서버
포트폴리오

지원자 : 김성원

프로젝트

1. IOCP채팅 프로그램

2. 던전앤파이터 모작

IOCP 채팅 프로그램

김성원

IOCP채팅 프로그램

개발 기간

- 20.10.05 ~ 20.11.06

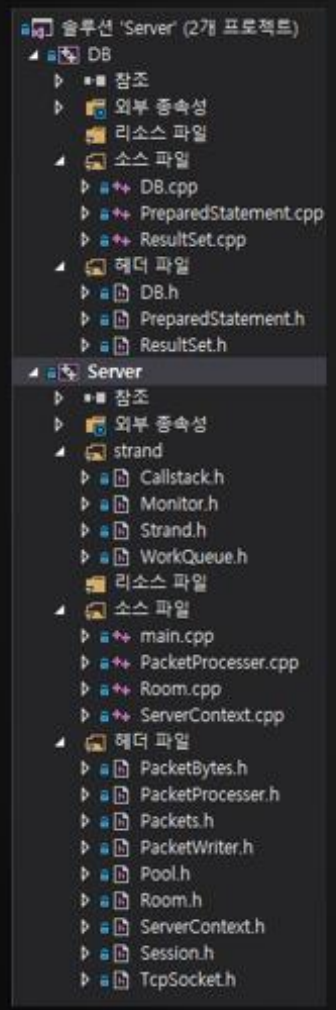
개발 환경

- 서버 : Visual 2017 c++, MySQL
- 클라이언트 : Visual 2017 c# , Unity

특징

- 프로젝트를 정적 라이브러리로 분리 개발
- 네트워크 포트설정을 통해 다른 컴퓨터의 DataBase 접근

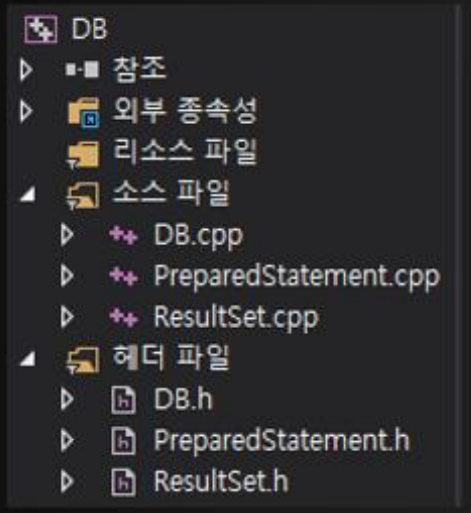
프로젝트 분리



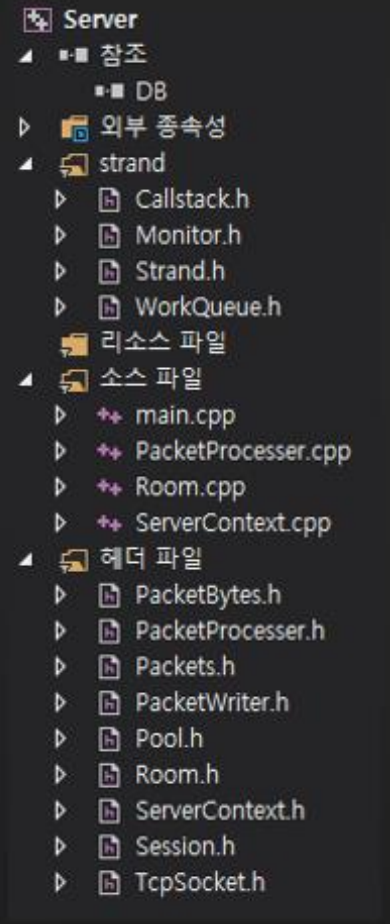
- 서버 로직 관련 프로젝트와 DB관련 프로젝트를 분리하여 제작
- DB프로젝트는 정적라이브러리 프로젝트로 서버 프로젝트에서 참조하여 사용

