로그인, 모니터링, 채팅, 게임 서버 포트폴리오

작성자 : 정찬훈

블로그 : https://vanilla-tech.tistory.com

GitHub : https://github.com/jch5158/GDB

사용 언어 : C/C++

목차

1-1) NetServer 네트워크 라이브러리

- **라이브러리 이름** :
  - **외부망용 서버/클라이언트 네트워크 라이브러리** : CNetServer, CNetClient
  - **내부망용 서버/클라이언트 네트워크 라이브러리** : CLanServer, CLanClient

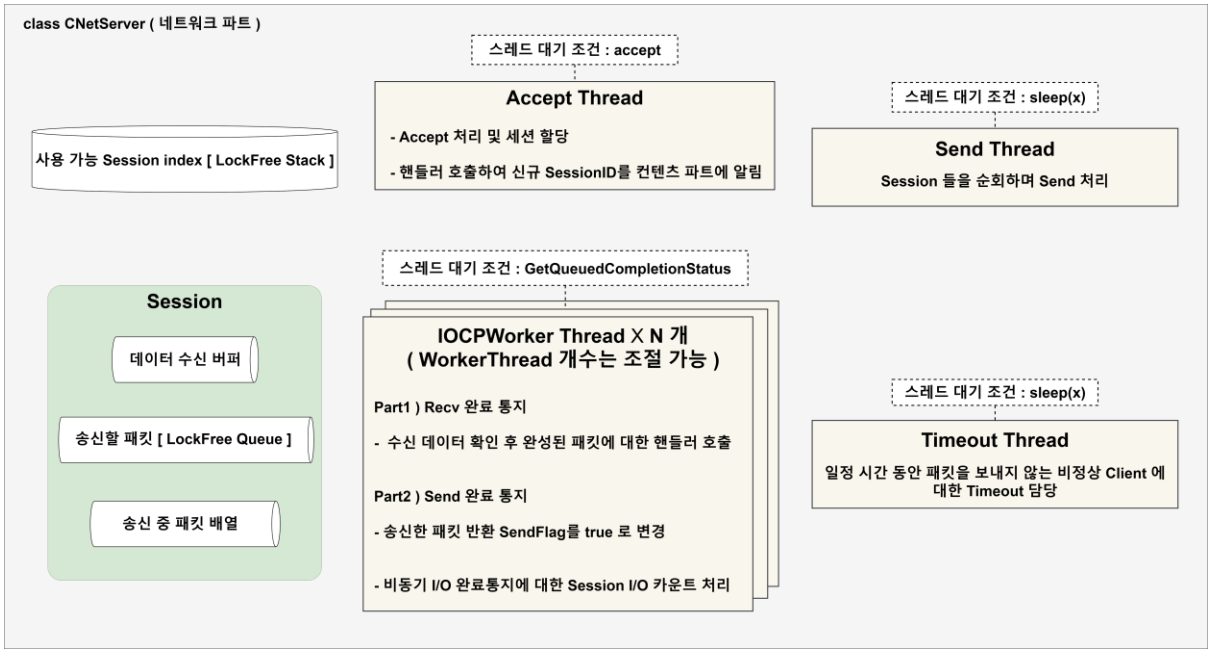
- **사용 모델** : Windows IOCP

- **사용처 :** 로그인 서버, 모니터링 서버, 채팅 서버

- **구현 목적 :**

  해당 네트워크 라이브러리 제작 목적은 네트워크 파트와 이를 상속받은 컨텐츠 파트를 분리하여 sessionID 라는 매개체를 통해 패킷 송.수신을 처리하여 프로젝트의 개발 및 유지 보수 용이성을 증대 시키기 위함

NetServer 스레드 구조도



부가 설명

- sessionID의 상위 2Byte는 index, 하위 6Byte는 clientID
- I/O Count 를 이용해서 세션 종료 여부를 결정
- CAS를 이용한 Session 반환 동기화
- 메시지(패킷) 참조 카운트에 의해서 관리
- Send, Recv 1회 제한

장점

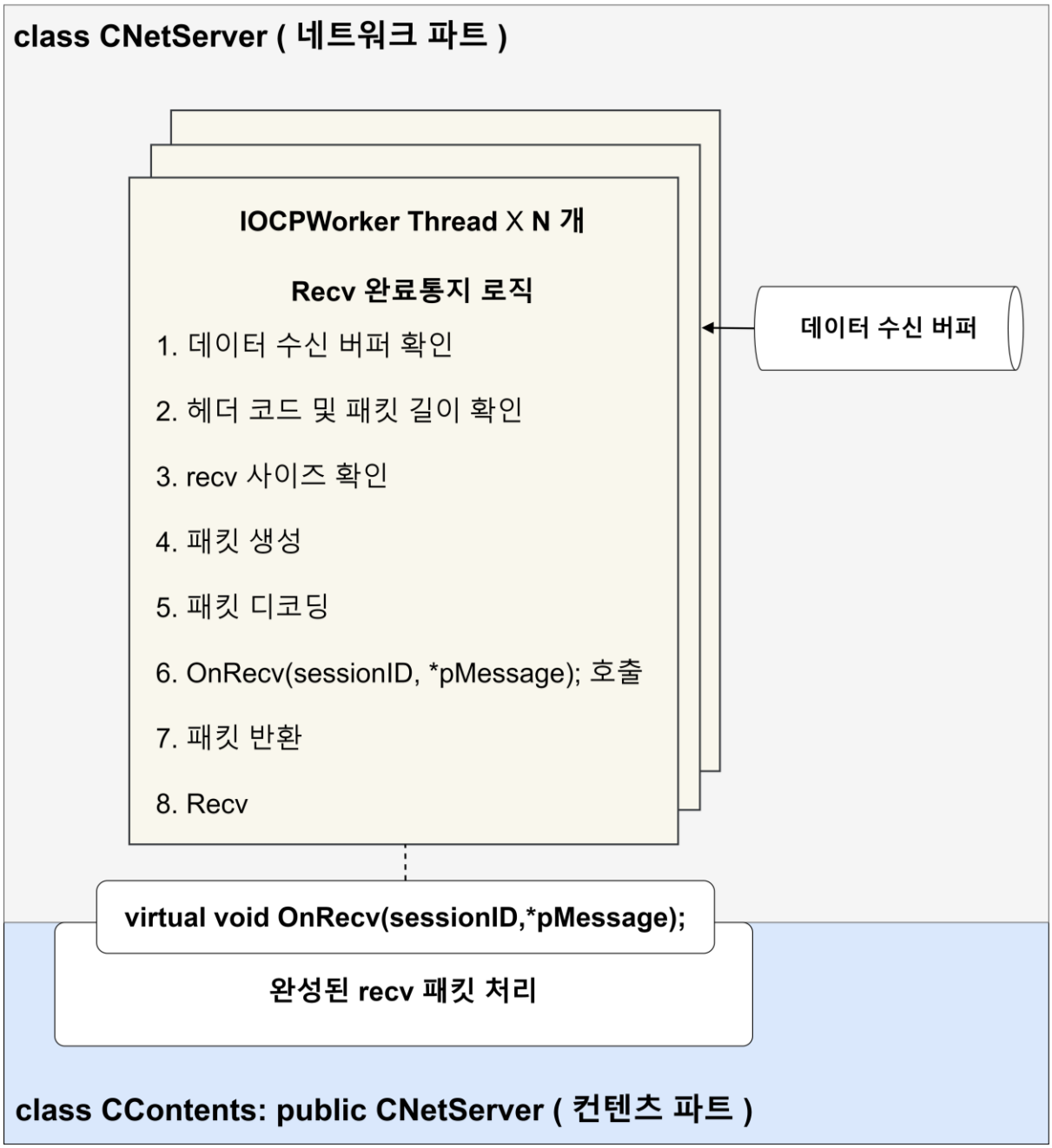
- 컨텐츠 파트와 네트워크 파트를 완전히 분리하여 유지보수 용이성 증대
- 컨텐츠 스레드 구성의 제한이 없음

