lect12 HW

시스템프로그래밍 1분반 12180626 성시열

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.

```
[12180626@linuxer1 ~]$ cp ../../linuxer1/cliping.c .
[12180626@linuxer1 ~]$ cp ../../linuxer1/servping.c .
[12180626@linuxer1 ~]$ ls
cli         ex1     ex3      ex5.c  f1.c   f92      mycat.c   newhw4.c
cli.c       ex11    ex33     ex6    f1out  f93?     mycp      outfiles
cliping.c   ex11.c  ex33.c   ex6.c  f2     fabc     mycp.c    path
cphw4       ex13    ex3.c    ex7    f3     hw33     myecho    path.c
cphw4.c     ex13.   ex4      ex7.c  f4     hw4      myecho.c  serv
d1          ex13.c  ex44     ex8    f5     hw4.c    myexec    serv.c
d2          ex1.c   ex444    ex.8   f6     mycat    myexec.c  servping.c
d3          ex2     ex444.c  ex8.c  f7     mycat2   mysh      sw2.wav
echo        ex22    ex44.c   ex9    f8     mycat2.c mysh.c    swvader03.wav
ex0         ex22.c  ex4.c    ex9.c  f9     mycat3   myxxd     x
ex0.c       ex2.c   ex5      f1     f91    mycat3.c myxxd.c   y.c
```

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

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

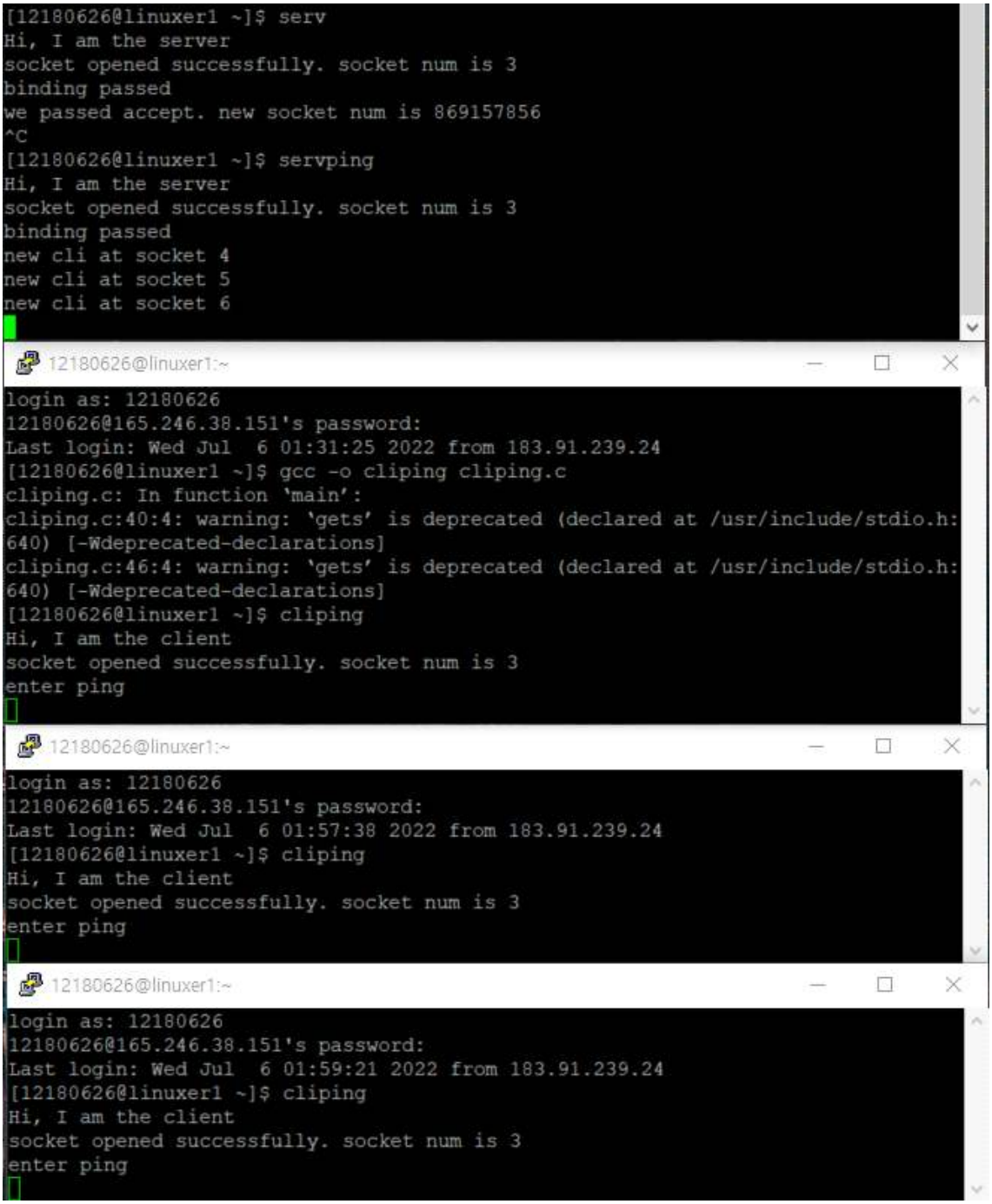
를 통해 현재 위치에 cliping.c와 servping.c을 복사한다.

```
[12180626@linuxer1 ~]$ vi cliping.c
[12180626@linuxer1 ~]$ vi servping.c
```

```
#define SERV_TCP_PORT 62608
#define SERV_ADDR "165.246.38.151"
```

```
#define SERV_TCP_PORT 62608
#define SERV_ADDR "165.246.38.151"
```

