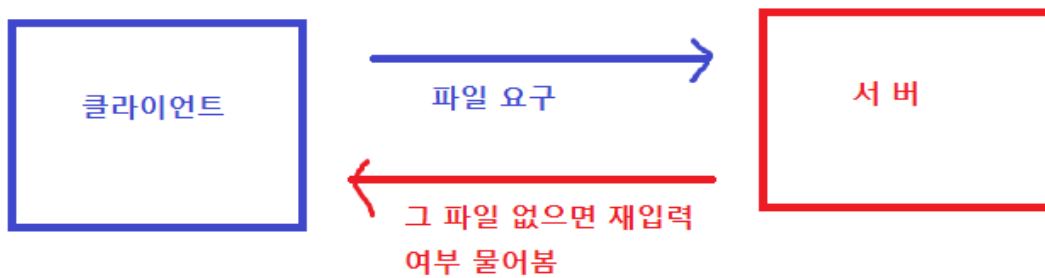


유닉스 프로그래밍 개인과제 11번

201619460 이성규

이 보고서에서는 제출한 hw11s.c 파일을 서버라고 칭하고, hw11c.c 파일을 클라이언트라고 칭하겠습니다.
전체 실행 결과 내용은 마지막에 넣도록 하겠습니다.

일단 코드의 간략한 작동 방식은 다음과 같습니다.



클라이언트는 명령행 인자를 통해, 서버에게 전송 받을 파일을 요구하고, 서버에서는 그 파일이 있는지 여부를 판단하여 파일이 없을시, 다른 파일 이름을 재입력 할 것인지, 그냥 전송을 받지 않을 것인지 물어봅니다. 이는 서버가 갖고 있는 파일 이름이 요구되거나, 전송받기를 거부할 때까지 계속 반복하여 물어봅니다.

코드와 실행결과를 보며 어떻게 수행이 되는지 확인하겠습니다.

먼저 서버 코드를 보겠습니다.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8
9  #define BUFFSIZE 4096
10 #define SERVERPORT 7799
11
12 int main(void) {
13     int s_sock, c_sock;
14     struct sockaddr_in server_addr, client_addr;
15     socklen_t c_addr_size;
16     char buf[BUFFSIZE] = {0};
17     FILE *rfp;
18
19     s_sock = socket(AF_INET, SOCK_STREAM, 0); // 소켓열기
20
21     int option = 1; //프로그램 종료후 다시 실행시켰을때, 오류 무시하기위해 사용
22     setsockopt(s_sock, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option));
23
24     bzero(&server_addr, sizeof(server_addr)); // server_addr 초기화
25
26     server_addr.sin_family = AF_INET;
27     server_addr.sin_port = htons(SERVERPORT);
28     server_addr.sin_addr.s_addr = htonl(INADDR_ANY); // 동작하는 컴퓨터의 IP주소 자동 할당!
29
30     if (bind(s_sock, (struct sockaddr*) &server_addr, sizeof(server_addr)) == -1) {
31         perror("[S] Can't bind a socket"); // 바인드 안된경우 예외처리
32         exit(1);
33     }
```

서버 코드

← 교수님께서 수업에서 사용하신 포트번호와 버퍼 사이즈를 사용하였습니다.

강의자료의 코드와 차이가 크지 않습니다.

위 페이지의 28라인에서, INADDR_ANY를 통해 동작하는 컴퓨터의 IP주소를 자동으로 할당받게 하였습니다. define으로 INADDR_ANY는 상수 0입니다. 동작 기기의 IP주소 할당뿐 아니라 컴퓨터 내에 두 개 이상의 IP를 할당 받아서 사용하는 경우 할당 받은 IP중 어떤 주소를 통해서 데이터가 들어오더라도 PORT번호만 일치하면 수신할 수 있게 해주어 이식성 증대의 의미가 있다고 알고 있습니다.

참고 자료 : <https://mintnlatte.tistory.com/555>

<https://m.blog.naver.com/PostView.nhn?blogId=wlsgh31&logNo=220071545203&proxyReferer=https:%2F%2Fwww.google.com%2F>

```
34
35     printf("[S] I'm waiting for client's connect! My port is %d.\n",SERVERPORT);
36
37     listen(s_sock,1); //1명의 요청 받는중
38
```

서버 코드

그 이후 강의 자료와 동일히 클라이언트를 대기 합니다. 35라인에서 클라이언트를 대기하고 있다고 출력하고 있고, 포트번호도 7799로 같이 출력하고 있습니다.