단점

- 컨텐츠 파트와 네트워크 파트간의 요청과정에서의 오버헤드

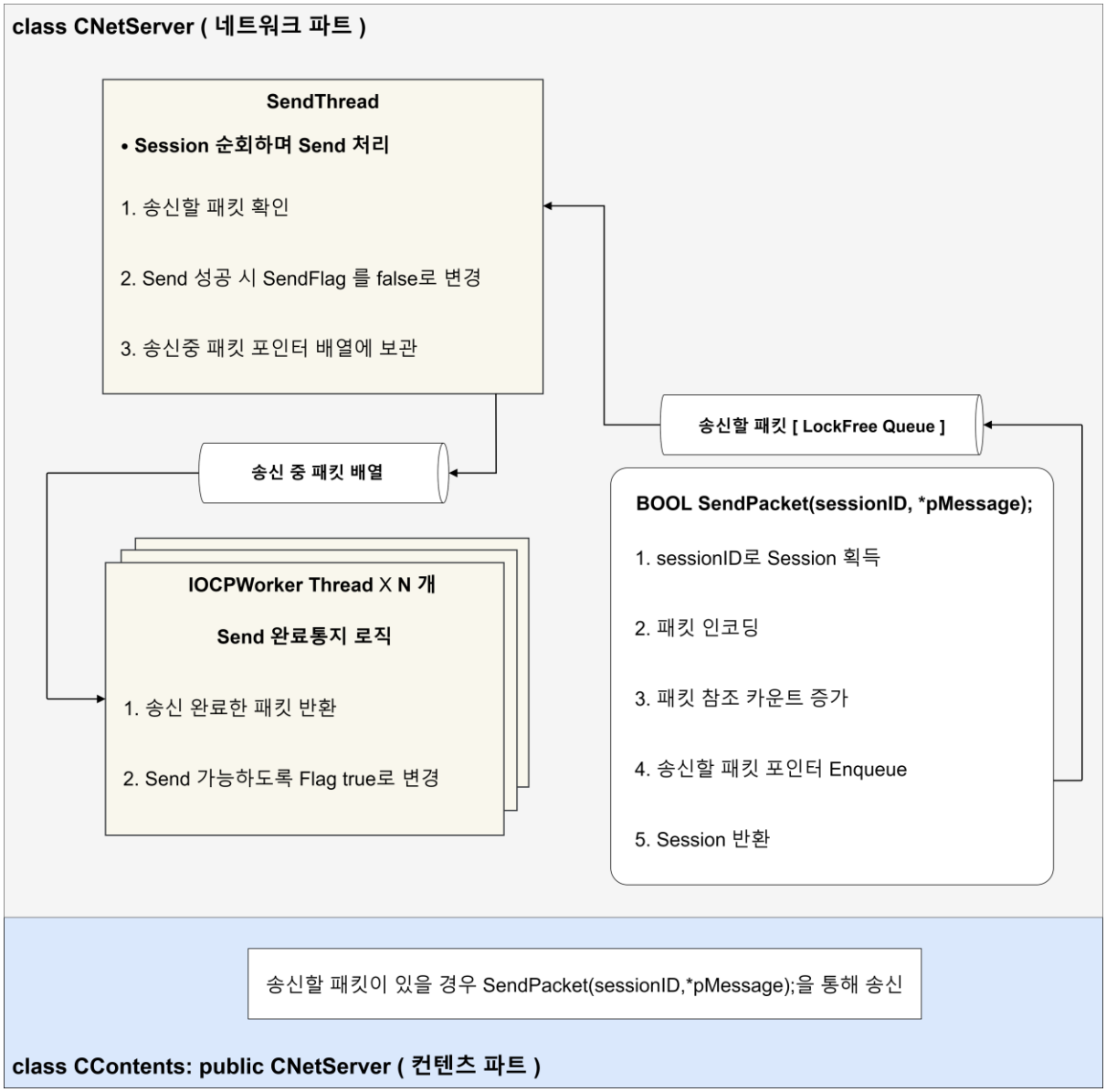
NetServer Recv 도식화

- 수신 버퍼 확인 후 패킷 완성되면 OnRecv 핸들러 호출하여 컨텐츠측으로 수신 패킷 전달



NetServer Send 도식화

- SendThread에서 주기적으로 Session들을 확인하여 sendFlag 확인 후 Send 처리
- Send 처리 후 송신 중 패킷은 배열에 보관



AcceptEx

AcceptEx는 비동기 방식을 이용하여 Accept를 처리하는 방식입니다. GQCS(); 를 호출한 IOCPWorkerThread를 통해 완료통지를 받기 때문에 여러 스레드에서 동시에 Accept를 처리할 수 있습니다.

AcceptEx 사용방법

- WSAIoctl 함수를 통해 AcceptEx 함수 포인터를 return 받는다.
- 세션 소켓과 Overapped, 연결 정보를 얻기위한 Buffer를 비동기 Accept 등록
- 완료통지가올 경우 등록한 Buffer를 통해 연결정보를 획득
- 연결 종료시 DisconnectEx 후 다시 비동기 Accept 등록

AcceptEx 장점

- 병렬로 Accept를 처리하기 때문에 처리율이 Accept보다 높다.
- 소켓을 미리 생성하여 재사용하기 때문에 소켓 생성, 삭제로 인한 오버헤드 감소

AcceptEx 단점

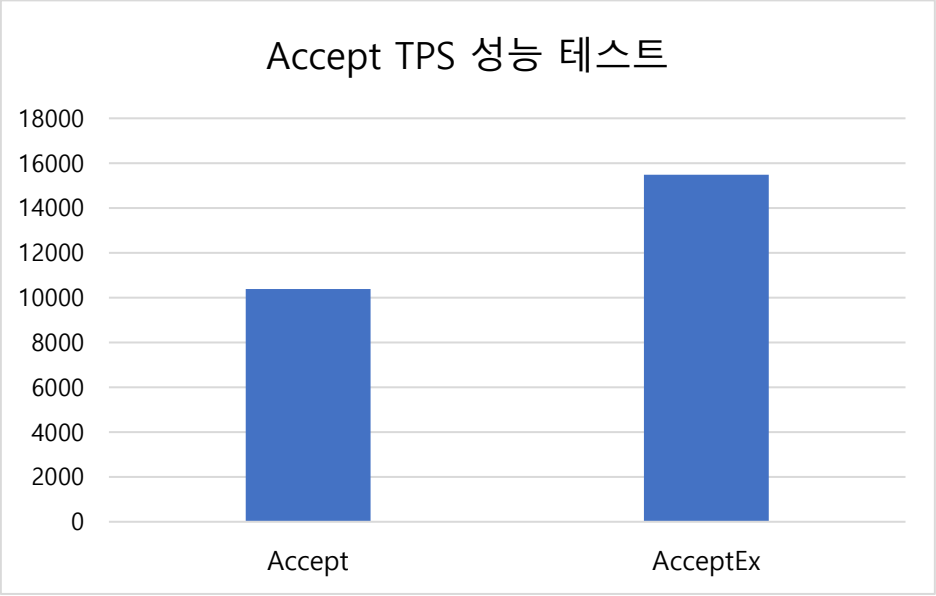
- Accept를 사용하는 것에 비해 구조가 복잡
- AcceptEx 에 대한 완료통지에서 CompletionKey는 사용 불가능

AcceptEx 성능 테스트

AcceptEx와 Accept의 성능을 확인하기 위해서 아래 방법으로 테스트를 진행하였습니다.

테스트 방법

1. 1개의 스레드당 5000개의 소켓을 생성 후 동기로 Connect
2. 5개의 스레드를 생성하여 총 25000명 Connect
3. 각각의 스레드가 5000개의 소켓으로 Connect 후 1초 뒤에 closesocket(); 후 재연결
4. 일정시간 후 Accept TPS의 평균을 구한다.



테스트 결과 AcceptEx가 더 좋은 결과를 보여준 것으로 확인되었습니다. 하지만, Accept 또한 1만 TPS가 나온 것과 구조의 심플함을 고려하였을 때 Accept를 사용하는 것도 문제가 되지 않을 것으로 생각됩니다.

1-3) MMOServer 네트워크 라이브러리

- **라이브러리 이름 :** CMMOServer

- **사용 모델 :** Windows IOCP

- **사용처 :** MMORPG 게임 서버

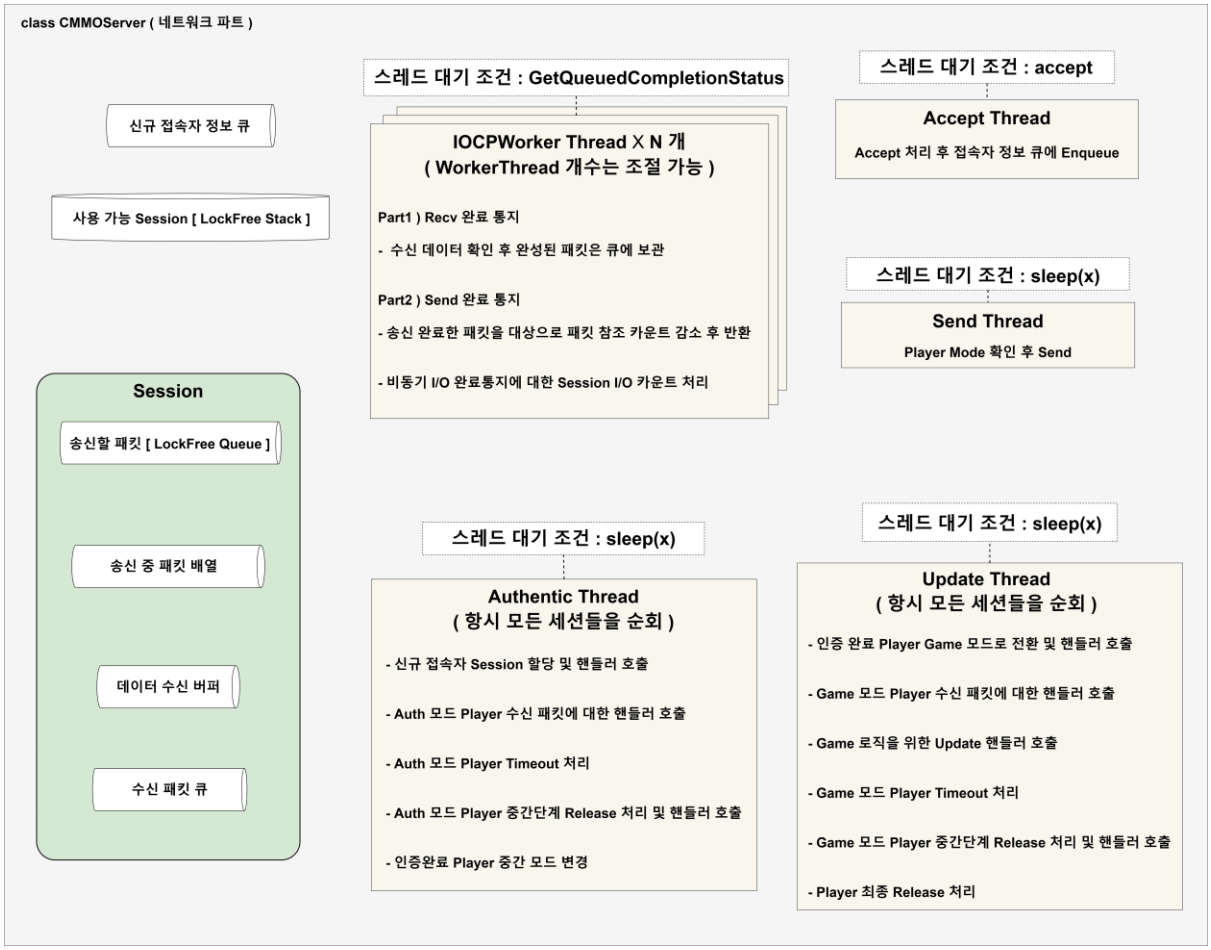
- **구현 목적 :**

  MMOServer는 MMORPG 게임 전용 라이브러리로서 NetServer 의 Session 검색 과정을 없애고 항시 루프를 통해 모든 세션을 대상으로 처리해야 할 작업이 있는지 확인,

