lec

fork는 쉬움

select는 어려움 – 이걸로 multi socket client user다루기

select(..., &rset, ...); // wait for packets at sockets marked 1 in rset

rset에 1로 mapping된 소켓은 모두 다룰수 있음 – multiple socket handle 가능

selelct를 사용할 때는 스테이트 개념을 이해해야 합니다.

2번 문제는 스테이트 개념을 이해하도록 하기 위해 낸 문제입니다.

ping이나 pang이 어떤 경우에는 되고 어떤 경우에는 에러가 되도록 하려면 클라이언트(소켓 x 에 있는)가 현재 어떤 스테이트인지를 항상 추적하고 있어야 합니다.

모든 클라이언트는(즉 모든 소켓은) 처음 연결이 되면 스테이트를 1로 세팅합니다.

그리고 handle\_protocol에서는 소켓 x 에 있는 클라이언트의 스테이트가 1이면 이 클라이언트로부터 온 첫 메시지이므로 ping만 패스시킵니다.

ping을 받은 클라이언트는 스테이트가 2로 전이됩니다.

나중에 이 클라이언트로부터 다시 메시지가 오면 handle\_protocol이 이 클라이언트의 스테이트가 2인 것을 감지하고 이번에는 pang이 왔는지 확인합니다.

\* 사실은 문제 2-1에 있는 코드가 문제 2의 해답입니다. 문제 2-1에 있는 코드를 잘 생각해보기 바랍니다.

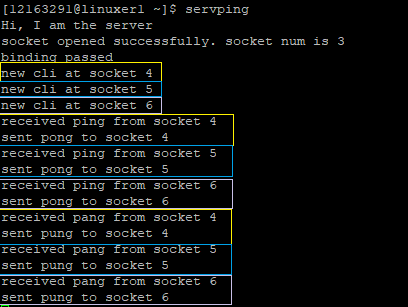
\* select를 사용할 때는 얼마나 많은 스테이트를 사용할 것인가, 각 스테이트에서는 어떤 액션을 취할 것인가를 결정하는 것이 제일 어렵습니다.

hw

1) Copy 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.

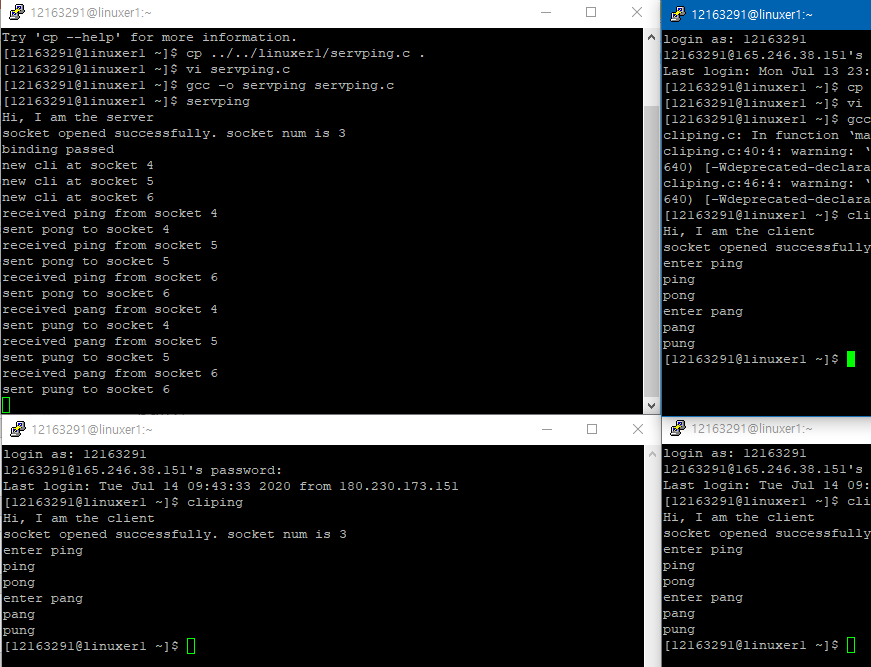
$ cp ../../linuxer1/cliping.c .

$ cp ../../linuxer1/servping.c .



cli와 연결된 socket 수가 4,5,6 3개 이므로

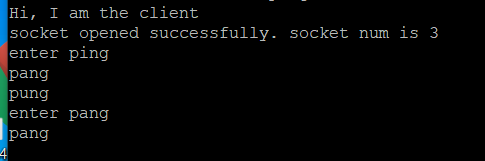
3개의 client를 handle하고 있는 것을 알 수 있습니다.



3개의 clilping을 servping에 동시에 연결시켜, ping pong 대화를 하였습니다.

