lect12 HW

시스템프로그래밍 1분반 12180626 성시열

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
.....

```
// now start ping-pong-pang-pung protocol
printf("cli => serv: ");
gets(buf);
printf(": serv => cli");
if(strcmp(buf, "ping")){
   close(x);
   exit(1);
}
write(x, buf, strlen(buf)); // send "pong"
y=read(x, buf,50); // read "pong"
write(1,buf,y);    // show it
printf("\n");
if(strcmp(buf, "pong")){
   close(x);
   exit(1);
}

printf("cli => serv: ");
gets(buf);
printf(": serv => cli");
if(strcmp(buf, "pang")){
   close(x);
   exit(1);
}
write(x, buf, strlen(buf)); // send "pung"
y=read(x, buf,50); // read "pong"
write(1,buf,y);    // show it
printf("\n");
if(strcmp(buf, "pung")){
   close(x);
   exit(1);
}

printf("Protocol completed. Start cheating.\n");
```

cli.c의 코드를 수정한다. 먼저 ping pong pang pung 프로토콜을 진행한다. 해당 단어를 입력받지 못하거나 전달을 못 받을 때는 오류로 처리하여 프로그램을 종료한다.

```
int f;
f = fork();
if(f==0){ //child
   for(;;){
      y = read(x, buf, 50); //1 = client send
      write(1, buf, y);
      printf("\n");
   }
}
else {
   for(;;){
      printf("cli at socket %d => serv : ", x);
      gets(buf);
      if(strcmp(buf, "bye")==0){
         close(x);
         exit(1);
      }
      write(x, buf, strlen(buf));
   }
}
```

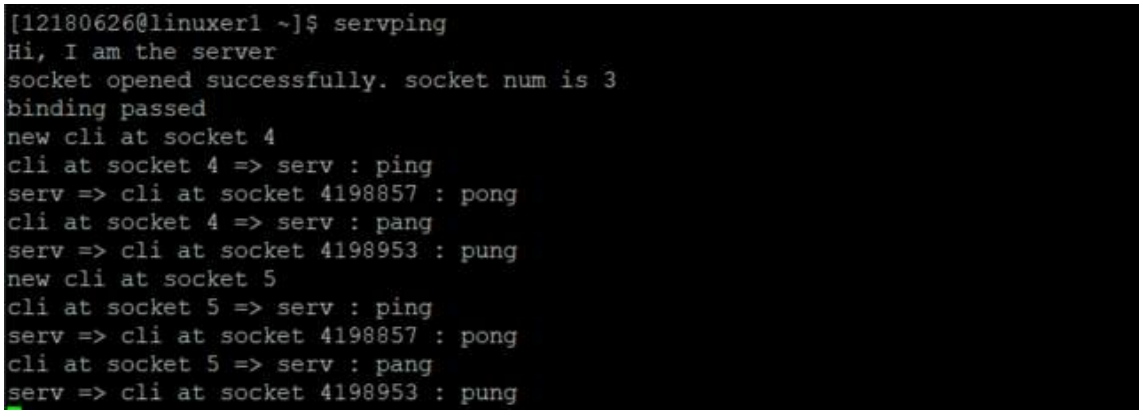
이후 fork()를 활용하여 child일 때는 값을 읽어오고, parent일 때는 값을 전송하는 역할을 나누어주었다. 값을 전송할 때는 bye라는 문자열을 입력하면 프로그램이 종료되도록 하였다.

