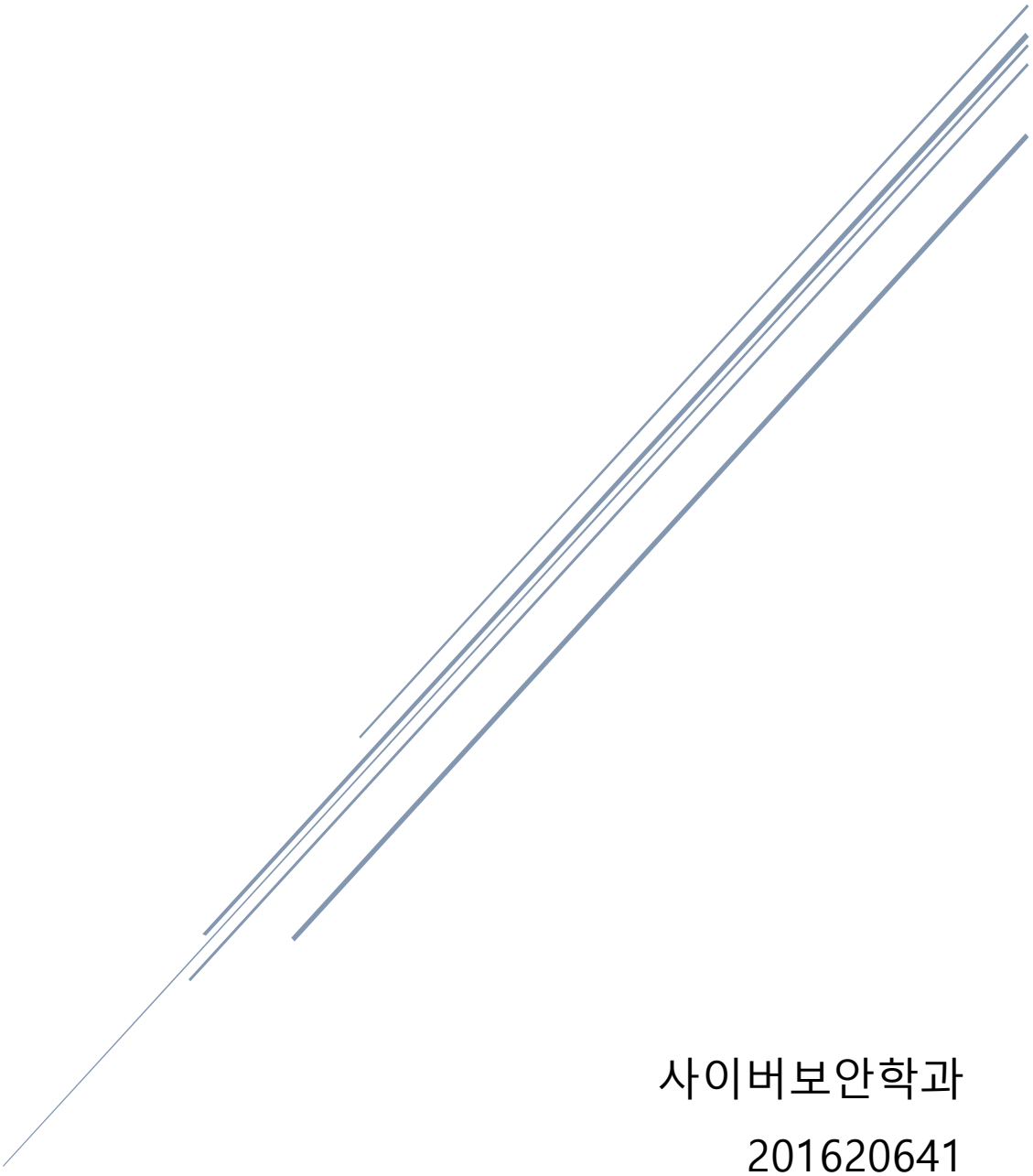
사원 관리 프로그램

Term Project



사이버보안학과

201620641

유 상 정

1. 서론

1.1. 목적

회사에서 직원들의 목록을 관리하는 것은 필수요소이다. 이를 위해서 각 회사마다 직원들의 정보를 관리하는 DataBase를 가지고 있다. 그리고 DataBase를 바탕으로 신입사원 채용이나, 직원들의 연봉 협상, 부서이동 등의 업무를 할 수 있다. 이 프로그램은 직원 관리자들에게 이러한 기능을 제공하고자, 만든 프로그램이다.

1.2. 요약

Server – Client Model의 사원 관리 DB프로그램을 만들었다. 이 프로그램은 Client가 Admin(관리자)가 되며, Server는 Client의 DB요청사항을 처리하는 역할이다. 또한 DB의 보안은 필수이므로 암/복호화 기능을 구현하였다. 하지만 일반 채널을 이용하면 문제가 있을 수 있으므로, key교환 channel을 bio socket을 이용하고, key 교환이 완료되면 일반 socket을 이용하여 Client가 사원 관리 DB프로그램을 사용한다. 참고로 Server program의 실행이 먼저 선행되어야 한다.

2. 본론

2.1. 기능

- Key 교환 과정

  a. Server가 1024bit의 RSA키쌍을 생성하고, bio Socket을 이용해 Client의 연결을 기다린다.

  b. Client와 연결이 완료되면, Client에게 bio socket을 통해 RSA 공개키를 배부한다.

  c. Client는 256bit의 AES 대칭키 (비밀키)를 생성한다.

  d. Client가 받은 RSA 공개키로 AES 키를 암호화하고, bio socket통해 Server에게 전송한다.

  e. Server는 bio socket을 통해 받은 암호화된 AES 비밀키를 복호화 한다.

  f. Key교환이 완료되었으므로 key교환 채널인 bio socket을 닫고, 통신 socket을 연결한다.

  g. 이후의 socket통신은 모두 AES-256 CBC모드로 암/복호화를 하여 통신한다.

- 사원 관리 DB프로그램 사용과정

➔ 키 교환 과정 이후, 로그인 전, 메뉴를 보여준다. 이 메뉴에서 회원가입, 로그인, 종료를 선택할 수 있다.

1. 로그인 전, 메뉴

1.1. 회원가입

a. 회원가입을 진행하기 위해서 admin_key를 입력해야 한다. 이때 admin_key는 Server의 define으로 정의되어 있으며 수정 가능하다. 현재는 1011이다. 또한 admin_key는 admin이 미리 알고 있다고 가정한다. 또한 admin_key의 입력 횟수는 5회로 제한하며, 5회를 모두 틀리면 Client에게 admin 자격이 없음을 알리고 프로그램을 종료 시킨 후, Server도 종료한다.

b. admin_key입력에 성공하면, Client는 회원가입을 진행한다. 이때, ID, PW, NAME을 입력한다. 그러면 Server가 ID의 중복여부를 판단하고, 중복이 아니라면 admin_data.csv에 저장한다.

c. admin_key 입력은 "*"로 나타난다.

1.2. 로그인

a. Client가 ID와 PW를 입력한다. 이때, PW 입력은 "*"로 나타난다.

b. login횟수의 제한은 5회이며, 횟수를 모두 사용하면 60초동안 프로그램 사용을 막는다. 그리고 60초 후, 이전 메뉴로 돌려보낸다.

c. Client가 Server에 로그인을 요청했을 때, 가입된 admin이 없으면 등록된 admin이 없다는 문장과 함께, 이전 메뉴로 돌려보낸다.

d. 등록된 admin이 존재할 때, Server가 admin_data.csv에서 요청 받은 ID와 PW를 admin_data.csv의 ID, PW들과 비교한다. 그리고 Client의 로그인이 성공이면 main menu로 보내고, 아니라면 login Try를 줄인다.

1.3. 종료

a. Client와 Server의 socket 연결을 끊고, Client와 Server program을 종료한다.

2. 로그인 후, 메뉴 (부서는 경영팀, 인사팀, 개발팀, 홍보팀만 존재)

2.1. 사원 등록

a. Admin이 등록할 사원의 부서, 이름, 사원 번호, 입사년도, 연봉을 입력한다.

b. Server가 employee_data.txt를 확인하여, 동일한 사원 번호가 있는지 검사한다.

c. 동일한 사원 번호가 있다면, 사원 등록을 거부한다. (사원 번호는 고유)

d. 동일한 사원 번호가 없으면, employee_data.txt와 부서명.txt에 사원 정보를 저장한다.

2.2. 부서 정보 조회

a. Admin이 특정 부서의 정보를 요청한다.

b. Server가 부서명.txt의 정보를 모두 읽어서 admin에게 전달한다.

2.3. 사원 정보 조회

a. Admin이 사원 이름, 사원 번호, 입사년도, 연봉 중에 하나를 선택하여 입력한다.

b. Server가 입력 받은 정보를 토대로 Employee_data.txt에서 관련된 모든 사원의 정보를 Admin에게 전달한다.

2.4. 사원 정보 삭제

a. Admin이 삭제할 사원의 사원 번호를 입력한다.

b. Server가 해당 사원 번호의 사원을 employee_data.txt에서 찾고, 제거한 후, 사원이 존재하는 부서명.txt에서도 제거한다.

2.5. 로그아웃

a. 로그인 세션을 종료하고, 로그인 전, 메뉴로 돌아간다.

2.6. 종료

a. Client와 Server의 연결을 끊고, 모두 종료한다.

2.2. 자가 진단표

	창의 단계 (단계에 체크√)	단계 기준	주요 창의적 내용
창의성	5 ☑	전혀 다른 창의적인 프로그램	Openssl을 이용한 암/복호화 통신 비대칭키를 통한 대칭키 교환 키교환에 이용된 BIO_socket
	4☐	예제 프로그램 또는 기존 프로그램을 참고로 새로운 프로그램 제작	

	3□	예제 프로그램 확장	키 교환 채널과 일반 통신 채널의 분리 System("clear")를 이용하여 ncurses처럼 구현 Linux에 없는 getch함수 구현 (비밀번호 "*" 표시) 파일을 이용한 DB구현	
	2□	예제 프로그램 일부 수정		
	1□	단순한 예제 프로그램 수준		
난이도 / 복잡도	복잡 단계 (단계에 체크)	단계 기준 (소스코드-코멘트 포함)	사용한 시스템호출과 라이브러리	고급 난이도 기술
	5 ✓	1200~ 라인 **(2062 라인)** **client.c: 960라인** **server.c: 1053라인** **common.h: 49라인**	**openssl관련 라이브러리** **RSA_generate_key_ex()** **PEM_write_RSAPublicKey()** **PEM_write_RSAPrivateKey()** **(rsa 키 생성 및 pem파일저장)** **BIO_write(),BIO_read()** **(PublicPEM파일을 1byte씩 전달)** **(암호화된 AES전달)** **Rand_bytes()** **(aeskey와 IV의 random생성)** **RSA_public_encrypt()** **RSA_private_decrypt()** **(RSA로 AES 암/복호화)** **BIO_do_accept()** **BIO_new_accept()** **(BIO socket관련) 더 있음** **EVP_CIPHER_CTX_new()** **EVP_EncryptInit_ex()** **-인자로** **EVP_aes_256_cbc()** **EVP_EncryptUpdate()** **EVP_EncryptFinal_ex()** **EVP_CIPHER_CTX_cleanup()**	Openssl을 이용한 암/복호화 Bio_socket을 통한 키교환 1024 RSA키쌍 생성 및 사용 AES 256 키 랜덤 생성 및 IV 랜덤 생성. AES-256 CBC모드 사용
	4□	800~1200라인		
	3□	500~800라인		
	2□	200~500라인		
	1□	~200라인		

			AES 암호화 복호화도 Encrypt를 Decrypt으로 바꾸면 됨 파일 고급 입출력 라이브러리 사용 fscanf, fwrite, fread, fopen, feof 등 <term.h>, <termios.h> 라이브러리를 사용 string관련 라이브러리 사용 strcpy, strcmp, strtok 등 socket관련 라이브러리 사용 memset 이외에 더 사용 함수를 전부쓰기에는 너무 많아서 이렇게만 써 놓겠습니다.	비밀번호를 “*”로 표시