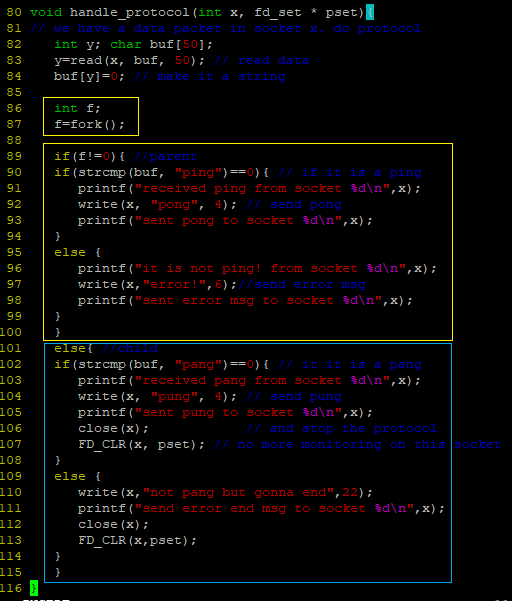
2) The server in Prob 1) cannot give error message to clients even when the client doesn't follow the protocol.

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. (if-else if문이니까)

 이렇게 된다는 의미

Modify servping.c so that it can send error message when the client sends something other than "ping" for the first message.

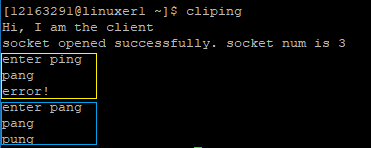
But make sure the server still sends "pung" when the client sends "pang" as the second message.



fork를 사용해 주었습니다.

우선 나와야 하는 ping 부분을 ( f !=0 ) 조건문을 통해 parent process로 돌려주었습니다.

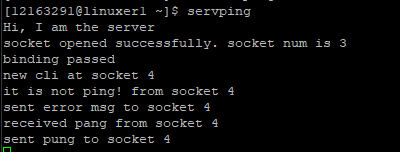
child process에 pang부분을 넣어주었습니다.



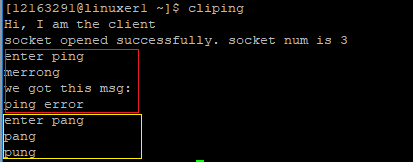
좀더 수정 필요?

..

역시 fork() 가아니다! 2-1) 문제처럼 state 상태를 확인해서 푸는문제이다. -> 다시하기

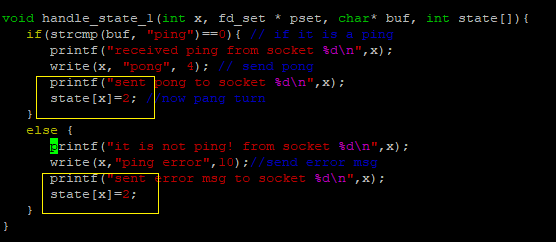


server 결과 입니다.



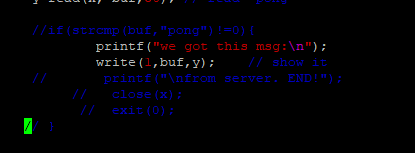
client 결과입니다.

2-1)의 코드를 변경하여 해결하였습니다.



server 에서 handle\_state\_1의 경우에는 무조건 state가 2로 갈 수 있도록 하였습니다.

즉, 무조건 pang 과정으로 가게됩니다. (ping경우에서 에러가 나더라도)



client에선, ping을 보낸후 server로부터 pang이아닌 에러값 string을 받았을 경우에

close 하는 부분을 주석처리 해주었습니다.

이를 통해, ping 경우에 에러가 나더라도 disconnection이 되지 않습니다.

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.)

- serverping.c

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

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

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

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

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

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

}

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

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

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

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

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

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

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

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.

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

}

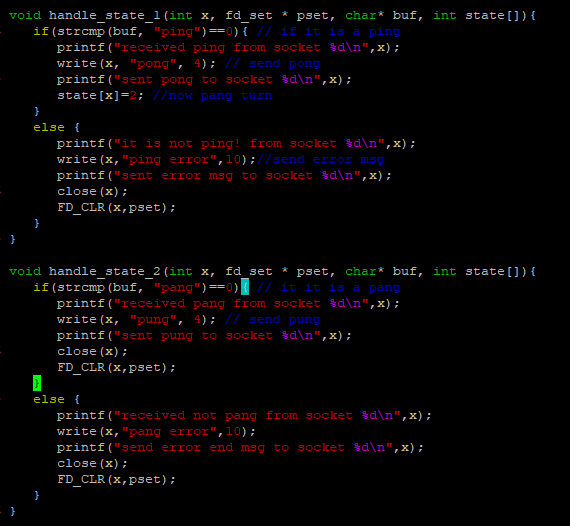
case마다 state라는 flag를 세워주는 것이라 생각하면 된다.

state == 1 인경우는 ping을 기다리는 상황이 되고,

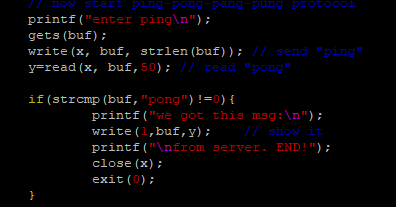
state == 2 인경우는 pang을 기다리는 상황이 된다.

state배열의 크기가 50이므로, 50가지의 case를 handle할 수 있다.