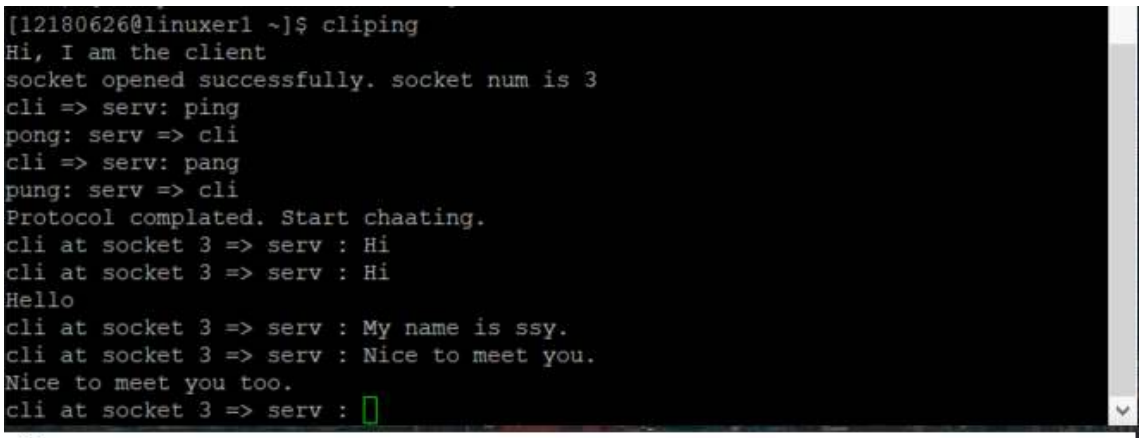
```
int socket_num[50];
int state[50];
void handle_protocol(int x, fd_set * pset, int state[]);
void handle_state_1(int x, fd_set * pset, char * buf, int state[]);
void handle_state_2(int x, fd_set * pset, char * buf, int state[]);
void handle_state_3(int x, fd_set * pset, char * buf, int state[]);
```

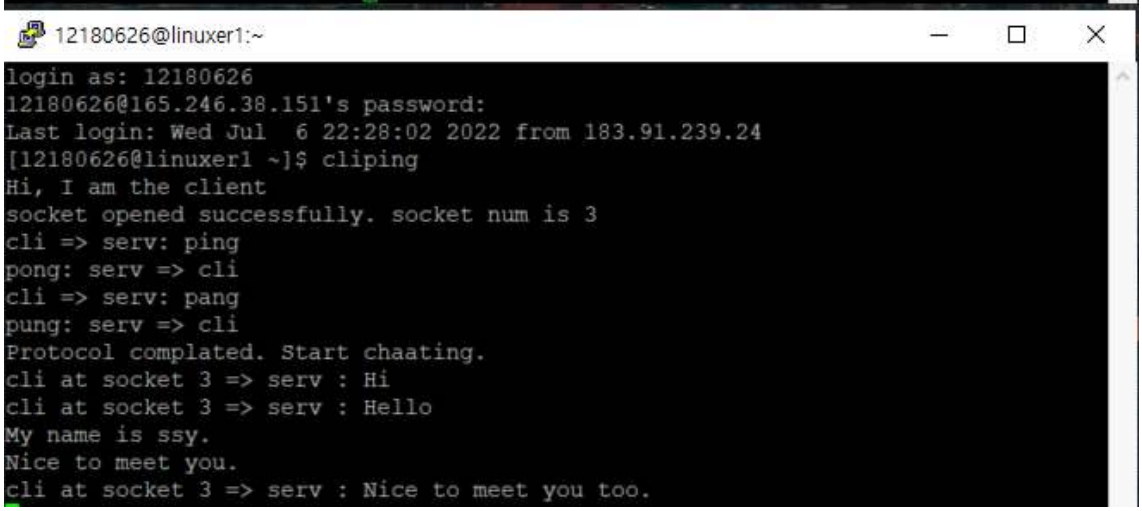
먼저 socket의 번호를 저장할 socket_num 배열을 선언한다.

```
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 at socket %d => serv : ping\n", x);
      write(x, "pong", 4); // send pong
      printf("serv => cli at socket %d : pong\n");
      state[x] = 2;
   }
   else {
      write(x, "protocol error", 14); // send pang
      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 at socket %d => serv : pang\n", x);
      write(x, "pung", 4); // send pong
      printf("serv => cli at socket %d : pung\n");
      state[x] = 3;
   }
   else { // if it is a pang
      write(x, "protocol error", 14); // send pang
      close(x);              // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}

void handle_state_3(int x, fd_set * pset, char * buf, int state[]){
   int i;
   for(i = 0; i < 50; i++){
      if(socket_num[i] == 1){
         if(i != x){
            write(i, buf, strlen(buf));
         }
      }
   }
}
```

handle_state_1과 handle_state_2는 기존과 비슷하게 가되, handle_state_3에서는 socket_num 배열을 0번째부터 49번째까지 탐색하면서 값이 1인 인덱스에서 그 값이 packet을 기다리는 자기 자신 socket이 아닐 때 buf의 값을 i라는 파일번호에 buf의 길이만큼 저장한다.







컴파일 후 실행결과 ping pong pang pung 과정을 거친 클라이언트끼리 채팅이 가능한 것을 알 수 있다.

