

프로젝트 개요

우리는 프로젝트 2를 통해서 클라이언트가 가지고 있는 파일 리스트를 전송하고 서버 측에서는 이를 토대로 리스트를 txt 파일로 작성하여 저장하였다. 프로젝트 3에서는 클라이언트가 서버가 가지고 있는 리스트를 수신하여 사용자가 볼 수 있으며 파일 다운로드 까지 가능한 기능을 추가로 구현한 것이다. 프로젝트 3가 가지고 있는 기능을 정리하자면 다음과 같다.

- **List Upload** : 파일을 업로드 하는 과정이다. 프로젝트 2와 거의 동일하다.
- **List Receive** : 클라이언트가 서버로부터 리스트를 수신하는 과정이다.
- **Download file** : 파일을 다운로드 하는 과정이다. 서버로부터 파일 정보를 받아 파일을 가지고 있는 클라이언트와 통신하여 파일 다운로드를 진행한다.

위 세가지 기능들을 구현 하려면 각각의 기능마다 해결 및 보완 해야 할 과제들이 존재했다.

List upload

프로젝트 2를 진행하면서 다소 아쉬웠던 점은 동일 한 파일의 경우에도 불하고 리스트에 다시 등록된다는 점이었다. 이에 대해서 프로젝트 3에서는 동일한 파일명의 경우에는 리스트에 등록되지 않도록 하는 기능을 추가하고자 한다.

List Receive

서버가 가지고 있는 리스트를 받아오는 과정이다. 서버의 list.txt 파일의 정보를 그대로 가져올 경우 List 의 순번이 없어 향후 다운로드 과정에서 파일을 선택에 어려움이 존재하다. 그러므로 사용자가 보다 보기 쉽도록 순번을 붙여서 보여주는 과정을 진행해야 한다.

Download file

본 프로젝트의 핵심적인 기능이자 가장 어려움이 많이 존재하는 부분이다. 우선 서버로부터 사용자가 선택한 파일의 정보 가져와야 한다. 이때 Port, Ip, 파일명 등의 정보를 나누어서 분류를 진행해야 한다. 클라이언트간의 통신이 이루어져야 하므로 클라이언트가 서버로서 작동을 할 수 있도록 만듦과 동시에 서버의 클라이언트로서도 작동해야 한다. 당연히 fork를 사용하여 병행적으로 작동시켜야 한다.

위 세가지 기능 공통적으로 가지고 있어야 할 것이 얼마큼 송수신 할 것 인가에 대한 문제이다. 가령 4개의 파일 리스트를 송수신 할 경우 4번의 송수신 과정을 거치면 된다. 하지만 이는 파일이 4개라고 정해져 있을경우지 추가로 생성되거나 삭제될 경우에는 recv로 인한 대기가 무한히 지속될 수도 있고 송신자가 보낸 것들을 제대로 수신하지 못할 경우가 발생한다. 이를 해결해주는 알고리즘 또한 필요하다.

위에서 언급한 문제 와 보완점 등을 해결한다면 프로젝트 3를 무사히 마칠 수 있을 것이며 완성하였다. 이제부터 어떻게 위에서 언급한 것들을 해결 했는지에 대해서 이야기 하고자 한다.

List upload

본 보고서에서는 프로젝트2에서 보완한 부분만 설명하고자 한다. 프로젝트 2에서는 같은 파일명이 있을 지라도 리스트에 추가하였다. 하지만 우리는 추가적인 기능으로 동일한 파일명이 있을시에는 추가하지 않는 방향으로 진행하고자 한다. 구현된 코드를 보면 다음과 같이 표현이 가능하다.

```
server.c      client.c      client2.c
165     case 1:{
166         printf("\n===== %s are uploading list =====\n",id );
167         listFile = fopen(listName,"a+");
168         send(new_fd,"recv_OK" ,strlen("recv_OK") +1,0);
169         recv(new_fd, cnt_ch, sizeof(cnt_ch),0);
170         count = atoi(cnt_ch);
171
172         while(count > 0){
173             recv(new_fd, msg, sizeof(msg),0);
174             send(new_fd,"recv_OK" ,strlen("recv_OK") +1,0);
175
176             listComp = fopen(listName,"r");
177             while(1){
178                 fgets(list_buf, sizeof(list_buf), listComp);
179                 inName = strtok(list_buf, " ");
180                 if(feof(listComp)) break;
181                 if(strcmp(inName, msg)==0) {
182                     list_non = 1;
183                     printf("%s is alraday in list\n",msg );
184                 }
185             }
186             fclose(listComp);
187
188             if(list_non ==0){
189                 strcat(msg," ");
190                 strcat(msg,id);
191                 strcat(msg," ");
192                 strcat(msg,client_IP);
193                 strcat(msg," ");
194                 strcat(msg,client_Port);
195                 strcat(msg,"\n");
196                 fputs(msg,listFile);
197                 printf("%s",msg);
198             }
199             list_non = 0;
200             count--;
201         }
202         count=0;
203         fclose(listFile);
204     }break;//case1;
```

