HW1 Report

2013-11421 이은서

1.Problem Statement

과제1은 간단한 채팅 프로그램을 만드는 것이다.

Specification

Server

- 서버는 여러 클라이언트에게 서비스를 제공해야 한다.
- 클라이언트가 deactivated인 경우 서버는 메시지를 저장했다가 클라이언트가 activate 되었을 때 그 동안 받은 메시지를 전송해주어야 한다.

Client

- 클라이언트는 할당된 id로 로그온 할 수 있다.
- 메시지를 보낼 수 있다. 이 때 메시지 길이는 140자로 제한하기로 한다.
- 메시지를 받을 수 있다. 이 때 한 번 읽은 메시지는 다시 볼 수 없는 것으로 한다.

계획

Server

- 여러 클라이언트의 요청을 처리해야 하기 때문에 select()를 이용한 multiplexing 서버를 만들어 서비스하도록 한다.
- 한 id는 한 곳에서만 activate되어야 한다. 이미 activate된 아이디로 로그인을 시도할 경우, 로그인에 실패했다는 메시지를 보낸다.
- 클라이언트의 상태(activate, deactivate)에 대해 알아야 한다.
- 스펙 문서에 클라이언트의 IP와 포트 번호를 알고 있어야 한다는 명시가 있다.
- 클라이언트가 deactivated일 때 메시지를 저장하기 위해 메시지 큐를 구현해야 한다. 구현의 편의를 위해 각 클라이언트별로 로그 파일을 만들어 관리하기로 했다.

Client

- 로그인 시도 할 때 올바른 아이디 형태인지 체크한다.

- 메시지를 140자까지 보낼 수 있다.
- Activate된 상태일 때 자신에게 오는 메시지는 항상 받되, 읽기 명령이 있을 때만 지난번에 읽은 마지막 메시지 다음부터 읽기 명령 전까지 받은 메시지를 표시하도록 한다.
- 이를 위해서 스레드를 나누어 한 스레드에서는 자신에게 오는 메시지를 계속 받고, 다른 스레드에서는 메시지 보내기 등의 명령을 처리하는 것으로 한다.

2.Implementation Details

Protocol Design

아래와 같이 프로토콜을 디자인했다.

- 프로토콜 사이즈는 공통적으로 144byte이다.

Client to Server

1. action code
    - activate/logon : 0
    - message send : 1
2. log on protocol
    - [action code | id]
3. message send
    - [action code | destination | source | message content]
    - message content는 140자까지 허용한다.

Server to Client

1. action code
    - activate : 0
    - message deliver : 1
2. log on 반응
    - log on 성공 : [action code | 0]
    - log on 실패 : [action code | 1]
3. message deliver
    - [action code | source | message content]

Client Manual

- log on
    i. 로그온 하려면 1~9 사이의 한자리 숫자를 id로 입력하면 됩니다.
    ii. 정상적으로 로그인 되면 `login success` 가 콘솔에 나타납니다.

- 메시지 보내기
    i. 콘솔에 `s` 를 입력합니다.
    ii. 콘솔에 `To :` 이 보이면 받을 사람의 id(0~9 사이의 한자리 숫자)를 입력합니다.
    iii. 콘솔에 `Content :` 이 보이면 보내고 싶은 메시지를 입력합니다.
- 메시지 읽기
    i. 콘솔에 `r` 를 입력합니다.
    ii. 콘솔에 지금까지 받은 메시지들이 나타납니다.
- deactivate/log off
    i. 콘솔에 `q` 를 입력합니다.