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.

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"

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

struct client{
   char name[20];
   char age[5];
};
struct client cli[50];

int socket_num[50];
```

servping.c 코드를 수정하여 client 구조체를 추가한다. 이 때 client는 name과 age라는 속성을 지닌다. 이후 struct client cli[50];를 통해 최대 50명의 클라이언트를 받는다.

```
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 in 1 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){
      strcpy(cli[x].name, buf);
      write(x, "age?", 4);
      state[x] = 4;
   }
   else if (state[x] == 4){
      strcpy(cli[x].age, buf);
      state[x] = 5;
      char say[100];
      strcpy(say, cil[x].name);
      strcat(say, ": start chatting");
      write(x, say, strlen(say));
   }
   else if (state[x] == 5){
      handle_state_3(x, pset, buf, state);
   }
}
```

이후 첫 번째 입력부터 다섯 번째 입력까지 상태에 맞는 행동을 지정한다.

첫 번째 입력, 두 번째 입력, 마지막 입력은 기존과 같이 handle_state_1,2,3 함수를 불러온다.

세 번째 입력일 때는 buf에 저장된 문자열을 x번째 클라이언트의 이름에 넣고, 클라이언트의 나이를 물어보는 "age?"문자열을 x를 통해 클라이언트에 전송한다. 이후 state[x] = 4로 상태를 변경한다.

네 번째 입력일 때는 buf에 저장된 문자열을 x번째 클라이언트의 나이에 넣고, state[x] = 4로 상태를 변경한다. 이후 say라는 문자열을 저장할 수 있는 변수를 선언하고, x번째 클라이언트의 이름을 say에 넣어준다. 이후 say 뒤에 ": start chatting"이라는 문자열을 이어붙인다. 이후 x에 "(클라이언트 이름) : start chatting"의 문자열을 write한다.

```
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 at socket %d => serv : pang\n", x);
      write(x, "pung. name?", 12); // send pong
      printf("serv => cli at socket %d : pung. name?\n");
      state[x] = 3;
   }
   else { // if it is a pang
      write(x, "protocol error", 14); // send pong
      close(x);             // and stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}

void handle_state_3(int x, fd_set * pset, char * buf, int state[]){
   char say[100];
   int i;
   for(i = 0; i < 50; i++){
      if(socket_num[i] == 1){
         if(i != x){
            strcpy(say, cli[x].name);
            strcat(say, " ");
            strcat(say, cli[x].age);
            strcat(say, " to ");
            strcpy(say, cli[i].name);
            strcat(say, " ");
            strcat(say, cli[i].age);
            strcat(say, ": ");
            strcat(say, buf);
            write(i, say, strlen(say));
         }
      }
   }
}
```

handle_state_1 함수는 기존과 그대로 작성한다.

handle_state_2 함수는 pang을 입력하여 넘겨주고, pung을 넘겨 받는 함수인데, pung을 넘겨받을 때 "pung. name?" 문자열을 넘겨준다.

handle_state_3 함수는 socket_num의 인덱스를 탐색하며 1인 값 중 클라이언트 자신이 아닐 때, say라는 문자열을 완성시킨다. say는 x번째 클라이언트의 이름과 나이, i번째 클라이언트의 이름과 나이를 이어 붙이고, 이후 buf를 통해 입력받은 문자열이 이어진 문자열이다. 이는 누가 누구에게 채팅을 보냈는지 한눈에 보기 위함이다.

이후 say 문자열을 i라는 파일에 say 문자열의 길이만큼 write한다.

```
   printf("cli => serv: ");
   fflush(stdout);
   gets(buf);
   printf("serv => cli: ");
   fflush(stdout);
   if(strcmp(buf, "ping")){
      close(x);
      exit(1);
   }
   write(x, buf, strlen(buf)); // send "ping"
   y=read(x, buf,50); // read "pong"
   write(1,buf,y);    // show it
   printf("\n");
   if(strcmp(buf, "pong")){
      close(x);
      exit(1);
   }

   printf("cli => serv: ");
   fflush(stdout);
   gets(buf);
   printf("serv => cli: ");
   fflush(stdout);
   if(strcmp(buf, "pang")){
      close(x);
      exit(1);
   }
   write(x, buf, strlen(buf)); // send "ping"
   y=read(x, buf,50); // read "pong"
   write(1,buf,y);    // show it
   printf("\n");
   if(strcmp(buf, "pung")){
      close(x);
      exit(1);
   }

   printf("cli => serv: ");
   fflush(stdout);
   gets(buf);
   strcpy(name, buf);
   write(x, buf, strlen(buf)); // send "ping"
   y=read(x, buf,50); // read "pong"
   write(1,buf,y);    // show it
   printf("\n");
   printf("cli => serv: ");
   fflush(stdout);
   gets(buf);
   write(x, buf, strlen(buf));
```