< Server 측 코드>

```
server.c      client.c      client2.c
152     case 1:{
153         dir_info = opendir("./my_file");
154         if(NULL != dir_info){
155             while(dir_entry = readdir(dir_info)){count++;}
156             count = count - 2;
157         }
158         sprintf(cnt_ch,"%d",count);
159         send(sockfd,cnt_ch, sizeof(cnt_ch), 0);
160         recv(sockfd, buf, sizeof(buf), 0);
161         closedir(dir_info);
162         count = 0;
163         dir_info = opendir("./my_file");
164         if(NULL != dir_info){
165             while(dir_entry = readdir(dir_info))
166             {
167                 if((strcmp(dir_entry->d_name,"..")&& strcmp(dir_entry->d_name,".")!=0)){
168                     send(sockfd, dir_entry->d_name, sizeof(dir_entry->d_name) +1, 0);
169                     recv(sockfd, buf, sizeof(buf), 0);
170                     printf("%s // Success \n",dir_entry->d_name);
171                 }
172             }
173             }closedir(dir_info);
174     }break;
```

< Client 측 코드>

디렉토리를 열고 접근하는 방식과 파일을 생성하여 리스트를 쓰는 과정은 프로젝트 2에서와 동일하니 이에대한 설명은 생략하도록 하겠다. 프로젝트 3에서 추가적으로 진행한 부분은 서버쪽 코드에서 176 줄 부터이다. 우선 비교할 용도로 다시 list.txt 파일을 연다. 파일의 끝이 나올때까지 안에 있는 내용을 읽어오면서 파일명을 비교한다. 이때 strtok 라는 함수를 사용하여 파일명만을 추출 해낸다. 비교한 결과 클라이언트에서 보낸 파일과 중복될 경우 list_non을 1로 설정하여 파일에 쓰는 과정을 진행하지 않도록 하도록 하였다. 이에 대한 실행 결과는 다음과 같이 보여진다.본 실험은 list.txt 가 비어 있는 상태에서 진행되었으며 mode 1이면 리스트를 업로드 하는 기능이 진행된다.

```

st2014146004@602-d: ~/server
st2014146004@602-d:~/server$ ./a.out
Server-socket() sokfd is OK...
Server-bind() is Ok...
listen() is OK...
accept is OK...
user1 is connected
=====
USER Information
ID : user1 , PW : asd
=====
Log -in success!! [user1] *^^*
IP : 220.149.128.101 Port : 4101

===== user1 are uploading list =====
korea.txt user1 220.149.128.101 4101
hello.txt user1 220.149.128.101 4101
ps.txt user1 220.149.128.101 4101
test.txt user1 220.149.128.101 4101

===== user1 are uploading list =====
korea.txt is alraday in list
hello.txt is alraday in list
ps.txt is alraday in list
test.txt is alraday in list
[

st2014146004@602-c: ~/client_folder
st2014146004@602-c:~/client_folder$ ./a.out
Server-socket() sokfd is OK...
Client - connect() is Ok
=====HELLO!=====
=====Please Log In=====
INPUT YOUR ID :user1
Input yout PW :asd
Log in Success!!
Server-socket2() sokfd is OK...
Server-bind() is Ok...
listen() is OK...

=====Select mode=====
1 : list upload
2 : list recive
3 : download file
select:1
korea.txt // Success
hello.txt // Success
ps.txt // Success
test.txt // Success

=====Select mode=====
1 : list upload
2 : list recive
3 : download file
select:1
korea.txt // Success
hello.txt // Success
ps.txt // Success
test.txt // Success

=====Select mode=====
1 : list upload
2 : list recive
3 : download file
select:

```

두번 연속으로 업로드를 진행한 과정이다. 첫번째입력 에서는 리스트에 파일에 작성되지만 두번째 입력에서는 파일명을 보내는 과정만 성공하고 서버쪽에서는 이미 있는 파일이라 하여 리스트에 작성되지 않는다.

```

st2014146004@602-d: ~/server
1 korea.txt user1 220.149.128.101 4101
2 hello.txt user1 220.149.128.101 4101
3 ps.txt user1 220.149.128.101 4101
4 test.txt user1 220.149.128.101 4101

```

서버에서 list.txt 파일을 열어본 모습이다. 예상대로 개의 파일 리스트만 존재하고 있음을 알아 볼 수 있다.

위 리스트 중복 방지 과정을 진행해 줌으로서 같은 파일명으로 된 리스트가 생성되는 것을 막아 사용자가 좀 더 보기 편하게 리스트를 볼 수 있게 되었다.