	완성 단계 (단계에 체크)	내용	주요 bug 및 프로그램의 문제점
완성도	5☑	모든 기능의 안정적인 동작	RSA 키쌍이 가끔씩 생성이 안되는 문제를 RSA_generate_key()의 함수를 RSA_generate_key_ex() 함수로 변경함으로써 해결 암/복호화 과정 중, 인코딩, 디코딩 문제로 인해 암호문 전부 전달 못하는 문제 발생했었다 ->> 1024를 암호화하면 1040됨을 깨 달아서, BUFSIZE를 1040으로 지정하여 해결 입력 buffer문제 ->> __fpurge(stdin) 사용하여 해결 키 교환이 local에서는 되지만 GCP에서 안되는 문제
	4☐	주요 기능을 포함한 대부분 동작 (일부 버그 존재)	
	3☐	제한된 환경에서 일부 기능 동작	
	2☐	컴파일 완성 후 주요 동작 불가	
	1☐	프로그램 제작 후 컴파일 불가	

			->> RSA키를 pem파일(이진파일)로 저장하고, pem파일을 1byte씩 보냄으로써 해결. (이전에는 파일전달x) 이외의 여러 에러를 해결하여, 문제가 없음을 판단하였음. 위의 기능에서 설명한 것. 모두 이용 가능

2.3. 구현내용 및 방법

1. RSA 키 생성

```
void RSA_key_generator(RSA** rsa_object, int bits)
{
    BIGNUM * e = NULL;
    e = BN_new();
    if(BN_set_word(e, RSA_F4) != 1)
    {
        fprintf(stderr,"BN_set_word error!\n");
        exit(-1);
    }

    *rsa_object = RSA_new();
    if(RSA_generate_key_ex(*rsa_object, 2048,e ,NULL) !=1) // RSA 키 생성
    {
        BN_free(e);
        fprintf(stderr,"RSA key generate fail!\n");
        exit(-1);
    }
```

(1024bit의 RSA 키쌍 생성)

Server의 RSA_key_generator함수를 통해 1024bit의 RSA key쌍을 생성한다. 이때, e값은 65537로 지정한다.

```
//RSA 키쌍에서 공개키를 추출
rsa_pub = RSAPublicKey_dup(*rsa_object);
if(!rsa_pub)
{
    fprintf(stderr,"pub key generate error!");
    exit(-1);
}

//public key 저장을 위한 public.pem 파일 생성
FILE *pub_fp = fopen("public.pem", "wb");
if(!pub_fp)
{
    fprintf(stderr, "fopen error!\n");
    exit(-1);
}

//public.pem 파일에 공개키 저장
if(PEM_write_RSAPublicKey(pub_fp, rsa_pub) < 0)
{
    fprintf(stderr,"PEM_write_RSAPubliceKey error!");
    exit(-1);
}
fclose(pub_fp);
```

(공개키 추출 후, pem파일에 저장)

```
//개인키 저장을 위한 private.pem 파일 생성
FILE *fp = fopen("private.pem", "wb");
if(!fp)
{
    fprintf(stderr, "fopen error!\n");
    exit(-1);
}


//private.pem 파일에 개인키 저장
if(PEM_write_RSAPrivateKey(fp, *rsa_object, NULL, NULL, 0, NULL, NULL) < 0)
{
    fprintf(stderr,"PEM_write_RSAPrivateKey error!");
    exit(-1);
}
fclose(fp);
```

(비밀키를 pem파일에 저장)

키쌍을 생성한 후, rsa_PublicKey를 추출하여 public.pem파일에 저장하고, 개인키를

private.pem파일에 저장한다. 이때, rsa_object자체를 개인키로 사용한다.

RSA_key_generator에서 사용한 함수들은 <openssl/rsa.h>에서 제공한다.

2. AES 키와 IV생성

```
RAND_bytes(aes_key, 32);        //32byte의 AES key를 랜덤 생성 (AES-256 mode)
printf("\nAES symmetric key : ");
for(int i = 0;i < 32; i++)
{
    printf("%02X",(unsigned char)aes_key[i]);   //생성한 AES key 출력
}
printf("\n");
```

```
RAND_bytes(iv,16);
printf("AES iv : ");
for(int i=0;i<16;i++)
{
    printf("%02X",(unsigned char)iv[i]);   //생성한 AES iv 출력
}
printf("\n");
```

Client가 RAND_bytes 함수를 통해서 랜덤한 256bit의 AES key와 128bit의 IV를 생성한다.

3. BIO socket을 통한 키 교환

```
server_bio = BIO_new_accept(HOSTADDRESS); // 새로운 BIO 접근 만든다.
                                          //HOSTADDRESS : (argv[1]:argv[2]
                                          //argv[1] : serverIP
                                          //argv[2] : serverPORT
```

(bio socket 생성)

```
BIO_do_accept(server_bio);          //binding
printf("connection listening..\n");

if (BIO_do_accept(server_bio) <= 0) // 클라이언트의 연결 요청을 기다림 --> 키교환 채널
{                                   //listen
    ERR_print_errors_fp(stderr);
    exit(-1);
}
```

(bio socket bind & listening)

BIO Socket은 키 교환 채널이다.

server에서 BIO_socket을 생성하고, BIO_do_accept()을 두 번 호출하여, bind하고 listen한다. 즉, client의 연결 요청을 기다린다.

```
socket_bio = BIO_new_connect(HOSTADDRESS); //server와의 connection 생성(socket 생성)
```

(client의 BIO Socket 생성)

Client는 server와의 Bio socket을 생성한다.

그리고 BIO_do_connect()을 호출하여 server에게 connect을 요청한다.

오른쪽 사진이 그 부분이다.

```
printf("connecting..\n");
if(BIO_do_connect(socket_bio) <= 0)     //connection 요청
{
    //연결 실패 오류문 호출 후 client 종료
    fprintf(stderr, "connection error!\n");
    exit(-1);
}

printf("Connected!\n");
printf("read RSA public key from server...\n");
```

(client의 connect요청부분)

```
client_bio = BIO_pop(server_bio); // 접속이 완료되면 이 부분 통과(accept부분)
printf("Client Connected, send public key to client...\n");
```

(server의 accept부분)

Server는 Client의 요청을 BIO_pop()함수를 통해 수용하여, Server와 Client간의 키 교환 채널이 형성된다.

```
FILE * fp = fopen("public.pem", "rb");
if(!fp)
{
    fprintf(stderr,"fopen error!");
    exit(-1);
}
```

```
n = BIO_write(client_bio, &size,sizeof(size));
assert(n >=0);

unsigned char c;
//public.pem의 내용을 한 바이트씩 사이즈만큼 전송
for(int i=0;i<= seek;i++){
    fread(&c, sizeof(unsigned char), 1, fp);
    BIO_write(client_bio, &c, sizeof(c));
}
```

(server가 public.pem파일을 1byte씩 전송하는 부분)

키 교환 채널이 형성된 후, Server가 Public.pem파일을 rb모드로 열어서, Client에게 파일의 size를 알려주고, 1Byte씩 Client에게 전송한다.

```
//server로부터 RSA 공개키 파일 크기 수신
if((BIO_read(socket_bio, (unsigned char*)&size, sizeof(size))) <= 0)
{
    fprintf(stderr,"read error!\n");
    exit(-1);
}
```

```
//공개키 저장을 위한 public.pem 파일 생성
FILE *fp = fopen("public.pem", "wb");
for(int i=0; i<=size; i++)
{
    //1바이트씩 수신해서 public.pem 파일에 씀
    BIO_read(socket_bio, &c, sizeof(c));
    buf[i] = c;
    fwrite(&c, sizeof(unsigned char), 1, fp);

}
fclose(fp);
fp = fopen("public.pem", "rb");             //public.pem 파일 읽음
```

(Client의 Public.pem파일 수신 부분)

Client는 Server로부터 public.pem 파일의 크기를 받고, 그 크기만큼 pem파일을 1byte씩받아서, public.pem을 생성한다. 그리고 public.pem파일을 rb모드로 다시 연다.

```
rsa_pubkey = PEM_read_RSAPublicKey(fp,NULL,NULL,NULL); //pem으로 부터 공개키를 rsa 로 읽는다.
fclose(fp);
```

Public.pem 파일로부터 RSA Publickey를 가져와서 저장한다.

```
unsigned char encrypted_key[BUF_SIZE] = {0,};
n = RSA_public_encrypt(32, aes_key, encrypted_key, rsa_pubkey, RSA_PKCS1_OAEP_PADDING); // 전달받은 공개키로 생성된 AES Key를 암호화함
assert(n>=0);   //n<0 이면 프로그램 종료

n = BIO_write(socket_bio, encrypted_key, n); // AES Key를 암호화한 값 전송
assert(n>=0);

n = RSA_public_encrypt(16, iv, encrypted_iv, rsa_pubkey, RSA_PKCS1_OAEP_PADDING); // 전달받은 공개키로 생성된 AES iv를 암호화함
assert(n>=0);

n = BIO_write(socket_bio, encrypted_iv, n); // AES iv를 암호화한 값 전송
assert(n>=0);
```

(client가 AES key와 IV를 RSA암호화 하고 전송하는 부분)

Client는 받은 공개키로, 생성한 AES key와 IV를 암호화하여, server에게 전송한다.

```
n = BIO_read(client_bio, encrypted_key, BUF_SIZE); // 클라이언트로부터 공개키로 암호화한 AES Key를 수신함
RSA_private_decrypt(n, encrypted_key, aes_key, rsa_object, RSA_PKCS1_OAEP_PADDING); // 개인키로 암호화된 AES Key를 복호화함

n = BIO_read(client_bio, encrypted_iv, BUF_SIZE); // 클라이언트로부터 공개키로 암호화한 AES iv를 수신함
RSA_private_decrypt(n, encrypted_iv, iv, rsa_object, RSA_PKCS1_OAEP_PADDING); // 개인키로 암호화된 AES iv를 복호화함
```

(server가 암호화된 AES key와 iv 받고 RSA복호화 하는 부분)

```
RSA_free(rsa_object);   //rsa 는 안씀으로 free
BIO_free(client_bio);   //키교환 채널 닫음
BIO_free(server_bio);
```

```
BIO_free(socket_bio);
// 키 교환 완료, 통신 시작
```

(server와 client가 키 교환 채널을 닫는 부분)

RSA와 BIO socket을 통한 AES key 교환이 끝났으므로, 키 교환 채널을 닫는다.

위의 일련의 과정을 통해서 AES 키 교환이 이루어 진다.

4. 일반 통신 채널 구성

BIO socket을 닫고, 일반 socket을 생성한다. 이때, client는 server에게 먼저 socket을 생성할 시간을 제공한다.

```
sleep(2); // 서버가 먼저 소켓 생성할 시간 부여
```

```
//일반 통신 채널 다시 열기
if ((server_socketfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
{
    fprintf(stderr, "socket error!!\n");
    exit(-1);
}
```

```
//bind 함수
if (bind(server_socketfd, (struct sockaddr*) & server_addr, sizeof(server_addr)) == -1)
{
    fprintf(stderr, "bind error!\n");
    exit(-1);
}
```

```
//client의 요청 기다림
if (listen(server_socketfd, 2) == -1) {
    fprintf(stderr, "listen error!\n");
    exit(-1);
}
```

```
//accept함수
client_addr_size = sizeof(client_addr);
if ((socketfd = accept(server_socketfd, (struct sockaddr*) & client_addr, &client_addr_size)) < 0)
{
    fprintf(stderr, "accept error!\n");
    exit(-1);
}
printf("Client와의 일반 통신 채널열렸습니다.\n");
```

(Server의 socket 생성에서 accept까지의 부분)

Server가 socket을 생성 및 bind하고, listen함수와 accept함수를 통해서 client의 연결 요청을 기다리고, client와 연결한다.

```
//socket 생성
if((socketfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
{
    fprintf(stderr,"socket error!\n");
    exit(-1);
}
```

```
//connection 요청
if ((connect(socketfd, (struct sockaddr*)&server_addr, sizeof(server_addr))) < 0) {
    printf("bind: connection refused\n");
    exit(-1);
}
printf("일반 통신 채널 open 완료!\n");
```

(client의 socket생성 및 connection 부분)

위는 client가 socket을 생성하고, server에게 connection을 요청하는 부분으로, server와 client간의 일반 통신 채널을 형성한다.