  스레드가 항시 모든 세션을 대상으로 처리해야 할 작업이 있는지 확인하기 때문에 **현재 서버에 접속한 클라이언트가 최대치에 가까울 때 최고의 성능을 발휘할 수 있는 네트워크 라이브러리**

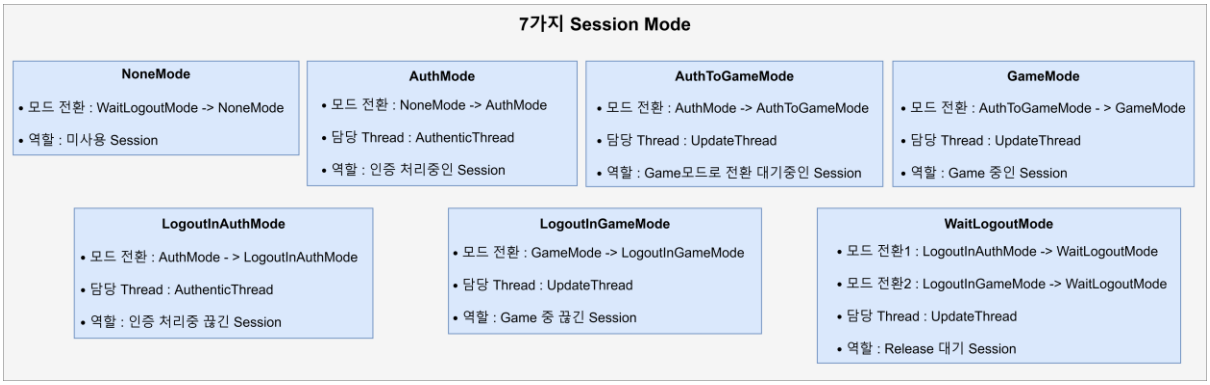
MMOServer 스레드 구조도

- Authentic Thread, UpdateThread는 항시 세션 포인터 배열을 순회하여 로직 처리



Session 모드

- 각 Session 모드의 전환 과정 및 담당 스레드



부가 설명

- 각각의 스레드는 세션 Mode 상태를 확인하여 이에 맞는 로직을 처리
- 락을 사용하지 않기에 Mode 변경 시 AuthMode -> AuthToGameMode -> GameMode 와 같은 절차를 통해 Session Mode를 변경
- Session 포인터 배열에 Session 클래스를 상속받은 Player 포인터 셋팅
- 메시지 참조 카운트에 의해서 관리
- Send, Recv 1회 제한

장점

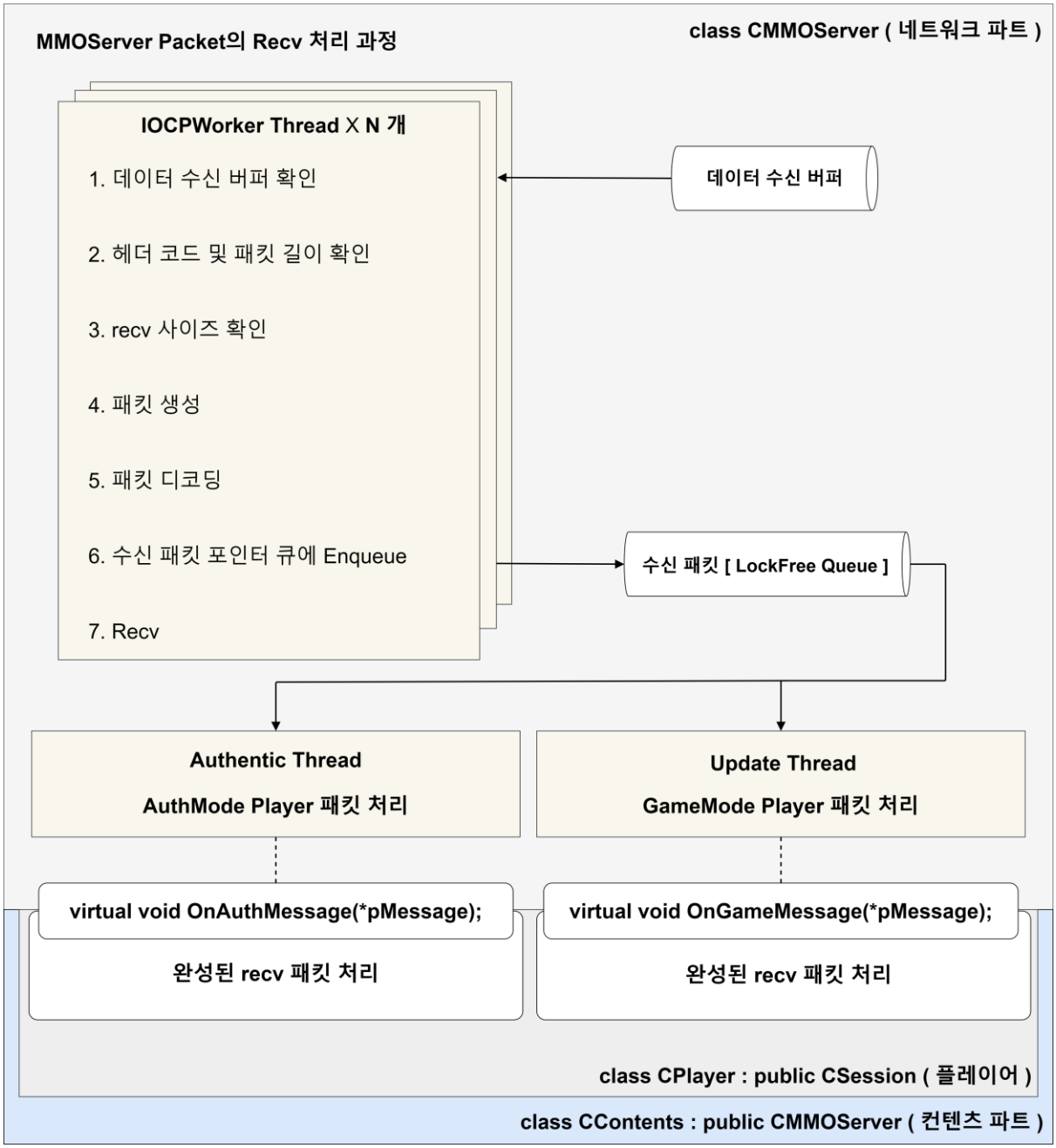
- Session 검색이 필요하지 않는 처리
- 상속받은 클래스에서 인증 스레드 및 게임 스레드를 생성할 필요가 없음
- 접속자가 최대일 때 최고의 효율성

단점

- 항시 준비된 세션들을 대상으로 한 루프로 CPU 사용률이 높음
- MMORPG 전용 네트워크 라이브러리
- 추가적인 컨텐츠 스레드 생성의 제약

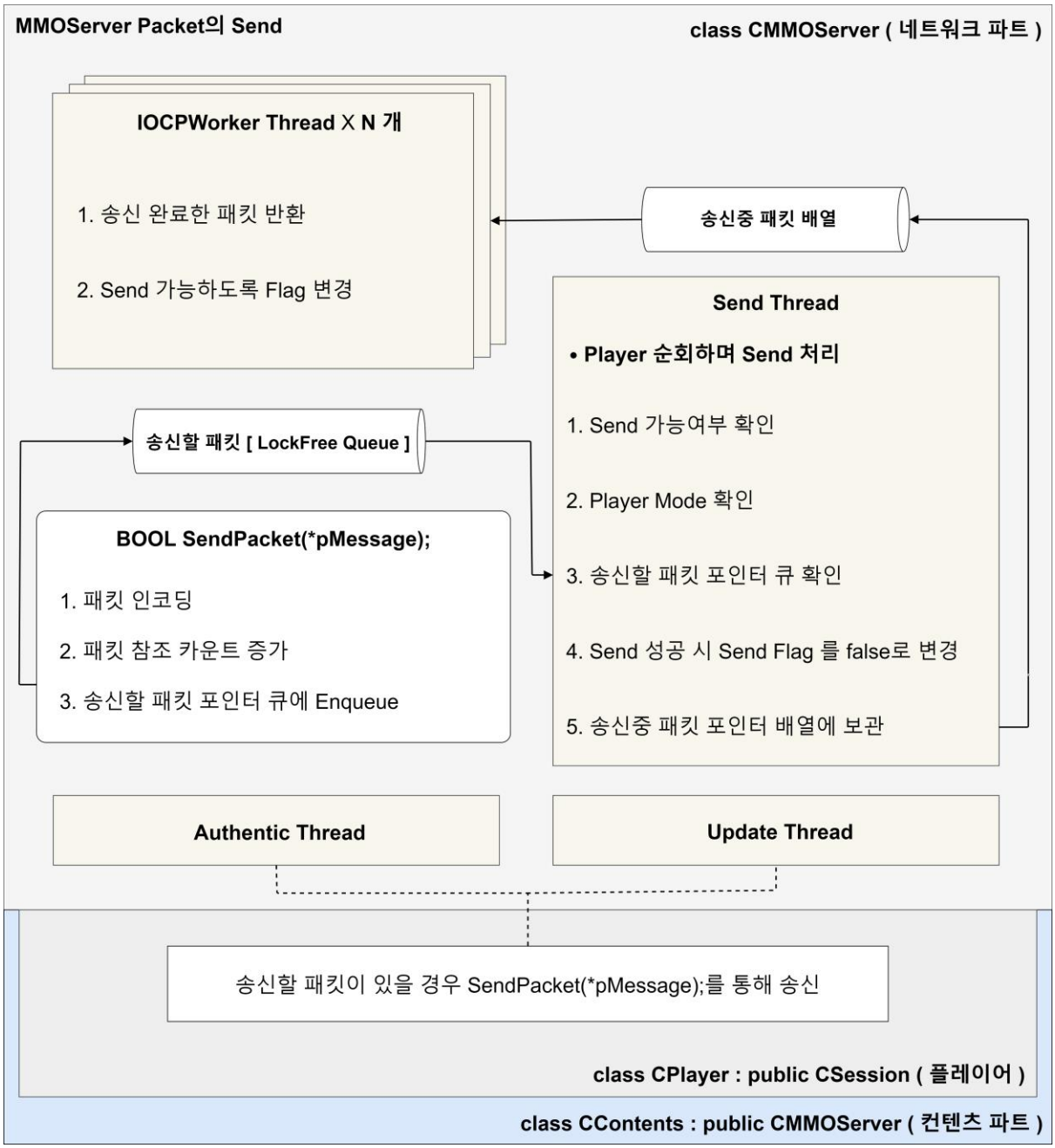
MMOServer Recv 도식화

- IOCPWorkerThread 는 완성된 수신 패킷을 큐에 Enqueue
- AuthenticThread, UpdateThread는 Player의 Mode 확인 후 핸들러를 통한 수신 패킷 처리



