****

**Embedded OS Report**

|  |  |  |
| --- | --- | --- |
| **이름** | **학과** | **학번** |
| **송근주** | **임베디드시스템학과** | **2015146019** |
| **변재중** | **임베디드시스템학과** | **2015146016** |

* **목차(Contents)**

1. **Term Project#2 – Group Chatting**
2. **개요**
3. **구현 방법**
4. **출력 결과**
5. **Term Project#3 – P2P FILE Transfer**
6. **개요**
7. **구현방법**
8. **출력 결과**
9. **Term Project#2 – Group Chatting**
10. **개요**

* **Term Project#1에서 구현한 Server에 Client1,2가 ID와 PW를 입력하여 접속하는 Server – Client Model를 이용하여 Term Project#2를 구현했다. 즉, Client1,2는 Server에 ID와 PW를 입력하여 접속하게 되면, Server에서 Group Chatting 환경으로 전환하는 방식이다. 그러나 여기에 중요한 조건이 들어가게 된다. 실제적인 Group Chatting 구현을 위해 Client1,2가 서로 메세지를 입력하는 동안에도 각자 입력하여 전송한 메시지들이 Client1,2 창에 나타나야 되는 비동기적 시스템을 구축해야 한다. 이러한 시스템을 구축하기 위해선 Term Project#1에서 구현한 Server와 Client1,2에 추가적인 역할을 하는 프로세스들이 필요하게 된다. 결과적으로 Term Project#2에서는 비동기적 시스템인 Group Chatting을 구현하기 위해 추가적인 프로세스 선언과 추가된 각 프로세스의 역할을 각각 Server와 Client1,2에 알맞게 구현하는 것을 요구하는 문제였다고 볼 수 있다.**

1. **구현 방법**

* **비동기적 시스템 Group Chatting을 구현하기 위해서는 Server와 Client1,2에 추가적인 프로세스 생성과 추가된 프로세스들의 역할을 정해줘야 된다는 걸 알게 되었다. 이에 우리는 먼저 Server와 Client1,2간의 구성도를 그려보면서 접근하였다.**

Client1’s massage

Client2’s massage

Client2’s ID/PW

Client1’s ID/PW

**위의 그림은 1차적으로 작성한 Server와 Client1,2간 구성도이다. 그러나 추가적인 프로세스들을 생성하고 생성한 프로세스들의 역할을 지정해주는 세세한 작업을 구상할 때 너무 대략적인 구성도여서 수정이 필요했다. 따라서 위의 구성도는 크게 Server와 Client1,2의 역할들만 다시금 인지하는 역할로 남겨 두었다.**

**⑧**

**⑦**

**⑥**

**⑤**

**③**

**④**

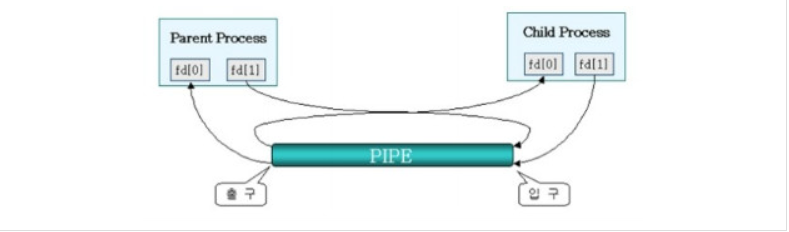
**②**

**①**

**위의 구성도는 Server와 Client1,2간 비동기적 시스템을 구축하기 위해 프로세스 생성을 추가한 구성도이다. 이러한 프로세스 생성은 fork()함수를 통해 생성할 수 있다. 위의 구성도를 통해 Server와 Client1,2간 각각 추가 생성된 프로세스들의 역할을 명시해 주었는데, 다음과 같이 명시해 주었다.**

1. **→ Client1 메시지 입력하여 Server에 전송**
2. **→ Server로부터 받은 Client2 메시지 출력**
3. **→ Client2 메시지 입력하여 Server에 전송**
4. **→ Server로부터 받은 Client1 메시지 출력**
5. **→ Client1로부터 받은 메시지 Server에 출력**
6. **→ Client2 메시지 Client1에 전송 (Server)**
7. **→ Client2로부터 받은 메시지 Server에 출력**
8. **→ Client1 메시지 Client2에 전송 (Server)**