5. 암/복호화 구현

```
void secure_write(int socketfd, char * buffer, int buf_size);  //write 및 암호화
char * secure_read(int socketfd,char * buffer, int buf_size);  //read 및 복호화
```

암/복호화를 구현하기 위해서, secure_write()와 secure_read() 함수를 정의하였다. 두 함수 모두, socketfd와 buffer(암호화할 data/복호화된 data 저장되는 곳), 그리고 buf_size를 매개변수로 받는다. 여기서 buf_size는 사실상 의미가 없다. 그저, read, write의 형식을 맞추고자 넣어 놓은 것이다. 실제로는 BUF_SIZE를 사용한다. (common.h에 define되어있음)

- Secure_write() 함수

```
int plain_length;      //평문 길이
int out_length = 0;        //CipherUpdate, CipherFinal 함수에서 out data 길이를 저장
int cipher_length;       //암호문 길이
char cipher[BUF_SIZE] = {0,};   //암호문
EVP_CIPHER_CTX *ctx;               //암호화 컨텍스트
                                   //-> 암호화 데이터의 정보 저장되는 구조체
```

필요한 변수들을 선언해 준다. 각 설명을 주석처리 해 두었다. 여기서 ctx는 암호화 context로서 Encryption, Decryption 등, 각 수행에 필요하거나 변경되는 정보들이 저장되는 구조체이다. 대칭키 암호에서 사용한다. 암/복화에서 EVP Cipher를 사용하는 이유는 여러 암호 알고리즘을 동일한 루틴으로 사용가능하기 때문이다.

```
plain_length = 1024;
```

평문 길이는 암호화될 data와 관계없이 1024로 고정한다. 왜냐하면, 어떤 data가 들어와도 암호화되는 data의 길이가 변하지 않게 하기 위해서이다. 1024bit를 암호화하면 1040bit가 된다. 이것은 인코딩, 디코딩에 의한 것이다.

```
//암호화를 위한 초기 설정
if((EVP_EncryptInit_ex(ctx,EVP_aes_256_cbc(), NULL, aes_key, iv)) != 1)
{
    fprintf(stderr, "EVP_EncryptInit error!\n");
    exit(-1);
}
```

(암호화를 위한 초기설정을 수행하는 함수)

EVP_EncryptInit_ex()함수를 통해서, AES-256 CBC모드로 암호모드를 설정하며, aes_key와 iv를 등록한다.

```
//암호화 수행
if((EVP_EncryptUpdate(ctx, cipher, &out_length, buffer, plain_length)) != 1)
{
    fprintf(stderr, "EVP_CipherUpdate error!\n");
    exit(-1);
}
cipher_length = out_length;
```

들어온 data의 암호화를 수행하고, 나온 길이를 cipher_length로 저장한다.

```
//패딩등 필요한 작업을 처리
if ((EVP_EncryptFinal_ex(ctx, cipher + out_length, &out_length)) != 1)
{
    fprintf(stderr,"EV_EnctyptFinal error!\n");
    exit(-1);
}
cipher_length += out_length;
cipher[cipher_length] = '\0';
```

마지막으로 패딩 등 필요한 작업을 처리해준다. 그리고 암호문 마지막에, '₩0' 값을 삽입한다.

```
    for(int i = cipher_length +1;i<BUF_SIZE;i++)
        cipher[i] = 0;

    EVP_CIPHER_CTX_cleanup(ctx);
    //암호화 완료
```

그리고 혹시나 모르는 경우를 대비하여, 암호화하고 처리되지 않은 뒷부분을 0으로 채우는 for문을 하나 처리했다. 하지만 큰 의미는 없을 것으로 예상한다.

최종적으로 EVP_CIPHER_CTX_cleanup(ctx) 함수를 호출하여, ctx구조체를 정리해준다. 이를 통해 AES-256/CBC모드로 암호화가 처리되었다.

```
if((write(socketfd,cipher,BUF_SIZE)) <= 0 )    //암호화된 data 전송
{
    fprintf(stderr,"write error!\n");
    exit(-1);
}
```

최종적으로 암호화된 data를 wirte하여 전송한다.

- secure_read()

```
int cipher_length = 0; // 수신한 암호문의 길이
int out_length = 0; // CipherUpdate, CipherFinal 함수에서 out 데이터 길이를 저장할 변수
int plain_length = 0; // 평문 길이
char plain[BUF_SIZE] = {0,}; // 복호화한 데이터
EVP_CIPHER_CTX *ctx; // EVP context 변수
```

secure_write()와 마찬가지로, 필요한 변수들을 선언해준다.

```
if(!(ctx = EVP_CIPHER_CTX_new()))   //ctx 초기화
{
    fprintf(stderr,"ctx new error!\n");
    exit(-1);
}
```

그리고 ctx 구조체를 초기화해준다.

```
//암호화된 data를 1byte씩 수신
for(int i =0;i<BUF_SIZE;i++)
{
    char temp;
    if((read(socketfd,&temp,1)) <= 0)
    {
        fprintf(stderr,"read error!\n");
        exit(-1);
    }
    buffer[i] = temp;
}
cipher_length = 1040;
```

암호화된 data를 1byte씩 수신해준다. 한 번에 받지 않고 1byte씩 받는 이유는 인코딩/디코딩 오류를 처리하는 과정에서 한 것인 것인데, 1byte씩 받는 것이, 좀 더 데이터의 유실없이 잘 전송되는 것 같아서, 이렇게 처리하였다.

```
//AES-256 cbc 모드로 복호화 수행
//복호화 전, 초기 설정
if ((EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, aes_key, iv)) != 1)
{
    fprintf(stderr,"EVP_DecryptInit error!\n");
    exit(-1);
}
```

복호화를 수행하기 전, 복호화 모드를 AES-256/CBC로 설정하고, AES key와 IV를 등록한다.

```
//복호화 수행
if ((EVP_DecryptUpdate(ctx, plain, &out_length, buffer, cipher_length)) != 1)
{
    fprintf(stderr,"EVP_DecryptUpdate error!\n");
    exit(-1);
}
plain_length = out_length;
```

실제로 복호화를 수행한다.

```
//패딩을 없애는 등, 필요한 작업 처리
if((EVP_DecryptFinal_ex(ctx, plain + out_length, &out_length)) !=1)
{
    fprintf(stderr,"EVP_DecryptFinal_ex error!\n");
    exit(-1);
}
plain_length += out_length;
plain[plain_length] = '\0';
```

마지막으로 복호화 수행 후, 필요한 작업을 처리해준다. EVP_DecryptFinal_ex()함수까지 수행되면, 복호화가 완료된다. EVP 함수들이 암/복호화가 똑 같은 루틴으로 수행됨을 알 수 있다.

```
EVP_CIPHER_CTX_cleanup(ctx);
// 복호화 수행 완료

for(int i = plain_length+1;i<BUF_SIZE;i++)
    plain[i] = 0;

strcpy(buffer,plain);

return buffer;
```

최종적으로 EVP_CIPHER_CTX_cleanup(ctx)를 통해 ctx 구조체를 정리하고, 혹시 모를 경우를 대비해, plain의 맨 뒷부분을 0으로 채우는 for문을 수행한 후, 평문을 return해준다.

6. 비밀번호입력의 "*" 출력

- password 함수

```
//pw를 * 로 표현하는 함수
char * password(char * pw)
{
    __fpurge(stdin);     //buffer 비우기
    int i = 0;
    while(i < BUF_SIZE){        //overflow 방지

        int temp;
        temp=getch();          //입력된 값 print되는 것 막는 함수
        if(temp == 10)break;  //enter 입력 시 break
        else if(temp == 127 )
        {
            if(i>0)
            {
                printf("\b");       //backspace를 print하여
                fputs(" ",stdout);  //시각적으로  *을 지운다.
                printf("\b");

                pw[i-1] = 0;        //실제로 입력된값을 지운다.
                i--;                //입력될 위치 조정
            }
            continue;
        }
        else{
            pw[i] = temp;
            i++;
            printf("*");
        }
    }
    pw[i] = '\0';       //문자열로 만들기 위해 null값 삽입
    printf("\n");
    return pw;
}
```

(pw입력시, "*"나타나게 하는 함수)

이 함수는 pw입력시, "*"이 나타나게 하는 함수이다. 실제로 이 함수에서 중요한 역할을 하는 함수는 getch()함수이다. 이 함수는 실제로 입력한 값이 화면에 나타나는 것을 막는 함수인데, window에서는 제공을 한다. 하지만 Linux에서는 제공하지 않아서, 직접구현 해

야 했다. 실제로 많이 당황한 부분이다.

- getch() 함수

```c
int getch(void)
{
  int ch;
  struct termios buff;
  struct termios save;

  tcgetattr(0, &save);            //원래의 터미널 설정 save
  buff = save;
  buff.c_lflag &= ~(ICANON|ECHO); //터미널 echo와 canonical 모드를 끔
  buff.c_cc[VMIN] = 1;            //입력 buffer를 1으로 만듬
  buff.c_cc[VTIME] = 0;           //출력 buffer를 0으로 만듬
  tcsetattr(0, TCSAFLUSH, &buff); //현재 터미널 모드를 바꿈
  ch = getchar();                 //키보드로 부터 한글자 입력을 받음
  tcsetattr(0, TCSAFLUSH, &save); //입력받은 후 터미널 설정을 원래대로 복원
  return ch;                      //입력된 값을 return
}
```

Linux gcc library에 화면에 출력하지 않고 입력 받는 getch() 함수가 존재하지 않아서 직접 구현하였다. 이 getch()함수는 터미널의 설정을 변경하여 터미널 echo와 canonical 모드를 끄고 입력 버퍼를 1로 만든 후, 키보드 입력을 읽고 터미널 설정을 원래대로 복원하는 방식으로 구현했다.

7. 로그인 전 메뉴

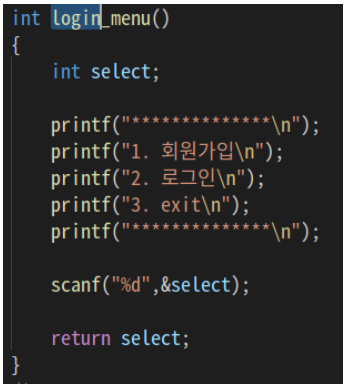
```c
void Client();          //로그인 전 Client section
int login_menu();       //로그인 전 메뉴
int sign_in();          //로그인
void sign_up();         //회원가입
```

(로그인 전, Client의 함수들)

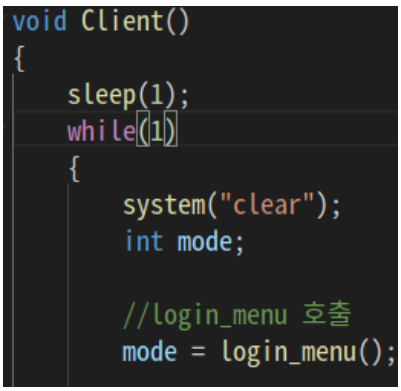
```c
void Server();                          //server routine 시작
int sign_in();                          //로그인 처리
int sign_up();                          //회원가입 처리
```

(로그인 전, Server의 함수들)

로그인 전에는 회원가입, 로그인, 종료 메뉴가 존재한다.



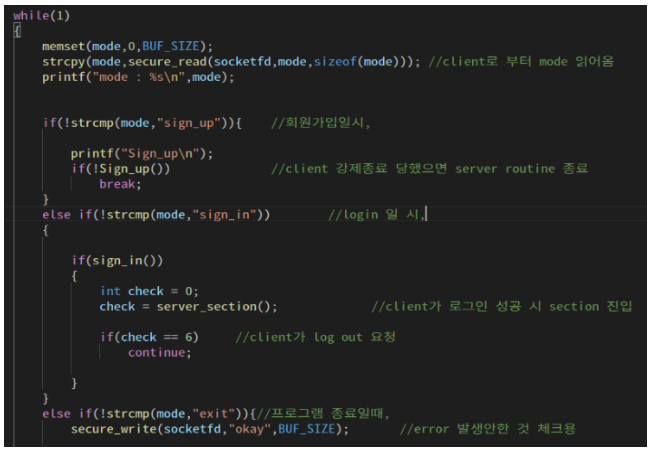
(로그인 전, 메뉴)



(Client 함수의 login_menu 처리)

Client는 로그인 전에 회원가입, 로그인, 종료 중 하나를 선택한다.