MMOServer Send 도식화

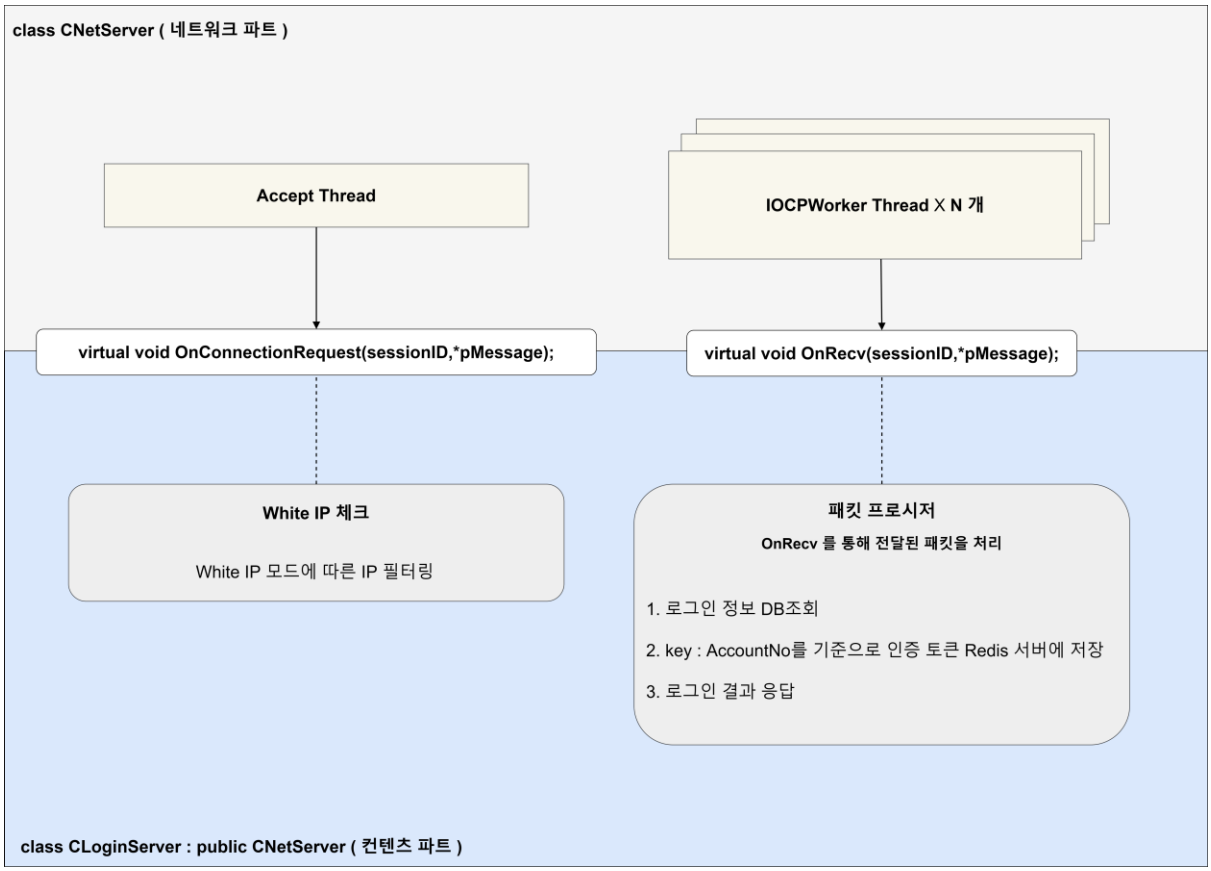
- SendPacket(); 은 컨텐츠에서 요청한 패킷을 큐에 Enqueue
- SendThread 는 일정 주기로 Player의 Mode 확인 후 송신할 패킷 Send 시도



로그인 서버

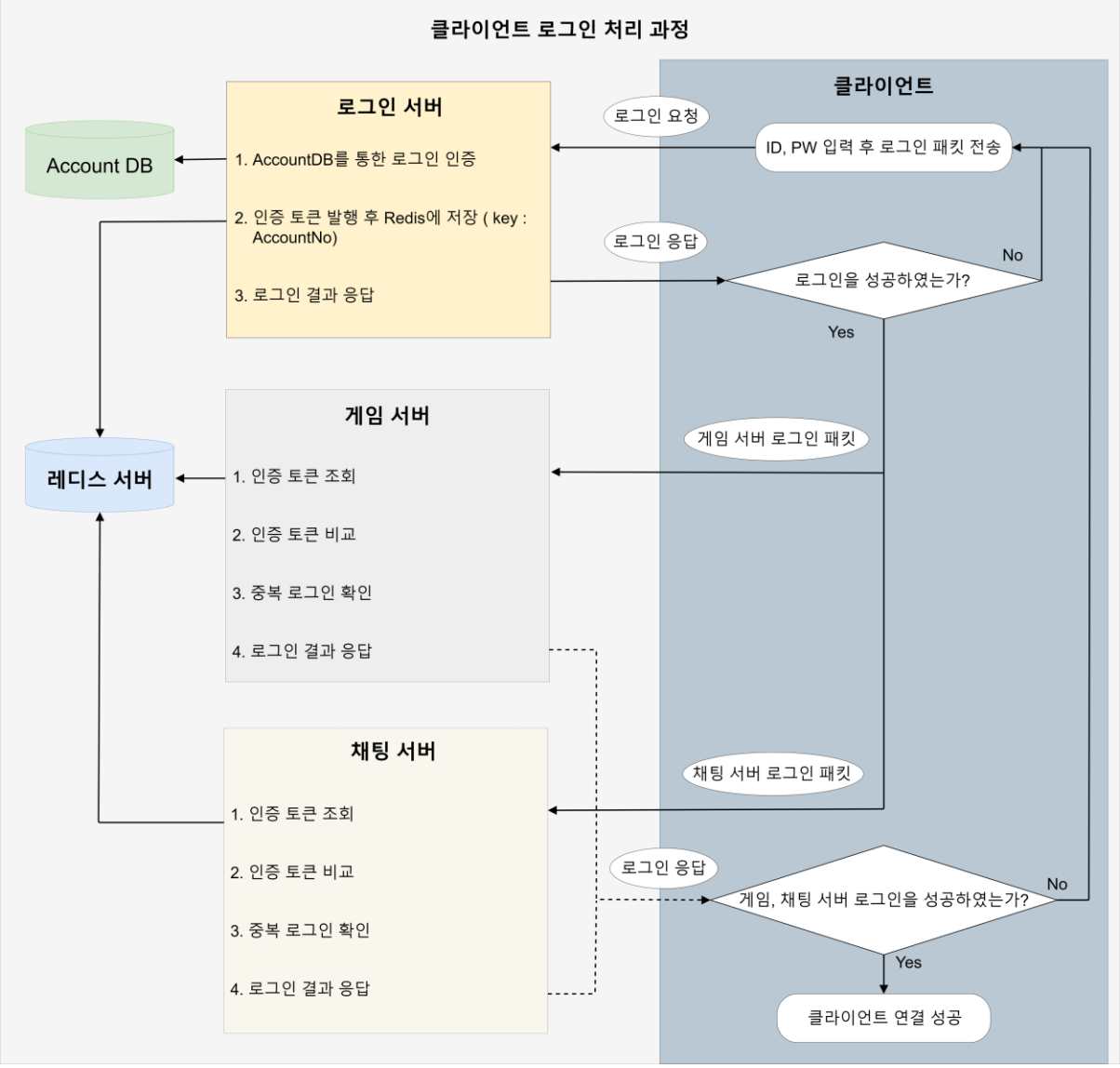
클라이언트의 ID, PASSWORD 를 DB에서 조회하여 올바른 로그인 정보인지 확인 후 로그인 정보가 DB에 있는 정보와 일치한다면 인증 토큰을 발행하여 AccountNo를 key로 하여 레디스 서버에 저장

로그인 서버 스레드 구조도



로그인 처리는 연결된 세션들간의 상호작용이 없이 독립적인 처리를 하기 때문에 Recv 완료통지를 받은 WorkerThread가 패킷을 직접 처리하도록 구성하였습니다.

