

# 과제 #12

• 201821688 김요셉

## 과제#12-1

- 기술적 문제점

1. Fork를 반복문 안에서 수행시 재귀호출이 발생하지 않도록 주의
  - > Parent와 child의 수행부분을 독립시켜 코드짜기
2. 연속적으로 fork가 일어나기 때문에 fork의 반환값이 달라지는 것에 대비하여 배열에 반환값 저장

```
41 //동시에 세 클라이언트까지 접속 가능
42 listen(s_sock,3);
43 c_addr_size = sizeof(struct sockaddr);
44
45 printf("[S] waiting for a client..\n");
46 c_sock = accept(s_sock, (struct sockaddr *) &client_addr, &c_addr_size);
47 if(c_sock == -1){
48     perror("[S] Can't accept a connection [-1]");
49     exit(1);
50 }
51 else if(c_sock == 0) {
52     perror("[S] Can't accept a connetion [0]");
53     exit(1);
54 }
55
56 //fork 3회 반복
```

P-server

Fork가 3회 진행되는 조건이므로  
최대 3개의 소켓과 동시에 통신 가능하도록 하였다 (42행)

```

56 //fork 3회 반복
57 for(i = 1; i<4; i++){
58     pidnum[i] = fork();
59
60     if(pidnum[i] < 0) { //fork 실패시
61         perror("Fork Faild");
62         return 1;
63     }
64
65     if(pidnum[i] == 0) { //fork 성공시 & child process
66         printf("[S] Connected: [%02d번째]client IP addr=%s port=%d PID=%d\n", i, inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port), getpid());
67
68         //1. say hello to client 연결수립되면 접속된 클라이언트에게 메세지 전송
69         if(send(c_sock, hello, sizeof(hello)+1, 0) == -1){
70             perror("[S] Can't send message");
71             exit(1);
72         }
73
74         printf("[S] [%02d번째] PID:%d I said Hello

```

문제점1/2를 해결하고자 65/88행의 조건문을 통해 parent/child의 영역을 독립시켰으며, 57/58/65행과 같이 반환값을 fork시도 마다 배열에 저장하였다.

```

81
82     printf("[S] [%02d번째] Client says: %s\n", i, buf);
83
84     close(c_sock);
85     exit(0);
86 }
87
88 if(i < 3){
89     printf("[S] waiting for a client..\n");
90     c_sock = accept(s_sock, (struct sockaddr *) &client_addr, &c_addr_size);
91
92     if(c_sock == -1){
93         perror("[S] Can't accept a connection [-1]");
94         exit(1);
95     }

```

# Page #5 – 시간에 따른 결과

# 과제#12-1

```
ubuntu@201821688:~/hw12$ ./p-server
[S] waiting for a client..
[S] waiting for a client..
[S] Connected: [01번째] client IP addr=10.0.0.172 port=58554 PID=262512
[S] [01번째] PID:262512 I said Hello to Client!
[S] waiting for a client..
[S] Connected: [02번째] client IP addr=10.0.0.172 port=58556 PID=262513
[S] [02번째] PID:262513 I said Hello to Client!
[S] [01번째] Client says: Hi~ I am Client!!

[S] [02번째] Client says: Hi~ I am Client!!

PID 262512 종료
PID 262513 종료
[S] Connected: [03번째] client IP addr=10.0.0.172 port=58558 PID=262514
[S] [03번째] PID:262514 I said Hello to Client!
█
```

```
ubuntu@201821688:~/hw12$ ./p-client & ./p-client_sleep & ./p-client
[1] 262509
[2] 262510
[C] Connecting...
[C] Connected!
[C] Connecting...
[C] Connected!
[C] Server says: Hello~ I am Server!

[C] I said Hi to Server!
[C] Server says: Hello~ I am Server!

[C] I said Hi to Server!
[1]- Done ./p-client
ubuntu@201821688:~/hw12$ [C] Connecting...
[C] Connected!
[C] Server says: Hello~ I am Server!

[C] I am going to sleep...
█
```

```
[S] [02번째] Client says: Hi~ I am Client!!

PID 262512 종료
PID 262513 종료
[S] Connected: [03번째] client IP addr=10.0.0.172 port=58558 PID=262514
[S] [03번째] PID:262514 I said Hello to Client!
[S] [03번째] Client says:
PID 262514 종료
ubuntu@201821688:~/hw12$ █
```

```
[C] I said Hi to Server!
[C] Server says: Hello~ I am Server!

[C] I said Hi to Server!
[1]- Done ./p-client
ubuntu@201821688:~/hw12$ [C] Connecting...
[C] Connected!
[C] Server says: Hello~ I am Server!

[C] I am going to sleep...
█
```

```
[C] I said Hi to Server!
[1]- Done ./p-client
ubuntu@201821688:~/hw12$ [C] Connecting...
[C] Connected!
[C] Server says: Hello~ I am Server!

[C] I am going to sleep...

[2]+ Done ./p-client_sleep
ubuntu@201821688:~/hw12$ █
```

