최종 보고서

-P2P 파일 전송 프로그램-

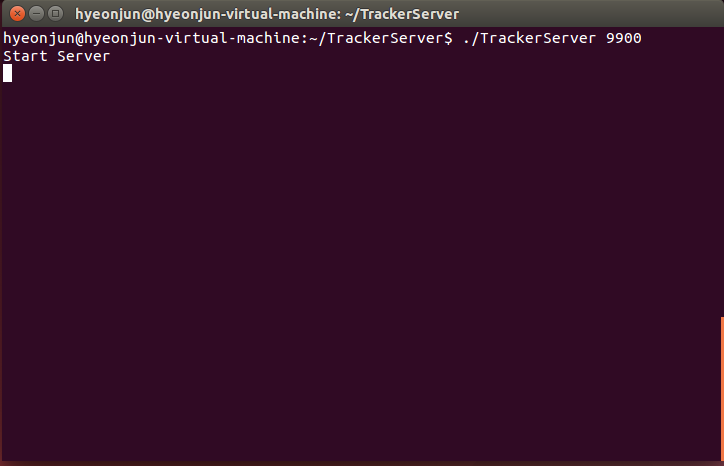
201321310 심현준

201321340 한재선

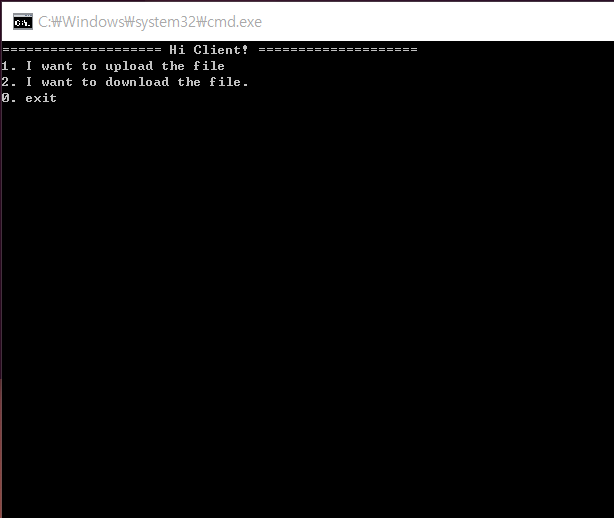
본 프로젝트는 네트워크 프로그래밍의 한 학기 프로젝트 입니다. 저희는 P2P 파일 전송(PC)와 P2P 파일 전송(Mobile)을 진행했습니다. P2P 파일 전송(PC)에서의 아파치 웹 서버와 중계 서버는 Linux에서 구축 및 구현 되었고, 클라이언트는 Windows와 Linux에서 구현 되었습니다. 소스코드는 클라이언트 1100 라인 이상, 중계서버는 200 라인 이상 정도가 됩니다. P2P 파일 전송(Mobile)에서는 중계서버를 두지 않고 디바이스끼리 통신을 하도록 하였습니다. 같은 Wifi상에서 상대방의 IP를 입력하여 파일을 전송하는 형태이며 Android에서 구현되었고 소스코드는 700라인 이상이며 그 외에 레이아웃과 리소스들로 구성이 되어있습니다.

먼저 P2P 파일 전송(PC) 프로그램에 대해 설명을 진행하겠습니다. P2P 파일 전송 프로그램이기 때문에 클라이언트들은 자신이 다운로드 받고자 하는 클라이언트의 IP/PORT를 알아야 합니다. 따라서 파일을 가지고 있는 클라이언트는 자신의 주소 정보를 중계 서버에 등록하고, 중계 서버는 파일을 다운로드 받고 싶어하는 클라이언트에게 파일을 가지고 있는 클라이언트의 주소 정보를 전달해주는 역할을 담당하게 됩니다. 하지만 여기서 끝내기 보다 클라이언트들이 자신이 다운로드 받고자 하는 파일에 대한 정보와 파일을 가지고 있는 클라이언트의 정보를 어떤 서버가 가지고 있는지를 알 수 있게 하자, 라는 목적에 의해서 아파치 웹 서버를 두어, 해당 정보들이 들어있는 파일을 주도록 합니다.

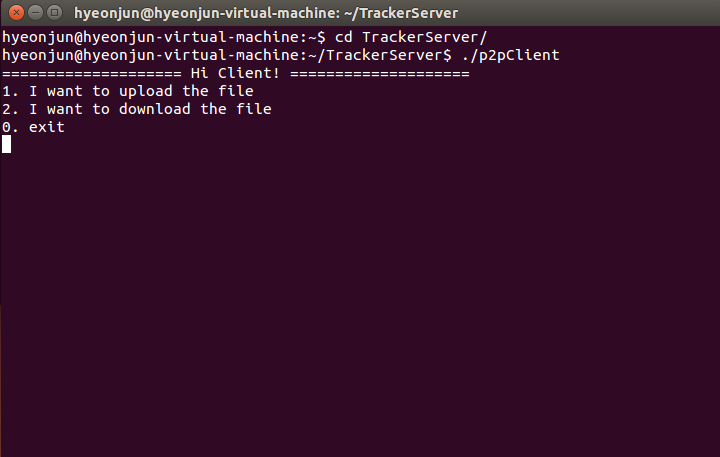
프로그램을 실행 시킬 때 먼저 아파치 웹 서버를 실행 시키고, 중계 서버를 실행 시킵니다. 그리고 두 플랫폼 중 하나로 클라이언트를 실행 시키면 업로드를 할 것인지 다운로드를 할 것인지를 묻는 프롬프트 창이 나오게 됩니다.