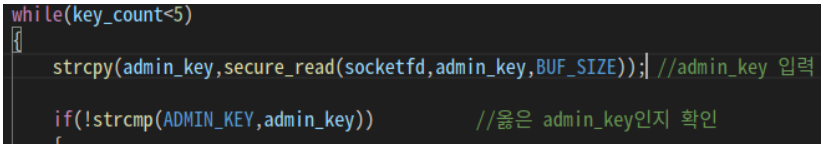
(Client의 mode에 대한 처리)



(Server의 mode에 대한 처리)

Client는 mode를 switch함수를 이용해서 진행할 함수를 결정하고, 각 메뉴에 따라 server에게 메시지를 전달한다. 그리고 server는 그 메시지에 따라서, 처리 로직을 결정한다. Client의 mode 처리 로직은 메뉴로서, 이 곳에 돌아올 때마다, system("clear")함수를 수행하여, 메뉴를 다시 띄워준다.

a. 회원가입



(server가 admin_key전달받는 부분)

Server는 client에게 admin_key를 전달받고, 옳은 admin인지 확인한다. 이때, 틀리면 key_count를 1 증가시키고, 다시 입력 받는다.

```
printf("admin key : ");      //틀릴 시, 회원가입 거부
strcpy(admin_key, password(admin_key));

secure_write(socketfd,admin_key,sizeof(admin_key)); //admin key 전달
strcpy(buf,secure_read(socketfd,buf,BUF_SIZE));     //server 로부터 메시지 전달 받
```

(client가 Server에게 admin_key 받는 부분)

이때, client는 server에게 전달받은 메시지에 따라서, 회원가입을 진행하거나 admin key를 다시 입력 받는다. 그리고 admin key 입력 시도 횟수를 모두 사용하면, Admin 자격이 없다는 메시지를 받고, 프로그램이 종료된다.

```
if(!strcmp(buf,"회원가입 요청이 허용되었습니다."))
{
    memset(buf, 0,BUF_SIZE);
    break;
}
else if(!strcmp(buf, "admin key가 틀렸습니다."))
{
    key_count++;
    printf("시도횟수 %d회 남았습니다.\n",5-key_count);
    memset(admin_key,0,BUF_SIZE);

    if(key_count == 5)
    {
        printf("모든 시도횟수를 사용하셨습니다.\n");
        printf("당신은 회사의 DB관리자가 아니라고 판단하여 

        memset(buf, 0,BUF_SIZE);

        sleep(2);

        BIO_free(socket_bio);
        system("clear");
        exit(1);
```

(client의 로직 처리부분)

```
printf("sign up\n");
printf("**************************\n");
printf("ID : ");
scanf("%s",admin_id);
secure_write(socketfd,admin_id,BUF_SIZE);

printf("P.W : ");
strcpy(admin_pw,password(admin_pw));
secure_write(socketfd,admin_pw,BUF_SIZE);

printf("admin_name : ");
scanf("%s",admin_name);
secure_write(socketfd,admin_name,BUF_SIZE);
printf("**************************\n");
strcpy(buf,secure_read(socketfd,buf,BUF_SIZE));
printf("%s\n",buf);
```

(client가 회원가입을 요청하는 부분)

Admin key를 맞추면, client는 ID, PW, admin_name을 입력하여 server에게 회원가입을 요청한다.

Server는 client로부터 받은 ID가 admin_data.csv에 존재하는지 검사하고, 존재한다면 ID가 존재한다는 메시지를 전달하고,

존재하지 않으면 admin_data.csv 에 회원가입정보를 저장한다.

```
//같은 id가 있는지 검사
while(!feof(admin_fp))
{
    fgets(str_tmp,BUF_SIZE,admin_fp);

    p = strtok(str_tmp,", \n");
    int cnt = 0;

    while(p != NULL)
    {
        strcpy(tmp[cnt],p);
        cnt++;
        p=strtok(NULL, ", \n");
    }
    if(!strcmp(tmp[0],sign_up_admin.id))
    {
        isExist = 1;
        fclose(admin_fp);
        break;
    }
}
```

(server가 같은 ID있는지 검사하는 부분)

b. 로그인

```c
while(login_try < 5)
{
    system("clear");

    printf("*****************************\n");
    printf("          Login              \n");
    printf("*****************************\n");
    printf(" ID : ");
    scanf("%s",login_id);
    printf(" P.W : ");
    while(getchar() != '\n');
    strcpy(login_pw,password(login_pw));

    secure_write(socketfd,login_id,BUF_SIZE);
    sleep(1);
    secure_write(socketfd,login_pw,BUF_SIZE);
    sleep(1);

    memset(buf, 0 , BUF_SIZE);
    strcpy(buf,secure_read(socketfd,buf,BUF_SIZE));
```

(client의 sign_in함수)

```c
strcpy(sign_in_admin.id,secure_read(socketfd,sign_in_admin.id,BUF_SIZE));
sleep(1);
strcpy(sign_in_admin.pw,secure_read(socketfd,sign_in_admin.pw,BUF_SIZE));

printf("입력받은 ID : %s\n",sign_in_admin.id);

FILE *admin_fp;
if((admin_fp = fopen(ADMIN_LIST,"rb")) == NULL)
{
    fprintf(stderr,"등록된 ID가 존재하지 않습니다.\n");
    secure_write(socketfd,"등록된 ID가 존재하지 않습니다.",BUF_SIZE);
    return FALSE;
}
```

(Server의 sign_in함수)

Client가 ID, PW를 server에게 전달하고, server는 전달받은 ID, PW가 맞는지 admin_data.csv파일을 열어서 확인한다. 만약 admin_data.csv파일에 존재하는 ID가 없으면 등록된 ID가 존재하지 않는다는 메시지와 함께, Client를 이전 메뉴로 돌려보낸다. 그리고 회원가입에서 했던 것처럼, admin_data.csv의 admin ID, PW와 일치하는지 검사한다.

일치한다면 로그인을 허용하고, 일치하지 않으면 loginTry를 1증가시킨다. LoginTry를 5회 모두 사용하면 프로그램사용을 60초간 중지시키고, 이전 메뉴로 돌려보낸다.

c. Exit(종료)

```c
else if(!strcmp(mode,"exit")){//프로그램 종료일때,
    secure_write(socketfd,"okay",BUF_SIZE);       //error 발생안한 것 체크용

    printf("Client가 연결 종료를 요청하였습니다.\n");
    printf("Client와의 연결을 종료합니다.\n");
    printf("program을 종료합니다.\n");

    close(socketfd);
    close(server_socketfd);
    sleep(1);
    exit(1);
```

(server의 exit처리 부분)

```c
case 3:{
        secure_write(socketfd,"exit",BUF_SIZE);
        memset(buf, 0,BUF_SIZE);
        printf("DB프로그램을 종료합니다.\n");
        sleep(1);
        close(socketfd);

        system("clear");
        exit(1);
        break;
    }
```

(client의 exit처리부분)

Client가 exit를 요청하면 server와 client모두 socket을 닫고, 프로그램을 종료한다.

8. 로그인 후 메뉴

```c
int server_section();                   //로그인 후 server section
void Employee_Registration();           //사원 등록 처리
void Department_Information();          //부서 정보 조회 처리
void Employee_Search();                 //사원 정보 조회 처리
void Employee_Delete();                 //사원 정보 삭제 처리
```

(Server의 함수들)

```
void client_section();  //로그인 후 client section
int print_menu(); //로그인 후 메뉴 선택하는 폼.
void Employee_Registration();   //사원 등록 함수
void Department_information();  //부서 정보 조회 함수
void Employee_Search(); //사원 정보 조회 함수
void Employee_Delete(); //사원 정보 삭제 함수
```

(client의 함수들)

```
printf("********* main menu **********\n");
printf("*****************************\n");
printf("       [1. 사원 등록]\n");
printf("       [2. 부서 정보 조회]\n");
printf("       [3. 사원 정보 조회]\n");
printf("       [4. 사원 정보 삭제]\n");
printf("       [5. 로그아웃]\n");
printf("       [6. 종료]\n");
printf("input(1~6): ");
```

(로그인 후 메뉴)

Admin이 로그인에 성공하면, Server는 server_section함수에 진입하고 Client는 client_section함수에 진입한다. 이 함수는 사원 등록, 부서 정보 조회, 사원 정보 조회, 사원 정보 삭제, 로그아웃, 종료의 6가지 기능을 수행한다. 처리 로직은 로그인 전 메뉴와 마찬가지로, Client가 메뉴를 선택하면, 관련 메시지를 보낸다. 그리고 server는 해당 server에 따라서 해당 메뉴의 처리 로직 함수로 들어간다.

a. 사원등록

```
system("clear");
printf("************ 사원 등록  *************\n");
printf("*************************************\n");
printf("**************부서 목록**************\n");
printf("*** 경영팀, 인사팀, 개발팀, 홍보팀 ***\n");
printf("*************************************\n");
printf("존재하는 부서 외에 입력할 수 없습니다.\n");

printf("사원의 부서 : "); scanf("%s",department);

if(!strcmp(department,"경영팀") || !strcmp(department,"인사팀")|| !strcmp
{
    break;
}
else{
    printf("등록되지 않은 부서입니다.\n");
    printf("다시 입력해 주세요.\n");
    sleep(1);
}
```

```
secure_write(socketfd,department,BUF_SIZE);

printf("employee_name : ");scanf("%s",employee_name);
secure_write(socketfd,employee_name,BUF_SIZE);

printf("employee_number : ");scanf("%s",employee_number)
secure_write(socketfd,employee_number,BUF_SIZE);

printf("join_year : "); scanf("%s", join_year);
secure_write(socketfd,join_year,BUF_SIZE);

printf("annual_salary : "); scanf("%s", annual_salary);
secure_write(socketfd,annual_salary,BUF_SIZE);

memset(buf,0,BUF_SIZE);
strcpy(buf,secure_read(socketfd,buf,BUF_SIZE));
fprintf(stderr,"%s\n",buf);
```

(Client의 Employee_Registration 함수 로직)

사원 등록 시, Admin은 부서명, 사원이름, 사원 번호, 입사년도, 연봉을 입력한다. 이 때, 존재하는 부서가 아니라면 부서명을 다시 입력 받는다. 그리고 admin은 Server의 응답을 기다린다.

```
FILE *employee_fp;
    if((employee_fp = fopen(EMPLOYEE_LIST,"a+t")) == NULL)
    {
        fprintf(stderr,"fopen error\n");
        exit(-1);
    }
```

```
//같은 사원번호가 있는지 검사
while(!feof(employee_fp))
{
    fgets(str_tmp,BUF_SIZE,employee_fp);
    fscanf(employee_fp, "%s %s %s %s %s",tmp[0],tmp[1],tmp[2],tmp[3],tmp[4]);

    if(!strcmp(tmp[2],employee_number))
    {
        isExist = 1;
        fclose(employee_fp);
        break;
    }
}
```

(Server가 employee_data.txt를 a+모드로 열어서 같은 사원 번호 있는지 검사.)

Server는 employee_data.txt를 확인해서, 입력 받은 사원 번호가 존재하는지 조회한다.

```
fprintf(employee_fp,"%s %s %s %s %s\n",department,employee_name, e
fclose(employee_fp);

FILE * department_fp;

char department_file[BUF_SIZE];
strcpy(department_file,department);
strncat(department_file,".txt",4);                              //부서

if((department_fp = fopen(department_file,"a+t")) == NULL)
{
    fprintf(stderr,"fopen error\n");
    exit(-1);
}

fprintf(department_fp,"%s %s %s %s %s\n",department,employee_name,
fclose(department_fp);
```

(server의 요청 받은 사원 번호가 존재하지 않을 때의 처리)

만약 존재한다면 사원 등록을 거부하고, 존재하지 않으면 employee_data.txt에 사원 정보를 저장하고, 입력 받은 부서명의 부서명.txt파일에도 사원 정보를 추가한다.

b. 부서 정보 조회

```
FILE * department_fp;
if((department_fp = fopen(department_file,"a+t")) == NULL)              //
{
    fprintf(stderr,"fopen error\n");
    exit(-1);
}
char temp_str[BUF_SIZE] = {0,};

while(!feof(department_fp))
{
    memset(temp_str,0,BUF_SIZE);
    fgets(temp_str,BUF_SIZE,department_fp);                  //파일 한줄 읽
    printf("%s",temp_str);
    secure_write(socketfd,temp_str,BUF_SIZE);               //client에게 전송
}

secure_write(socketfd,"파일 끝 도달",BUF_SIZE);
```

(Server의 Department_Information() 함수)

