FSSN 프로젝트 결과물 보고서

2017104018 컴퓨터공학과 이학주

1. 프로젝트 개요

A. 풀스택 서비스 네트워킹 수업에 있는 소켓, ZeroMQ를 강의 자료에 있는 Python이 아닌 C++ 언어를 사용하여 작성한다.

B. 개발 및 실행 환경: Macbook pro M1, Monterey

C. 언어 작성 환경: visual studio for MAC

2. 작성 내용 소개

A. 소켓

```
struct sockaddr_in server_addr;
socklen_t size;

server = socket(AF_INET, SOCK_STREAM, 0);

if (server < 0)
    exit(1);

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htons(INADDR_ANY);
server_addr.sin_port = htons(portNum);


bind(server, (struct sockaddr*)&server_addr,sizeof(server_addr));
```

대부분의 소켓 초기 세팅은 다음과 같이 설정하였다.

TCP 통신의 경우, socket 생성자의 두 번째 인자에 SOCK_STREAM,

UDP 통신의 경우 SOCK_DGRAM 으로 설정한다.

(lec-03-prg-01-tcp-echo-server.cpp)

수신, 발신 방식 및 문자열 비교

```
do{
    cout << "received: ";
    recv(client, buffer, bufsize, 0);
    cout << buffer << endl;
    if(strcmp(buffer, "quit") == 0){
        isExit = true;
        break;
    }
    send(client, buffer, bufsize, 0);

} while(!isExit);
```

- 통신을 할 때 char[1024] buffer와 함께 전달하면서 receive 한 데이터를 버퍼에 담고, 출력하고, 그 값의 조건에 따라 수행을 해야하는 경우, strcmp를 사용하였다. (lec-03-prg-01-tcp-echo-server.cpp)

1:N 통신 -> Server에서 Thread를 사용하지 않을 때

```
while (1)
{
    while(1){
        cout << "received: ";
        recv(client, buffer, bufsize, 0);
        cout << buffer << endl;
        if(strcmp(buffer, "quit") == 0){
            clientNum--;
            break;
        }
        send(client, buffer, bufsize, 0);
    }
    client = accept(server,(struct sockaddr *)&server_addr,&size);
    if(client >0){
        cout << "Client connected by IP address 127.0.0.1 with Port number ";
        std::cout<< ntohs(server_addr.sin_port)<<std::endl;
    } else {
        isExit = true;
    }
    if(isExit)
        close(client);
}
cout << "echo-server is de-activated" <<endl;
```

동시에 다수와 통신을 할 수 없으며, 연결 중인 하나의 클라이언트가 나가면 새로운 클라이언트를 연결할 수 있도록 되어있다.
(lec-03-prg-05-tcp-echo-server-socketserver.cpp)

1:N 통신 -> Server에서 Thread를 사용할 때

```
while(true){
    csock = (int*)malloc(sizeof(int));
    if((*csock = accept( server, (sockaddr*)&client_addr, &addr_size))!= -1){
        std::cout<< "client connected by IP address 127.0.0.1 with Port number ";
        std::cout<< ntohs(client_addr.sin_port)<<std::endl;
        threadNum++;
        pthread_create(&thread_id, 0, &SocketHandler, (void*)csock ); // 쓰레드 활성화
        pthread_detach(thread_id);
    }
    else{
        fprintf(stderr, "Error accepting %d\n", errno);
    }
}
```

Main 함수에서는 새로운 클라이언트가 연결되는 것을 기다리며, 연결 성공 시 Thread를 생성하여 다음의 함수를 작동시킨다.
(lec-03-prg-06-tcp-echo-server-multithread.cpp)

```
void* SocketHandler(void* lp){
    int *server = (int*)lp;
    std::thread::id this_id = std::this_thread::get_id();
    char buffer[1024];
    int bufSize = 1024;
    bool isExit = false;

    do{
        recv(*server, buffer, bufSize, 0);
        if(strcmp(buffer, "quit") == 0){
            threadNum--;
            std::cout<<"active threads are remained: " << threadNum << std::endl;
            //연결된 client 없을 때 -> y: 프로그램 종료, n: 대기
            if(threadNum == 0){
                char Yn;
                std::cout<< "Do you want to exit program? (y/n) ";
                std::cin>>Yn;
                if (Yn == 'y'){
                    exit(1);
                }
            }
            isExit = true;
            goto FINISH;
        } else{
            std::cout<<"echoed: "<<buffer << "  By Thread-" << this_id <<std::endl;
        }
        send(*server, buffer, bufSize, 0);
    } while(!isExit);
    free(server);
FINISH:
    free(server);
    return 0;
}
```