List Receive

리스트를 읽어서 보내는 과정은 어렵지 않게 진행 할 수 있었다. 순서로 표현하자면 다음과 같다.

- 서버쪽의 list.txt 파일에 있는 리스트 개수 세기, count 값에 저장
- Count 값을 문자열로 변환시켜 클라이언트에게 전송
- 서버와 클라이언트는 count 값만큼 송수신 과정 진행
- 서버는 클라이언트에게 리스트에 있는 데이터를 보낼때 순번 값을 추가하여 전송

이 과정에서 잘 다루어 주어야 할 부분은 형변환이다. Count 값과 num 값에 대해서 atoi 와 sprintf를 사용해 주어서 int 와 문자열에 대해 자유롭게 오갈 수 있어야 한다. 형변환을 해주는 이유는 통신을 하는데 주로 사용되는 send, recv 를 사용하려면 문자열 이어야 하며 count 같이 숫자를 세기 위해서는 int 가 더 편하기 때문이다. 자세한 코드는 다음과 같다.

server.c	client.c	client2.c
<pre>205 case 2:{ 206 printf("\n===== %s are request the list =====\n",id); 207 listFile = fopen(listName,"r"); 208 count = 0; 209 while(!feof(listFile)){ 210 fgets(list_buf, sizeof(list_buf), listFile); 211 count++; 212 }fclose(listFile); 213 214 char cnt_ch[50]= {0,}; 215 sprintf(cnt_ch,"%d",count); 216 send(new_fd,cnt_ch, sizeof(cnt_ch), 0); 217 recv(new_fd, msg, sizeof(msg), 0); 218 219 num=1; 220 221 listFile = fopen(listName,"r"); 222 while(count!=num){ 223 fgets(list_buf, sizeof(list_buf), listFile); 224 sprintf(num_ch,"%d",num); 225 strcat(num_ch," "); 226 strcat(num_ch,list_buf); 227 228 send(new_fd,num_ch ,sizeof(num_ch) ,0); 229 recv(new_fd, msg, sizeof(msg),0); 230 num++; 231 } 232 count=0; 233 num=1; 234 fclose(listFile); 235 }break; //case2</pre>		

<Server 측 코드>

코드를 살펴 보면 앞서 이야기 했던 내용들이 들어있는 사실을 알 수 있다. 먼저 파일의 갯수가 몇개인지를 알기위하여 list.txt. 파일을 연다. (listName 이라는 변수에 list.txt 가 설정되어 있다.) 파일은 열었으면 EOF 가 나올때까지 개수를 센다. 이때 feof 를 사용하였다. 이때 마지막 널값 까지 개수를 센다. 그래서 클라이언트 쪽에서 count 에서는 널값을 고려하여 프로그래밍을 진행하여야 한다.

Count 값을 송수신 한 후 번호를 부여해 준다. 번호 부여에 관여하는 변수는 num 이다.

별 어려운 것 없이 num 을 앞에 붙여주고 이 데이터를 보내면 쉽게 끝나는 부분이다.

server.c	client.c	client2.c
<pre> 176 case 2:{ 177 recv(sockfd, cnt_ch, sizeof(cnt_ch),0); 178 send(sockfd,"recv_OK" ,strlen("recv_OK") +1,0); 179 count = atoi(cnt_ch); 180 printf("=====File List=====\\n"); 181 while(count>1){ 182 recv(sockfd, buf, sizeof(buf), 0); 183 send(sockfd, "recv_OK", sizeof("recv_OK") +1, 0); 184 printf("%s",buf); 185 count--; 186 }count = 0; 187 printf("=====\\n"); 188 189 }break; </pre>		