Admin이 조회할 부서명을 입력한다. 그리고 Server는 fopen을 사용해서 입력 받은 부서명.txt를 열고 fgets를 이용하여 사원마다의 정보를 admin에게 전송한다. 그리고 feof()를 이용해서 파일의 끝을 체크하고, Client에게 파일 끝이라는 메시지를 전송한다.

```
while (1)
{
    memset(buf, 0, BUF_SIZE);
    strcpy(buf, secure_read(socketfd, buf, BUF_SIZE));

    if (!strcmp(buf, "파일 끝 도달"))
    {
        memset(buf, 0, BUF_SIZE);
        break;
    }
    else
    {
        char tem[5][BUF_SIZE] = { 0, };
        char* p = strtok(buf, " \n");
        int cnt = 0;
        while (p != NULL)
        {
            strcpy(tem[cnt], p);
            if (cnt == 4)
                printf("%s", tem[cnt]);
            else
                printf("%s     ", tem[cnt]);
            cnt++;

            p = strtok(NULL, " \n");

        }
        printf("\n");
    }
}
```

(Client의 Department Information 함수)

```
printf("************************************************\n");
printf("부서 정보 조회를 마칩니다.\n");
printf("이전 메뉴로 돌아가고자 한다면, 아무거나 입력해주세요.\n");

char input[BUF_SIZE];
scanf("%s", input);
memset(buf, 0, BUF_SIZE);
```

(정보를 볼 시간 부여하기위한 것)

Admin은 server에게 전송 받은 부서 정보를 본다. 이때, strtok를 이용해서 사원의 정보를 하나씩 잘라서 출력한다. 이때, 부서 정보를 보는 시간을 부여하기 위해서, Admin이 정보를 그만 보고 싶을 때, 아무런 입력을 하도록 한다. (scanf이용).

c. 사원 정보 조회

Admin이 사원 정보 조회를 할 방식을 선택한다. 그리고 그 방식에 따라서 각 선택에 따라서, 입력하는 값이 달라진다. server는 각 방식에 따라서 strcmp로 비교하고, server화면에 출력할 print가 달라진다. (요청 받은 "~" : "~")

```
printf("***************** 사원 정보 조회 ******************\n");
printf("************************************************\n");
printf("****** 검색 옵션 ********************************\n");
printf("       [1. 사원 이름으로 검색]\n");
printf("       [2. 사원 번호로 검색]\n");
printf("       [3. 연봉으로 검색]\n");
printf("       [4. 입사년도로 검색]\n");
printf("Input(1~4) : ");
```

(사원 정보 조회 메뉴)

```
FILE *employee_fp;
if((employee_fp = fopen(EMPLOYEE_LIST,"a+t")) == NULL)
{
    fprintf(stderr,"fopen error\n");
    exit(-1);
}
```

(Server의 employee_data.txt fopen)

Server는 요청 받은 정보와 일치하는 사원을 찾기 위해서 employee_data.txt를 연다.

```c
//찾는 내용 있는지 검사
while(!feof(employee_fp))
{
    memset(send_tmp, 0, BUF_SIZE);
    memset(str_tmp, 0 , BUF_SIZE);
    fgets(str_tmp,BUF_SIZE,employee_fp);    //맨 첫줄 제외

    str_tmp[strlen(str_tmp)] = '\0';
    strcpy(send_tmp,str_tmp);

    p = strtok(str_tmp," \n");

    while(p != NULL)
    {
        if(!strcmp(p,buf))              //검색 요청사항이 있으면
        {
            printf("%s",str_tmp);
            secure_write(socketfd,send_tmp,BUF_SIZE);    //정보 전부 전달
            break;
        }
        p=strtok(NULL, " \n");
    }
}
memset(buf,0,BUF_SIZE);
secure_write(socketfd,"파일 끝 도달",BUF_SIZE);
```

(server의 정보가 존재하는 사원 검색 루틴)

Server가 입력 받은 정보가 한 줄의 정보를 strtok로 자른 p와 같은 지 여부를 판단하고, 같은 경우가 있으면 한 줄의 정보를 Client에게 전달한다. Client는 전달받은 정보를 잘라서 출력하여 admin이 확인한다. 그리고 파일 끝 도달 메시지를 보내고 함수를 종료한다. Client는 파일 끝 메시지를 받으면 함수 탈출을 위해서 admin의 입력을 기다린다.

d. 사원 정보 삭제

사원 정보 삭제 루틴은 사원번호를 입력 받고 하나의 사원만 삭제한다. 왜냐하면 사원번호만이 고유한 데이터이기 때문이다. 또한 한번에 여러 사원 정보를 삭제했을 경우, 잘못되는 경우를 방지하기 위해서 이다.

```c
void Employee_Delete()
{
    memset(buf, 0, BUF_SIZE);
    system("clear");

    printf("****************** 사원 정보 삭제 ******************\n");
    printf("*****************************************************\n");
    printf("삭제할 사원의 번호 : "); scanf("%s", buf);

    secure_write(socketfd, buf, BUF_SIZE);          //삭제할 사원의 번호 전달
    sleep(1);
    memset(buf, 0, BUF_SIZE);
    secure_read(socketfd, buf, BUF_SIZE);

    printf("%s\n", buf);
    sleep(1);

    return;
}
```

(Client의 사원 정보 삭제 함수)

Admin이 삭제할 사원의 사원 번호를 입력하고 Server로부터 성공, 실패의 메시지를

받게 된다.

```
FILE *employee_fp;
if((employee_fp = fopen(EMPLOYEE_LIST,"a+t")) == NULL)        //Userli
{
    fprintf(stderr,"fopen error!\n");
    return;
}

FILE *temp_fp;
if((temp_fp = fopen("./temp.txt","w+t")) == NULL)        //Userlist 파
{
    fprintf(stderr,"fopen error\n");
    exit(-1);
}
```

(Server의 두개의 파일 포인터를 이용한 두개의 파일 fopen)

Server는 사원 정보 삭제 루틴을 구현하기 위해서 employee_data.txt와 temp.txt의 파일을 연다. 왜냐하면 한 줄을 삭제하고 다음 줄을 위를 올리는 것이 파일 하나로는 어려웠기 때문이다. (방법이 있을지도 모르나, 실패를 많이 했음)

```
while(!feof(employee_fp))
{
    memset(str_tmp, 0 , BUF_SIZE);

    fgets(str_tmp,BUF_SIZE,employee_fp);
    strcpy(str_tmp2,str_tmp);

    char *p = strtok(str_tmp," \n");

    if(p==NULL) break;
    strcpy(delete_depart,p); //삭제된 부서 저장
    p = strtok(NULL," \n");
    p = strtok(NULL," \n");

    if(!strcmp(p,delete))              //검색 요청사항이 있으면
    {
        strcpy(delete_tmp,str_tmp2);    //삭제 내용 저장
        check = 1;                  //삭제한 놈 있음을 check
    }
    else
    {
        //삭제하는 것 발견 못했을때
        fwrite(str_tmp2,1,strlen(str_tmp2),temp_fp);        //삭
    }
```

(사원 정보 삭제 루틴 구현)

```
fclose(employee_fp);
fclose(temp_fp);
system("rm employee_data.txt");
system("mv temp.txt employee_data.txt");
```

(system 함수를 이용하여 삭제 구현)

Server는 employee_data.txt를 읽고, 삭제하고자 요청한 사원 번호와 일치하는 것이 있는지 검사한다. 또한 파일로부터 읽은 사원의 부서를 strcpy를 통해 저장한다. 그리고 일치하지 않는 정보는 temp.txt에 fwrite하고, 일치하는 정보가 있으면, check =1로 설정하고 fwrite하지 않는다. 그리고 employee_data.txt가 파일에 끝에 도달하면, 삭제를 요청한 사원을 제외하고 전부 temp.txt에 복사가 된다. 마지막으로 파일 포인터를

fclose하고 system함수를 이용해서 employee_data.txt를 삭제한 후 temp.txt를 employee_data.txt로 이름을 변경한다.

```
if(check)
{
    strncat(delete_depart,".txt",4);                        //부

    FILE * department_fp;
    if((department_fp = fopen(delete_depart,"a+t")) == NULL)
    {
        fprintf(stderr,"fopen error\n");
        exit(-1);
    }
    FILE *temp_fp;
    if((temp_fp = fopen("./temp.txt","wt")) == NULL)        //Use
    {
        fprintf(stderr,"fopen error\n");
        exit(-1);
    }
```

(삭제하는 사원의 부서명.txt에서 삭제)

위에서 일치하는 사원이 있으면 check = 1로 설정하였으므로, 저장해둔 부서명.txt 와 temp.txt를 열고, employee_data.txt에서의 사원 정보 삭제 루틴처럼 똑같이 처리한다.

```
char delete_txt[BUF_SIZE] = {0,};
strcpy(delete_txt,"rm ");
strcat(delete_txt,delete_depart);

char rename_txt[BUF_SIZE] ={0,};
strcpy(rename_txt,"mv temp.txt ");
strcat(rename_txt,delete_depart);

system(delete_txt);
system(rename_txt);
printf("Client가 요청한 사원 삭제를 성공했습니다.\n");
secure_write(socketfd,"삭제를 성공했습니다.",BUF_SIZE);
```

(부서명.txt 삭제에서 system 함수 이용)

부서명은 고정된 값이 아니므로, strcpy와 strcat 함수를 이용해서 확인한 부서명에 관한 삭제 문자열을 완성한다. (“rm”+”부서명.txt”, ”mv temp.txt” + ”부서명.txt”)

마지막으로 admin에게 요청한 사원 삭제를 성공했다는 메시지를 보내준다.

```
else
{
    printf("Client가 없는 data를 요청했습니다.\n");
    secure_write(socketfd,"존재하지 않는 사원입니다.",BUF_SIZE);
}
```

만약 없는 사원정보 삭제를 요청했으면 존재하지 않는 사원이라고 메시지를 보낸다.

e. 로그아웃

```
else if (select == 5)
{
    secure_write(socketfd, "log out", BUF_SIZE);
    strcpy(buf, secure_read(socketfd, buf, BUF_SIZE));

    memset(buf, 0, BUF_SIZE);

    printf("DB Section을 종료하고 로그아웃합니다.\n");
    break;
}
```

(client의 로그아웃 부분)

```
else if(!strcmp(section_mode,"log out"))
{
    secure_write(socketfd,"ok",BUF_SIZE);
    break;
}
```

(server의 logout부분)

Admin이 로그아웃을 선택하면 server에게 log out 메시지를 보낸다. 그리고 server는 log out 메시지를 받으면 ok라는 메시지를 보낸다. 이 것은 admin에게는 보이지 않는다. 그리고 DB section을 종료하고 이전 메뉴로 돌아간다.

(Server: server_section() -> Server() / Client: client_section() -> Client())

f. 종료

```
else if (select == 6)
{
    secure_write(socketfd, "exit", BUF_SIZE);
    strcpy(buf, secure_read(socketfd, buf, BUF_SIZE));

    memset(buf, 0, BUF_SIZE);

    printf("DB프로그램을 종료합니다.\n");

    sleep(1);
    close(socketfd);

    system("clear");
    exit(1);
}
```

(Client의 exit 부분)

```
else if(!strcmp(section_mode,"exit"))
{
    secure_write(socketfd,"ok",BUF_SIZE);
    printf("Client가 연결 종료를 요청하였습니다.\n");
    printf("Client와의 연결을 종료합니다.\n");
    printf("program을 종료합니다.\n");

    close(socketfd);            //socket
    close(server_socketfd);     //close
    sleep(1);
    exit(1);                    //server 종료
}
```

(Server의 exit 부분)

Admin이 종료 메뉴를 선택하면, Client가 server에게 exit 메시지를 보내고, server는 ok라는 메시지를 보냈다. 그리고 각각 socket을 close하고 프로그램을 종료한다.

