

정보보호

HW07 : DES Cipher Chatting

| | |
|-------|---------------|
| 제 출 일 | 2018년 10월 24일 |
| 담당교수 | 류재철 |
| 학 과 | 컴퓨터공학과 |
| 학 번 | 201302423 |
| 이 름 | 신종욱 |

1. 코드분석

■ Client Main함수

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

    pthread_t send_thread, recv_thread;
    void * thread_result;
    // SIG HANDLER
    signal(SIGINT, (void *)handler);
    if ( argc != 5 ) {
        fprintf(stderr, "Usage : %s <ip> <port> <name> <key>\n", argv[0]);
        exit(1);
    } //ip port 이름에 키 값까지 인자를 더받아야한다
    sprintf(msg.name, "%s", argv[3]);
    sprintf(key, "%s", argv[4]); //키값과 이름은 저장
    serv_sock = socket(PF_INET, SOCK_STREAM, 0); //소켓 생성
    if ( -1 == serv_sock ) {
        fprintf(stderr, "[!] ERROR : Socket()\n");
    } //생성이 잘됐는지 확인
    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]));
    //주소와 포트를 지정

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

    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);
    //쓰레드로 생성후 작동
    close(serv_sock);

    return 0;
}
```

일단 실행시 인자를 ip주소, port번호, 사용자 name, 암호화 **key** 값을 입력받아서 진행된다. 전부가 올바르게 들어왔다면 소켓연결에 성공하고 SEND와 RECV 쓰레드가 실행된 상태이다

■ 전역변수, Send, Recv

```
unsigned int des(unsigned char * msg,unsigned char *key, unsigned int msg_len, int mode);
TALK msg;
char key[KEY_SIZE];//키를 저장하기위해 전역변수로 선언

void * send_msg(void * arg)
{
    int sock = (int)arg;
    while(1) {
        fgets(msg.msg, MSG_SIZE, stdin);
        printf("\033[F\033[J");//라인 정리
        fprintf(stdout, "[%s] %s", msg.name, msg.msg);//메세지 깔끔히 출력
        des(msg.msg,key, strlen(msg.msg), 1);//Des 암호화
        write(sock, (void *)&msg, sizeof(TALK));//전송
        memset(msg.msg, 0x0, MSG_SIZE);//초기화
    }
}

void * recv_msg(void * arg)
{
    int sock = (int)arg;
    int str_len;
    int padding;
    while(1) {
        TALK tmp;
        str_len = read(sock, (void *)&tmp, sizeof(TALK));
        if ( -1 == str_len ) {
            return (void *)1;
        }//메세지를 읽는다
        padding=des(tmp.msg,key, BLOCK_SIZE, 2);//des복호화후 패딩값을 저장
        tmp.msg[padding]='\0';//마지막 위치에 0을 지정하여서 출력시 패딩부분 잘리도록 설정
        fprintf(stdout, "[%s] %s", tmp.name, tmp.msg);//출력
        memset(tmp.msg, 0x0, MSG_SIZE);//초기화
    }
}
```

Send를 할땐 입력을 한후 엔터를 칠 경우 라인이 정리되고 메시지를 보낸 자신의 이름과 함께 깔끔하게 출력된다.

하지만 서버(소켓)으로 보내기전에 des암호화를 진행한 후 소켓에 쓴다.

Recv를 할땐 들어온 파일을 읽고 복호화 작업을 해준다 이때 패딩값이 나올 수 있는데 이전과제에서 des의 함수는 padding이 제외된 즉 평문값의 길이를 리턴하도록 구현해 놔기 때문에 그 길이의 문자열인덱스에 null값을 넣어 출력이 깔끔하게 되도록 하였다.

■ DES함수 코드분석

```
unsigned int des(unsigned char * msg,unsigned char *key, unsigned int msg_len, int mode)
{
    DES_key_schedule des_ks;
    DES_cblock des_key = {0, };
    DES_cblock iv = {0, };
    unsigned int i,result, padding;

    unsigned char block_in[BLOCK_SIZE] = {0, };
    unsigned char block_out[BLOCK_SIZE] = {0, };

