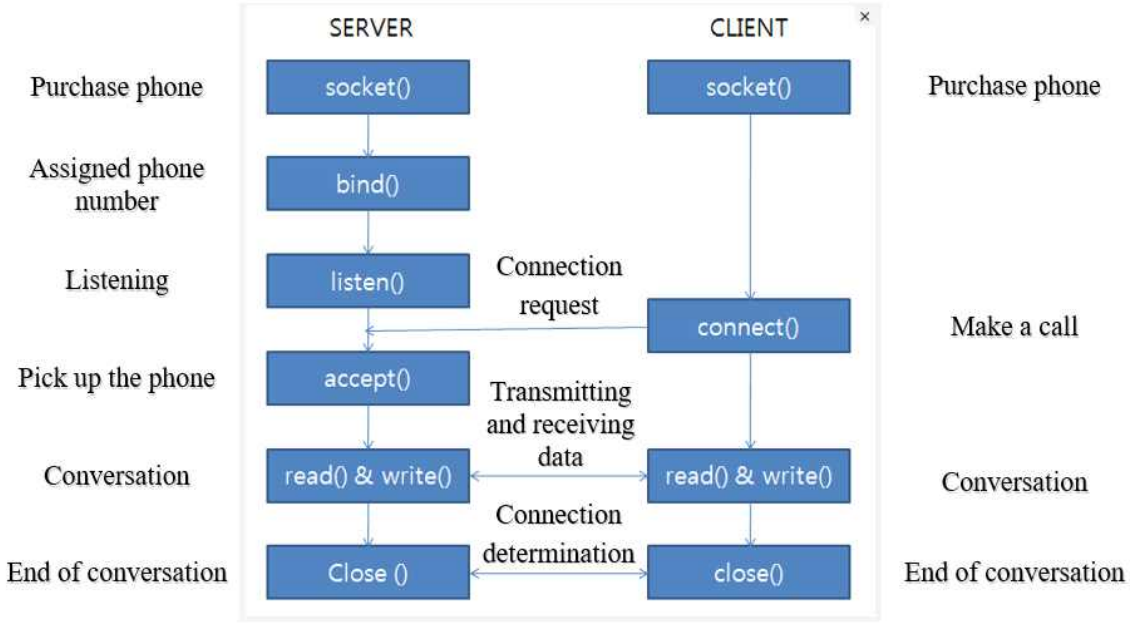
Operating System Project

김 태 연

1. Server.c



- 요약 -

소켓 프로그래밍에서 서버는 socket() -> bind() -> listen() 까지 외부 통신에 반응할 수 있도록 초기세팅이 된 후, 각 client가 connection 요청을 보내면 accpet()를 통해 소켓번호를 할당한다. 이때, 각 client에 id를 할당하여 각 커넥션별로 thread를 개별적으로 생성하고, pthread_detach()를 통해 부모 thread로부터 분리한다. 각 thread에서는 어느 client가 메시지를 보냈는지를 read()를 통해 파악하고, 전달받은 메시지를 전체 client에게 write()를 통해 broadcasting한다. 마지막으로 connection이 끊기면 close를 이용해 해당 소켓의 연결을 끊으며, 종료된 thread는 외부함수에서 명시적 호출없이도 스스로 할당된 자원을 반납한다.

1. header 추가부분

```
7   #include "stdio.h"
8   #include "stdlib.h"
9   #include "string.h"
10  #include "sys/types.h"
11  #include "sys/socket.h"
12  #include "netinet/in.h"
13  #include "arpa/inet.h"
14  #include <pthread.h>
15  #define BUFFSIZE 100
```

1) stdio.h,stdlib.h,string.h : 기본적인 c language이용 프로그램 바디제작에 사용
2) types.h, socket.h, in.h, inet.h : 소켓프로그래밍부 사용
3) pthread.h : 쓰레드이용에 사용
4) BUFFSIZE 100 : 단일 메시지 최대크기 100byte 제한

2. 전역변수 선언

```
18  int client_num = 0;
19  int client_id[5];
20  int portnum = 0;
21  pthread_mutex_t mutx;
```

5) client_num : 현재 접속된 클라이언트 개수 파악을 위해 사용
6) client_id[5] : 클라이언트 아이디부여를 위해 사용 / 최대 클라이언트수 5개 제한
7) portnum : 초기 포트값을 입력받기 위해 사용.
8) mutx : 클라이언트 아이디관리 및 write,read 함수부 critical section 관리를 위해 mutex사용

3. main함수 내 변수

```
61      struct sockaddr_in server_addr, client_addr;
62      int server_fd, client_fd;
63      int client_addr_size = 0;
64      int issuccess = 0;
65      pthread_t thread;
```

9) sockaddr_in server_addr, client_addr : binding과 accept부에서 서버설정, 통신방식 설정을 위해 사용
10) server_fd, client_fd : 서버 열기와 클라이언트 커넥션 설정을 위한 각 소켓번호 할당을 위해 사용
11) client_addr_size : accept부에서 client_addr 사이즈를 저장하기 위해 임시로 사용하는 변수
12) issuccess : 서버 open시 각 단계에서 오류발생확인용 return값 저장을 위해 사용
13) thread : 쓰레드 생성을 위해 사용

4. 포트 입력 부

```
67      printf("Portnum : ");
68      scanf("%d",&portnum);
```

14) 최초 서버 실행 시, 포트를 입력받음.