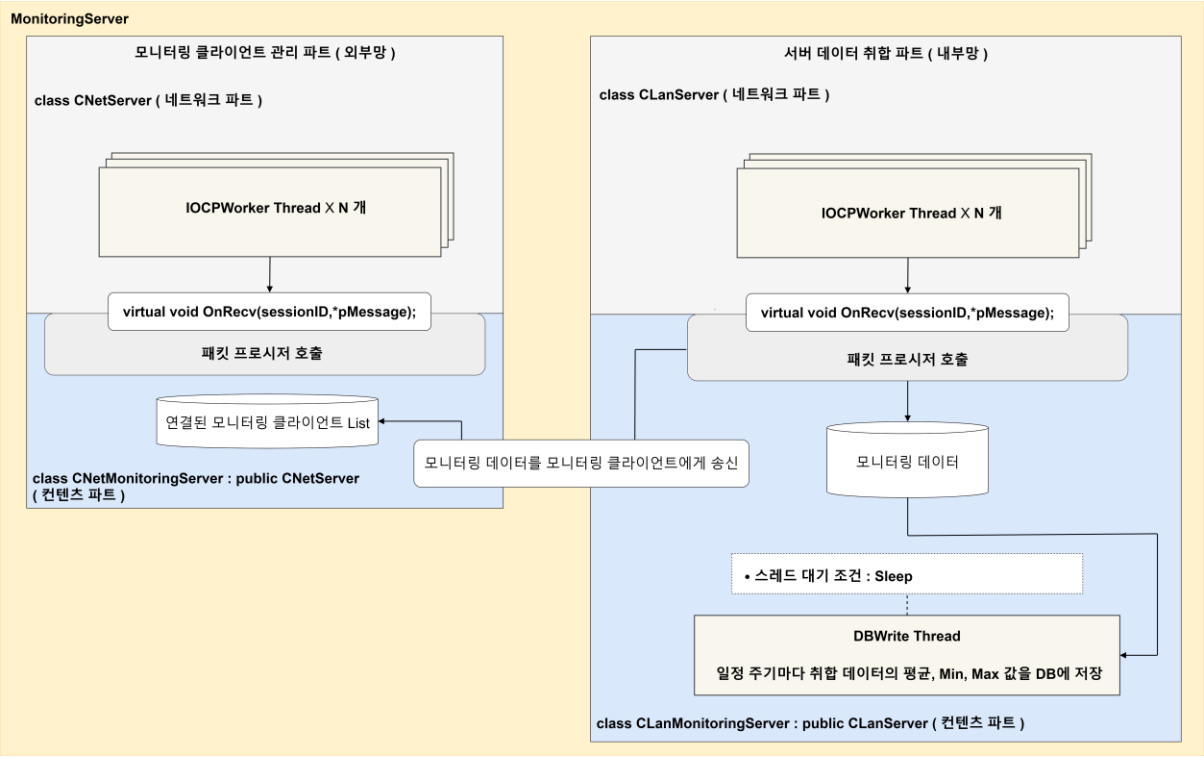
로그인 서버를 이용한 게임, 채팅 서버 로그인 과정 도식화



모니터링 서버

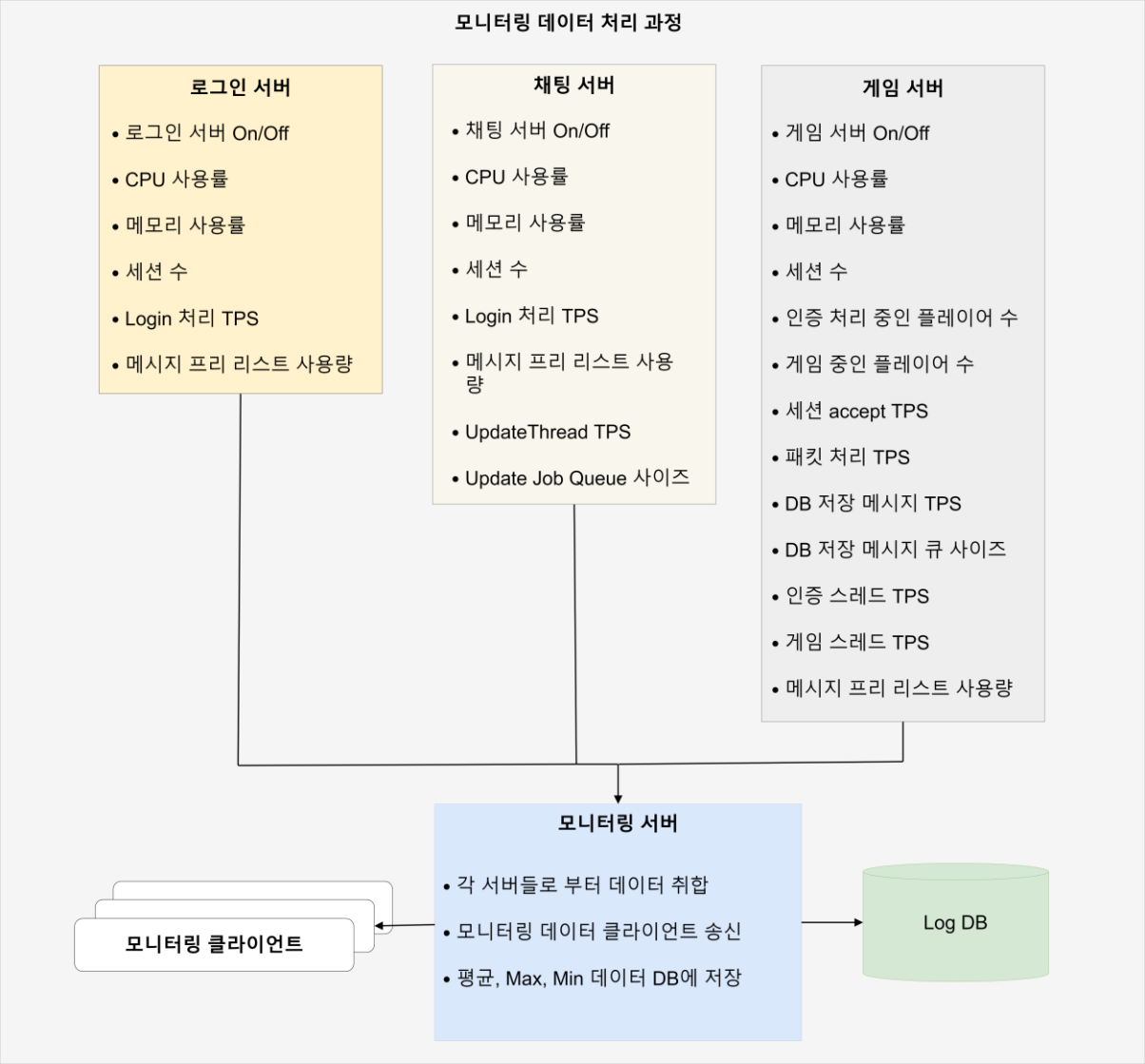
- 모니터링 클라이언트는 특정 토큰 값을 이용하여 모니터링 서버에 로그인
- 로그인 서버, 채팅 서버, 게임 서버의 상태 데이터를 취합하여 모니터링 클라이언트에게 송신
- 일정 주기마다 수집했던 수집 데이터의 Max, Min, 평균을 DB에 저장
- LanServer, NetServer를 이용하여 내부망, 외부망 통신을 구분

모니터링 서버 스레드 구조도



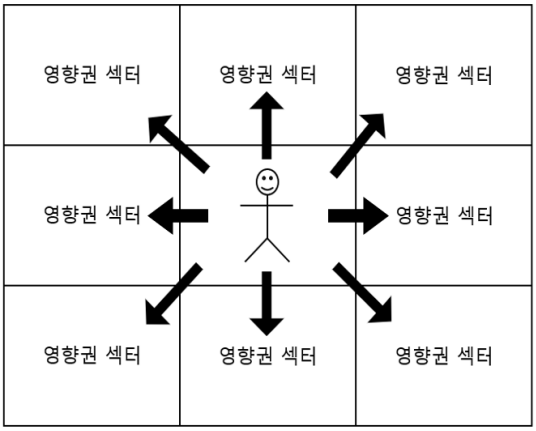
모니터링 데이터 처리는 연결된 세션들간의 상호작용이 없이 독립적인 처리를 하기 때문에 Recv 완료통지를 받은 WorkerThread가 패킷을 직접 처리하도록 구성하였습니다.

모니터링 데이터 처리 도식화



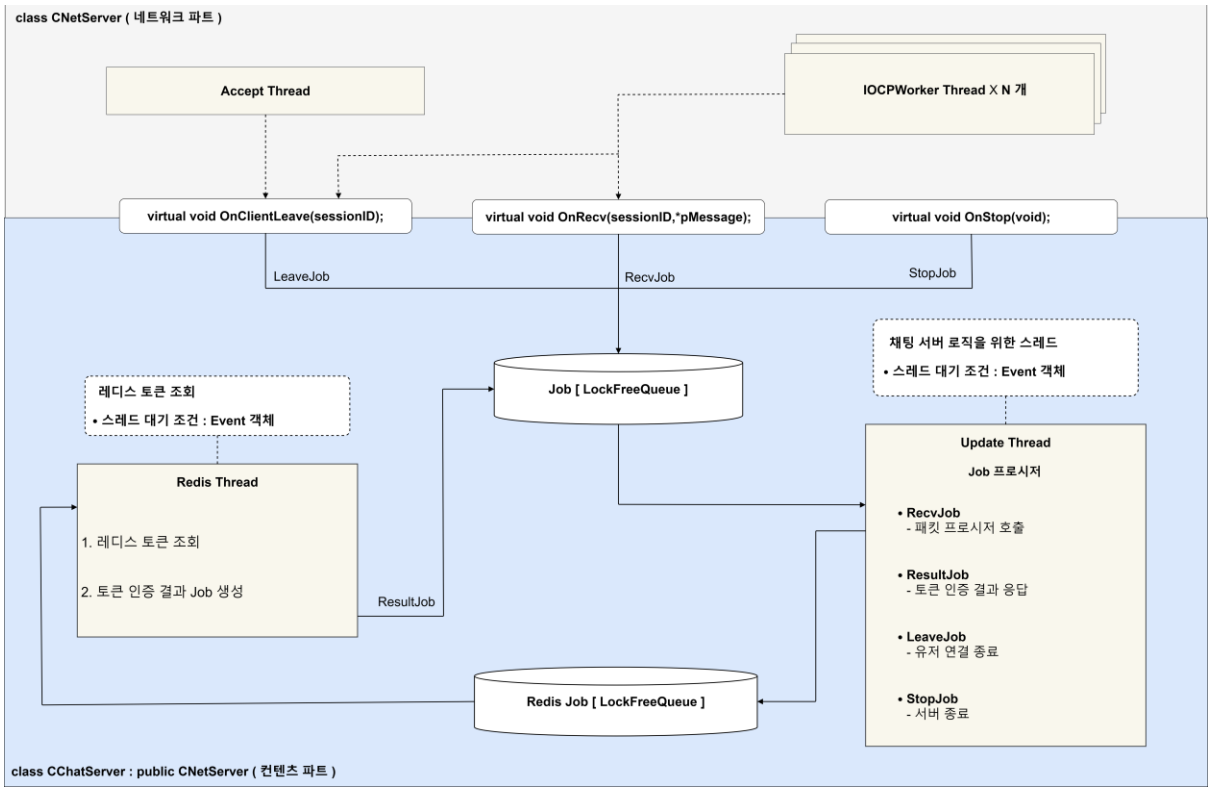
MMORPG 채팅 서버

- NetServer 네트워크 라이브러리 사용
- MMORPG 채팅 서버이기에 영향권 섹터를 기반으로한 컨텐츠 처리



(Player의 영향권 섹터입니다.)