각 state 마다의 process는 내가 정할 수 있다.



server 코드는 강의노트에 주어진 대로 수정하여 실행해 주었습니다.



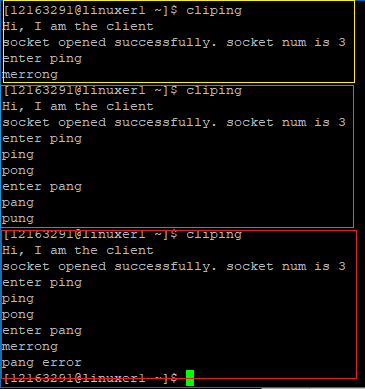
cli 코드에선, ping이후, (만약 ping을 보내지 않으면 pong이 아닌 다른 string이 오게되는 것을 이용하여)

pang이오지않게되면 = 에러가 났다는 string내용이 오게되면,

cli를 종료해주었습니다.



serverping 의 결과입니다.



cliping의 결과입니다.

각 같은색의 박스에 해당하는 부분끼리 연관된 부분입니다.

처음 노란부분에서는 처음 ping에서부터 merrong을 보내 protocol error 를 만들어주었습니다

그러자 바로 disconnect되었습니다.

다음 파란부분에서는 성공적으로 과정이 이루어지도록 하였습니다.

마지막 빨간 부분에서는 다음pang에서부터 merrong을 보내 protocol error 를 만들어주었습니다

그러자 바로 disconnect되었습니다

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

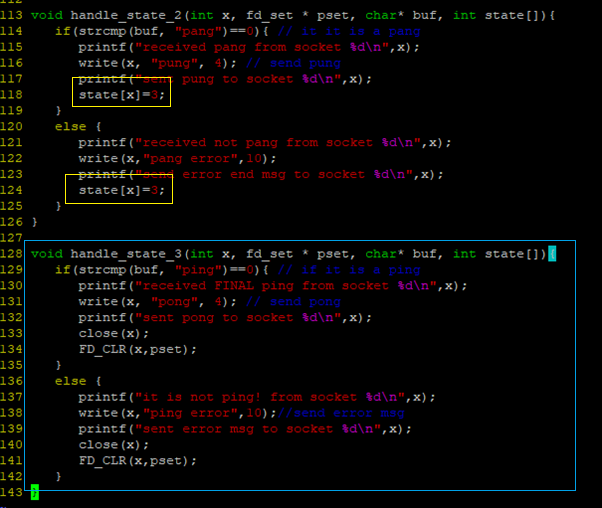
cli=>serv: ping (final ping)

serv=>cli: protocol completed

이는 server 에 state 3의 추가와,

client에서 get과 write의 데이터를 주고받는 일련 과정을 한번 더 추가함으로써

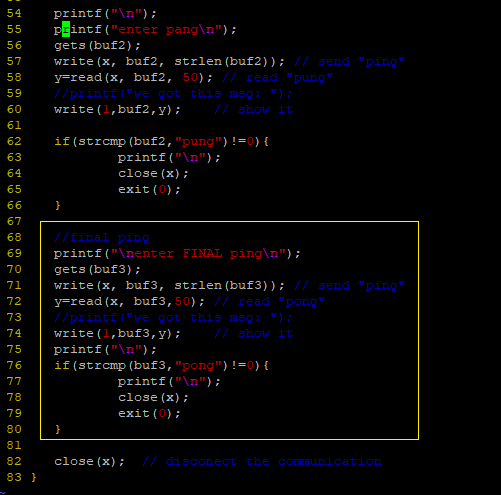
구현할 수 있습니다.



server code

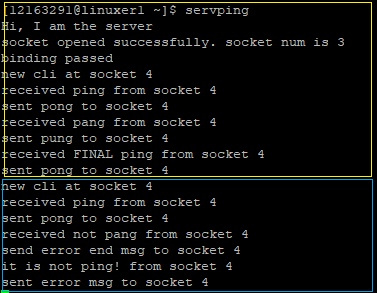
handle\_state\_2에서 끝이나면, 무조건 state 3으로 가도록 코드를 수정해주었고,

state3 코드를 첫 state1과 같이 ping을 받는 과정을 반복하여주었습니다.



client.c code

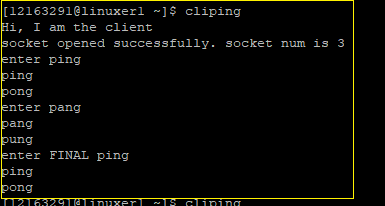
데이터를 주고받는 내용을 한번더 추가해 주었습니다.



servping.c

노란 부분은 성공적으로 state 3까지마친경우 입니다.

파란 부분은 두번째 pang과정에서 protocol error가 발생한 경우입니다.



cliping.c

성공적으로 state3까지 간후, 종료된 것을 확인할 수 있었습니다