select를 사용하는 채팅서버에서 주의할 점은 select()에서만 패킷을 기다려야 한다는 것입니다. 여기에서 기다려야 여러 소켓에 대해 동시에 패킷을 기다릴 수 있습니다. 코드의 다른 위치에서 read()로 패킷을 기다리면 read에 지정한 소켓에서만 패킷을 기다리므로 다른 클라이언트가 보내는 패킷을 처리할 없습니다. handle\_protocol() 에서 read()하는 것은 이미 패킷이 들어온 상태이므로 추가 대기 없이 바로 패킷을 읽으므로 상관이 없지만 그 외의 다른 곳에서 read를 하면 아직 도착하지 않은 패킷을 기다리게 되므로 서버가 블록킹이 됩니다.

숙제 4번에서 모든 클라이언트에게 패킷을 보내려면

모든 소켓에 대해 루프를 돌면서 스테이트가 3 이상인 소켓에 쓰면됩니다.

for(i=0;i<50;i++){

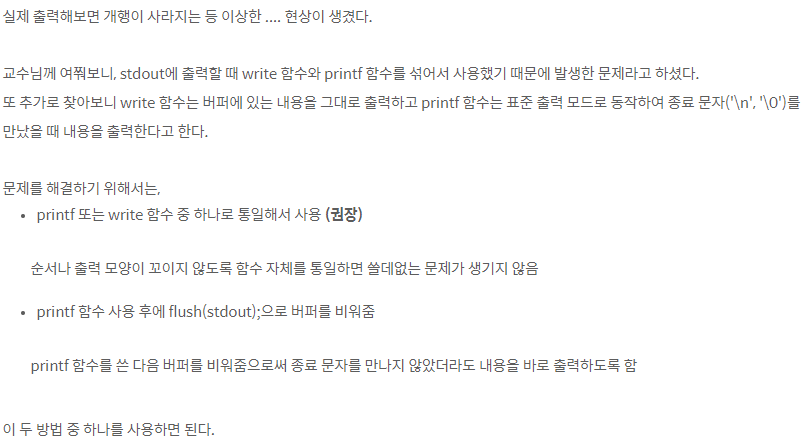
if (state[i]==3){

write(i, buf, strlen(buf));

}

자기 자신은 빼려면 i==x 인 경우는 제외하면 됩니다.

+printf 와 write 함수를 같이 쓰면, window에 출력되는 결과의 순서가 뒤섞이는 경우가 있는데,



이는 위의 글과 같이 하나의 함수로 통알하여 사용하거나,

printf함수 사용후, fflush를 통해 버퍼를 지워주면된다.

나는 printf fflush방법을 선택

hw

4) Modify the protocol such that the server relays a message from a client to all other clients after the “ping-pong-pang-pung” sequence is completed.

The clients should fork itself after the “ping-pong-pang-pung” sequence so that the parent part keeps reading while the child part keeps writing.

**The server does not fork** since it doesn't do the chatting by itself; it just relays a message from one client to all other clients.

The server checks state[] array to see which socket is ready to receive message.

cli at socket 3 => serv: ping

serv => cli at socket 3 : pong

cli at socket 3 => serv: pang

serv => cli at socket 3 : pung. Protocol completed. Start chatting.

cli at socket 4 => serv: ping

serv => cli at socket 4 : pong

cli at socket 4 => serv: pang

serv => cli at socket 4 : pung. Protocol completed. Start chatting.

cli at socket 5 => serv: ping

serv => cli at socket 5 : pong

cli at socket 5 => serv: pang

serv => cli at socket 5 : pung. Protocol completed. Start chatting.

cli at socket 3 => serv: hello

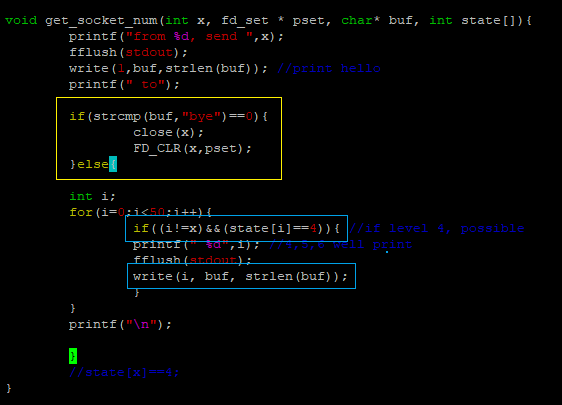
serv => cli at socket 4, 5 : hello

cli at socket 4 => serv: hi

serv => cli at socket 3, 5: hi

.................