## 과제#12-2

- 기술적 문제점

1. 서버로부터 저장되는 파일의 이름이 'download\_file'로 고정되어 있어서 동시에 작업을 진행할 경우 한 파일에 반복해서 덮어씌워진다.

- 문제점 1을 위한 해결방안

1. 서버로부터 파일을 요청할 때마다 scanf()를 통해 저장할 이름을 결정한다.
  - > 매번 이름을 결정하는 것은 10개의 쓰레드를 통해 동시에 작업이 일어나는 것을 보이게 하자는 이번 과제에 부적절하다.
2. 기존에 high-level file I/O 형식으로 파일 전달하던 것을 low-level file I/O로 전환, open() 명령어 사용시 'O\_EXCL'을 활용하여 중복파일 여부를 확인한다.

# Page #8

---

```
137 int main(void) {
138     int i, s_sock, c_sock[MAX_THREADS+1], *status;
139     struct sockaddr_in server_addr, client_addr;
140     socklen_t c_addr_size;
141     char server_ip[20] = {0,};
142     int t_count = 1;
143
144     //해당 프로그램이 작동되는 서버 ip 받아올 것!
145     GetMyIpAddr(server_ip);
146
147     //소켓생성, 리턴값은 소켓의 파일디스크립터
148     s_sock = socket(AF_INET, SOCK_STREAM, 0);
149
150     int option = 1;
151     setsockopt(s_sock, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option));
152
153     //memset과 같은 기능인데 0으로 초기화
154     //소켓의 주소 구조체를 0으로 초기화
155     bzero(&server_addr, sizeof(server_addr));
156
157     //해당 소켓은 네트워크를 이용하는 인터넷 소켓이다
158     server_addr.sin_family = AF_INET;
159     server_addr.sin_port = htons(SERVERPORT);
160     server_addr.sin_addr.s_addr = inet_addr(server_ip); //jcloud에 할당된 ip 받아와서 넣기
161
162     if(bind(s_sock, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1){
163         perror("Can't bind a socket");
164         exit(1);
165     }
166
167     while(1){
168
```

main함수는 listen 이전까지 기본적으로 소켓프로그래밍을 순서대로 진행된다.



