**Computer Networking HW 3**

**FTP implementation using UDP sockets – 14조**

1. **UDP**

서버와 클라이언트를 UDP 소켓을 사용하여 구현한다. UDP는 대표적인 Unreliable 전송 방식으로, 서버와의 연결이 필요없이 상대방의 소켓만 알고 있다면 상대방에게 데이터를 보낼 수 있다.

1. **구현**

FTP 서버(Fserver.c)는 시작하면 루프 안에서 클라이언트의 접속을 기다린다. 소켓을 만든 후, 루프 안에서 recvfrom으로 클라이언트의 접속을 기다리며, 클라이언트의 명령어에 따라서 파일을 전송하거나 받을 수 있다. 단, 처음에 만든 소켓은 클라이언트의 명령어를 받는 데만 사용하며, 클라이언트의 명령어에 따라 파일을 보내거나 받아야 할 때는 별도의 소켓(별도의 포트 번호 사용)을 열어, 그곳에서 병렬적인 스레드를 열어 수행한다. 이는 pthread로 구현할 수 있고, 컴파일을 할 때는 **gcc에 –lpthread 옵션을 주어야 한다.**  
 FTP 클라이언트(Fclient.c)는 시작하면 루프 안에서 명령어를 받는다.

|  |  |
| --- | --- |
| connect [IP] [SOCKET] | 주어진 IP와 포트에 연결. 이는 서버가 기본으로 작동하는 포트로, 명령어를 주고받는 포트이다. |
| quit | 프로그램을 끝낸다. |

이 안에서 connect를 통해 접속을 할 경우, 서버의 명령어 소켓에 접속(정확히는 연결하는 방식의 접속이 아니라, 그 소켓의 정보를 저장하여 준비하는 것이다. UDP는 연결을 하지 않는다.)하게 되며, 다음과 같은 명령어를 사용 가능하다.

|  |  |
| --- | --- |
| get [FILENAME] | 서버가 가지고 있는 파일을 클라이언트로 받아온다. |
| put [FILENAME] | 클라이언트가 가지고 있는 파일을 서버에게 보낸다. |
| close | 연결을 끝낸다. |

서버와 클라이언트는 별개의 디렉토리에서 돌아가게 하는 것이 좋다.

1. **소켓 방식**

기본적으로 서버는 하나의 소켓을 만들고, 이를 명령어 (get, put)를 받는 데 사용한다. get이나 put 명령어를 받으면 서버는 **파일 전송용 소켓**을 하나 더 만들며, (기존 port 번호에서 +1 씩 더한 포트 번호를 사용하는 방법을 사용한다.) 이렇게 새로 만든 소켓 번호를 클라이언트에게 알려주게 된다. 서버는 이 파일 전송용 소켓을 만들어 두고 있고, 클라이언트는 받은 소켓 번호를 사용해 그 소켓을 저장한다.(IP는 기존 서버의 IP를 그대로 사용한다.)

|  |
| --- |
|  |
| 처음에 ./Fserver 44444로 소켓을 만들면, 이 44444 소켓은 여러 클라이언트가 동시 공유. 서버는 루프를 돌면서 각 클라이언트의 명령어를 처리해 준다. |
|  |
| 각 클라이언트의 요청에 따라서 새 스레드, 새 소켓을 만들어 클라이언트와 연결시킨다.  즉, 클라이언트는 명령어 소켓과 파일 전송 소켓, 2개의 소켓을 알게 된 셈이다.  파일 전송이 끝나면 클라이언트와 서버는 그 소켓을 닫아버린다.  서버는 처리하는 클라이언트 수만큼 소켓을 운용하고 끝나면 제거한다. |

클라이언트는 한 프로세스가 한 클라이언트를 의미하지만, 여러 클라이언트를 받을 필요가 있는 서버의 경우는 각각의 파일 전송은 스레드를 사용해 수행한다. put이나 get 명령어를 받는다면 특정 구조체에 클라이언트의 정보와 새 소켓의 정보를 담은 뒤, 그것을 스레드의 매개변수로 넣어 사용하면, 스레드를 작동시킬 수 있고, 서버는 다시 명령어를 받는 루프로 돌아와 다른 클라이언트의 요청을 동시에 처리 가능하다.

1. **데이터 전송**

UDP는 TCP와 많은 차이가 있어 대용량의 파일을 반드시 전송 가능하다는 보장이 없다. 이를 해결하기 위해 전송 방식을 고안했다. 중요한 건 데이터 전송 순서로, 이 실습은 localhost끼리의 FTP를 다루므로, 데이터가 손상된다는 걱정을 배제한다.  
 파일을 보내는 함수는 SendData(), 받는 함수는 ReceiveData()를 사용한다. 서버나 클라이언트는 모두 이 두 함수를 가지고 있어야 한다.(물론 서버냐 클라이언트냐에 따라서 이 함수의 부가적인 모습은 조금씩 차이가 있다. 서버의 경우 멀티스레딩을 사용하므로 매개변수를 pthread에서 요구하는 대로 특정 구조체에 넣어야 하며, 클라이언트는 함수의 매개변수로 직접 넣을 수 있다.)  
 고안한 방식은 RDT 이론과 GO-BACK N 이론을 적절히 혼합한 방법을 사용하기로 한다.