```
ubuntu@201619460:~/hw11$ gcc -o hw11s hw11s.c
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.
```

(서버의 실행창)

여기까지 서버 코드를 보고 잠시 클라이언트 코드로 가겠습니다.

```
9  #define BUFFSIZE 4096
10
11 int main(int argc, char* argv[]) {
12
13     if(argc != 4) {
14         printf("< Usage: %s IP_to_access PortNumber file >\n", argv[0]);
15         return 1;
16     }
17
18     int c_sock;
19     struct sockaddr_in server_addr, client_addr;
20     socklen_t c_addr_size;
21     char buf[BUFFSIZE] = {0};
22     char saveFname[10];
23     FILE *wfp;
24
```

클라이언트 코드

클라이언트 코드의 #include는 강의 자료와 동일해서 부피를 줄이고자 잘라냈습니다. 버프사이즈는 서버와 동일하고,

13라인~16라인에 인자 수에 따른 예외 처리를 해주었습니다.

13~16 라인으로 인해 명령행 인자 수가 맞지 않으면 실행이 되지 않습니다.

```
ubuntu@201619460:~/hw11$ gcc -o hw11c hw11c.c
ubuntu@201619460:~/hw11$ ./hw11c
< Usage: ./hw11c IP_to_access PortNumber file >
ubuntu@201619460:~/hw11$ ./hw11c aaaa
< Usage: ./hw11c IP_to_access PortNumber file >
ubuntu@201619460:~/hw11$
```

클라이언트의 실행창

```

25 c_sock = socket(AF_INET, SOCK_STREAM, 0); //소켓열기
26
27 bzero(&server_addr, sizeof(server_addr)); // server_addr 초기화
28
29 server_addr.sin_family = AF_INET;
30 server_addr.sin_port = htons(atoi(argv[2])); // 입력받은 인자로 포트번호 입력 (check)
31 server_addr.sin_addr.s_addr = inet_addr(argv[1]); // 입력받은 인자로 ip 입력 (check)
32
33
34 if (connect(c_sock, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1) {
35     perror("[C] Can't connect to a Server"); // 커넥트 예외처리
36     exit(1);
37 }

```

클라이언트 코드

그럼 인자를 맞추어서 클라이언트 쪽에서 ./hw11c 10.0.0.134 7799 aaa.txt 로 접속을 시도하겠습니다. 각각 제 ip주소, 위에 서버 실행창에 나온 포트번호, 서버에 없는 파일명입니다.

```

ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 aaa.txt
[C] Server dont't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :

```

클라이언트의 실행창

그러면 클라이언트 실행창 에서 해당 파일이 서버에 없다고 다시 입력할건지 물어봅니다.

```

ubuntu@201619460:~/hw11$ gcc -o hw11s hw11s.c
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.

```

(서버의 실행창)

```

ubuntu@201619460:~/hw11$ gcc -o hw11s hw11s.c
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.
[S] Connected: client IP addr=10.0.0.134 port=51688
[S] Client request file name : aaa.txt
[S] I don't have file
[S] I'm waiting for client's reply!

```

서버 실행창

서버의 실행창에는 컨넥트된 클라이언트의 IP주소와 포트를 출력하고,
(Connected: client IP addr=10.0.0.134 port=51668)

클라이언트가 요구하는 파일이름과 그 파일을 서버가 갖고있는지 출력합니다.
(Client request file name : aaa.txt)
(I don't have file)

그리고 클라이언트에서 파일을 새로입력할지, 그냥 파일 전송을 안 받을지 응답을 기다리고 있다고 출력이 나옵니다.
(I'm wating for client's reply!)

이 실행창의 결과를 코드로 살펴보겠습니다.

```

39
40 strcpy(buf,argv[3]); // 다운 받을 파일 이름을 while 문돌기전에도 제대로 입력받기위해 옮김
41
42 //1. 전송 받을 파일이름 sever에 전달
43 if (send(c_sock ,argv[3], sizeof(argv[3]), 0) == -1) {
44     perror("[C] Can't send file name\n");
45     exit(1);
46 }
47
48 saveFname[0] = 'C';
49

```

클라이언트 코드

일단 이전 페이지 까지 서버와 클라이언트가 컨넥트가 되었고, 클라이언트에서 메시지를 전달합니다.
 지금 과정이 전 페이지의 aaa.txt 명령행 인자 문자열(위 코드에서 43라인의 argv[3])을 서버에 보내는 과정입니다.

그리고 40라인의 코드와 48라인의 코드를 약간 이야기하자면,
 saveFname 문자열은 saving File name의 뜻으로, Server의 파일을 클라이언트에서 같은 공간에 저장하게 될 시 이름이 같으면 안되니까, Server의 파일이름 맨앞에 C만 붙인 꼴로 복사하여 저장하고자 만든 문자열입니다. 이 문자열은 while문이 시작될 때 buf의 내용으로 갱신이 되는데, 그 에 맞게 첫 번째 루프에서도 적용이 될 수 있게 48라인에서 buf로 받은 명령행 인자를 옮겨주는 것입니다.

아무튼 클라이언트 코드가 전송받을 파일 이름(이 실행에선 aaa.txt)을 server에 전달합니다.