(lec-03-prg-06-tcp-echo-server-multithread.cpp)
이로써 Server는 다수의 Client와 통신할 수 있다.
이곳에서 간단한 추가 기능을 구현하였다.

- 더 이상 클라이언트가 존재하지 않을 때, 프로그램을 종료할 지의 여부를 결정하는 질문을 받아서 사용자가 입력에 따라 클라이언트의 추가 요청을 기다릴지, 프로그램을 종료할 지를 결정한다.

Client에서 Receive를 담당하는 쓰레드 생성

```
int *csock = int *csock
if(connect(*csock,(struct sockaddr *)&server_addr, sizeof(server_addr)) != -1){
    pthread_create(&thread_id,0, &ReceiveHandler, (void*)csock);
    pthread_detach(thread_id);
}
```

Server에서 send를 받지 못하면, 그대로 기다리기만 하는 것을 반복한다.
Thread를 사용하여 Receive를 담당하는 함수를 생성하였다.

```
void* ReceiveHandler(void* lp){
    int *csock = (int*)lp;
    char buffer[1024];
    int bufSize = 1024;
    int bytecount;
    bool isExit = false;

    do{
        recv(*csock, buffer, bufSize, 0);
        if(strcmp(buffer, "") != 0 ){
            std::cout<<"> received: "<< buffer << std::endl;
        }
        buffer[0] = '\0'; //버퍼 비우기
        if(csock == NULL){
            isExit = true;
            break;
        }
    } while(!isExit);
    isExit = false;
}
```

Connect가 되었을 때 쓰레드를 생성하여 Receive를 담당하는 Handler를 생성한다.
(lec-03-prg-07-tcp-echo-client-multithread.cpp)

위의 내용을 확장시켜 1:N이 아닌, M:N 통신을 구축하였다.

```
void* SocketHandler(void* lp){
    int *server = (int*)lp;
    group_queue.push_back(*server);
    std::thread::id this_id = std::this_thread::get_id();
    char buffer[1024];
    int bufSize = 1024;
    int bytecount;
    bool isExit = false;
    do{

        recv(*server, buffer, bufSize, 0);

        if(strcmp(buffer, "quit") == 0){
            threadNum--;
            isExit = true;
            goto FINISH;
        } else{
            std::cout<<"received ( "<< buffer << " ) and echoed to " << threadNum << " clients"<<std::endl;
        }
        for(int i=0, end = group_queue.size(); i<end; ++i){
            send(group_queue[i], buffer, bufSize, 0);
        }
    } while(!isExit);
    // 큐 안에 클라이언트 데이터 삭제 및 threadNum 감소
    for(int i=0, end = group_queue.size(); i<end; ++i){
        if(group_queue[i] == *server){
            group_queue.erase(group_queue.begin() + i);
        }
    }
    free(server);
FINISH:
    free(server);
    return 0;
}
```

(lec-03-prg-08-tcp-echo-server-multithread-chat.cpp)

새로운 connection이 발생하면 thread가 생성되고, queue에 client들의 정보를 가지고 있도록 넣은 후, send 할 때 queue에 있는 모든 클라이언트에게 send하여 각 클라이언트가 보낸 메시지를 다른 클라이언트에서도 볼 수 있도록 한다.

group_queue는 전역 변수 vector로 선언하여 소켓 데이터를 추가해주고 관리하도록 하였다.

