IPP 일학습병행 인턴

학습 내용 정리

인턴 기간 : 2021.8.1 ~ 2022.2.28

회사 : ㈜시스원

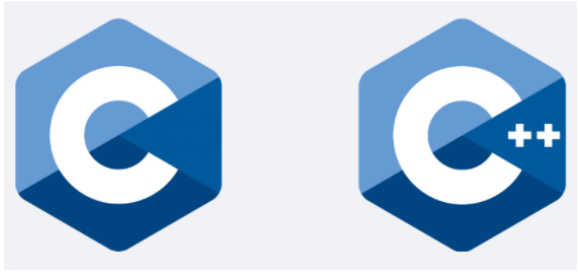
이름 : 김덕용

개발환경

Rocky Linux™

- VirtualBox를 이용해 리눅스 VM 개발환경 구축
- CentOS, Ubuntu, Rocky Linux 설치 => 주로 **Rocky Linux**에서 작업

개발 언어 및 사용한 DB



- 개발 언어 : C, C++
- 사용한 DB : PostgreSQL, Oracle Database 19c

1. java기반 작업

처음에 c, c++ 이 익숙하지 않았기 때문에 학부생 때 많이 사용한 java를 이용해 간단한 과제들을 수행

1. Java 파일 복사, 비교 프로그램
   - Java의 Stream 기능 사용

2. Java 파일 원격 업로드 프로그램 (소켓 프로그래밍)
   - Java의 TcpSocket, TcpServerSocket, Thread를 사용하여 Client, Server 프로그램 작성

3. Java로 간단한 인메모리 DB 흉내내기
   - Java HashMap 사용

=> java로 구현한 위의 1,3번을 c/c++로 재구현

2. C 기반 멀티쓰레드 서버 구현

1. 멀티쓰레드 기반 에코서버를 구현한 후 소켓을 통해 파일을 받아 저장하도록 구현
2. 파일명은 현재 시간명으로 저장되도록 함

※ 참고 서적 : 열혈 TCP/IP 프로그래밍

client1차과제.c

C_Project > workspace > client1차과제.c > main(int, char * [])

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<pthread.h>

void error_handling(const char* message);
void *send_file(void* arg);

#define BUF_SIZE 1024

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

    serv_sock = socket(AF_INET,SOCK_STREAM,0);
    if(serv_sock == -1){
        error_handling("socket error!!\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(connect(serv_sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr)) == -1){
        error_handling("connect error!!\n");
    }else{
        fputs("server connect success!!\n",stdout);
    }

    int thread_param;
    pthread_create(&t_id,NULL, send_file,(void*)&serv_sock);
    pthread_join(t_id,(void*)&thread_param);
    close(serv_sock);
    return 0;
}


void* send_file(void* arg){
    int serv_sock = *((int*)arg);
    char file_name[BUF_SIZE] = {0, };
    char message[BUF_SIZE] = {0, };
```

server1차과제.c

C_Project > workspace > server1차과제.c > ...

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<time.h>
#include<pthread.h>

#define _REENTRANT

void error_handling(const char* message);
void* handling(void* arg);

int clnt_cnt = 0; // 서버에 접속한 클라이언트수
int clnt_socks[256];

pthread_mutex_t mutx;

int main(int argc, char* argv[]){
    int serv_sock,client_sock;
    struct sockaddr_in serv_addr, client_addr;
    socklen_t client_addr_size;
    pthread_t t_id;

    pthread_mutex_init(&mutx, NULL);
    serv_sock = socket(AF_INET,SOCK_STREAM,0);

    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(bind(serv_sock,(struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1){
        error_handling("bind error!!\n");
    }
    if(listen(serv_sock, 5) == -1){
        error_handling("listen error!!\n");
    }

    while(1){
        client_addr_size = sizeof(client_addr_size);
        client_sock = accept(serv_sock,(struct sockaddr*)&client_addr,&client_addr_size);
        if(client_sock == -1){
            error_handling("accept error!!\n");
        }else{
            pthread_mutex_lock(&mutx);
```

