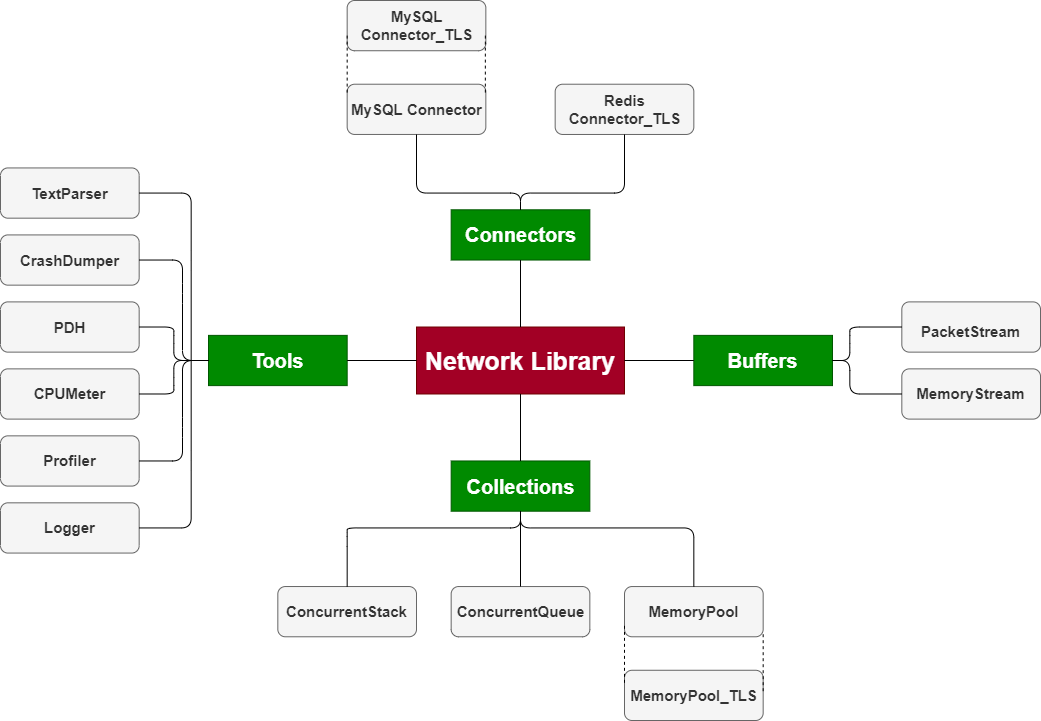
**목차**

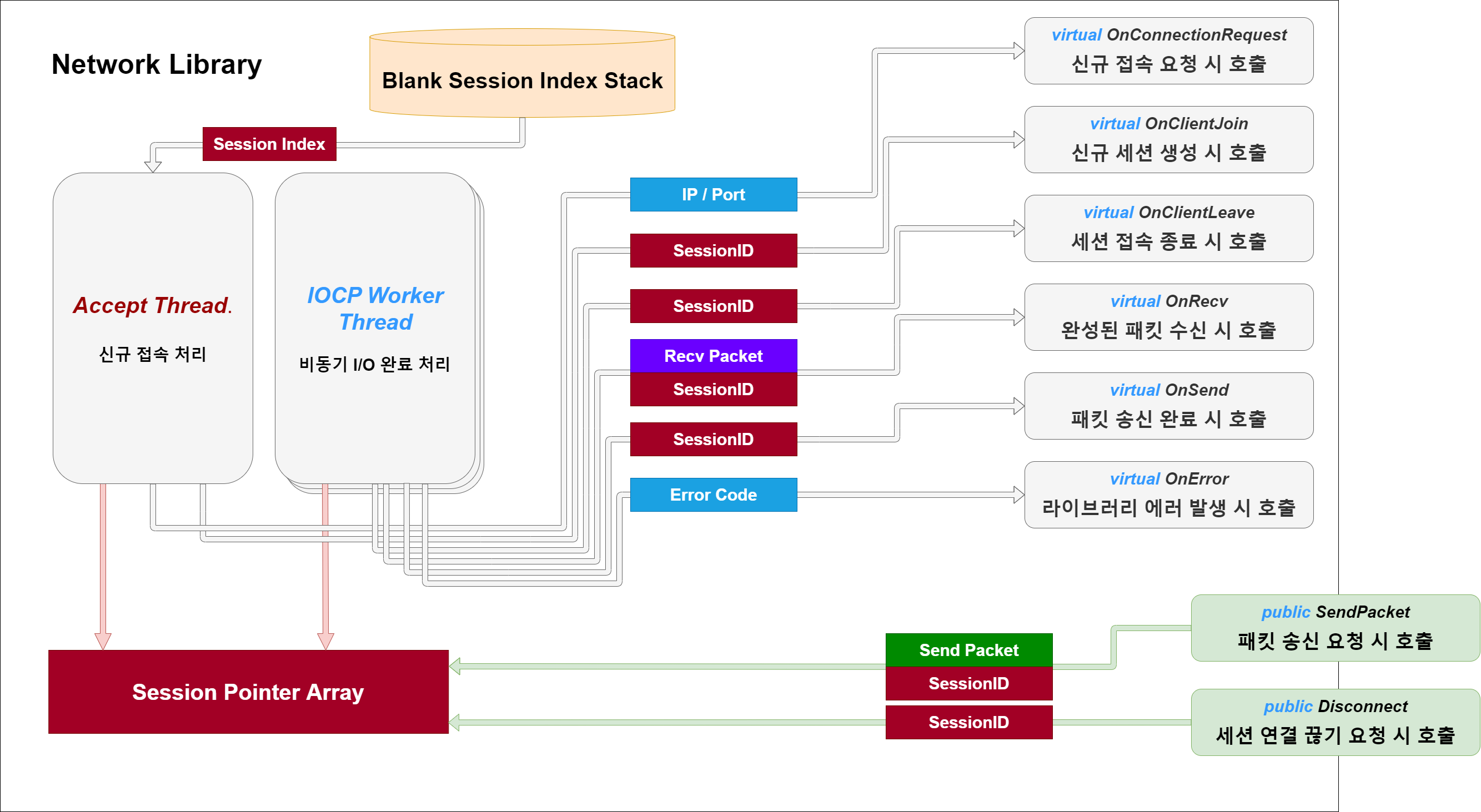
1. **네트워크 라이브러리 소개**
   1. Memory Pool TLS
   2. Lock Free Algorithm Debugging
2. **서버 소개**
   1. Chat Server
   2. Login Server
   3. Monitor Server
   4. Game Server