```

49
50 //2. 클라이언트에게 파일 이름 받아옴.
51 if (recv(c_sock, buf, BUFFSIZE, 0) == -1) {
52     perror("[S] Can't receive file name.\n");
53     exit(1);
54 }
55
56 int request = 0; // 파일 이름 재전송위한 인수
--

```

(서버코드)

그러면 서버에서 클라이언트에게서 메시지를 받습니다.
 파일 이름을 메시지로써 받게 되고,
 서버 코드는 while문으로 진입합니다.

<p>그 후에 서버에서는</p> <p>그 파일이름을 자신이 갖고 있는지 확인하고</p> <p>갖고 있다면 바로 while문을 탈출해서 파일을 보낼 준비를 하고, 클라이언트에게 파일이 있다는 메시지를 보내게 됩니다.</p> <p>(72라인 : 주석 3.2에 해당)</p> <p>파일이 있다는 메시지는 “Y”입니다.</p> <p>반대로 파일이 없다면 일단 클라이언트에서 새로운 파일 이름을 줄 건지, 아니면 그냥 전송을 취소할건지 모르니까 일단 파일이 없다고 request 변수를 통해 입력해둔 뒤, 클라이언트에게 파일이 없다는 신호를 보내게 됩니다. (71라인 : 주석 3.1에 해당)</p> <p>파일이 없다는 메시지는 “N”입니다.</p>	<pre> 57 58 while(1){ 59 60 printf("[S] Client request file name : %s\n", buf); 61 62 //3. 파일을 갖고있는지 여부 메세지 전송 (서버코드) 63 rfp = fopen(buf,"r"); 64 if (rfp == NULL) { 65 printf("[S] I don't have file\n"); 66 67 if (send(c_sock, "N",sizeof("N"), 0) == -1){ 68 perror("[S] Can't send message that I have file.\n"); 69 exit(1); 70 } 71 request = -1; // 3.1 파일 없다고 표시하고 메세지 보냄 72 }else{ //3.2 파일이 있다는 메세지 보냄 73 74 printf("[S] I have file : %s\n",buf); 75 76 if (send(c_sock, "Y",sizeof("Y"), 0) == -1){ 77 perror("[S] Can't send message that I don't have file.\n"); 78 exit(1); 79 } 80 break; 81 } 82 83 printf("[S] I'm waiting for client's reply!\n"); 84 </pre>
--	--

서버는 파일이 있는 경우 Y의 메시지를 보낸 후 파일을 전송하지만,
파일이 없는 경우 클라이언트에게 N의 메시지를 보낸 후 예는 클라이언트의 응답을 대기하게 됩니다.

이제 클라이언트 코드로 와보겠습니다. 클라이언트 코드도 메시지를 보낸 후 while문에 진입해 있습니다.