3. HTTP 멀티쓰레드 소켓 서버 구현

1. 구현한 서버를 특정 포트에서 열고 주소를 입력했을 때 화면에 성공 혹은 ERROR가 출력되도록 구현

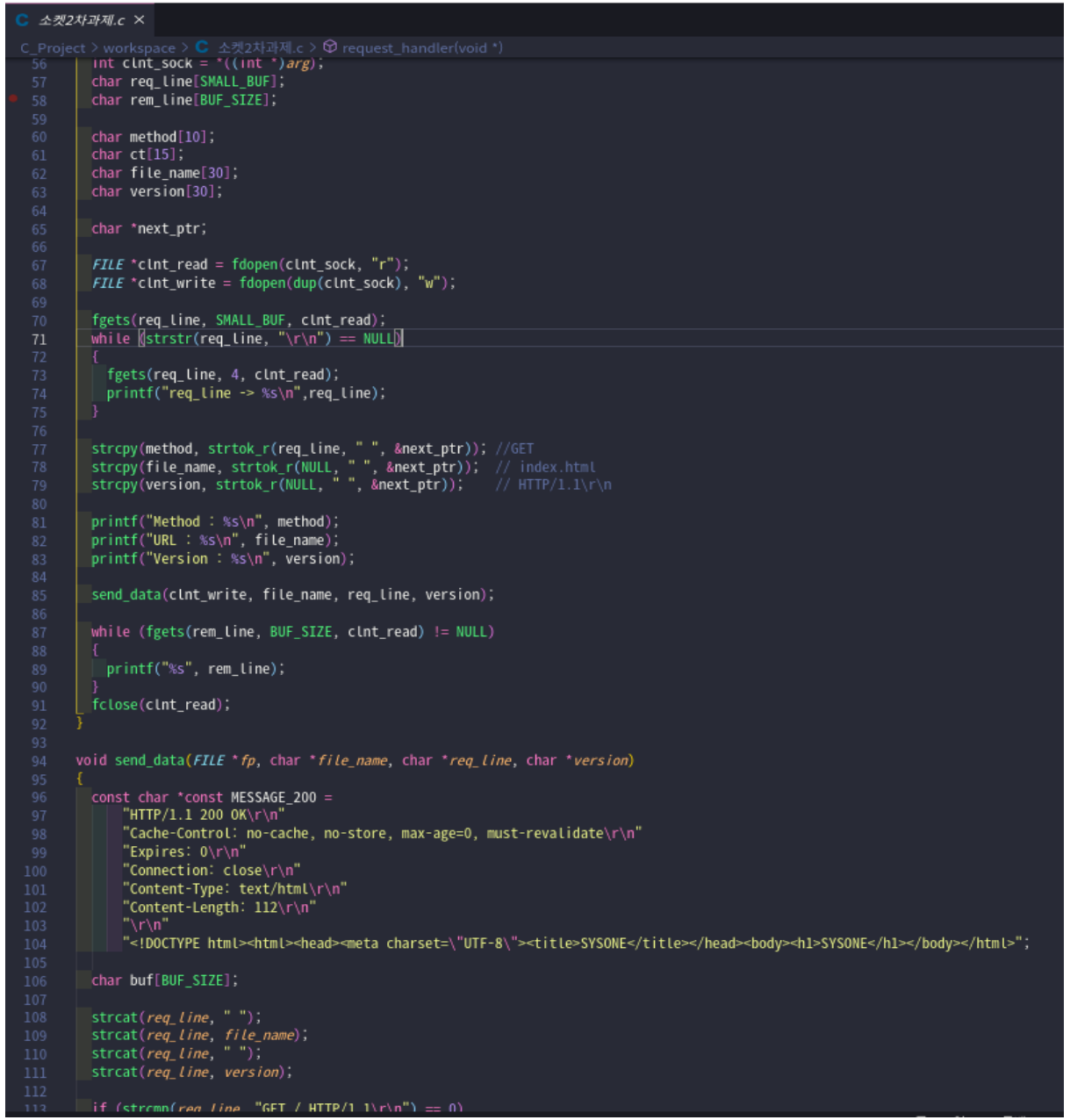
2. 웹 브라우저(클라이언트)가 서버로 접속을 시도했을 때 request header에 담긴 정보들을 소켓에서 read()한 후 원하는 데이터를 형식에 맞게 콘솔에 출력

출력 예시) Method : GET
URL : /
Version : HTTP/1.1

Connection : keep-alive
Host : 192.168.56.101
User-Agent : Mozilla/5.0
...

구현할 때 어려웠던 부분

- http request header와 response header의 구조를 이해하고 전송된 데이터에서 원하는 데이터만 추출하는데 어려움을 겪음