이후 vi cliping.c와 vi servping.c를 통해 port 번호와 IP 주소를 변경한다.



컴파일 후 실행해본 결과 cliping을 여러개의 putty 창에서 실행시켰을 때 servping에서 여러 client를 handling 하는 것을 확인할 수 있다.

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

```
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
enter ping
pang
pung
enter pang
pang

[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
enter ping
ping
pong
enter pang
pang
pung
```

servping을 실행시킨 뒤 새로운 putty창에서 cliping을 실행시켰다.
ping을 입력하라고 할 때 pang을 입력하면 pung을 전달받고
다시 pang을 입력받을 때는 전달받는 값이 없음을 알 수 있다.
ping을 입력하라고 할 때 ping을 입력하면 pong을 전달받고
pang을 입력하라고 할 때 pang을 입력하면 pung을 전달받는 것을 확인하였다.
첫 번째 경우에서 오류메세지를 보내도록 servping.c를 수정한다.

첫 번째 입력과 두 번째 입력을 구분하여 첫 번째 입력일 때, ping아닌 다른 것을 입력하면 오류가 발생, 두 번째 입력일 때, pang아닌 다른 것을 입력하면 오류가 발생하게 함수를 생성한다.

```
int seq[50];
void handle_protocol(int x, fd_set * pset);
void first_input(int x, fd_set * pset, char buf[], int seq[]);
void second_input(int x, fd_set * pset, char buf[], int seq[]);
```

함수 두 개를 추가로 생성하고, 몇 번째 입력인지 저장할 정수형 array seq[50]을 생성한다.

```
if (x==s1){
   s2 = accept(s1, (struct sockaddr *)&cli_addr, &xx);
   seq[s2] = 1;
   printf("new cli at socket %d\n",s2);
   FD_SET(s2, &pset);
}else{
   handle_protocol(x, &pset);
```

x값은 반복문이 반복된 횟수이며, 탐색하는 socket의 번호이다. (x == s1)은 x가 새로운 클라이언트를 받는 socket의 번호(대부분 3번)일 때 실행하고, else에서는 새로운 클라이언트를 받지 않을 때 사용한다.