```
50 while(1){
51     strncpy(saveFname+1,buf,9);
52     saveFname[9] = '\0'; // buf에 saveFname 배열보다 더 많은 문자있을때 예외처리
53
54     //4. 보낸 파일이름이 서버가 갖고있는지 응답받음
55     if (recv(c_sock, buf, BUFSIZE, 0) == -1) {
56         perror("[C] Can't receive message about having file.\n");
57         exit(1);
58     }
59
60     //4.1 파일 없다 응답 받음
61     if (buf[0] == 'N') //서버로 부터 응답으로 그 파일이 없음을 나타냄.문자열비교 함수를 쓸 수 있으나, 코드가 길어져서 버퍼0번항목만 비교함.(strcmp)
62     {
63         printf("[C] Server don't have file.\n");
64         printf("[C] 다른 파일이름을 입력하시겠습니까? Y/N : \n");
65         int check = -1;
66
67         while(check == -1){
68             scanf("%c",&buf[0]);
69             if(buf[0] != 'Y' && buf[0] != 'N'){
70                 printf("[C] 응답은 Y또는 N 한글자로만 해주십시오.\n");
71             }else{
72                 check = 0;
73             }
74         }
75     }else if(buf[0] == 'Y') //4.2 파일이 있다 응답받음
76     {
77         printf("[C] Server have file.\n");
78         break;
79     }else{
80         perror("[C] 코드상으로 잘못된 정보가 들어왔습니다.\n");
81         exit(1);
82     }
```

그러면 클라이언트 측에서는 서버에서 보낸 메시지를 받습니다.(라인 54 : 주석 4.) 그리고 그 받은 메시지가 파일이 있다는 응답("Y") 이면, 바로 break를 통해 while문을 빠져나가 파일을 받을 준비를 하게 되고,(라인 75 : 주석 4.2) 없다는 응답("N") 이면 사용자에게 다른 파일 이름을 입력할건지 묻게 됩니다. (라인 60 : 주석 4.1) 없다는 응답에서는 check 변수가 사용자의 응답으로 Y와 N가 들어왔을 경우에만 변경되도록 하여 제대로 된 입력을 할 때까지 반복하여 입력을 받습니다.
이 부분이 지금 클라이언트 실행창의 부분입니다.

```
ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 aaa.txt
[C] Server don't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
```

이 부분에서 Y나 N 말고 다른 문자를 입력하면 제대로 된 문자를 입력하라하고, 다시 입력을 받습니다.

```
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
a
[C] 응답은 Y또는 N 한글자로만 해주십시오.
```

N을 입력하게 된다면 파일이름을 추가로 입력 받지 않고 프로그램이 종료하게 됩니다.

이 경우 서버에서도 응답을 받고 전송하지 않는다는 걸 인지하여 프로그램을 종료하게 됩니다.

```
[C] 응답은 Y또는 N 한글자로만 해주십시오.
N
파일 전송을 하지 않고 종료합니다.
ubuntu@201619460:~/hw11$
```

```
[S] I don't have file
[S] I'm waiting for client's reply!
[S] 파일을 보내지 않고 종료합니다.
ubuntu@201619460:~/hw11$
```

클라이언트의 실행창

서버의 실행창

위 코드에서 사용자가 클라이언트에서 고르는 Y 또는 N의 메시지가 서버에게 전해지게 됩니다.

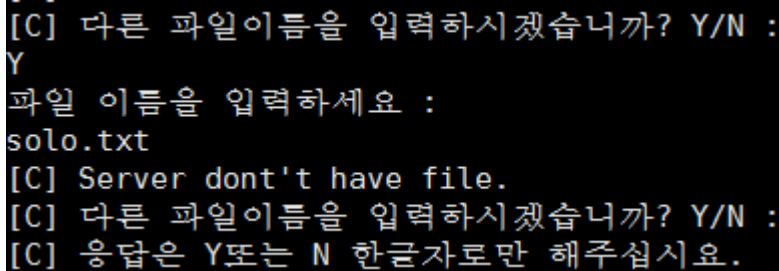
```
83
84     if ( buf[0] == 'N' ){ //4.1.1 파일 안받겠다고 메세지줌
85
86         if (send(c_sock ,buf, BUFSIZE, 0) == -1) {      클라이언트 코드
87             perror("[C] Can't send message");
88             exit(1);
89         }
90
91         printf("파일 전송을 하지 않고 종료합니다.\n");
92         close(c_sock);
93
94         return 0;
95     }else if( buf[0] == 'Y'){//4.1.2 파일 이름 재전송
96         printf("파일 이름을 입력하세요 : \n");
97         scanf("%s",&buf[0]);
98         if (send(c_sock ,buf, BUFSIZE, 0) == -1) {
99             perror("[C] Can't send message");
100             exit(1);
101         }
102     }
103 }
```

사용자가 만약 N. 즉 파일 전송취소를 고를시(라인 84 : 주석 4.1.1)

위에서 보았던 실행창처럼 파일 전송을 하지 않고 서버에 N의 메시지를 준 후 클라이언트가 종료하게 됩니다.

만약 Y. 새로운 파일 이름을 골랐을 시(라인 95 : 주석 4.1.2)

사용자에게 새로운 파일 이름을 입력받으며, 그 받은 새로운 파일 이름값이 서버에게 전달이 되게 됩니다.



```
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
Y
파일 이름을 입력하세요 :
solo.txt
[C] Server dont't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
[C] 응답은 Y또는 N 한글자로만 해주십시오.
```

새로운 파일 이름값을 서버에 전달한 이후, 클라이언트 코드는 while문의 처음으로 돌아가서 서버로 부터 그 이름의 파일이 있는지 응답을 기다리게 됩니다.