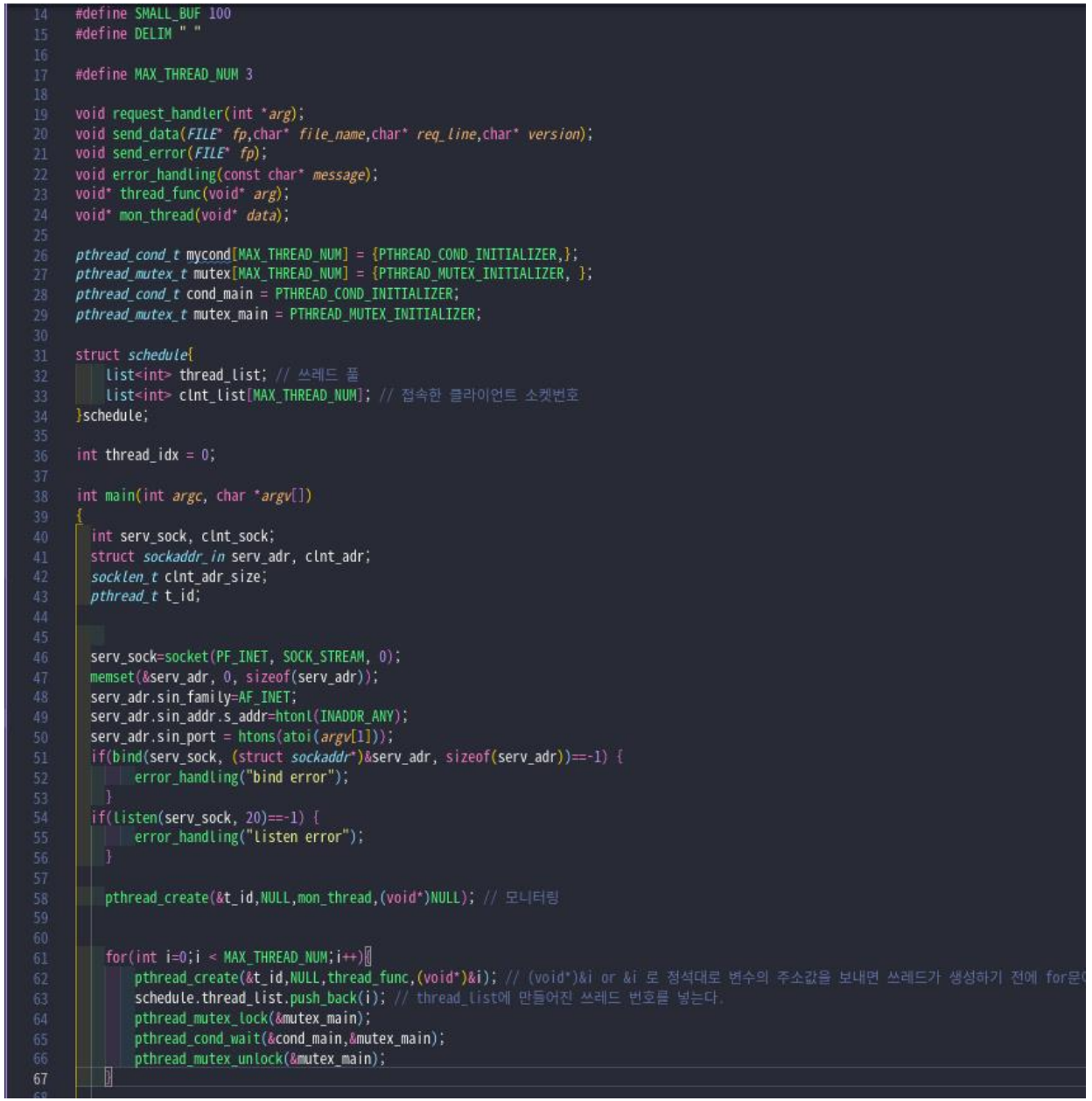

소켓2차과제.c
C_Project > workspace > 소켓2차과제.c > request_handler(void *)
int clnt_sock = *((int *)arg);
char req_line[SMALL_BUF];
char rem_line[BUF_SIZE];
char method[10];
char ct[15];
char file_name[30];
char version[30];
char *next_ptr;
FILE *clnt_read = fdopen(clnt_sock, "r");
FILE *clnt_write = fdopen(dup(clnt_sock), "w");
fgets(req_line, SMALL_BUF, clnt_read);
while (strstr(req_line, "\r\n") == NULL)
{
fgets(req_line, 4, clnt_read);
printf("req_line -> %s\n",req_line);
}
strcpy(method, strtok_r(req_line, " ", &next_ptr)); //GET
strcpy(file_name, strtok_r(NULL, " ", &next_ptr)); // index.html
strcpy(version, strtok_r(NULL, " ", &next_ptr)); // HTTP/1.1\r\n
printf("Method : %s\n", method);
printf("URL : %s\n", file_name);
printf("Version : %s\n", version);
send_data(clnt_write, file_name, req_line, version);
while (fgets(rem_line, BUF_SIZE, clnt_read) != NULL)
{
printf("%s", rem_line);
}
fclose(clnt_read);
}
void send_data(FILE *fp, char *file_name, char *req_line, char *version)
{
const char *const MESSAGE_200 =
"HTTP/1.1 200 OK\r\n"
"Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\n"
"Expires: 0\r\n"
"Connection: close\r\n"
"Content-Type: text/html\r\n"
"Content-Length: 112\r\n"
"\r\n"
"<!DOCTYPE html><html><head><meta charset=\"UTF-8\"><title>SYSONE</title></head><body><h1>SYSONE</h1></body></html>";
char buf[BUF_SIZE];
strcat(req_line, " ");
strcat(req_line, file_name);
strcat(req_line, " ");
strcat(req_line, version);
if (strcmp(req_line, "GET / HTTP/1.1\r\n") == 0)

4. HTTP 멀티쓰레드 소켓 서버 (스레드풀)

1. 프로그램을 처음 실행할 때 지정한 개수만큼의 thread를 미리 생성 (3~10개)
2. 첫번째 요청 -> 0번 thread, 두번째 요청 -> 1번 thread, 세번째 요청 -> 2번 thread, 네번째 요청 -> 0번 thread 식으로 쓰레드를 부여
3. C++의 list<int>와 pthread mutex를 thread 개수만큼 생성하고 이를 사용하여 thread별 클라이언트 소켓 목록을 관리하도록 구현
4. 새로운 소켓이 접속되면 현재 순번thread의 소켓 목록(스레드 풀)에 접속된 소켓 디스크립터 추가
   (접속될 때마다 쓰레드를 만들지 않고 미리 만들어놓은 쓰레드를 사용)

구현할 때 어려웠던 부분

- 아직 쓰레드가 익숙하지 않아 조건변수와 세마포를 통해 동기화 시키는 부분이 헷갈렸고, 미리 쓰레드를 만들어 놓고 클라이언트가 접속할 때마다 스레드를 새롭게 생성하여 부여하는 것이 아닌 기존의 쓰레드를 할당하고 접속이 종료되면 다시 쓰레드를 반환해 스레드 풀에 저장해 놓는 형태의 서버 구조를 이해하고 구현하는 것에 어려움을 겪었다.



```
#define SMALL_BUF 100
#define DELIM " "

#define MAX_THREAD_NUM 3

void request_handler(int *arg);
void send_data(FILE* fp,char* file_name,char* req_line,char* version);
void send_error(FILE* fp);
void error_handling(const char* message);
void* thread_func(void* arg);
void* mon_thread(void* data);

pthread_cond_t mycond[MAX_THREAD_NUM] = {PTHREAD_COND_INITIALIZER,};
pthread_mutex_t mutex[MAX_THREAD_NUM] = {PTHREAD_MUTEX_INITIALIZER, };
pthread_cond_t cond_main = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex_main = PTHREAD_MUTEX_INITIALIZER;

struct schedule{
    list<int> thread_list; // 쓰레드 풀
    list<int> clnt_list[MAX_THREAD_NUM]; // 접속한 클라이언트 소켓번호
}schedule;

int thread_idx = 0;

int main(int argc, char *argv[])
{
  int serv_sock, clnt_sock;
  struct sockaddr_in serv_adr, clnt_adr;
  socklen_t clnt_adr_size;
  pthread_t t_id;


  serv_sock=socket(PF_INET, SOCK_STREAM, 0);
  memset(&serv_adr, 0, sizeof(serv_adr));
  serv_adr.sin_family=AF_INET;
  serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
  serv_adr.sin_port = htons(atoi(argv[1]));
  if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1) {
      error_handling("bind error");
    }
  if(listen(serv_sock, 20)==-1) {
      error_handling("listen error");
    }

    pthread_create(&t_id,NULL,mon_thread,(void*)NULL); // 모니터링


    for(int i=0;i < MAX_THREAD_NUM;i++){
        pthread_create(&t_id,NULL,thread_func,(void*)&i); // (void*)&i or &i 로 정석대로 변수의 주소값을 보내면 쓰레드가 생성하기 전에 for문
        schedule.thread_list.push_back(i); // thread_list에 만들어진 쓰레드 번호를 넣는다.
        pthread_mutex_lock(&mutex_main);
        pthread_cond_wait(&cond_main,&mutex_main);
        pthread_mutex_unlock(&mutex_main);
    }
```

5. C 언어 싱글 쓰레드 다중 접속 서버 구현

지금까지 진행했던 멀티쓰레드 방식의 서버가 아닌 싱글쓰레드 기반으로 다중 접속이 가능하도록 SELECT, EPOLL을 사용해 서버를 구현 => 멀티플렉싱 서버

