정보보호
Homework 06

201204025
김대래

과목 명 : 정보보호
담당 교수 : 류재철
분반 : 00 분반

Index

1. 실습 환경

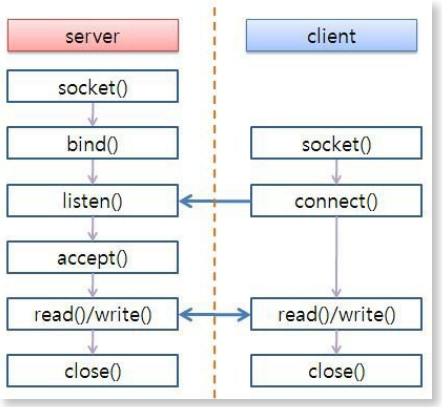
OS : Linux Ubuntu (Virtual Box)
Language : C
Tool : vim, gcc

2. 주요 개념

I. 서버-클라이언트 통신 프로그램

A. 소켓 통신

본 과제에서 주로 다루는 TCP/IP 소켓 통신 프로그램은 서버와 클라이언트 모두 포트를 통해 연결 및 데이터를 주고받는 방식으로 이 때 소켓 함수를 통해 상호작용을 할 수 있다.



• 서버 프로세스의 소켓이 클라이언트의 요청을 기다리는데 이를 수행하기 위해 서버는 먼저 클라이언트가 서버를 찾는데 사용할 수 있는 주소를 설정(바인드)한다.
• 주소가 설정되면 서버는 클라이언트가 서비스를 요청하기를 기다린다.
• 클라이언트 대 서버 데이터 교환은 클라이언트가 소켓을 통해 서버에 연결하면 이뤄지며 서버는 클라이언트의 요청을 수행한 후 클라이언트에 응답을 송신한다.

B. TCP / IP

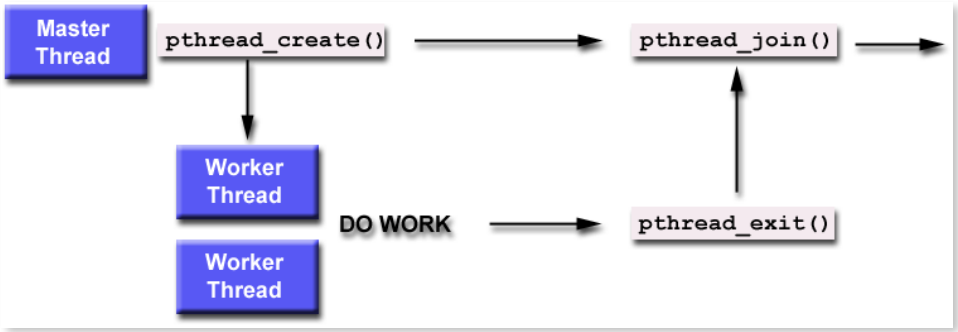
• TCP

포트를 사용하여 통신을 하는 방법에는 TCP 와 UDP 프로토콜 두가지가 있는데 TCP는 두 프로그램 간의 통신이 처음 시작될 때부터 끝날 때까지 계속 연결을 유지하는 연결지향(Connection oriented) 방식으로 전화와 비유할 수 있다. (Stream)

• IP

IP 주소는 1바이트 4자리의 주소로 이루어진다. (예 : 202.131.293.70)
하나의 IP 주소에서는 여러가지 작업을 동시에 할 수 있어야 하는데, 이 때 포트를 사용하여 여러 서비스를 사용한다.
0부터 1023 사이의 포트는 시스템에 예약되어 있으며, 1024부터 65535까지의 포트는 사용자가 임의를 사용할 수 있다.

C. Pthread (POSIX thread)



- 스레드
  스레드는 어떠한 프로그램 내에서, 특히 프로세스 내에서 실행되는 흐름의 단위를 말하며 여러개의 클라이언트를 처리하는 서버/클라이언트 모델의 서버프로그래밍 작업을 위해서 주로 사용된다.
- Pthread
  흔히 Pthread 라고 불리우는 POSIX thread는 POSIX 에서 표준으로 제안한 thread 함수모음으로 thread 를 지원하기위한 C 표준 라이브러리 셋을 제공한다.
  위 그림에서와 같이 pthread_create() 함수를 통해 worker 스레드를 생성하고 각각의 작업들을 수행한 뒤 pthread_exit() 함수로 worker를 종료하고 pthread_join() 함수로 join한다.
  (조인 시 다른 스레드가 종료될 때까지 기다리며 모든 스레드가 종료되면 프로세스 종료)
- mutex
  mutex 는 여러개의 쓰레드가 공유하는 데이타를 보호하기 위해서 사용되는 도구로써, 보호하고자 하는 데이타를 다루는 코드영역을 단지 **'한번에 하나의 쓰레드만'** 실행가능하도록 하는 방법으로 공유되는 데이타를 보호한다.
  이러한 코드영역(하나의 쓰레드만 점유가능한)을 critical section 이라고 한다.