UDP 통신을 적용하여 클라이언트의 등록과, 해지. 등록을 성공한 클라이언트에 한하여 메시지를 전달하고 받을 수 있는 통신을 적용하였다.

```
if(strcmp(buffer, "#REG") == 0) { // 등록
    found = false;
    for(int i=0, end = group_queue.size(); i<end; ++i){ // 포트 번호 비교를 통해 데이터 비교
        if(group_queue[i].sin_port == client_addr.sin_port)
            found = true;
    }
    if (!found){
        group_queue.push_back(client_addr); //등록
        std::cout<<"> client registered ('127.0.0.1', " <<  ntohs(client_addr.sin_port) << ")\n";
    }
} else if (strcmp(buffer, "#DEREG") ==0 || strcmp(buffer, "quit") == 0){
    found = false;
    for(int i=0, end = group_queue.size(); i<end; ++i){ // 삭제
        if(group_queue[i].sin_port == client_addr.sin_port){
            found = true;
            group_queue.erase(group_queue.begin() + i);
        }
    }
    if (found){
        std::cout<<"> client de-registered ('127.0.0.1', " <<  ntohs(client_addr.sin_port) << ")\n";
    }
```

서버가 클라이언트에게 “#REG” 메시지를 받으면, group queue에 존재하지 않을 시, 등록해 주는 과정. “#DEREG”, “quit”메시지를 받으면 group_queue에서 삭제하지 않도록 한다.
(lec-03-prg-11-udp-echo-server-socketserver-chat.cpp)

```
} else{
    found = false;
    for(int i=0, end = group_queue.size(); i<end; ++i){
        if(group_queue[i].sin_port == client_addr.sin_port){
            found = true;
        }
    }
    if(group_queue.size() == 0){
        std::cout<<"> no clients to echo\n";
        continue;
    }

    if (found){ // 전체 보내기
        for(int i=0, end = group_queue.size(); i<end; ++i){
            sendto(server, buffer, bufSize, 0, (struct sockaddr *)&group_queue[i], sizeof(group_queue[i]));
        }
        std::cout<<"> received ( "<< buffer<<" ) and echoed to "<< group_queue.size() << " clients \n";
    } else {
        std::cout<<"> ignores a message from un-registered client\n";
    }
}
```

그 이외의 메시지를 받으면, 등록되어 있는 경우에만 메시지를 모든 클라이언트에게 전달하는 과정을 구축하였다.
(lec-03-prg-11-udp-echo-server-socketserver-chat.cpp)

B. ZeroMQ

ZeroMQ의 대부분은 Python 코드를 함께 기입하며 C++에 알맞게 코드를 수정하여 크게 변화하는 부분이 없다. Python 코드를 참조하며, C++에서는 다르게 적용됐던 부분들 중 주요 사항에 대해 명세 한다.

ZeroMQ를 구축하는 과정에서는 기본적으로 선언부에 공통점이 있다.

```
zmq::context_t context;
zmq::socket_t server(context, zmq::socket_type::rep);
zmq::message_t request;
```

zmq::context_t 타입을 선언하고,
zmq::socket_t 타입을 선언할 때 생성자로 context와 socket type을 함께 기입한다.

```
le(1){
 server.recv(&request);
 std::string rqst = std::string(static_cast<char*>
     (request.data()), request.size());

 std::cout << "Received request: [" << std::string(static_cast<char*>
         (request.data()), request.size())  << "]" << std::endl;
```

String을 사용하는 과정에서 static_cast를 통해 타입 변환을 수행하여 출력할 수 있도록 하였다.

```
  zmq::message_t reply{input.length()};
  memcpy(reply.data(), input.data(), input.length());
  server.send(reply);
```

string 데이터에 전송할 데이터를 변경해준 후, memcpy를 통해서 zmq::message_t 타입에 복사하였고, 메시지를 send 하였다.

