**System Programming lect12 homework**

정보통신공학과 12215229 김민지

1) Download cliping.c and servping.c into your directory, modify IP and port number appropriately, and compile them. Run the server first and run client 3 times each in different window. Check if the server can handle multiple clients at the same time.

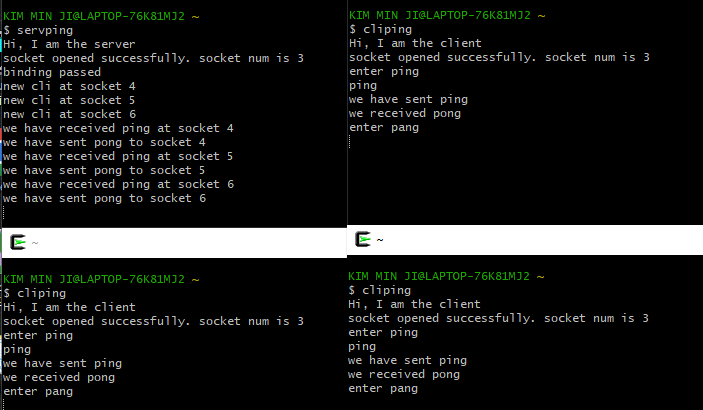
cliping1 -> servping: ping

cliping2 -> servping: ping

cliping3 -> servping: ping

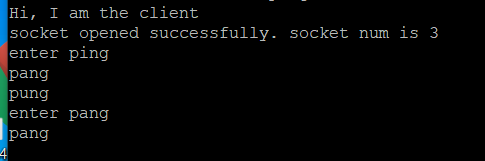
……………



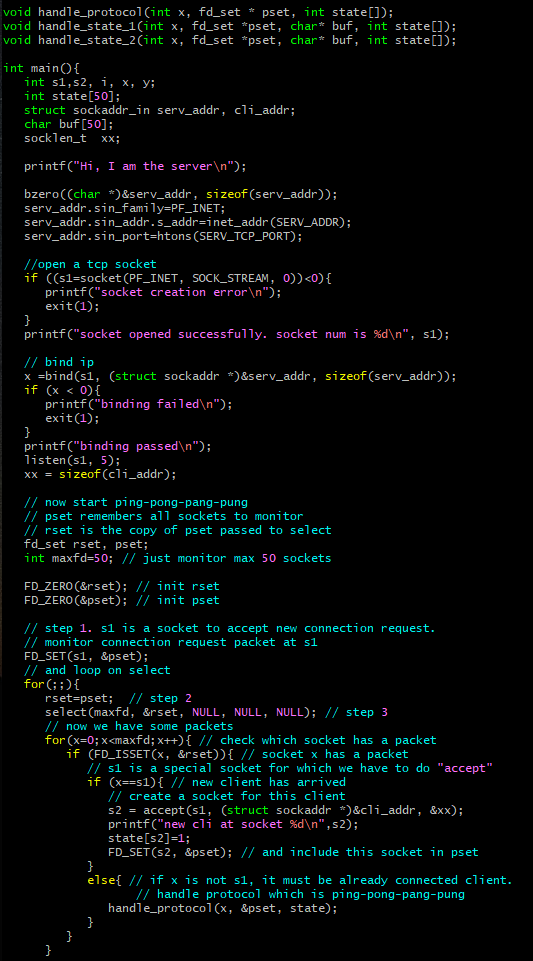


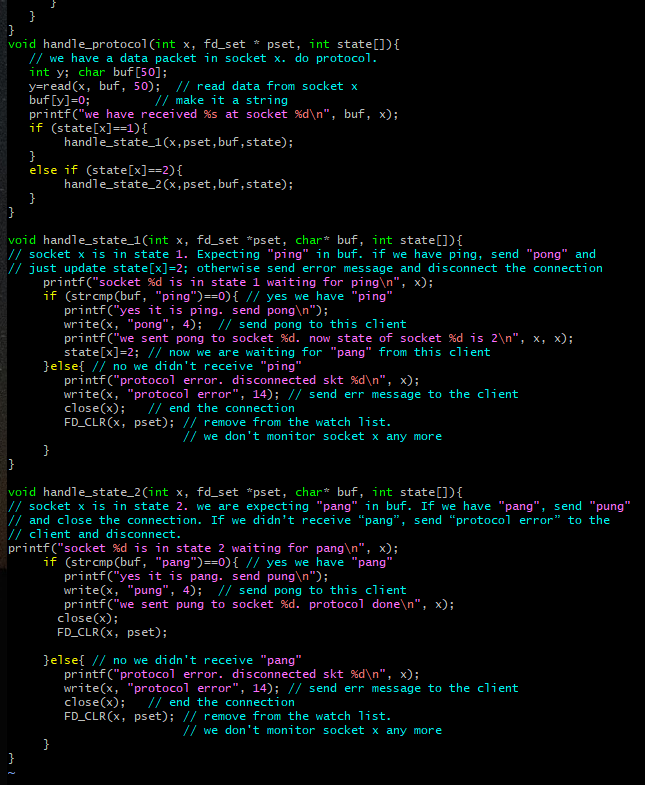
servping.c, clipping.c 파일을 디렉토리에 복사하고, port number와 ip주소를 수정한다. 컴파일 후 실행하니 서버가 3개의 클라이언트를 각각 처리할 수 있었다.

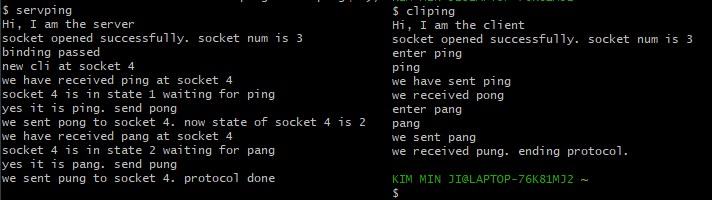
2) The server in Prob 1) cannot give error message to clients even when the client doesn't follow the protocol. Run server and run client and let the client send "pang" instead of "ping" as the first message. The server gives "pung" instead of error message as below.