-중계 서버 시작-

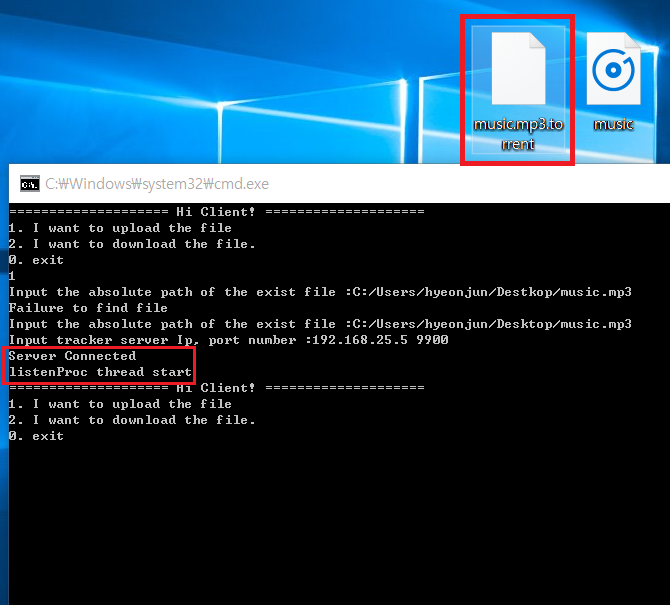


-윈 클라이언트 시작-



-리눅스 클라이언트 시작-

업로드를 하기 위해서는 1번을 클릭하게 되며, seed file을 만들 파일의 절대경로를 입력하게 됩니다. 그 다음에 자신이 up loader임을 등록하기 위한 중계 서버의 IP/PORT 번호를 입력하고, 서버에게 up loader가 되었음을 알려주는 번호를 부여 받게 됩니다. 따라서 seed file에는 파일의 총 바이트 수, 자신을 up loader로 등록하고자 하는 서버의 IP와 PORT번호, 실제 파일의 이름, 파일의 경로, up loader로 등록된 번호가 저장됩니다. 생성된 seed file을 웹 브라우저를 통해서 웹 서버에 등록을 하면 업로드의 준비과정이 끝나게 되며, 다운로드를 하려는 클라이언트의 연결 요청을 받기 위한 thread를 생성하게 되고, 연결 요청이 들어오게 되면 업로드를 하는 thread를 생성하게 됩니다.



-서버와 연결 후 torrent 파일을 만들고 연결 요청을 받기 위한 스레드 생성-

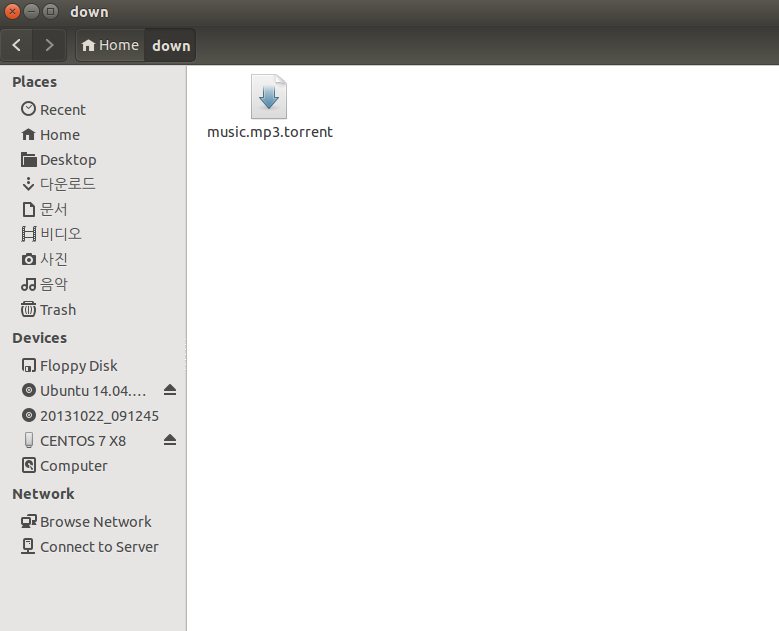


-파일을 업로드 하기 전 웹 브라우저 모습-

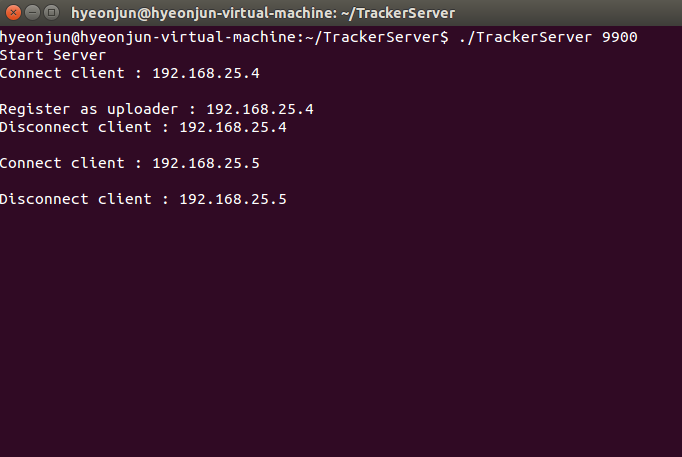


-파일을 업로드 한 후 웹 브라우저의 모습 -

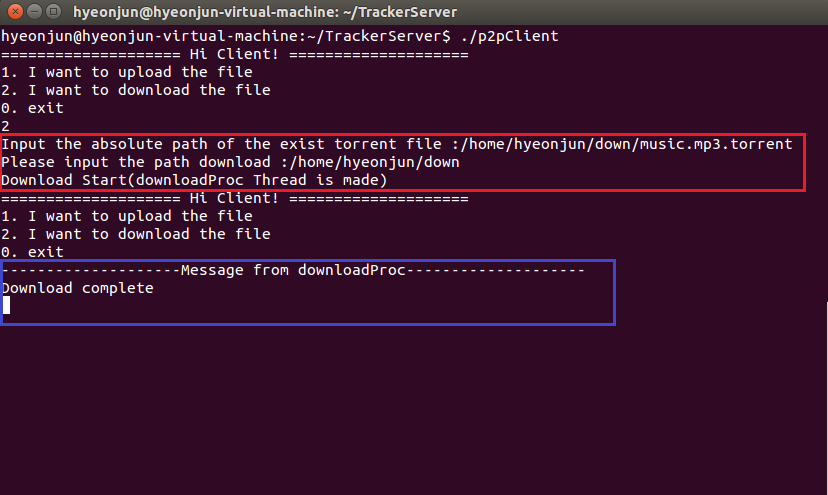
다운로드를 하기 위해서는 seed file이 필요하게 됩니다. 따라서 웹 브라우저를 통해서 웹 서버에 등록된 seed file을 다운로드 받습니다. 그리고 2번을 클릭하여 실행하면 seed file이 존재하는 절대 경로를 입력하고, 다운로드 받고자 하는 절대 경로를 입력합니다. 이 작업들이 끝나게 되면 seed file을 읽어서 다운로드 받고자 하는 파일을 가지고 있는 클라이언트의 주소를 알고 있는 중계 서버의 IP와 PORT번호를 얻게 되며, 중계 서버에게 연결요청을 합니다. 연결이 완료되면 중계서버에게 seed file에서 읽은 up loader의 번호를 전송하게 되며, 서버는 해당 up loader의 IP/PORT 번호를 전송해줍니다. 이러한 준비가 끝나게 되면 다운로드를 진행하는 thread를 생성하게 됩니다.



