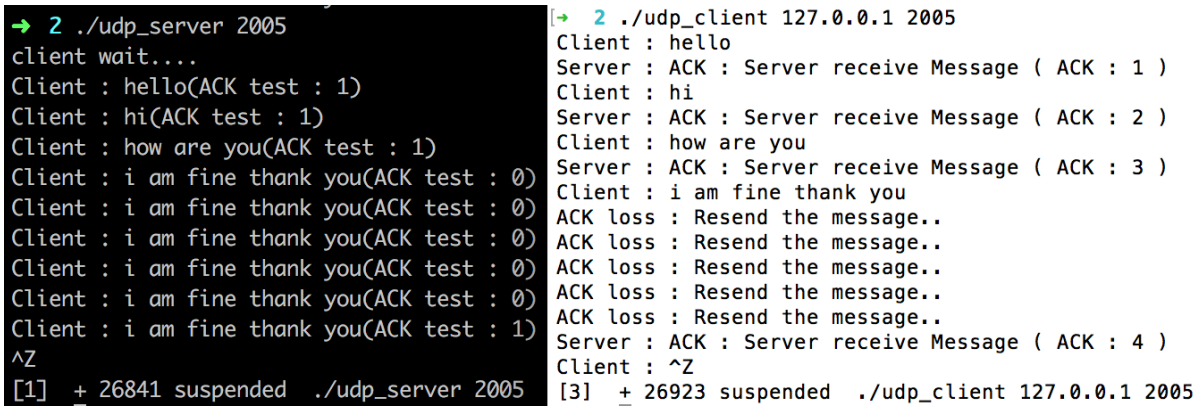
데이터 통신
Project2 : UDP Chatting Program

컴퓨터 전공
김다빈 2015004375

Project2 : UDP Chatting Program

1. 결과 화면



```
➜ 2 ./udp_server 2005
client wait....
Client : hello(ACK test : 1)
Client : hi(ACK test : 1)
Client : how are you(ACK test : 1)
Client : i am fine thank you(ACK test : 0)
Client : i am fine thank you(ACK test : 0)
Client : i am fine thank you(ACK test : 0)
Client : i am fine thank you(ACK test : 0)
Client : i am fine thank you(ACK test : 0)
Client : i am fine thank you(ACK test : 1)
^Z
[1]  + 26841 suspended  ./udp_server 2005
```

```
➜ 2 ./udp_client 127.0.0.1 2005
Client : hello
Server : ACK : Server receive Message ( ACK : 1 )
Client : hi
Server : ACK : Server receive Message ( ACK : 2 )
Client : how are you
Server : ACK : Server receive Message ( ACK : 3 )
Client : i am fine thank you
ACK loss : Resend the message..
ACK loss : Resend the message..
ACK loss : Resend the message..
ACK loss : Resend the message..
ACK loss : Resend the message..
Server : ACK : Server receive Message ( ACK : 4 )
Client : ^Z
[3]  + 26923 suspended  ./udp_client 127.0.0.1 2005
```

- server 실행시 “client wait…” 실행 : client가 들어올 때 까지 기다린다.
- client가 연결되면 client는 메세지를 서버에 보낸다.
- server는 메세지 확인 후 ACK 메세지를 client에 돌려 보낸다. 클라이언트는 받은 ack 메세지 수를 count 한다.
- ACK test : 메세지를 전달 받지 못했을 경우(메세지 프레임이 깨졌을 경우)를 실험하기 위하여 rand 함수를 통해 0 또는 1을 랜덤으로 받아 1이 나올 경우 ACK 메세지를 전달하고 0이 나올 경우 수신대기 상태로 넘어가게 된다.
- 클라이언트는 10초 이상 ACK 메세지를 받지 못하면 timeout 되어 ACK loss 를 클라이언트에 띄우고 직전에 보낸 메세지와 동일한 메세지를 다시 Server에 건내준다.
- +) 포트번호, ip 직접 입력 ex : ./echo_server 포트번호

2. 코드 설명

- Server Code

1. 서버를 먼저 실행하기 위하여 create socket부분에 소켓을 생성하고, 제대로 생성이 안됐을 때를 체크해준다.
2. server_address, client_address 등 기본 서버 세팅을 해준다.(//server 부분)
3. 생성 소켓에 로컬 주소를 할당 성공여부를 반환한다.(//bind)
4. 해당 소켓의 연결을 기다린다. 연결 성공시 0을 반환한다.(//listen)
5. client가 연결 요청이 들어 왔을 시, 연결을 받아 들인다.(//accept)
6. while 문 안에서는 thread_recv 함수를 요청해 클라이언트에게서 데이터를 받아오게 되는데, 내용을 받으면 ACK 메세지를 클라이언트에게 돌려주게 된다.
7. int random = rand()%2 : 0또는 1을 랜덤으로 받아 0이 나오면 수신 대기 상태가 되고 1이 나오면 ACK 메세지를 전송하여 타임아웃 테스트를 진행한다.

```
int random=rand()%2; // ACK 전달 및 타임아웃 테스트
printf("(ACK test : %d)\n",random); // random==1 ack 전송, random==0 재수신 대기

if(random==1){
    if((status = pthread_create(&thr_id[1], NULL,&thread_sendACK, &accp_sock))!= 0){ // 송신 스레드
        printf("#3 Thread create error: %s\n", strerror(status));
        exit(0);
    }
}else continue;
```

