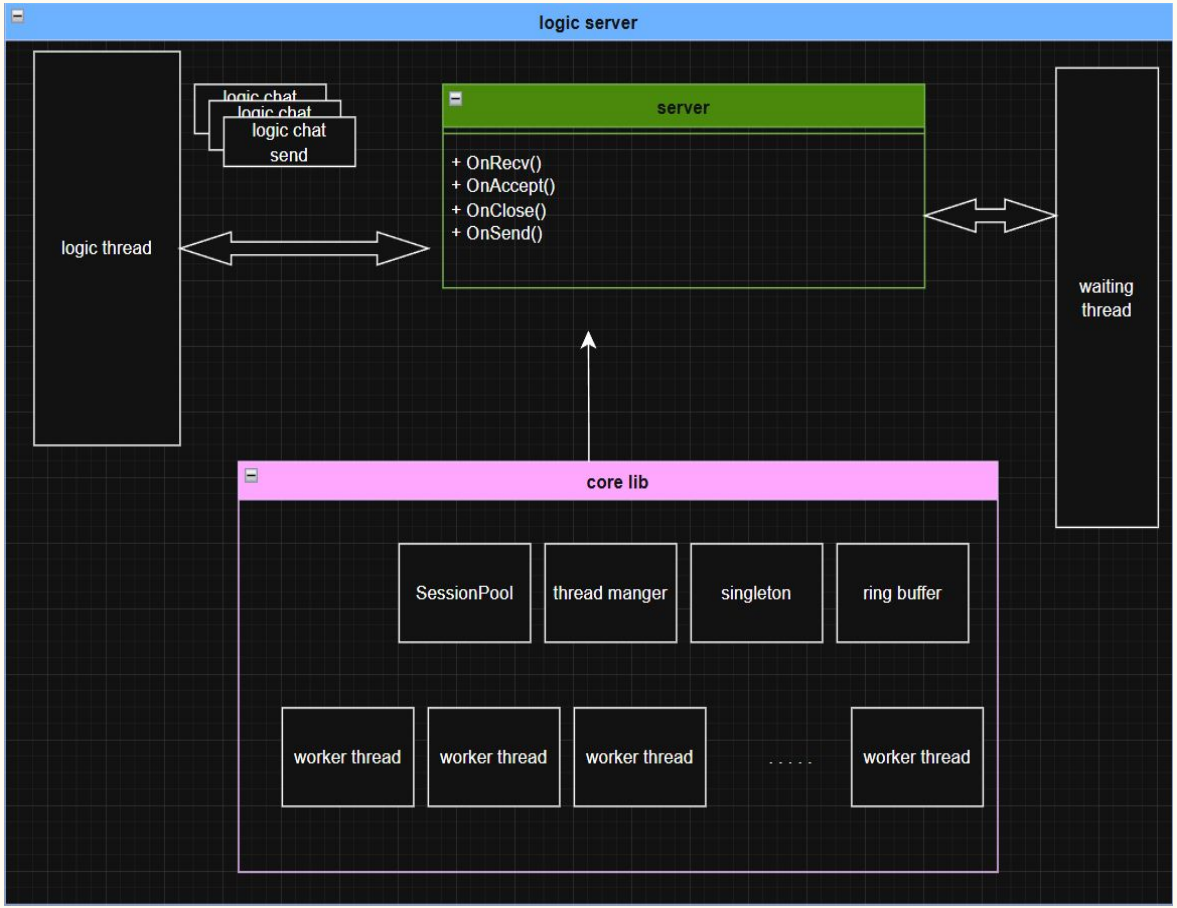
포트 폴리오

문진웅
010-5120-2097
jazzmaster0415@gmail.com
git : https://github.com/true-bear/logic-server