포트입력요청

```
root@notion-virtual-machine: /mnt/hgfs/VM_Ubuntu/socket_chat
root@notion-virtual-machine:/mnt/hgfs/VM_Ubuntu/socket_chat# ./server
Portnum :
```

포트입력

```
root@notion-virtual-machine: /mnt/hgfs/VM_Ubuntu/socket_chat
root@notion-virtual-machine:/mnt/hgfs/VM_Ubuntu/socket_chat# ./server
Portnum : 1233
```

5. mutex init

```
75      if(pthread_mutex_init(&mutx, NULL))
76          error("Mutex Init Error!");
```

15) mutual exclusive 사용을 위한 initializing

6. socket initializing

```
78      server_fd = socket(AF_INET, SOCK_STREAM, 0);
79      if(server_fd == -1){
80          printf("socket() error!\n");
81          exit(1);
82      }
```

16) AF_INET : IPv4 인터넷 프로토콜 사용

17) SOCK_STREAM : TCP/IP 프로토콜 사용

18) -1 반환 시 실패. 실패시 “socket() error!”메시지 출력

19) 위 작업을 통해 socket 하나를 생성.

7. binding

```
84      issuccess = bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr));
85      if(issuccess == -1) {                                   //bind 호출
86          printf("bind() error!\n");
87          exit(1);
88      }
```

20) 생성한 소켓을 server socket으로 커널에 등록하는 과정.

21) -1 반환 시 실패. 실패시 “socket() error!”메시지 출력

```
70      memset(&server_addr, 0x00, sizeof(server_addr));
71      server_addr.sin_family = AF_INET;
72      server_addr.sin_port = htons(portnum);
73      server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

22) 이때, 사전에 server_addr structure에 IPv4 사용선언(AF_INET), 포트넘버, IP를 할당.

23) 이때, INADDR_ANY를 사용하여 고정 IP를 사용하지 않고, 현재컴퓨터의 IP를 사용하나 localhost 내부에선 127.0.0.1로 통신이 가능하므로 크게 상관없음.

8. listen

24) 클라이언트 접속 요청을 확인.

25) -1 반환 시 실패. 실패시 “wait state fail!”메시지 출력

```
90      issuccess = listen(server_fd, 5);
91      if(issuccess == -1){                          //소켓을 수동 대기모드로 설정
92          printf("Wait state fail!\n");
93          exit(1);
94      }
```

9. client 관리부

```
100     while(1){
101         client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &client_addr_size);   //accept()시작
102         pthread_mutex_lock(&mutx);                          //현재 커넥트된 각 클라이언트의 갯수를 파악하기 위해서 사용
103         client_id[client_num++] = client_fd;                //클라이언트 넘버 저장
104         pthread_mutex_unlock(&mutx);
105         pthread_create(&thread, NULL, client_connection, (void*)client_fd);     //각 client 소켓별로 thread생성
106         printf("Connected to Client : %d\n", client_fd);
107         pthread_detach(thread);                             //parents에서 thread 분리. 종료시 pthread_join없이도 할당된 자원 반환.
108     }
109     close(server_fd);
110     return 0;
```

26) 클라이언트 접속요청시 accept()로 접속 허락. 접속 허락시 커널이 자동으로 소켓 생성하여 반환하므로 client_fd로 해당 소켓주소를 임시저장.
27) 새로 생선된 클라이언트를 관리하기 위해 클라이언트 소켓주소를 클라이언트 아이디로 할당. 이때, 클라이언트 아이디 할당부분은 추후 클라이언트 접속해제 시 동시에 사용될 수 있는 부분이기에 mutex lock으로 보호.
28) 새로 생성된 각각의 클라이언트와 통신을 위한 write, read 함수 사용을 개별적으로 하기위한 thread 생성.
29) 개별 쓰레드까지 생성되면 Connected to Client : [클라이언트넘버] 메시지 출력
30) 새로 생성된 클라이언트의 쓰레드를 parent 쓰레드로부터 분리시켜 명시적으로 pthread_join을 하지 않고도 개별 쓰레드 종료 시 독립적으로 리소스 반환을 할 수 있도록 처리.

10. client 별 thread 실행 작업