**네트워크 라이브러리 소개**

**구성도**

****

**구조**

****

**설계 목적**

* + 모든 서버에서 사용할 수 있는 범용적인 네트워크 라이브러리를 구현한다.
  + Session ID를 사용하여 *Network Layer*와 *Contents Layer*를 구분한다.
  + IOCP를 사용하여 비동기 Network I/O에 대한 완료 처리를 멀티 스레드로 처리한다.
  + Network Layer에서 동기화 객체를 사용하지 않고 스레드 Safe하게 구현한다.

**고려 사항**

* + Content Layer에서 Lock이 빈번한 경우 라이브러리의 최대 효과를 볼 수 없다.

**프로토콜 구조**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

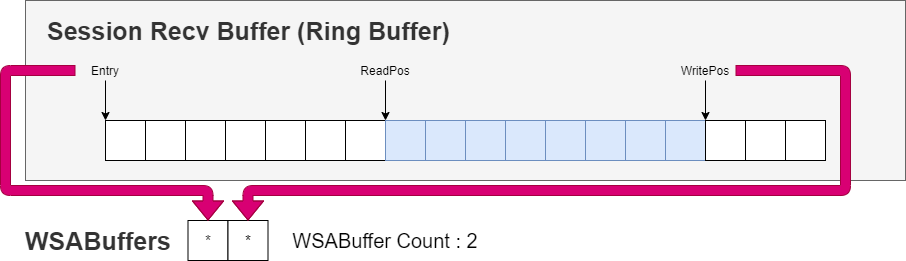
텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + Entry부터 5Bytes는 Network Layer에서만 접근 가능한 헤더 영역이다.
  + 실질적인 사용은 헤더 공간을 제외한 Write Pos부터 사용 가능하다.