코드 파일명 - servping\_clifork.c / clifork.c



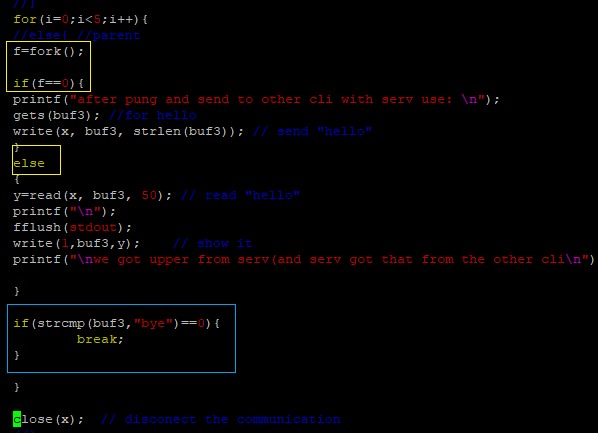
servping\_clifork.c

state 4까지 ( pang까지 완료한 client들의 sokcet num이 i의 값으로 출력되게 됩니다.)

파란 박스 부분의 내용을 보면, 어떠한 string을 보낸 cli의 해당 x 를 제외한 i값들에게 write로 string값을 보낼 수 있도록 합니다.

즉, 자신을 제외한 다른 socket들에게 받은 어떠한 string을 보내는 것을 확인할 수 있습니다.

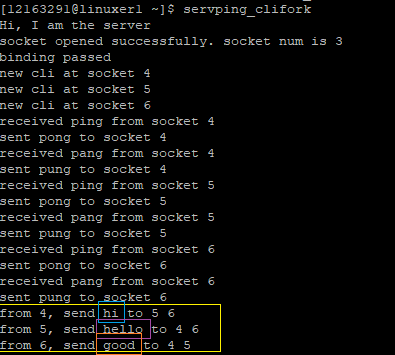
만약 bye를 받게되면 disconnect 하며 end할 수 있도록 해주었습니다.



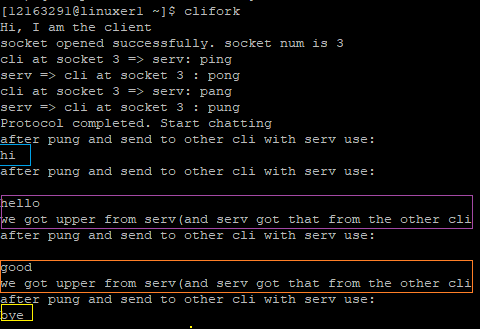
clifork.c

client에 fork를 가미하여, cli 가 serv에게 계속 stirng을 보내고, 받을 수 있도록 하였습니다.

여기에서도



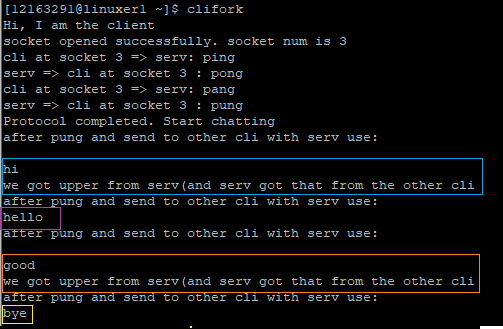
servping\_clifork.c 결과



socket 4 clifork.c

hi를 5,6 에보내고,

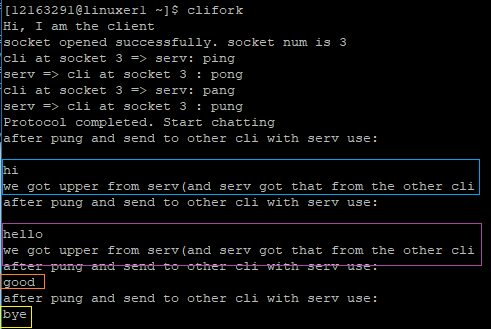
아래의 과정을 통해 5에서 hello를, 6에서 good을 받은 것을 확인할 수 있었습니다.



socket 5 clifork.c

hello를 4,6 에보내고,

아래의 과정을 통해 4에서 hi를, 6에서 good을 받은 것을 확인할 수 있었습니다.



socket 6 clifork.c

good를 4,5 에보내고,

아래의 과정을 통해 4에서 hi를, 5에서 hello을 받은 것을 확인할 수 있었습니다.