**위의 역할들을 구현해주기 위해서 우리는 추가적인 fork()함수를 통해 나온 부모프로세스와 자식프로세스 간의 통신이 필요하다는 것을 알게 되었다. 따라서 부모프로세스와 자식프로세스 간의 통신을 지원하는 명령문을 찾은 결과, 가장 간단하면서도 직관적으로 이해가 되는 pipe() 명령문을 찾게 되었다. pipe() 명령문은 부모와 자식 프로세스간 데이터를 통신해주는 명령문으로, 구조는 다음과 같다.**



**또한 pipe() 명령문은 아래와 같이 정의 되어있다.**

**#include<unistd.h>**

**int pipe(int fd[2]);**

**즉 크기가 2인 int형 배열을 통해 fd[0]은 데이터를 입력 받을 수 있는 파일 디스크립터를 담는 역할을 하고, fd[1]은 데이터를 출력할 수 있는 파일 디스크립터를 담는 역할을 한다. 따라서 이전에 역할을 명시한 프로세스들 중 ⑤,⑦,⑥,⑧ 프로세스에 pipe()명령문을 이용하면 비동기적 시스템을 구축할 수 있다는 걸 알게 되었다.**

**Read fd1[0]**

**Client1’s message**

**Write fd1[1]**

**Client1’s message**

**Client1 message transfer**

**Client2 message transfer**

**Client2 message transfer**

**Write fd2[1]**

**Client2’s message**

**Read fd2[0]**

**Client2’s message**

**Client1 message transfer**

**③**

**⑦**

**⑧**

**⑥**

**⑤**

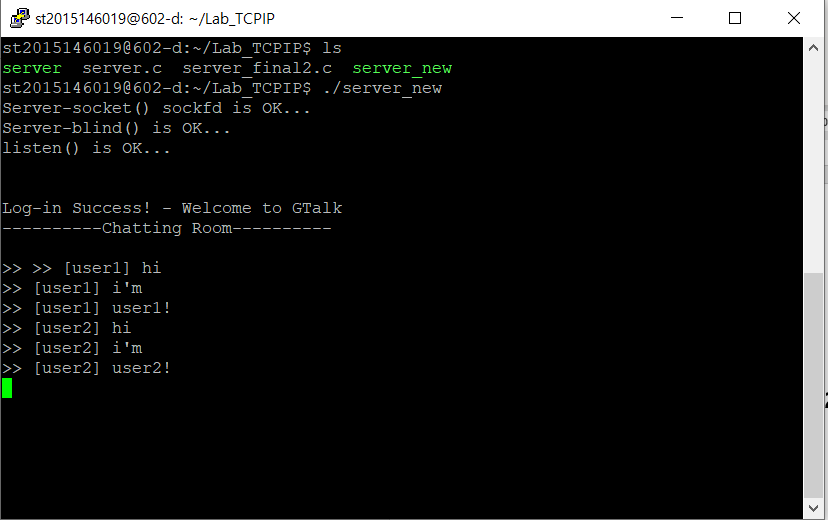
**④**

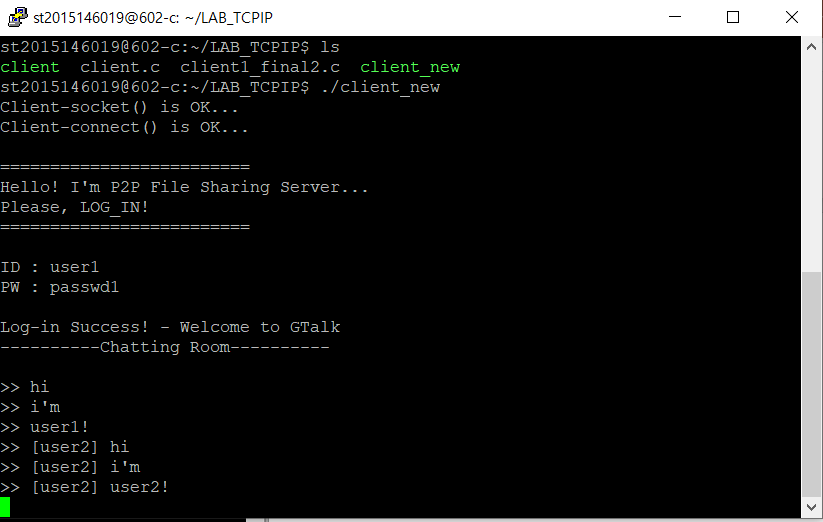
**②**

**①**

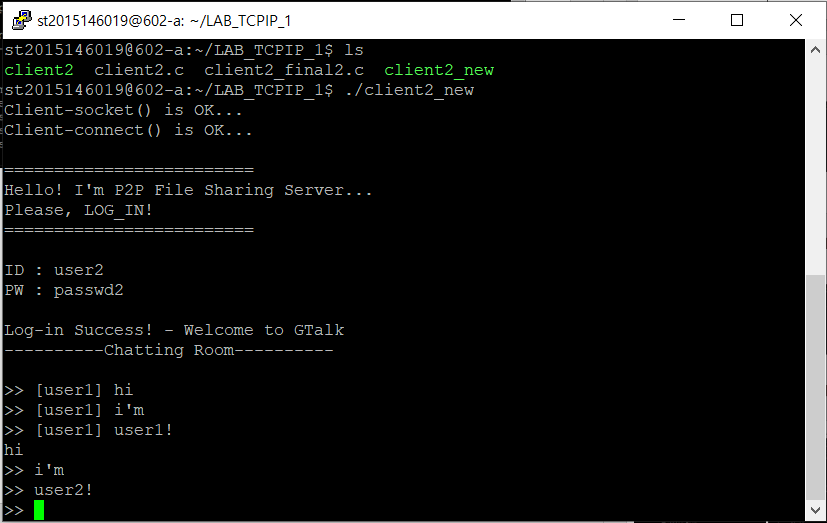
**위 구성도는 최종적으로 비동기적 시스템 Group Chatting을 구현하기 위한 Server와 Client1,2간의 구성도이다. 결과적으로 pipe() 명령문을 이용함으로써 Server에서 fork()로 생성된 부모와 자식 프로세스 간의 Client1,2의 메시지들을 비동기적으로 저장 및 불러올 수 있게 되었다. 