String message를 나누기 위해 따로 split 함수를 생성하여 나누어주었다.

```
vector<string> split (string input, char delimiter){
    vector<string> answer;
    stringstream ss(input);
    string temp;

    while(getline(ss, temp, delimiter)) {
        answer.push_back(temp);
    }
    return answer;
}
```

main(int, char * []) (function)

Polling 감지

```
zmq::context_t context;
string clientID(argv[1]);

zmq::socket_t subscriber(context, zmq::socket_type::sub);
subscriber.setsockopt(ZMQ_SUBSCRIBE, "", 0);
subscriber.connect("tcp://127.0.0.1:5557");

zmq::socket_t pusher(context, zmq::socket_type::push);
pusher.connect("tcp://127.0.0.1:5558");

int random;
string input;
zmq::message_t message;
zmq::pollitem_t p[] = {
    {subscriber, 0, ZMQ_POLLIN, 0}
};
```

소켓을 선언한 후, zmq::pollitem_t 타입의 배열을 만들고, 위의 코드와 같이 등록시킨다.

```
int rc = zmq::poll(p ,1 ,100);
if (rc != 0){
    if(p[0].revents !=0 & ZMQ_POLLIN){
        subscriber.recv(&message);
        cout<< clientID<<": received status => "<<std::string(
    }
} else{
    random = rand () % 100 + 1;
    if(random < 10){
        input = "b'(" + clientID + ":ON)'";
        zmq::message_t message(input.length());
        memcpy(message.data(), input.data(), input.length());
        pusher.send(message, 1);
        cout<<clientID<<": send status - activated" << endl;
```

Zmq::poll 을 통해 Poll의 여부를 감지한다.

P[0]에 subscriber을 등록했기 때문에, p[0]에서 poll이 발생하였을 때를 감지하였다(revents.)

poll을 감지하면 수신을 하고, 그렇지 않으면 Send를 할 수 있도록 하여 무작정 수신하기만을 기다리는 현상을 없앴다.

Dirty p2p를 구현하기 위해 local ip 주소를 받아오는 방식이 python 코드에서 구현한 것처럼 구동되지 않아 검색하여 참고하였다.

```
string get_local_ip(){
    int sock = socket(PF_INET, SOCK_DGRAM, 0);
    sockaddr_in loopback;

    if (sock == -1) {
        std::cerr << "Could not socket\n";
        return "127.0.0.1";
    }

    std::memset(&loopback, 0, sizeof(loopback));
    loopback.sin_family = AF_INET;
    loopback.sin_addr.s_addr = 1337;   // can be any IP address
    loopback.sin_port = htons(9);      // using debug port

    if (connect(sock, reinterpret_cast<sockaddr*>(&loopback),
sizeof(loopback)) == -1) {
        close(sock);
        return "127.0.0.1";
```

```
    }

    socklen_t addrlen = sizeof(loopback);
    if (getsockname(sock, reinterpret_cast<sockaddr*>(&loopback),
&addrlen) == -1) {
        close(sock);
        return "127.0.0.1";
    }
    int name = getsockname(sock,
reinterpret_cast<sockaddr*>(&loopback), &addrlen);
    close(sock);

    char buf[INET_ADDRSTRLEN];
    if (inet_ntop(AF_INET, &loopback.sin_addr, buf,
INET_ADDRSTRLEN) == 0x0) {
        return "127.0.0.1";
    } else {
        return buf;
    }
}
```

실패하는 경우 127.0.0.1 로 리턴하도록 하였고, 성공할 경우 172.30.1.60 으로 리턴이 되었다.
(lec-05-prg-12-p2p-dechat.cpp)

앞서 언급한 Thread 사용 방식을 적용하니 dirty p2p에서는 오류가 발생하였기 때문에, 새롭게 선언하는 방식을 찾아보았고, 다음과 같이 Thread를 선언하였다.

```
    });
    std::thread db_thread([&ip_addr, &port_subscribe](){
        user_manager_nameserver(ip_addr, port_subscribe);
    });
    std::thread relay_thread([&ip_addr, &port_chat_publisher, &port_chat_collector](){
        relay_server_nameserver(ip_addr, port_chat_publisher, port_chat_collector);
    });
    std::thread user_thread([&ip_addr_p2p_server, &port_subscribe, &ip_addr, &user_name, &p
        user_registeration(ip_addr_p2p_server, port_subscribe, ip_addr, user_name, port_cha
    });

    beacon_thread.join();
    cout<<"p2p beacon server is activated." << endl;
    db_thread.join();
    cout<<"p2p subscriber database server is activated."<<endl;
    relay_thread.join();
    cout<<"p2p message relay server is activated." <<endl;
    user_thread.join();
```

실행을 해보니, thread가 모두 실행되었지만, main함수가 이어서 실행되지 않는 현상이 발생하였다.