1. SELECT 사용 버전 구현
2. EPOLL + EDGE TRIGGER + NON-BLOCKING SOCKET 사용 버전 구현

조건

- 커넥션 관리는 다중화 기반임을 감안해 std::map 또는 std::unordered_map을 이용하여 socket descripter를 key, 현재 전송중인 파일의 이름을 value로 하여 전송중인 데이터 및 클라이언트의 상태를 추적할 수 있도록 구현

6. EPOLL + EDGE TRIGGER 서버 멀티 파일 전송

두 개의 파일을 입력해서 파일 이름 길이, 파일 이름, 파일 크기 , 파일 내용을 따로 보내도록 구현

파일 이름1 길이	int
파일 이름1	char*
파일 크기1	int
파일 내용1	char*
파일 이름2 길이	int
파일 이름2	char*
파일 크기2	int
파일 내용2	char*

조건

1. Client에서 두 개의 파일명을 입력받고 왼쪽과 같은 형태로 데이터를 나눠서 전송
2. 표의 첫번째 열은 읽기 상태를 나타낸다. (총 8개의 상태)
3. 정해진 크기만큼 읽어졌을 때 현재의 상태를 처리하고 다음 상태로 넘어간다.
4. 현재 상태에서 읽어야 할 만큼 값을 다 읽었을 경우 다음 상태로 넘어가고 최종 상태(파일 저장)까지 완료되었을 경우 첫번째 상태(파일 이름 크기)로 돌아가도록 구현

구현할 때 어려웠던 부분

- non-blocking mode에서 read()할 때는 한번에 지정한 크기만큼 전부 읽어진다는 보장이 없어서 클라이언트가 접속하면 그 때의 저장할 파일명, 읽기 상태, 현재까지 읽은 바이트 수를 std::map에 저장해 상태를 관리하도록 할 때 아직도 포인터와 주소값이 완벽하게 숙달되지 않아서 포인터 문제로 어려움을 겪었다.

- 클라이언트가 여러 번 데이터를 나눠서 보내거나 끊어져서 전달되도 문제없이 서버에서 데이터를 읽어오도록 하는 부분이 이해하는데 어려움이 있었고, TCP 입출력 버퍼를 정확히 이해해야만 구현할 수 있는 프로그램이라 많이 헤맸다.

```
76    epfd = epoll_create(EPOLL_SIZE);
77    ep_events = (struct epoll_event*)malloc(sizeof(struct epoll_event) * EPOLL_SIZE);
78
79    setnonblockingmode(serv_sock); // 넌블로킹으로 설정함
80    event.events = EPOLLIN; // 수신할 데이터가 존재하는 상황
81    event.data.fd = serv_sock;
82    epoll_ctl(epfd,EPOLL_CTL_ADD,serv_sock,&event);
83
84    while(1){
85      event_cnt = epoll_wait(epfd,ep_events, EPOLL_SIZE, -1);
86      if(event_cnt == -1){puts("epoll_wait error!");break;}
87
88      printf("event 발생\n");
89      for(i = 0; i < event_cnt; i++){
90        if(ep_events[i].data.fd == serv_sock)
91        {
92          clnt_adr_sz = sizeof(clnt_adr);
93          clnt_sock = accept(serv_sock,(struct sockaddr*)&clnt_adr, &clnt_adr_sz);
94          setnonblockingmode(clnt_sock);
95          event.events = EPOLLIN | EPOLLET; // 수신할 데이터가 존재하는 상황 , 이벤트의 감지를 엣지트리거 방식으로 설정
96          event.data.fd = clnt_sock;
97          epoll_ctl(epfd,EPOLL_CTL_ADD,clnt_sock,&event);
98
99          m[clnt_sock];
100         printf("connected client : %d\n", clnt_sock);
101         printf("map : ");
102         for(pair<int,struct FileReadData> atom : m){
103           printf("%d ",atom.first);
104         }
105         printf("\n");
106       }
107       else
108       {
109         struct FileReadData& fileData = m[ep_events[i].data.fd];
110         long read_len;
111         while(1){
112           read_len = read(ep_events[i].data.fd, fileData.read_curr_ptr+fileData.read_len_curr, fileData.goal_read
113
114           if(read_len == 0){
115             epoll_ctl(epfd,EPOLL_CTL_DEL,ep_events[i].data.fd,NULL);
116             close(ep_events[i].data.fd);
117             m.erase(ep_events[i].data.fd);
118             printf("closed client : %d\n",ep_events[i].data.fd);
119             break;
120           }
121           else if(read_len < 0)
```

7. 자료구조 및 알고리즘 직접 구현

- **double linked list (std::list) 구현**
  - 연산자 오버로딩 사용
  - template 사용
  - 추가로 iterator 기능 구현

- **이진 탐색 트리 (std::map) 구현**

- **해시 테이블 (std::unordered_map) 구현**
  - 동적 슬롯(체이닝 기법)으로 충돌 처리 (버킷마다 단일 연결 리스트를 연결)

8. DB와 연계된 프로그래밍 환경 구축

- **개발 환경 구축**

1. 리눅스 환경에서 PostgreSQL 설치
2. 설치 후 PostgreSQL 명령어 학습
3. 학습한 명령어로 DB 사용자 생성, DB 클러스터 생성, DB 테이블 생성 등 수행

- **개발 환경 구축 시 어려웠던 부분**

  생각보다 DB 환경 구축이 경로 설정이나 버전 때문에 고쳐야 하는 부분이 많았어서 어려움을 겪었다.