2.4. 동작실험

- **동작실험 환경: Google Cloud Platform**

  Instance1: Client (Ubuntu 18.04) (IP: `10.178.0.3`)

  Instance2: Server (Ubuntu 18.04) (IP: `10.178.0.2`)

- **동작실험의 아쉬운 점: GCP가 한글을 입력하면 깨지는 현상이 발생**

1. 실행 및 AES키교환

```
sangjeong100@sj-instance-1:~$ ./client
usage : ./client "YOURIP" "PORT"
sangjeong100@sj-instance-2:~$ ./server
usage : ./server "YOURIP" "PORT"
```

(인자없이 실행했을 경우)

```
sangjeong100@sj-instance-2:~$ ./server 10.178.0.2 4000
server의 종료를 원하면 Ctrl + C 를 눌러주세요.
connection listening..
Client Connected, send public key to client...
Recieved Symmetric Key : 9C 64 D2 00 3E BC 37 11 7D 2F 92 87 1F 8B ED 8E 6C F9 E6
 32 F8 CE 25 81 6F BC 9C B7 02 4D B0 64
Recieved AES iv : B2 11 2B 06 F5 62 86 EB C6 BE AB 4C E0 A1 6C A4
키 교환 채널을 닫고 일반 채널을 여는중..
Client와의 일반 통신 채널열렸습니다.
```

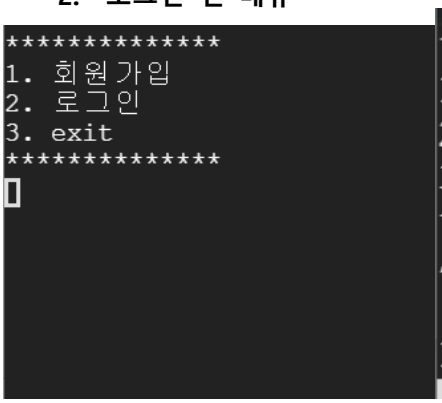
(제대로 된 Server실행 및 키교환, 이중 통신채널 표시)

```
sangjeong100@sj-instance-1:~$ ./client 10.178.0.2 4000
connecting..
Connected!
read RSA public key from server...

Recieved Public key
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAx63h6F1+GUIRzA/s8xsLRZRunDpE5zILgPyUl5MExdJwm7Ho6aq
ZKIuXMaAhEdaPWqkKZWaRSkmzEWau6m3Q08gTo3wDOGxC5qbTtLxSMgt+JQY+or8
poyrxS6WMHp9/sTOO6PPIU4j3sTIhkkxbjKJGTJKgEHy9Z0V6TxrVq+gECoP0v+U
AnuxdGVOZV0WWcZmXT5FKaJdehVQeHGEcEwzcUJG8J+pQsfoBZd6yxA8222f7d21
lCiaXdQX3AlmifsTdzOaeAFI6xzF0BhcyE8gjAXVULS4zFi+nHTiT2RT1tzfvpZt
WsgkrNM/kH7yUBDUY0BqQ92lZQqgOwuORwIDAQAB
-----END RSA PUBLIC KEY-----


AES symmetric key : 9C64D2003EBC37117D2F92871F8BED8E6CF9E632F8CE25816FBC9CB7024DB
064
AES iv : B2112B06F56286EBC6BEAB4CE0A16CA4
키 교환 채널을 닫고 일반 채널을 여는중..
일반 통신 채널 open 완료!
connected!
```

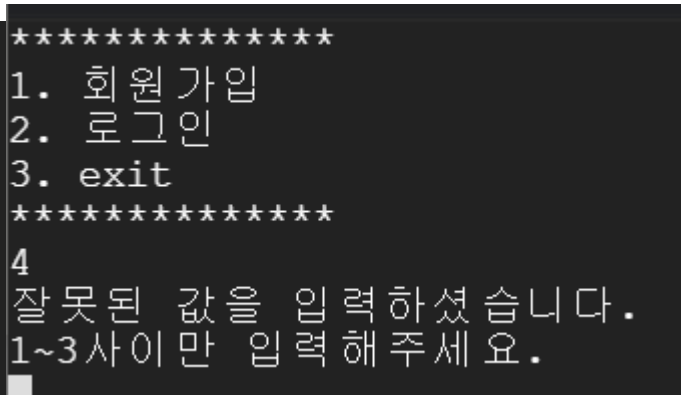
(제대로 된 Client실행 및 키교환, 이중 통신채널 표시)

Server와 Client가 Server의 IP와 PORT를 인자로 실행하여 키 교환 및 키교환 채널을 닫고, 일반 통신 채널을 연 모습이다. 구현 과정은 복잡했지만, 실행화면은 단순하다. Client가 받은 RSA 공개키를 출력하고, 생성한 AES 키와 iv를 Server가 받아서 출력한다.

2. 로그인 전 메뉴



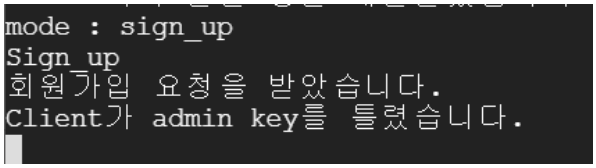
(Client의 로그인 전 메뉴)



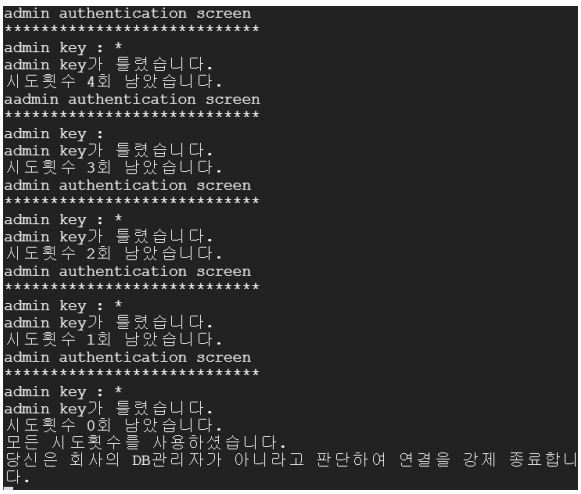
(1~3이외의 다른 값 입력한 경우)

a. 회원가입

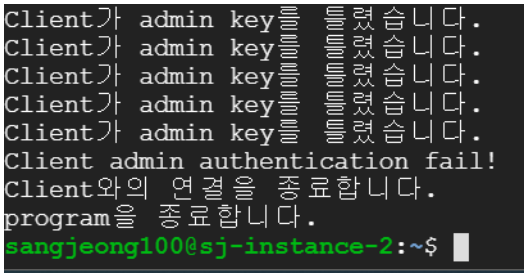
(Client의 회원가입 선택 후, 인증 틀린 경우)



(Server의 log 화면)

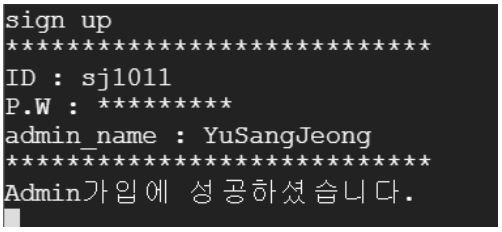


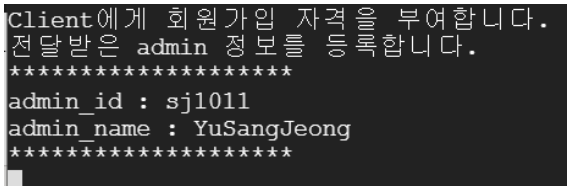
(Client가 admin 인증 모두 실패한 경우)



(Server의 log 화면)

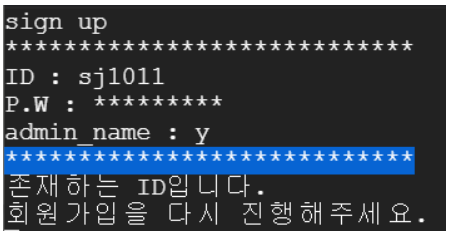
Admin 인증 실패 처리가 잘 이루어 지고, Admin 인증이 모두 실패한 경우에 Client와 Server모두 종료되는 모습 확인할 수 있다.

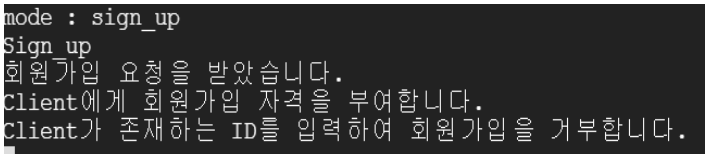




(admin인증 완료 후 회원가입 모습)　　(server의 log 화면)

인증 성공 후, 회원가입이 잘 이루어진 모습 확인할 수 있다.

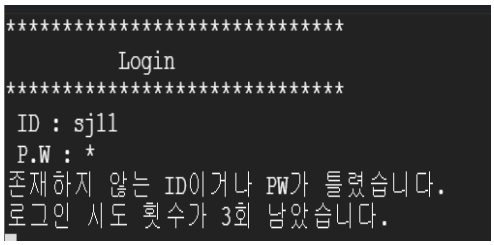


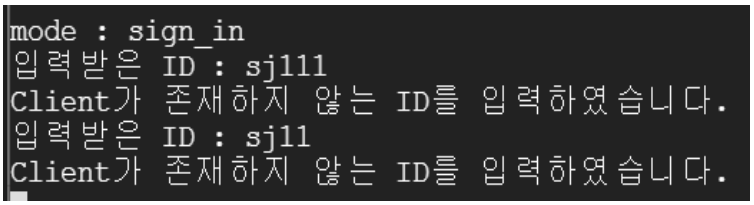


(이미 존재하는 ID로 회원가입 요청한 경우)　　(server의 log화면)

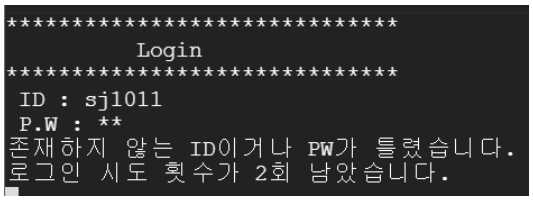
이미 존재하는 ID에 대한 처리가 잘 이루어 진다.

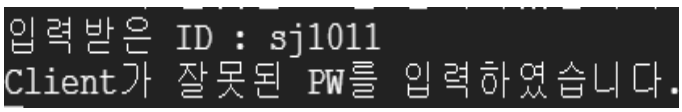
b. 로그인





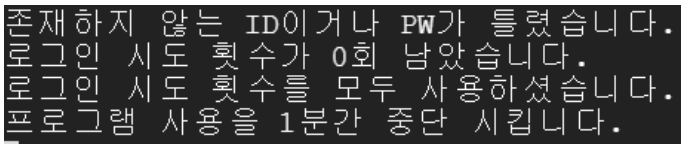
(존재하지 않는 ID 입력한 경우)　　(Server의 log 화면)



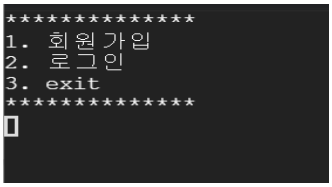


(PW가 틀린 경우)　　(Server의 log 화면)

ID를 틀리거나 PW를 틀리는 경우 모두 "존재하지 않는 ID이거나 PW가 틀렸습니다." 라는 메시지를 띄운다. 왜냐하면 ID가 틀린 것인지, PW가 틀린 것인지 유추하지 못하도록 하기위해서이다.

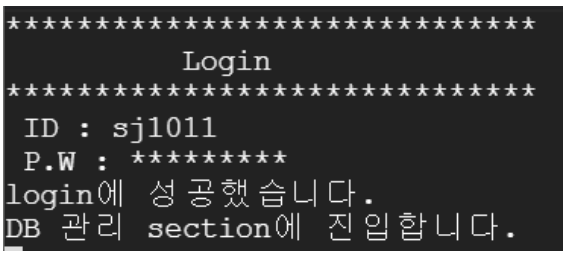


(Client가 로그인 시도 횟수 모두 사용한 경우)

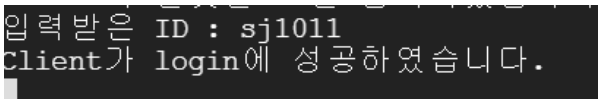


(Client의 60초 후 모습)

로그인 시도 횟수를 모두 사용한 경우에, 60초 후 다시 메뉴선택화면이 나옴을 확인할 수 있다.