그 외

1. select 사용
    - 여러 클라이언트에서 오는 요청을 동시에 처리해야 하는데, pthread나 프로세스를 사용하는 방법 대신 select를 사용하는 방법을 채택하였다. 이번 과제에는 해당하지 않지만 사용자 연결이 많아지는 경우 thread나 process만으로 연결을 감당할 수 없다. 또 thread나 process를 사용한 비동기 통신의 경우 프로세스간 통신에도 신경을 써야해서 select함수를 사용하여 멀티플렉싱을 하도록 하였다.
    - select함수는 여러개의 fd를 관찰하면서 변화가 있는 fd의 수를 리턴한다. 이 때 변화를 일으킨 fd는 fd_set을 통해 알 수 있다. 구현에서는 FD_ISSET이라는 함수를 통해 변화한 fd를 알아낸다.
    - 타임아웃이 발생하여 리턴한 경우는 0을 리턴한다.
    - client로부터 연결 요청이 서버 소켓으로 들어오는 것을 처리하는 부분이 58줄부터 있다. 연결 요청을 받아들이면 리턴되는 소켓을 reads에 추가시켜 모든 클라이언트와의 통신을 하나로 멀티플렉싱할 수 있다.
    - 서버 소켓 fd이외의 fd가 변화하면 클라이언트의 소켓 fd가 변화했다는 뜻이므로 해당 소켓에서 메시지를 읽어 처리한다(73줄부터). 클라이언트로부터 연결이 끊어지는 것도 이 경우에 해당하기 때문에 걸러주고 해당 소켓을 닫는다(79줄부터).
    - 그 이외의 것은 디자인한 프로토콜로 메시지를 파싱하여 적절한 처리를 한다.
2. pthread 사용
    - 서버의 경우는 클라이언트와 1:n 연결을 위해 select를 사용하였으나 client의 경우에는 server에서 오는 메시지를 받는 일과 클라이언트 프로그램 전체 흐름을 제어하는 일만 구분하여 하면 되기 때문에 pthread를 사용하여 서버로부터 오는 메시지를 계속 읽는 thread를 만들었다(45줄). 194줄부터는 메시지를 받는 thread의 메인이 되는 함수인 msgrecieve가 있다.
    - thread간 전역변수가 공유되는 것을 이용하여 recieve thread에서 계속 메시지를 저장하고 메인에서는 이를 읽는 방법을 사용하였다.
3. 메시지 관련

- 메시지 길이와 클라이언트에서 아카이브에 저장가능한 메시지 수는 임의로 140자와 128개로 정하였다.
- 메시지 길이가 140자로 정해지면서 프로토콜도 144byte가 되었다.
- 서버에서는 id별로 로그 파일에 메시지를 저장한다.
- 클라이언트에서는 클라이언트 프로그램 내부 아카이브에 받은 메시지를 저장해두었다가 읽기 멍령이 들어오면 아카이브에 저장된 메시지를 모두 출력한다.

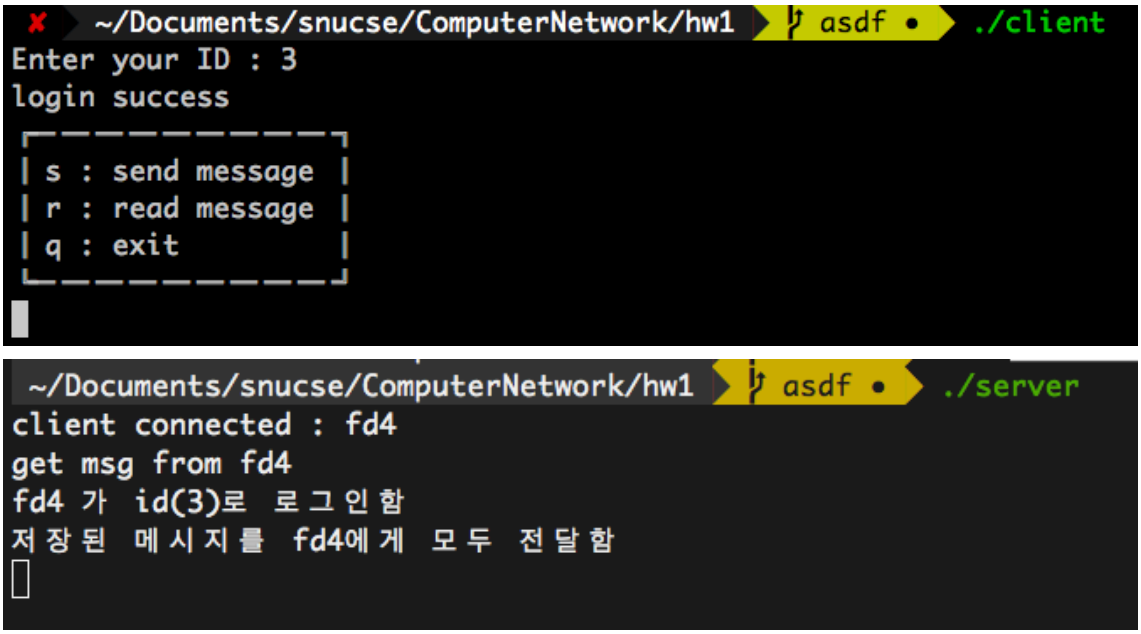
4. 입력 관련
   - fgets로 정해진 길이만큼 읽다보니 정해진 길이보다 더 긴 길이의 입력을 받았을 때 버퍼에 입력이 남아있는 경우가 있어 fseek을 통해 버퍼의 끝을 찾아서 거기서부터 새로운 입력을 받는 것으로 문제를 해결하였다. 이 두개를 합친 함수가 client에 readsize 이다.
5. id 관련
   - 스펙에서 대략 10개정도의 클라이언트 아이디를 만들라고 해서 가장 간단하게 1부터 9까지의 숫자로 아이디를 만들었다.