보완이 더 필요함

cil에서는 bye를 만났을 경우, for문의 남은 반복없이

parent process도 kill해야한다

serv 에서는

어떻게 끝내줄지

5) Modify your code in Problem 4) such that the server attaches the client’s name and age in the message. For this purpose, the server should ask name and age for each client and store them in cli[] array which is an array of client{} structure to store name and age of each client. cli[x] will remember the client information whose socket number is x.

struct client{

char name[20]; // this client's name

char age[5]; // this client's age as a string

};

......

struct client cli[50]; // max 50 clients

cli aaa=> serv: ping

serv => cli aaa: pong

cli aaa=> serv: pang

serv => cli aaa: pung. name?

cli aaa=> serv: aaa

serv => cli aaa: age?

cli aaa => serv: 19

cli bbb=> serv: ping

serv => cli bbb: pong

cli bbb=> serv: pang

serv => cli bbb: pung. name?

cli bbb=> serv: bbb

serv => cli aaa: age?

cli aaa => serv: 22

cli ccc=> serv: ping

serv => cli ccc: pong

cli ccc=> serv: pang

serv => cli ccc: pung. name?

cli ccc=> serv: ccc

serv => cli aaa: age?

cli aaa => serv: 21

serv => cli aaa: start chatting

serv => cli bbb: start chatting

serv => cli bbb: start chatting

cli aaa => serv: "hello there"

serv=> cli bbb: "aaa 19 to bbb 22: hello there"

serv=> cli ccc: "aaa 19 to ccc 21: hello there"

cli bbb=> serv: "hi"

serv => cli aaa: "bbb 22 to aaa 19: hi"

serv => cli ccc: "bbb 22 to ccc 21: hi"

servchat.c

clichat.c



우선 서버부분에서 state를 5까지 늘려주었습니다.

3의경우는 이름을 받고,

4의경우는 나이를 받고

5의경우에는chat을 시작합니다.

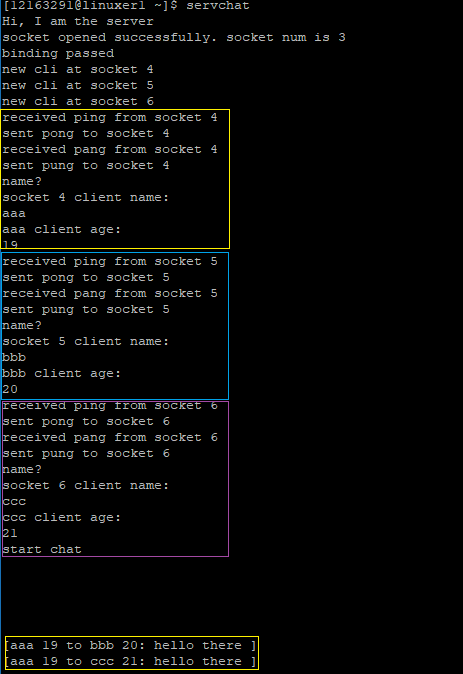


clichat.c 의 경우에도 거의 동일합니다.

이름을받고, 나이를 받은다음 (전은 ping pang 과정입니다.)

fork를 통해 대화를 할 수 있도록 합니다.

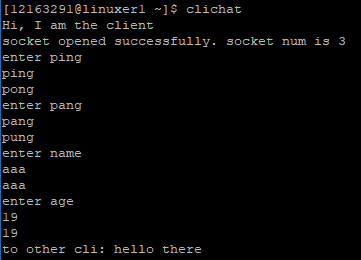
bye가 종료의 trigger역할이 되도록 설정하였습니다.