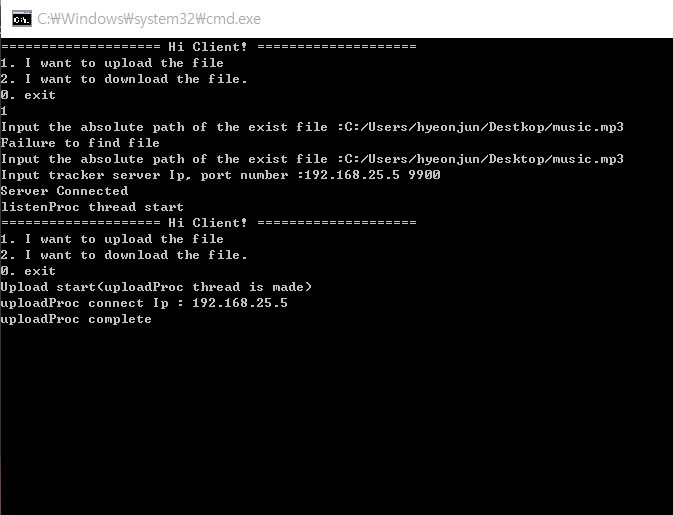
* 웹 서버로부터 Seed file 다운로드 -



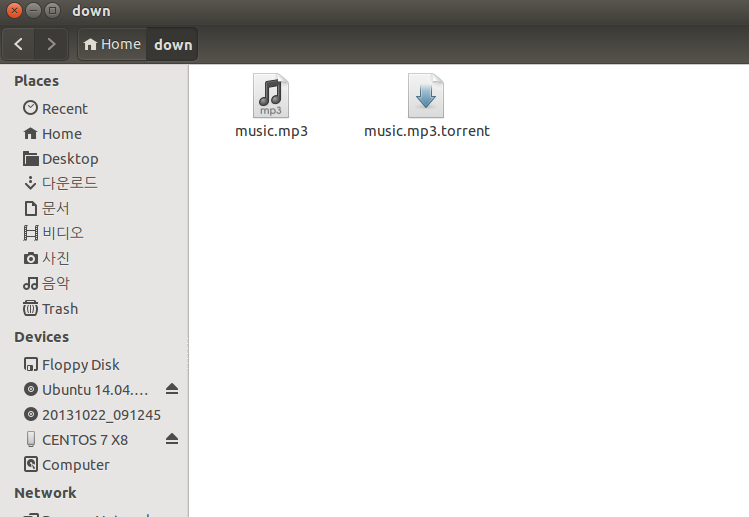
* 다운로드를 진행 할 클라이언트의 연결 후 서버의 모습 -



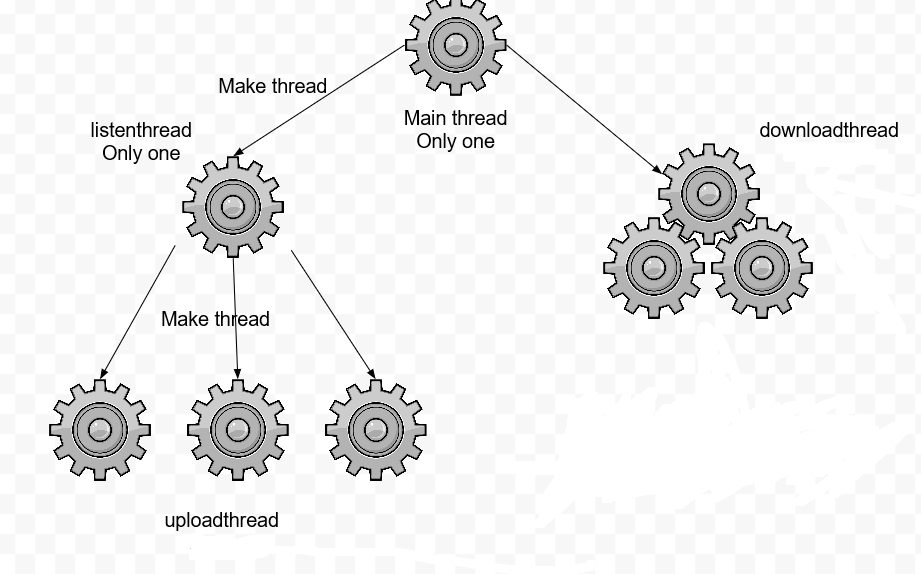
* 다운로드 완료 후 클라이언트의 모습-



* 업로드 완료 후 클라이언트 모습-



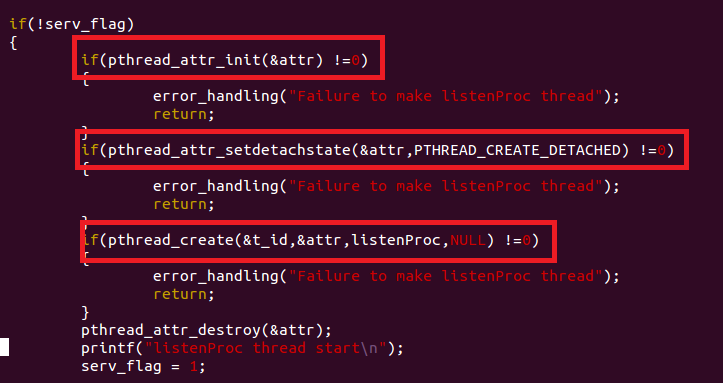
* 윈도우 클라이언트에게 다운로드 받은 파일의 모습-

P2P 파일 전송(PC)에서의 thread의 모델은 다음과 같습니다.

즉, main thread에서는 프로그램 설명에서의 업로드, 다운로드를 선택할 수 있게 하며 사용자의입력을 받아가면서 진행하는 업로드의 준비작업, 다운로드의 준비작업을 진행하게 됩니다. 업로드의 준비 작업이 끝나게 되면 main thread는 선택창으로 돌아와서 사용자에게 다른 작업을 진행 할 수 있도록 해주고, 클라이언트의 요청을 받을 수 있는 listen thread를 생성하여 블록상태에 진입하도록 합니다. 요청은 요청대로 받아가며, 업로드는 업로드대로 진행 할 수 있도록 upload thread를 생성해줍니다.

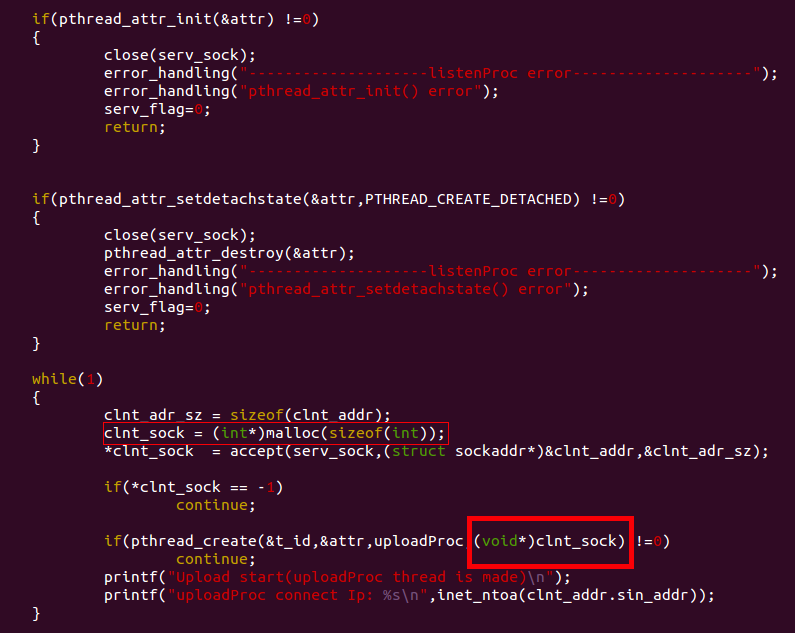
자신이 up loader가 될 수 있지만, down loader도 될 수 있다고 생각해서 업로드를 함과 동시에 다운로드를 진행 할 수 있도록 하기 위해서 다운로드를 담당하는 download thread를 생성해줍니다.