< Client 측 코드>

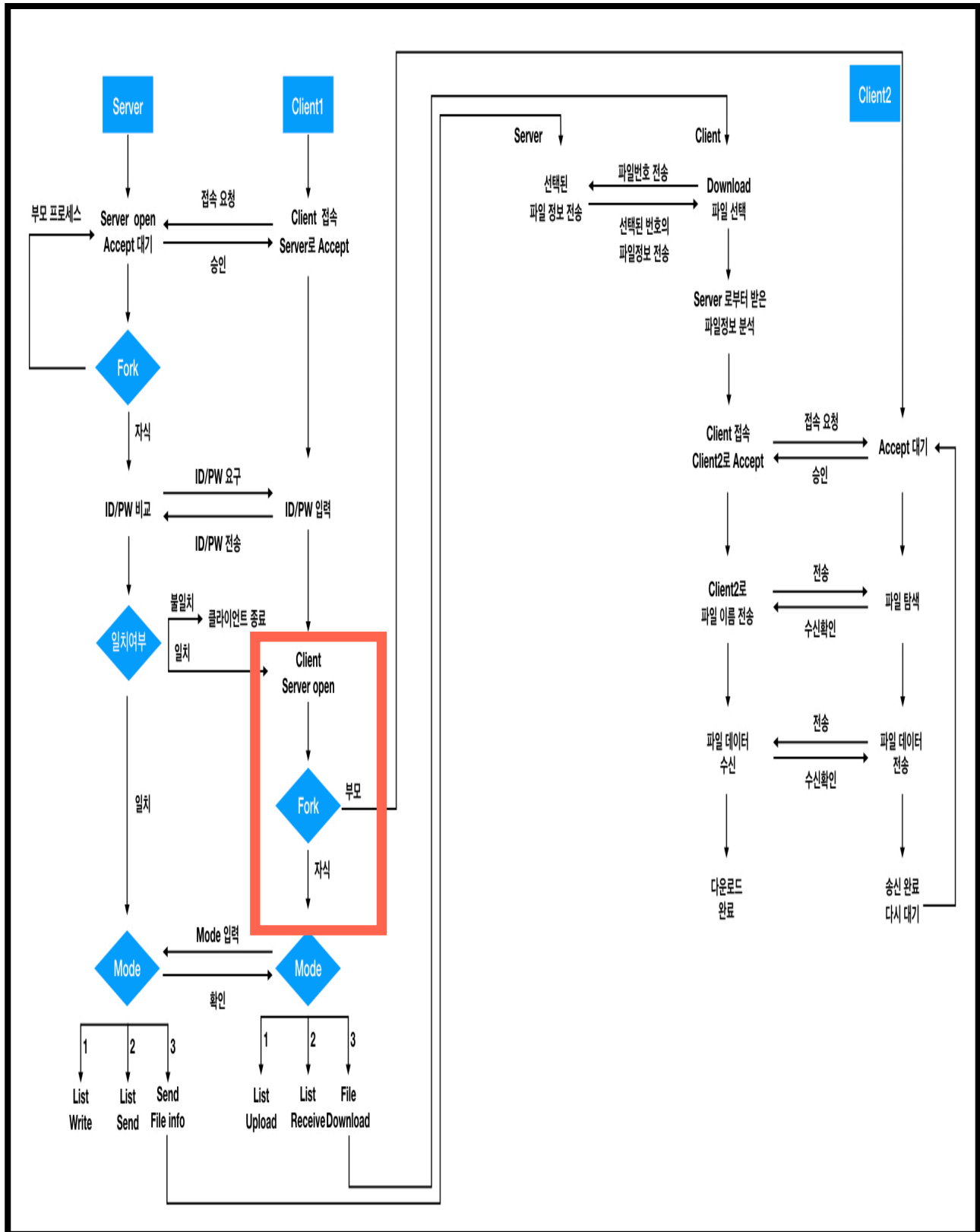
클라이언트 측 코드는 너무나도 간단하다. 서버쪽에서 파일의 갯수를 센 것을 받은 이후 정수로 형변환 한 다음 파일의 개수 만큼 송수신을 진행하면 된다. 그리고 위에서 언급 되었듯이 널값을 고려해 카운트값이 1이 될때 까지만 송수신을 하도록 진행 해 주었다.
리스트를 받아오는 과정의 실행모습은 다음과 같이 진행된다.

st2014146004@602-d: ~/server	st2014146004@602-c: ~/client_folder
<pre> st2014146004@602-d:~/server\$./a.out Server-socket() sockfd is OK... Server-bind() is Ok.... listen() is OK... accept is OK... user1 is connected ===== USER Information ID : user1 , PW : asd ===== Log -in success!! [user1] *^^* IP : 220.149.128.101 Port : 4101 ===== user1 are request the list ===== accept is OK... user2 is connected ===== USER Information ID : user2 , PW : asd ===== Log -in success!! [user2] *^^* IP : 220.149.128.102 Port : 4102 ===== user2 are uploading list ===== hello2.txt user2 220.149.128.102 4102 user2.txt user2 220.149.128.102 4102 embedded_OS.txt user2 220.149.128.102 4102 test2.txt user2 220.149.128.102 4102 ===== user1 are request the list ===== </pre>	<pre> =====Please Log In===== INPUT YOUR ID :user1 Input your PW :asd Log in Success!! Server-socket2() sockfd is OK... Server-bind() is Ok.... listen() is OK... =====Select mode===== 1 : list upload 2 : list recive 3 : download file select:2 =====File List===== 1 korea.txt user1 220.149.128.101 4101 2 hello.txt user1 220.149.128.101 4101 3 ps.txt user1 220.149.128.101 4101 4 test.txt user1 220.149.128.101 4101 =====Select mode===== 1 : list upload 2 : list recive 3 : download file select:2 =====File List===== 1 korea.txt user1 220.149.128.101 4101 2 hello.txt user1 220.149.128.101 4101 3 ps.txt user1 220.149.128.101 4101 4 test.txt user1 220.149.128.101 4101 5 hello2.txt user2 220.149.128.102 4102 6 user2.txt user2 220.149.128.102 4102 7 embedded_OS.txt user2 220.149.128.102 4102 8 test2.txt user2 220.149.128.102 4102 =====Select mode===== 1 : list upload 2 : list recive 3 : download file select: </pre>

실행 모습을 보면 코드대로 잘 실행되고 있는 모습을 볼 수 있다.