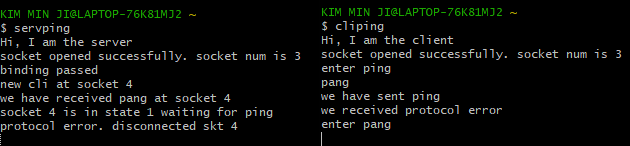
Modify servping.c so that it can send error message when the client sends something other than "ping" for the first message. You need to remember the state of each socket to do this. The beginning state of each socket is 1 which means this socket is waiting for the first message “ping”. If a socket at state 1 receives a message other than “ping”, the server should send protocol error message. Try to solve this problem by yourself and look at the code in Problem 2-1) if you need help.







(ping일 때)



(ping이 아닐 때)

문제 2-1번의 코드를 servping.c 코드에 추가하였다. 변경된 점은 먼저 handle\_state\_1(), handle\_state\_2() 함수를 만들었다. 1은 ping이 들어왔는지 확인하고 ping이면 state 숫자를 2로 바꿔준다. ping이 아니라면 write()로 오류메시지를 클라이언트에게 보낸다. 그리고 프로토콜을 종료하고 해당 소켓을 0으로 초기화시킨다. 2는 pang이 들어왔는지 확인하고 아니라면 write()로 오류메시지를 클라이언트에게 보낸다. 그리고 프로토콜을 종료하고 해당 소켓을 0으로 초기화시킨다.

그리고 메인함수에서는 클라이언트와 연결되면 state[s2] 값을 1로 바꾸고 handle\_protocol() 함수를 실행한다.

실행결과로 클라이언트에서 ping 이 아닌 다른 글자를 입력하면 클라이언트 화면에 “we received protocol error” 메시지가 출력되었다.