8-1. DB와 연계된 대화형 프로그램 구현

구현 요구사항

1. 프로그램 최초 실행시 ID와 PW를 받는 프롬프트 출력
2. 터미널 설정을 통해 비밀번호 입력시에는 화면에 안 뜨게 할 것 (termios.h 라이브러리 사용)
3. 입력받은 ID, PW를 DB에 있는 유저 데이터와 대조하여 일치하면 메인메뉴로 이동
4. 메인 메뉴에서는 메뉴 테이블에 있는 ID번호 or alias 명령어를 이용하여 해당 메뉴 id에 해당하는 게시물을 조회할 수 있는 화면으로 이동
5. 게시물 조회 하면에서는 새 게시물 작성, 삭제, 페이지 이동, 메뉴로 이동, 다른 게시판으로 이동 등의 명령어 구현
6. 임의의 메뉴에서 0을 입력하면 로그인이 풀리면서 ID, PW 입력 창으로 이동
7. 프로그램 종료 : SIGINT에 대한 시그널 핸들러 구현

8-1. DB와 연계된 대화형 프로그램 구현

메인 메뉴 화면 예시)

```
사용자 : 김덕용
===== 메뉴 =====

0. 로그아웃
1. 공지/notice
2. 게시판/bbs
3. 토론방/agora

번호 혹은 go alias >  0
로그아웃 되었습니다.
=====아무 키나 누르세요=====

사용자 : 김동길
===== 메뉴 =====

0. 로그아웃
1. 공지/notice
2. 게시판/bbs
3. 토론방/agora

번호 혹은 go alias >  2
(메뉴 번호를 2로 하여 게시물 목록 화면 상태로 이동)
```

게시물 목록 화면 예시)

```
사용자 : (member_name)
===== (menu_name) =====

        5 yhhong  김덕용  테스트   2021-10-28 14:00
        4 yhhong  김덕용  테스트   2021-10-28 14:00
        3 yhhong  김덕용  테스트   2021-10-28 14:00
        2 yhhong  김덕용  테스트   2021-10-28 14:00
        1 yhhong  김덕용  테스트   2021-10-28 14:00

=====

0: 로그아웃, C (본문) : 작성, D (번호): 삭제, M (메뉴로), N (다음 페이지), P (이전 페이지), go alias
> C 테스트2

사용자 : (member_name)
===== (menu_name) =====

        6 yhhong  김덕용  테스트2  2021-10-28 15:00
        5 yhhong  김덕용  테스트   2021-10-28 14:00
        4 yhhong  김덕용  테스트   2021-10-28 14:00
        3 yhhong  김덕용  테스트   2021-10-28 14:00
        2 yhhong  김덕용  테스트   2021-10-28 14:00
        1 yhhong  김덕용  테스트   2021-10-28 14:00

=====

0: 로그아웃, C (본문) : 작성, D (번호): 삭제, M (메뉴로), N (다음 페이지), P (이전 페이지), go alias
> D 4
```

8-1. DB와 연계된 대화형 프로그램 구현

구현할 때 어려웠던 부분

- C언어를 사용해 PostgreSQL DB를 연계해서 대화형 프로그램을 작성하려면 C libpq 라이브러리를 사용해야 한다. 그렇기 때문에 libpq에서 제공하는 함수들의 기능을 학습하여 사용하는 것이 시간이 좀 걸렸고, 데이터를 가져올 때 쓰는 SQL 문법들을 사용해 데이터를 프로그램에 뿌려주는 부분에서 sprintf() 를 사용해서 생기는 데이터의 범위를 침범하는 오류 때문에 한참을 헤맸었지만, snprintf() 를 사용해 안정적으로 데이터를 저장하는 법을 터득하고 오류를 수정했던 경험이 가장 기억에 남는다.

```c
void go_bbs(PGresult** menu_name_res, PGconn** conn,char* member_name,char* member_id,int number,int* check){
  int curr_page = 0;
  int tuple = 0;
  int column = 0;
  char input[20] = {0,};
  char query[200] = {0,};

  memset(query,0,sizeof(query));
  snprintf(query,sizeof(query),"select menu_name from tbl_menu where menu_idx=%d",number);
  *menu_name_res = PQexec(*conn,query);

  if(PQresultStatus(*menu_name_res) != PGRES_TUPLES_OK){
    fprintf(stderr,"menu_name_res No data not found\n");
    PQclear(*menu_name_res);
    PQfinish(*conn);
    exit(-1);
  }

  while(true){ // 게시판
    system("clear");
    printf("사용자 : %s\n",member_name);
    printf("===== %s =====\n\n ",PQgetvalue(*menu_name_res,0,0));

    memset(query,0,sizeof(query));
    snprintf(query,sizeof(query),"select article_idx,member_id,member_name,title,write_date from tbl_article where
    PGresult* select_article_res = PQexec(*conn,query);

    tuple = PQntuples(select_article_res);
    column = PQnfields(select_article_res);
    for(int i = 0;i < tuple; i++){
      for(int j = 0; j < column; j++){
        printf("%s   ",PQgetvalue(select_article_res,i,j));
      }
      printf("\n");
    }
    PQclear(select_article_res);


    printf("\n=====\n\n");
    printf("0: 로그아웃, C (본문) : 작성, D (번호) : 삭제, M(메뉴로), N (다음 페이지), P(이전 페이지), go alias\n> ");
    fgets(input,sizeof(input),stdin);
    input[strlen(input) - 1] = '\0';

    char* first_input;
```

