**UDP 서버-클라이언트 네트워크 통신 학습**컴퓨터네트워크 02분반

컴퓨터공학과

2142851 김형준

**목차**

1. **UDP server-client 네트워크 통신 소스코드 분석**
   1. UDPServer.cpp
   2. UDPClient.cpp
2. **UDPServer.cpp, UDPClient.cpp 통신 실행 결과 출력**
   1. Server Capture
   2. Client Capture
3. **파일 송수신**
   1. 몇 가지 함수 분석
   2. 순서도
   3. 코드
   4. 코드 설명
   5. 결과 화면
4. **UDP server-client 네트워크 통신 소스코드 분석**
5. UDPServer.cpp

#include "../Common.h"

// 포트: 9000

// 버퍼 크기: 512

#define SERVERPORT 9000

#define BUFSIZE    512

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

{

    // 소켓 함수 반환값 저장

    int retval;

    // 소켓 생성

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, 0);

    // 유효하지 않은 소켓의 경우 오류 출력 후 프로그램 종료

    if (sock == INVALID\_SOCKET) err\_quit("socket()");

    // bind()

    // 서버 주소 설정 구조체

    struct sockaddr\_in serveraddr;

    // 구조체 0으로 초기화

    memset(&serveraddr, 0, sizeof(serveraddr));

    // AF\_INET: IPv4 사용

    serveraddr.sin\_family = AF\_INET;

    // INADDR\_ANY: 어떤 주소든, 데이터 수락

    serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    // 서버 포트 번호 설정

    serveraddr.sin\_port = htons(SERVERPORT);

    // 소켓을 포트에 바인딩 (bind())

    retval = bind(sock, (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

    // 실패한 경우

    if (retval == SOCKET\_ERROR) err\_quit("bind()");

    // 데이터 통신에 사용할 변수

    // 주소 정보 저장 구조체

    struct sockaddr\_in clientaddr;

    // 주소 길이 저장

    socklen\_t addrlen;

    // 데이터 저장

    char buf[BUFSIZE + 1];

    // 클라이언트와 데이터 통신

    // 수신하는 동안 무한루프

    while (1) {

        // 데이터 받기

        addrlen = sizeof(clientaddr);

        // recvfrom(): 클라이언트로부터 데이터 수신

        retval = recvfrom(sock, buf, BUFSIZE, 0,

            (struct sockaddr \*)&clientaddr, &addrlen);

        // 수신에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("recvfrom()");

            break;

        }

        // 받은 데이터 출력

        buf[retval] = '\0';

        char addr[INET\_ADDRSTRLEN];

        inet\_ntop(AF\_INET, &clientaddr.sin\_addr, addr, sizeof(addr));

        // [UDP/주소:포트] 메시지

        printf("[UDP/%s:%d] %s\n", addr, ntohs(clientaddr.sin\_port), buf);

        // 데이터 보내기

        retval = sendto(sock, buf, retval, 0,

            (struct sockaddr \*)&clientaddr, sizeof(clientaddr));

        // 전송에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("sendto()");

            break;

        }

    }

    // 소켓 닫기

    close(sock);

    return 0;

}

1. UDPClient.cpp

#include "../Common.h"

// IP 주소 저장하는 char 타입 배열

char \*SERVERIP = (char \*)"127.0.0.1";

// 포트: 9000

// 버퍼 크기: 512

#define SERVERPORT 9000

#define BUFSIZE    512

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

{

    // 소켓 함수의 반환값 저장

    int retval;

    // 명령행 인수가 있으면 IP 주소로 사용

    if (argc > 1) SERVERIP = argv[1];

    // 소켓 생성

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, 0);

    // 유효하지 않은 경우

    if (sock == INVALID\_SOCKET) err\_quit("socket()");

    // 소켓 주소 구조체 초기화

    // serveraddr: 서버 주소 정보 설정 구조체

    struct sockaddr\_in serveraddr;

    // memset: 구조체 0으로 초기화

    memset(&serveraddr, 0, sizeof(serveraddr));

    // AF\_INET: IPv4 주소 사용

    serveraddr.sin\_family = AF\_INET;

    // sin\_addr: 서버 IP 주소 설정

    inet\_pton(AF\_INET, SERVERIP, &serveraddr.sin\_addr);

    // sin\_port: 서버 포트 번호 설정

    serveraddr.sin\_port = htons(SERVERPORT);

    // 데이터 통신에 사용할 변수

    // peeraddr: 주소 정보 저장 구조체

    struct sockaddr\_in peeraddr;