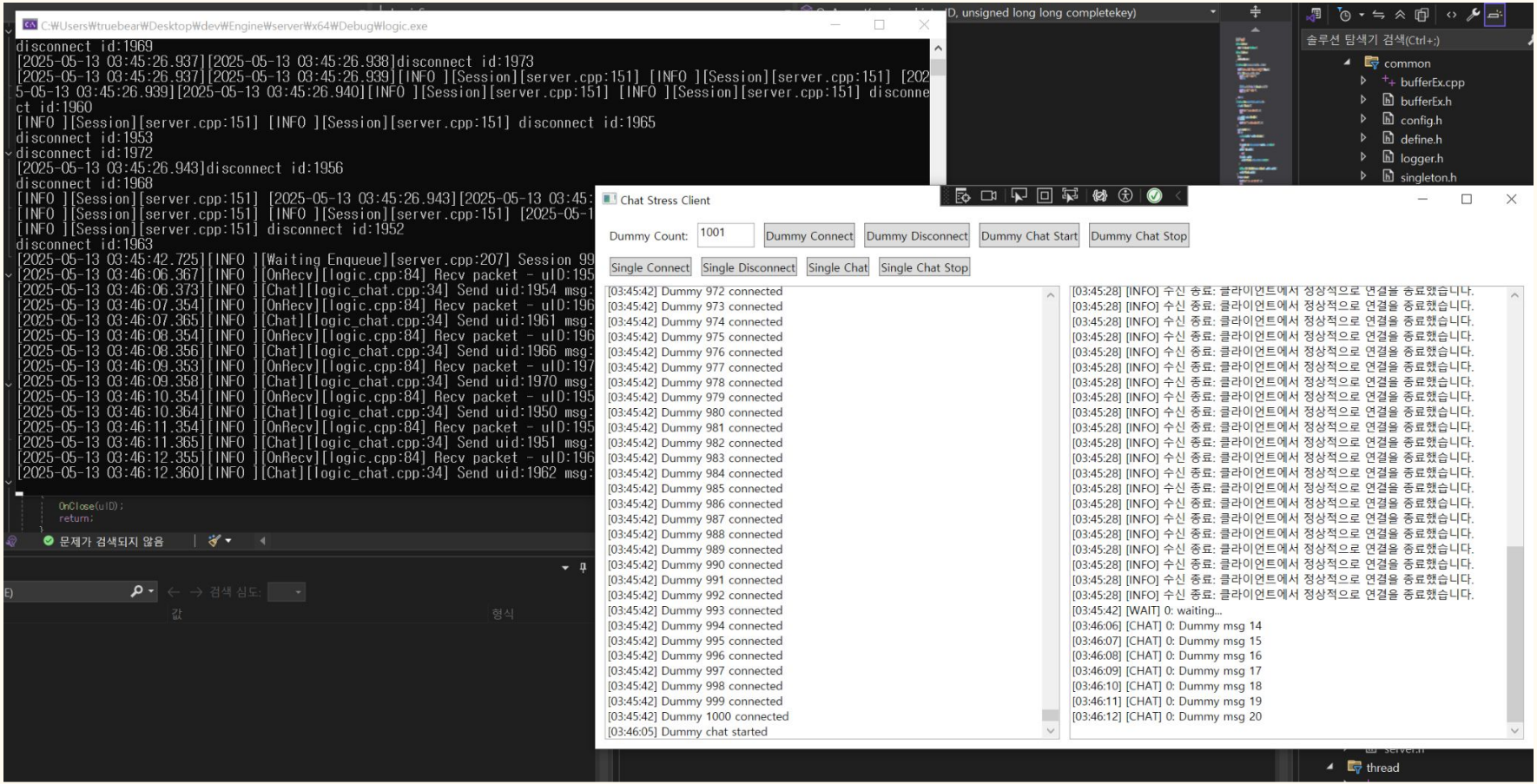


<core : 정적 라이브러리>

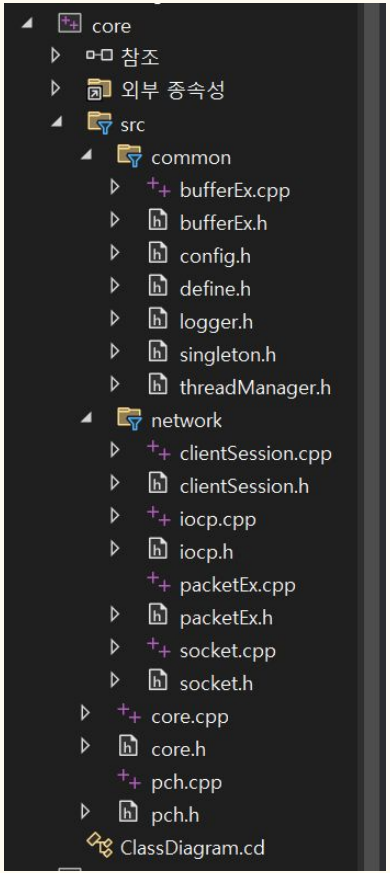
네트워크 처리 , 세션 스레드 매니저 등
전반적인 IO 및 세션 부분의 기능들을
정의하는 라이브러리 프로젝트