2-1) Modify the server such that it disconnects the connection if the client doesn't follow the protocol. You need to keep track of the state of each client to do this. (The client will act strange when the server disconnects it. You don't have to change the client code for this since we don't care about what happens to the client when it does not follow the protocol.)

int state[50]; // state of each client (state of each client socket)

// 1: the server is waiting for "ping" from this client

// 2: the server is waiting for "pang" from this client

...........

for(x=0;x<maxfd;x++){ // check all fd

if (FD\_ISSET(x, &rset)){ // if we have packet in socket x

if (x==s1){ // if x is the connection accepting socket, we have a new client

// and we must have the connection request packet(SYN) at x

s2=accept(s1, ........); // now s2 is this client's socket

printf(“new cli at socket %d. it’s sate is 1\n”, s2);

state[s2]=1; // init the state of this client.

// the server is expecting "ping" from this client

FD\_SET(s2, &pset);

}else{ // we must have the data packet at socket x

handle\_protocol(x, &pset, state);

} // else

}//if

}//for

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

void handle\_protocol(int x, fd\_set \* pset, int state[]){

// we have data packet in socket x. state[x] shows the state of socket x.

// handle the protocol.

int y; char buf[50];

y=read(x, buf, 50); // read the data

buf[y]=0; // make it a string

printf("we have received %s at socket %d\n", buf, x);

if (state[x]==1){ // the state of this socket is 1 meaning we are

// expecting "ping" from this socket

handle\_state\_1(x, pset, buf, state);

}else if (state[x]==2){ // expecting "pang"

handle\_state\_2(x, pset, buf, state);

}

}

void handle\_state\_1(int x, fd\_set \*pset, char\* buf, int state[]){

// socket x is in state 1. Expecting "ping" in buf. if we have ping, send "pong" and

// just update state[x]=2; otherwise send error message and disconnect the connection

printf(“socket %d is in state 1 waiting for ping\n”, x);

if (strcmp(buf, "ping")==0){ // yes we have "ping"

printf(“yes it is ping. send pong\n”);

write(x, "pong", 4); // send pong to this client

printf("we sent pong to socket %d. now state of socket %d is 2\n", x, x);

state[x]=2; // now we are waiting for "pang" from this client

}else{ // no we didn't receive "ping"

printf(“protocol error. disconnected skt %d\n”, x);

write(x, "protocol error", 14); // send err message to the client

close(x); // end the connection

FD\_CLR(x, pset); // remove from the watch list.

// we don't monitor socket x any more

}

}

void handle\_state\_2(int x, fd\_set \*pset, char\* buf, int state[]){

// socket x is in state 2. we are expecting "pang" in buf. If we have "pang", send "pung"

// and close the connection. If we didn’t receive “pang”, send “protocol error” to the

// client and disconnect.

printf(“socket %d is in state 2 waiting for pang\n”, x);

if (strcmp(buf, "pang")==0){ // yes we have "pang"

printf(“yes it is pang. send pung\n”);

write(x, "pung", 4); // send pong to this client

printf("we sent pung to socket %d. protocol done\n", x);

close(x);

FD\_CLR(x, pset);

}else{ // no we didn't receive "pang"

printf(“protocol error. disconnected skt %d\n”, x);

write(x, "protocol error", 14); // send err message to the client

close(x); // end the connection

FD\_CLR(x, pset); // remove from the watch list.

// we don't monitor socket x any more

}

}

2번에서 작성한 코드와 동일하다. 클라이언트가 서버에게 보낸 메시지가 ping, pang 이 아니라면 write()으로 오류메시지를 보내고, close(x); FD\_CLR(x,pset); 을 하여 클라이언트와 연결을 종료한다.

3) Modify the protocol such that the server expects a final “ping” again from the client. Make sure the server give error message and disconnect the client if the client doesn't follow the protocol.

cli=>serv: ping

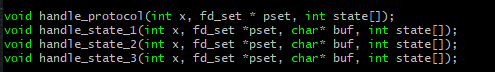
serv=>cli: pong

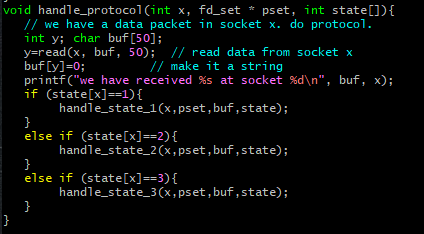
cli=>serv: pang

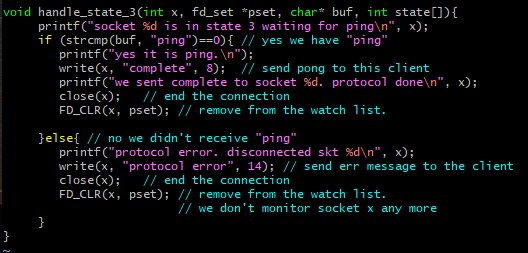
serv=>cli: pung

cli=>serv: ping (final ping)