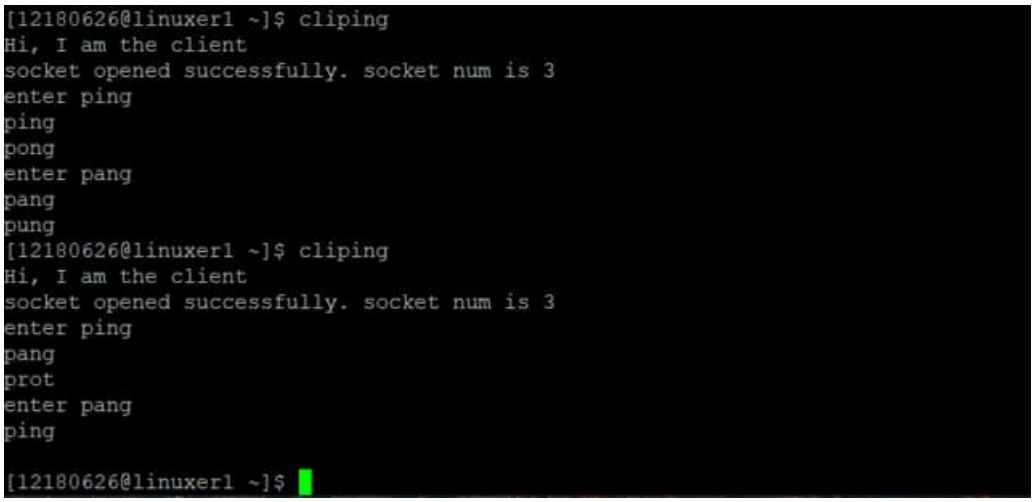
새로운 클라이언트가 들어올 때, accept로 s1에 새로운 클라이언트가 들어올 때까지 기다렸다가 새로운 클라이언트를 받는 순간 s2에는 새로운 클라이언트를 위한 socket 번호가 저장된다. 이러한 클라이언트가 들어온 자리의 seq를 1로 지정해준다. 새롭게 들어오지 않은 클라이언트는 handle_protocol 함수를 실행한다. 이 때 x 값은 3 이후에 "1"이 들어있는 socket의 index이다.

```
void handle_protocol(int x, fd_set * pset){
// we have a data packet in socket x. do protocol
   int y; char buf[50];
   y=read(x, buf, 50); // read data
   buf[y]=0; // make it a string
   if (seq[x] == 1){ // if it is a ping
      first_input(x, pset, buf, seq);
   }
   else if (seq[x] == 2){
      second_input(x, pset, buf, seq);
   }
}

void first_input(int x, fd_set * pset, char buf[], int seq[]){
   if (strcmp(buf, "ping")==0){ // if it is a ping
      printf("received ping from socket %d\n",x);
      write(x, "pong", 4); // send pong
      printf("sent pong to socket %d\n",x);
      seq[x] = seq[x] + 1;
   }
   else {
      printf("received NOT PING from socket %d\n",x);
      write(x, "protocol error", 4); // send pong
      printf("sent pung to socket %d\n",x);

      close(x);            // end stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}

void second_input(int x, fd_set * pset, char buf[], int seq[]){
   if (strcmp(buf, "pang")==0){ // if it is a ping
      printf("received pang from socket %d\n",x);
      write(x, "pung", 4); // send pong
      printf("sent pung to socket %d\n",x);
   }
   else { // if it is a pong
      printf("received NOT PANG from socket %d\n",x);
      write(x, "protocol error", 4); // send pong
      printf("sent protocol error to socket %d\n",x);

      close(x);            // end stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}
```

handle_protocol의 조건문을 첫 번째 순서일 때(seq[x] = 1)는 first_input 함수를 실행하고, 두 번째 순서일 때(seq[x] = 2)는 second_input 함수를 실행하였다.

first_input함수에서는 이전의 handle_protocol에서 사용한 코드를 바탕으로 조건문을 입력받은 buf가 ping일 때와 ping이 아닐 때로 구분하여 ping이 아닐 때(else) error를 출력한다. 올바르게 입력받았다면 seq 값을 1 더해준다.

second_input함수에서는 이전의 handle_protocol에서 사용한 코드를 바탕으로 조건문을 입력받은 buf가 pang일 때와 pang이 아닐 때로 구분하여 pang이 아닐 때(else) error를 출력한다.