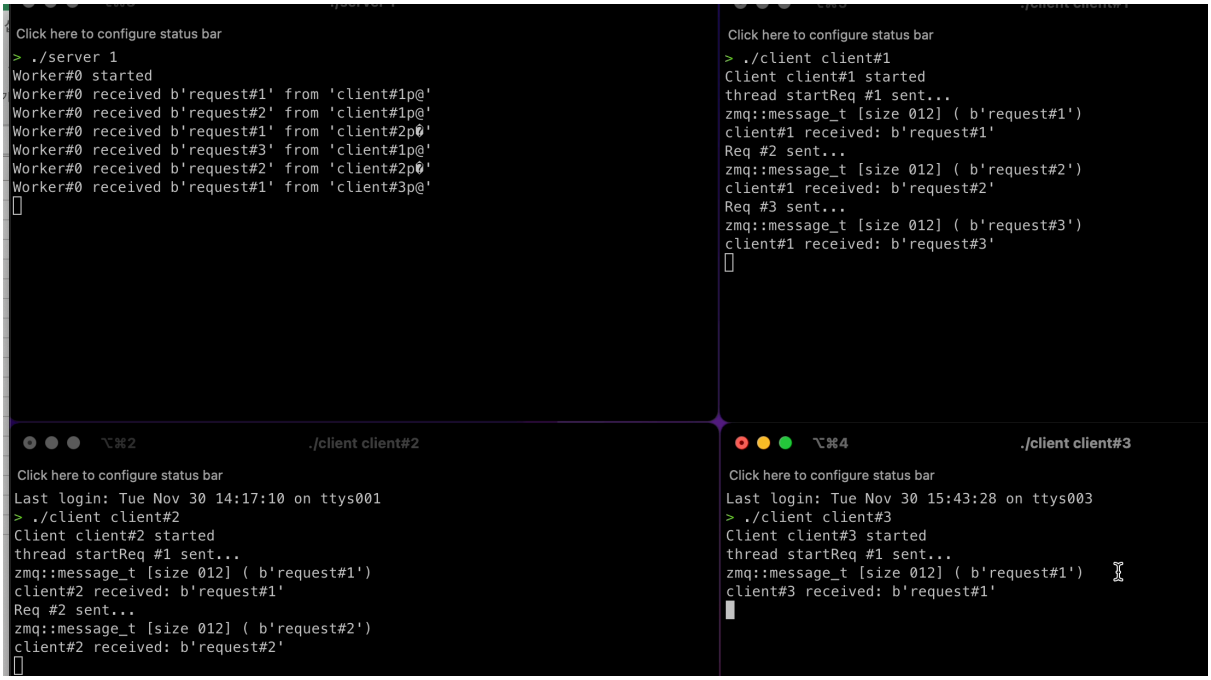
```
    });
    std::thread user_thread([&ip_addr_p2p_server, &port_subscribe, &ip_addr, &user_name, &port_chat_publishe
        user_registeration(ip_addr_p2p_server, port_subscribe, ip_addr, user_name, port_chat_publisher, port
    });

    beacon_thread.join();
    cout<<"p2p beacon server is activated." << endl;
    db_thread.join();
    cout<<"p2p subscriber database server is activated."<<endl;
    relay_thread.join();
    cout<<"p2p message relay server is activated." <<endl;
    user_thread.join();


} else{
    ip_addr_p2p_server = name_server_ip_addr;
    cout<<"p2p server found at "<<ip_addr_p2p_server<< ", and p2p client mode is activated."<<endl;
    user_registeration(ip_addr_p2p_server, port_subscribe, ip_addr, user_name, port_chat_publisher, port_cha
```

이 부분에서 해결하지 못하여 user registeration 후 통신하는 부분을 함수로 구현하여 쓰레드로 실행시켰다. (name server를 찾았을 경우에는 기본 함수로 작동되게 실행하였으며, 찾지 못했을 경우에는 user_thread로 선언하여 함께 Thread로 구현되도록 하였다.)