DB.lib	2020-11-10 오전 2:36	Object File Library	987KB
DB.pdb	2020-11-10 오전 2:36	Program Debug ...	956KB
mysqlcppconn-7-vs14.dll	2020-07-04 오전 1:48	응용 프로그램 확장	8,775KB
Server.exe	2020-11-10 오전 2:36	응용 프로그램	828KB
Server.ilk	2020-11-10 오전 2:36	Incremental Linke...	3,308KB
Server.pdb	2020-11-10 오전 2:36	Program Debug ...	2,844KB

기능별 Class 분리



- PreparedStatement Class
  DB의 insert 부분을 관리
- ResultSet Class
  DB의 Table을 조회



- WorkQueue Class
  쓰레드들에게 작업 분배
- Strand Class
  multi thread의 보장되지 않은
  작업순서를
  순서대로 처리하게 해줌
- PacketProcesser Class
  패킷 종류별로 작업 처리
- PacketBytes Class
  패킷 직렬화 클래스

Client 관리

```
using pSession = std::shared_ptr<Session>;
class Session : public std::enable_shared_from_this<Session>
{
    PacketProcesser &pp_;
    PacketBytes receiveBytes_;
    TcpSocket socket_;
```

- 클라이언트는 접속 시 Session을 할당
- Session들은 list로 관리

Recv

```cpp
using pSession = std::shared_ptr<Session>;
class Session : public std::enable_shared_from_this<Session>
{
    PacketProcesser &pp_;
    PacketBytes receiveBytes_;
    TcpSocket socket_;

    void RecieveHandler(unsigned char *bytes, int length)
    {
        printf("bytes %d receivedWn", length);

        receiveBytes_.WriteJustBytes(bytes, length);

        while (true)
        {
            auto res = pp_.PacketProcess(shared_from_this(), receiveBytes_);
            if (res == PacketProcesser::Error::None)
            {
                continue;
            }

            if (res == PacketProcesser::Error::WrongPacketKind)
            {
                this->socket_.Close(TcpSocket::CloseReason::WrongPacket);
                return;
            }
            break;
        }

        auto me = shared_from_this();
        socket_.Receive([me](unsigned char *b2, int len) mutable {
            me->RecieveHandler(b2, len);
        });
    }
```

- TCP/IP특성 상 Packet이 잘려서 또는 합쳐서 올 수 있기 때문에 Packet을 재조립하는 작업이 필수
- 수신받은 Bytes를 PacketProcess에 넘겨 하나의 Packet이 완성되는지 체크
- Packet에 이상이 없다면 해당 패킷을 조립
- 이상한 종류의 패킷이 도착하면 그 소켓을 닫음

Recv

```
PacketProcesser::Error PacketProcesser::PacketProcess(pSession sess, PacketBytes& receiveBytes)
{
    ushort readOffset = 0;

    if (receiveBytes.writeOffset_ < 4) {
        return Error::NotEnoughPacketBytes;
    }

    auto pkKind = receiveBytes.ReadUInt16(readOffset);
    if (!IsValidPacketKind(pkKind)) {
        return Error::WrongPacketKind;
    }

    unsigned short pkLen = receiveBytes.ReadUInt16(readOffset);

    if (pkLen > receiveBytes.writeOffset_) {
        return Error::NotEnoughPacketBytes;
    }

    packetHandleMap[(PacketKind)pkKind](sess, receiveBytes, readOffset);

    receiveBytes.ShiftLeftBytes(pkLen);

    return Error::None;
}
```

1. 패킷의 헤더부분 (앞 4비트)이 도착하지 않았다면 다시 수신
2. 패킷의 종류가 정의 되어있지 않다면 실패 메시지
3. 패킷을 조립할 바이트의 길이가 충분하지 않다면 다시 수신
4. 패킷을 만들만큼 충분히 수신했다면 수신 받은 패킷의 크기만큼 쉬프트

Bytes 직렬화 & Packet 재조립

```cpp
enum class PrimitiveType
{
    Char,
    Byte,
    Int,
    UInt,
    Short,
    UShort,
    String
};

struct MemberVariable
{
    PrimitiveType type;
    ushort Offset;
};

class DataType
{
public:
    DataType(std::initializer_list<MemberVariable> inMvs) : members_(inMvs) {}
    const std::vector<MemberVariable> &GetMemberVariables() const
    {
        return members_;
    }

private:
    std::vector<MemberVariable> members_;
};

#define OffsetOf(c, mv) ((size_t) & (static_cast<c *>(nullptr)->mv))
```