채팅 서버 스레드 구조도

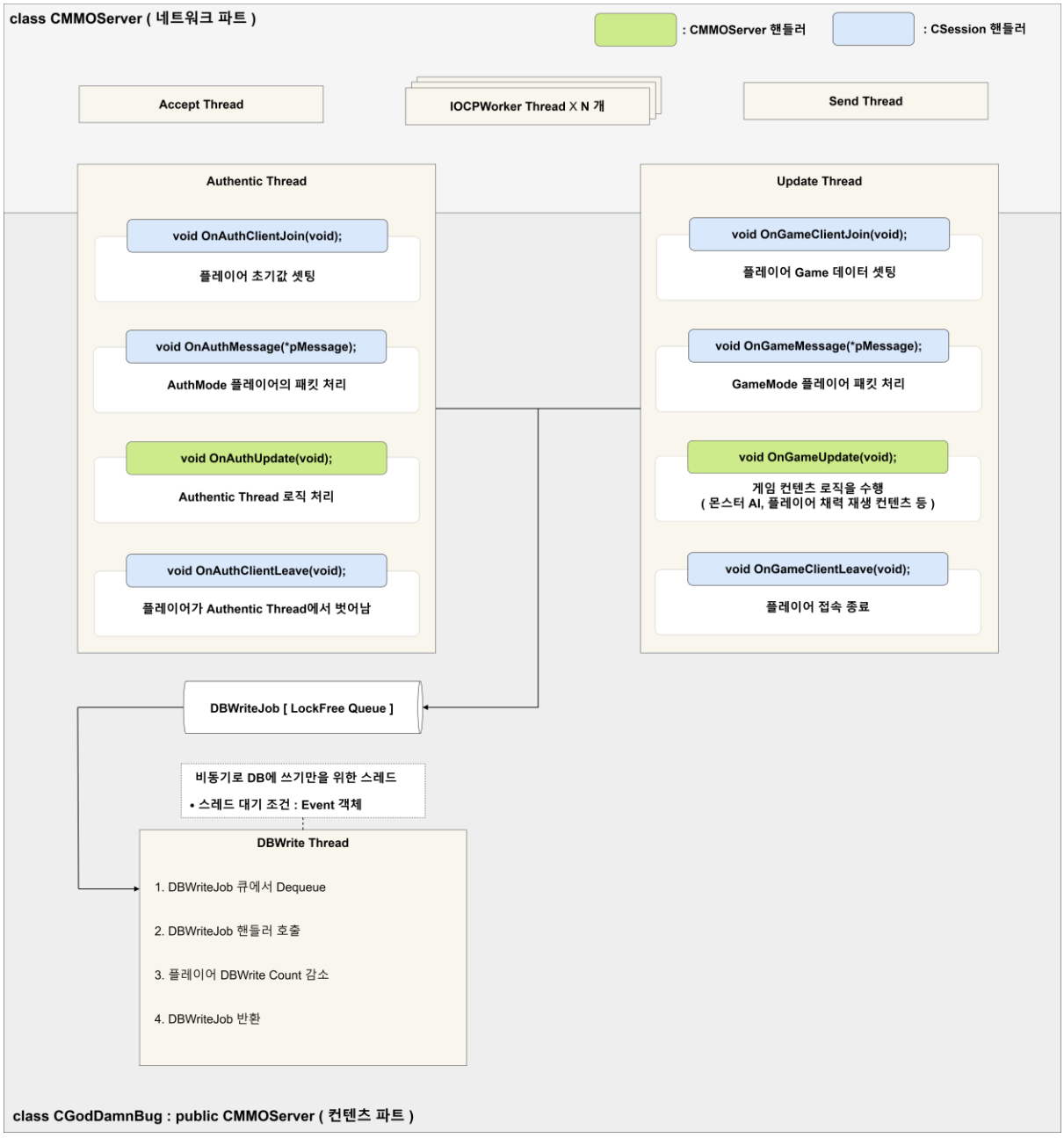


멀티 스레드 구조에서는 Player 이동에 따른 섹터 동기화가 필요하기 때문에 동기화 작업이 필요 없는 싱글 스레드 구조로 구성

MMORPG 게임 서버

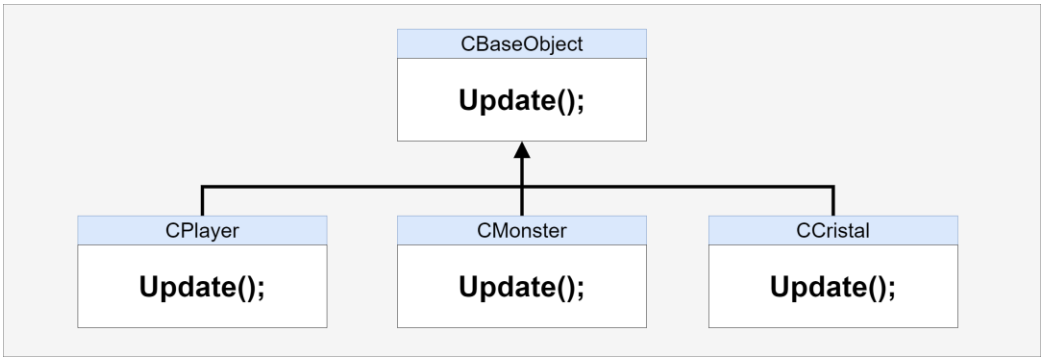
게임 서버는 MMOServer 네트워크 라이브러리를 이용하여 제작하였습니다.

게임 서버 스레드 구조도



게임 오브젝트

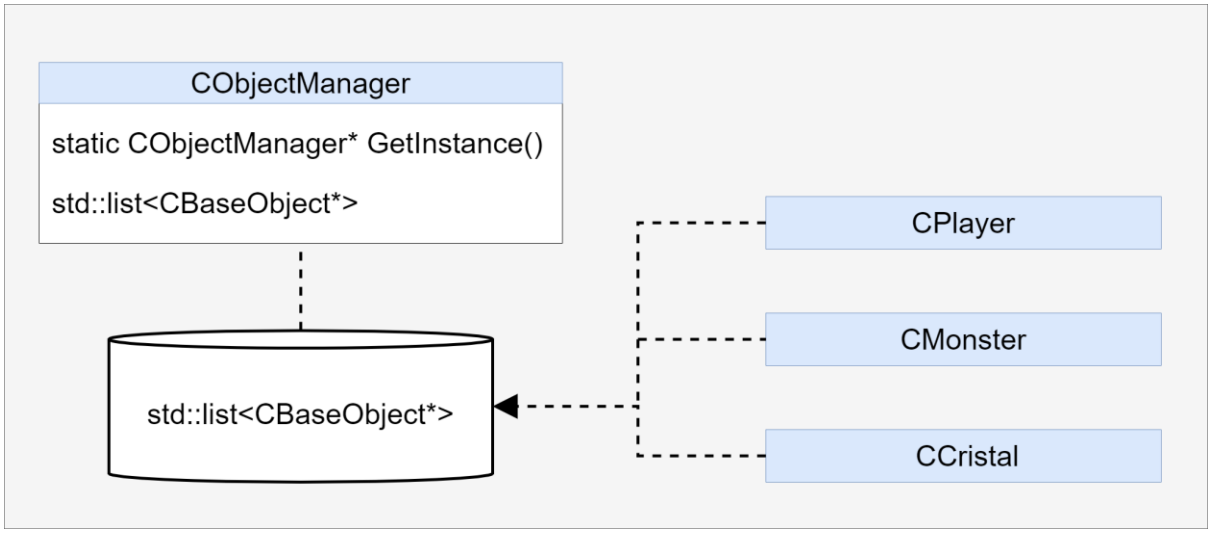
컨텐츠에 사용될 게임 오브젝트는 CBaseObject 추상 클래스를 상속받아 Update 핸들러를 필수적으로 오버라이딩 하도록 하였습니다.



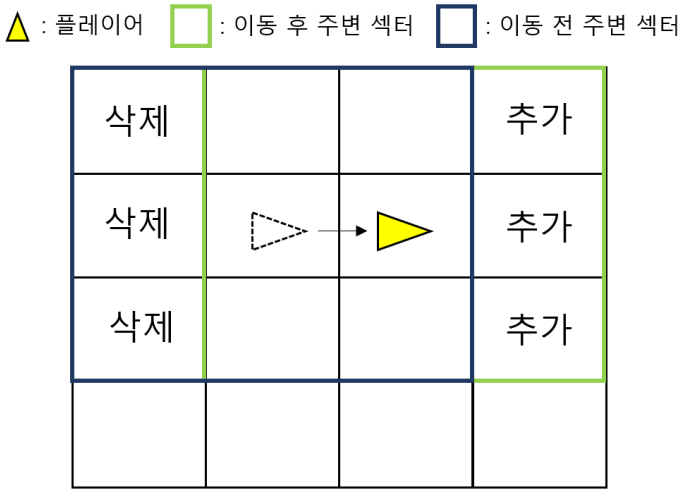
게임 오브젝트 관리

생성된 게임 오브젝트들은 **싱글톤 패턴**으로 구현된 CObjectManager의 std::list<CBaseObject*>에 등록하여 관리합니다.

CObjectManager는 std::list<CBaseObject*>에 등록된 오브젝트들을 순회하면서 Update() 핸들러를 호출합니다.