    // addrlen: peeraddr의 길이

    socklen\_t addrlen;

    // buf: 데이터 저장

    char buf[BUFSIZE + 1];

    // len: buf의 길이

    int len;

    // 서버와 데이터 통신

    // 서버에 데이터 전송

    while (1) {

        // 데이터 입력

        printf("\n[보낼 데이터] ");

        // 입력 실패 or EOF 도달 시 fgets => NULL

        if (fgets(buf, BUFSIZE + 1, stdin) == NULL)

            break;

        // '\n' 문자 제거

        len = (int)strlen(buf);

        if (buf[len - 1] == '\n')

            buf[len - 1] = '\0';

        // 입력이 없다면

        if (strlen(buf) == 0)

            break;

        // 데이터 보내기

        retval = sendto(sock, buf, (int)strlen(buf), 0,

            (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

        // 전송에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("sendto()");

            break;

        }

        printf("[UDP 클라이언트] %d바이트를 보냈습니다.\n", retval);

        // 데이터 받기

        addrlen = sizeof(peeraddr);

        retval = recvfrom(sock, buf, BUFSIZE, 0,

            (struct sockaddr \*)&peeraddr, &addrlen);

        // 수신에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("recvfrom()");

            break;

        }

        // 송신자의 주소 체크

        if (memcmp(&peeraddr, &serveraddr, sizeof(peeraddr))) {

            printf("[오류] 잘못된 데이터입니다!\n");

            break;

        }

        // 받은 데이터 출력

        buf[retval] = '\0';

        printf("[UDP 클라이언트] %d바이트를 받았습니다.\n", retval);

        printf("[받은 데이터] %s\n", buf);

    }

    // 소켓 닫기

    close(sock);

    return 0;

}

1. **UDPServer.cpp, UDPClient.cpp 통신 실행 결과 출력**
2. UDPServer.cpp  
   텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

   자동 생성된 설명
3. UDPClient.cpp  
   텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

   자동 생성된 설명
4. **파일 송수신**
5. 몇 가지 함수 분석

**sendto()**

retval = sendto(sock, buf, retval, 0, (struct sockaddr \*)&clientaddr, sizeof(clientaddr));

|  |  |
| --- | --- |
| sock | 데이터를 보낼 소켓 |
| buf | 보낼 데이터가 저장된 버퍼 |
| retval | 보낼 데이터의 길이 |
| 0 | flag 값. (기본적으로 0으로 설정 |
| (struct sockaddr \*)&clientaddr | 수신자의 주소 정보를 나타내는 sockaddr 구조체에 대한 포인터 |
| sizeof(clientaddr) | clientaddr 구조체의 크기. (데이터를 보내는데 필요한 크기) |

**fread()**

size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);

|  |  |
| --- | --- |
| ptr | 읽어온 데이터를 저장할 버퍼 포인터 |
| size | 각 데이터 크기 |
| nmemb | 읽어올 데이터 수 |
| stream | 읽어올 파일의 포인터 |