serv=>cli: protocol completed

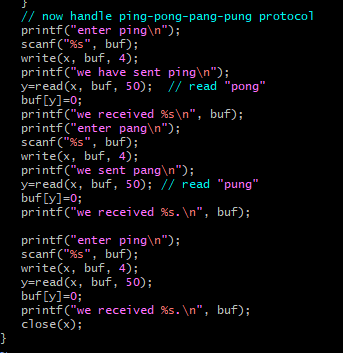






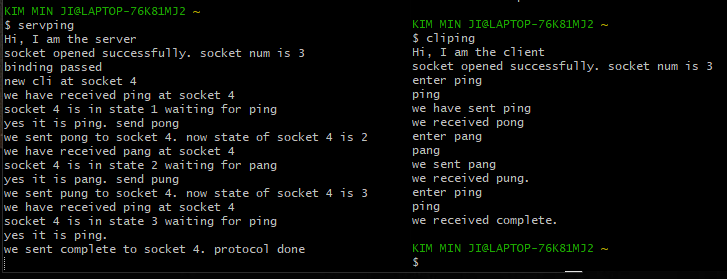
(servping.c) 에 추가한 코드

ping을 추가로 받기 위해 handle\_state\_3 함수를 추가한다. state 2에서 pang이 들어오면 state를 3으로 바꾼다. 그리고 state\_3 함수에서는 ping을 받으면 complete를 클라이언트에게 보내고 연결을 종료하고, x를 0으로 초기화한다.



(cliping.c) 에 추가한 코드

ping을 한번 더 보내기 위해 cliping.c 코드도 수정한다. pang을 보낸 후, ping을 보내기 위해 사용자에게 입력받고, 서버에게 메시지를 받은 후 close(x)로 프로세스를 종료한다.



클라이언트와 서버가 ping-pong-pang-pong-ping-complete 메시지를 주고받고 연결이 종료된다.

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.