3. 과제 6-1 코드 분석

I. chat.h 헤더 파일

```
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <pthread.h>
#include <errno.h>
#include <signal.h>

#include <openssl/des.h>

#define NAME_SIZE    64
#define MSG_SIZE     256

pthread_mutex_t mutex;

typedef struct {
    char name[NAME_SIZE];
    char msg[MSG_SIZE];
} TALK;

void handler(void)
{
    fprintf(stdout, "\n\033[F\033[J");
    fprintf(stdout, "-------------------------------------\n");
    fprintf(stdout, ">> EXIT \n");
    fprintf(stdout, "-------------------------------------\n");
    exit(1);
}
```

- 서버-클라이언트 통신 프로그램 구현을 위하여 c 라이브러리를 사용 (헤더 참조)

- 채팅 프로그램 구현을 위하여 작성자와 내용을 담기 위한 구조체 TALK 정의

- handler 함수는 프로그램 종료 (^C)를 누를 경우 종료 메시지를 표출하기 위한 함수

II. 통신 프로그램 - 서버

```
int main(int argc, char *argv[])
{
    int                 serv_sock;
    struct sockaddr_in  serv_addr;

    int                 clnt_sock;
    struct sockaddr_in  clnt_addr;
    socklen_t           clnt_addr_size;

    pthread_t           thread;

    // SIG HANDLER
    signal(SIGINT, (void *)handler);

    if ( argc != 2 ) {
        fprintf(stderr, "Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    if ( pthread_mutex_init(&mutex, NULL) ) {
        fprintf(stderr, "[!] ERROR : Mutext Init\n");
    }

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);
    if ( -1 == serv_sock ) {
        fprintf(stderr, "[!] ERROR : Socket()\n");
    }
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family        = AF_INET;
    serv_addr.sin_addr.s_addr   = htonl(INADDR_ANY);
    serv_addr.sin_port          = htons(atoi(argv[1]));

    if ( -1 == bind(serv_sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) ) {
        fprintf(stderr, "[!] ERROR : Bind()\n");
    }
    if ( -1 == listen(serv_sock, CLNT_MAX_NUM) ) {
        fprintf(stderr, "[!] ERROR : Listen()\n");
    }
```

- pthread mutex를 초기화
- 서버의 소켓 프로그래밍을 위하여 socket()함수를 사용한 반환 값을 serv_sock에 저장
- 프로그램 실행 시 argument를 port 생성
- bind()함수를 통해 서버 주소 설정
- listen() 함수로 클라이언트의 접속을 기다림

```
while(1) {
    clnt_addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);
    if ( -1 == clnt_sock ) {
        fprintf(stderr, "[!] ERROR : Accept()\n");
    }

    pthread_mutex_lock(&mutex);
    clnt_socks[clnt_num++] = clnt_sock;
    pthread_create(&thread, NULL, clnt_manage, (void *)clnt_sock);
    pthread_mutex_unlock(&mutex);
    fprintf(stdout, "[!] New User : %s\n", (char *)inet_ntoa(clnt_addr.sin_addr));
}
```

- 클라이언트가 접속을 요청하면 accept() 후 worker 스레드를 생성하여 각 클라이언트 별 작업을 할 수 있도록 함
- 새로운 유저가 정상적으로 받았으면 클라이언트의 ip 주소값을 출력

```
int clnt_num = 0;
int clnt_socks[CLNT_MAX_NUM];

void * clnt_manage(void * arg)
{
    int     clnt_sock   = (int)arg;
    int     str_len     = 1;
    int     i, j;

    TALK    msg;

    while ( 0 != (str_len = read(clnt_sock, (void *)&msg, sizeof(TALK))) ) {
        pthread_mutex_lock(&mutex);
        for ( i = 0 ; i < clnt_num ; i++ ) {
            if ( clnt_sock != clnt_socks[i] ) {
                write(clnt_socks[i], (void *)&msg, str_len);
            }
        }
        fprintf(stdout, "[%s] %s", msg.name, msg.msg);
        memset(msg.msg, 0x0, sizeof(msg.msg));
        pthread_mutex_unlock(&mutex);
    }

    // client exit
    sprintf(msg.msg, "--- Exit ---\n");
    str_len = strlen(msg.msg);
    fprintf(stdout, "[%s] %s", msg.name, msg.msg);

    pthread_mutex_lock(&mutex);
    for ( i = 0 ; i < clnt_num ; i++ ) {
        if ( clnt_sock == clnt_socks[i] ) {
            for ( j = i ; j < clnt_num - 1 ; j++ ) {
                clnt_socks[j] = clnt_socks[j+1];
            }
        }
        else {
            write(clnt_socks[i], (void *)&msg, NAME_SIZE + str_len);
        }
    }
    pthread_mutex_unlock(&mutex);
}
```