9. C, C++로 클래스 자료형 직접 구현

조건

1. C++ 언어에 대한 기본을 좀 더 다지기 위해 C++을 사용해 클래스 정의, 연산자 오버로딩, 객체 메모리 관리 및 생명주기 등에 대해 다시 한 번 학습하며 구현했다.

2. C언어를 사용해 클래스를 구현한다. 단, C는 연산자 오버로딩을 제공하지 않으므로 클래스의 각 연산에 해당하는 함수 및 구조체를 만들어 구현한다.

9. C, C++로 클래스 자료형 직접 구현

구현한 연산들

C++ 클래스		
String()	멤버 문자열은 1byte char *로 할당 후 ₩0 넣을 것 (매개변수가 없으므로 비어있는 문자열)	
String(const char *)	초기화하면서 받은 것과 똑같은 문자열로 생성	
virtual ~String()	소멸자, 멤버 문자열 해제할 것(추후 상속에 대응하기 위해 virtual로 선언)	
String(const String&)	초기화하면서 b객체와 똑같은 문자열을 생성	Deep copy구현할 것
a + b	앞 객체의 문자열과 뒤 객체의 문자열을 붙인 새로운 객체 반환	
a += b	앞 객체의 문자열에 뒤 객체의 문자열을 덧붙일 것	
a = b	a객체의 기존 char *를 해제하고 b객체와 똑같은 문자열을 만들어 복사할 것	Deep copy구현할 것
a[1]	char *의 배열의 해당 위치 char값 리턴	위치가 잘못되었을 경우 throw "invalid";
a == b	strcmp로 a객체의 문자열과 b객체의 문자열을 비교해서 0이면 true 0이 아니면 false	
a != b	a == b와 반대 결과	
a <= b	strcmp로 a객체의 문자열과 b객체의 문자열을 비교해서 0보다 작거나 같으면 true 아니면 false	
a < b	strcmp로 a객체의 문자열과 b객체의 문자열을 비교해서 0보다 작으면 true 아니면 false	
a >= b	strcmp로 a객체의 문자열과 b객체의 문자열을 비교해서 0보다 크거나 같으면 true 아니면 false	
a > b	strcmp로 a객체의 문자열과 b객체의 문자열을 비교해서 0보다 크면 true 아니면 false	
int size()	strlen 호출해서 현재 객체의 문자열 길이 반환	
char *contents()	멤버 char *를 반환	
int find(b)	strstr 함수를 호출해서 b 문자열의 내용이 a 문자열에 포함되었는지 확인하고 포함되었으면 포함된 위치 반환	
String *substr(1, 3)	입력받은 시작 위치부터 끝 위치까지 자른 새로운 문자열을 반환할 것 시작 위치가 0보다 작을 경우 0으로 조정 끝 위치가 현재 문자열 길이보다 클 경우 문자열 길이 -1로 조정	
멤버변수 : char *str;	문자열 길이에 맞게 동적 할당, 대입연산자, 복사생성자, += 등 연산자 호출시 기존것 해제하고 새로운 문자열에 맞게 생성할 것	
C 클래스		
typedef struct S_String { char *str; } String_S;	클래스 구조체 선언	
String_S *copyString(String_S *b)	초기화하면서 받은 것과 똑같은 문자열로 생성해서 해당 char*를 반환, 받은 문자열이 NULL일 경우 1byte로 할당 후 ₩0 넣은 것을 반환	
String_S *newString(const char *b)	초기화하면서 받은 것과 똑같은 문자열로 생성해서 해당 char*를 반환, 받은 문자열이 NULL일 경우 1byte로 할당 후 ₩0 넣은 것을 반환	
void deleteString(String_S *this)	free로 받은 문자열을 해제 if (this->str) free(this->str); free(this);	
int assignString(String_S *this, String_S *b)	this객체의 기존 char *를 해제하고 b객체와 똑같은 문자열을 만들어 복사할 것 성공하면 0 실패하면 -1 반환	
int concatString(String_S *this, String_S *b)	this 객체의 문자열에 b 객체의 문자열을 덧붙일 것	
char charAt(String_S *this, int b)	this 객체의 문자열의 i번째 위치 문자 반환	
int isStringSame(String_S *this, String_S *b)	strcmp(this->str, b->str) 이 0이면 1, 0이 아니면 0 반환	
int isStringDifferent(String_S *this, String_S *b)	strcmp(this->str, b->str) 이 0이면 0, 0이 아니면 1 반환	
int compareString(String_S *this, String_S *b)	strcmp(this->str, b->str) 의 값을 그대로 반환	
String_S *concatNewString(String_S *this, String_S *b)	this->str에 b->str를 덧붙여서 생성한 새로운 객체 반환	
String_S *subString(String_S *this, int begin, int end)	입력받은 시작 위치부터 끝 위치까지 자른 새로운 문자열을 반환할 것 시작 위치가 0보다 작을 경우 0으로 조정 끝 위치가 현재 문자열 길이보다 클 경우 문자열 길이 -1로 조정	
int findString(String_S *this, String_S *b)	l = strstr(this->str, b->str) 호출 후 NULL이 아니면 l - this->str 반환	

string.cpp

C++_Project > 클래스작성과제 > string.cpp > ···

```cpp
#include<iostream>
using namespace std;

const char* malloc_error = "malloc_error";

class String{
private:
  char* str;
public:
  String();
  String(const char* b);
  String(const String& obj);
  virtual ~String();
  int find(const char* b);
  String substr(int begin, int end) const;


  int size() const{
    return strlen(str);
  }
  char* contents() const {
    return str;
  }

  char operator[] (int idx);
  String& operator= (const String& obj);
  String operator+ (const String& obj) const;
  String& operator+= (const String& obj);

  bool operator== (const String& obj) const;
  bool operator!= (const String& obj) const;
  bool operator<= (const String& obj) const;
  bool operator>= (const String& obj) const;
  bool operator< (const String& obj) const;
  bool operator> (const String& obj) const;

};

bool String::operator< (const String& obj) const{
  if(strcmp(this->contents(),obj.contents()) < 0){
    return true;
  }
  return false;
}

bool String::operator> (const String& obj) const{
  if(strcmp(this->contents(),obj.contents()) > 0){
```

string.c

C++_Project > 클래스작성과제 > string.c > concatNewString(String_S * this, String_S *

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct S_String{
  struct S_String* this;
  char* str;

  char* (*getStr)(struct S_String* this);

  int (*findString)(struct S_String* this, struct S_String* b);
  int (*assignString)(struct S_String* this, struct S_String* b);
  int (*concatString)(struct S_String* this, struct S_String* b);
  char (*charAt)(struct S_String* this, int b);
  int (*isStringSame)(struct S_String* this, struct S_String* b);
  int (*isStringDifferent)(struct S_String* this, struct S_String* b);
  int (*compareString) (struct S_String* this, struct S_String* b);

}String_S;

//생성자,소멸자
String_S* newString(const char* b);  //문자열 입력 생성자
String_S* copyString(String_S* b);   // 객체 복사 생성자
String_S* concatNewString(String_S* this,String_S* b);  // this뒤에 b를 붙
String_S* subString(String_S* this,int begin, int end); //시작위치부터 끝
void deleteString(String_S* this);