위의 그림과 같은 방식으로 프로그램을 만들기 위해서 몇 가지 부분을 신경 썼습니다. 수업에서 사용하던 책인 TCP/IP 프로그래밍에서는 잘 가르쳐주지 않은 부분이기 때문에 운영체제 시간에 배운 것을 생각해보면서 진행해 보았습니다. Chapter 18에서 주로 나타난 예제에서는 main thread가 흐름을 멈추고 자신이 생성한 thread가 종료할 때까지 기다려 주는 모습이 많이 보였습니다. Process 또는 Thread를 생성하고 난 다음에 종료가 되면 메모리에서는 내려 올 수 있어도, 커널에서 해당 Process 또는 Thread에 대한 정보를 가지고 있어서, 이것을 소멸시켜주는 함수나 혹은 자신이 생성한 Process나 Thread가 반환한 값을 읽어 들여서 소멸을 유도하는 방법을 쓰는 것으로 알고 있습니다. 하지만 그렇게 되면 main thread의 실행을 방해 한다, 라는 생각이 강해서 위의 그림과 같이 구현하기 위해서 어떤 방법이 좋을까 생각하던 도중에 생성된 thread를 main thread와 분리 시키는 것으로 선택했습니다. pthread\_detach() 함수는 main thread에서 생성된 thread를 분리시키고, 분리된 thread가 종료되면 스스로 소멸되는 기능을 합니다. 하지만 이 함수도 thread가 생성된 후에 이 작업을 진행하기 때문에, 만약에 이 함수가 실행 되기 전에 thread가 종료되어 버리면 기능을 적용 받지 못해서 소멸되지 않을 수 있다라는 생각에 thread를 생성하기 전에, 즉 pthread\_create()함수에서 thread의 속성을 부여하는 공간에 생성되는 thread는 main thread와 분리시킨다, 라는 값을 넣어주어서 생성됨과 동시에 분리되며, 종료되면 소멸할 수 있도록 진행 하였습니다.

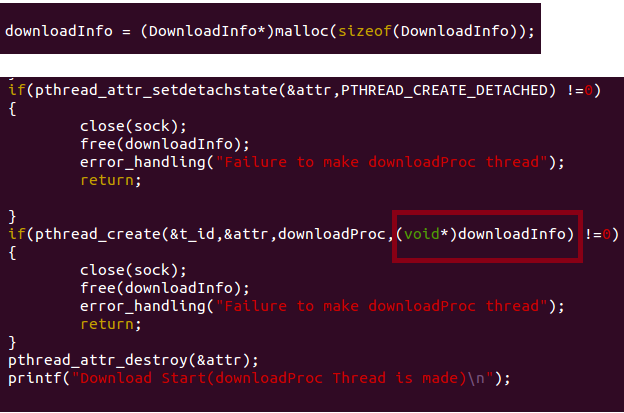


* Thread를 생성과 동시에 분리하기 위한 작업(대표적으로 Main에서 listen thread를 생성)-

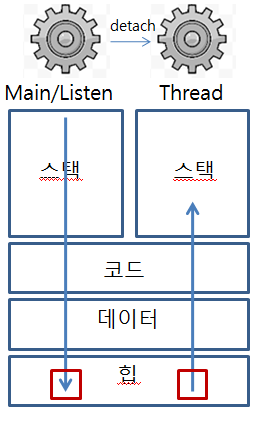
운영체제 시간에 배웠던 thread의 메모리 구조를 생각하면서 코드를 작성하였던 부분이 있습니다. 이 프로그램에서의 Main과 listen thread는 thread를 생성하면서 자신의 스택에서 할당한 변수에 대한 포인터 값을 넘겨주는 것이 아닌 하나의 프로세스에서 thread들이 공용으로 사용할 수 있는 힙 영역에 할당한 변수에 대한 포인터 값을 넘겨줍니다. 이렇게 진행한 이유는 만약에 Main, listen thread가 thread를 생성해서 자신의 스택에서 할당한 변수의 포인터 값을 넘겨주었을 때, 만들어진 thread가 실행되지 않고 Main과 listen thread의 작업을 수행하여 스택에 할당한 변수의 값을 변경하게 되었을 때, 아까 만들어진 thread가 수행하려고 할 때 변경된 값으로 수행하는 현상이 발생할 수 있다고 생각합니다. 따라서 저희는 스택 영역이 아닌 힙 영역에 변수를 할당하여 어떤 thread가 먼저 수행이 되었어도 전달해준 값에 변동이 없도록 하는 프로그램을 작성하였습니다.



* listen thread가 힙 영역에 할당하여 upload thread에게 전달하는 코드 –



* Main thread에서 힙 영역에 할당하여 download thread에게 전달하는 코드 -



* 힙 영역으로 thread에게 값을 전달 –

프로젝트를 진행하면서 1학년 때, 배워도 왜 배우는지 잘 몰랐던 파일 입출력에 대해서 다시 공부하였습니다. 컴퓨터는 바이너리 데이터를 다루기 때문에, 어떠한 파일이라도 binary stream로 연결한다면 모든 데이터를 주고 받을 수 있을 것이다, 라는 생각을 토대로 진행하였습니다. 여기서 한가지 응용을 하였는데, binary stream을 형성해서 파일을 읽어 들이면, 파일의 내용이 순서대로 들어오는 것을 1학년 때 관찰한 경험이 있어서 파일도 메모리 구조와 같이 offset이 존재 할 것이고 이것을 마음대로 조종하는 fseek()라는 함수를 써본 경험이 있어서 자신이 다운로드 받고자 하는 경로에 파일을 다운로드 받다가 중단 되었을 경우, 다시 up loader에게 연결 요청을 해서 파일을 다운로드 할 때, 이미 어느 정도 다운로드 받은 파일이 존재 하는 경로에 다시 이어서 다운로드 받고자 한다면, up loader에게 자신의 다음에 받을 파일의 offset을 전송하고 up loader는 down loader에게 offset 이후로의 파일 데이터를 전송해 주는 방법을 적용하였습니다.