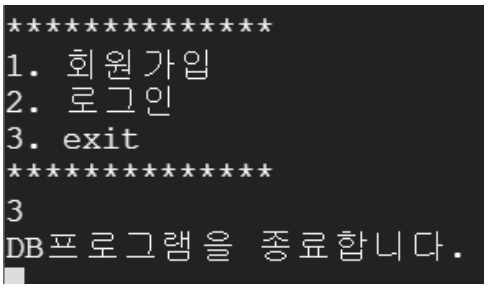
(Client가 로그인 성공한 경우)



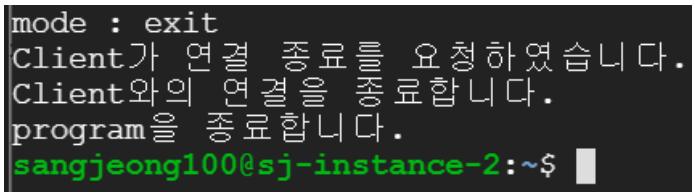
(Server의 log 화면)

Client가 로그인에 성공한 경우가 잘 이루어 짐을 확인했다. 이 이후부터는 Admin이라 칭할 수 있다.

c. Exit



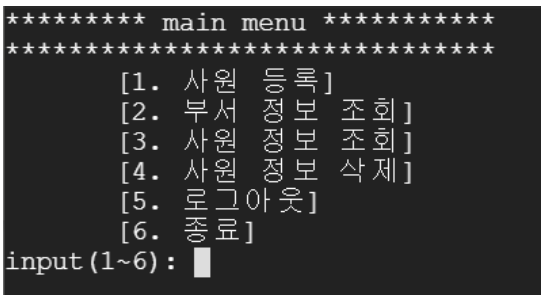
(Client가 exit 요청한 경우)



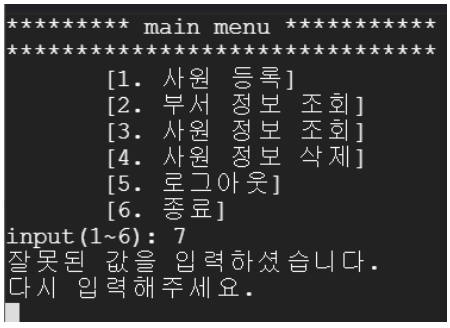
(Server의 log 화면)

Client가 DB프로그램 종료를 요청한 경우, Client와 Server둘다 종료된다. 이때, Client는 system("clear")를 호출하고 종료한다.

3. 로그인 후 메뉴

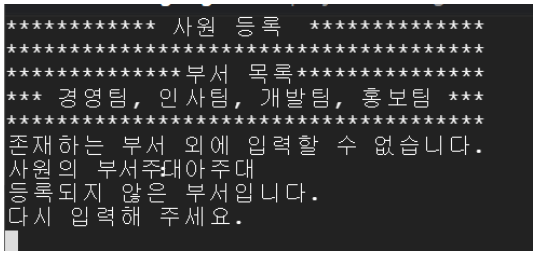


(로그인 후 메뉴)

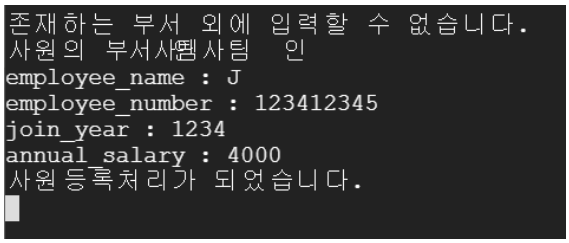


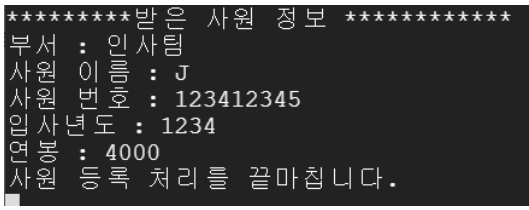
(1~6이외의 값 입력한 경우)

a. 사원 등록



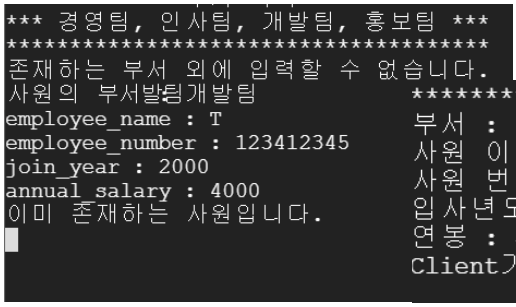
(존재하지 않는 부서 입력한 경우)

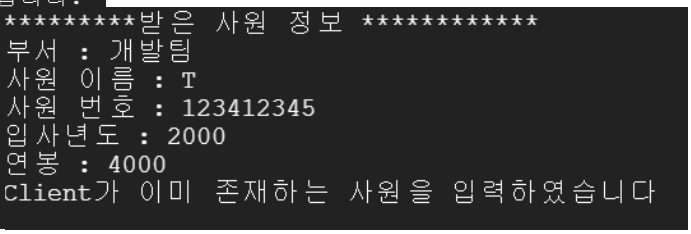




(존재하는 부서 입력하여 사원정보 입력한 경우) (Server의 log 화면)

GCP의 한글 입력 시 깨지는 현상에 의해서 이상하게 보이는 것일 수 있으나, Server의 log화면을 확인해보면, 전달은 문제없이 됐 음을 알 수 있다.

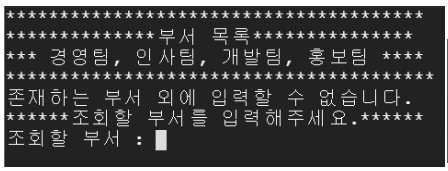


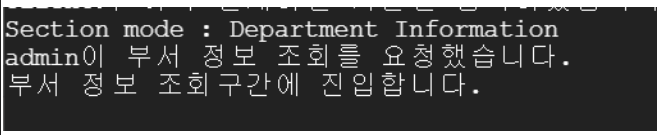


(이미 존재하는 사원 번호 입력한 경우) (Server의 log 화면)

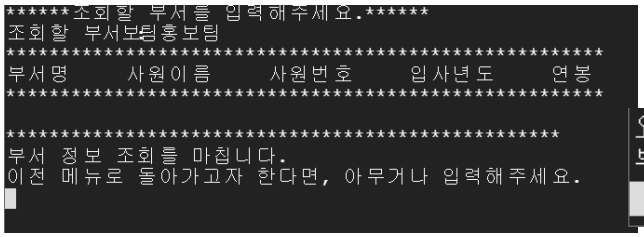
사원 번호는 고유한 번호이므로, employee_data.txt에 이미 존재한다면 사원 등록을 거부함을 확인했다.

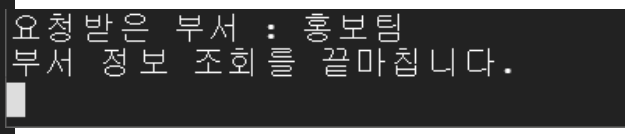
b. 부서 정보 조회





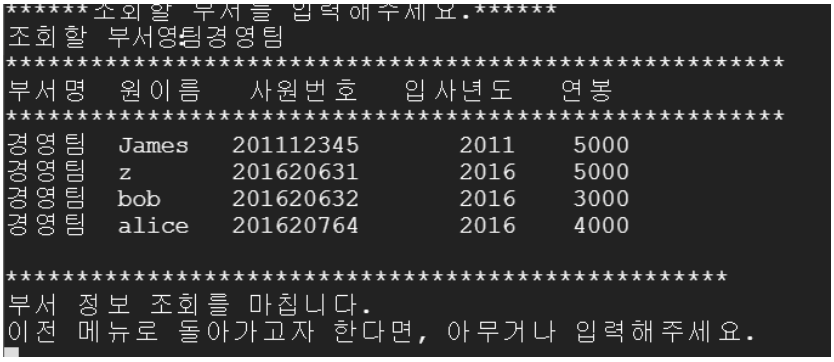
부서 정보 조회의 초기 화면이 잘 나타남을 확인했다.





(부서명.txt 파일이 존재하지 않았을 때) (server의 log 화면)

부서명.txt가 존재하지 않는 경우, server는 내용이 없는 부서명.txt를 만들고 admin에게 아무런 정보도 전달하지 않는다. 그리고 이전 메뉴를 돌아가기 위해 아무거나 입력해 야함을 확인했다.

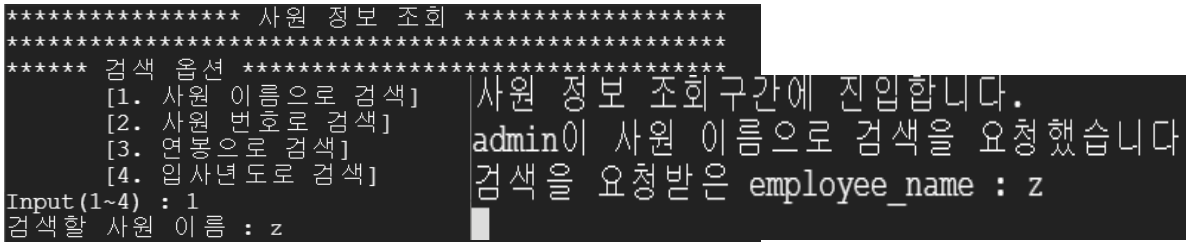


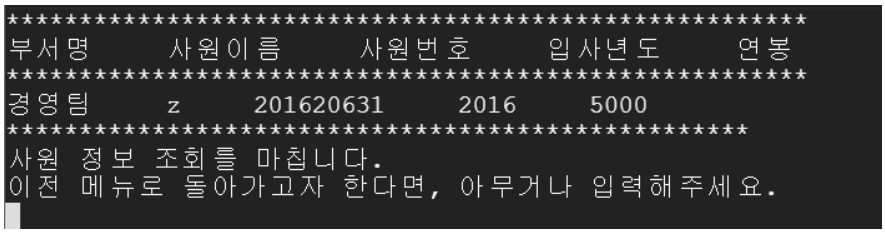
(부서원이 존재하는 부서 조회)

부서원 모두 출력됨을 확인했다.

c. 사원 정보 조회

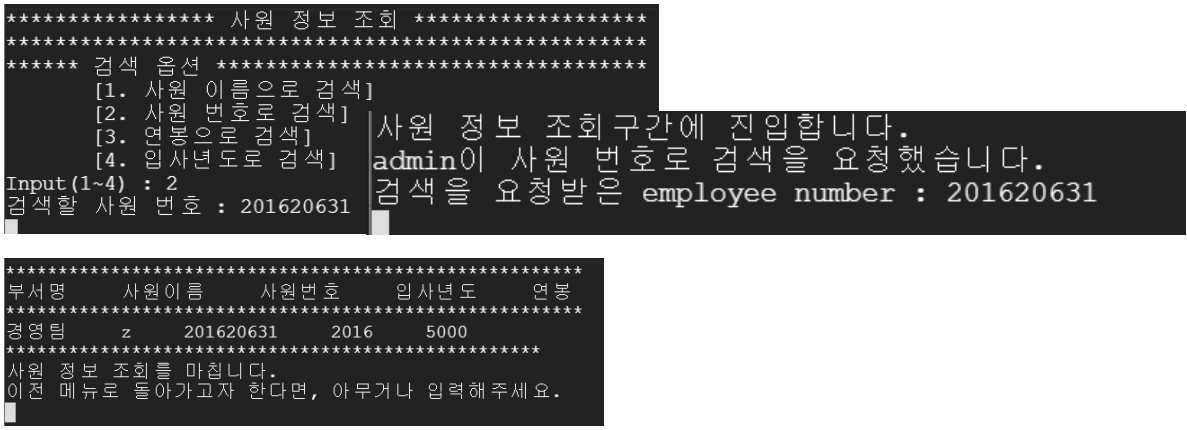
(사원 이름으로 검색한 경우)





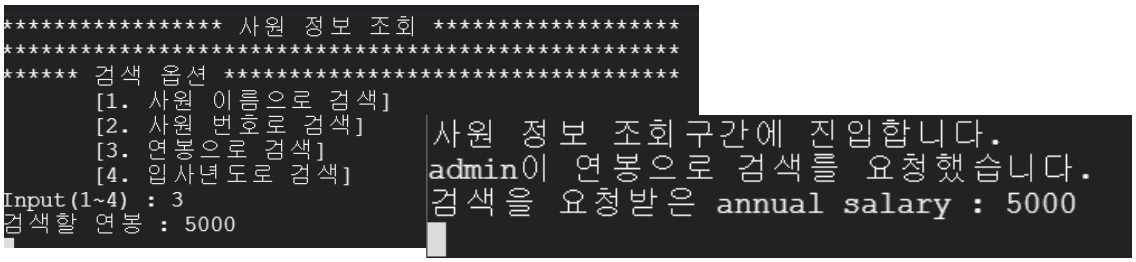
이름으로 검색한 경우 잘 나타남을 확인했다.