Download File

이는 프로젝트3의 핵심 기능이라 할 수 있을만큼 중요한 부분이다. 위의 기능들과 달리 서버와 클라이언트 간의 통신만 진행하는 것이 아니라 다른 클라이언트와의 통신도 진행하기 때문에 전체적인 흐름의 파악이 중요하다. 다음은 우리가 설계한 프로젝트 3의 시스템 흐름도이다.



위의 시스템 흐름도에서 가장 중요하게 바라볼 부분은 빨간색으로 표시한 부분이다. 나머지 좌측면에 있는 부분은 프로젝트2 와 앞서 자세히 이야기한 바가 있다. 빨간색으로 표시한 부분을 설명하자면 클라이언트가 서버로서 작동하게 될 부분이다. 즉 클라이언트간의 통신을 실시할때 fork 를 통해 병행 프로세스를 만들어준다. 코드를 통해 바라보면 다음과 같다.

server.c	client.c	client2.c
<pre> 84 85 if((strcmp(buf,"Log in fail, Incorrect ID")&& strcmp(buf,"Log in fail, Incorrect PW")) != 0){ 86 87 //////////////////////////////////client as server 88 sockfd2 = socket(AF_INET, SOCK_STREAM, 0); 89 if(sockfd2 == -1){ 90 perror("Server-socket2() error lol!"); 91 exit(1); 92 } 93 else printf("Server-socket2() sockfd is OK...\n"); 94 95 my_addr.sin_family = AF_INET; 96 97 my_addr.sin_port = htons(Client_PORT_INT); 98 99 my_addr.sin_addr.s_addr = INADDR_ANY; 100 101 memset(&(my_addr.sin_zero), 0,8); 102 103 if(setsockopt(sockfd2, SOL_SOCKET, SO_REUSEADDR, (char*)&val, sizeof(val))<0){ 104 perror("setsockopt"); 105 close(sockfd2); 106 return -1; 107 } 108 109 if(bind(sockfd2, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) 110 { 111 perror("Server-bind() error lol...!"); 112 exit(1); 113 } 114 else printf("Server-bind() is Ok....\n"); 115 116 if(listen(sockfd2, BACKLOG) ==-1) 117 { 118 perror("listen() error lol"); 119 exit(1); 120 }else printf("listen() is OK...\n"); 121 122 sin_size = sizeof(struct sockaddr_in); 123 ////////////////////////////////// 124 125 pid = fork(); 126 127 if(pid ==0){ </pre>		

서버를 열때와의 코드와 동일하게 작성 되었으며 IP 값과 Port 값만 변경되어 작성되었다.

위 과정을 거친 후 자식 프로세스는 기존 서버와의 통신을 하여 다운로드 하는 과정 이외의 것들을 진행한다. 부모 프로세스는 다른 클라이언트간의 통신을 위해서 대기하기 시작한다.

클라이언트가 모드 3을 선택하면 시스템의 흐름도 처럼 서버, 클라이언트, 클라이언트2 끼리의 통신이 진행된다. 흐름도의 Client2 라고 표시되어 있는 부분은 이해를 돕기 위한 부분이다. 클라이언트라면 1,2 무엇이든지 간의 위의 기능 수행이 가능하다.

이제 다운로드 과정을 살펴봐야 하는데 2부분으로 나누어 볼 수 있다.

- Server 와 Client 간의 통신 : client 가 파일을 선택하고 파일정보를 받아오는 과정이다.
- Client 와 Client 간의 통신 : Server 로부터 받은 파일정보를 가지고 Client 간 통신을 통해 파일을 다운로드 하는 과정이다.