* download thread에서의 파일 offset값 전송 -



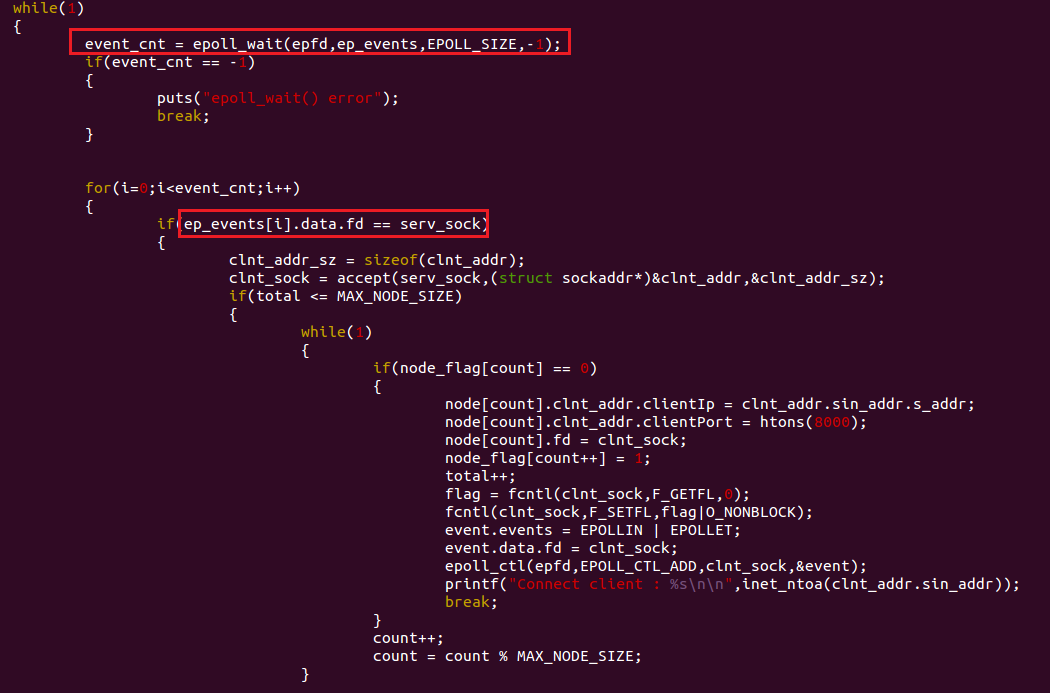
* upload thread에서의 파일 offset값 수신 후 fseek를 활용한 오프셋 이동 후 파일 전송-

중계 서버를 구현하기 위해서 Multi thread, Multi process, Multiplexing, 이 세가지 중 하나를 선택해야 했습니다. 최종적으로 Multi thread, Multiplexing에서 고민을 많이 했었는데, 간단히 생각해보면 Multi thread는 웹 서버와 웹 브라우저 같이 전송해야 할 양이 많은 경우에 적합하다고 생각하였고, 온라인 게임이나 혹은 주고 받는 데이터의 양이 크지 않을 경우에는 Multiplexing이 적합하다고 판단하였습니다.

Multiplexing에서도 select와 epoll이 있는데, select는 Linux와 Windows에 쓸 수 있는, 즉 이식성이 뛰어나지만 관찰할 대상을 운영체제에서 지속적으로 전달해줘서 오버헤드가 커집니다. 따라서 관찰대상의 범위나 변경사항만 전달하자라는 목적으로 select보다 이식성은 떨어지지만 상대적으로 성능이 좋은 epoll을 선택하게 되었습니다.

epoll에서도 레벨 트리거와 엣지 트리거가 존재하는데 레벨 트리거는 이벤트가 발생했을 시 처리하지 않으면 지속적으로 알림을 발생시키게 됩니다. 따라서 처리를 중단하고 다른 작업을 수행하게 되면 알림이 쌓이게 되고, 오버헤드 발생으로 인해서 성능이 떨어지게 됩니다. 그에 반해 엣지 트리거는 이벤트가 발생했을 시 한 번만 알림을 발생하여, 이벤트 처리와 자신이 수행하고자 하는 일을 분리하여 할 수 있다는 장점을 가지고 있어서 엣지 트리거 방식을 선택하게 되었습니다.

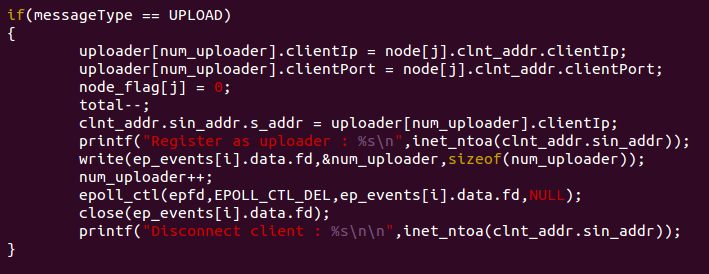
중계 서버의 Multiplexing 구조에서 연결 요청에 대한 것은 연결을 함과 동시에 해당 클라이언트 들의 IP와 PORT번호를 저장하게 됩니다. 왜냐하면 중계 서버의 존재 목적은 up loader의 IP/PORT를 down loader에서 전송하는 것인데, 처음 연결 요청을 받을 때는 클라이언트가 down loader인지 up loader인지를 구분하기 어렵기 때문입니다.



* 중계 서버의 서버소켓에 관한 이벤트 발생시 수행하는 작업 –

위 코드를 보게 되면 서버에 대한 연결 요청이 왔을 때 임시공간에 클라이언트의 IP와 PORT번호를 입력하게 됩니다. 여기서 flag배열을 두게 되는데, 만약에 임시공간에 자리가 없을 경우에는 더 이상 요청을 받을 수 없지만 임시공간에 자리가 있을 경우, 예를 들어서 0~ 1023번째까지 공간이 있다면 1023까지 찼다고 가정하였을 때, 앞자리에서 비어있는 공간이 존재할 가능성도 있습니다. 따라서 원형 큐를 모델로 삼아서 flag배열의 인덱스와 임시공간의 인덱스를 맞춰서 해당 임시공간에 자리가 비어있는지 확인하게 되고 비어있지 않으면 count값을 증가시키되 MAX값에 도달하면 %를 이용해서 다시 0부터 수행할 수 있는 구조를 생각했습니다.