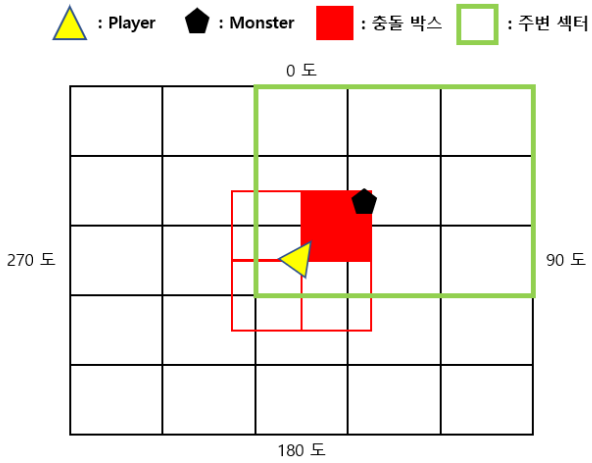
플레이어 이동



1. 플레이어가 이동할 때마다 클라이언트가 일정간격으로 이동 패킷을 송신
2. 서버는 수신 받은 이동 패킷을 주변섹터에 송신
3. 섹터 위치 수정 후 추가 및 삭제된 섹터에 오브젝트 생성 및 삭제 패킷 송신

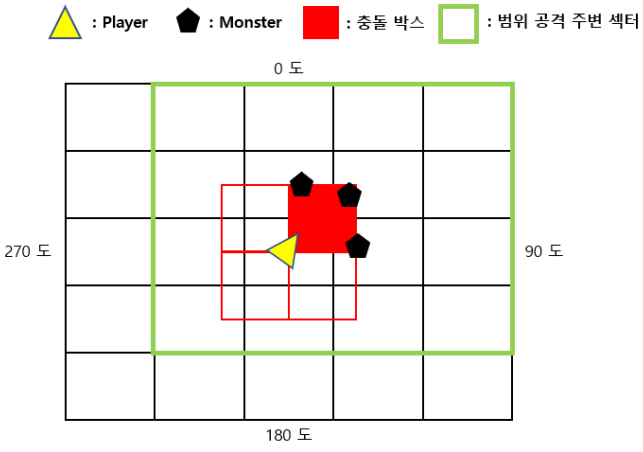
플레이어 공격

- **기본 공격**



1. 플레이어 주변 섹터에 공격 액션 패킷을 송신
2. 플레이어 방향을 기준으로 충돌박스 범위 내 몬스터 검색
3. 데미지 계산 후 몬스터 주변 섹터에 데미지 패킷 송신
4. 몬스터가 사망하였다면 몬스터 주변 섹터에 몬스터 사망 패킷 송신

- **범위 공격**



1. 플레이어 주변 섹터에 공격 액션 패킷을 송신
2. 플레이어 방향을 기준으로 충돌박스 범위 내 충돌 몬스터 3마리 검색
3. 데미지 계산 후 플레이어 방향을 기준으로 4 X 4 주변 섹터에 데미지 패킷 송신
4. 몬스터가 사망하였다면 4 X 4 주변 섹터에 몬스터 사망 패킷 송신

몬스터 로직

- **몬스터 이동**

  이동 후 X = 몬스터 X + cosf(몬스터 방향) * 몬스터 이동거리;
  이동 후 Y = 몬스터 Y + sinf(몬스터 방향) * 몬스터 이동거리;

  1. 위의 식을 이용하여 이동 좌표를 얻은 후 주변 섹터에 이동 패킷 송신
  2. 추가 및 삭제된 섹터에 몬스터 생성 및 삭제 패킷 송신

- **공격한 플레이어 위치로 이동**

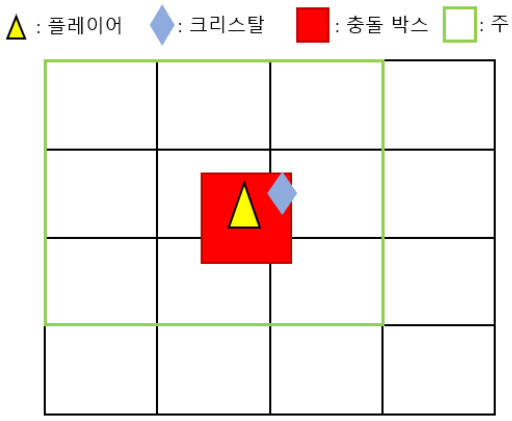
  몬스터 이동 방향 = atan2(플레이어 Y - 몬스터 Y , 플레이어 X - 몬스터 X);

  1. 위의 식을 이용하여 공격한 플레이어로 향하는 방향을 얻음
  2. 방향 정보를 이용하여 이동 좌표를 얻은 후 주변 섹터에 이동 패킷 송신
  3. 추가 및 삭제된 섹터에 몬스터 생성 및 삭제 패킷 송신

- **공격한 플레이어 공격**

  1. 공격한 플레이어가 공격 범위 내에 있는지 확인
  2. 몬스터 주변 섹터에 공격 액션 패킷을 뿌림
  3. 데미지 계산 후 플레이어 주변 섹터에 데미지 패킷 송신

아이템 줍기

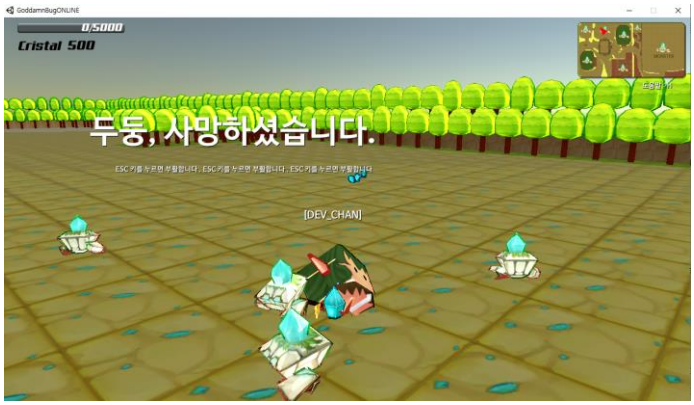


1. 플레이어 주변 섹터에 아이템 줍기 액션 패킷 뿌림
2. 충돌 박스 내에 크리스탈 검색 후 플레이어 크리스탈 카운트 증가
3. 플레이어 주변 섹터에 크리스탈 줍기 패킷 송신

플레이어 휴식

1. 일정시간 간격으로 플레이어 HP 회복
2. 몬스터 공격 및 플레이어가 움직일 경우 해제 후 HP 보정 패킷 송신

플레이어 사망



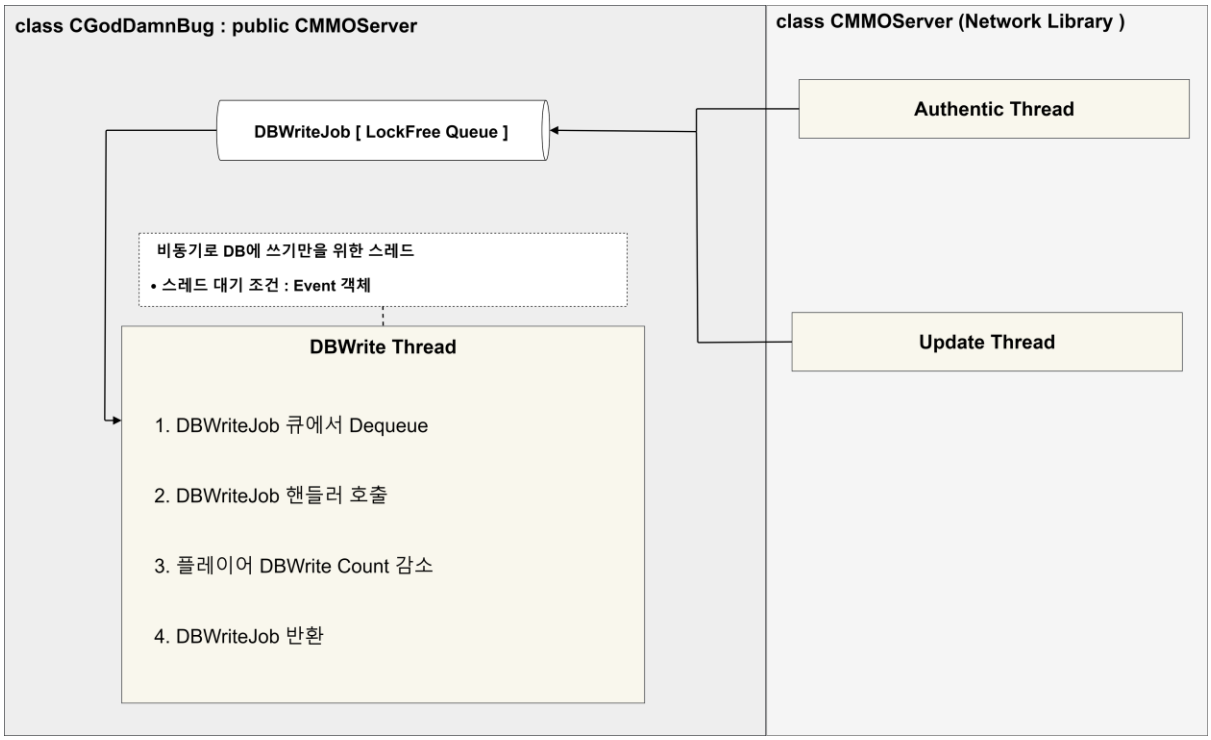
1. 플레이어 사망 시 주변 섹터에 플레이어 사망 패킷 송신
2. 보유 크리스탈 감소 후 죽은 자리에 크리스탈 생성 패킷 송신

플레이어 재시작

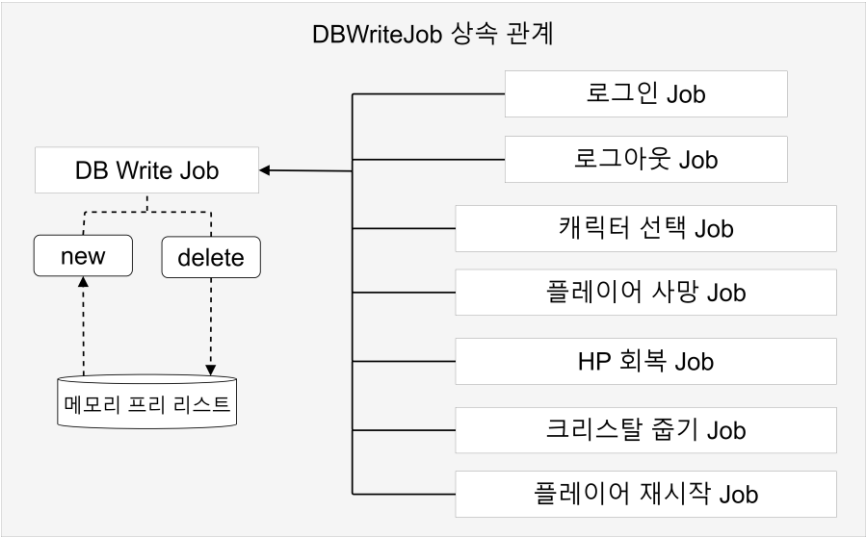
1. 주변 섹터에 플레이어 오브젝트 삭제 패킷 송신
2. HP리셋 후 리스폰 좌표로 섹터 셋팅
3. 주변 섹터에 플레이어 오브젝트 생성 패킷 송신