추가적으로 pipe()명령문은 논리적으로 부모와 자식 프로세스 간의 통신 명령어이지만, 이를 활용하여 병형 프로세스간 통신으로 보이게끔 구현하였다. Client1에서는 pipe() 명령문을 이용한 Server로부터 보내는 Client2 메시지를, Client2에서는 그 반대의 Client1 메시지를 받음으로 최종적인 비동기적 Group Chatting 시스템을 구현하였다.**

1. **출력 결과**

* **아래는 Server와 Client1,2간 Group Chatting 출력 결과이다.**

**<Server 출력 화면>**

**<Client 1 출력 화면>**

****

**<Client 2 출력 화면>**

1. **Term Project#3 – P2P File Transfer**
2. **개요**

* **Term Project#2에서 비동기적 group chatting 시스템을 구현했다면 이를 활용하여 추가로 파일 전송 및 요청을 서비스적으로 구현할 수 있다. 파일 전송 및 요청을 다루기 위해서 group chatting 도중 예를 들어 Client1에서 ‘[FILE]’이라는 메시지를 입력했을 때 Server는 Client1의 IP와 Port 정보를 함께 Client2로 전송해준다. 여기서 Client1의 IP와 Port 정보를 같이 보내는 이유는 P2P 통신 서비스 구현을 위해서이다. P2P 통신이란 Peer To Peer 즉 동등한 관계에서의 통신을 의미하는데, P2P network에서는 Server – Client의 주종관계가 없다는 것이 그 특징이다. 즉 이는 Client1이 파일 전송을 요청하게 되면 Client1은 Server와의 통신을 끊고 Client2와 통신을 하게끔 Server의 역할이 된다는 것이다. Client2는 Server로부터 Client1의 IP와 Port 정보를 받았으면 이를 이용하여 Client1으로 접속하는 Client의 역할을 하게 된다. 즉 P2P 파일 전송을 하기 위해선 기존의 Server와의 연결을 해제하고 Client1,2가 새로운 Server – Client 통신 관계를 가지도록 구현하는 것이 첫번째 과제라고 볼 수 있다. 이렇게 Client1,2간 P2P network(새로운 Server와 Client)를 구축했다면 새로운 Server와 Client간 에 .txt 파일을 전송하고 받는 기능을 각각 알맞게 추가하는 것이 마지막 과제라고 볼 수 있다.**

