포트폴리오

개요

1. 프로젝트명 : IOCP 기반 GameServer & DBServer
2. 개발 기간 : 2025.04 ~ 2025.07
3. 개발 언어 : C++, Python
4. 시스템 구성: GameServer ↔ DBServer (TCP 통신 기반, 비동기 병렬 처리 구조)
5. 참여 인원 : 1명
6. github : https://github.com/ChoiJaeho180/CPP_Server

구현 내용 요약

1. **GameServer ↔ DBServer 간 비동기 병렬 처리 및 샤드 구조 설계**
2. Memory Pool & Custom Allocator System
3. TaskQueue 기반 게임 로직 분산 처리 구조 설계
4. IOCP 기반 네트워크 입출력 처리
5. CMS 자동화 시스템

세부 구현 내용

GameServer ↔ DBServer 간 비동기 병렬 처리 및 샤드 구조 설계

개요

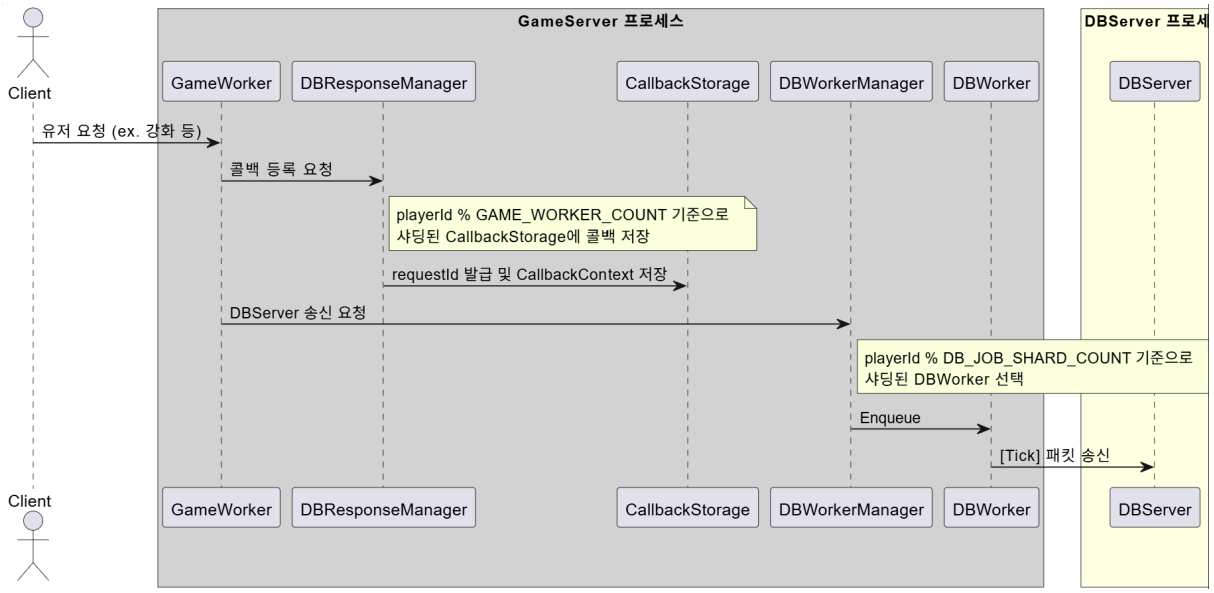
GameServer의 DB I/O 병목을 제거하기 위해 **DBServer를 분리**하고, `playerId` 기반의 샤딩 구조를 통해 **유저 단위 요청은 순차적으로**, 전체 요청은 **병렬로 처리되도록 설계했습니다.**

핵심 설계

- **양방향 비동기 송/수신**
  - GameServer ↔ DBServer 간 통신은 IOCP 기반으로 비동기 처리
- **샤딩 기반 멀티스레드 구조**
  - `playerId % 샤드 수`로 요청을 특정 샤드에 분배하여 병렬 처리
- **순차 처리 보장**
  - 샤드마다 `전용 큐` + `전용 스레드` 스레드로 구성되어, 동일 유저 요청은 순차적으로 처리