# Page #9

```
166
167 while(1){
168
169     //동시에 최대 10개의 통신
170     listen(s_sock, 10);
171
172     c_addr_size = sizeof(struct sockaddr);
173
174     //c_sock[]에 각 클라이언트의 fd저장 > filetrans함수로 전달해서 사용
175     //server를 계속 돌리면서 10개 중 일부가 끝났으면 다른 client가 접근 가능하게 만들건가
176     //아니면 10개가 다 끝나면 새롭게 server를 시작하게 만들건가?
177     c_sock[t_count] = accept(s_sock, (struct sockaddr *) &client_addr, &c_addr_size);
178     if(c_sock[t_count] == -1){
179         perror("Can't accept a connection");
180         exit(1);
181     }
182
183     printf("** Check: s_sock = %d, c_sock = %d\n", s_sock, c_sock[t_count]);
184     printf("** Check: client IP addr=%s port=%d\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
185
186     if(pthread_create(&tid[t_count], NULL, &filetrans, &c_sock[t_count])){
187         perror("Failed to create thread");
188         goto exit;
189     }
190     printf("서버에 접속한 클라이언트의 누적숫자 : %d\n", t_count);
191     t_count++;
192
193     //10개의 스레드만 작업할 때의 경우
194     //if(t_count == MAX_THREADS+1){
195     //    printf("서버에 누적 10개의 스레드가 작업하였습니다.\n");
196     //    break;
197     //}
198     //동시에 10개의 스레드만 작업 가능할 경우(11개부터 서버 종료, 10개 이하의 경우 서버 계속 run)
199     if(t_count == MAX_THREADS+1){
200         printf("서버에 누적 10개의 스레드가 작업하였습니다.\n");
201         t_count = 1;
202     }
203
204
205
206 }
```

Bind에 이어지는 listen부터는 while문 안에서 이루어진다.

과제#12-2의 목표는 동시에 10개의 스레드를 이용해 서버와 통신하는 것이다.

스레드를 생성하기 위해서는 먼저 클라이언트의 접속을 수락해야한다. 이를 위해 accept가 반복적으로 수행되며 클라이언드를 수락하고 새로운 소켓을 반환한다. 해당 소켓은 파일 전달을 위한 함수에서 서버-클라이언트 간의 통신을 위해 주소값을 인자로 넘겨준다.

Accept가 진행될 때마다 카운트를 증가시켜 누적되는 접속수를 확인한다.

## Server

```
45 void* filetrans (void *arg) {
46     int c_sock = *((int *)arg);
47     int *ret, indent = *((int *)arg);
48     char filebuf[BUFSIZE] = {0};
49     char buf[BUFSIZE] = {0};
50     int send_data_size, send_file_size;
51     char *send_file_name;
52     int fd;
53
54     //1. 클라이언트와 연결수립되면 접속된 클라이언트에게 메세지 전송
55     if(send(c_sock, hello, sizeof(hello)+1, 0) == -1){
56         perror("Can't send message");
57         exit(1);
58     }
59
60     printf("클라이언트와 연결 성공\n");
61
62     //2. recv 클라이언트가 요청하는 파일명
63     bzero(buf, sizeof(buf));
64
65     if(recv(c_sock, buf, BUFSIZE, 0) == -1) {
66         perror("Can't receive request filename");
67         exit(1);
68     }
69
70     //서버에 클라이언트가 요구하는 파일 유무 확인하기 위한 파일명 변수
71     send_file_name = buf;
72     printf("클라이언트가 요청한 파일명: %s\n", send_file_name);
73
74     //동시 10개 쓰레드 작업시 클라이언트에서 fopen시 파일이 덮어쓰는 것을 방지하기 위해 open으로 변경(O_EXCL사용을 위해)
75     fd = open(send_file_name, O_RDONLY);
76
77     //해당 파일이 디렉토리에 없는 경우
78     if(fd == -1) {
79         printf("요청받은 파일은 디렉토리에 없는 파일입니다\n");
80
81         if(send(c_sock, nofile, sizeof(nofile), 0) == -1){
82             perror("Can't send message");
83             close(fd);
84             exit(1);
85         }
86     }
87     //파일 전송
```

클라이언트로부터 요청을 확인하고 요청받은 파일을 클라이언트에게 전송하는 filetrans 함수이다.

Filetrans 함수는 먼저 클라이언트와 연결되었음을 알리는 메세지를 보낸다. 해당 메세지는 전역변수로 설정하였다. 클라이언트는 서버와 연결되었음을 확인하면 명령행인자로부터 받은 파일의 이름을 서버에 존재하는지 확인차 전송한다.