```
int f;
f = fork();
int i;
if(f==0){ //child
   for(i=0; i<10; i++){
      y = read(x, buf, 50); //x = client send
      write(1, buf, y);
      printf("\n");
   }
}
else {
   for(i=0; i<10; i++){
      printf("cli at socket %d => serv : ", x);
      gets(buf);
      if(strcmp(buf, "bye")==0){
         close(x);
         exit(1);
      }
      write(x, buf, strlen(buf));
   }
}
```

fflush(stdout)는 출력버퍼에 있는 데이터를 즉시 출력하는 함수이다. 클라이언트에서 name과 age를 받을 수 있게 gets와 read, write들을 순서에 맞게 추가하였다.

```
[12180626@linuxer1 ~]$ servping
Hi, I am the server
socket opened successfully. socket num is 3
binding passed
new cli at socket 4
cli at socket 4 => serv : ping
serv => cli at socket 4200017 : pong
cli at socket 4 => serv : pang
serv => cli at socket 4200113 : pung. name?
new cli at socket 5
cli at socket 5 => serv : ping
serv => cli at socket 4200017 : pong
cli at socket 5 => serv : pang
serv => cli at socket 4200113 : pung. name?
new cli at socket 6
cli at socket 6 => serv : ping
serv => cli at socket 4200017 : pong
cli at socket 6 => serv : pang
serv => cli at socket 4200113 : pung. name?
```

```
12180626@linuxer1:~
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
cli => serv: ping
serv => cli: pong
cli => serv: pang
serv => cli: pung. name?
cli => serv: aaa
serv => cli aaa: age?
cli => serv: 19
cli at socket 3 => serv : aaa: start chatting
```

```
12180626@linuxer1:~
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
cli => serv: ping
serv => cli: pong
cli => serv: pang
serv => cli: pung. name?
cli => serv: bbb
serv => cli bbb: age?
cli => serv: 22
cli at socket 3 => serv : bbb: start chatting
```

```
12180626@linuxer1:~
[12180626@linuxer1 ~]$ cliping
Hi, I am the client
socket opened successfully. socket num is 3
cli => serv: ping
serv => cli: pong
cli => serv: pang
serv => cli: pung. name?
cli => serv: ccc
serv => cli ccc: age?
cli => serv: 21
cli at socket 3 => serv : ccc: start chatting
```

컴파일 한 뒤 servping을 실행시키고 3개의 cliping을 실행시켜보았다. 각각 ping pong pang pung을 거치고 이름과 나이를 입력하였다.

```
cli at socket 3 => serv : hello there
cli at socket 3 => serv : bbb 19 to aaa 22: hi
```

```
aaa 22 to bbb 19: hello there
hi
cli at socket 3 => serv :
```

```
aaa 22 to ccc 21: hello there
bbb 19 to ccc 21: hi
```

aaa에서 채팅을 보낸 결과 bbb와 ccc 클라이언트에서 누가 누구에게 보냈는지와 채팅 내용이 전달됨을 확인하였다.

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.
코드를 대폭 수정하였다.

servping.c

```
struct client{
   char name[20];
   char age[5];
   int partner;
};
struct client cli[50];

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_state_4(int x, fd_set * pset, char * buf, int state[]);
void handle_state_5(int x, fd_set * pset, char * buf, int state[]);
void handle_state_6(int x, fd_set * pset, char * buf, int state[]);
```

먼저 채팅을 주고받을 파트너 저장할 partner라는 정수형 변수를 구조체에 추가하였다. 또한, 모든 state에 대해 handle_protocol에서는 함수호출만을 진행하고, handle_state_1 ~ 6을 통해 각 state에 맞는 행동을 실행한다.
socket_num 배열은 새로운 클라이언트가 들어올 때 1로 지정하고 이후에 조건문에 활용하기 위해 추가하였으나, state[x] == 3이라는 조건문을 통해 새로운 클라이언트인지 판별할 수 있게 되었다.