1. **구현 방법**

* **먼저 group chatting 도중 Client1이나 2에서 ‘[FILE]’ 메시지를 입력하면 P2P network 모드로 들어가기 위해 Client1,2의 IP주소와 Port번호를 같이 보내기 위해 최초에 Client1과2가 Server에 로그인 할 때 Client1과 2의 IP와 Port번호를 등록한다. 따라서 예를 들어 group chatting 도중 Client1으로부터 ‘[FILE]’이라는 메시지를 Server가 입력 받게 되면, Client1의 IP와 Port번호를 Client2에게 전송한다. 이에 맞춰서 Client1은 fork() 함수를 이용하여 새롭게 생성한 프로세스에 Server와의 연결을 끊고 P2P network 모드의 Server로 작동하게끔 구현했다. Client2도 이에 맞춰 fork()함수로 새로운 프로세스를 생성하여 Server와의 연결을 끊고 P2P network 모드의 Client로 들어가게 되는데 이때 수신한 Client1의 IP와 Port번호를 통해 Server인 Client1으로 접속하게끔 구현하였다. 이렇게 P2P network를 구현하게 되면 Client1은 Server가 되고 Client2는 Client가 된다.**

**Client2에서는 파일 목록을 전송하는 기능 그리고 Client1에서 요청한 목록 번호를 받아 해당 목록에 있는 파일의 정보를 전송하는 기능을 가지도록 구현하였다.**

**Client1에서는 Client2에서 보낸 파일 목록을 받는 기능과 원하는 파일 목록 번호를 전송하는 기능, 그리고 마지막으로 받은 원하는 목록 파일의 정보를 토대로 파일을 생성하는 기능으로 구현하였다.**

**우리는 파일 목록을 불러오는 기능을 구현함에 있어서 리눅스 기반에서 디렉토리 파일을 다루는 헤더함수 dirent.h를 이용하였다. 헤더파일dirent.h에는 디렉토리의 구조를 구조체 dirent로 정의하고 있는데, 구조는 다음과 같다.**

**struct dirent {**

**long d\_ino; //l-노드 번호**

**off\_t d\_off; //offset**

**unsigned short d\_reclen; //파일 이름 길이**

**char d\_name[NAME\_MAX+1]; //파일 이름**

**}**

**이를 통해 지정된 디렉토리 스트림을 열어 처음을 가리키는 opendir 함수를 이용한다. 선언은 아래와 같다.**

**#include<dirent.h>**

**DIR \*opendir(const char \*dirname);**

**반환값**

* **성공 시: 디렉토리 스트림의 포인터**
* **실패 시: NULL**

**다음으로 디렉토리의 목록을 쭉 읽어가야 되므로 dirp(디렉토리 포인터)가 가리키는 디렉토리 내 파일 목록의 주소를 반환하고 dirp는 다음 파일목록으로 이동하는 readdir함수를 이용한다.**

**선언은 아래와 같다.**

**#include<dirent.h>**

**struct dirent \*readdir(DIR \*dirp);**

**반환값**

* **성공 시: 파일정보가 담긴 dirent 구조체**
* **실패 시: NULL**

**dirent.h 헤더파일을 통해 디렉토리를 열고 목록을 출력했으면 마지막에는 무조건 닫아주는 작업으로 FILE 구조체에서 fclose()와 비슷한 기능을 하는 closedir함수를 이용한다. 선언은 아래와 같다.**