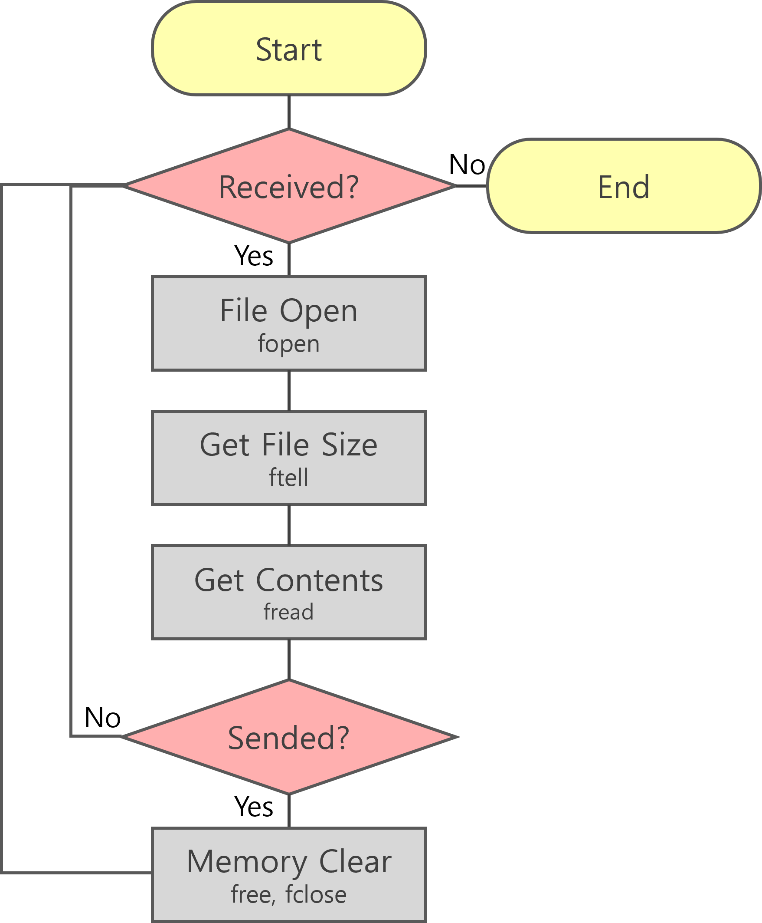
**feof() – 사용하지 않음**

while (!feof(fp))

파일(fp)의 끝에 도달했을 경우, 0이 아닌 값(true)을 반환

파일(fp)의 끝에 도달하지 않았을 경우, 0(false)을 반환

1. 순서도

UDPServer.c의 순서도

1. 코드

UDPServer.c

#include "../Common.h"

// 포트: 9000

// 버퍼 크기: 512

#define SERVERPORT 9000

#define BUFSIZE    512

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

{

    // 소켓 함수 반환값 저장

    int retval;

    // 소켓 생성

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, 0);

    // 유효하지 않은 소켓의 경우 오류 출력 후 프로그램 종료

    if (sock == INVALID\_SOCKET) err\_quit("socket()");

    // bind()

    // 서버 주소 설정 구조체

    struct sockaddr\_in serveraddr;

    // 구조체 0으로 초기화

    memset(&serveraddr, 0, sizeof(serveraddr));

    // AF\_INET: IPv4 사용

    serveraddr.sin\_family = AF\_INET;

    // INADDR\_ANY: 어떤 주소든, 데이터 수락

    serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    // 서버 포트 번호 설정

    serveraddr.sin\_port = htons(SERVERPORT);

    // 소켓을 포트에 바인딩 (bind())

    retval = bind(sock, (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

    // 실패한 경우

    if (retval == SOCKET\_ERROR) err\_quit("bind()");

    // 데이터 통신에 사용할 변수

    // 주소 정보 저장 구조체

    struct sockaddr\_in clientaddr;

    // 주소 길이 저장

    socklen\_t addrlen;

    // 데이터 저장

    char buf[BUFSIZE + 1];

    char fileName[BUFSIZE];

    // 클라이언트와 데이터 통신

    // 수신하는 동안 무한루프

    while (1) {

        // 데이터 받기

        addrlen = sizeof(clientaddr);

        // recvfrom(): 클라이언트로부터 데이터 수신

        retval = recvfrom(sock, buf, BUFSIZE, 0,

            (struct sockaddr \*)&clientaddr, &addrlen);

        // 수신에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("recvfrom()");

            break;

        }

        printf("=================================================\n");

        // 받은 데이터 출력

        buf[retval] = '\0';

        char addr[INET\_ADDRSTRLEN];

        inet\_ntop(AF\_INET, &clientaddr.sin\_addr, addr, sizeof(addr));

        // [UDP/주소:포트] 메시지

        printf("[UDP/%s:%d] %s\n", addr, ntohs(clientaddr.sin\_port), buf);

        // 요청된 파일 이름 추출

        if (sscanf(buf, "request \"%[^\"]\"", fileName) != 1) {

            printf("Format: request \"FILENAME\"\n");

            printf("=================================================\n");

            continue;

        }

        printf("- The server received a request from a client\n");

        printf("FILENAME: %s\n", fileName);

        char \*contents = NULL;

        long fileSize;

        size\_t result;

        // 파일 열기

        FILE \*fp = fopen(fileName, "r");

        if (fp == NULL) {

            err\_display("fopen()");

            break;

        }

        // 파일 크기 계산

        fseek(fp, 0, SEEK\_END);

        fileSize = ftell(fp);

        rewind(fp);

        // 파일 크기에 맞게 메모리 할당

        contents = (char \*)malloc(fileSize \* sizeof(char));

        if (contents == NULL) {

            perror("Failed. (contents == NULL)");

            fclose(fp);

            return 1;

