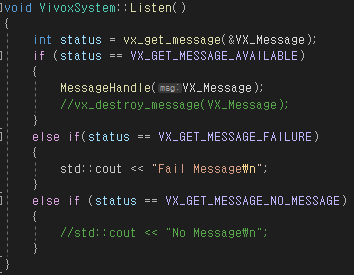
**2018180020 박재우(클라이언트) 24주차 기록**

**Vivox**

지난주에 이어서 Vivox를 구현한다.

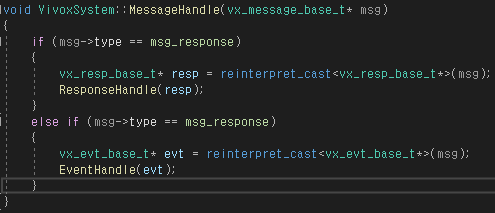
저번주에는 로그인이 잘 진행되지 않았다. request함수를 이용해 요청을 했지만 요청이 처리되지 않는 문제가 있었다. 요청을 하면 Vivox에서 응답을 보내주는데 그 응답을 받지 않는 문제였다. Listen함수를 만들었지만 request 이후 바로 불리도록 하였는데 바로 응답이 오지 않는 것 같다.



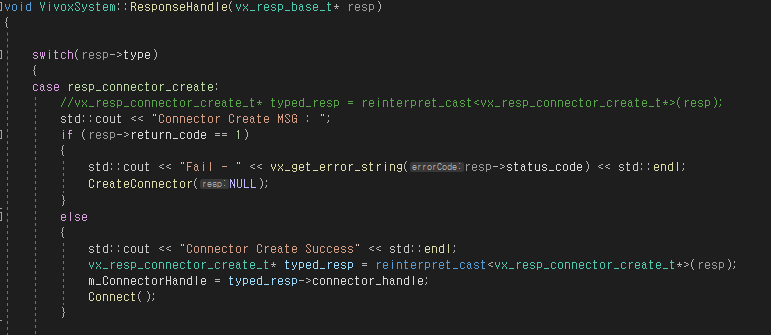
Listen함수는 vx\_get\_message함수를 사용해 메시지를 받아온다. 반환 값은 이 메시지가 유효한지, 유효하지 않은 지 그리고 메시지가 오지 않았는지 확인할 수 있는 값이다.

메시지가 정상적으로 도착하면 그 메시지에 대한 처리를 해주어야 한다. 공식 문서에는 메시지를 처리하고 vx\_destroy\_message를 호출 하라고 하지만 저 부분에서 오류가 터지면서 프로그램이 죽는다. 원인을 파악하고 있지만 아직 찾지 못해 일단 제외하였고 제외해도 문제없이 잘 작동한다.

다음은 메시지를 처리하기 위한 MessageHandle함수이다. 우리가 처리할 메시지의 종류는 2가지가 있다. Response와 Event이다. Request함수로 요청을 보내서 돌아오는 응답은 Response 메시지이다. 만약 vx\_req\_connector\_create를 만들어 요청했다면 돌아오는 응답메세지의 타입은 Response이고 Response구조체는 vx\_resp\_connector\_create이다.



MessageHandle에서 메시지의 타입을 비교해 알맞게 형변환을 해주고 ResponseHandle 함수로 값을 넘겨준다. EventHandle함수도 있기는 있지만 아직 Event가 넘어오는 경우가 없어서 일단은 비어 있다.



ResponseHandle의 일부분이다. switch문으로 구조체의 type으로 어떤 응답이 왔는지 확인하고 알맞은 처리를 해주면 된다. 구조체의 멤버에 return\_code와 status\_code가 있다. Return\_code는 요청이 정상적으로 처리가 되었는지 확인할 수 있다. 1이라면 실패이고 0이라면 성공이다. Status\_code는 오류코드로 Vivox 공식문서에서 무슨 오류인지 알 수 있고 vx\_get\_error\_string 함수에 넣으면 이 오류코드가 어떤 오류인지 글자로 출력해준다.

위의 코드는 Connector를 만들 때 돌아오는 메시지이다. 처음에 Connector Create 메세지가 도착했다는 문자열을 출력한 뒤 Return\_code가 1일 때 Fail문자열을 출력하고 성공하면 Success 문자열을 출력하게 했다. 처음에는 Connector 생성이 실패하는 경우를 생각하지 않고 있었다. Connector쪽은 문제가 발견되지 않고 있었는데 간헐적으로 한번씩 프로그램이 제대로 작동하지 않을 때가 있었다. 혹시나 해서 검사를 해봤더니 HTTPS오류가 떴었다. 이 오류를 공식문서에서 찾아보니 BackOff로 다시 시도하라고 한다. 아마도 네트워크 문제로 추측된다. 그래서 실패했을 때 CreateConnector함수를 호출해 다시한번 Connector를 생성하는 요청을 보내도록 했다. 생성 오류는 아무리 많이 발생해도 3번정도로 끝나고 결국엔 성공했다.