**세션 수신 버퍼 등록**

**Case 1.** 세션 수신 버퍼에 남은 공간이 아래와 같을 경우 **WritePos**와 **Entry** 두 곳을 수신 버퍼로 등록.

****

**Case 2.** 세션 수신 버퍼에 남은 공간이 아래와 같을 경우 **WritePos**를 수신 버퍼로 등록.

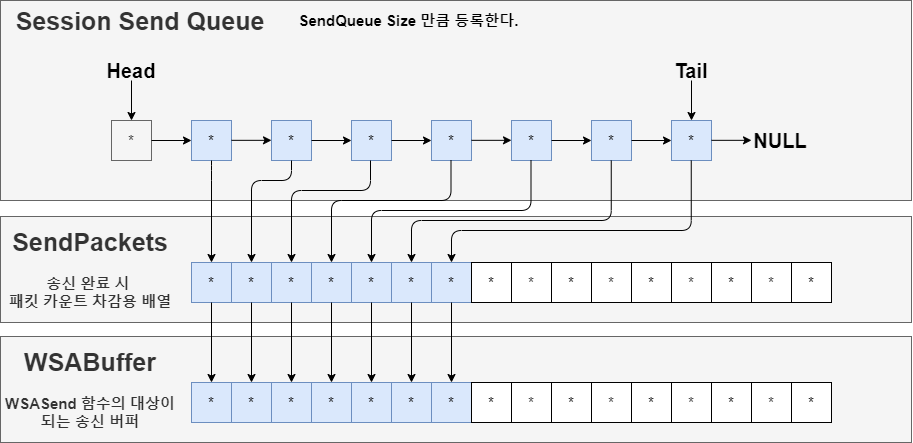
**테이블이(가) 표시된 사진

자동 생성된 설명**

**세션 송신 버퍼 등록**

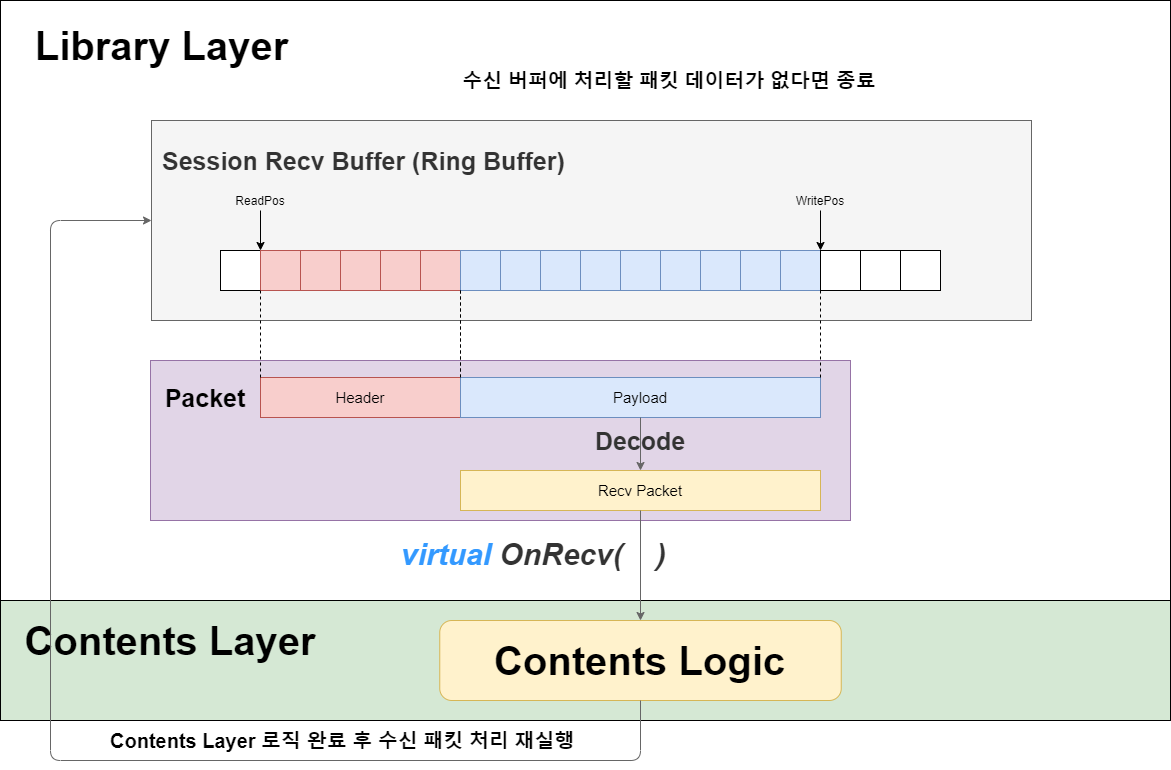
세션 SendQueue Size 만큼 SendPackets, WSABuffer에 패킷 포인터 등록

**[예외]** 지역 변수로 준비한 WSABuffer 배열의 최대 사이즈를 초과하는 것을 방지. (남은 패킷은 다음 기회에 등록한다)

****

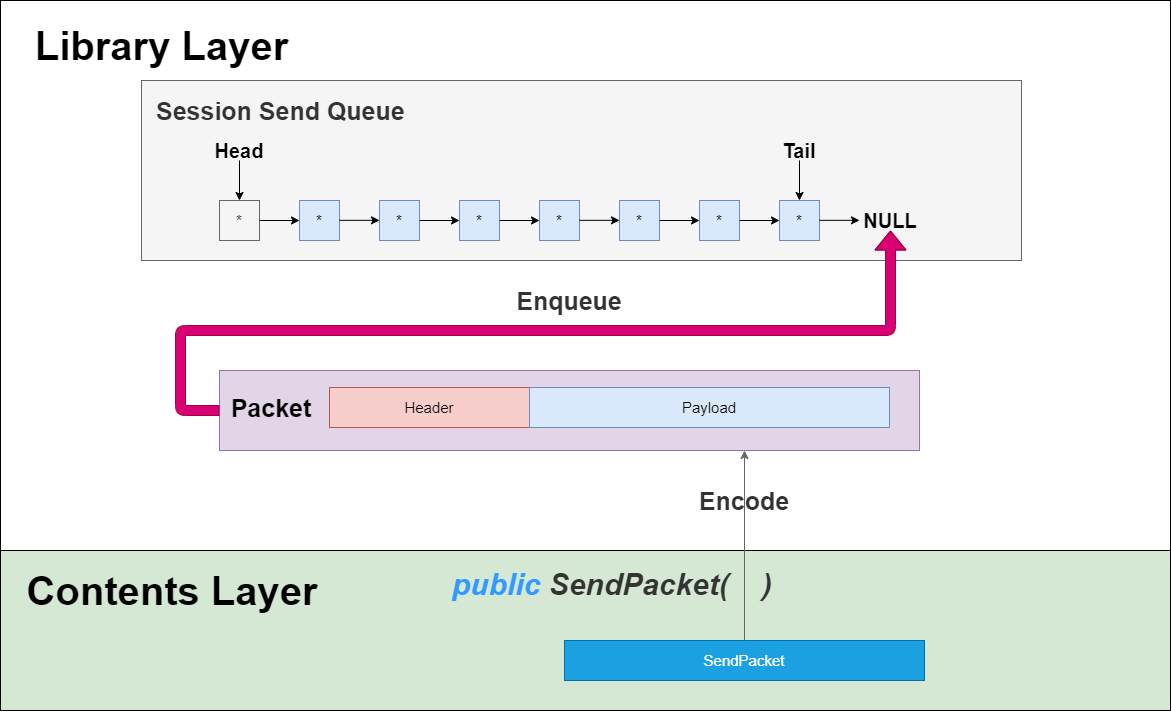
**세션 수신 패킷 처리**

수신 완료 통지를 IOCP에서 Dequeue한 경우 IOCP Worker Thread에서 아래 절차를 진행한다.