## Client

```
63 //1. recv msg from server
64 //리시브 호출시 패킷이 도착할 때까지 프로세스가 멈춰서 대기
65 if(recv(c_sock, buf, BUFSIZE, 0) == -1){
66     perror("Can't receive message");
67     exit(1);
68 }
69 //서버와의 연결확인 메세지 출력
70 printf("%s\n", buf);
71
72 //서버에게 요청하는 파일 이름 전송
73 if(send(c_sock, request_file_name, sizeof(request_file_name)+1, 0) == -1){
74     perror("요청사항 보낼 수 없음");
75     exit(1);
76 }
77
78 printf("서버에 <%s> 다운로드 요청\n\n", request_file_name);
79
```

# Page #11

## Server

```
62 //2. recv 클라이언트가 요청하는 파일명
63 bzero(buf, sizeof(buf));
64
65 if(recv(c_sock, buf, BUFSIZE, 0) == -1) {
66     perror("Can't receive request filename");
67     exit(1);
68 }
69
70 //서버에 클라이언트가 요구하는 파일 유무 확인하기 위한 파일명 변수
71 send_file_name = buf;
72 printf("클라이언트가 요청한 파일명: %s\n", send_file_name);
73
74 //동시 10개 쓰레드 작업시 클라이언트에서 fopen시 파일이 덮어쓰이는 것을 방지하기 위해 open으로 변경(O_EXCL사용을 위해)
75 fd = open(send_file_name, O_RDONLY);
76
77 //해당 파일이 디렉토리에 없는 경우
78 if(fd == -1) {
79     printf("요청받은 파일은 디렉토리에 없는 파일입니다\n");
80
81     if(send(c_sock, nofile, sizeof(nofile), 0) == -1){
82         perror("Can't send message");
83         close(fd);
84         exit(1);
85     }
86 }
87 //파일 전송
88 else{
89     //3. 전송할 파일 이름 보내주기
90     send(c_sock, send_file_name, sizeof(send_file_name), 0);
91
92     //4. 클라이언트로부터 파일 이름 수신여부 확인
```

클라이언트로부터 파일명을 전달 받은 서버는 해당 파일이 디렉토리 내에 있는지 확인한다.

Open()함수의 특정 flag는 파일이 존재하지 않을 경우 -1을 반환하는 것을 이용한다. 해당 파일이 존재함을 확인하면 파일이름을, 없으면 파일이 없음을 안내한다.

```
80 //파일에 수신 데이터 저장
```

```
81 //3. 서버로부터 확인받은 파일 정보 수신 및 출력
```

```
82 bzero(buf, BUFSIZE);
```

```
83
84 if(recv(c_sock, buf, BUFSIZE, 0) == -1){
85     perror("Can't receive filename");
86     exit(1);
87 }
```

```
88 down_file_name = buf;
```

```
89 printf("서버 측의 <%s> 파일 확인완료\n", down_file_name);
90
```

## Client

# Page #12

---

```
91
92 //4. 클라이언트로부터 파일 이름 수신여부 확인
93 if(recv(c_sock, buf, BUFSIZE, 0) == -1){
94     perror("Can't receive file name confirm");
95     close(fd);
96     exit(1);
97 }
98
```

Server

클라이언트는 서버로부터 해당 파일이 존재함을 확인 받는다.

서버는 클라이언트가 해당 파일이 존재함을 확인했는지 메시지를 수신한다.

```
80 //파일에 수신 데이터 저장
81 //3. 서버로부터 확인받은 파일 정보 수신 및 출력
82 bzero(buf, BUFSIZE);
83
84 if(recv(c_sock, buf, BUFSIZE, 0) == -1){
85     perror("Can't receive filename");
86     exit(1);
87 }
88 down_file_name = buf;
89 printf("서버 측의 <%s> 파일 확인완료\n", down_file_name);
90
91 //4. 서버에 파일명 수신 확인
92 if(send(c_sock, success, sizeof(success)+1, 0) == -1){
93     perror("요청사항 보낼 수 없음");
94     exit(1);
95 }
```