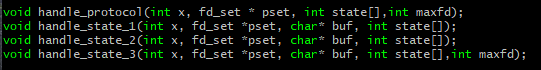
cli at socket 3 => serv: hello

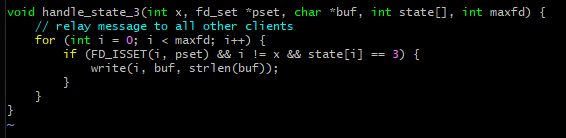
serv => cli at socket 4, 5 : hello

cli at socket 4 => serv: hi

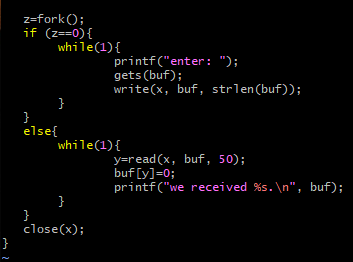
serv => cli at socket 3, 5: hi

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

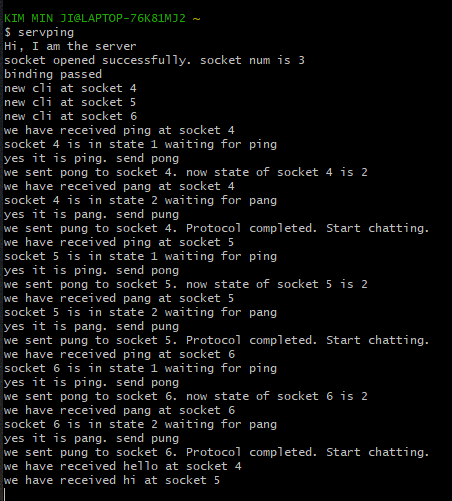


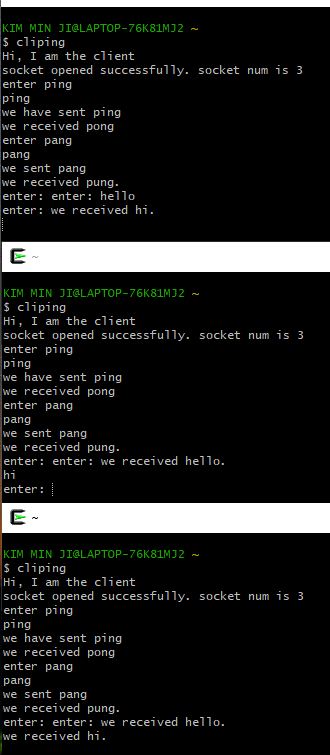


servping.c 코드를 위와 같이 수정했다. handle\_state\_3함수에서 연결되어있는지를 FD\_ISSET으로 확인하고, 메시지를 보낸 클라이언트에게는 메시지가 안가게 하기 위해서 i!=x로 확인하고, 상태가 3인지를 and 조건으로 연결하고 조건이 맞다면 write() 함수로 메시지를 보낸다.



cliping.c 코드를 위와 같이 수정한다. z=fork를 하여 자식프로세스는 메시지를 입력받도록 하고, 부모프로세스는 메시지를 받도록 한다.





메시지를 보내면 다른 클라이언트에게 전송되는 것을 확인할 수 있었다.

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"

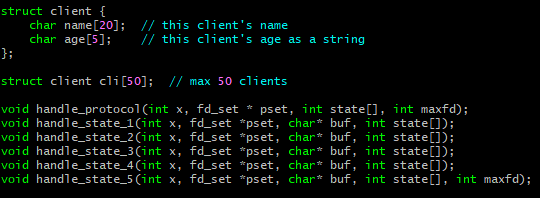
serv=> cli bbb: "aaa 19 to bbb 22: hello there"

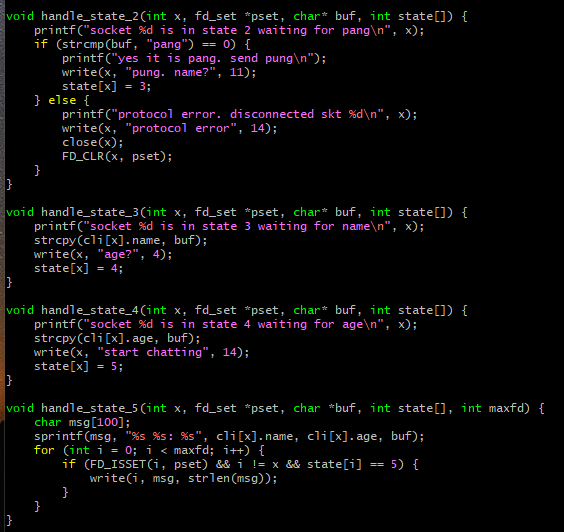
serv=> cli ccc: "aaa 19 to ccc 21: hello there"

cli bbb=> serv: "hi"

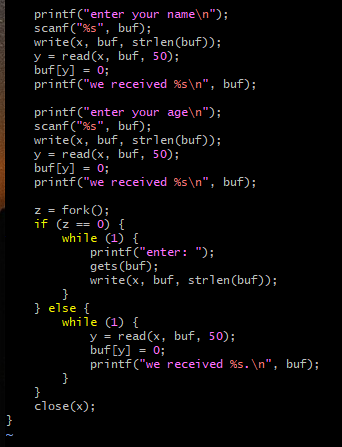
serv => cli aaa: "bbb 22 to aaa 19: hi"