<server>

- 코어를 상속하고 io들어오는
  패킷들을 처리하고 로직을
  실행하는 서버
- 로직을 위한 로직 스레드
- 대기열을 위한 대기열 스레드 구현


C:₩Users₩truebear₩Desktop₩dev₩Engine₩server₩x64₩Debug₩logic.exe
disconnect id:1969
[2025-05-13 03:45:26.937][2025-05-13 03:45:26.938]disconnect id:1973
[2025-05-13 03:45:26.937][2025-05-13 03:45:26.939][INFO][Session][server.cpp:151] [INFO][Session][server.cpp:151] [2025-05-13 03:45:26.939][2025-05-13 03:45:26.940][INFO][Session][server.cpp:151] [INFO][Session][server.cpp:151] disconnect id:1960
[INFO][Session][server.cpp:151] [INFO][Session][server.cpp:151] disconnect id:1965
disconnect id:1953
disconnect id:1972
[2025-05-13 03:45:26.943]disconnect id:1956
disconnect id:1968
[INFO][Session][server.cpp:151] [2025-05-13 03:45:26.943][2025-05-13 03:45:
[INFO][Session][server.cpp:151] [INFO][Session][server.cpp:151] [2025-05-1
[INFO][Session][server.cpp:151] disconnect id:1952
disconnect id:1963
[2025-05-13 03:45:42.725][INFO][Waiting Enqueue][server.cpp:207] Session 99
[2025-05-13 03:46:06.367][INFO][OnRecv][logic.cpp:84] Recv packet - uID:195
[2025-05-13 03:46:06.373][INFO][Chat][logic_chat.cpp:34] Send uid:1954 msg:
[2025-05-13 03:46:07.354][INFO][OnRecv][logic.cpp:84] Recv packet - uID:196
[2025-05-13 03:46:07.365][INFO][Chat][logic_chat.cpp:34] Send uid:1961 msg:
[2025-05-13 03:46:08.354][INFO][OnRecv][logic.cpp:84] Recv packet - uID:196
[2025-05-13 03:46:08.356][INFO][Chat][logic_chat.cpp:34] Send uid:1966 msg:
[2025-05-13 03:46:09.353][INFO][OnRecv][logic.cpp:84] Recv packet - uID:197
[2025-05-13 03:46:09.358][INFO][Chat][logic_chat.cpp:34] Send uid:1970 msg:
[2025-05-13 03:46:10.354][INFO][OnRecv][logic.cpp:84] Recv packet - uID:195
[2025-05-13 03:46:10.364][INFO][Chat][logic_chat.cpp:34] Send uid:1950 msg:
[2025-05-13 03:46:11.354][INFO][OnRecv][logic.cpp:84] Recv packet - uID:195
[2025-05-13 03:46:11.365][INFO][Chat][logic_chat.cpp:34] Send uid:1951 msg:
[2025-05-13 03:46:12.355][INFO][OnRecv][logic.cpp:84] Recv packet - uID:196
[2025-05-13 03:46:12.360][INFO][Chat][logic_chat.cpp:34] Send uid:1962 msg:
OnClose(uID);
return;
문제가 검색되지 않음
검색 심도:
값
형식
D, unsigned long long completekey)
솔루션 탐색기 검색(Ctrl+;)
common
bufferEx.cpp
bufferEx.h
config.h
define.h
logger.h
singleton.h
thread
Chat Stress Client
Dummy Count:
1001
Dummy Connect
Dummy Disconnect
Dummy Chat Start
Dummy Chat Stop
Single Connect
Single Disconnect
Single Chat
Single Chat Stop
[03:45:42] Dummy 972 connected
[03:45:42] Dummy 973 connected
[03:45:42] Dummy 974 connected
[03:45:42] Dummy 975 connected
[03:45:42] Dummy 976 connected
[03:45:42] Dummy 977 connected
[03:45:42] Dummy 978 connected
[03:45:42] Dummy 979 connected
[03:45:42] Dummy 980 connected
[03:45:42] Dummy 981 connected
[03:45:42] Dummy 982 connected
[03:45:42] Dummy 983 connected
[03:45:42] Dummy 984 connected
[03:45:42] Dummy 985 connected
[03:45:42] Dummy 986 connected
[03:45:42] Dummy 987 connected
[03:45:42] Dummy 988 connected
[03:45:42] Dummy 989 connected
[03:45:42] Dummy 990 connected
[03:45:42] Dummy 991 connected
[03:45:42] Dummy 992 connected
[03:45:42] Dummy 993 connected
[03:45:42] Dummy 994 connected
[03:45:42] Dummy 995 connected
[03:45:42] Dummy 996 connected
[03:45:42] Dummy 997 connected
[03:45:42] Dummy 998 connected
[03:45:42] Dummy 999 connected
[03:45:42] Dummy 1000 connected
[03:46:05] Dummy chat started
[03:45:28] [INFO] 수신 종료: 클라이언트에서 정상적으로 연결을 종료했습니다.
[03:45:42] [WAIT] 0: waiting...
[03:46:06] [CHAT] 0: Dummy msg 14
[03:46:07] [CHAT] 0: Dummy msg 15
[03:46:08] [CHAT] 0: Dummy msg 16
[03:46:09] [CHAT] 0: Dummy msg 17
[03:46:10] [CHAT] 0: Dummy msg 18
[03:46:11] [CHAT] 0: Dummy msg 19
[03:46:12] [CHAT] 0: Dummy msg 20