//함수
int assignString(String_S* this,String_S* b);
int concatString(String_S* this,String_S* b);
char charAt(String_S* this, int b);
int isStringSame(String_S* this, String_S* b);
int isStringDifferent(String_S* this,String_S* b);
int compareString(String_S* this, String_S* b);
int findString(String_S* this, String_S* b);

char* getStr(String_S* this); //현재 str값 반환


int main(){
  /* newString, concatString, assignString, deleteString */
  String_S* a = newString("hello");
  String_S* b = newString("banana");
  a->concatString(a,b);
  printf("바꾸기 전 a : %s\n",a->getStr(a));

  if(a->assignString(a,b) == 0){
```

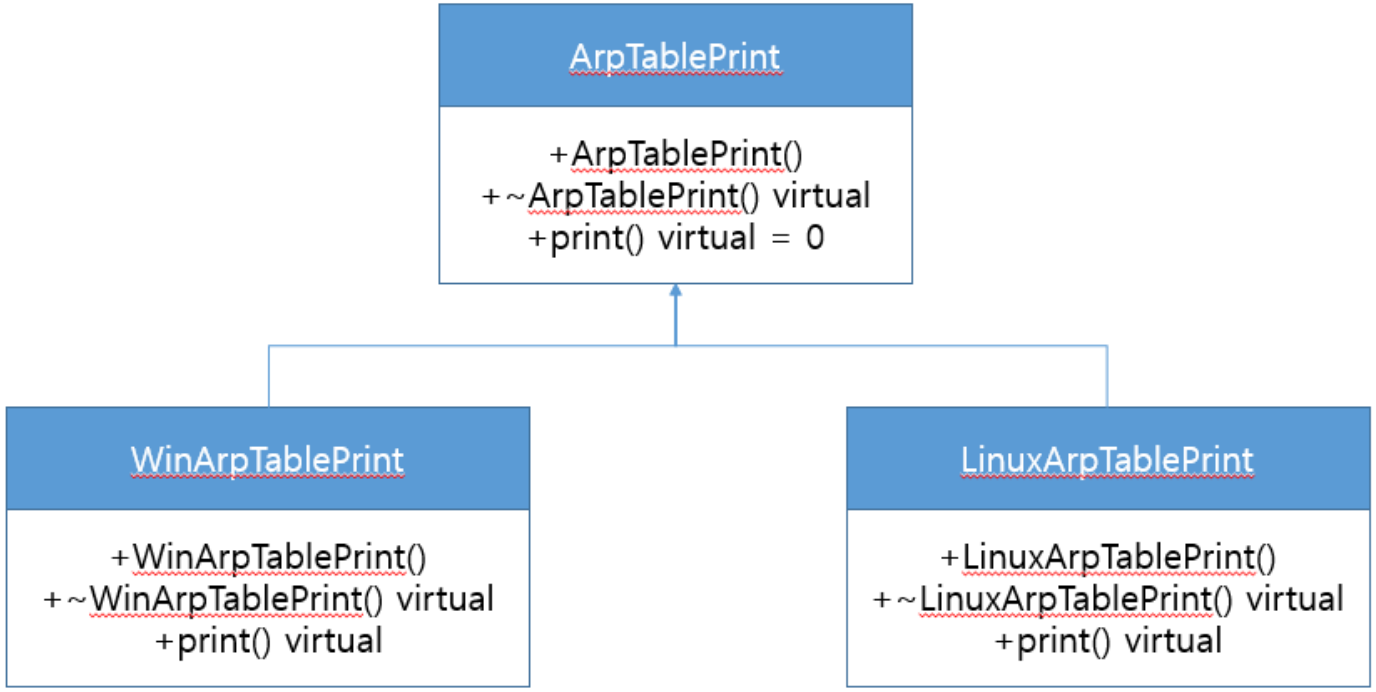
10. 다형성을 이용한 크로스플랫폼 구현

조건

1. Windows 및 Linux 시스템의 ARP 테이블 수집해서 화면에 테이블 정보 출력하는 프로그램
2. 컴파일시 파일 선택을 통해 소스코드 변경 없이 Windows, Linux 양쪽 모두에서 컴파일 가능하도록 구현
3. C에서는 구조체와 함수포인터로 구현
4. C++에서는 상속, 가상함수, 추상클래스 개념을 이용

10. 다형성을 이용한 크로스플랫폼 구현

클래스 구성도



10. 다형성을 이용한 크로스플랫폼 구현

구현하면서 어려웠던 부분

- windows에서의 변수 타입과 사용 함수 등 Linux와 다른 부분들이 있어서 Linux 와 Windows 각각 따로 공부해서 구현해야 하는 점이 다소 어렵게 느껴졌다.

- 하나의 헤더파일에 모든 코드를 넣지 않고 나눠서 코드를 관리하기 편하도록 구현하는 구조 자체를 이해하는데 어려움이 있었다.

11-1. Packet 분석 및 저장 (1)

지금까지 습득한 모든 지식들이 사용되었고, **구현하면서 가장 어려운 업무였다**

구현 조건

1. 프로세스 두 개를 생성 (pcapwriter, pcapanalyzer)
2. pcapwriter 프로세스는 pcap라이브러리를 이용해 패킷을 수집하고 system v shared memory를 통해 pcapanalyzer에 전달한 후 1분 단위로 pcap 파일에 저장한다.
3. pcapanalyzer 프로세스는 pshared mutex, pshared cond를 이용해서 signal을 받았을 시 shared memory에 접근해서 pcap 헤더 정보를 화면에 출력한다.

11-2. Packet 분석 및 저장 (2)

1. 캡처한 네트워크 패킷을 판별한 후 아래와 같이 패킷의 데이터를 분석해 정보들을 출력

```
===== PCAP =====
  Captured Time    : %d.%06d
  Captured Length :
  Length             :