```
50     while(1){
51         strncpy(saveFname+1,buf,9);
52         saveFname[9] = '\0'; // buf에 saveFname 배열보다 더 많은 문자있을때 예외처리
53
54         //4. 보낸 파일이름이 서버가 갖고있는지 응답받음
55         if (recv(c_sock, buf, BUFSIZE, 0) == -1) {
56             perror("[C] Can't receive message about having file.\n");
57             exit(1);
58         }
59     }
```

클라이언트
(다시 while 처음으로 돌아옴)

잠깐 이 코드에서, 위 페이지에서 살짝 언급한 파일의 저장이름 saveFname은 새로 입력한 파일이름으로 갱신이 됩니다. 즉 새로운 파일 이름을 입력받을 때마다 while문 처음에서 저장할 이름이 갱신됩니다. (라인 51 ~ 52)

그러면 다시 서버코드로 가보겠습니다.

위에서 언급했듯, 서버코드는 응답을 대기하고 있었습니다.

```
83     printf("[S] I'm waiting for client's reply!\n");
84
85     if (request == -1) //5. 파일이 없어서 응답대기          서버코드
86     {
87         if (recv(c_sock, buf, BUFFSIZE, 0) == -1) {
88             perror("[S] Can't receive massege\n");
89             exit(1);
90         }
91         if(buf[0] == 'N' && buf[1] == '\0') //5.1. 파일 안받음
92         {
93             printf("[S] 파일을 보내지 않고 종료합니다.\n");
94             close(c_sock);
95             close(s_sock);
96             return 0;
97         }else{ // 5.2. 파일이름 다시 입력받음
98             rfp = fopen(buf,"r");
99             if (rfp != NULL){
100                 if (send(c_sock, "Y",sizeof("Y"), 0) == -1){
101                     perror("[S] Can't send message that I don't have file.\n");
102                     exit(1);
103                 }
104                 printf("[S] I have file : %s\n",buf);
105             }
106             break;
107         }
108     }
109 }
110
111 }
```

서버코드는 클라이언트로부터 두 가지 경우의 응답을 받습니다. “N” 또는 ‘N외 의 다른 응답’입니다.

서버에서는 응답으로 N을 받았을 경우 클라이언트 프로그램이 했던 것과 같이 그대로 프로그램을 종료합니다.

(91 라인 : 주석 5.1)

N이외의 다른 내용을 받았을 때는, 이를 파일 이름으로 인식하고 그 이름의 파일이 있는지 확인합니다.

(97 라인 : 주석 5.2)

만약 새로 받은 파일이름을 갖고 있다면 파일이 있다는 “Y” 메시지를 클라이언트에 보내고 파일을 전송하게 되고, 응답을 대기하고 있던 클라이언트 쪽에서도 그 응답을 받아 파일을 전송 받게 됩니다.

아래 코드는 클라이언트 코드로, 이 경우 (라인75 : 주석 4.2) 에 해당합니다.

