), 0);
            printf("Sent %d bytes.\n", bytes_sent);
        }
    } //end while(1)

    printf("Closing socket...\n");
    CLOSESOCKET(socket_peer);

    printf("Finished.\n");
    return 0;
}
```

- getaddrinfo() 호스트 주소(도메인)을 기반으로 호스트의 IP를 받아온다.
- getnameinfo() 호스트의 IP를 기반으로 호스트 주소(도메인)을 받아온다.

서버로부터 요청이 올 때 / 사용자로부터 입력이 올 때를 구분하여 사용한다.



특정 도메인에 HTTP Request를 보낸 후 HTTP code Resopnse를 받음

tcp_server_toupper.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define ISVALIDSOCKET(s) ((s) >= 0)
#define CLOSESOCKET(s) close(s)
#define SOCKET int
#define GETSOCKETERRNO() (errno)
```

```
int main() {

    printf("Configuring local address...\n");
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    //IPv4, TCP
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    struct addrinfo *bind_address;
    //8080 포트에 설정하기
    getaddrinfo(0, "8080", &hints, &bind_address);


    printf("Creating socket...\n");
    SOCKET socket_listen;
    //소켓 생성
    socket_listen = socket(bind_address->ai_family,
            bind_address->ai_socktype, bind_address->ai_protocol);

    if (!ISVALIDSOCKET(socket_listen)) {
        fprintf(stderr, "socket() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }


    printf("Binding socket to local address...\n");
    //포트번호 할당
    if (bind(socket_listen,
                bind_address->ai_addr, bind_address->ai_addrlen)) {
        fprintf(stderr, "bind() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }
    freeaddrinfo(bind_address);


    printf("Listening...\n");
    //듣기
    if (listen(socket_listen, 10) < 0) {
        fprintf(stderr, "listen() failed. (%d)\n", GETSOCKETERRNO());
        return 1;
    }

    fd_set master;
    FD_ZERO(&master);
    //listen socket 관심 설정
    FD_SET(socket_listen, &master);
    SOCKET max_socket = socket_listen;

    printf("Waiting for connections...\n");

    while(1) {
```

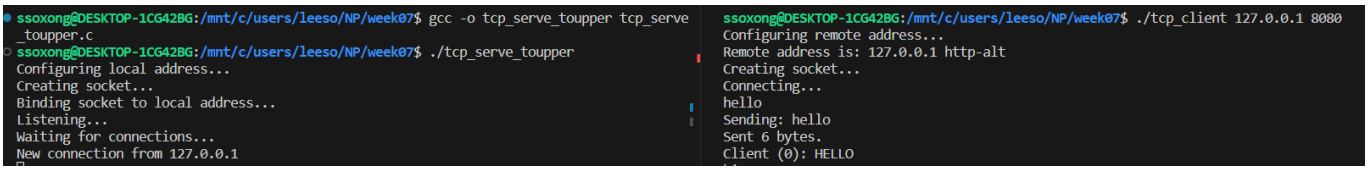
```
        //원본 복사
        fd_set reads;
        reads = master;
        //소켓 관찰
        if (select(max_socket+1, &reads, 0, 0, 0) < 0) {
            fprintf(stderr, "select() failed. (%d)\n", GETSOCKETERRNO());
            return 1;
        }

        SOCKET i;
        for(i = 1; i <= max_socket; ++i) {
            //요청 발생
            if (FD_ISSET(i, &reads)) {
                //듣기 소켓이면 -> 새 client 연결 요청
                if (i == socket_listen) {
                    struct sockaddr_storage client_address;
                    socklen_t client_len = sizeof(client_address);

                    //클라이언트와 연결
                    SOCKET socket_client = accept(socket_listen,
                            (struct sockaddr*) &client_address,
                            &client_len);
                    if (!ISVALIDSOCKET(socket_client)) {
                        fprintf(stderr, "accept() failed. (%d)\n",
                                GETSOCKETERRNO());
                        return 1;
                    }

                    //해당 client를 관심있게 설정
                    FD_SET(socket_client, &master);
                    if (socket_client > max_socket)
                        max_socket = socket_client;

                    char address_buffer[100];
                    //client의 ip 얻기
                    getnameinfo((struct sockaddr*)&client_address,
                            client_len,
                            address_buffer, sizeof(address_buffer), 0, 0,
                            NI_NUMERICHOST);
                    printf("New connection from %s\n", address_buffer);

                }
                //듣기 소켓이 아니라면
                //기존 연결한 client 중에서 요청이 온 것
                else {
                    char read[1024];
                    //받기
                    int bytes_received = recv(i, read, 1024, 0);

                    //이상 응답시 해당 소켓 관심대상에서 제외 (연결 끊기)
                    if (bytes_received < 1) {
                        FD_CLR(i, &master);
                        CLOSESOCKET(i);
                        continue;
```

```
                    }

                    int j;
                    //받은 메세지 대문자로 변환 후 보내기
                    for (j = 0; j < bytes_received; ++j)
                        read[j] = toupper(read[j]);
                    send(i, read, bytes_received, 0);
                }

            } //if FD_ISSET
        } //for i to max_socket
    } //while(1)



    printf("Closing listening socket...\n");
    CLOSESOCKET(socket_listen);

    printf("Finished.\n");

    return 0;
}
```

- socket_listen 듣기 소켓 설정해서 클라이언트 연결 요청 받기

- socket_listen 외의 소켓 연결된 client들을 의미하며, 해당 소켓에서 요청이 오는 것은 메세지를 전송했다는 뜻.

--> select를 이용해 다중통신 구현 가능



tcp_server_chat.c

```
else {
    char read[1024];
    int bytes_received = recv(i, read, 1024, 0);
    if (bytes_received < 1) {
        FD_CLR(i, &master);
        CLOSESOCKET(i);
        continue;
    }

    SOCKET j;
```

```
        for (j = 1; j <= max_socket; ++j) {
            //관심있는 모든 소켓에 대하여
            if (FD_ISSET(j, &master)) {
                //듣기소켓이거나 메세지 보낸 클라이언트는 제외
                if (j == socket_listen || j == i)
                    continue;
                else
                    send(j, read, bytes_received, 0);
            }
        }
    }
```

toupper 부분에서 기존 연결 클라이언트의 요청을 위의 코드로 변경시 클라이언트로부터 온 메세지가 연결중인 다른 클라이언트에게 전달된다.