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

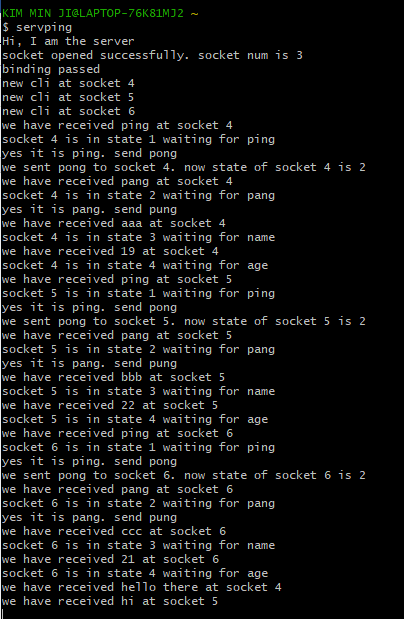


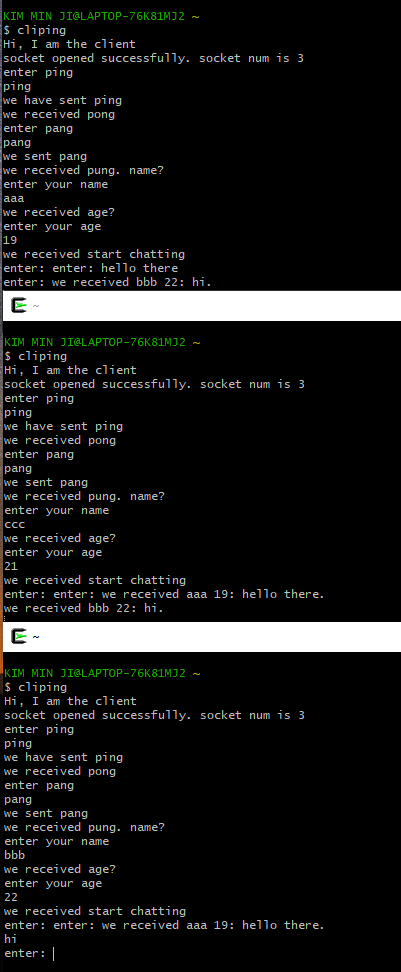


서버코드를 위와 같이 수정했다. client 구조체를 만들고 이름과 나이를 포함하게 한다. 그리고 50명의 클라이언트를 받을 수 있는 배열을 만든다. handle\_state\_ 함수를 5까지 만든다. 상태 2에서 pang을 받으면, 이름을 물어보고 상태 3으로 넘어간다. 상태3에서는 이름을 cli[x].name에 저장하고 나이를 물어보고 상태 4로 넘어간다. 상태4에서는 나이를 cli[x].age에 저장하고 상태 5로 넘어간다. 상태 5에서는 문제 4와 같이 채팅을 하는데, sprintf()를 사용해서 이름과 나이가 함께 전송되도록 수정했다.



클라이언트코드에서 위 부분을 추가한다. 이름과 나이를 입력받고 서버에게 보낸다. 그리고 자식프로세스를 만들어 대화하는 부분은 4번코드와 같다.





대화를 할 때 메시지를 입력한 클라이언트의 이름과 나이가 함께 보여지는 것을 알 수 있었다.