```
50 while(1){
51     strncpy(saveFname+1,buf,9);
52     saveFname[9] = '\0'; // buf에 saveFname 배열보다 더 많은 문자있을때 예외처리
53
54     //4. 보낸 파일이름이 서버가 갖고있는지 응답받음          클라이언트
55     if (recv(c_sock, buf, BUFFSIZE, 0) == -1) {
56         perror("[C] Can't receive message about having file.\n"); (다시 while 처음으로 -> 응답 대기중 이였음
57         exit(1);          돌아옴)
58     }
59
60     //4.1 파일 없다 응답 받음
61     if (buf[0] == 'N') //서버로 부터 응답으로 그 파일이 없음을 나타냄.문자열비교 함수를 쓸 수 있으나, 코드가 길어져서 버퍼0번항목만 비교함.(strcmp)
62     {
63         printf("[C] Server dont't have file.\n");
64         printf("[C] 다른 파일이름을 입력하시겠습니까? Y/N : \n");
65         int check = -1;
66
67         while(check == -1){
68             scanf("%c",&buf[0]);
69             if(buf[0] != 'Y' && buf[0] != 'N'){
70                 printf("[C] 응답은 Y또는 N 한글자로만 해주십시오.\n");
71             }else{
72                 check = 0;
73             }
74         }
75     }else if(buf[0] == 'Y') //4.2 파일이 있다 응답받음
76     {
77         printf("[C] Server have file.\n");
78         break;
79     }else{
80         perror("[C] 코드상으로 잘못된 정보가 들어왔습니다.\n");
81         exit(1);
82     }
83 }
```


만약 새로 받은 파일 이름도 서버가 갖고 있지 않는 파일이라면, 서버 코드의 while의 시작점으로 돌아가서 없다는 "N" 신호를 클라이언트에 주게 되고, 여태까지 한 것과 동일하게 파일이름을 다시 받을 건지 while문을 돌며 반복하게 됩니다. (71 ,67 라인 : 주석 3.1)

바로 이전 페이지에서 나와 있는 클라이언트 코드는 (라인 60 : 주석 4.1)부분에서 서버의 "N"신호를 받고 이후 위에서 설명한 것과 같이 다시 반복하게 됩니다.

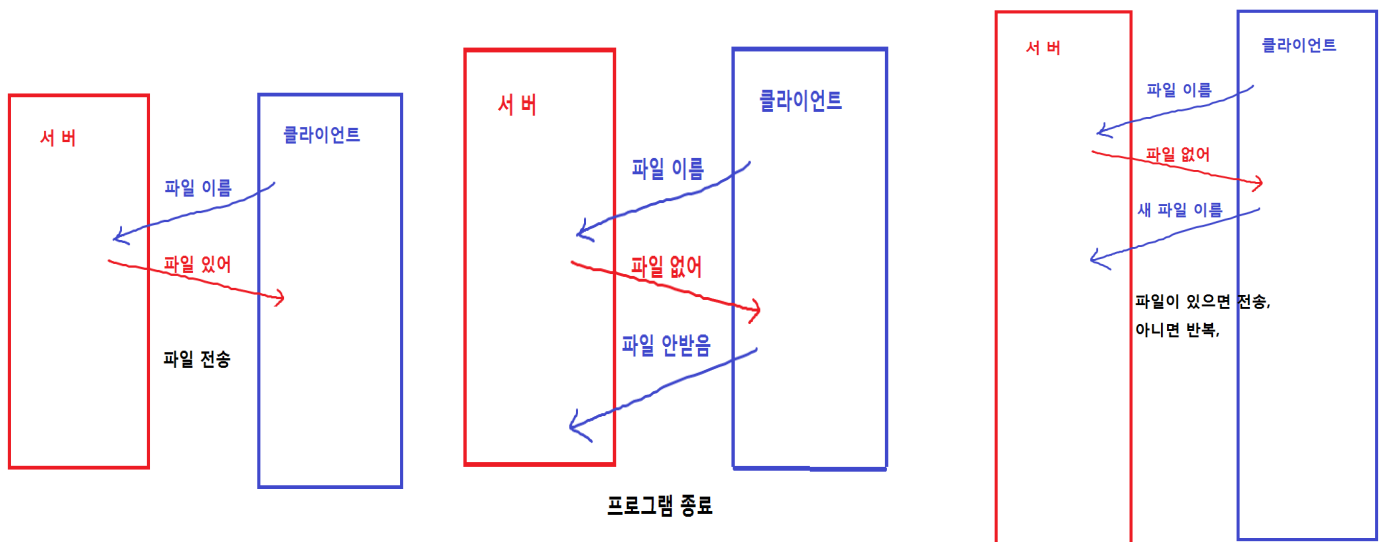
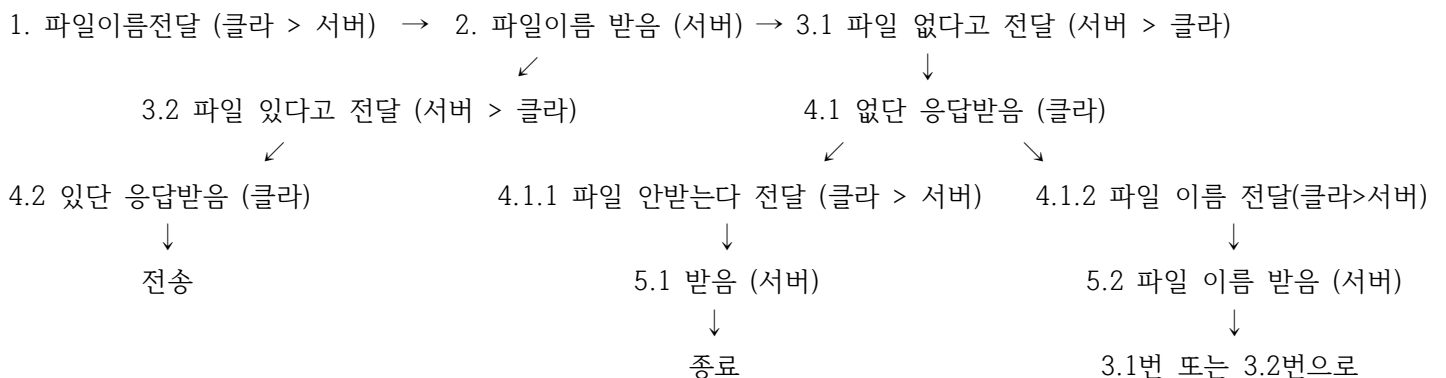
```