- TCP/IP의 특징으로 packet이 끊어지거나 이어져 올 수 있으므로
  직렬화된 packet을 재조립하는
  과정은 필수

- 넘겨받는 데이터의 offset을 기록
- C# 과 통신 해야하므로
  변환 과정에서 엔디안 통일

Write

```
void WriteInt8(char v)
{
    bytes_[writeOffset_] = v;
    writeOffset_ += 1;
}

void WriteUInt8(unsigned char v)
{
    bytes_[writeOffset_] = v;
    writeOffset_ += 1;
}

void WriteInt32(int v)
{
    auto pVal = reinterpret_cast<int *>(bytes_ + writeOffset_);
    *pVal = htonl(*((int *)&v));
    writeOffset_ += 4;
}

void WriteFloat(float v)
{
    auto pVal = reinterpret_cast<uint32 *>(bytes_ + writeOffset_);
    *pVal = htonl(*((uint32 *)&v));
    writeOffset_ += 4;
}

void WriteString(std::string str)
{
    const uint8_t *p = reinterpret_cast<const uint8_t *>(str.c_str());
    ushort len = static_cast<ushort>(str.length());
    WriteUInt16(len);
    WriteJustBytes(p, len);
}
```

Read

```
short ReadInt16(ushort &readOffset)
{
    auto v = ntohs(*((short *)(bytes_ + readOffset)));

    readOffset += 2;
    return v;
}

ushort ReadUInt16(ushort &readOffset)
{
    auto v = ntohs(*((ushort *)(bytes_ + readOffset)));
    readOffset += 2;
    return v;
}

uint32 ReadUInt32(ushort &readOffset)
{
    auto v = ntohl(*((uint32 *)(bytes_ + readOffset)));
    readOffset += 4;
    return v;
}

int ReadInt32(ushort &readOffset)
{
    auto v = ntohl(*((int *)(bytes_ + readOffset)));
    readOffset += 4;
    return *(int *)(&v);
}

float ReadFloat(ushort &readOffset)
{
    auto v = ntohl(*((uint32 *)(bytes_ + readOffset)));
    readOffset += 4;
    return *(float *)(&v);
}
```

SQL

- MySQL connection c++
- Query는 인젝션 공격을 보안하는PreparedStatemet를 사용

```
bool DB::InsertData(std::string id, std::string pw, std::string name)
{
    ResultSet resultSet(mysqlInstance_->con, "INSERT INTO userinfo VALUES (?, ?, ?, ?)");
    vector<string> sqlVec = { id,pw,name,"0" };
    resultSet.SetSqlStr(sqlVec);
    return resultSet.ExecuteUpdate();
}
```

- SQL 인젝션(Injection) 공격
  - DB에서 처리되는 쿼리문을 주입하여 비정상적으로 데이터베이스에 접근하는 해킹기법

```
using namespace std;

class PreparedStatement
{
public:
    PreparedStatement(sql::Connection* con, string str)
    {
        preStmt_ = con->prepareStatement(str.c_str());
    }

    virtual ~PreparedStatement()
    {
        delete preStmt_;
    }

    void SetSqlStr(vector<string>& strVec) const;
    bool ExecuteUpdate() const;

protected:
    sql::PreparedStatement* preStmt_;
};
```

- PreparedStatement를 편하게 이용하기 위해 class 구현
- 상속 과정에서 사용할 변수는 protected 선언
- 소멸자를 가상함수로 선언함으로써 부모 클래스의 자료형으로 선언된 자식 클래스의 소멸자도 호출되도록 함

```cpp
using namespace std;

class ResultSet : public PreparedStatement
{
public:
    ResultSet(sql::Connection* con)
        : PreparedStatement(con, "SELECT * FROM userinfo")
    {
        ExecuteQuery_();
    }

    ResultSet(sql::Connection* con, string str)
        : PreparedStatement(con, str)
    {
        ExecuteQuery_();
    }

    virtual ~ResultSet()
    {
        delete res_;
    }

    bool CheckIdPw(string id, string pw) const;
    string GetName(string id) const;
    int GetMoney(string id) const;
private:
    sql::ResultSet* res_;
    void ExecuteQuery_();
};
```