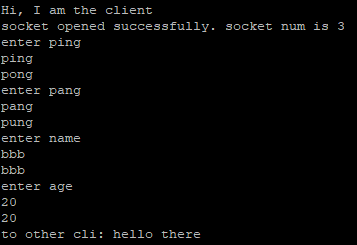
server결과는 위와 같습니다.

ping, pong을 끝내고 이름과 나이정보를받은후, 채팅을 시작합니다.

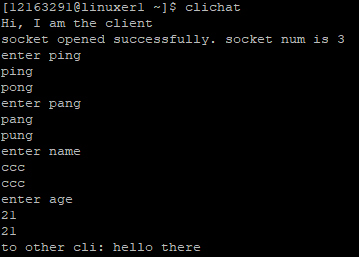
저는 aaa client에서 hello there을 보냈습니다.



aaa



bbb



ccc

결과 순서가 뭔가 엉켰지만 hellothere이 다른 client에 가긴 하였다는 것을 확인할 수 있었습니다.

수정이 필요합니다.

6) Modify your code in Problem 5) such that the client can now specify which client it wants to chat with. Add "partner" to client{} strucure to remember the socket number of the chatting partner. The server should ask which partner the clients wants to talk with and remember the partner's socket number in the client{} structure. Assume if cli A points to cli B as a partner, cli B also points to cli A as a partner.

struct client{

char name[20]; // this client's name

char age[5]; // this client's age as a string

int partner; // the socket number of the chatting partner of this client

};

cli aaa=> serv: ping

serv => cli aaa: pong

cli aaa=> serv: pang

serv => cli aaa: name?

cli aaa=> serv: aaa

cli bbb=> serv: ping

serv => cli bbb: pong

cli bbb=> serv: pang

serv => cli bbb: name?

cli bbb=> serv: bbb

cli ccc=> serv: ping

serv => cli ccc: pong

cli ccc=> serv: pang

serv => cli ccc: name?

cli ccc=> serv: ccc

cli ddd=> serv: ping

serv => cli ddd: pong

cli ddd=> serv: pang

serv => cli ddd: name?

cli ddd=> serv: ddd

serv=>cli aaa: chat partner?

cli aaa=>serv: bbb

serv=>cli bbb: chat partner?

cli bbb=>serv: aaa

serv=>cli ccc: chat partner?

cli ccc=>serv: ddd

serv=>cli ddd: chat partner?

cli ddd=>serv: ccc

serv => cli aaa: start chatting

serv => cli bbb: start chatting

serv => cli ccc: start chatting

serv => cli ddd: start chatting

cli aaa => serv: hello there

serv=> cli bbb: aaa to bbb: hello there

cli bbb=> serv: hi

serv => cli aaa: bbb to aaa: hi

cli ccc => serv: hear me

serv=> cli ddd: ccc to ddd: hear me

cli ddd=> serv: hi there

serv => cli ccc: ddd to ccc: hi there

5)번의 client 스트럭트를 확장시킨후,

비슷하게 진행해 주면 됩니다..

7) Implement a chatting server. The state of the client during the protocol is as follows. At any moment multiple pair of clients should be able to talk at the same time.

state 1 : The server is expecting "hello" for this client. When "hello" arrives, the server

sends "name?"

state 2 : The server is expecting client ID from this client. The server rembers this

client's ID in cli[x].name, where x is the socket number of this client. The server

asks "ready?".

state 3 : The server is expecting "yes" from this client. The server sends all client name

whose state is greater than or equal to 3.

state 4 : The server is expecting the chatting partner's ID from this client. The server

remembers partner socket number in cli[x].partner. Send "go" to the client.

state 5 : The client is now in chatting session. The server is expecting some chat

message from this client. The server sends this message to cli[x].partner.

All client's initial state is 1.

cli eee => serv: hello

serv => cli eee: name?

cli eee=> serv: eee

serv => cli eee: ready?

cli eee => serv : yes

serv => cli eee: client list (aaa bbb ccc …..)

cli eee=> serv : bbb

serv => cli eee: go

................

cli bbb => serv : yes

serv => cli bbb : client list (aaa bbb ccc eee ...)

cli bbb => serv : eee

serv => cli bbb : go

cli eee => serv : hi how are you

serv => cli bbb : hi how are you

cli bbb => serv : hi there

serv => cli eee : hi there

..............