```
   for(i=0;i<1000;i++){ // should be infinite loop in real life
      rset=pset;   // step 2
      select(maxfd, &rset, NULL, NULL, NULL); // step 3
      // now we have some packets
      for(x=0;x<maxfd;x++){ // check which socket has a packet
         if (FD_ISSET(x, &rset)){ // socket x has a packet
            // s1 is a special socket for which we have to do "accept"
            // otherwise do ping-pong-pang-pung
            if (x==s1){ // new client has arrived
               // create a socket for this client
               s2 = accept(s1, (struct sockaddr *)&cli_addr, &xx);
               printf("new cli at socket %d\n",s2);
               state[s2] = 1;
               FD_SET(s2, &pset); // and include this socket in pset
            }else{ // data packet, do ping-pong-pang-pung protocol
               handle_protocol(x, &pset, state);
            }
         }
      }
   }
```

main 루프를 20번 반복에서 1000번 반복으로 변경해주면서 클라이언트와 프로토콜을 진행하다가 금방 끝날 경우를 방지한다. x==s1 즉 새로운 클라이언트가 들어올 때 state[s2] = 1로 변경해주어 handle_protocol 내 함수가 원활히 진행될 수 있게 한다.

```
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);
   }
   else if (state[x] == 4){
      handle_state_4(x, pset, buf, state);
   }
   else if (state[x] == 5){
      handle_state_5(x, pset, buf, state);
   }
   else if (state[x] == 6){
      handle_state_6(x, pset, buf, state);
   }
}
```

handle_protocol에서는 파일번호 x에 저장된(client로부터 넘겨받은) 값을 최대 50byte 만큼 read 하여 buf에 저장한다. 이후 해당 값을 문자열로 마감하기 위해 buf[y] = 0을 해준다. 이후 state[x]의 값에 따라 해당하는 함수들을 호출한다.

```
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 at socket %d => serv : ping\n", x);
      write(x, "pong", 4); // send pong
      printf("serv => cli at socket %d : pong\n", x);
      state[x] = 2;
   }
   else {
      write(x, "protocol error", 14); // send pung
      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 at socket %d => serv : pang\n", x);
      write(x, "pung. name?", 12); // send pong
      printf("serv => cli at socket %d : pung. name?\n", x);
      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
   }
}
```

handle_state_1, 2는 ping pong pang pung 관련 부분이다. 위 handle_protocol에서 client로부터 넘어온 x값을 read한 buf 값이 ping일 때, 출력문을 출력하고 "pong"을 x에 저장한 뒤, state[x]를 2로 바꿔준다. x에 저장된 값은 client로 보내지는 값이다.

이후 state[x]가 2일 때 handle_state_2를 실행한다.
상위 반복문에 의해 다시 handle_protocol에서 client로부터 넘어온 x값을 read한 buf 값이 pang일 때, 출력문을 출력하고 "pung. name?"을 x에 저장한 뒤, state[x]를 2로 바꿔준다. x에 저장된 값은 client로 보내지는 값이다. 클라이언트는 pung. name?을 확인하고 자신의 이름을 입력하여 server로 전송하게 된다. 이 값은 다시 상위 반복문에 의해 buf에 저장되게 될 것이다.
두 함수의 else는 ping / pang이 입력되지 않았을 때 진행하는 오류처리이다.

```
void handle_state_3(int x, fd_set * pset, char * buf, int state[]){
   printf("cli at socket %d => serv : %s\n", x, buf);
   strcpy(cli[x].name, buf);
   write(x, "age?", 4);
   printf("serv => cli at socket %d : age?", x);
   state[x] = 4;
}

void handle_state_4(int x, fd_set * pset, char * buf, int state[]){
   printf("cli at socket %d => serv : %s\n", x, buf);
   strcpy(cli[x].age, buf);
   write(x, "chat partner?", 13);
   printf("serv => cli %s: chat partner?\n", cli[x].name);
   state[x] = 5;
}
```

handle_state_3에서는 클라이언트로부터 입력받은 이름을 출력문에 포함하여 출력하고 strcpy를 통해 cli[x].name에 저장한다.
"age?"라는 문자열을 서버에서 클라이언트에게 보내도록 x에 저장한다.
이후 출력문을 출력하고 state[x]를 4로 저장한다.
handle_state_4에서는 클라이언트로부터 입력받은 나이를 출력문에 포함하여 출력하고 strcpy를 통해 cli[x].age에 저장한다.
"chat partner?"라는 문자열을 서버에서 클라이언트에게 보내도록 x에 저장한다.
이후 출력문을 출력하고 state[x]를 5로 저장한다.