****

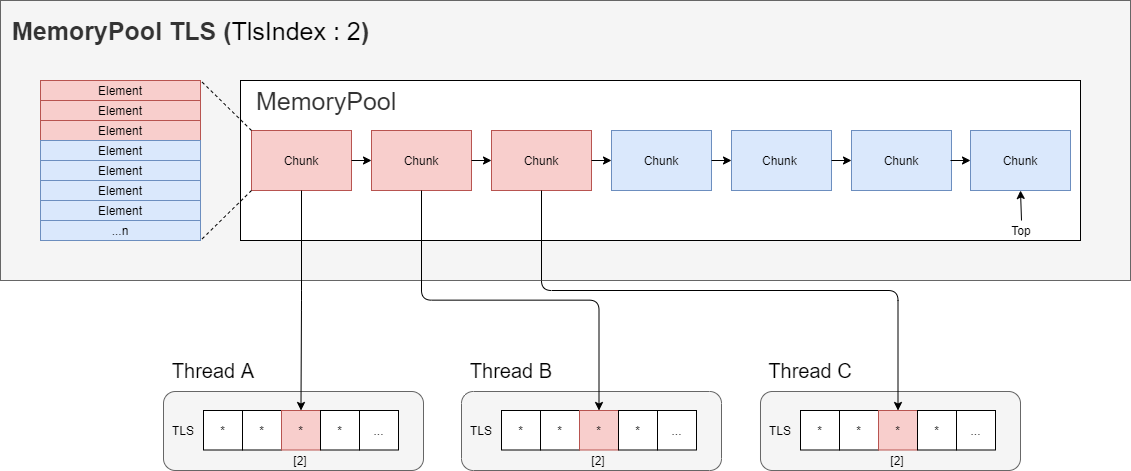
**세션 송신 패킷 등록**

Contents Layer에서 Packet Pool에서 새로운 패킷을 할당 받고 Send Packet을 호출할 경우 아래 절차 진행

****

**1. Memory Pool TLS**

1. **설계 목적 및 구조**
   * 현재 라이브러리에서 패킷 메모리 할당 및 해제는 하나의 **공용 메모리풀**을 대상으로 실행 중.
   * 때문에 Lock-Free Stack 구조이며, 스레드 간 경합이 발생할 경우 스레드 개수에 비례하여 성능 저하.
   * 위 이유로 TLS를 이용하여 할당에서의 스레드 경합을 최소화하기 위함.

****

1. **Chunk 구조**
   * 할당 및 반환을 위한 Chunk 구조는 아래와 같다.
   * Elements 사이즈가 클수록 경합 빈도는 낮아지지만, 메모리 사용량이 늘어난다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **메모리 할당 절차**
   1. 메모리 할당 요청을 받았다.
   2. TLS에 세팅 된 Chunk가 없었다.
   3. Memory Pool에서 새로운 Chunk를 할당(경합 지점)받고 TLS에 세팅 후 Element를 반환해준다.
   4. 메모리 할당 재요청을 받았다.
   5. TLS에 세팅 되어있는 Chunk에서 다음 인덱스의 Element를 반환해준다.
2. **메모리 반환 절차**
   1. 메모리 해제 요청을 받았다.
   2. TLS에 세팅 된 Chunk의 Free Count를 차감한다.
   3. 차감한 Free Count가 MAX였다.
   4. 해당 Chunk를 Memory Pool에 반환한다. (경합 지점)
3. **성능 테스트**
   * Heap과 기본 메모리풀을 사용하였을 때, 스레드 개수에 비례하여 성능이 저하됨. (경합 지점에서 동기화를 하므로)
   * TLS를 사용한 메모리풀의 경우, Chunk 할당 이후부터는 스레드의 배타적 공간을 사용하므로 스레드 개수와 상관없이 동일한 속도를 보이며 성능이 향상됐다.

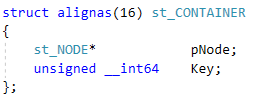
테이블이(가) 표시된 사진

자동 생성된 설명 **테이블이(가) 표시된 사진

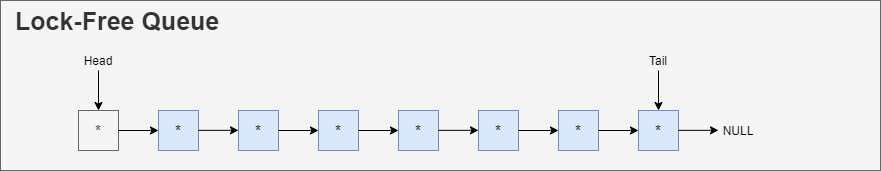
자동 생성된 설명**

**2. Lock Free Queue**

1. **구현 목적**
   1. 멀티 스레드 기반의 Lock-Free 라이브러리 구현을 위해 사용.
   2. 멀티 스레드 **디버깅 능력 및 이슈 추적 능력 증진**을 위해 구현.
2. **기본 구조**
   1. 노드의 추가, 삭제 및 Head, Tail의 변경을 위해 ***InterlockedCompareExchange/ InterlockedCompareExchange128*** 함수를 사용.
   2. 아래는 해당 함수 사용을 위한 컨테이너 구조체이며, Head, Tail의 메모리 구조다.