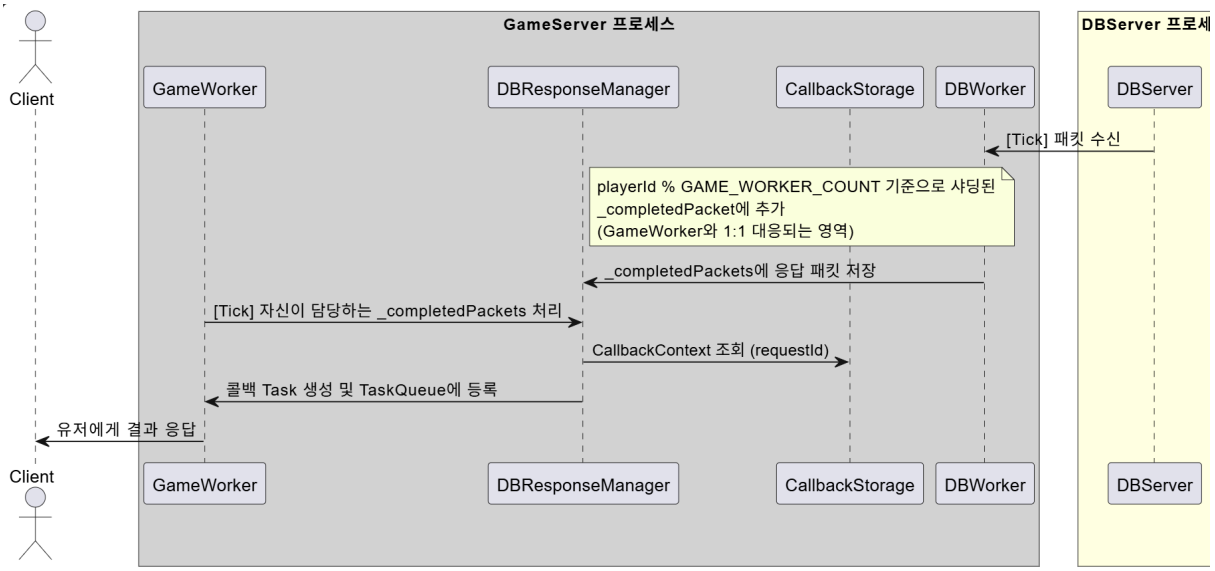
요청 흐름 : GameServer → DBServer

1. `GameWorkerThread` 가 DB 작업 요청 시, `CallbackStorage`에 콜백 함수 등록 및 `requestId` 발급
2. 요청 패킷은 `DBWorkerManager`를 통해 `playerId` 로 샤딩되어, 담당 `DBWorker 큐` 에 패킷 추가
3. `DBWorkerThread` 가 패킷을 DBServer로 비동기 전송



응답 흐름 : DBServer → GameServer

1. **DBServer : 패킷 수신 → DB Query → 패킷 전송 과정**
    a. `NetWorkerThread`가 패킷 수신 후, `PacketWorkerManager`에 전달
    b. `PacketWorkerManager`가 샤딩하여 `PacketWorker` 큐에 등록
    c. `PacketWorkerThread`가 DB 쿼리 실행하고, `응답 패킷` 생성해 `NetWorkerManager`에 전달
    d. `NetWorkerManager` 는 `응답 패킷`을 샤딩하여, `NetWorkerThread` 가 GameServer 송신
2. **GameServer : 응답 패킷 수신 → 게임 로직 반영**
    a. `DBWorkerThread`는 응답 패킷을 샤딩하여, 각 `GameWorkerThread` 전용 응답 큐에 등록
    b. `GameWorkerThread` 응답 큐를 순차적으로 처리하며, 이때 패킷 `requestId`로 콜백 함수를 조회하고 게임 로직에 반영



Memory Pool & Custom Allocator System

개요

- 잦은 객체 생성/삭제로 인한 힙 할당 비용과 메모리 단편화를 줄이기 위해, SList 기반 lock-free 메모리 풀 설계 및 구현했습니다.
- 용도에 따라 `디버깅` , `오버플로 방지` , `재사용 최적화` 등을 위한 전용 할당기를 분리 설계했습니다.