```
void handle_state_5(int x, fd_set * pset, char * buf, int state[]){
   char sbuf[100];
   printf("cli %s => serv : %s\n", cli[x].name, buf);
   int i;
   for(i = 0; i < 50; i++){
      if(strcmp(cli[i].name, buf)==0){
         cli[x].partner = i;
         break;
      }
   }
   write(x, "start chatting", 14);
   printf("serv => cli %s : start chatting\n", cli[x].name);
   state[x] = 6;
}

void handle_state_6(int x, fd_set * pset, char * buf, int state[])
   char sbuf[100];
   int p;
   p = cli[x].partner;
   printf("cli %s => serv : %s\n"), cli[x].name, buf;
   sprintf(sbuf, "%s to %s : %s", cli[x].name, cli[p].name, buf);
   printf("serv => cli %s : \"%s\"\n", cli[p].name, sbuf);
   write(p, sbuf, strlen(sbuf));
}
```

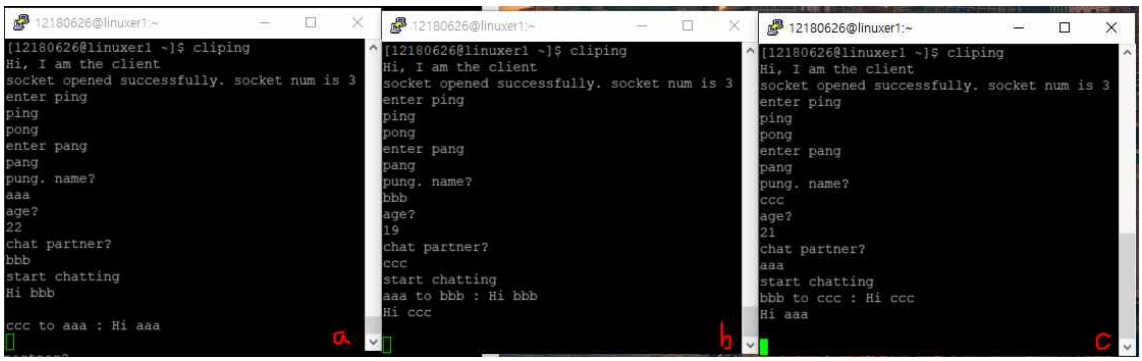
handle_state_5에서는 클라이언트로부터 입력받은 partner의 이름을 출력하고, sbuf라는 문자열을 저장할 변수를 생성한다. 이후 클라이언트 목록을 탐색하며 클라이언트 이름과 입력

받은 값과 같을 때, 클라이언트 parter을 그 번호로 저장한다. 이후 반복문을 빠져나온다.
필요한 정보를 모두 받았으므로 start chatting이라는 문자열을 x에 저장하여 클라이언트에게 전송한다. 이후 servping 측에도 출력을 해준 뒤, state[x] = 6으로 변경한다.
handle_state_6에서는 어떤 클라이언트가 어떤 클라이언트에게 어떤 메시지를 보냈는지 나타낸다. sprintf()를 활용하여 sbuf에 출력하고 싶은 문장을 만들고 printf()를 통해 출력한다.
여기서 cli[x]는 보내는 클라이언트, cli[p]는 x의 파트너인 받는 클라이언트를 의미한다.
write를 통해 해당 문장을 받는 클라이언트인 p파일에 저장한다.

cliping.c