* 1. 각 스레드는 위 구조체를 사용하여 스레드마다 배타적으로 CAS, DCAS 함수의 인자를 준비한 후, 비교 및 변환을 원자적으로 실행
  2. 위의 이유로 해당 Lock-Free Queue의 구조는 아래와 같은 List 형식이다.

****

1. **함수 작동 방식**
   * **Enqueue**
     1. 메모리풀에서 신규 노드를 할당 받는다.
     2. 현재 Tail의 Next Node를 할당 받은 노드로 갱신한다. (CAS)
     3. 할당 받은 노드를 현재 Tail로 갱신한다. (DCAS)
   * **Dequeue**
     1. 현재 Head의 Next Node에서 값을 출력한다.
     2. 현재 Head의 Next Node를 현재 Head로 갱신한다. (DCAS)
     3. 이전 Head를 메모리풀에 반환한다.
2. **검증테스트**

* **초기값 세팅 후 아래 스레드마다 아래 절차 반복.**
  + 1. Dequeue 체크
    2. 초기값 체크
    3. 변경 데이터 유지 체크
    4. 원복 데이터 유지 체크
    5. Enqueue 체크

1. **디버깅 환경**

|  |  |  |
| --- | --- | --- |
| **로그 구조체** | 테이블이(가) 표시된 사진  자동 생성된 설명텍스트이(가) 표시된 사진  자동 생성된 설명 | - 메모리상에서 확인하고자 하는 로그 명세 목록 |
| **로그 배열** |  | - 구조체 배열을 전역 또는 클래스의 프로퍼티로 선언하여 사용  - Interlock 함수로 스레드마다 접근할 배열 고유 인덱스 획득 |
| **로그 넘버링** | 텍스트이(가) 표시된 사진  자동 생성된 설명 | - 임의로 지정하는 코드상의 로직 파악을 위한 상수 |
| **로그 기록** |  | - 확인하고자 하는 당시 상황의 값을 매개변수로 하여 **SetLog()** 함수를 통해 로그 기록. |

1. **버그리포트 1 : Dequeue에서 Head Next가 NULL인 상황**
   1. **로그 디버깅**

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 테이블이(가) 표시된 사진  자동 생성된 설명 | **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** |
| 1. pHeadNext : NULL  2. Capacity : 1  3. TailNextNode : TailNode | 1. Head과 Tail이 동일 노드 참조  2. (1)의 상황임에도 Capacity는 1  3. TailNextNode가 NULL이 아님 | 1. Capacity가 1이므로 Dequeue 실행  2. Head를 HeadNextNode로 갱신(Tail 추월)  3. 이전 HeadNode 반환 (Tail 반환) |
| 4 | **5** | **6** |
| 테이블이(가) 표시된 사진  자동 생성된 설명 | 테이블이(가) 표시된 사진  자동 생성된 설명 | 테이블이(가) 표시된 사진  자동 생성된 설명 |
| 1. Enqueue에서 반환했던 Head(Tail)Node 할당  2. AllocNode Next 초기화 | 1. SnapTail 생성  2. SnapTailNextNode를 AllocNode로 갱신  3. AllocNode와 TailNode는 동일한 상황  4. 이로 인해 TailNextNode는 TailNode | 1. TailNextNode에 AllocNode 추가했음으로 Capacity 증가 (0 -> 1)  2. 하지만 실제 HeadNextNode는 NULL |

* 1. **시나리오**

|  |  |
| --- | --- |
|  | 1. Head와 Tail Node가 동일한 상황  2. Capacity는 1  3. TailNextNode는 NULL이 아님. |
|  | 1. 다른 스레드의 Dequeue  2. Head는 Tail 추월  3. Tail Node 반환. |
|  | 1. 다른 스레드의 Enqueue  2. 메모리풀에서 TailNode 할당 받음.  3. AllocNode 초기화 후 TailNextNode로 갱신.  4. TailNextNode가 Tail이 되는 상황.  5. Capacity 상승. |

1. **버그리포트 2 : TailNextNode가 NULL이 아닌 상황 (1)**
   1. **로그 디버깅**

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **테이블이(가) 표시된 사진  자동 생성된 설명** |  | **테이블이(가) 표시된 사진  자동 생성된 설명** |
| 1. Tail Key : 107  2. 컨텍스트 스위칭 | 1. 다른 스레드에서 Tail 갱신  2. Key 갱신 (107 -> 108) | 1. 14044 스레드로 돌아와서 현재 시점의 TailNode Snapshot 생성.  2. TailNextNode NULL이라 판단. |
| **4** | **5** | **6** |
| **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** |
| 1. TailNextNode를 AllocNode로 갱신 | 1. 이전 Key와 현재 TailKey가 다름.  2. Tail 갱신 실패. | 1. Capacity 상승  2. 하지만 Tail 갱신은 실패.  3. TailNextNode가 NULL이 아닌 상황 발생. |

