<Socket Programming Report>

컴퓨터네트웍개론

SWE\_3022

2015310464 원종원

1. 통신 성공 스크린샷

텍스트, 스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명1)IPV6\_SERVER

텍스트, 스크린샷, 모니터, 전자기기이(가) 표시된 사진

자동 생성된 설명2)IPV4\_CLIENT

2. 전달받은 토큰

RANDOM1:77777445302152224762

RANDOM2:98689825037687481215

RANDOM3:88580942878359415853

RANDOM4:57027579122965182410

RANDOM5:98274512274821551502

3. 특별한 경험

1)send() 와 recv() 및 sleep()

서버를 구현하면서 가장 에러가 많이 났던 부분이 바로 recv()였습니다.

Ipv4 client 가 Ipv4 server 로부터 recv해오면서 두번에 걸쳐서 받아오거나, 코드가 실행중에 recv()상태에 들어가서 무한히 대기중인 상황이 있었습니다.

Ipv4 server 에게 정상적으로 메시지를 보내고 나서, recv가 제대로 되지 않았던 것은, 서버와 통신하는 동안 생기는 delay를 생각하지 못했기 때문이었습니다. 우리가 사람과 전화통화를 할 때에 내가 발언한뒤 상대방에게 대답을 듣기까지의 delay가 생기듯이, 통신 프로그램 또한 이러한 delay를 계산해주어야 한다는 것입니다. 즉

(client 가 server로부터 메시지를 듣고 새로운 메시지를 생성하여 전송하는 데까지 걸리는 딜레이) + (client 메시지가 네트워크를 통해 server로 전송되는 동안 걸리는 딜레이)

+ (server가 client로부터 메시지를 듣고 새로운 메시지를 생성하여 전송하는 데까지 걸리는 딜레이)

+ (server 메시지가 네트워크를 통해 client로 전송되는 동안 걸리는 딜레이)

가 총 딜레이 이므로, client 입장에서 send()와 recv()사이에 sleep하는 시간을 둠으로써 문제가 해결되었습니다.

또한 코드가 실행중에 recv()상태에 들어가서 무한히 대기중인 상황이 있었는데, 이는 상대방으로부터 더 이상 오는 메시지가 없는데 나는 하염없이 기다리고 있는 상황이었습니다. Ip fragmentation을 예로 들면, 마지막 fragmentation에 MF bit를 0으로 설정하여 receiver입장에서 마지막 이라는 것을 알 수 있었지만, 소켓 프로그래밍의 경우에는 그러한 server로부터 오는 메시지에 end flag가 없었습니다. 따라서 client 입장에서 더 이상 오는 메시지가 없으면 (정확히는, receiver buffer에 남은 segment가 없으면) recv()를 중단하도록 해야 했습니다. 그래서 recv()에 MSG\_DONTWAIT 라는 변수를 넣어서, 더 이상 오는 메시지가 없으면 recv()를 탈출하도록 하였습니다.(setsockopt를 통해 timeout을 넣어 일정시간동안 메시지가 안오면 recv()를 중단하도록 하는 방법도 시도해보았습니다.)

2)Ipv6 address

저의 호스트는 할당받은 ipv6 주소가 없었기 때문에, miredo와 같은 터널링을 통해 ipv6주소를 사용할 수 있도록 세팅해주어야 했습니다. 처음 윈도우 환경에서 miredo와 같은 application을 찾아보던중 윈도우에 teredo라는 기본 서비스가 있다는 것을 알게되었고, 이를 이용하여 ipv6 통신을 할 수 있었습니다.

3)bind() ERROR

이전과 똑 같은 환경에서 실행하였는데, 처음 실행할 때는 bind()에서 에러가 나지 않다가, 두번째 실행때 bind()에서 에러가 나는 경우가 있었습니다. 그 이유는 이미 쓰인 포트에 소켓이 다시 접속할 수 없기 때문인데, 이를 해결하기 위해 setsockopt를 통해 SO\_REUSEADDR 옵션을 넣어주거나, port번호를 바꾸는 방법으로 해결하였습니다.

4)fork()

Concurrent 서버를 구축하기 위해 ipv6 server에서 부모 프로세스가 fork()하여 5개의 자식 프로세스를 만드는 방법을 사용했습니다. 결과적으로는 5개의 자식 프로세스가 각각 connection을 만들고 토큰을 받은 이후에 바로 ipv4 client에게 넘겨주는 방식으로 통신에 성공하였습니다. 그러나 문제는 ipv6 서버가 5개를 초과하는 자식 프로세스를 만들게 된다는 것이었습니다. 이를 위해 static bool forked 라는 변수를 두어 cpu가 부모프로세스를 실행중일 때 fork()를 딱 5번까지만 하도록 시도하였으나, cpu가 자식프로세스중 하나를 실행중이다가 부모 프로세스를 다시 실행하려고 하면 forked가 다시 정의되어 flag역할을 하지 못하는 것을 확인했습니다. 그래서 이 문제는 해결하지 못했습니다.

4. setup to connect

우선 ipv6 server를 먼저 실행시키고, 일정시간마다 ipv4 client와 지속적으로 연결을 시도하도록 설정하였습니다. 이렇게 한 이유는 ipv6 server코드가 실행된 이후에 ipv4 client 코드가 실행되므로, ipv4 client와 연결이 성공할 때까지 계속해서 시도하도록 한 것입니다. 이후 ipv6 server는 fork()한다음 각각의 자식 프로세스가 ipv6 client로부터 accept를 기다리고 있는 상태가 됩니다.

그다음 ipv4 client를 실행시켜 ipv6 server와 연결한다음, ipv4 server와의 연결을 시도하도록 하였습니다.

모든 연결이 끝나면 ipv6 server에는 connected to V4\_CL 이라는 메시지가 뜨고,

Ipv4 client에는 IPV6 server is connected / IPV4 server is connected 라는 메시지가 뜨게 됩니다.

이후 ipv4 client가 ipv4 server와 통신을 한 다음 ipv6 server가 토큰을 받게되면, 받는 즉시 ipv4 client로 넘겨주게 되고, ipv4 client는 이 토큰을 모아 정해진 형식으로 만든다음 ipv4 server에 토큰을 넘겨주게 됩니다.

5. 실행환경

language : C++

IDE : Microsoft Visual studio 2017

OS : Linux Ubuntu (Oracle VM VirtualBox)

\* Oracle VM VirtualBox 에서의 네트워크 환경 : 어댑터에 브리지