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

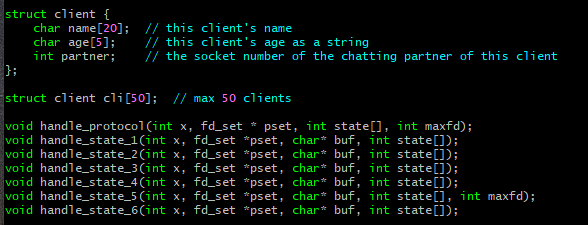
serv => cli aaa: bbb to aaa: hi

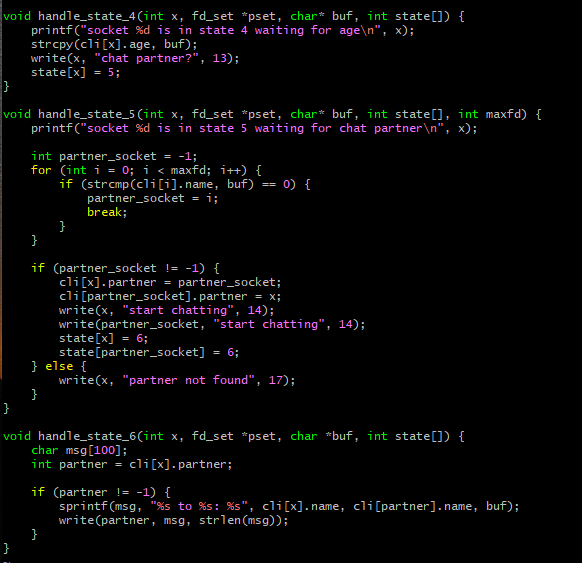
cli ccc => serv: hear me

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

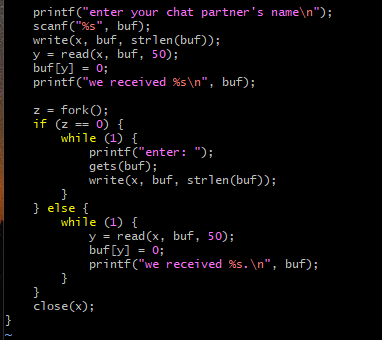
cli ddd=> serv: hi there

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

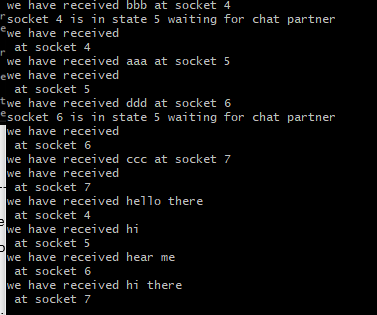


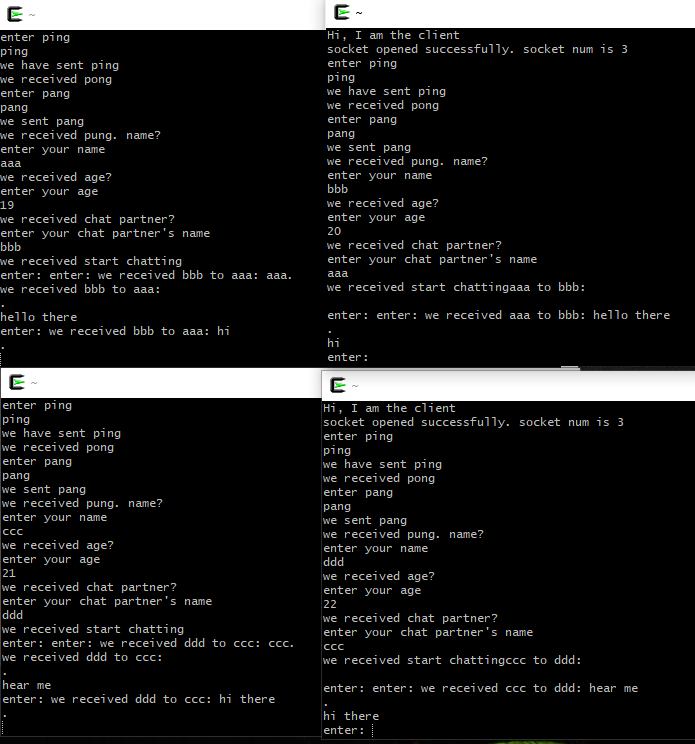


서버 코드를 위와 같이 수정한다. 구조체에 partner 변수를 추가한다. 그리고 handle\_state함수를 6개를 만든다. 3까지는 위와 같고, 4는 나이를 저장하고, 파트너를 물어본다. 5에서는 파트너의 이름에 해당하는것을 strcmp를 통해서 찾고, 해당하는 숫자 i를 cli[x].partner에 저장한다. 맞는 이름이 없다면 partner not found를 출력한다. 6에서는 sprintf를 통해 수신자, 발신자 확인이 가능하게 하고, write()으로 partner에게 메시지를 보낸다.



클라이언트 코드에서는 파트너의 이름을 받는 부분만 추가한다.





문제의 예시와 같이 타이핑했다.

aaa-bbb, ccc-ddd가 대화하는 것을 확인할 수 있었다.

------------------------------------------------------------------------------------------------------------

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

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

8) Modify your chatting server in Problem 7 such that the server checks whether the client's name is in "login.txt" file. If yes, proceed as before; if not, ask the client's name again. If the client fails to give registered name for 5 times, the server should disconnect this client.