Client

```
96
97 //open해서 이미 파일 있을 경우 물어보기
98 //중복되는 파일명으로 저장되는 경우 방지 #1 (hw12내 filetest.c에서 코드연습 진행 후 옮김)
99 //초기에 밑의 #2방식으로만 코드를 짰으나 여러 프로세스가 동시에 작업되는 것을 표현하기에 부적절하여 #1의 방식을 기본으로 설정함
100 for(i = 1; i < 10; i++) {
101     memset(num, '\0', sizeof(num));
102     sprintf(num, "%d", i); //num에 정수를 문자열로 저장
103     strcat(num, down_file_name); //해당 정수를 저장하려는 파일명 앞에 붙이기
104     strcpy(savefilename, num); //직관적인 변수명으로 전환
105     fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
106     if(fd > 0) { //open 성공시 (=중복되는 파일명 x)
107         printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
108         break;
109     }
110 }
111
112 //중복되는 파일명으로 저장되는 경우 방지 #2
113 //1~9 + <파일명>으로도 중복되는 파일명이 저장되는 경우는 유저에게 저장하려는 이름 묻기
114 if(fd == -1){ //파일 열기 실패시
115     printf("<%=s> 파일을 다운받아 저장할 영문 20자 미만 이름입력\n", down_file_name);
116     scanf("%s", savefilename);
117     fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
118     if(fd > 0) {
119         printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
120     }
121     else {
122         while(1){
123             printf("중복되는 이름의 파일명입니다.\n다른 이름을 입력해주세요\n");
124             scanf("%s", savefilename);
125             fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
126             if(fd > 0) {
127                 printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
128                 break;
129             }
130         }
131     }
132 }
133
```

## Client

서버로부터 파일이 있음을 확인한 클라이언트는 파일을 전송 받아 중복되지 않는 이름으로 저장할 준비를 해야한다.

Strcat함수를 사용하면 두 문자열을 합칠 수 있다. 하지만 단순히 strcat을 사용할 경우 1. 파일 확장자 뒤에 문자열이 이어지거나 2. 단순히 파일이름 앞 부분에 긴 문자열을 연결하는 방법이 된다.

전달 받은 파일명 앞에 1~9를 붙여 직관적이고 최대한 짧은 파일명을 만들기 위해 좌측과 같은 방법을 사용하였다.

>계속

# Page #14

```
96
97 //open해서 이미 파일 있을 경우 물어보기
98 //중복되는 파일명으로 저장되는 경우 방지 #1 (hw12내 filetest.c에서 코드연습 진행 후 옮김)
99 //초기에 밑의 #2방식으로만 코드를 짰으나 여러 프로세스가 동시에 작업되는 것을 표현하기에 부적절하여 #1의 방식을 기본으로 설정함
100 for(i = 1; i < 10; i++) {
101     memset(num, '\0', sizeof(num));
102     sprintf(num, "%d", i); //num에 정수를 문자열로 저장
103     strcat(num, down_file_name); //해당 정수를 저장하려는 파일명 앞에 붙이기
104     strcpy(savefilename, num); //직관적인 변수명으로 전환
105     fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
106     if(fd > 0) { //open 성공시 (=중복되는 파일명 x)
107         printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
108         break;
109     }
110 }
111
112 //중복되는 파일명으로 저장되는 경우 방지 #2
113 //1~9 + <파일명>으로도 중복되는 파일명이 저장되는 경우는 유저에게 저장하려는 이름 묻기
114 if(fd == -1){ //파일 열기 실패시
115     printf("<%=s> 파일을 다운받아 저장할 영문 20자 미만 이름입력\n", down_file_name);
116     scanf("%s", savefilename);
117     fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
118     if(fd > 0) {
119         printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
120     }
121     else {
122         while(1){
123             printf("중복되는 이름의 파일명입니다.\n다른 이름을 입력해주세요\n");
124             scanf("%s", savefilename);
125             fd = open(savefilename, O_CREAT|O_EXCL|O_WRONLY, 0644); //저장하려는 파일이 이미 존재하는지 확인
126             if(fd > 0) {
127                 printf("<%=s>이름으로 파일을 저장합니다\n", savefilename);
128                 break;
129             }
130         }
131     }
132 }
133
```

Client

배열 num을 null로 초기화하고 정수를 sprintf를 통해 문자열로 num에 저장한다. 정수는 반복문을 거듭하며 1부터 9까지 증가한다.