컴파일 후 실행해본 결과 올바르게 입력하였을 때 ping pong pang pung이 잘 진행된 모습이다. 새로운 cliping을 실행하여 첫 번째 입력때 pang을 입력하였을 때, servping 측에서 에러 메시지가 출력되고, cliping 측에서도 ping pong pang pung이 잘 진행되지 않는 것을 확인할 수 있다.

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
                    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.
          ……..
}
```

```
[12180626@linuxer1 ~]$ vi servping.c
[12180626@linuxer1 ~]$ gcc -o servping servping.c
[12180626@linuxer1 ~]$ servping
Hi, I am the server
socket opened successfully. socket num is 3
binding passed
new cli at socket 4
received pang from socket 4
sent pung to socket 4
received pang from socket 4
sent pung to socket 4
new cli at socket 4
```

```
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
enter ping
ping
pong
enter pang
pang
pung
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
enter ping
pang
protocol error
enter pang
ping

[12180626@linuxer1 ~]$
```

위에 주어진 코드대로 servping.c를 수정한 뒤 컴파일하고 실행하였다.
첫 번째 입력에서 ping, 두 번째 입력에서 pang을 입력하면 ping pong pang pung이 잘 진행됨을 알 수 있다.
새로운 클라이언트를 불러와 첫 번째 입력에서 pang을 입력하면 에러 메시지가 출력되고 원활히 진행되지 않음을 확인할 수 있다.

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

```
void handle_state_1(int x, fd_set * pset, char buf[], int state[]){
   if (strcmp(buf, "ping")==0){ // if it is a ping
      printf("cli => serv: ping\n");
      write(x, "pong", 4); // send pong
      printf("serv => cli: pong\n");
      state[x] = 2;
   }
   else {
      write(x, "protocol error", 14); // send ping
      close(x);              // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}

void handle_state_2(int x, fd_set * pset, char buf[], int state[]){
   if (strcmp(buf, "pang")==0){ // if it is a ping
      printf("cli => serv: pang\n");
      write(x, "pung", 4); // send pong
      printf("serv => cli: pung\n");
   }
   else { // if it is a pang
      write(x, "protocol error", 14); // send ping
      close(x);              // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
```

먼저 위 형태로 만들기 위해 servping.c의 hadle_state_1, hadle_state_2를 수정하였다.

```
[12180626@linuxer1 ~]$ vi servping.c
[12180626@linuxer1 ~]$ gcc -o servping servping.c
[12180626@linuxer1 ~]$ servping
Hi, I am the server
socket opened successfully. socket num is 3
binding passed
new cli at socket 4
cli => serv: ping
serv => cli: pong
cli => serv: pang
serv => cli: pung
```

컴파일 후 실행결과 원하는 형태로 잘 출력됨을 확인하였다.

클라이언트의 마지막 ping을 인식하기 위해서 상태 하나의 경우를 추가한다.

```
void handle_state_2(int x, fd_set * pset, char buf[], int state[]){
   if (strcmp(buf, "pang")==0){ // if it is a ping
      printf("cli => serv: pang\n");
      write(x, "pung", 4); // send pong
      printf("serv => cli: pung\n");
      state[x] = 3;
   }
   else { // if it is a pang
      write(x, "protocol error", 14); // send pung
      close(x);              // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}
```

먼저 handle_state_2가 오류가 아닐 때 실행문을 마치고 state[x] = 3으로 지정하는 코드를 추가한다.