===== Ethernet =====
  Source MAC       :
  Destination MAC :
  EtherType          :
===== IpV4 =====
  Source Addr       :
  Destination Addr :
  Header Length    : ip_header_length * 4
  Ip Data Length   : ip_total_length
===== Tcp/Udp =====
  Source Port        :
  Destination Port   :
  Tcp Data Length  :
```

11-3. 패킷의 정보가 담긴 PCAP파일 관리 기능 추가(3)

1. pcap writer 프로세스에 **pcap 파일을 관리하는 thread** 추가
2. 프로그램 시작시 pcap 파일 저장 경로를 절대경로로 설정하고 해당 경로에 pcap 파일을
   저장하도록 기능 추가
3. **main thread**는 해당 경로에 pcap 파일명 규칙을 아래와 같이 설정해 저장한다.
   - > 연월일_시분초_마이크로초_순번.pcap
   - > 순번은 01로 시작해서 같은 시분초에 파일이 겹칠 경우 순번을 증가시킨다.

4. 파일 관리 thread 시작시 모든 pcap 파일의 목록을 std::list에 저장하고 시간 순으로 정렬
5. **main thread**는 새로운 pcap 파일을 저장할 때마다 **파일 관리 thread**에 pthread signal과 함께 파일 이름을 전달
6. **파일 관리 thread**는 mutex 안에서 wait을 걸고 있다가 signal을 받으면 list를 통해 전달받은
   파일명을 pcap 파일 list의 맨 뒷부분에 넣고 unlock 하도록 한다.

11-4. 패킷 분석 내용 DB 테이블에 저장(4)

- 패킷을 분석해 문자열 형태로 추출해낸 데이터들을 libpq 라이브러리를 이용해 PostgreSQL을 연결하고 추출한 데이터들을 그대로 테이블로 생성하여 컬럼별로 DB에 저장한다.

```
===== PCAP =====
  Captured Time    : %d.%06d
  Captured Length :
  Length           :

===== Ethernet =====
  Source MAC       :
  Destination MAC :
  EtherType        :
===== IpV4 =====
  Source Addr      :
  Destination Addr :
  Header Length    : ip_header_length
  Ip Data Length   : ip_total_length
===== Tcp/Udp =====
  Source Port      :
  Destination Port   :
  Tcp Data Length  :
```

- 분석한 데이터 형태

```
dkyongdb=# select * from packet;
   captured_time    | captured_length | length |    source_mac     | destination_mac   | ethertype |  source_addr   | destination_addr | header_length | ip_data_length | source_port | destination_port | tcp_data_length
--------------------+-----------------+--------+-------------------+-------------------+-----------+----------------+------------------+---------------+----------------+-------------+------------------+-----------------
 1640824191.499997 |              54 |     54 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 20.43.132.130    |            20 |          10240 |       37732 |              443 |               0
 1640824191.500305 |              60 |     60 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 20.43.132.130  | 10.0.2.15        |            20 |          10240 |         443 |            37732 |               6
 1640824205.890361 |              85 |     85 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 168.126.63.1     |            20 |          18176 |       42605 |               53 |              31
 1640824205.890385 |              85 |     85 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 168.126.63.1     |            20 |          18176 |       42605 |               53 |              31
 1640824205.892889 |             215 |    215 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 168.126.63.1   | 10.0.2.15        |            20 |          51456 |          53 |            42605 |             161
 1640824205.892974 |             152 |    152 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 168.126.63.1   | 10.0.2.15        |            20 |          35328 |          53 |            42605 |              98
 1640824205.893321 |              74 |     74 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          15360 |       48060 |              443 |              20
 1640824205.894485 |              60 |     60 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 40.77.226.250  | 10.0.2.15        |            20 |          11264 |         443 |            48060 |               6
 1640824205.894552 |              54 |     54 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          10240 |       48060 |              443 |               0
 1640824205.894850 |             571 |    571 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          11522 |       48060 |              443 |             517
 1640824205.895203 |              60 |     60 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 40.77.226.250  | 10.0.2.15        |            20 |          10240 |         443 |            48060 |               6
 1640824206.351785 |            2974 |   2974 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 40.77.226.250  | 10.0.2.15        |            20 |          36875 |         443 |            48060 |            2920
 1640824206.351874 |              54 |     54 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          10240 |       48060 |              443 |               0
 1640824206.352194 |            1029 |   1029 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 40.77.226.250  | 10.0.2.15        |            20 |          63235 |         443 |            48060 |             975
 1640824206.352216 |              54 |     54 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          10240 |       48060 |              443 |               0
 1640824206.352880 |             147 |    147 | 08:00:27:77:a7:52 | 52:54:00:12:35:02 |       800 | 10.0.2.15      | 40.77.226.250    |            20 |          34048 |       48060 |              443 |              93
 1640824206.353173 |              60 |     60 | 52:54:00:12:35:02 | 08:00:27:77:a7:52 |       800 | 40.77.226.250  | 10.0.2.15        |            20 |          10240 |         443 |            48060 |               6
(17개 행)

dkyongdb=#
```

- 네트워크 패킷을 캡처할 때마다 패킷을 분석해서 DB 테이블에 저장

12. 패킷의 개별 TCP FLOW 관리

1. 패킷의 개별 TCP 세션들을 관리하기 위해 std::unordered_map을 사용자화 한 타입 사용
   -> 일반적인 std::unordered_map이 아닌 뒤에 3번째, 4번째 인자인 **hasher와 comparer을 추가로 구현**하여 TCP 세션을 관리하기 위함이다.

2. 헤더 파싱 결과 IPv4이면서 TCP 패킷인 경우 map에 **TCP Flow를 생성해서 넣는다.**
   -> main thread에서는 패킷 헤더 파싱만 하고, 데이터를 분석 및 처리하여 db에 넣는 부분은 따로 작업용 thread를 생성해서 처리하도록 한다. (성능 향상을 위해)

3. 기존에 생성된 Flow가 있을 경우 기존 Flow를 map에서 가져와 사용한다.

4. 현재 받은 **패킷의 TCP FLAG를 분석**하여 어느 한 방향에서 RST FLAG가 들어오거나
   양방향 모두에서 FIN FLAG가 들어오면 TCP 통신을 하고 있던 패킷들이 서로 연결을 종료
   했다고 판단하여 해당 TCP Flow를 map에서 삭제한다.

감사합니다.