비동기 DB Write Thread

- Authentic Thread, Update Thread 블락 방지를 위해 DB Write 작업은 비동기로 처리

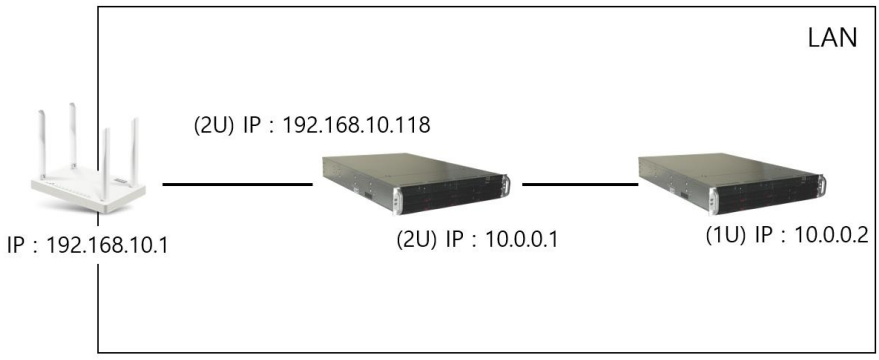


DB Write 클래스



- 다형성을 이용한 DB Write 처리
- 메모리 프리리스트를 이용한 new, delete 오버로딩

서버 테스트 환경



- 물리적으로 분리되어 있는 서버 2대를 직접 연결하여 테스트를 진행

- Windows 서버 OS를 사용하여 서버가 바인딩할 포트를 수동으로 인 바운드 규칙에 등록

- 2U 는 채팅, 게임, 로그인, 모니터링 서버를 위한 서버

- 1U 는 더미 클라이언트 프로세스 전용 서버

물리적으로 서버가 분리된 테스트 환경의 이점

- 제작한 네트워크 라이브러리는 해당 환경에서 7일간 Echo방식으로 스트레스 테스트를 통해서 네트워크 라이브러리를 검증 후 컨텐츠를 제작

- 네트워크 라이브러리 스트레스 테스트 도중 발생한 TCP의 혼잡제어 연결 끊김 현상을 경험하고 이로 인한 에러 코드[ERROR_SEM_TIMEOUT(121)] 를 확인

- Send, Recv 각각 180만 ~ 200만 TPS 정도 나오는 상황에서 너무 많은 트래픽량으로 인해 이더넷 자체가 다운되는 상황을 경험

채팅 서버 테스트

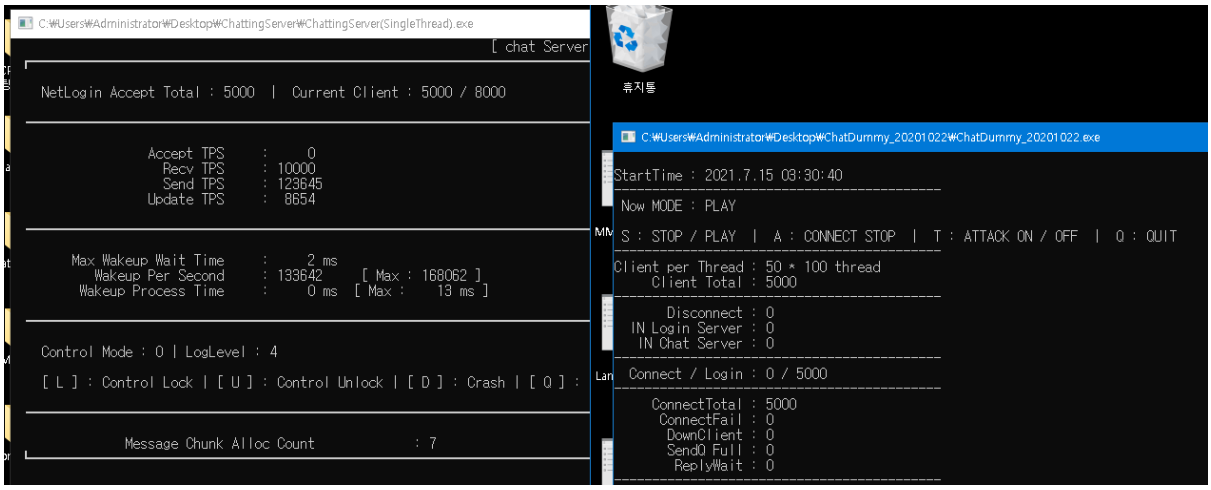
- 채팅 서버의 정상작동 확인을 위해 아래와 같이 테스트를 진행하였습니다.

채팅 서버 테스트 방법

1. 1만 5천명의 더미 클라이언트를 연결
2. 랜덤한 확률로 채팅 보내기 및 섹터 이동 패킷 송신
3. 연결 끊김 없이 테스트 7일, 랜덤한 확률로 연결 끊기 테스트 2~3일

채팅 테스트 통과 조건

- 더미 클라이언트 의도와 상관없이 세션이 끊기는 현상이 없어야 함
- 총 연결 클라이언트 수와 채팅 서버에서 Accept한 클라이언트 수가 동일해야 함
- Update TPS 1.5만 TPS 이상 나와야 함
- 더미 클라이언트가 수신받지 못한 패킷이 있으면 안 됨
- 더미가 송신한 패킷에 대한 응답이 50ms 이하로 나와야 함



(좌측 콘솔 : 채팅 서버, 우측 콘솔 : 더미 클라이언트)

더미 클라이언트는 **총 연결 클라이언트 수, 연결 실패, 연결 끊김, SendQ Full 횟수, 수신받지 못한 패킷 수** 등을 확인할 수 있습니다.

게임 서버 테스트

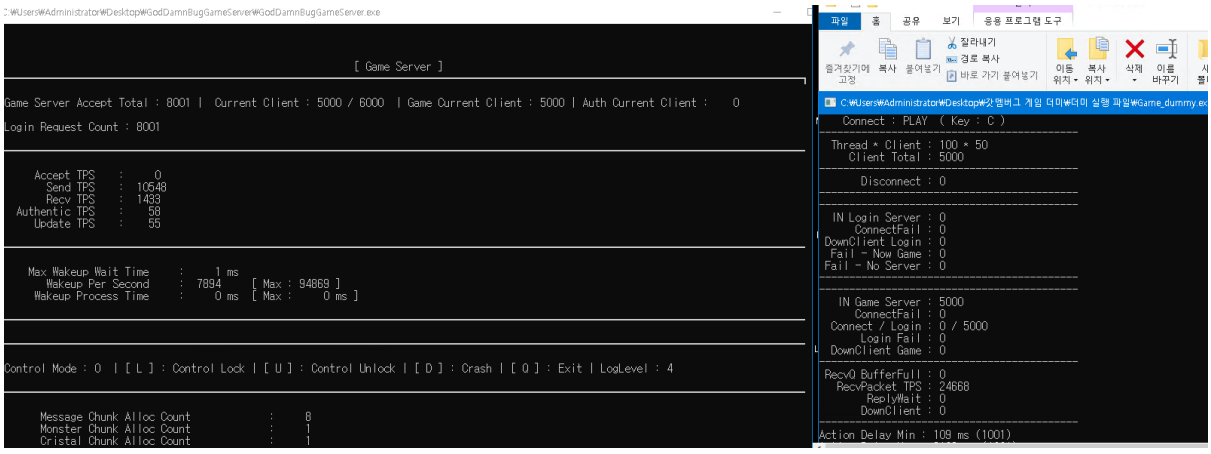
- 게임 서버의 정상 작동 확인을 위해 아래와 같이 테스트를 진행하였습니다.

게임 서버 테스트 방법

1. 5천명의 더미 클라이언트 연결
2. 랜덤한 확률로 컨텐츠 패킷 송신
3. 연결 끊김 없이 테스트 7일, 랜덤한 확률로 연결 끊기 테스트 2~3일

게임 서버 테스트 통과 조건

- 더미 클라이언트 의도와 상관없이 세션이 끊기는 현상이 없어야 함
- 총 연결 클라이언트 수와 채팅 서버에서 Accept한 클라이언트 수가 동일해야 함
- 더미 클라이언트가 수신받지 못한 패킷이 있으면 안 됨
- 더미가 송신한 패킷에 대한 응답이 50ms 이하로 나와야 함
- 컨텐츠 처리 결과가 DB에 잘 반영 되어야 함



(좌측 콘솔 : 게임 서버, 우측 콘솔 : 더미 클라이언트)

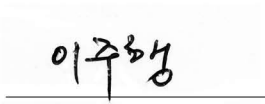
더미 클라이언트는 **총 연결 클라이언트 수, 연결 실패, 로그인 실패, 연결 끊김, RecvQ Full 횟수, Recv 패킷 TPS, 수신받지 못한 패킷 수** 등을 확인할 수 있습니다.

PROCADEMY Certified
GameServer Programmer

정찬훈
Chanhun Jeong
date of birth 12/26/1996

Certification Date
08 / 23 / 2021
Certificate No.
WCBM35SP

Certificate No 확인검증
https://procademy.co.kr/cert



Juhaeng Lee
Chief Executive Officer
PROCADEMY GAMECODI, Inc.

본 증서는 PROCADEMY 에서 1년 이상의 기간 동안 게임서버 프로그래밍 기술을 연구하고 개발 하였음을 증명 합니다.
본 증서는 PROCADEMY 정규과정의 모든 교육과 테스트를 완벽하게 통과 하였음을 증명 합니다.