**#include<dirent.h>**

**Int closedir(DIR \*dp);**

**반환값**

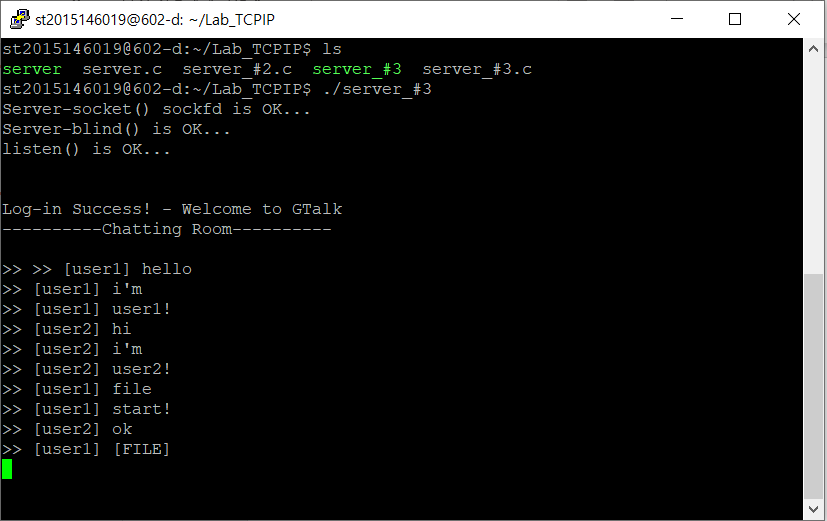
* **성공 시: 0**
* **실패 시: -1**

**이렇게 dirent.h 헤더파일부터 dirent 구조체를 통한 3가지 함수들을 이용하면 현재 디렉토리에 있는 파일들의 목록을 불러올 수 있다.**

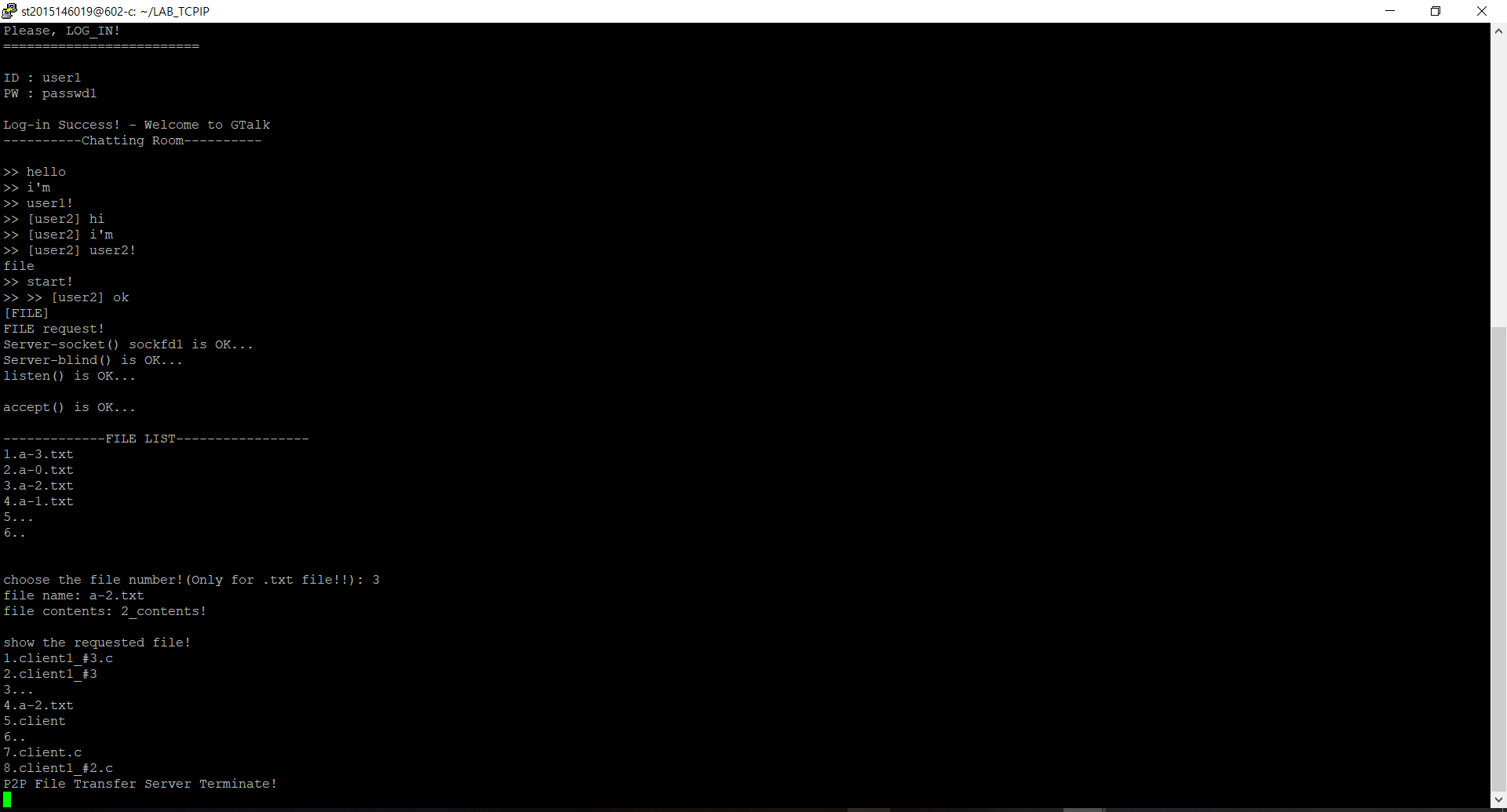
**다시 앞에 예시를 통해서 보게 되면 Client1이 P2P network에서 Server역할을 하고 Client2는 Client역할을 하는 구조로 되어있다. 즉 Client1이 ‘[FILE]’ 메시지를 보내 file 전송을 요청했으므로, Client2는 자신의 file목록을 Client1으로 전송한다. 이때 .txt 파일 생성은 FILE 구조체를 사용했으며, #include <sys/stat.h>을 선언하여 리눅스 기반에서 mkdir함수와 chdir함수를 이용하여 ‘FILE’ 이라는 이름의 폴더 내에서 .txt 파일을 생성하도록 구현하였다. Client1은 수신한 Client2의 파일 목록을 통해 원하는 번호를 입력하여 보내게 되면, Client2는 받은 번호를 통해 파일 목록에 해당되는 파일을 전송하면 된다. 우리는 파일을 전송하는 과정을 FILE 구조체의 fopen 함수와 fgets 함수를 이용하여 전송할 파일의 이름과 파일의 내용을 전송하는 방식으로 구현하였다. 파일의 이름은 번호에 해당되는 파일의 이름을 string의 형태로 send함수를 통해 전송하였고, 파일의 내용은 fopen(filename,”r”)와 fgets 함수를 이용하여 ‘buffer’라는 문자열에 넣어 send 함수를 통해 전송하였다. 따라서 Client1은 수신한 파일의 정보(이름,내용)를 통해 FILE 구조체로 fopen(filename,”w”)와 fprintf 함수를 통해 파일을 생성하는 방식으로 파일 수신 과정을 구현하였다.**

**이렇게 최종적으로 group chatting 중 P2P File Transfer 과정을 구현하였으며, 결과적으로 Client1과 2간 파일이 전송된 모습을 WinSCP 프로그램을 통해서도 확인할 수 있었다.**

1. **출력 결과**

* **아래는 Client1에서 ‘[FILE]’ 메시지를 입력하여 Client1이 Server가 되고 Client2가 Client가 되는 P2P network 구조를 가진 File Transfer 출력 결과이다.**