* 1. **시나리오**

|  |  |
| --- | --- |
|  | 1. SnapshotTail Key 세팅  2. 컨텍스트 스위칭 |
|  | 1. Thread B에서 Enqueue 완료  2. Tail Key 갱신 |
|  | 1. SnapshotTail Node 세팅  2. SnapshotTailNextNode를 NULL로 판단  3. SnapshotTailNextNode를 AllocNode로 갱신  4. SnapshotTail Key와 현재 TailKey 미일치.  5. Tail 갱신 실패. |

1. **버그리포트 3 : TailNextNode가 NULL이 아닌 상황 (2)**
   1. **로그 디버깅**

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** |
| 1. SnapshotTail 생성  2. 컨텍스트 스위칭 | 1. 22832스레드의 SnapshotTail이 다른 스레드의 EnQ, DeQ를 거치며 메모리풀에 반환.  2. 이후 13828 스레드의 Enqueue 작업에서AllocNode가 된 상황. | 1. AllocNoded 초기화  2. 컨텍스트 스위칭 |

|  |  |  |
| --- | --- | --- |
| **4** | - | - |
| **테이블이(가) 표시된 사진  자동 생성된 설명** |  |  |
| 1. 22832 스레드에서 SnapshotTailNext가 NULL이라 판단.  2. TailNextNode를 AllocNode로 갱신.  3. SnapshotTail이 현재 Tail과는 다른 상황.  4. Tail 갱신 실패.  5. Capacity 증가. | 6. 이후 13828 스레드로 돌아왔을 때 해당 스레드의 AllocNodeNext가 NULL이 아닌 상황  7. Tail을 AllocNode로 갱신.  8. TailNextNode가 NULL이 아닌 상황 발생. |  |

* 1. **시나리오**

|  |  |
| --- | --- |
|  | 1. Thread A SnapshotTail 생성  2. 컨텍스트 스위칭 |
|  | 1. Thread A의 SnapshotTail이 다른 스레드의 EnQ, DeQ를 거치며 메모리풀에 반환.  2. 이후 Thread B의 Enqueue 작업에서 AllocNode로 할당.  3. AllocNode 초기화  4. 컨텍스트 스위칭 |
|  | 1. ThreadA로 돌아왔을 때 SnapshotTail의 Next가 NULL이라 판단  2. SnapshotTailNextNode를 AllocNode로 갱신  3. 현재 Tail과 SnapshotTail이 일치하지 않음.  4. Tail 갱신 실패. |
|  | 1. Thread B로 돌아왔을 때 AllocNode의 Next가 Thread A로 NULL이 아닌 상황.  2. TailNextNode를 AllocNode로 갱신.  3. TailNextNode가 NULL이 아닌 상황. |

1. **버그 1,2,3 상황 해결**
   1. 버그리포트 1, 2, 3의 상황은 모두 TailNextNode가 NULL이 아닌 상황에서 발생했다.
   2. 하지만 Lock-Free Queue에서 버그리포트 2, 3의 상황은 에러로 보지 않는다. 즉, Dequeue 상황에서 버그리포트 2, 3의 상황인 경우해당 스레드가 Tail을 한 칸 밀고 Dequeue를 진행한다.

|  |  |
| --- | --- |
|  | 1. TailNextNode가 NULL이 아닌지 체크. |
|  | 2. 현재 Tail을 TailNextNode로 갱신. |
|  | 3. HeadNextNode를 Head로 갱신.  4. Capacity 차감. |

1. **버그리포트4. Dequeue에서 동일 값을 반환하는 상황**
   1. **로그 디버깅**

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** | **테이블이(가) 표시된 사진  자동 생성된 설명** |
| 1. HeadNextNode를 Head로 갱신.  2. 이전 Head 반환.  3. 컨텍스트 스위칭 | 1. 28948 스레드의 HeadNextNode 반환되는 상황. | 1. 다른 스레드의 Enqueue에서 HeadNextNode 를 할당받음.  2. HeadNextNode에 해당 스레드의 EnqData 세팅. |

|  |  |  |
| --- | --- | --- |
| **4** | **-** | **-** |
| **테이블이(가) 표시된 사진  자동 생성된 설명** |  |  |
| 1. 다시 28948 스레드로 돌아와서 HeadNextNode의 데이터 참조.  2. 하지만 이전에 획득했던 HeadNextNode 의 데이터와 다른 상황. |  |  |