Strcat을 통해 파일명 앞에 숫자를 연결한다.

해당 문자열을 직관적으로 사용하기 위해 num에서 savefilename으로 복사한다.

Savefilename에 저장된 파일명이 클라이언트의 디렉토리 내에 생성한다. O\_CREAT | O\_EXCL을 통해 해당 파일명이 이미 존재할 경우 -1이 반환되어 해당 파일명이 이미 존재함을 확인할 수도 있다.

반복문을 통해 1+파일명부터 9+파일명까지 확인한다.

반복문이 끝날 때까지 중복되는 파일명일 경우 유저가 직접 파일명을 설정할 수 있도록 하였다.

```
99 //파일 전송을 위해 읽어들이는 데이터의 크기 확인
100 send_data_size = read(fd, filebuf, BUFSIZE);
101 printf("read file size : %dbyte\n", send_data_size);
102
103 if(send_data_size == -1){
104     perror("fread error");
105     close(fd);
106     exit(1);
107 }
108 else if (send_data_size == 0) {
109     perror("파일 읽는 중 문제 발생\n");
110     close(fd);
111     exit(1);
112 }
113
114 //5. 파일 전송
115 if((send_file_size = send(c_sock, filebuf, send_data_size, 0)) == -1){
116     perror("send file error");
117     close(fd);
118     exit(1);
119 }
120 else if (send_file_size == 0){
121     perror("send의 반환값이 0. 정상적으로 파일이 전송되지 않음\n");
122     close(fd);
123     exit(1);
124 }
125 }
126 //최종 전송된 파일의 데이터 크기 확인
127 printf("send %dbyte\n", send_file_size);
```

## Server

클라이언트로부터 요청 받은 파일을 read를 통해 버퍼에 저장한다. 정상적으로 버퍼에 저장되었는지 확인 및 클라이언트로 파일 전송시 사용하기 위해 변수에 따로 저장한다. Read/write 함수의 경우 코딩을 할 때 문제가 발생하는 경우가 빈번하기 때문에 read함수의 반환값 및 perror를 적극 활용한다.

버퍼에 저장된 데이터를 read를 통해 읽은 데이터의 크기(반환값)만큼 클라이언트에 전송한다. Read때와 마찬가지로 오류 발생 여부를 확인한다.

# Page #16

```
134 //5. 파일 수신
135 data_size = recv(c_sock, buf, BUFSIZE, 0); //서버로부터 수신된 데이터의 크기 저장
136 if(data_size == -1){
137     perror("Can't receive file");
138     close(fd);
139     exit(1);
140 }
141
142 printf("수신 받은 파일 크기 : %dbyte\n", data_size);
143
144 write_data_size = write(fd, buf, data_size); //버퍼에 저장된 수신 데이터를 파일에 쓰기, 정상적으로 쓰여졌는지 확인하기 위해 반환값 저장
145 printf("write 데이터 크기 : %dbyte\n", write_data_size);
146 if(write_data_size > 0){
147     printf("파일수신 성공!\n<%s>를 확인해보세요\n", savefilename);
148 }
149 else {
150     perror("파일수신 실패\n");
151     //fclose(fp);
152     close(fd);
153     exit(1);
154 }
155
156 close(fd);
157 close(c_sock);
158
159 return 0;
160 }
```

Client

서버로부터 전송 받은 데이터를 버퍼에 저장한다. Recv의 반환값을 활용해 해당 데이터가 정상적으로 전송되었는지 간접적으로 확인한다. 버퍼에 저장된 내용을 write함수를 통해 파일에 저장한다. 마찬가지로 write의 반환값을 통해 전송받은 크기 만큼 파일에 입력되었는지 확인한다. 파일과 소켓을 닫고 종료한다.