00_core



Core Library

- core: IOCP 기반 비동기 네트워크 서버의 핵심 관리 클래스
    - 멀티스레드 워커 스레드
    - 세션 풀을 통한 클라이언트 연결 관리
    - Accept/Send/Recv 이벤트 처리
    - 서버 초기화 및 종료 처리

- clientSession: IO처리를 담당하는 세션 클래스

- iocp: iocp 핸들을 이용해서 api들을 모아서 래핑한 클래스

- packetEx: 로직 처리를 위한 패킷틀을 정의한 클래스

- socket: 소켓함수의 기능을 모아놓은 클래스

- bufferEx : 원형 버퍼 클래스

- threadManager: 여러 스레드들의 생성 파괴를 관리를 위한 클래스

00_core

```cpp
void Core::WorkerThread()
{
    while (mIsRunThread)
    {
        Iocp::IocpEvents events;
        GQCSEx(events, 5);

        for (int i = 0; i < events.m_eventCount; ++i)
        {
            auto& getIoEvent = events.m_ioArray[i];

            if (getIoEvent.lpOverlapped == nullptr)
            {
                LOG_INFO("Core", "thread shutdown");
                return;
            }

            unsigned long ioSize = getIoEvent.dwNumberOfBytesTransferred;
            OverlappedIoEx* over = reinterpret_cast<OverlappedIoEx*>(getIoEvent.lpOverlapped);

            if (!over)
            {
                LOG_ERR("Core", "Overlapped nullptr");
                continue;
            }
            {
                LOG_ERR("Core", "Overlapped nullptr");
                continue;
            }

            int sessionID = over->mUID;
            IO_TYPE ioType = over->mIOType;

            auto session = GetSession(sessionID);
            if (!session)
            {
                LOG_ERR("Core", "session nullptr id:{}", sessionID);
                continue;
            }
```

```cpp
bool Core::CreateSessionPool()
{
    int total = mMaxSession + mMaxWaiting;

    for (auto i = 0; i < total; ++i)
    {
        auto session = std::make_unique<ClientSession>();
        if (!session)
            return false;

        session->SetUniqueId(i);

        const auto& listenSocket = mListenSocket.GetSocket();
        if (!session->AcceptReady(listenSocket, i))
            return false;

        mSessionPool.emplace(i, std::move(session));
    }
    return true;
}

ClientSession* Core::GetSession(unsigned int uID) const
{
    auto it = mSessionPool.find(uID);
    return (it != mSessionPool.end()) ? it->second.get() : nullptr;
}
```