핵심 설계

- **정렬된 고정 크기 기반 메모리 풀**
  - 각 풀은 `정렬된 고정 크기 블록을 재사용` 하여 외부 단편화를 줄이고 할당 속도 향상
  - 요청 크기를 기준으로 정해진 단위(32/128/256 등)로 분류하여 내부 단편화 감소
- **SLIST 기반 Lock-Free 구조**
  - Windows의 `InterlockedPushEntrySList` 등 사용
  - **멀티스레드 환경에서의 락 경합없이 빠른 할당/해제를 지원하여 동시성 최적화**
- **Object Pool 기반 객체 관리**
  - `placement new`와 커스텀 소멸자를 통해 생성/해제를 Pool 내에서 직접 관리
  - **운영체제 호출 비용 없이 객체 재사용 가능**
- **Custom Allocator System**

- `StompAllocator` : 오버플로 감지
- `PoolAllocator` : 메모리 풀 기반 할당기
- `StlAllocator` : STL컨테이너에 PoolAllocator 적용

메모리 할당 및 해제 흐름

> 메모리 요청 시 Pool에서 재사용 가능한 블록을 우선 할당하고, 부족 시 OS에서 aligned 메모리를 직접 할당합니다. 해제 시에는 크기 구분 후 Pool 또는 OS에 반환하며, SLIST 기반으로 Lock-Free 처리가 이루어집니다.

TaskQueue 기반 게임 로직 분산 처리 구조 설계

개요

- 공유 자원에 대한 경합과 동기화 부하를 줄이기 위해, 독립 실행 단위인 **ZoneInstance 마다 TaskQueue 를 할당하고, 해당 큐를 단일 스레드가 직렬로 처리하는 구조**를 설계했습니다.

핵심 설계

- **락 최소화 및 순차 실행**
  - 각 ZoneInstance는 전용 큐를 `단일 스레드만 처리`함으로써 경합 최소화 및 순차 실행 보장
- **중첩 실행 방지**
  - `thread_local` 를 활용하여 `단일 스레드`가 여러 큐를 동시에 처리 방지
- **스레드 부하 분산 처리**
  - `할당된 시간 초과` 또는 `중첩 상황` 발생 시, 일감을 `전역 큐`에 등록하여, 여유있는 스레드가 이어서 처리하도록 분산 구조

IOCP 기반 네트워크 입출력 처리

개요

- Windows IOCP를 활용하여 비동기 네트워크 모델을 구현하고, `커서 기반 버퍼 구조`를 도입하여, **락 없이도 효율적인 입출력 처리**와 **메모리 복사 최소화**를 달성했습니다.

핵심 설계

- **비동기 입출력 요청 모델 사용**
  - IOCP를 활용하여 입출력 이벤트를 커널 수준에서 관리하고, 효율적인 멀티스레드 처리를 구현
- **메모리 복사 최소화**
  - `ReceiveBuffer` : `vector<BYTE>`기반 커서 구조로, 읽기/쓰기 포인터만 이동시켜, memcpy를 최소화하여 수신 처리 성능 최적화
  - `SendBuffer`: 미리 할당된 블록에서 일정 크기만큼 예약/사용하여 송신 시 메모리 할당/해제없이 송신 처리 성능 최적화
- **세션 기반 연결 관리**
  - `Session` 단위로 연결 상태와 입출력 버퍼를 관리하고, 안전한 종료를 위해 참조 카운팅과 상태 제어 적용

CMS 자동화 시스템

개요

- 게임 내 설정 데이터를 JSON으로 생성하면, **헤더 생성 → 프로젝트 자동 등록 → 런타임 캐싱까지 전체 프로세스를 자동화**하는 시스템을 구현했습니다.

핵심 설계

- **Desc 헤더 자동 생성 (초기 1회)**
  - JSON 구조를 분석하여 `XXXDesc.h` 구조체 및 직렬화 매크로 자동 생성
- **Visual Studio 프로젝트 자동 반영**
  - `vcxproj`, `filters` 에 새 파일 자동 등록
- **런타임 캐시 시스템**
  - `CmsCached<T>`: 각 CMS 테이블 데이터를 `cmsId` 기준으로 정적 캐싱하여, 빠른 조회

- `CmsManager` : `CmsCached`들을 초기화하고 외부에 접근 인터페이스 제공