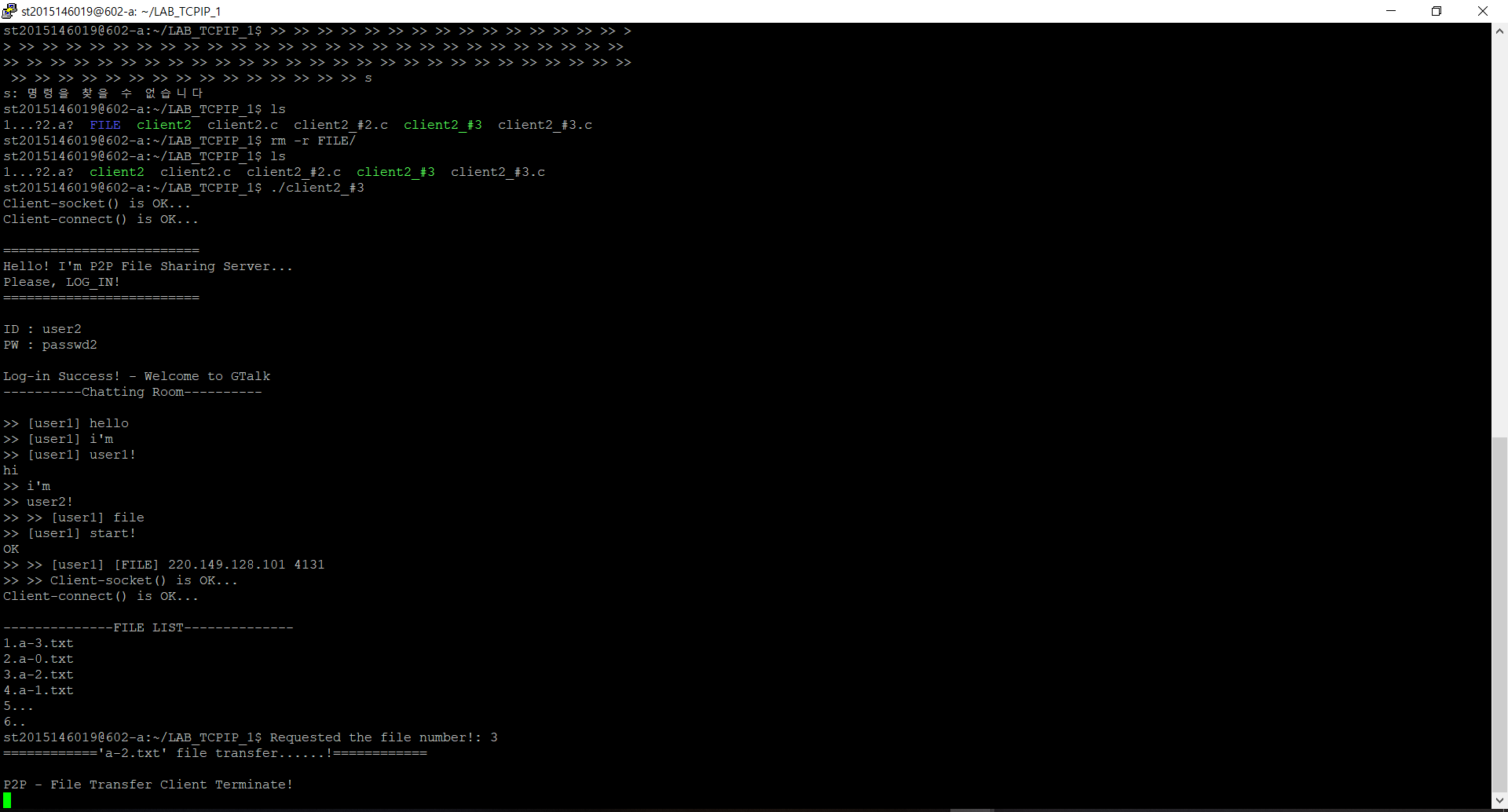
**<Server 화면,user1(Client1)의 [FILE] 요청>**