```
   // now start ping-pong-pang-pung protocol
   int i;
   for(i=0; i<5; i++){
      if(i==0)
         printf("enter ping\n");
      else if(i==1)
         printf("enter pang\n");
      gets(buf);
      write(x, buf, strlen(buf));
      y = read(x, buf, 50);
      write(1, buf, y);
      printf("\n");
   }

   int f;
   f = fork();
   if(f==0){ //child reading
      for(i=0; i<10; i++){
         y = read(x, buf, 50); //x = client send
         write(1, buf, y);
         printf("\n");
      }
   }
   else { //parent writing
      for(i=0; i<10; i++){
         //printf("cli at socket %d => serv : ", x);
         gets(buf);
         if(strcmp(buf, "bye")==0){
            close(x);
            exit(1);
         }
         write(x, buf, strlen(buf));
         printf("\n");
      }
   }
   close(x);
}
```

먼저 ping, pang, 이름, 나이, 파트너 총 5번의 입력이 필요하므로 5번 반복해주는 반복문을 생성한다. 입력을 하면 buf에 저장이 되고, buf의 문자열 길이만큼 x에 저장된다. 또한 x 파일에 저장된 문자열을 buf에 50만큼 저장하고, 이를 1번 파일(출력장치)에 저장하여 읽어온 byte만큼 출력한다.
첫 번째 반복할 때는 ping을 두 번째 반복할 때는 pang을 입력할 수 있도록 printf()을 활용한다.
fork()를 통해 child와 parent로 분류된다. child일 때는 읽어오고, parent일 때는 작성하여 보내는 역할을 정한다. 이후 이에 맞는 코드를 작성한다.



컴파일 후 출력해본 결과 파트너로 지정한 클라이언트에게만 메시지가 전달되는 것을 알 수 있다.

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
.....

cliping7.c

```
// now start ping-pong-pang-pung protocol
int i;
for(i=0; i<4; i++){
   if(i==0)
      printf("enter hello\n");
   gets(buf);
   write(x, buf, strlen(buf));
   y = read(x, buf, 50);
   write(1, buf, y);
   printf("\n");
}
```

이전 문제에서 5번 반복하던 코드를 4번만 반복한다.
이후 ping pong pang pung이 아닌 문제에서 주어진대로 hello를 사용자가 입력할 수 있도록 printf()를 사용한다.

servping7.c

handle_protocol은 state[x]가 1부터 5일 때 각각 handle_state_1, 2, 3, 4, 5를 실행한다.

```
void handle_state_1(int x, fd_set * pset, char * buf, int state[]){
   if (strcmp(buf, "hello")==0){
      printf("cli at socket %d => serv : hello\n", x);
      write(x, "name?", 5);
      printf("serv => cli at socket %d : name?\n", x);
      state[x] = 2;
   }
   else {
      write(x, "protocol error", 14); // send pung
      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[]){
   printf("cli at socket %d => serv %s\n", x, buf);
   strcpy(cli[x].name, buf);
   write(x, "ready?", 6);
   printf("serv => cli %s : ready?\n", cli[x].name);
   state[x] = 3;
}
```

handle_state_1에서는 클라이언트로부터 받은 buf라는 값이 hello가 맞는지 비교하고 맞다면 if문을 실행한다. 몇 번 클라이언트로부터 왔는지 출력하고 name?이라는 문자열을 클라이언트로 보내기 위해 write(x, ...)을 사용한다. 이후 출력문을 작성하고
state[x]를 2로 변경한다.
else는 hello라는 문자열로 시작하지 않을 때 진행할 오류처리다.
handle_state_2에서는 클라이언트로부터 받은 buf라는 값을 포함한 출력문을 출력하고
그 buf 값을 cli[x].name에 저장한다. 여기서 x는 새로 연결된 클라이언트의 소켓번호이다.
이후 ready?라는 문자열을 클라이언트에 전송하기 위해 write(x)를 사용한다.
이후 출력문을 출력하고, state[x]를 3으로 변경한다.

```
void handle_state_3(int x, fd_set * pset, char * buf, int state[]){
   if(strcmp(buf, "yes")==0){
      printf("cli %s => serv : yes\n", cli[x].name);
      char sbuf[100];
      strcpy(sbuf, "client list (");
      int i;
      for(i=0; i<50; i++){
         if(x != i && state[i]>=3){ // 3 : new client
            strcat(sbuf, cli[i].name);
            strcat(sbuf, " ");
         }
      }
      strcat(sbuf, ")");
      write(x, sbuf, strlen(sbuf));
      state[x] = 4;
   }
   else{
      write(x, "protocol error", 14); // send pung
      close(x);                       // end stop the protocol
      FD_CLR(x, pset); // no more monitoring on this socket
   }
}
```

handle_state_3은 클라이언트로부터 입력받는 값인 buf가 "yes"와 같을 때, 해당 실행문을 실행하고, 그렇지 않을 때는 error 처리를 한다.
조건문에서는 임시로 문자열을 저장할 sbuf 선언하고, 문제 형식에 맞게 "client list ("를 출력한다. 이후 반복문을 통해 자기 자신이 아니면서 state[i]가 3보다 큰 경우의 조건을 걸어 이미 연결되어있는 클라이언트들을 모두 출력한다. 조건이 3보다 큰 경우인 이유는 state[x]가 3보다 커야지 연결이 완료된(ready?에 대한 답변으로 yes가 온) 상태이기 때문에 그러한 클라이언트를 파악하기 위함이다.
이후 write를 통해 sbuf에 저장된 "client list (... ... ...)"를 해당 길이만큼 x에 write하고, state[x]를 4로 변경한다.