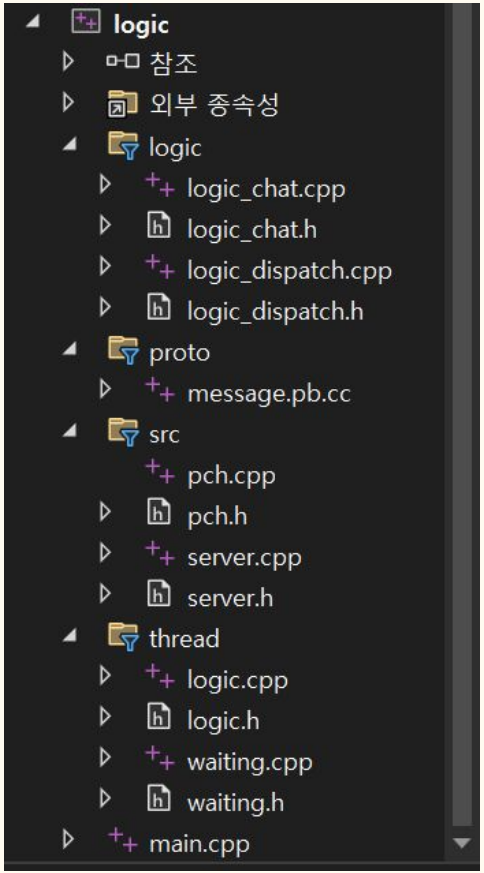
worker thread

- GetQueuedCompletionStatusEx : **io를 지정된 갯수**만큼 한꺼번에 가져와서 처리
- OverlappedEx : 오버랩드 상속받아서 확장 구조체 정의 한 모델로 사용
- 타입에 따라서 OnClose, OnRecv, OnSend, OnAccept 호출