연결 요청 후, 클라이언트에게 Upload에 관련된 것인지, Download에 관련된 것인지에 대한 메시지를 받게 됩니다. 해당 메시지에 따라서 수행하는 작업이 다르게 되는데 Upload같은 경우는

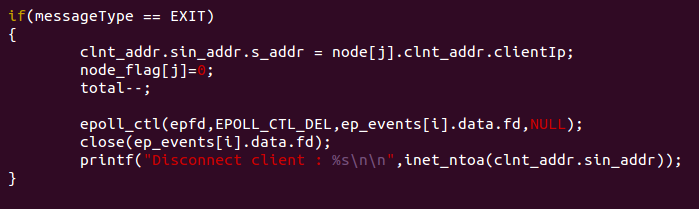


위의 코드와 같이 임시공간에 저장되었던 해당 클라이언트의 IP와 PORT번호를 업로더 공간에 저장하게 됩니다. 업로더 공간에 저장되었을 경우 임시공간을 비워줘야 하기 때문에 flag배열에 0의 값을 대입하게 되고, 등록된 업로더 공간의 인덱스를 클라이언트에게 전송해줍니다.

Download 경우에는 클라이언트가 seed file을 읽어서 업로더의 번호를 서버에 전송하고 서버는 그 번호를 인덱스 값으로 사용하여 해당 업로더 클라이언트의 IP와 PORT번호를 전송해 줍니다.

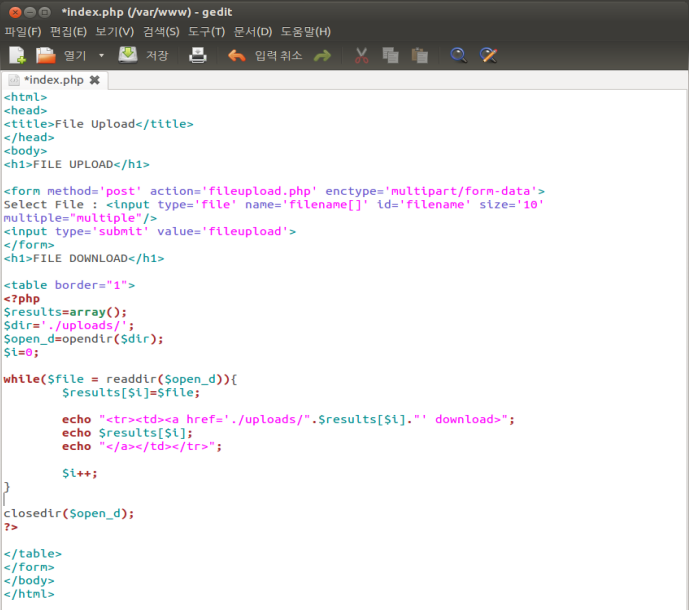


여기서 조금 더 신경 썼던 부분은 만약에 클라이언트가 연결을 한 후에 오류로 인해서 소켓을 닫았을 경우, 메시지 타입의 어느 쪽도 속할 수 없게 됩니다. 하지만 지금까지 배운 TCP/IP 프로그래밍에서 생각해 본다면 연결을 닫았기 때문에 서버의 버퍼 입장에서는 파일 전송의 끝을 알려주는 0의 값이 들어오게 되며, Multiplexing 구조 상 버퍼에 값이 들어왔으니까 read함수를 호출 할 것이고 그러면 0 값을 읽을 수 있게 됩니다. 0일 경우에서는 다른 if문을 적용하여 소켓을 닫아주는 작업을 별도로 진행하게 됩니다.



* 위 설명에 해당하는 코드 -

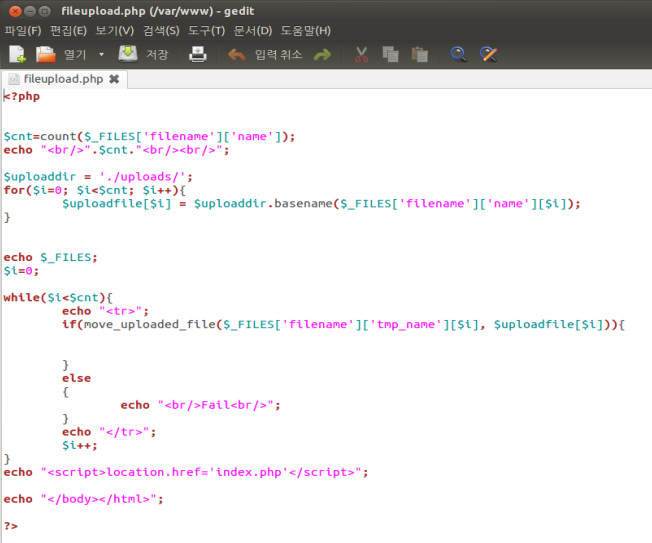
Seed file을 업로드하고 다운로드를 하기 위해서 Linux에서 apach2 를 실행시키고 웹 브라우저에서 index.php로 접속합니다.



3)

2)

1. Input 요소에서 name 속성값을 배열로 지정하고, multiple 속성을 사용하여 사용자가 파일을 선택할 때 뜨는 파일 탐색기에서 파일을 여러 개 선택할 수 있습니다. 사용자가 선택한 파일들은 post 방식으로 fileupload.php 로 전송합니다.
2. apache의 최상위 루트인 www(윈도우에서는 C:\wamp\www , Linux에서는 /var/www) 에 uploads 라는 하위 디렉토리를 $dir 변수로 지정합니다. Opendir 함수를 이용하여 해당 디렉토리를 엽니다
3. index.php에서 uploads 디렉토리에 존재하는 모든 파일을 테이블 형식으로 출력합니다.(Linux에서는 기본적으로 /var/www 폴더에 접근할 수 있는 권한이 없기 때문에 접근하려는 디렉토리는 권한을 별도로 설정해주어야 합니다.) 이 때 테이블 각 행은 uploads 디렉토리에 존재하는 각각의 파일들을 링크 형식으로 지정한 것입니다. 클라이언트가 이 링크를 클릭하면 다운로드 할 수 있도록 링크에 download속성을 넣었습니다.



4)

3)

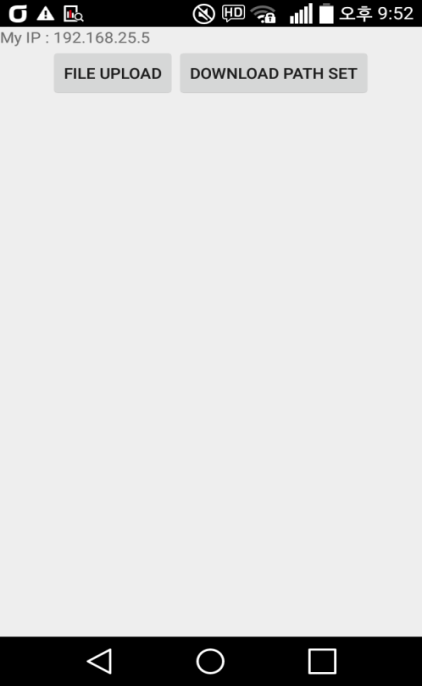
2)

1)

1. 사용자가 선택하여 전송한 파일들의 개수를 count하여 $cnt에 넣습니다
2. 업로드할 파일들을 uploads 디렉토리에 업로드하기 위해 $uploaddir 에 지정했습니다.
3. 선택한 파일들을 $uploadfile 배열에 넣습니다.
4. 선택한 파일들을 move\_upload\_file을 사용하여 uploads 디렉토리에 업로드합니다. 모두 업로드 하고 나면 index.php 를 불러와서 페이지를 갱신하여 지금 업로드한 파일들까지 모두 테이블에 출력해줍니다.