        }

        // 파일 내용 읽기

        result = fread(contents, 1, fileSize, fp);

        if (result != fileSize) {

            perror("Failed. (result != fileSize)");

            free(contents);

            fclose(fp);

            return 1;

        }

        retval = sendto(sock, contents, sizeof(char) \* fileSize, 0, (struct sockaddr \*)&clientaddr, sizeof(clientaddr));

        // 전송에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("sendto()");

            break;

        }

        printf("- The server sent \"%s\" to the client\n", fileName);

        printf("=================================================\n");

        // 메모리와 파일 닫기

        free(contents);

        fclose(fp);

    }

    // 소켓 닫기

    close(sock);

    return 0;

}

UDPClient.c

#include "../Common.h"

// IP 주소 저장하는 char 타입 배열

char \*SERVERIP = (char \*)"127.0.0.1";

// 포트: 9000

// 버퍼 크기: 10000

#define SERVERPORT 9000

#define BUFSIZE    10000

#define FILECOUNT  10

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

{

    // 소켓 함수의 반환값 저장

    int retval;

    // 명령행 인수가 있으면 IP 주소로 사용

    if (argc > 1) SERVERIP = argv[1];

    // 소켓 생성

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, 0);

    // 유효하지 않은 경우

    if (sock == INVALID\_SOCKET) err\_quit("socket()");

    // 소켓 주소 구조체 초기화

    // serveraddr: 서버 주소 정보 설정 구조체

    struct sockaddr\_in serveraddr;

    // memset: 구조체 0으로 초기화

    memset(&serveraddr, 0, sizeof(serveraddr));

    // AF\_INET: IPv4 주소 사용

    serveraddr.sin\_family = AF\_INET;

    // sin\_addr: 서버 IP 주소 설정

    inet\_pton(AF\_INET, SERVERIP, &serveraddr.sin\_addr);

    // sin\_port: 서버 포트 번호 설정

    serveraddr.sin\_port = htons(SERVERPORT);

    // 데이터 통신에 사용할 변수

    // peeraddr: 주소 정보 저장 구조체

    struct sockaddr\_in peeraddr;

    // addrlen: peeraddr의 길이

    socklen\_t addrlen;

    // buf: 데이터 저장

    char buf[BUFSIZE + 1];

    // len: buf의 길이

    int len;

    // File List

    char fileNames[FILECOUNT][30] = {"novel.txt", "anthem.txt", "computer\_network.txt"};

    // 서버와 데이터 통신

    // 서버에 데이터 전송

    while (1) {

        // 데이터 입력

        printf("\n[보낼 데이터] (request \"FILENAME\")\n");

        printf("<File List>\n");

        for(int i = 0; i < FILECOUNT; i++) {

            if (fileNames[i][0] == '\0') continue;

            printf("- %s\n", fileNames[i]);

        }

        printf("=> ");

        // 입력 실패 or EOF 도달 시 fgets => NULL

        if (fgets(buf, BUFSIZE + 1, stdin) == NULL)

            break;

        // '\n' 문자 제거

        len = (int)strlen(buf);

        if (buf[len - 1] == '\n')

            buf[len - 1] = '\0';

        // 입력이 없다면

        if (strlen(buf) == 0)

            break;

        // 데이터 보내기

        retval = sendto(sock, buf, (int)strlen(buf), 0,

            (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

        // 전송에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("sendto()");

            break;

        }

        printf("[UDP 클라이언트] %d바이트를 보냈습니다.\n", retval);

        char fileName[BUFSIZE];

        if (sscanf(buf, "request \"%[^\"]\"", fileName) != 1) {

            printf("Format: request \"FILENAME\"\n");

            continue;

        }

        printf("FILENAME: %s\n", fileName);

        // 데이터 받기

        addrlen = sizeof(peeraddr);

        retval = recvfrom(sock, buf, BUFSIZE, 0,

            (struct sockaddr \*)&peeraddr, &addrlen);

        // 수신에 실패한 경우

        if (retval == SOCKET\_ERROR) {

            err\_display("recvfrom()");

            break;

        }

        // 송신자의 주소 체크

        if (memcmp(&peeraddr, &serveraddr, sizeof(peeraddr))) {

            printf("[오류] 잘못된 데이터입니다!\n");

            break;

        }

        // 받은 데이터 출력

        buf[retval] = '\0';

        printf("[UDP 클라이언트] %d바이트를 받았습니다.\n", retval);

        printf("[받은 데이터]\n");

        printf("%s\n", buf);

        printf("\nThe client received \"%s\" from the server.\n", fileName);

    }

    // 소켓 닫기

    close(sock);

    return 0;

}

1. 코드 설명

- **파일 이름 받기**

char fileName[BUFSIZE];

파일 이름은 BUFSIZE보다 “request ”의 길이보다 작으므로, 대략 BUFSIZE만큼의 배열을 생성하였습니다.

        if (sscanf(buf, "request \"%[^\"]\"", fileName) != 1) {

            printf("Format: request \"FILENAME\"\n");

            printf("=================================================\n");

            continue;

        }

sscanf를 사용하여, 큰 따옴표 사이에 낀 부분을 fileName에 저장하였습니다.