- PreparedStatement를 상속받은 ReultSet class
- PreparedStatement는 ExecuteUpdate 이용(INSERT)
- ResultSet은 하나의 쿼리를 반환하는 ExecuteQuery를 이용(SELECT)

시연 영상

던전앤파이터 모작

김성원

던전앤파이터 모작

개발 기간

- 20.09.07 ~ 20.09.29

개발 환경

- visual studio 2017

개발 언어

- C++

개발 인원

- 서버 1명(본인), 클라이언트 3명

구현 내용

구현 장면 : 던전앤파이터 인던

서버의 역할 : 클라이언트간 캐릭터의 위치, 행동 상태 전달

- 패킷 직렬화를 통한 서버-클라이언트간 통신 구현

패킷 구조 설계

- TCP/IP는 stream 방식으로 recv가 된다.
  - 패킷이 잘라지거나 연이어 들어 올 수 있기에 별도의 처리를 해주어야 함.

- 패킷의 처음 1byte 부분을 패킷의 길이 , 다음 1byte를 패킷의 종류로 약속
  - 단점 – 패킷의 길이와 종류가 256가지를 넘을 수 없어 넘길 수 있는 정보의 한계가 있음.
  - 이번 프로젝트에서는 많은 양을 필요로 하지 않으므로 허용범위

패킷 구조

```
#define S2C_USER_INFO 1
struct sc_packet_clientinfo
{
    unsigned char size;
    unsigned char type;
    bool online;
    int id;
    char job;
    float x, y, z;
    int state;
    char myIp[15];
    u_int myPort;

    sc_packet_clientinfo()
    {
        size = sizeof(sc_packet_clientinfo);
        type = S2C_USER_INFO;
    }
};
```