P2P 파일 전송(Mobile)의 시작은 Bluetooth를 모티브로 삼은 것입니다. Wifi를 이용한 기기간의 직접적인 통신으로 동영상, 음악 등의 파일을 1대1이 아닌 다수 대 다수로 전송할 수 있는 방안을 생각하게 되었습니다.

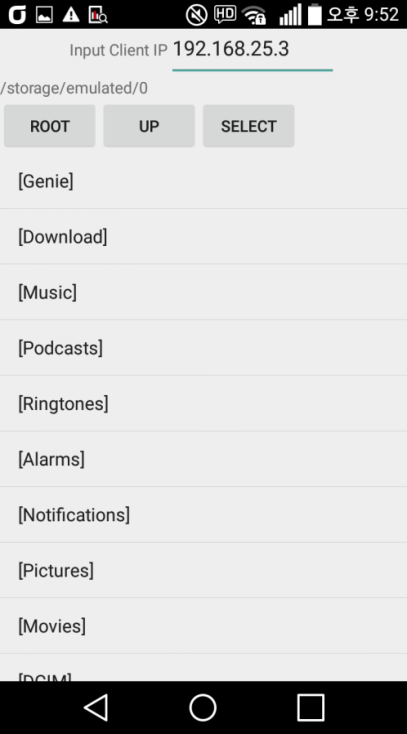
먼저 P2P 파일 전송(Mobile)에서의 동작방식과 PC와 다른 UI구성에 대해서 스크린샷 을 통해 설명을 진행하겠습니다. 앱을 처음 실행하게 되면 Main UI thread(Main activity)가 보이게 되고 listen thread를 생성하게 됩니다.



* Main activity -

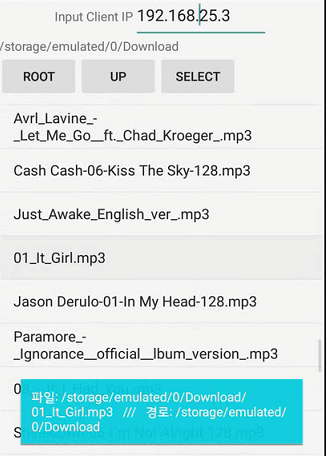
Listen thread를 곧바로 생성하는 이유는 파일 업로드의 개념이 PC와 다르기 때문입니다. 앞에서 Bluetooth를 모티브로 삼았다고 했었는데, 어떤 기기가 다운로드 받고 싶은 파일을 선택하는 것이 아니라 다른 기기가 업로드를 하고 싶은 파일을 선택하는 것에 초점이 맞추어져 있기 때문입니다. 따라서 어떤 기기가 통신을 요청할지 모르기 때문에 Listen thread를 곧 바로 생성하게 되며 연락 요청이 오게 되면 download thread를 생성해서 통신을 하게 됩니다.

사용자가 파일을 업로드 하려고 한다면 FILE UPLOAD버튼을 클릭하게 됩니다. FILE UPLOAD 버튼 클릭 시 Upload activity가 호출되고 다음과 같은 모습을 띄게 됩니다.



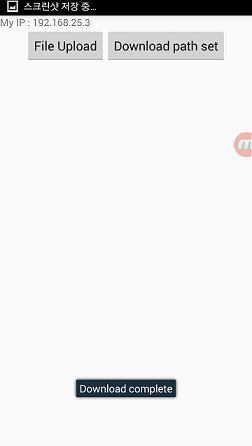
* Upload activity -

P2P 파일 전송(PC)에서의 불편했던 점은 GUI방식이 아니기 때문에 경로를 직접 입력해야 한다라는 점이 가장 큰 문제였습니다. 따라서 Android에서는 GUI방식이 PC보다 만들기 쉬웠기 때문에 사용자가 좀 더 사용을 편하게 할 수 있는 방안을 생각하게 되었고, 현재 기기에서의 모든 디렉토리와 파일의 이름을 보여줄 수 있는 파일 리스트 뷰를 생각하게 되었습니다. 처음 보이는 화면은 디렉토리들이며 해당 디렉토리를 누르면 그 디렉토리 내부의 파일 및 디렉토리를 보여주게 됩니다. 디렉토리는 선택할 때 마다 내부에 대한 정보를 리스트 뷰로 보여주지 만 파일을 선택 할 때는 해당 파일의 경로명을 토스트로 나타나게 하였습니다.

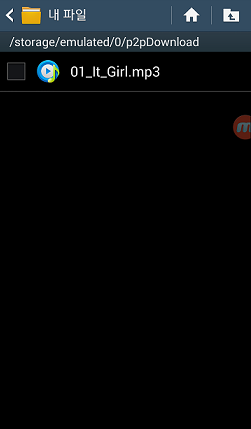


그 다음 select버튼을 클릭하게 되면 파일 전송이 시작됩니다.

다운로드를 받는 기기에서는 다운로드가 완료되면 다운로드 완료 토스트를 띄우게 되고 경로를 설정하는 버튼을 통해서 경로를 변경시키지 않았다면 디폴트 경로인 p2pDownload 디렉토리에 파일이 저장됩니다.