    DES_string_to_key(key,&des_key);
    DES_set_key_checked(&des_key,&des_ks);

    memcpy(block_in, msg, msg_len);

    if(mode==1){
        if(msg_len < BLOCK_SIZE){
            padding = BLOCK_SIZE - msg_len;
            int count=padding;
            while(count>=1){
                block_in[BLOCK_SIZE -count] = padding;
                count--;
            }
            //들어온 입력값이 블록사이즈보다 작다면 나머지 칸을 패딩으로 다 채워준다
            DES_ncbc_encrypt(block_in,block_out,BLOCK_SIZE,&des_ks,&iv,DES_ENCRYPT);
            result=BLOCK_SIZE;
        }//암호화 할땐 결과가 항상 블록사이즈
        else if(mode==2){
            DES_ncbc_encrypt(block_in,block_out,BLOCK_SIZE,&des_ks,&iv,DES_DECRYPT);
            padding = block_out[BLOCK_SIZE-1];
            int count=padding;
            while(count>=2){
                if( block_out[BLOCK_SIZE-count]!=block_out[BLOCK_SIZE-count+1]) break;
                count--;
            }//패딩값 만큼의 비트수가 패딩값으로 채워져 있는지 확인. 다르다면 break문으로 중단
            if(count==1)
                result = BLOCK_SIZE-padding;
            else result =BLOCK_SIZE;
        }//패딩이 있다면 count가 1까지 감소 했을 것임으로 result는 블록사이즈에서 패딩값을 빼준다.
        memcpy(msg,block_out,BLOCK_SIZE);

        return result;
    }
}
```

des.h를 참고하여 암호화할 때 들어가는 타입들이 다르기 때문에 iv벡터와 des_key를 알맞은 타입을 설정해주었다.

패딩은 부족한 만큼의 크기의 값을 부족한 인덱스에 모두 대입 해주었다.

■ Server main함수

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

if ( pthread_mutex_init(&mutex, NULL) ) {
    fprintf(stderr, "[!] ERROR : Mutext Init\n");
} //쓰레드의 뮤텍스 초기화 실패시 에러메세지

serv_sock = socket(PF_INET, SOCK_STREAM, 0);
if ( -1 == serv_sock ) {
    fprintf(stderr, "[!] ERROR : Socket()\n");
} //소켓단에서 연결이 실패할시 에러메세지
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");
} //바인드를 해준다.실패시 에러 메세지
if ( -1 == listen(serv_sock, CLNT_MAX_NUM) ) {
    fprintf(stderr, "[!] ERROR : Listen()\n");
} //서버 소켓이 클라이언트 요청을 기다리고 실패시 에러 메세지

while(1) {
    clnt_addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);
    //클라이언트의 접속을 허락해주는 함수로 성공했으면 소켓번호로 양수를 리턴해준다.
    if ( -1 == clnt_sock ) {
        fprintf(stderr, "[!] ERROR : Accept()\n");
    }

    pthread_mutex_lock(&mutex);
    clnt_socks[clnt_num++] = clnt_sock; //클라이언트가 늘어날때마다 추가
    pthread_create(&thread, NULL, clnt_manage, (void *)clnt_sock); //새로운 쓰레드 생성
    pthread_mutex_unlock(&mutex);
    fprintf(stdout, "[!] New User : %s\n", (char *)inet_ntoa(clnt_addr.sin_addr));
    //생성완료 메세지 출력
}

return 0;
```

서버부분은 정확하진 않지만 제가 아는 범위 까지 주석을 달아서 설명하였습니다.

■ Client manage

```
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); //쓰레드 사용시 뮤텍스를 이용하여서 상호배제가 일어나도록 구현
        for ( i = 0 ; i < clnt_num ; i++ ) {
            if ( clnt_sock != clnt_socks[i] ) {
                write(clnt_socks[i], (void *)&msg, str_len);
            } //클라이언트 소켓의 메시지를 읽고 저장
        }
        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];
            }
        } //클라이언트가 종료 됐을때 한개를 줄이기위한 과정
        else {
            write(clnt_socks[i], (void *)&msg, NAME_SIZE + str_len);
        }
    }
    pthread_mutex_unlock(&mutex);
}
```

■ 실행결과 화면

[illegible]

느낀점 : 소켓프로그래밍을 오랜만에 해서 함수들을 다시 확인 하느라 복잡하게되어 좋았다.