58 while(1){
59
60     printf("[S] Client request file name : %s\n", buf);
61
62     //3. 파일을 갖고있는지 여부 메세지 전송
63     rfp = fopen(buf, "r");
64     if (rfp == NULL) {
65         printf("[S] I don't have file\n");
66
67         if (send(c_sock, "N", sizeof("N"), 0) == -1){
68             perror("[S] Can't send message that I have file.\n");
69             exit(1);
70         }
71         request = -1; // 3.1 파일 없다고 표시하고 메세지 보냄
72     }else{ //3.2 파일이 있다는 메세지 보냄
73
74         printf("[S] I have file : %s\n", buf);
75
76         if (send(c_sock, "Y", sizeof("Y"), 0) == -1){
77             perror("[S] Can't send message that I don't have file.\n");
78             exit(1);
79         }
80         break;
81     }

```

**서버 코드
(while로 다시 돌아옴)**

코드 속 주석으로 대략 정리해보면 다음과 같습니다.



그러면 이제 서버에 존재하는 파일이름을 받았고, 파일을 전송하는 코드를 보겠습니다.

서버 코드입니다.

```
112 // 파일에 딱 버퍼 사이즈 만큼 : Fsize = BUFFSIZE와 같음. 그러면 한번 더 루프 돌아서 0 을 send, Fsize =0;
113 // 버퍼보다 약간 부족 : Fsize < BUFFSIZE일테고, 바로 break됨 (send 0 안함) Fsize = 0아님
114
115 int Fsize;
116 while(1){
117     memset(buf,0,BUFFSIZE);
118     Fsize = fread(buf, 1, BUFFSIZE, rfp);
119
120     if(Fsize < BUFFSIZE){
121         if (send(c_sock, buf, Fsize, 0) == -1) {
122             perror("[S] Can't send file Contents\n");
123             exit(1);
124         }
125         break;
126     }
127     if (send(c_sock, buf, BUFFSIZE, 0) == -1) {
128         perror("[S] Can't send file Contents\n");
129         exit(1);
130     }
131 }
132
133 if(Fsize != 0)
134 {
135     if (send(c_sock, buf, 0, 0) == -1) {
136         perror("[S] Can't send file Contents\n");
137         exit(1);
138     }
139 }
140
141
142 printf("[S] 파일 전송을 완료하였습니다.\n");
143
144 fclose(rfp);
145 close(c_sock);
146 close(s_sock);
147
148 return 0;
149 }
```

이전 과정에서 클라이언트가 원하는 파일을 열었고, fread로 파일의 항목을 1바이트씩 Buffsize만큼 읽어 buff에 저장하였습니다. (118 라인)

fread가 성공하면 읽은 항목 수를 리턴 하고 읽을 항목이 없으면 0을 리턴 하므로, Fsize에 리턴값을 저장을 하여 얼마만큼의 바이트를 읽었는지 파악했습니다. (118 라인)

이때 읽은 바이트가 Buffsize보다 작아지면 파일을 다 읽었다는 뜻이므로 while문을 빠져 나오게 했습니다. (120~125 라인)

클라이언트코드 에서 0바이트의 값을 받는 것으로 while문을 빠져나가게 하기 위해서 133라인에 0바이트를 보내지 않는경우에 대해 보낼수 있도록 처리해놨습니다. (112, 113라인 주석 설명)

클라이언트 코드입니다.

recv함수도 fread와 비슷하게

리턴값이 수신한 데이터의 크기임을 이용해서,

파일을 열고, (105~109 라인)

while 문을 돌며 (112 라인)

서버로부터 0바이트를 받았을 때 break 하도록 했습니다. (120 라인)

서버 코드에서 마지막 전송을

할 때 0바이트를 보내도록 코드를 해놓았으므로

제대로 코드가 작동하였습니다.