```
23  void * client_connection(void *arg){
24      int client_fd = (int)arg;
25      int str_len = 0;
26      char message[BUFFSIZE];
27      int i;
28
29      printf("New thread id is %lu\n",pthread_self());
30      while((str_len = read(client_fd, message, sizeof(message))) != 0){
31          printf("Client %d : %s\n", client_fd, message);
32          send_message(message, client_fd);
33      }
34      pthread_mutex_lock(&mutx);
35      for(i=0; i<client_num; i++){                    //클라이언트 접속종료시 현재 등록된 클라이언트수 정리
36          if(client_fd == client_id[i]){
37              for( ; i<client_num-1; i++)
38                  client_id[i] = client_id[i+1];
39              break;
40          }
41      }
42      client_num--;
43      pthread_mutex_unlock(&mutx);
44      printf("Dead thread id is %lu\n",pthread_self());
45      printf("Disconnected to Client : %d\n", client_fd);
46      close(client_fd);
47  }
```

31) 새로운 클라이언트 쓰레드의 아이디 출력.
32) while문 내에서 클라이언트로부터 입력이 왔는지 확인후, 입력이 왔으면 message 버퍼에 임시저장후, 해당 메시지를 send_message 함수를 통해 전체 클라이언트에게 broadcasting
33) 클라이언트로부터 접속이 끊기면 해당 클라이언트 id를 client_id[] 배열에서 제거후,

현재 클라이언트 수감소(client_num--) 이때, 클라이언트 숫자 관리부분은 클라이언트 추가부분과 겹칠수 있는 부분이기에 mutual exclusive 처리.

34) 제거할 클라이언트의 쓰레드ID를 pthread_self()함수를 이용해 출력후, close(client_fd)를 통해 연결 해제 및 함수 종료. 이때, pthread_detach()를 통해 부모 쓰레드로부터 독립된 개별쓰레드이므로 입력받은 리소스 자동반환.

11. send_message()

```
49  void send_message(char * message, int client_fd){                  //전체 client로 message를 broadcasting해주는 함수
50      int i;
51      char send_message[100] = "";
52      pthread_mutex_lock(&mutx);
53      sprintf(send_message, "Client %d : %s\n", client_fd, message);
54      for(i=0; i<client_num; i++)
55          write(client_id[i], send_message, sizeof(send_message));    //개별 client로 message 전송
56      pthread_mutex_unlock(&mutx);
57  }
```

35) 개별 클라이언트로부터 메시지를 받았을 경우 실행되는 함수로, 받은 클라이언트의 메시지를 모든 클라이언트들에게 write함수를 사용하여 전송시켜주는 함수. 이때, write동안 접속이 끊기면 안되므로 mutual exclusive 처리.

2. Client.c

1. header 추가부분

```
7   #include "stdio.h"
8   #include "stdlib.h"
9   #include "string.h"
10  #include "sys/types.h"
11  #include "sys/socket.h"
12  #include "netinet/in.h"
13  #include "arpa/inet.h"
14  #include <pthread.h>
15  #define BUFFSIZE 100
```

36) 본 내용은 Server.c 부분과 동일.
37) stdio.h,stdlib.h,string.h : 기본적인 c language이용 프로그램 바디제작에 사용
38) types.h, socket.h, in.h, inet.h : 소켓프로그래밍부 사용
39) pthread.h : 쓰레드이용에 사용
40) BUFFSIZE 100 : 단일 메시지 최대크기 100byte 제한

2. main함수 내 변수

```
31      char buffer[BUFFSIZE];
32      struct sockaddr_in server_addr;
33      int server_fd;                                  //server_fd : 각 소켓 번호
34      int client_addr_size = 0;
35      int issuccess = 0;
36      int str_len = 0;
37      pthread_t thread;
```

41) buffer : 서버로부터 받은 메시지를 임시 저장하기위한 변수
42) sockaddr_in server_addr : 서버와 connect를 위한 서버설정, 통신방식 설정을 위해 사용

3. 포트 입력 부

```
67      printf("Portnum : ");
68      scanf("%d",&portnum);
```

43) 최초 서버 실행 시, 포트를 입력받음.

포트입력

```
root@notion-virtual-machine: /mnt/hgfs/VM_Ubuntu/socket_chat
root@notion-virtual-machine:/mnt/hgfs/VM_Ubuntu/socket_chat# ./client
Portnum : 1233
```

4. 소켓 생성 부

```
42      server_fd = socket(PF_INET, SOCK_STREAM, 0);
43      if(server_fd == -1){                                    // 소켓 생성
44          printf("socket() error!\n");
45          exit(1);
46      }
```

44) PF_INET : IPv4 인터넷 프로토콜 사용

45) SOCK_STREAM : TCP/IP 프로토콜 사용

46) -1 반환 시 실패. 실패시 “socket() error!”메시지 출력

47) 위 작업을 통해 socket 하나를 생성.

5. 서버 연결 부