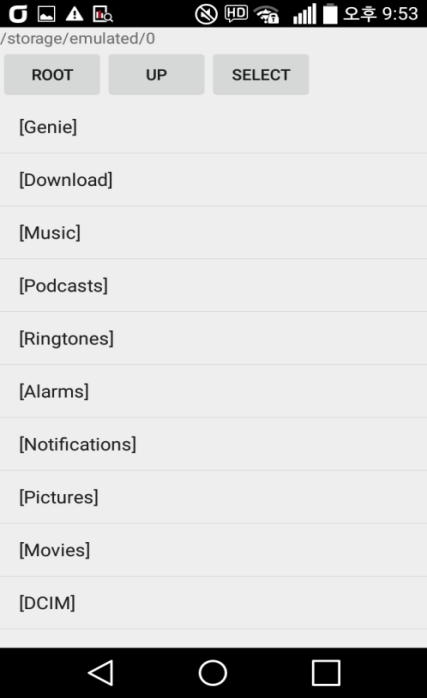


* 다운로드 완료 –



* 다운로드 받은 파일 –

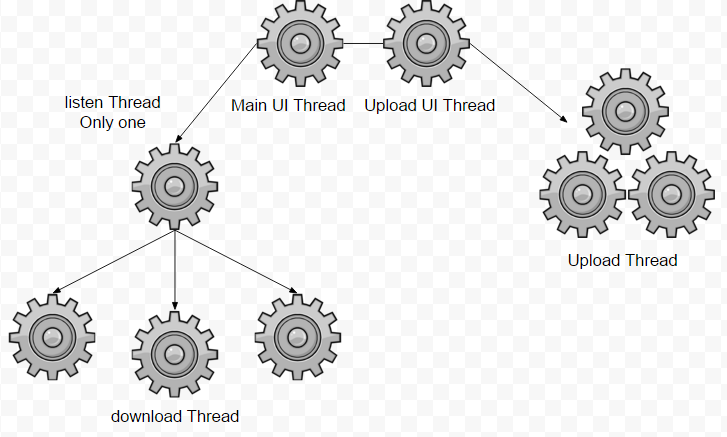
만약 디폴트 경로에 다운로드 받기를 원하지 않는다면 DOWNLOAD PATH SET을 눌러서 경로를 바꾸는 작업을 수행하면 됩니다.



* Download 경로 변경 –

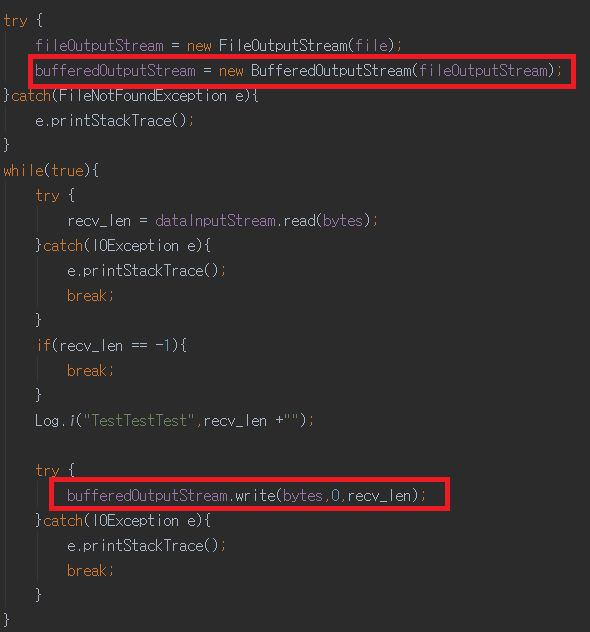
원하는 디렉토리를 클릭하여 select를 누르게 되면 다운받는 경로가 변경됩니다.

P2P 파일 전송(Mobile)의 thread 모델은 다음과 같습니다.

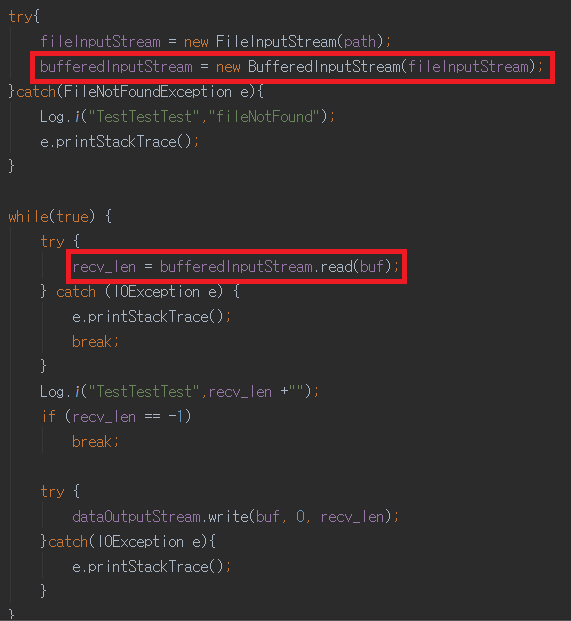


UI스레드의 수행과 별개로 download와 Upload thread가 생성되기 때문에 위의 동작 방식과 같은 과정을 거친 후 다른 기기의 IP를 입력하여 다른 기기에 파일을 바로 전송 할 수 있다는 장점을 가지고 있고, 업로드 activity를 통해서 업로드 할 파일을 선택하는 것과 상관없이 download thread로 인해서 파일을 전송 받을 수 있는 여러 개의 작업을 동시에 할 수 있다는 장점도 가지고 있습니다.

Thread를 사용하면서 신경 썼던 부분이 있다면 메모리와 보조기억장치에 대한 것입니다. 컴퓨터 구조 시간에서 파일의 내용을 메모리로 불러들여서 내용을 수정하였을 때, 수정 된 내용에 대해서 보조기억장치에 계속 접근하여 수정하는 것이 부담이 되기 때문에 나중에 수정 된 내용을 저장하는 방식에 대해서 공부한 적이 있었습니다. 그것을 통해서 알게 된 것이 메모리와 보조기억장치간의 접근이 자주 일어나면 성능저하가 발생할 수 있다라는 것이었고, 운영체제 시간에도 메모리 공간이 부족하였을 때 보조기억장치에 잠시 내려놓고 수행하는 작업에 대해서 공부한적이 있었는데, 이 두 가지의 과목에서 배운 내용을 접목해보면, 1KB의 배열을 통해서 보조기억장치에 있는 파일의 내용을 읽어서 1KB를 전송해주는 작업보다 2MB의 공간을 통해서 보조기억장치에 있는 파일을 읽고 전송해주는 작업이 더 좋을 수 있다는 생각을 가지게 되었습니다. 2학년 때 Java를 수강하면서 BufferOuput과 BufferInput에 대한 내용을 접한 적이 있었고, 그 내용을 가지고 프로그램에 적용시켰습니다. BufferOutput과 BufferInput은 Default size로 2MB를 가지고 있으며 따로 read write를 수행하지 않아서 스스로 버퍼를 채우는 역할을 합니다. 따라서 모바일 기기가 PC보다 성능이 떨어진다는 것을 생각하였을 때 제가 생각한 방법이 성능을 좀 더 높이는 것에 도움이 될 수 있을 것입니다.



* Download Thread중 일부 -



* Upload thread 중 일부 -

이번 프로젝트는 1학년 때 배운 파일 입출력에 관한 내용과 2학년 때 배운 컴퓨터 구조, 웹 프로그래밍, Java, Android Programming과 3학년 때 배운 운영체제, 네트워크 프로그래밍의 내용들을 같이 접목 시킨다는 것이 가장 큰 의미를 가지는 것 같습니다.