- 클라이언트가 접속을 성공하면 worker thread가 client manage 함수를 통해 관리
- read() 함수를 통해 접속한 클라이언트가 보내는 TALK을 확인하며 해당 메시지를 모든 클라이언트에게 write()하여 전송
- 서버에서도 출력을 통해 메시지를 확인

- 이때, TALK에 담긴 msg가 암호화 되어 있으면 서버측에서는 확인 불가능

III. 통신 프로그램 - 클라이언트

```
int main(int argc, char *argv[])
{
    int                 serv_sock;
    struct sockaddr_in  serv_addr;

    pthread_t           send_thread, recv_thread;
    void *              thread_result;

    key = (char *)malloc(sizeof(char) * KEY_SIZE);
    printf(">> Input Key : ");
    scanf("%s ", key);

    // SIG HANDLER
    signal(SIGINT, (void *)handler);

    if ( argc != 4 ) {
        fprintf(stderr, "Usage : %s <ip> <port> <name>\n", argv[0]);
        exit(1);
    }
    sprintf(msg.name, "%s", argv[3]);

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);
    if ( -1 == serv_sock ) {
        fprintf(stderr, "[!] ERROR : Socket()\n");
    }
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family        = AF_INET;
    serv_addr.sin_addr.s_addr   = inet_addr(argv[1]);
    serv_addr.sin_port          = htons(atoi(argv[2]));

    if ( -1 == connect(serv_sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) ) {
        fprintf(stderr, "[!] ERROR : Connect()\n");
    }

    pthread_create(&send_thread, NULL, send_msg, (void *)serv_sock);
    pthread_create(&recv_thread, NULL, recv_msg, (void *)serv_sock);

    pthread_join(send_thread, &thread_result);
    pthread_join(recv_thread, &thread_result);

    close(serv_sock);

    return 0;
}
```

- 클라이언트는 프로그램 실행 시 ip주소, port번호, user이름을 arguments로 입력
- 각각 주소값, 포트번호, TALK객체의 name에 저장
- socket()함수를 통해 소켓 프로그래밍을 할 수 있도록 하고
- 서버 측에 connect() 요청을 실시
- 정상 적으로 연결이 되었으면 pthread_create 함수를 통해 각각의 worker 스레드가 send_msg와 recv_msg를 실시할 수 있도록 함

```
void * send_msg(void * arg)
{
    int sock = (int)arg;

    while(1) {
        fgets(msg.msg, MSG_SIZE, stdin);
        printf("\033[F\033[J");
        fprintf(stdout, "[%s] %s", msg.name, msg.msg);
        des(msg.msg, "asdf", strlen(msg.msg), 1);
        write(sock, (void *)&msg, sizeof(TALK));
                memset(msg.msg, 0x0, MSG_SIZE);
    }
}

void * recv_msg(void * arg)
{
    int sock = (int)arg;
    int str_len;

    while(1) {
        TALK tmp;
        str_len = read(sock, (void *)&tmp, sizeof(TALK));
        if ( -1 == str_len ) {
            return (void *)1;
        }
        des(tmp.msg, "asdf", strlen(tmp.msg), 2);
        fprintf(stdout, "[%s] %s", tmp.name, tmp.msg);
    }
}
```

- send_msg와 recv_msg 함수를 통해 채팅을 실시
- send_msg는 입력값(stdin)을 받아 TALK 구조체(msg)의 내용(msg)에 저장
- 해당 입력 값을 출력하고 des 암호화 후 write()를 통해 서버로 송신

- recv_msg는 서버로 부터 read()한 내용을 TALK 구조체(tmp)에 저장
- 암호화 된 tmp.msg를 복호화 하여 tmp.msg에 다시 저장하여 해당 내용을 출력

4. 과제 6-2 DES를 이용한 암복호화

I. 과제 개요

주어진 채팅 프로그램의 송•수신 시 암•복호화하여 서버로부터 주고 받도록 한다.

두 개의 클라이언트가 각자의 메시지를 암호화하여 주고 받는 것이 가능해야하며 해당 내용은 서버에서는 암호화된 메시지, 각 클라이언트는 정상적인 메시지를 받을 수 있어야 한다.