생성에 성공하면 세선에 로그인 해야한다. 방금 설명한 것처럼 Connector 생성이 한번에 잘 되지 않을 수도 있고 Connector 생성 성공하기 전에 로그인 요청이 보내질 수도 있기 때문에 확실하게 Connector가 생성된 이후에 로그인을 시도해야 한다. 그래서 성공 메시지가 도착했을 때 메시지 처리하는 동작으로 로그인 시도를 하면 된다. Return\_code가 1이 아닐 때(성공했을 때) Connect 함수를 불러 로그인을 시도한다. Connector 생성에 대한 응답에는 Connector Handle이 같이 들어온다. 이 Handle을 저장해 나중에 요청을 생성할 때 사용하도록 한다. 물론 Connector 생성 요청을 할 때 Connector Handle을 지정해 줄 순 있지만 비워 두면 Vivox에서 알아서 생성해주기 때문에 일단 비워 두고 생성된 Connector Handle을 응답에서 받아오는 방법으로 했다.

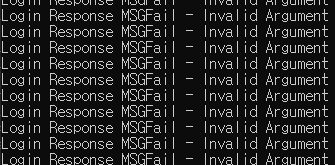


Connect 함수는 로그인을 위한 요청을 보내는 함수이다.

Req 구조체를 만들고 변수를 채워 넣어 요청을 하면 된다. 위와 같이 변수를 정해주고 요청을 보냈다. 주석 친 부분은 Account Handle인데 Connector Handle과 마찬가지로 비워 두면 알아서 생성해주고 생성된 Handle은 응답으로 받을 수 있다. Access\_token은 vx\_debug\_generate\_token함수로 생성할 수 있다. 여기서 Vivox 홈페이지에서 생성한 API key가 사용된다.



로그인의 응답이다. 만약 실패한다면 다시 로그인을 시도하도록 만들었고 성공하면 받아온 정보를 출력하게 했다.



돌려보니 로그인에 오류가 계속 터져 나왔다. 변수가 잘못 들어간 것 같다.



찾아보니 acct\_name은 양 옆에 ‘ . ’ 을 붙여야 했다. 수정한 상태로 실행해보니 잘 성공한 것을 확인할 수 있었다.

다음은 JoinChannel함수이다. 이 함수는 실제로 음성채팅을 하기위해 채널에 입장하는 요청을 보내는 함수이다. Login이 성공했을 때 호출되고 레벨을 이동할 때 호출되도록 한다. 

채널에 입장하기 위해 먼저 채널에 대한 URI가 필요하다. 해당되는 채널의 URI를 만드는 함수는 vx\_get\_random\_channel\_uri\_ex이다. 앞에 접두사를 붙이는데 confctl-e-는 에코채널이다. e 대신에 g와 d를 붙일 수 있는데 각각 Non-Positional 채널과 Posisitional 채널이다. Positional을 사용자의 위치에 따라 목소리가 들리고 안 들리고 하는 채널이다.

채널은의 uri의 형식은 confctl-[e, g, d][-.채널이름.@](mailto:-.채널이름.@)도메인 이다. 저 채널이름이 같은 uri로 접속을 시도하면 같은 채널이름에 접속한 사용자끼리 음성채팅이 가능하다. 에코채널의 경우는 자신의 목소리가 다시 돌아오는 1인 전용 채널이기 때문에 공식문서에서는 채널이름에 사용자의 고유 이름을 넣은 것을 권장한다. 또는 위의 주석 친 코드처럼 무작위 uri를 생성하는 것 또한 상관없다. 지금은 에코채널에 접속하기 위한 uri를 만들어주는 함수를 사용했는데 입력되는 Channel 인자만 사용자의 닉네임으로 받아오게 하여 접속을 시도하면 문제될 것이 없다. 애초에 지금은 테스트를 위해 에코채널로 하고 있다. 우리 게임은 서버에서 방을 만들어서 4명이서 플레이 하는 게임이기 때문에 이 Channel에는 서버에서 만들어진 방에 고유한 코드 같은 것을 만들어서 그 코드로 접속할 수 있도록 만들 것이다.



채널에 접속을 성공하면 오는 응답에 대한 처리이다. 지금은 딱히 처리할 일이 없기 때문에 실패하면 실패한 내용의 로그를 출력하고 성공하면 성공한 로그를 출력한 후 반환된 Session Handle을 저장하도록 했다.



위 사진은 Connector를 만들고 채널에 접속하기까지의 로그를 찍은 모습이다.

모두 문제없이 잘 진행되었고 이 후 마이크에 말하면 내 목소리가 다시 들려오는 것까지 확인할 수 있었다. 다음 주에는 더 개발하여 에코채널에서 벗어나 다른 사람들과 Vivox로 음성채팅이 가능하도록 만들도록 한다.