3.Results

로그인

로그인 성공

- 로그인이 성공한 경우, 서버는 클라이언트에게 deactivate된 동안 온 메시지를 보낸다.



로그인 실패

- 로그인이 실패한 경우, 클라이언트가 종료된다.

```
~/Documents/snucse/ComputerNetwork/hw1  asdf •  ./client
Enter your ID : 3
login failed
```

```
~/Documents/snucse/ComputerNetwork/hw1  asdf •  ./server
client connected : fd4
get msg from fd4
fd4 가 id(3)로 로그인함
저장된 메시지를 fd4에게 모두 전달함
client connected : fd5
get msg from fd5
Invalid login : client disconnected : fd5
```

메시지 보내기

Client

- deactivate상태인 5와 activate상태인 1에게 메시지를 보냈다.

```
s
=====Send Message=====
To : 5
Content :
sending message. hello, world! World of Warcraft!
======================
 ┌──────────┐
 | s : send message |
 | r : read message |
 | q : exit         |
 └──────────┘
s
=====Send Message=====
To : 1
Content :
hello, world!
======================
 ┌──────────┐
 | s : send message |
 | r : read message |
 | q : exit         |
 └──────────┘
```

Server

- deactivate상태인 5에게 보내는 메시지는 저정하고 activate상태인 1에게 메시지를 바로 전달한다.

```
get msg from fd4
메시지가 왔어요! fd4가 보냄
store message.
client connected : fd5
get msg from fd5
fd5 가 id(1)로 로그인함
저장된 메시지를 fd5에게 모두 전달함
get msg from fd4
메시지가 왔어요! fd4가 보냄
메시지 바로 전달함 : 13hello, world!
```

메시지 읽기

5에서 메시지 읽기

- 5로 로그인 후 메시지 읽기를 실행했다.

```
r
=====Read Message=====
From : 3
Content :
sending message. hello, world! World of Warcraft!
======================
 ┌──────────┐
 | s : send message |
 | r : read message |
 | q : exit         |
 └──────────┘
```

1에서 메시지 읽기

- 1에서 메시지 읽기를 실행했다.

```
r
=====Read Message=====
From : 3
Content :
hello, world!
======================
 ┌──────────┐
 | s : send message |
 | r : read message |
 | q : exit         |
 └──────────┘
```

클라이언트 종료

Client

Server

4.Discusion

1. 입출력 처리의 어려움 C로 구현했기 때문에 input, output을 처리하는 과정이 번거로웠다.

2. 프로토콜 디자인

- 구현을 쉽게 하기 위해서 모든 프로토콜을 일정한 길이로 정했는데, login같은 경우 실제로 정보를 담고 있는 부분이 매우 적어서 비효율적이다. 하지만 클라이언트 당 한 번만 하기 때문에 그냥 모든 프로토콜의 길이를 일정하게 하기로 했다.
- 메시지 보내기/받기의 경우는 메시지 내용, 보내는 사람, 받는 사람만 있는데 시간을 함께 보내도 좋을 것 같다. 현재 구현에서는 클라이언트의 소켓들을 차례로 돌며 요청 이 있는 것을 처리하도록 되어 있는데 이 때 시간 순서가 뒤바뀌어 처리될 수 있는 가능성이 있다. 예를 들어 c가 1초에 메시지를 b에게 보내고, a가 2초에 메시지를 b에게 보냈는데 서버에서 a의 메시지를 먼저 처리하는 경우 b는 c보다 a의 메시지를 먼저 보게 된다. 시간을 함께 보내면 시간순으로 정렬한 메시지를 받을 수 있을 것이다.