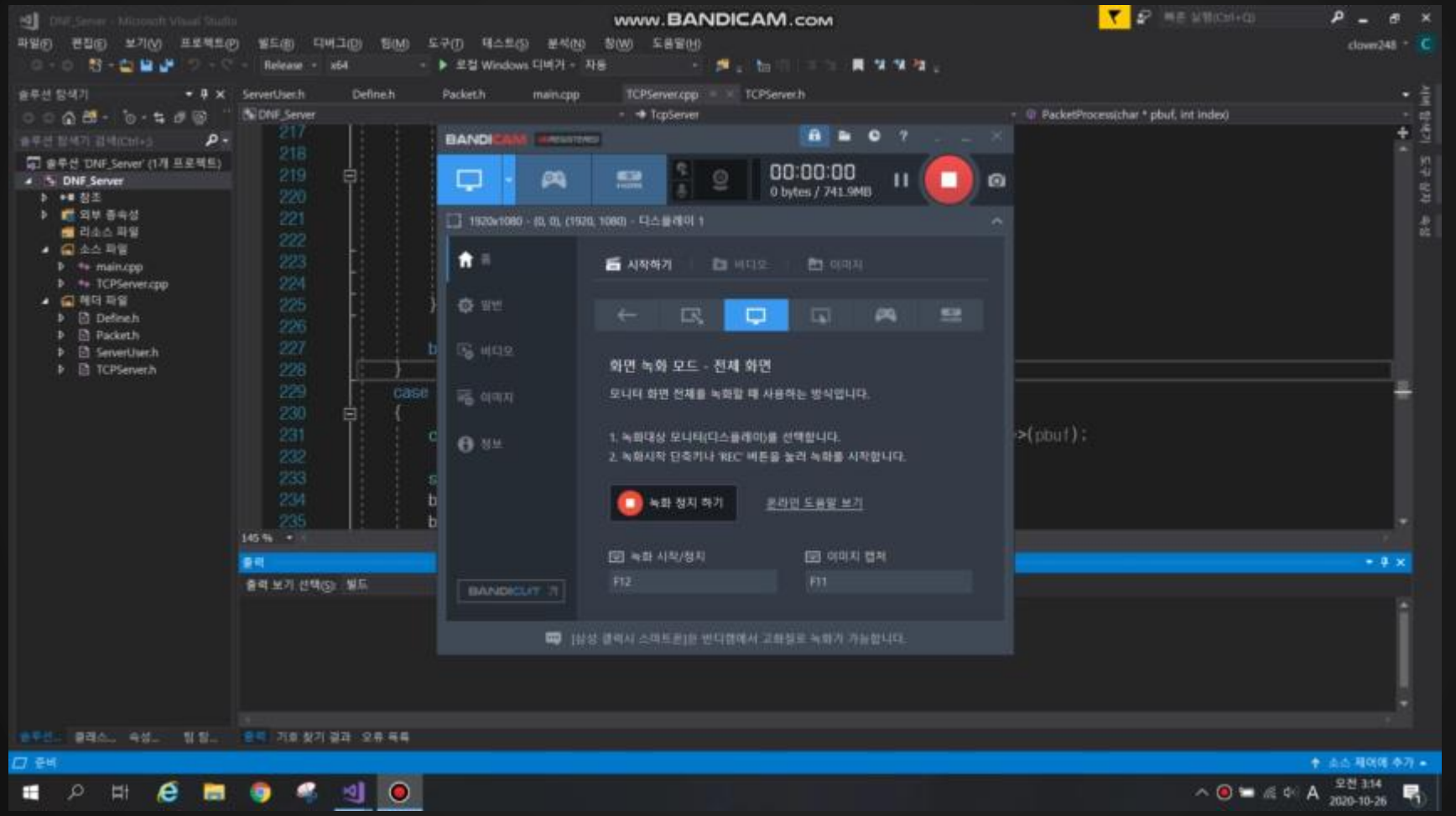
- 매크로를 사용해 패킷 종류 결정
- 생성자로 패킷을 만들 때
  공통적인 정보를 미리 입력
- 패킷 네이밍 규약
  - S2C(Server to Client)
    - 서버에서 클라이언트로 보내는 패킷
  - C2S(Client to Server)
    - 클라이언트에서 서버로 보내는 패킷
- 같은 구조의 다른 멤버변수를 가진
  패킷들이 존재 (C2S 패킷도 동일)

TcpServer.cpp

```
void TcpServer::PacketProcess(char* pbuf, int index)
{

    switch (pbuf[1])
    {
        if (index <= 0 || index > MAX_USERS)
            break;
    case C2S_LOGIN:
    {
        cout << "C2S_LOGIN" << endl;
        cs_packet_log *packet = reinterpret_cast<cs_packet_log*>(pbuf);

        sc_packet_login_ok buf;
        buf.id = index;

        Server_Send(buf, index);
        break;
    }

    case C2S_LOGIN_TRY:
    {
```

- C2S 패킷이 수신되면 패킷의 종류마다 각 작업 후
  S2C패킷을 클라이언트에게 전송한다.

시연 영상


www.BANDICAM.com
00:00:00
0 bytes / 741.9MB
화면 녹화 모드 - 전체 화면
모니터 화면 전체를 녹화할 때 사용하는 방식입니다.
1. 녹화대상 모니터(디스플레이)를 선택합니다.
2. 녹화시작 단축키나 'REC' 버튼을 눌러 녹화를 시작합니다.
녹화 정지 하기
온라인 도움말 보기
F12
F11

아쉬운 점

- 클라이언트들이 바로 던전으로 접속하여 접속 시점이 동일하지 않고 몬스터 동기화가 안됨
  - 로비서버 구현 필요 및 몬스터 위치도 동기화 시키는 방향으로 개선 필요
- 플레이어의 위치를 담은 패킷을 처리하는 부분
  - 위치좌표를 그대로 적용시키면 캐릭터가 끊겨 이동하기 때문에 클라이언트 상에서 보간처리 필요