```
48      memset(&server_addr, 0x00, sizeof(server_addr));
49      server_addr.sin_family = AF_INET;                   //IPv4 설정
50      server_addr.sin_port = htons(portnum);              //포트설정
51      server_addr.sin_addr.s_addr = htonl(INADDR_ANY);    //자신의 ip주소로 설정. 어차피 localhost 127.0.0.1 사용해서 상관없음
52
53      connect(server_fd, (struct sockaddr*) &server_addr, sizeof(server_addr));       //서버 연결
54      printf("Open!! client\n");
```

48) Connect()를 이용, 주소정보에 서버의 주소와 포트번호를 지정하고 서버와의 연결을 시도.

6. writing thread 생성 및 read 부

```
56      pthread_create(&thread, NULL, writing, (void*)server_fd);
57      while(1) {
58          str_len = read(server_fd, buffer, BUFFSIZE);
59          if(str_len != -1){
60          printf("%s", buffer);
61          }
62      }
63      close(server_fd);
64      return 0;
```

49) 항상 read함수가 대기중이므로 writing을 위해 scanf에서 block됨을 방지하기 위해 별도의 writing thread를 생성.

50) 마찬가지로, 생성된 쓰레드를 pthread_detach()를 이용해 해당 쓰레드 종료시 자동으로 리소스를 반환하도록 설정.

51) while문내에서 지속적으로 서버로부터온 메시지를 수신하며 수신시 해당내용을 커맨드에 출력.

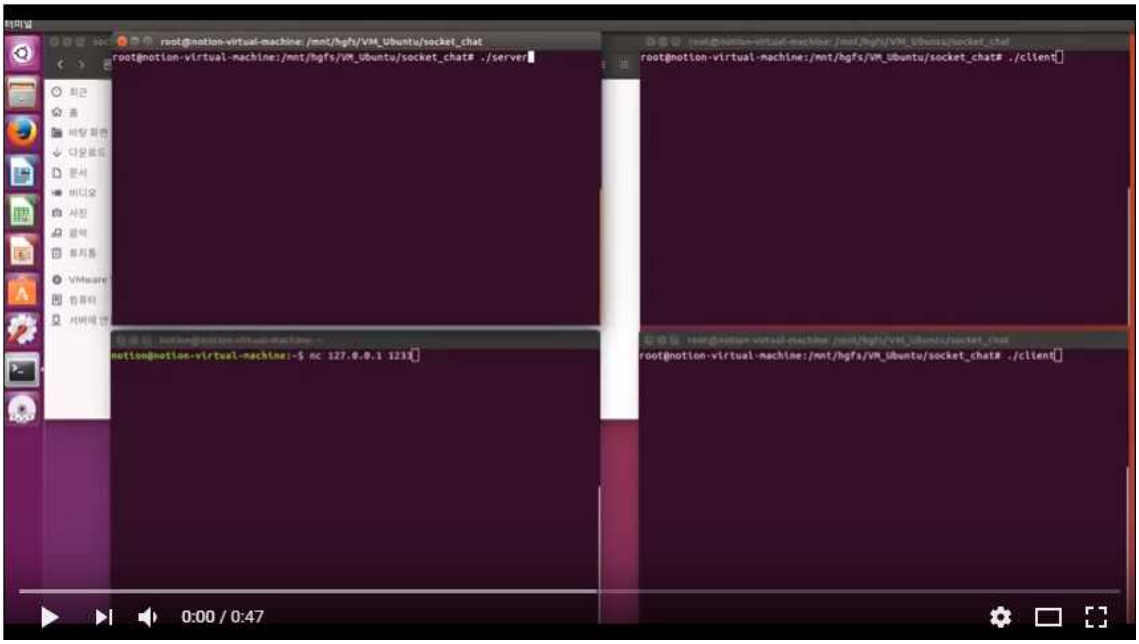
7. writing 부

```
20  void * writing(void *arg){                              //서버로 메시지를 전송하는 함수.
21      int server_fd = (int)arg;
22      char message[BUFFSIZE];
23      while(1){
24          scanf("%s", message);
25          write(server_fd, message, strlen(message));
26      }
27  }
```

52) read와 writing을 동시에 진행할 수 없으므로, 별도의 writing thread를 생성해서

scanf를 이용해 메시지를 입력받을 수 있도록 제작. 메시지 입력 후 엔터 입력 시 write()함수를 이용해 서버로 해당 메시지를 전송. 메시지는 서버에서 수신되어 서버가 전체 client로 broadcasting함.

7. 해당 소스코드가 돌아가는 과정은 하단의 youtube를 통해 확인하실 수 있습니다.



https://www.youtube.com/watch?v=npSVcH8RHoQ