```
void handle_state_4(int x, fd_set * pset, char * buf, int state[]){
   printf("cli %s => serv : %s\n", cli[x].name, buf);
   int i;
   for(i = 0; i < 50; i++){
      if(strcmp(cli[i].name, buf)==0){
         cli[x].partner = i;
         break;
      }
   }
   write(x, "go", 2);
   printf("serv => cli %s : go\n", cli[x].name);
   state[x] = 5;
}
```

handle_state_4에서는 입력받은 값을 출력하고, 반복문을 통해 입력받은 이름이 클라이언트 목록의 이름에 있는지 확인하고, 몇 번째 인덱스에 같은 이름이 있는지 찾아 그 값을 cli[x].partner에 저장한다. 이는 클라이언트가 채팅을 하고싶은 사람과 연결이 된 것을 알 수 있다. 이후 go를 보내주기 위해 write(x)를 사용한다. 출력문을 출력하고 state[x]를 5로 변경한다.

```
void handle_state_5(int x, fd_set * pset, char * buf, int state[]){
   char sbuf[100];
   int p;
   p = cli[x].partner;
   printf("cli %s => serv : %s\n", cli[x].name, buf);
   sprintf(sbuf, "%s to %s : %s", cli[x].name, cli[p].name, buf);
   printf("serv => cli %s : \"%s\"\n", cli[p].name, sbuf);
   write(p, sbuf, strlen(sbuf));
}
```

handle_state_5에서는 본격적인 채팅이 시작된다. 반복문을 계속 돌아도 state[x]의 변화가 없으므로 해당 함수가 계속 실행된다. 입력받은 값을 출력하고, 누가 누구에게 어떤 채팅을 보내는지 출력한다. 이후 이 값을 write(p)를 통해 보내고 싶은 대상에게 이 내용을 전달한다.

```
[12180626@linuxer1 ~]$ servping7
Hi, I am the server
socket opened successfully. socket num is 3
binding passed
new cli at socket 4
new cli at socket 5
cli at socket 4 => serv : hello
serv => cli at socket 4 : name?
cli at socket 4 => serv aaa
serv => cli aaa : ready?
cli aaa => serv : yes
cli at socket 5 => serv : hello
serv => cli at socket 5 : name?
cli at socket 5 => serv bbb
serv => cli bbb : ready?
cli bbb => serv : yes
cli bbb => serv : aaa
serv => cli bbb : go
cli aaa => serv : bbb
serv => cli aaa : go
cli aaa => serv : Hi
serv => cli bbb : "aaa to bbb : Hi"
cli bbb => serv : Hello
serv => cli aaa : "bbb to aaa : Hello"
cli aaa => serv : my name is aaa
serv => cli bbb : "aaa to bbb : my name is aaa"
cli bbb => serv : my name is bbb
serv => cli aaa : "bbb to aaa : my name is bbb"
```

```
[12180626@linuxer1 ~]$ cliping7
Hi, I am the client
socket opened successfully. socket num is 3
enter hello
hello
name?
aaa
ready?
yes
client list ()
bbb
go
Hi

bbb to aaa : Hello
my name is aaa

bbb to aaa : my name is bbb
```

```
[12180626@linuxer1 ~]$ cliping7
Hi, I am the client
socket opened successfully. socket num is 3
enter hello
hello
name?
bbb
ready?
yes
client list (aaa )
aaa
go
aaa to bbb : Hi
Hello

aaa to bbb : my name is aaa
my name is bbb
```

컴파일 후 출력해본 결과 코드를 구성한대로 실행됨을 알 수 있었다.