에러사항



lec-05-prg-11-dealer-router-async-client-thread.cpp

클라이언트에서 Thread를 사용하여 receive를 담당하는 함수를 따로 생성하여 구현하였을 때, 서버에서 식별 시 identity의 뒷부분에 알 수 없는 이름이 포함되는 현상이 발생하였다. 이를 해결하지 못하였다.

예외처리 사항

```cpp
zmq::socket_t worker(context, zmq::socket_type::dealer);
worker.connect("inproc://backend");
cout<<"Worker#"<< id<< " started"<<endl;

while(1){
    zmq::message_t msg;
    zmq::message_t iden;

    worker.recv(iden);
    string identity = string(static_cast<char*>(iden.data()), iden.size());

    worker.recv(msg);
    string received = string(static_cast<char*>(msg.data()), msg.size());
    cout<<"Worker#"<<id<<" received "<< received<<" from '"<<identity<<"'"<<endl;
    memcpy(msg.data(), received.data(), received.size());
    memcpy(iden.data(), "",0);
    worker.send(iden, ZMQ_SNDMORE);
    worker.send(msg);
```

dealer를 사용하여 Server에서 Client로 send할 때, poll을 감지하지 못하는 상황이 발생하였고, 이를 해결하기 위해서 빈 데이터를 한 번 보내서 테스트해보니 인식을 하였다. Send를 할 때 ZMQ_SNDMORE을 통해 더 보낸다는 것을 포함시켜 이후에 메시지를 정상적으로 송신할 수 있도록 하였다.
(lec-05-prg-09-dealer-router-async-server.cpp)

3. 소스코드 파일 실행 방식

동일 directory의 file_execute_table.xlsx에서 참고한다.

ZeroMQ의 경우, include path와 library path를 직접 기입해주고, c++ 버전도 다르게 넣어주어야 ZeroMQ의 경로를 찾아 실행이 성공적으로 수행되었다.

M1 mac path
- Include path: /opt/homebrew/include
- Library path: /opt/homebrew/lib

Other mac path
- Include path: /usr/local/include
- Library path: /usr/local/include or /usr/local/lib
-

챕터	분류	파일명	파일생성코드	실행코드	실행영상
TCP 1:1	server	lec-03-prg-01-tcp-echo-server.cpp	g++ {filename} -o ./server	./server	TCP_1-1.mov
	client	lec-03-prg-02-tcp-echo-client.cpp	g++ {filename} -o ./client	./client	
TCP 1:1 complete	server	lec-03-prg-03-tcp-echo-server-complete.cpp	g++ {filename} -o ./server	./server	TCP_1-1_complete.mov
	client	lec-03-prg-04-tcp-echo-client-complete.cpp	g++ {filename} -o ./client	./client	
TCP 1:N non-async	server	lec-03-prg-05-tcp-echo-server-socketserver.cpp	g++ {filename} -o ./server	./server	TCP_1-N_non-async.mov
	client	lec-03-prg-04-tcp-echo-client-complete.cpp	g++ {filename} -o ./client	./client	
TCP 1:N	server	lec-03-prg-06-tcp-echo-server-multithread.cpp	g++ {filename} -o ./server	./server	TCP_1-N_multithread.mov
	client	lec-03-prg-04-tcp-echo-client-complete.cpp	g++ {filename} -o ./client	./client	
TCP M:N	server	lec-03-prg-08-tcp-echo-server-multithread-chat.cpp	g++ {filename} -o ./server	./server	TCP_M-N_multithread.mov
	client	lec-03-prg-07-tcp-echo-client-multithread.cpp	g++ {filename} -o ./client	./client	
UDP 1:N	server	lec-03-prg-10-udp-echo-server-socketserver.cpp	g++ {filename} -o ./server	./server	UDP_1-N.mov
	client	lec-03-prg-09-udp-echo-client-multithread.cpp	g++ {filename} -o ./client	./client	
UDP M:N	server	lec-03-prg-11-udp-echo-server-socketserver-chat.cpp	g++ {filename} -o ./server	./server	UDP_M-N.mov
	client	lec-03-prg-09-udp-echo-client-multithread.cpp	g++ {filename} -o ./client	./client	