```
105 wfp = fopen(saveFname, "w");
106 if (wfp == NULL){
107     perror("[C] Can't File open\n");
108     exit(1);
109 }
110
111 int Fsize;
112 while(1)
113 {
114     memset(buf,0,BUFFSIZE);
115
116     if ((Fsize = recv(c_sock, buf, BUFFSIZE, 0)) == -1) {
117         perror("[C] Can't receive file data.\n");
118         exit(1);
119     }
120     if (Fsize == 0)
121     {
122         break;
123     }
124     fwrite(buf,1,Fsize,wfp);
125 }
126
127
128 printf("[C] 파일 수신을 완료하였습니다.\n");
129
130 fclose(wfp);
131 close(c_sock);
132
133 return 0;
134 }
```

클라이언트 코드

1. 1)코드 시행 결과 (한 개의 실행이며, 사진 순서대로 출력 값)

서버

```
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.
[S] Connected: client IP addr=10.0.0.134 port=51926
[S] Client request file name : aaa.txt
[S] I don't have file
[S] I'm waiting for client's reply!
```

클라이언트

```
ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 aaa.txt
[C] Server dont't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
```

```
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.
[S] Connected: client IP addr=10.0.0.134 port=51926
[S] Client request file name : aaa.txt
[S] I don't have file
[S] I'm waiting for client's reply!
[S] Client request file name : solo.txt
[S] I don't have file
[S] I'm waiting for client's reply!
```

```
ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 aaa.txt
[C] Server dont't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
Y
파일 이름을 입력하세요 :
solo.txt
[C] Server dont't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
[C] 응답은 Y또는 N 한글자토만 해주세요.
```

```
ubuntu@201619460:~/hw11$ ./hw11s
[S] I'm waiting for client's connect! My port is 7799.
[S] Connected: client IP addr=10.0.0.134 port=51926
[S] Client request file name : aaa.txt
[S] I don't have file
[S] I'm waiting for client's reply!
[S] Client request file name : solo.txt
[S] I don't have file
[S] I'm waiting for client's reply!
[S] 파일을 보내지 않고 종료합니다.
ubuntu@201619460:~/hw11$
```

```
ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 aaa.txt
[C] Server don't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
Y
파일 이름을 입력하세요 :
solo.txt
[C] Server don't have file.
[C] 다른 파일이름을 입력하시겠습니까? Y/N :
[C] 응답은 Y또는 N 한글자토만 해주십시오.
N
파일 전송을 하지 않고 종료합니다.
ubuntu@201619460:~/hw11$
```

2) 전송 및 diff 결과 포함 시행 :

```
ubuntu@201619460:~/hw11$ ./hw11s & ./hw11c
[1] 4007106
< Usage: ./hw11c IP_to_access PortNumber file >
ubuntu@201619460:~/hw11$ [S] I'm waiting for client's connect! My port is 7799.

ubuntu@201619460:~/hw11$ ./hw11c 10.0.0.134 7799 test
[S] Connected: client IP addr=10.0.0.134 port=52666
[S] Client request file name : test
[S] I have file : test
[S] 파일 전송을 완료하였습니다.
[C] Server have file.
[C] 파일 수신을 완료하였습니다.
[1]+  Done                  ./hw11s
ubuntu@201619460:~/hw11$ diff test Ctest
ubuntu@201619460:~/hw11$
```

서버 실행

클라이언트 직접 입력 값
(클라이언트 실행)

서버 코드에서 출력

클라이언트에서 출력

test 복사로 Ctest 생성됨.

> 기존파일

```
This file is for test~
Yeah homework 11 unix! yeah~
~
```

```
~
~
~
~
"test" 3L, 53C
```

> 생성된 파일

```
This file is for test~
Yeah homework 11 unix! yeah~
~
~
```

```
~
~
~
~
"Ctest" 3L, 53C
```

```
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

참고 자료 : <https://mintnlatte.tistory.com/555>

3. 파일 저장 형식 :

저장 위치를 다르게 하거나 저장되는 파일 이름을 다르게 해야 했습니다.

저는 전송받은 파일의 이름을 다르게 하는 쪽으로 생각했습니다.

일본 파일의 확장자를 바꾸게 되는 일이 생길 수 있겠다고 생각하였고,

그래서 기존 파일 이름을 복사하되, Copy의 C를 맨 앞에 붙인 이름으로 저장하고자 하였습니다.

그래서 배열 `save file name`이란 의미의 `saveFname[10]`을 선언하여서 전송받는 파일의 이름에 C를 붙인 꼴의 새로운 이름으로 저장하였습니다.

```
char saveFname[10];
```

[illegible]