|  |
| --- |
| * SendData()는 ‘윈도우’를 운용한다. 0번부터 많게는 수만 개까지의 윈도우를 힙 영역에 만들어 사용한다. 처음에 이 윈도우는 전부 0 값으로 초기화되어있다.  윈도우의 각 인덱스는 데이터의 seq 번호에 대응된다. * SendData()는 보낼 파일에서 일정 바이트(상수로 DATA\_BLOCK에 선언된 값 만큼)를 가져온 후, 버퍼에 넣은 후 그 앞에 seq 번호를 넣는다. * ReceiveData()는 파일을 받게 될 경우, 그 파일 앞에 들어있는 seq 번호를 추출하며, 원하는 seq 번호가 맞다면 그 seq 번호를 갖는 ACK(n)을 보낸 후, seq를 제외한 파일 데이터를 추출하여 파일에 쓴다. 그 후, 다음 데이터를 기다린다. * SendData()는 파일 데이터를 보낸 후 recevfrom으로 ACK(n)를 받을 때까지 기다린다. ACK(n)를 받을 경우, 그곳의 n을 추출하여 원하는 seq과 일치하는지를 확인. 일치하다면 n번째 윈도우에 1 값을 넣은 후, 다음 seq으로 넘어가 반복한다. * SendData()는 보낼 파일의 끝까지(eof) 도달했다면 endoffile 메시지를 보낸다. ReceiveData()는 그걸 받았다면 파일 디스크립터를 닫는다. * 각 호스트들은 소켓을 닫고 스레드 종료. 클라이언트는 다시 명령어 창으로 돌아온다. |

Stop\_and\_wait 방식을 베이스로 하여, 송신자가 데이터(n)를 보낸 후 ACK를 기다리며, 수신자는 그 데이터(n)을 기다리다가, 데이터를 받게 되면 읽은 후 ACK(n)을 보낸다. 이런 식으로 한 블록 한 블록 쌓아서 파일을 전송을 완료한다.  
 Localhost 전송을 다루었기 때문에 에러 체크는 구현되지 않았고, retransfer 함수가 있긴 하나 개념상 넣었을 뿐, 실제로 저것은 구현되어있지 않다.

1. **작동 화면**

|  |
| --- |
|  |
| **클라이언트 3명이 접속.** |

위 사진은 클라이언트 3명이 접속하여, 2명은 put 명령어를 통해 클라이언트 폴더 내에 있는 파일을 서버에 저장하려 하며, 1명은 get 명령어를 통해 서버가 가진 파일을 받아오려고 한다.

|  |
| --- |
|  |
| **서버는 1초 간격으로 전송 상황을 표시** |

서버는 3개의 스레드(클라이언트 1을 위한 ReceiveData(), 클라이언트 2를 위한 ReceiveData(), 클라이언트 3을 위한 SendData())를 실행한다. 이 순간에도 서버 프로세스 자체는 계속 루프에서 명령어를 받을 수 있기 때문에 멀티 클라이언트 구현이 가능하다. (기본적으로 명령어를 주고받는 소켓은 서버가 살아있는 한은 계속 유지된다.)

|  |
| --- |
|  |
| **하나의 파일 전송 완료** |

위 사진에서 클라이언트 2의 파일이 전송 완료되었다. 서버는 이를 출력한 후, 그 스레드를 닫게 되며, 이 ‘test8.zip’을 위해 생성한 소켓은 이제 소멸된다. 다른 두 파일은 아직도 송/수신되고 있다. 클라이언트는 다시 명령어 루프로 돌아가며, 이때 또다른 요청을 하는 것도 가능하다.

|  |
| --- |
|  |
| **전송에 성공하였다.** |

클라이언트는 서버에세 1GB 가량의 파일과 100MB 가량의 파일을 보냈고, 서버로부터 200MB 가량의 파일을 받았다. 원 파일과 비교하니 손상이 없었다.  
 이 코드에서는 한 번에 파일을 20바이트씩 전송하도록 블록값을 설정했으나, 두 소스 파일의 코드 맨 위에 상수로 정의된 DATA\_BLOCK을 수정하면 한 번에 전송 가능한 데이터의 크기를 설정 가능하다. 단, 이 프로그램들은 에러 감지가 구현되어있지 않아서 성능이 다소 떨어지는 컴퓨터의 리눅스에서 지나치게 크기가 큰 바이트를 한번에 전송하면 손상이 생길 위험이 있다.  
  
(+) 윈도우는 대략 500000000정도의 크기를 사용한다. Char 배열로 다룬다.  
(+) 접속 가능한 클라이언트는 대략 30명 정도로 설정했다. 각 클라이언트가 보낸 명령어를 받아 서버는 그 클라이언트의 정보를 구조체에 저장한 후 스레드에 전달한다. 접속한 클라이언트 수를 다루는 int 변수도 서버가 가지고 있으며, 각 스레드는 이 변수의 포인터 값을 가져가서 스레드가 끝나면 이 변수을 1만큼 줄여, 클라이언트가 작업을 끝넀다는 것을 알린다.

차동민 (조장) : 프로그램 설계/구현, 보고서 작성

이동우 : 토론

유세프 : 토론

이재열 : 토론

원윤주 : 토론