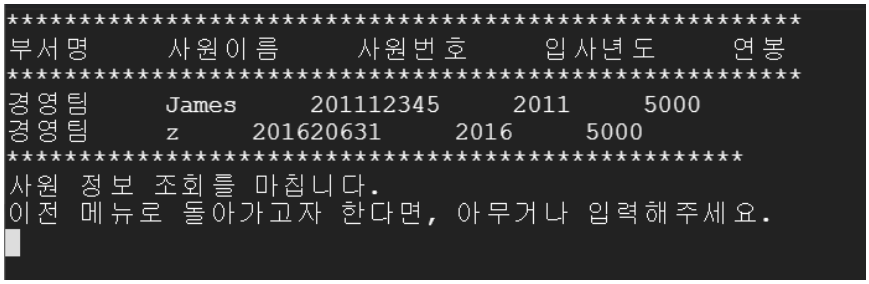
(사원 번호로 검색한 경우)



사원 번호로 정보 조회한 경우 문제없이 나타남을 확인했다.

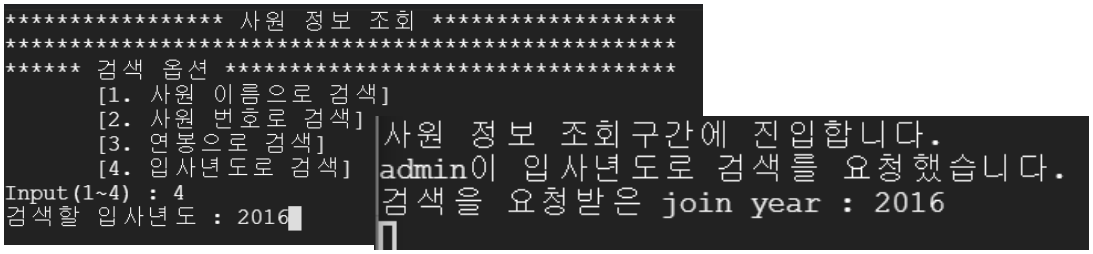
(연봉으로 검색한 경우)

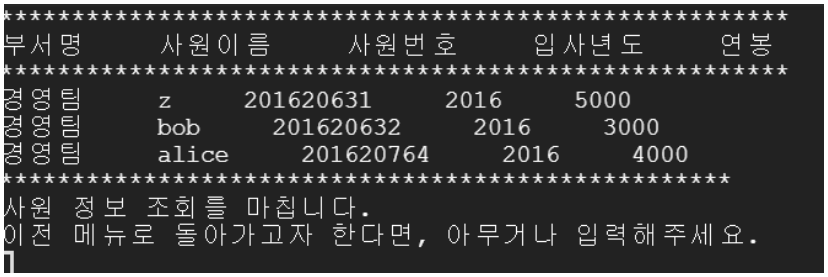




연봉으로 정보 조회한 경우에 문제없이 나타남을 확인했다.

(입사년도로 검색한 경우)



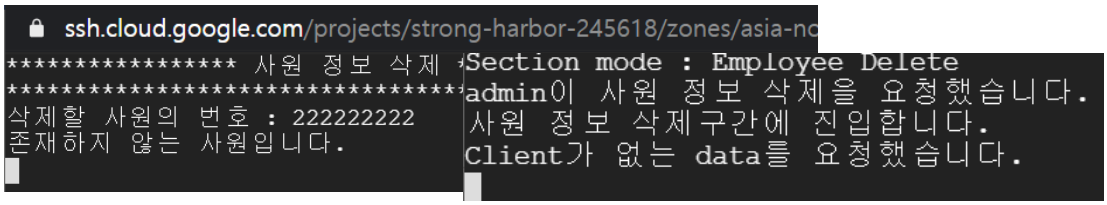


입사년도로 정보 조회 요청한 경우에 문제없이 나타남을 확인했다.

d. 사원 정보 삭제

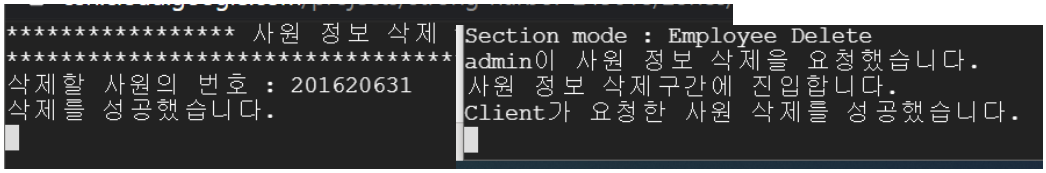
사원 정보 삭제의 초기 화면 및 Server의 log가 잘 나타났다.

(Client가 없는 사원 번호를 입력한 경우)

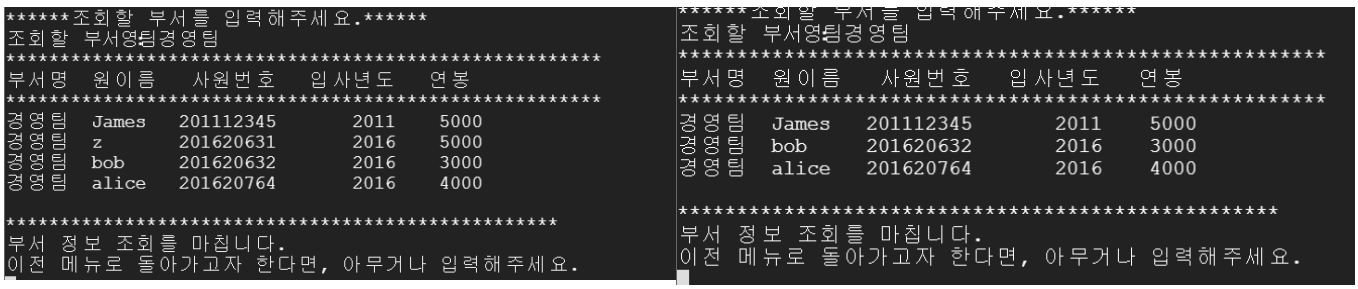


Client가 존재하지 않는 사원의 번호를 입력한 경우에 처리가 되지 않음을 확인하였다.

(Client가 존재하는 사원의 번호를 입력한 경우)



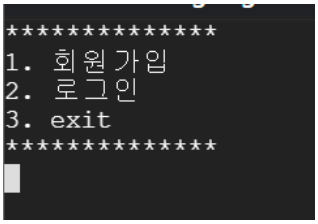
Z 사원의 사원 번호인 201620631의 삭제요청이 성공된 화면이다.



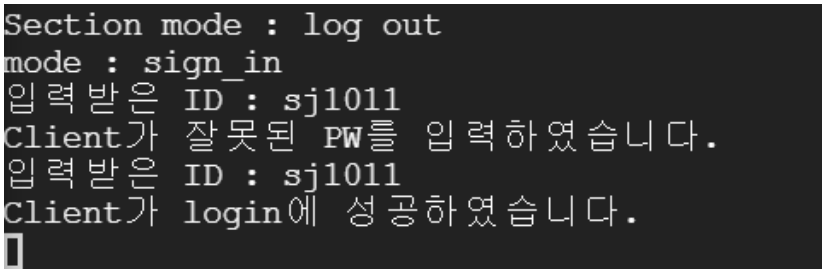
Z사원 정보 삭제 전 경영팀 조회 결과와 삭제 후의 경영팀 조회 결과이다.

위의 사진을 보면 z 사원의 정보 삭제가 성공적으로 이뤄졌음을 확인할 수 있다.

e. 로그아웃

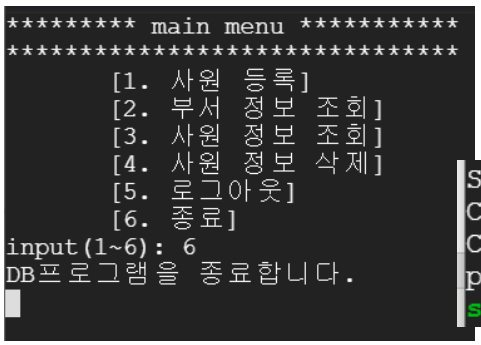


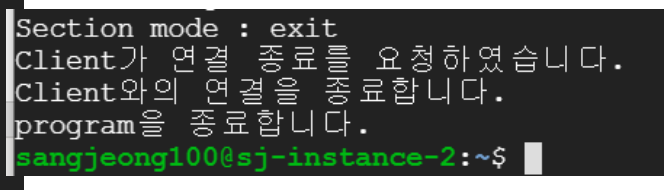
로그아웃 선택하여, 이전 메뉴로 돌아간 메뉴이며, Server가 Client의 Section mode 선택이 log out임을 인식했다는 것을 확인하였다.



로그아웃 후, 로그인 전 메뉴의 이용이 문제없이 되어서, 재 로그인에 성공한 것을 Server의 log를 통해 확인할 수 있다.

f. 종료





Client가 종료를 요청하여서 Server가 이를 수용하고, Client와 Server모두 종료되었음을 확인하였다.

3. 결론 및 토의

이번 종합 프로젝트의 주제를 사원 DB 관리 프로그램으로 진행하였다. 그래서 실제 회사의 DB 관리자가 필요한 것이 무엇일지 고민해보았다. 그 중, 가장 중요하다고 생각한 부분은 역시 보안측면이라고 생각했다. 그리고 '어떤 암호를 써야하나?'를 고민해보았다. 그래서 결정한 것이 RSA를 통해서 AES 대칭키를 공유하고 AES로 암/복호화를 진행하는 것이었다. 하지만 처음 암/복호화를 구현하려고 하였을 때, 어떤 식으로 접근하는지 몰라서 많이 고생하였다. 하지만 C언어에서 좋은 암/복호화 서비스를 제공하는 openssl을 알게 되었고, 나에게 정말 필요한 서비스임을 깨 달았다. 그래서 openssl을 사용하고자 노력하였고, 많은 우여곡절을 겪었지만 암/복호화 시스템을 구축하는데 성공하였다. 또한 한가지 생각한 점으로, 키 교환 채널을 따로 만들면 조금 더 안전하지 않을까 생각하였다. 그래서 openssl에서 제공하는 bio socket을 사용하여 키교환을 진행하고, 키 교환이 완료되면 일반 socket으로 암/복호화 통신을 진행했다. 이것으로 통신 적인 보안측면은 성공적으로 구현하였다고 판단했다. 다음으로 회사 DB 프로그램에 중요하다고 생각한점은 관리자 인증이다. 그래서 DB프로그램의 회원가입을 하기위해선 admin_key로 admin 인증을 하도록 했다. 이 admin key는 admin이라면 미리 알고 있어야 한다고 생각했다. 그래서 admin_key는 미리 공유되었다고 가정했다. 혹시라도 잘못된 사용자가 admin의 pc에 접속해서 몰래 DB를 건드릴 경우를 방지하고자 만든 시스템이다. 이 것을 만듦으로써 보안적인 측면을 한층 더 상승시켰다. 다음으로 파일을 이용한 DB를 구현하는 것이었다. 물론 DB서버를 이용하면 훨씬 쉬웠을 것이다. 하지만 이번 프로젝트는 수업에서 배운 내용을 이용하는 것이었으므로 고급 파일 입출력을 사용하기로 결정했다. 하지만 이 것을 구현하던 중, 많은 문제를 마주쳤다. 그것은 fopen할 때, 파일을 여는 mode에 따라서 많이 달라지는 것이다. 즉, 난 파일이 있으면 읽고, 없으면 생성하기 위해서 "w+" mode로 fopen하였다. 하지만 이 mode로 열었을 때, 내용이 있는 경우에 모든 내용을 지워버리는 것이었다. 그래서 이를 해결하기위해, "a+" mode로 바꾸었다. 이 문제를 해결하면서 고급 파일 입출력에 대해, 더욱 깊게 공부하는 계기가 되었다. 그래서 본 기능인 DB관리 메뉴도 잘 구현하였다.

이번 프로젝트의 진행은 나에게 있어서 정말 큰 도전이 였다. 왜냐하면 혼자서 2000줄이 넘는 코드를 짜본 것은 처음이기도 했으며, C언어에서 암호화를 구현하는 것이나 리눅스에서 비밀번호 입력을 "*"로 표시하는 것 등 많은 것이 처음이 였다. 또한 수많은 오류와 마주쳤지만, 그 중, segment falut와 마주쳐서 굉장히 어려움을 느꼈다. 그래서 이를 해결하고자 gdb의 사용법을 공부하였고, 결국 gdb를 통해서 많은 segment falut를 해결하였다. 그래서 이번 프로젝트는 정말 나의 실력향상에 도움이 되었으며, 다른 프로그램을 만들 때에도 할 수 있을 것이라는 자신감이 크게 붙었다. 무엇보다 C언어의 실력이 많이 상승한 것 같다. 이 것으로 최종 프로젝트 결과보고서를 끝마치겠다.