****

* **Client1의 [FILE]메시지 입력, P2P Server모드로 전환.**

**Client2에서의 파일 목록을 받아 3번 목록인 ‘a-2.txt’파일 요청.**

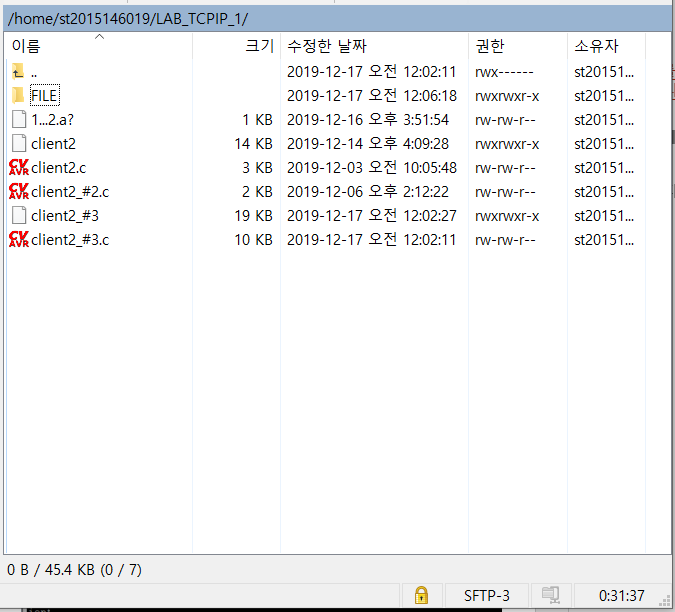
**수신한 ‘a-2.txt’파일이 있는지 파일 목록을 호출하여 확인.**

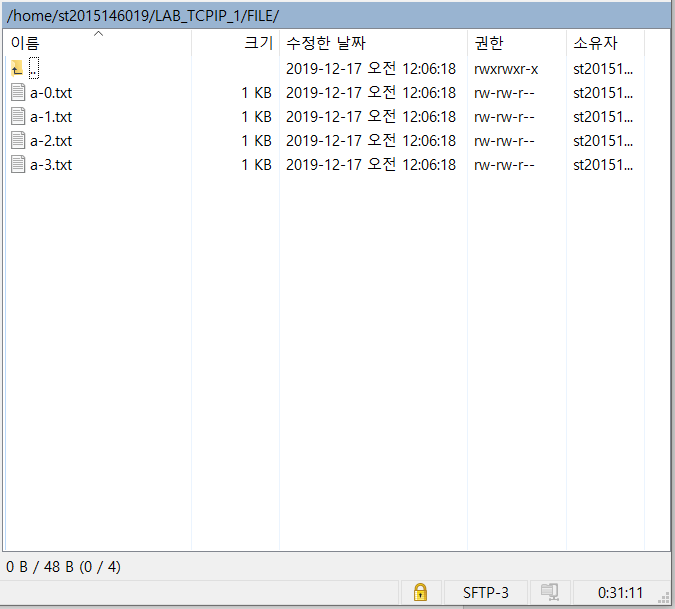
****

* **Client2는 Client1로부터 ‘[FILE]’ 메시지와 함께 Client1의 IP주소와 Port번호 수신하여 P2P Client모드로 전환.**

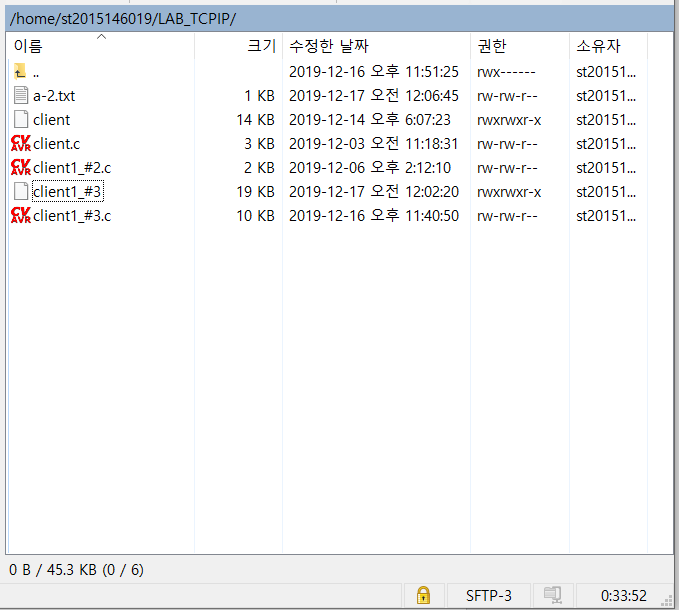
**Client2의 파일 목록을 Client1으로 전송.**

**Client1으로부터 수신한 요청 파일 목록 번호 출력 및 해당 파일 이름 및 내용 전송.**

* **WinSCP로 확인한 Client1,2의 P2P File Transfer 결과.**



**<Client2 화면, FILE폴더내에 .txt파일 생성화면>**

****

**<’a-2.txt’ 파일을 수신한 Client1 화면>**