- Server 와 Client 간의 통신

이 과정은 우리가 지금까지 해왔던 방식과 동일하게 진행 해 볼 수 있다. Client 파일번호를 선택하면 Server 는 해당되는 파일 번호를 뽑아와 클라이언트에게 전송 한다. Client 는 Server 로 부터 받은 파일 정보를 분석하여 실제 파일을 가지고 있는 Client 에게 접속을 시도한다. 코드는 다음과 같다.

```
server.c      client.c      client2.c
191     case 3:{
192         char fileNum[10] = {0,};
193         int file_int;
194
195         printf("Select file : ");
196         scanf("%s", fileNum);
197         file_int = atoi(fileNum);
198
199         send(sockfd, fileNum, sizeof(fileNum), 0);
200         recv(sockfd, buf, sizeof(buf), 0);
201
202         send(sockfd, "recv_OK", sizeof("recv_OK") +1, 0);
203         recv(sockfd, buf, sizeof(buf), 0);
204
205
206         char d_filename[40] = {0,};
207         char d_port[40] = {0,};
208         char d_IP[40] = {0,};
209
210         char* ptr = strtok(buf, " ");
211         while (ptr != NULL)
212         {
213             ptr = strtok(NULL, " ");
214             if(count == 0) strcpy(d_filename,ptr);
215             if(count == 2) strcpy(d_IP,ptr);
216             if(count == 3) strcpy(d_port,ptr);
217             count++;
218         }
219         int cl_port;
220         cl_port = atoi(d_port);
221         printf("file name : %s, IP : %s Port : %d\n",d_filename,d_IP,cl_port);
222         count = 0;
```

<Client 측 코드>

```
server.c      client.c      client2.c
236     case 3:{
237         printf("\n===== %s are want's to file download =====\n",id );
238         listFile = fopen(listName,"r");
239         count = 0;
240         while(!feof(listFile)){
241             fgets(list_buf, sizeof(list_buf), listFile);
242             count++;
243         }fclose(listFile);
244
245         char fileNum[10]= {0,};
246         int file_int;
247         send(new_fd, "recv_OK", sizeof("recv_OK") +1, 0);
248         recv(new_fd, fileNum, sizeof(fileNum), 0);
249
250         file_int = atoi(fileNum);
251         num=1;
252
253         listFile = fopen(listName,"r");
254         while(count!=num){
255             fgets(list_buf, sizeof(list_buf), listFile);
256             sprintf(num_ch,"%d",num);
257             strcat(num_ch, " ");
258             strcat(num_ch,list_buf);
259             if(file_int == num){
260                 send(new_fd,num_ch ,sizeof(num_ch) ,0);
261                 recv(new_fd, msg, sizeof(msg),0);
262                 printf("%s Downloading %s",id, list_buf);
263             }
264             num++;
265         }
266         fclose(listFile);
267     }break;//case3
```

<Server 측 코드>

Server 가 번호를 받아온 과정부터 설명해 보도록 하겠다. Client로 부터 번호를 받아오면 이를 정수형으로 형변환 한다. List를 연 후 순서를 세어 해당되는 번호가 맞을 경우에 Client 로 파일 정보를 전송한다. Client는 Server 로부터 파일 정보를 받아들이며 이를 한칸의 띄어 쓰기로 글자를 나누어 구분한다. 누누어진 글자들 중 우리에게 필요한 정보만을 따로 저장하는 과정을 진행한다. 여기 까지 진행 하면 파일 다운로드 과정에서 Server의 역할은 끝이 나게 된다. 이제 클라이언트 간 통신이다.

파일정보에서 IP, PORT 정보를 추출해 내었다. 이에 해당되는 곳으로 연결을 시도하면 Client2의 부모 프로세스가 요청을 받아 줄 것이다. 그 이후 파일명을 보내고 파일데이터를 받는 과정을 진행하면 된다.

```
server.c | client.c | client2.c
246
247     FILE *downFile;
248     size_t n_size;
249
250     char root[40] = {"/Download/"};
251     int msg_count=0;
252     strcat(root,d_filename);
253
254     send(sockfd3,d_filename ,sizeof(d_filename) +1,0);
255     recv(sockfd3, buf, sizeof(buf),0);
256     printf("Downloading %s\n", d_filename);
257
258     send(sockfd3,"recv_OK" ,strlen("recv_OK") +1,0);
259     recv(sockfd3, buf, sizeof(buf),0);
260     msg_count = atoi(buf);
261
262     downFile = fopen(root,"a");
263     while(msg_count>0){
264         send(sockfd3,"recv_OK" ,strlen("recv_OK") +1,0);
265         recv(sockfd3, msg, sizeof(msg)+1,0);
266         fputs(msg,downFile);
267         msg_count--;
268     }
269     printf("Download Success\n");
270     fclose(downFile);
```