# Page #17

```
126 //최종 전송된 파일의 데이터 크기 확인
127 printf("send %dbyte\n", send_file_size);
128
129 close(fd);
130 close(c_sock);
131 ret = (int *)malloc(sizeof(int));
132 *ret = indent;
133 pthread_exit(ret);
134 }
```

Server

126~134행을 통해 전송된 파일의 크기를 확인하고 파일 및 소켓을 종료한다.  
상태종료값을 인자로 전달하며 스레드를 종료한다.

191행의 `t_count++`를 통해 서버와 연결되었던 소켓의 숫자를 센다. 10개의 소켓과 통신한 경우 최종적으로 `t_count`는 11이 된다.

최초 1회 이후에도 스레드 10개를 동시에 돌리기 위해서는 `t_count`를 초기화해준다. 193~197행의 코드가 그것이다.

```
186 if(pthread_create(&tid[t_count], NULL, &filetrans, &c_sock[t_count])){
187     perror("Failed to create thread");
188     goto exit;
189 }
190 printf("서버에 접속한 클라이언트의 누적숫자 : %d\n", t_count);
191 t_count++;
192
193 //동시에 10개의 스레드만 작업 가능할 경우(11개부터 서버 종료, 10개 이하의 경우 서버 계속 run)
194 if(t_count == MAX_THREADS+1){
195     printf("서버에 누적 10개의 스레드가 작업하였습니다.\n");
196     t_count = 1;
197 }
198
199
200
201 }
202 exit:
203 for(i=0; i<t_count; i++){
204     pthread_join(tid[t_count], (void **) &status);
205     printf("Thread no.%d ends: %d\n", t_count, *status);
206 }
207
208 close(s_sock);
209
210 return 0;
211 }
```

Server

# Page #18

---

```
205  
206     }  
207     exit:  
208         for(i=0; i<t_count; i++){  
209             pthread_join(tid[t_count], (void **) &status);  
210             printf("Thread no.%d ends: %d\n", t_count, *status);  
211         }  
212  
213     close(s_sock);  
214  
215     return 0;  
216 }
```

Server

쓰레드 종료대기 이후 서버의 소켓도 종료  
하고 서버 프로그램을 마친다.

# Page #19 - 결과

The screenshot shows a VS Code editor with a C program named `server.c` and its execution output in the terminal. The program is designed to handle 10 concurrent client connections using threads. The terminal output shows the server successfully connecting to 10 clients, receiving data, and sending responses. The output is as follows:

```
[3] Done ./client 10.0.0.172 7799 t2.c
서버와 연결 성공

ubuntu@201821688:~/hw12$
서버에 <t1.c> 다운로드 요청

서버 측의 <t1.c> 파일 확인완료
서버와 연결 성공
<t1.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 2133byte

write 데이터 크기 : 2133byte
파일수신 성공!
<t1.c>를 확인해보세요
서버에 <t2.c> 다운로드 요청

서버 측의 <t2.c> 파일 확인완료
<t2.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 1737byte
write 데이터 크기 : 1737byte
파일수신 성공!
<t2.c>를 확인해보세요
서버에 <first.c> 다운로드 요청

서버와 연결 성공

서버 측의 <first.c> 파일 확인완료
<first.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 2133byte
write 데이터 크기 : 2133byte
파일수신 성공!
<first.c>를 확인해보세요
서버와 연결 성공

서버에 <thread.c> 다운로드 요청

서버 측의 <thread.c> 파일 확인완료
<thread.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 2309byte
파일수신 성공!
<thread.c>를 확인해보세요
서버와 연결 성공

서버에 <iptest.c> 다운로드 요청

서버 측의 <iptest.c> 파일 확인완료
<iptest.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 649byte
write 데이터 크기 : 649byte
파일수신 성공!
<iptest.c>를 확인해보세요
서버와 연결 성공

서버에 <iptest.c> 다운로드 요청

서버 측의 <iptest.c> 파일 확인완료
<iptest.c>이름으로 파일을 저장합니다
수신 받은 파일 크기 : 649byte
write 데이터 크기 : 649byte
파일수신 성공!
<iptest.c>를 확인해보세요
ubuntu@201821688:~/hw12$
```

동시에  
10개의 쓰레드가  
서버와  
통신한 모습