session pool

- 잦은 힙할당 해제 대신 풀을 미리 만들어서 연결만 해주는 방식으로 구현

01_logic serve



Logic

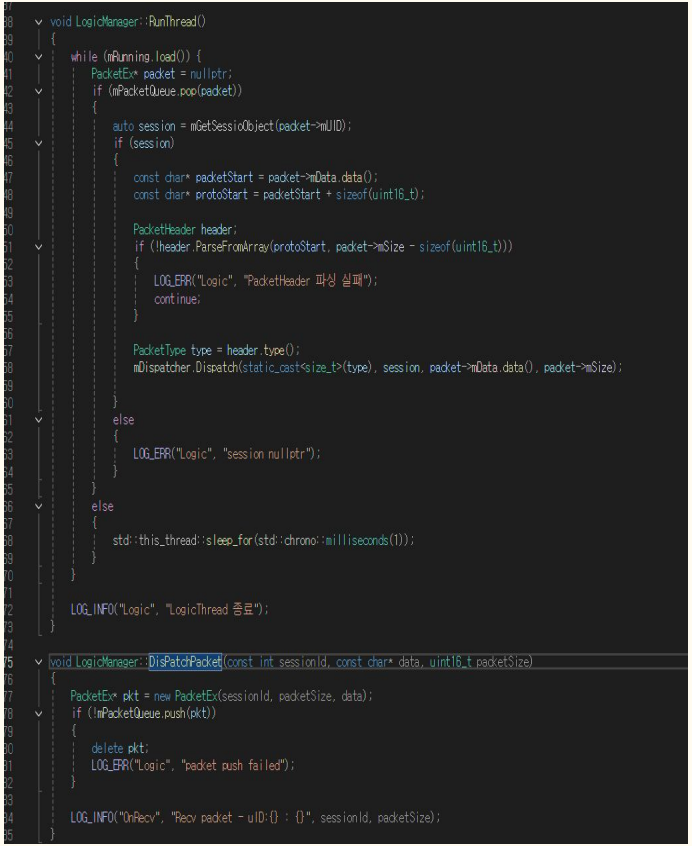
- logic: 로직 처리 담당하는 스레드

- waiting: 대기열 처리 담당하는 스레드

- server: **core를 상속** 받아서 OnRecv, OnSend, OnClose, OnAccept 등의 처리를 받는 클래스

- logic_dispatch : 로직들을 등록하고 찾아서 처리할 수 있게 해주는 클래스

01_logic server



```
void LogicManager::RunThread()
{
    while (mRunning.load()) {
        PacketEx* packet = nullptr;
        if (mPacketQueue.pop(packet))
        {
            auto session = mGetSessionObject(packet->mUID);
            if (session)
            {
                const char* packetStart = packet->mData.data();
                const char* protoStart = packetStart + sizeof(uint16_t);

                PacketHeader header;
                if (!header.ParseFromArray(protoStart, packet->mSize - sizeof(uint16_t)))
                {
                    LOG_ERR("Logic", "PacketHeader 파싱 실패");
                    continue;
                }

                PacketType type = header.type();
                mDispatcher.Dispatch(static_cast<size_t>(type), session, packet->mData.data(), packet->mSize);

            }
            else
            {
                LOG_ERR("Logic", "session nullptr");
            }
        }
        else
        {
            std::this_thread::sleep_for(std::chrono::milliseconds(1));
        }
    }

    LOG_INFO("Logic", "LogicThread 종료");
}

void LogicManager::DisPatchPacket(const int sessionId, const char* data, uint16_t packetSize)
{
    PacketEx* pkt = new PacketEx(sessionId, packetSize, data);
    if (!mPacketQueue.push(pkt))
    {
        delete pkt;
        LOG_ERR("Logic", "packet push failed");
    }

    LOG_INFO("OnRecv", "Recv packet - uID:{} : {}", sessionId, packetSize);
}
```

Logic thread

- DisPatchPacket() OnRecv 처리가 된 패킷들을 boost::lockfree::queue에 넣어주는 함수

- 스레드 큐에서 빼와서 dispatch로 분기를 태워서 처리를 해주는 방식

01_logic server

```cpp
void WaitingManager::RunThread()
{
    while (mRunning.load())
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(10));

        auto session = TryDequeue();
        if (!session)
            continue;

        if (LogicServer::Get().HasFreeSlot())
        {
            LogicServer::Get().BindSession(session);
        }
        else
        {
            Enqueue(session);
            continue;
        }
    }
}
```

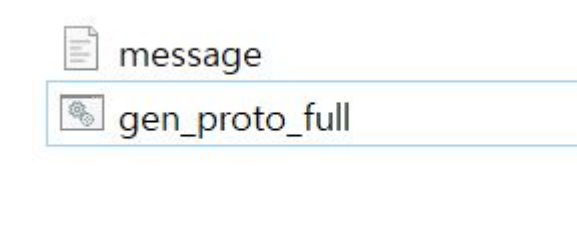
```cpp
{
    std::lock_guard<std::mutex> lock(mActiveSessionLock);
    if ((int)mActiveSessionMap.size() < mMaxSession)
    {
        mActiveSessionMap[uID] = session;
        session->RecvReady();
    }
    else
    {
        WaitingPacket packet;
        auto* header = new PacketHeader();
        header->set_type(PacketType::WAITING);
        header->set_length(packet.ByteSizeLong());

        packet.set_allocated_header(header);
        packet.set_message("waiting...");
        packet.set_waiting_number(WaitingManager::Get().Size());

        int size = packet.ByteSizeLong();
        std::vector<char> buf(size);
        if (!packet.SerializeToArray(buf.data(), size))
        {
            LOG_ERR("WaitingPacket", "Serialize 실패");
            return;
        }

        session->SendPacket(buf.data(), size);

        WaitingManager::Get().Enqueue(session);
        LOG_INFO("Waiting Enqueue", "Session {} pushed to waiting queue", uID);
    }
}
```

Waiting thread

- OnAccept 시 세션의 여유가 있으면 수신을 걸어주고, 없으면 대기열 큐 스레드에 넣어주는 로직

- 웨이팅 스레드에서는 주기적으로 돌면서 세션 여유분 체크하고 조건에 부합하면 소켓 할당하고 수신을 걸어줌

02_protobuf



protobuf

- gen_proto_pull.bat: 윈도우용 배치 파일. message.proto를 컴파일하고 지정된 디렉토리에 복사하는 간단한 배치파일 구현

- 서버는 c++ 클라이언트는 c#으로 구현해서 간단하게 배치파일 실행으로 이기종 언어의 통신을 도와줌