파일을 다운로드 받는 입장인 클라이언트의 코드이다. 여태 우리가 해왔던 방식이랑 흡사하다. 파일명을 보내고 파일의 총 길이를 수신한 다음 그에 해당되는 길이만큼 송수신을 진행한다. 다운로드 한 파일은 Download 폴더에 저장이 된다. 시스템의 흐름만 알고있다면 구현이 가능했다.

```

281  if(pid >0){
282      while(1){
283          new_fd = accept(sockfd2, (struct sockaddr *)&their_addr, &sin_size);
284          FILE *sendFile;
285          size_t n_size;
286          char root[40] = {"/my_file/"};
287          char file_msg[512];
288          int msg_count=0;
289          char msgc_ch[20];
290
291          recv(new_fd, buf, sizeof(buf),0);
292          send(new_fd,"recv_OK" ,strlen("recv_OK") +1,0);
293          strcat(root,buf);
294
295          sendFile = fopen(root,"r");
296          while(fgets(file_msg,sizeof(file_msg),sendFile) !=NULL) msg_count++;
297          fclose(sendFile);
298
299          sprintf(msgc_ch,"%d",msg_count);
300          recv(new_fd,buf,sizeof(buf),0);
301          send(new_fd,msgc_ch,sizeof(msgc_ch),0);
302
303          sendFile = fopen(root,"r");
304          while(fgets(file_msg,sizeof(file_msg),sendFile) !=NULL)
305          {
306              recv(new_fd,buf,sizeof(buf),0);
307              send(new_fd,file_msg,sizeof(file_msg) +1,0);
308          }
309          fclose(sendFile);
310          close(new_fd);
311      }
312      close(sockfd2);
313  }

```

클라이언트가 파일을 전송해 줄때의 코드이다. 다른 클라이언트로 부터 접속이 들어와 파일을 요구하면 파일을 찾아서 보내주는 과정을 진행한다. 여기에서도 파일의 총 길이를 체크하여 보내준다. 마찬가지로 단편적인 부분만 보면 크게 어려움 없이 진행 할 수 있는 부분이었다.

Download 기능의 실행모습

```
st2014146004@602-d: ~/server
st2014146004@602-d:~/server$ ./a.out
Server-socket() sokfd is OK...
Server-bind() is Ok...
listen() is OK...
accept is OK...
user1 is connected
=====
USER Information
ID : user1 , PW : asd
=====
Log -in success!! [user1] *^^*
IP : 220.149.128.101 Port : 4101
accept is OK...
user2 is connected
=====
USER Information
ID : user2 , PW : asd
=====
Log -in success!! [user2] *^^*
IP : 220.149.128.102 Port : 4102

===== user1 are request the list =====

===== user1 are want's to file download =====
user1 Downloading embedded_OS.txt user2 220.149.128.102 4102
[

st2014146004@602-c: ~/client_folder
st2014146004@602-c:~/client_folder$ ./a.out
Server-socket() sokfd is OK...
Client - connect() is Ok

=====
=====HELLO!=====
=====Please Log In=====
INPUT YOUR ID :user1
Input your PW :asd
Log in Success!!
Server-socket2() sokfd is OK...
Server-bind() is Ok...
listen() is OK...

=====Select mode=====
1 : list upload
2 : list receive
3 : download file
select:2

=====File List=====
1 korea.txt user1 220.149.128.101 4101
2 hello.txt user1 220.149.128.101 4101
3 ps.txt user1 220.149.128.101 4101
4 test.txt user1 220.149.128.101 4101
5 hello2.txt user2 220.149.128.102 4102
6 user2.txt user2 220.149.128.102 4102
7 embedded_OS.txt user2 220.149.128.102 4102
8 test2.txt user2 220.149.128.102 4102

=====
=====Select mode=====
1 : list upload
2 : list receive
3 : download file
select:3
Select file : 7
file name : embedded_OS.txt, IP : 220.149.128.102 Port : 4102
Server-socket3() sokfd is OK...
Client3 - connect() is Ok

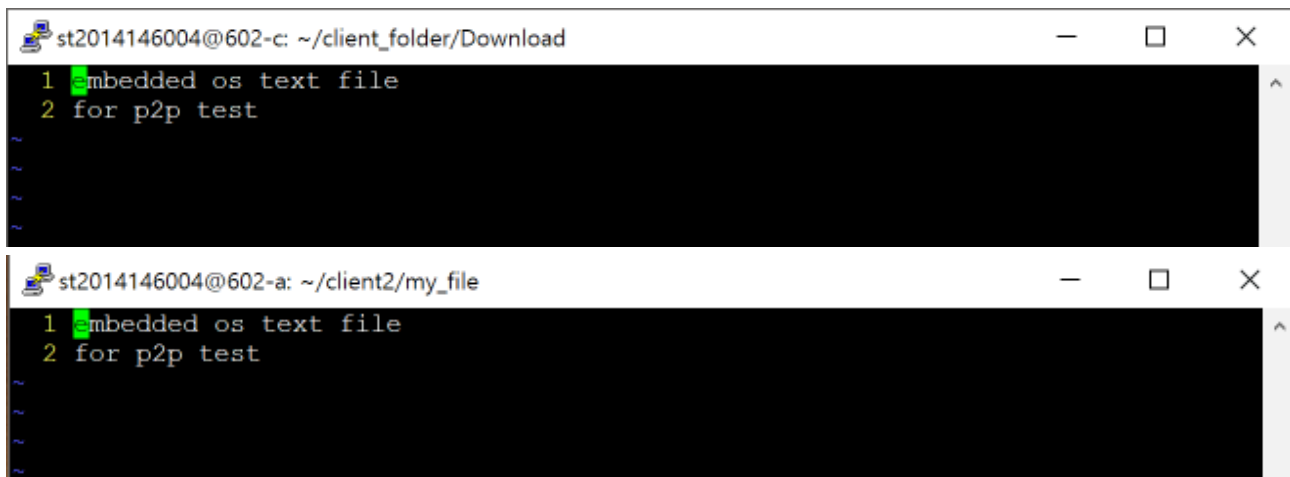
Downloading embedded_OS.txt
Download Success

=====Select mode=====
1 : list upload
2 : list receive
3 : download file
select:
```