8. void *thread_recv(void *arg) : 수신 스레드 함수 이다. recvfrom 함수를 이용하여 클라이언트에게서 데이터를 받아온다.

```
void *thread_recv(void *arg) { //수신 스레드 함수

    int accp_sock=(int) *((int*) arg);
    int addrlen;
    int status;

    addrlen=sizeof(client_addr);


    nbyte=recvfrom(accp_sock,buf,MAXLINE,0,(struct sockaddr*)&client_addr,&addrlen);
    if(nbyte<0){
        perror("Recv Fail");
        exit(0);
    }

    //pthread_mutex_lock(&mutex);

    //pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
    close(accp_sock);

}
```

9. void *thread_sendACK(void *arg) : ACK 메세지를 보내는 송신 함수이다.

```
void *thread_sendACK(void *arg) { //송신 스레드 함수

    int accp_sock=(int) *((int*) arg);
    int addrlen,len;
    char* ack="ACK : Server receive Message";

    addrlen=sizeof(client_addr);

    len=strlen(ack);

    nbyte=sendto(accp_sock,ack,strlen(ack),0,(struct sockaddr*)&client_addr,sizeof(client_addr));
    if(nbyte<0){
        perror("Send Fail");
        exit(0);
    }

    memset(buf,0,MAXLINE+1);

    pthread_exit(NULL);
    close(accp_sock);

}
```

- Client Code

1. 통신을 위한 소켓을 생성한다.
2. 생성 후 기본 서버 세팅을 해준다.
3. connect를 통해서 서버 프로그램과 연결해 준다.
4. select 함수 : 블로킹 함수. 소켓 디스크립터의 변화를 확인한다.
5. if (FD_ISSET(sock, &readfds)) : 만약 송신 후 ACK 메세지를 서버에서 보내면 그것을 받아 출력하고 ACK 메세지를 받은 횟수를 기록한다.
6. 그렇지 않으면 ACK 메세지의 손실이 일어난 것이므로 thread_resend 함수를 통해 메세지를 재전송한다.
7. ACK 메세지를 돌려받을 때 까지 반복한다.

```
while(1){

    //인자로 지정한 스레드 id가 종료하기를 기다립니다.

    if (select(sock+1, &readfds, NULL, NULL, &tv) < 0)
    {
        perror("on select");
        exit(1);
    }

    if (FD_ISSET(sock, &readfds)){

        if((status = pthread_create(&thr_id[1], NULL,&thread_recv, &sock))!= 0) {// 수신 스레드
            printf("#2 Thread create error: %s\n", strerror(status));
            exit(0);
        }

        pthread_join(thr_id[1], NULL);
        printf("Server : %s ( ACK : %d )\n",buf,++ackmsg);

        memset(buf,0,BUFSIZE+1);
    }
    else{
        printf("ACK loss : Resend the message..\n");
        if((status = pthread_create(&thr_id[2], NULL,&thread_resend, &sock))!= 0) {// 송신 스레드
            printf("#3 Thread create error: %s\n", strerror(status));
            exit(0);
        }

        FD_ZERO(&masterfds);
        FD_SET(sock, &masterfds);

        memcpy(&readfds, &masterfds, sizeof(fd_set));
        continue;
    }

    if((status = pthread_create(&thr_id[0], NULL,&thread_send, &sock))!= 0) {// 송신 스레드
        printf("#1 Thread create error: %s\n", strerror(status));
        exit(0);
    }

    pthread_join(thr_id[0], NULL);

}
```

8. void *thread_resend(void *arg) : 만약 ACK 메세지를 받지 못하면, 기존에 입력했던 데이터를 buf에 저장했다가 재전송하는 함수.

```
void *thread_resend(void *arg) {// 송신 스레드 함수

    int accp_sock=(int) *((int*) arg);
    int len;

    len=strlen(buf);

    nbyte=sendto(accp_sock,buf,strlen(buf),0,(struct sockaddr*)&server_addr,sizeof(server_addr));
    if(nbyte<0){
        perror("Send Fail");
        exit(0);
    }


    pthread_exit(NULL);
    close(accp_sock);

}
```