* 1. **시나리오**

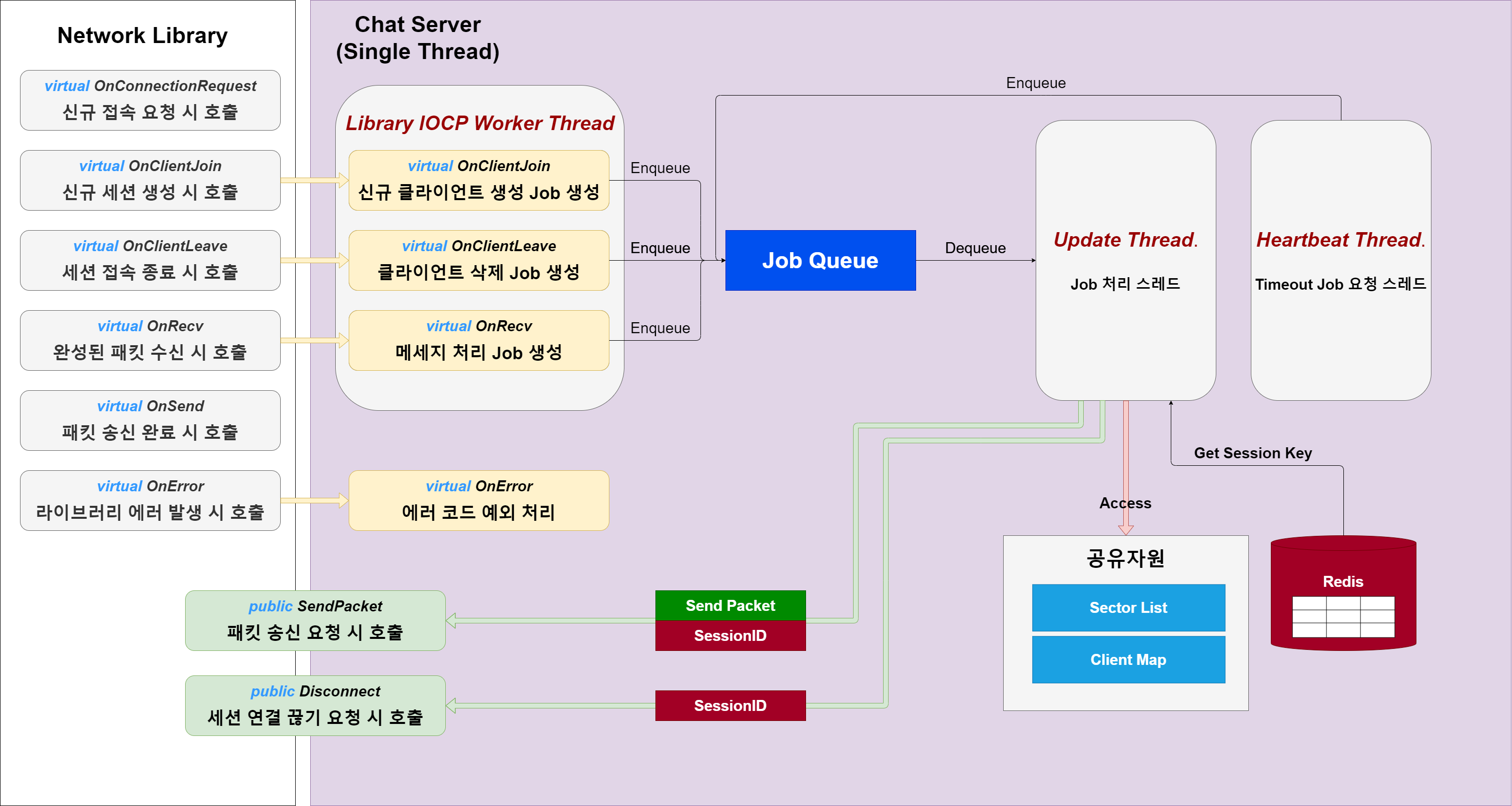
|  |  |
| --- | --- |
| **텍스트, 시계, 스크린샷이(가) 표시된 사진  자동 생성된 설명** | 1. HeadNextNode를 Head로 갱신.  2. 이전 Head 반환.  3. 컨텍스트 스위칭 |
|  | 1. 다른 스레드의 Enqueue에서 HeadNextNode 를 할당받음.  2. HeadNextNode에 해당 스레드의 EnqData 세팅. |
|  | 1. 다시 28948 스레드로 돌아와서 HeadNextNode의 데이터 참조.  2. 하지만 이전에 획득했던 HeadNextNode 의 데이터와 다른 상황. |

* 1. **해결**
     + 해당 에러는 Head를 먼저 갱신함으로 인해 다른 스레드가 Snapshot\_HeadNextNode에 접근할 수 있는 상황에서 발생했다.
     + 때문에 Head를 갱신하기 전에 데이터를 먼저 뽑고 Head를 갱신한다.

|  |  |
| --- | --- |
| **텍스트, 시계, 스크린샷이(가) 표시된 사진  자동 생성된 설명** | 1. HeadNextNode Data를 Destination에 세팅. |
| **텍스트, 시계, 스크린샷이(가) 표시된 사진  자동 생성된 설명** | 2. Head를 HeadNextNode로 갱신 |

**서버 설계**

**채팅서버 (싱글스레드 구조)**

****

**설계 목적**

* + LockFreeQueue를 사용하여 라이브러리 측 네트워크 이벤트를 하나의 스레드로 처리한다.
  + Contents Layer 스레드간의 공유 자원에 대한 동기화 이슈를 신경쓰지 않는다.

**본 설계의 단점**

* + 서버의 모든 로직 처리를 하나의 스레드에서 처리하고 있으므로, 서버의 전체적인 병렬성 및 응답성이 저하된다.
  + IOCP Worker Thread에서 생성하는 Job을 하나의 LockFreeQueue에 Enqueue하는 과정에서 스레드간의 경합으로 인해 CPU 사용율 향상 및 Interlocked 함수로 인해 캐시 라인이 잠기면서 다른 스레드가 돌지 못하는 상황이 발생할 수 있다.