User1 이 서버로 접근하여 리스트 목록을 받은 다음 파일을 다운로드 하는 과정이다. 이때 User2도 서버에 접근을 하고 있어야 파일 다운로드가 가능하다. 다운로드 된 파일은 Download 폴더에 저장된다.

```
st2014146004@602-c:~/client_folder$ ls
Download a.out client.c client_pro2.c my_file pro2
st2014146004@602-c:~/client_folder$ cd Download/
st2014146004@602-c:~/client_folder/Download$ ls
embedded_OS.txt
st2014146004@602-c:~/client_folder/Download$
```

다음과 같이 Download 폴더에 저장된 모습을 볼 수 있다. 다운로드 된 폴더를 열어보면 다음과 같다.

The image shows two terminal windows side-by-side. The top window has a title bar 'st2014146004@602-c: ~/client_folder/Download' and contains the text '1 embedded os text file' and '2 for p2p test'. The bottom window has a title bar 'st2014146004@602-a: ~/client2/my_file' and contains the same text: '1 embedded os text file' and '2 for p2p test'. Both windows have a black background with green and yellow text.

위에 있는 것이 다운로드 되어 다운로드 폴더에 저장된 파일의 모습이고 아래의 있는 것은 User2가 가지고 있던 원본 파일의 모습이다. 결과를 보면 알 수 있듯이 제대로 다운로드가 이루어진 것을 살펴 볼 수 있다.

프로젝트 결과

우리는 초반부에 언급한 List upload, List receive , File download 기능 구현시 생각했던 문제 및 보완점에 대해 해결해 나아가면서 프로젝트 3를 구현할 수 있었다. 본 프로젝트를 진행함에 있어서 가장 오랜 시간이 걸리고 고민이 많았던 부분은 시스템 흐름도를 작성하는 것이었다. 시스템이 어떻게 작동되는 지에 대한 이해와 설계가 없었다면 프로젝트3를 성공적으로 구현하지 못했을 수 있다. 그만큼 전체적인 틀을 잡는 것에 대한 중요성을 느낀 프로젝트였다. 프로그래밍을 진행할때 가장 오류가 많았던 부분은 시스템이 무한대기하는 현상이 생기는 것 이었다. 이는 recv 때문에 생기는 것으로 판단되었고 우리는 다소 번거롭더라도 recv 와 send의 짝을 맞춰주어 recv 혼자 대기하는 현상을 방지하였다. 이 송수신 과정을 얼마나 진행할 지에 대해서도 어려움이 있었다. 우리가 사용한 Count 라는 변수를 사용해 직접 몇줄인제 셀 수도 있으며 Null 값이 오면 종료하는 방식을 사용할 수도 있다. 또한 recv가 일정 시간 이상 진행될 경우 강제로 다음 단계로 진행하도록 할 수도 있다. 여러가지 방법이 있겠지만 우리는 Count 하는 방식을 사용했다. 이 방식을 디렉토리나 파일을 두번 열어야 한다. 하지만 정확하게 얼마큼 송수신 할지 알 수 있기 때문에 송수신 과정중에 무한대기나 파일 일부를 못받아오는 현상은 생기지 않는다. 시스템의 안정도에 좀 더 초점을 맞추었기 때문에 Count 하는 방식을 사용했다.

위와 같은 고민들을 하나씩 해결해 나아가면서 본 프로젝트를 완수 할 수 있었다. 운영체제 과목과 관련해 이론만 습득한 것 뿐 아니라 부분적으로 구현 해 볼 수 있어서 좋은 경험이었다고 생각한다. 다소 아쉬운 점은 좀 더 공부하여 세마포어, 교착상태 해결 과 관련된 알고리즘을 실제로 구현해 보지 못한 것이다. 이와 관련해서는 추가적으로 공부하여 본 프로젝트에 사용될 수 있도록 하고싶다.