파일생성코드	g++ ${fileName} -I/opt/homebrew/include -L/opt/homebrew/lib -lzmq -o ./만들실행파일 -std=c++17			
챕터	분류	파일명	실행코드	실행영상
REQ-REP	server	lec-05-prg-01-req-rep-basic-server.cpp	./server	req-rep-basic-server.mov
	client	lec-05-prg-02-req-rep-basic-client.cpp	./client	
PUB-SUB Basic	server	lec-05-prg-03-pub-sub-basic-server.cpp	./server	pub-sub-basic-server.mov
	client	lec-05-prg-04-pub-sub-basic-client.cpp	./client	
PUB-SUB Pull-Push	server	lec-05-prg-05-pub-sub-and-pull-push-server.cpp	./server	pub-sub-and-pull-push-server.mov
	client	lec-05-prg-06-pub-sub-and-pull-push-client.cpp	./client	
PUB-SUB Pull-Push -v2	server	lec-05-prg-07-pub-sub-and-pull-push-server-v2.cpp	./server	pub-sub-and-pull-push-server-v2.mov
	client	lec-05-prg-08-pub-sub-and-pull-push-client-v2.cpp	./client ${클라이언트 이름}	
Dealer-Router-singleThread	server	lec-05-prg-09-dealer-router-async-server.cpp	./server 1	dealer-router-async-server-single-thread.mov
	client	lec-05-prg-10-dealer-router-async-client.cpp	./client ${클라이언트 이름}	
Dealer-Router-four Thread	server	lec-05-prg-09-dealer-router-async-server.cpp	./server 4	dealer-router-async-server-four-thread.mov
	client	lec-05-prg-10-dealer-router-async-client.cpp	./client ${클라이언트 이름}	
p2p-dechat	server	lec-05-prg-12-p2p-dechat.py	./dechat ${클라이언트 이름}	p2p-dechat.mov
	client			

4. 느낀점

C++을 사용하면서 여태껏 나는 Thread를 한 번도 사용한 적이 없었다. 이번 프로젝트를 통해 Thread를 다양한 방법으로 사용해볼 수 있는 경험을 쌓는 기회가 되었다. 또한 메시지를 주고받을 때 데이터 형식을 하나씩 변환시켜줘야 하는 점에서도 어려움이 있었으며, 문자 뿐만 아니라 함수 혹은 Thread로 호출할 때 socket type을 parameter로 넘겨 그대로 사용하고 싶을 때, 형식 혹은 규칙에 의해 제한되는 점이 많아 검색을 통해 이해하는데 시간이 많이 들인 것 같다. 그 결과 Python으로 제공된 예제 코드에 비해 굉장히 난잡한 코드가 만들어졌다. C++을 주력 언어로 생각하고 있었지만, 아직까지도 많이 부족한 점이 많다고 느꼈다. 또한 그만큼 Python이 C++에 비해 쉽게 구현할 수 있다고 조심스럽게 주장해본다. 코딩 테스트를 진행할 때 C++을 사용하기로 마음을 먹었으니 이번 프로젝트를 하면서 부족한 부분들을 보완해 나가며 기초적인 부분부터 다시 하나씩 검토해보기로 다짐했다.

이번 학기에 들어오면서 Mac을 사용하기 시작하였는데, visual studio에서 C++ 개발환경을 구축해야 했고 M1 Mac이 이전 세대와 비교하였을 때 호환성 혹은 include 저장 경로 등이 달랐기 때문에 헷갈린 점이 많기도 하였으며, ZeroMQ를 사용하기 위해서 실행 시 include와 library 경로를 설정해주는 것에도 어려움이 있었다. 다른 자료를 참고해보니 include와 library path를 절대적으로 등록할 수 있는것으로 확인하였지만, 직접 적용해보니 실행되지 않아 파일을 생성할 때 직접 명시해야하는 번거로움이 있었다. M1 Mac의 호환성의 문제인지, 잘못된 환경을 구축해 둔 것인지 아직 알아보지는 못했지만 이후의 개발하는 과정들을 대비하여 학기가 끝난 후 더욱 최적화된 환경을 구축해두기로 결심하였다.