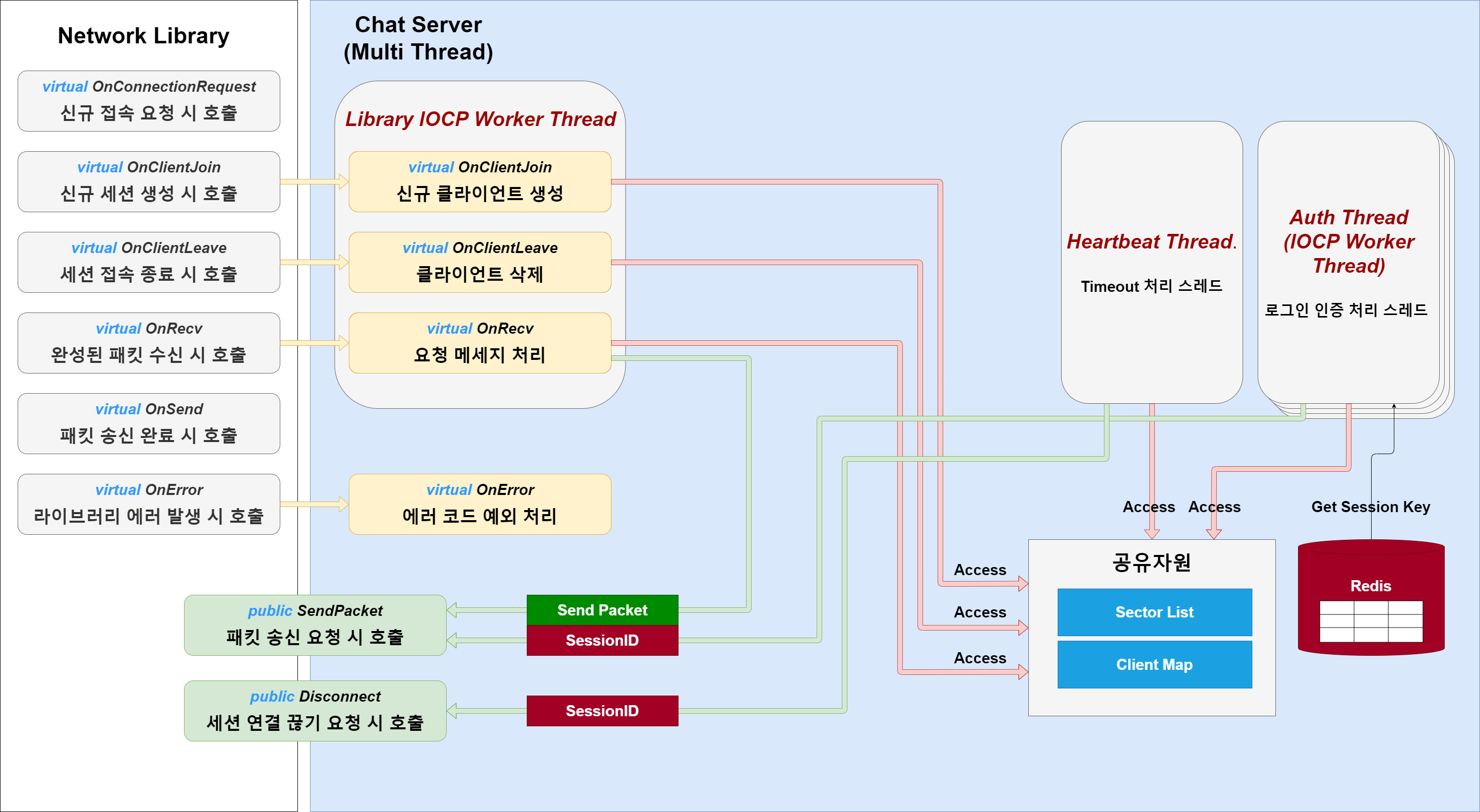
**IOCP Update Thread vs Event Update Thread**

**LockFree JobQueue vs Lock JobQueue**

**동기 Send vs 비동기 Send (ZeroCopy)**

**Lock Library vs LockFree Library**

**채팅서버 (멀티스레드 구조)**

****

**설계 목적**

* + IOCP Worker Thread를 사용하여 Network I/O에 대한 로직을 멀티스레드로 처리 한다.
  + 기존 싱글스레드로 처리시 성능 저하의 원인이 됐던 Redis 접근을 별도의 AuthThread로 분산하여 로직 처리에 대한 성능을 향상시킨다.

**본 설계의 단점**

* + 모든 스레드의 로직 처리 과정에서 공유 자원 접근이 필요하므로 Lock이 필요하다.
  + 이로인해 기존 설계 목적과는 정반대로 스레드의 로직 처리 병렬성이 저하된다.

**SRWLock vs CriticalSection**

**동기 Send vs 비동기 Send**

**Lock Library vs LockFree Library**

**IOCP AuthThread vs Event AuthThread**

**싱글스레드 구조 vs 멀티스레드 구조**

**채팅 서버 동작 검증 테스트**