II. 문제 해결 과정

A. 주어진 소켓 통신 프로그램을 분석
B. 클라이언트 측에서 송•수신을 하는 코드를 확인
C. write하기 전과 read한 후의 메시지를 암•복호화
D. key 값 입력받을 수 있도록 작성
E. key 값을 입력하게 되면 개행 문자가 포함되어 수신 시 이상해지는 현상 발견
F. key 값 입력시 개행 문자가 무시될 수 있도록 "%s "로 띄어쓰기 추가

III. 소스 코드

```
key = (char *)malloc(sizeof(char) * KEY_SIZE);
printf(">> Input Key : ");
scanf("%s ", key);
```

- 암•복호화 키 입력 받기
- 개행 문자 무시하도록 띄어쓰기 추가

```
void * send_msg(void * arg)
{
    int sock = (int)arg;

    while(1) {
        fgets(msg.msg, MSG_SIZE, stdin);
        printf("\033[F\033[J");
        fprintf(stdout, "[%s] %s", msg.name, msg.msg);
        des(msg.msg, key, strlen(msg.msg), 1);
        write(sock, (void *)&msg, sizeof(TALK));
                memset(msg.msg, 0x0, MSG_SIZE);
    }
}
```

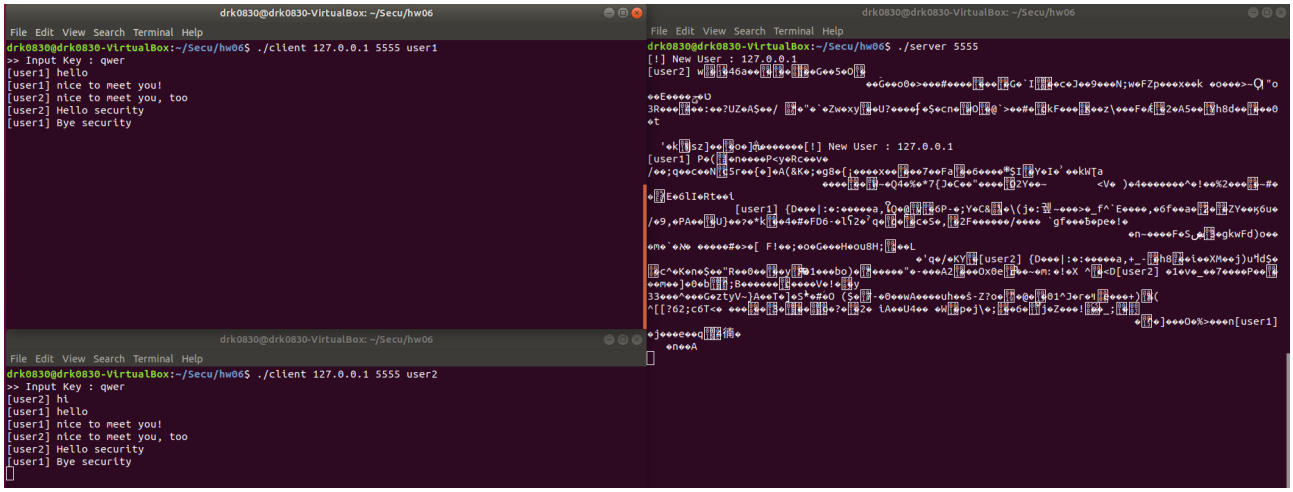
-입력 받은 msg.msg를 암호화 후 전송하기 위하여
- write()하기 전에 des암호화

```
void * recv_msg(void * arg)
{
    int sock = (int)arg;
    int str_len;

    while(1) {
        TALK tmp;
        str_len = read(sock, (void *)&tmp, sizeof(TALK));
        if ( -1 == str_len ) {
            return (void *)1;
        }
        des(tmp.msg, key, strlen(tmp.msg), 2);
        fprintf(stdout, "[%s] %s", tmp.name, tmp.msg);
    }
}
```

- read()를 통해 서버로 부터 수신한 메시지를 tmp에 저장
- 해당 메시지를 출력하기 전 복호화하여 정상적인 메시지 출력

5. 결과 화면


drk0830@drk0830-VirtualBox: ~/Secu/hw06
File Edit View Search Terminal Help
drk0830@drk0830-VirtualBox:~/Secu/hw06$ ./client 127.0.0.1 5555 user1
>> Input Key : qwer
[user1] hello
[user1] nice to meet you!
[user2] nice to meet you, too
[user2] Hello security
[user1] Bye security
drk0830@drk0830-VirtualBox: ~/Secu/hw06
File Edit View Search Terminal Help
drk0830@drk0830-VirtualBox:~/Secu/hw06$ ./client 127.0.0.1 5555 user2
>> Input Key : qwer
[user2] hi
[user1] hello
[user1] nice to meet you!
[user2] nice to meet you, too
[user2] Hello security
[user1] Bye security
drk0830@drk0830-VirtualBox: ~/Secu/hw06
File Edit View Search Terminal Help
drk0830@drk0830-VirtualBox:~/Secu/hw06$ ./server 5555
[!] New User : 127.0.0.1
[user2] New User : 127.0.0.1
[user1] New User : 127.0.0.1
[user1] hello
[user2] hi
[user1] nice to meet you!
[user2] nice to meet you, too
[user2] Hello security
[user1] Bye security