만약 sscanf의 반환 값이 1이 아니라면, format을 지키지 않았다고 판단하였습니다.

continue를 통해 while문 처음으로 돌아가, 다시 입력을 받을 준비를 합니다.

**- 파일 내용 받기**

        char \*contents = NULL;

        long fileSize;

        size\_t result;

contents: 파일의 내용을 담을 char 타입의 포인터 변수

fileSize: 파일의 크기(길이)를 계산할 함수

result: fread의 반환 값을 저장할 변수

        // 파일 열기

        FILE \*fp = fopen(fileName, "r");

        if (fp == NULL) {

            err\_display("fopen()");

            break;

        }

fopen 함수를 이용하여 파일을 엽니다.

만약, fopen을 사용하여 받은 값이 NULL이라면 err를 출력하고, break하여 중지합니다.

        // 파일 크기 계산

        fseek(fp, 0, SEEK\_END);

        fileSize = ftell(fp);

        rewind(fp);

contents 변수에 내용을 입력 받기 위해 realloc을 사용하려 했으나, realloc을 많이 사용하는 코드가 작성되었습니다.

위 방법으로 코딩할 경우, 반복적으로 호출하게 되어 메모리를 재할당하는 작업이 빈번하게 일어나게 되므로 효율이 떨어집니다. 이는 성능에 영향을 끼칠 수 있다고 판단하여, 코드를 수정하였습니다.

파일의 크기를 계산한 후 그 크기에 맞는 동적 할당을 시키는 방법을 선택하였습니다.

파일의 크기를 계산하며, fp 포인터가 파일의 끝으로 이동하였기에, rewind를 이용하여 fp가 가리키는 부분을 첫 부분으로 되돌렸습니다.

        // 파일 크기에 맞게 메모리 할당

        contents = (char \*)malloc(fileSize \* sizeof(char));

        if (contents == NULL) {

            perror("Failed. (contents == NULL)");

            fclose(fp);

            return 1;

        }

이젠, realloc을 사용할 필요가 없으니 1번 malloc 함수를 사용하여 메모리를 할당하였습니다.

        // 파일 내용 읽기

        result = fread(contents, 1, fileSize, fp);

        if (result != fileSize) {

            perror("Failed. (result != fileSize)");

            free(contents);

            fclose(fp);

            return 1;

        }

fread를 통해, contents에 파일의 내용을 저장하였습니다.

**- 메모리 초기화**

        // 메모리와 파일 닫기

        free(contents);

        fclose(fp);

**- UCPClient.c의 변경 내용**

    // File List

    char fileNames[FILECOUNT][30] = {"novel.txt", "anthem.txt", "computer\_network.txt"};

무슨 파일이 있는지 확인할 수 있도록, fileNames 변수를 생성하였습니다.

        // 데이터 입력

        printf("\n[보낼 데이터] (request \"FILENAME\")\n");

        printf("<File List>\n");

        for(int i = 0; i < FILECOUNT; i++) {

            if (fileNames[i][0] == '\0') continue;

            printf("- %s\n", fileNames[i]);

        }

        printf("=> ");

입력 형식과 파일의 목록을 보여주는 부분을 추가하였습니다.

char fileName[BUFSIZE];

        if (sscanf(buf, "request \"%[^\"]\"", fileName) != 1) {

            printf("Format: request \"FILENAME\"\n");

            continue;

        }

형식이 맞지 않을 경우 UDPServer.c와 같은 방식으로 코딩하였습니다.

1. 결과 화면

**- 첫 실행 시**

UDPClient

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명UDPServer

스크린샷, 텍스트, 보라색, 라일락이(가) 표시된 사진

자동 생성된 설명

**- 잘못 기입한 경우**

UDPClient

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

UDPServer

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**- 잘 입력한 경우**

UDPClient

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

UDPServer

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **추가**