```
int state[50];
void handle_protocol(int x, fd_set * pset, int state[]);
void handle_state_1(int x, fd_set * pset, char * buf, int state[]);
void handle_state_2(int x, fd_set * pset, char * buf, int state[]);
void handle_state_3(int x, fd_set * pset, char * buf, int state[]);
```

```
void handle_protocol(int x, fd_set * pset, int state[]){
// we have a data packet in socket x. do protocol
   int y; char buf[50];
   y=read(x, buf, 50); // read data
   buf[y]=0; // make it a string
   if (state[x] == 1){ // if it is a ping
      handle_state_1(x, pset, buf, state);
   }
   else if (state[x] == 2){
      handle_state_2(x, pset, buf, state);
   }
   else if (state[x] == 3){
      handle_state_3(x, pset, buf, state);
   }
}
```

state[x] == 3일 때, 마지막 프로토콜임을 알려주는 함수인 handle_state_3이라는 함수를 추가하고 handle_protocol 함수 내에서 state[x]==3일 때 handle_state_3를 호출한다.

```
void handle_state_3(int x, fd_set * pset, char * buf, int state[]){
   if (strcmp(buf, "ping")==0){ // if it is a ping
      printf("cli => serv: ping(final ping)\n");
      write(x, "protocol completed", 18); // send pong
      printf("serv => cli: protocol completed\n");
      close(x);
      FD_CLR(x, pset);
   }
   else { // if it is a pang
      write(x, "protocol error", 14); // send pung
      close(x);              // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}
```

handle_state_3에서는 ping을 입력 받았을 때 마지막 ping임을 전달하고 close를 통해 종료시킨다.

ping이 아닌 다른 문자열이 입력될 때는 이전과 같이 오류 메시지를 출력한 뒤 close를 통해 종료한다.

```
    // now start ping-pong-pang-pung protocol
    printf("enter ping\n");
    gets(buf);
    write(x, buf, strlen(buf)); // send "ping"
    y=read(x, buf,50); // read "pong"
    write(1,buf,y);     // show it
    printf("\n");

    printf("enter pang\n");
    gets(buf);
    write(x, buf, strlen(buf)); // send "ping"
    y=read(x, buf, 50); // read "pong"
    write(1,buf,y);
    printf("\n");

    printf("enter ping\n");
    gets(buf);
    write(x, buf, strlen(buf)); // send "ping"
    y=read(x, buf, 50); // read "pong"
    write(1,buf,y);
    printf("\n");

    close(x);  // disconnect the communication
```

원래 cliping.c 코드를 보면 ping 한번, pang 한번 총 두 번만 입력받고 close를 했다면 ping 한번을 더 받아서 저장 후 출력하는 코드를 추가하였다.

```
[12180626@linuxer1 ~]$ vi servping.c
[12180626@linuxer1 ~]$ gcc -o servping servping.c
[12180626@linuxer1 ~]$ servping
Hi, I am the server
socket opened successfully. socket num is 3
binding passed
new cli at socket 4
cli => serv: ping
serv => cli: pong
cli => serv: pang
serv => cli: pung
cli => serv: ping(final ping)
serv => cli: protocol completed
```

```
[12180626@linuxer1 ~]$ vi cliping.c
[12180626@linuxer1 ~]$ gcc -o cliping cliping.c
cliping.c: In function 'main':
cliping.c:40:4: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:
640) [-Wdeprecated-declarations]
cliping.c:47:4: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:
640) [-Wdeprecated-declarations]
cliping.c:54:4: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:
640) [-Wdeprecated-declarations]
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
enter ping
ping
pong
enter pang
pang
pung
enter ping
ping
protocol completed
```

컴파일 후 실행한 결과 ping pong pang pung이 잘 진행된 모습이고, 이후 ping을 입력하니 프로토콜이 완료됐다는 문자열과 함께 프로토콜이 종료됨을 알 수 있다.