* Unordered\_map을 사용하신 이유가 뭔가요?
* Unordered map은 탐색속도가 상수시간이고 순서가 중요하지 않다는 장점이 있는데 이 경우에는 정수형인 ID를 사용하여 탐색을 하는 경우가 많았고 ID의 순서가 중요하지 않기에 사용했습니다.
* 단점은 없나요?

해시 충돌로 인해 성능이 저하될 수 있고 버킷으로 인해 메모리 사용량이 늘어난 다는 단점이 있습니다.

* Unordered map은 내부적으로 어떻게 구현되어 있는지 아시나요?

해시 테이블로 구현되어 있습니다.

해시 테이블이란 Key값을 해싱하여 나온 해쉬 값에 의해 적절한 버킷에 저장하는 것을 해시 테이블 구조라고 합니다.

* 해싱이란 ?

Key를 통해 특정 값을 얻어내는 작업을 말합니다.

그리고 그 값을 Value를 저장할 공간의 인덱스로 사용하는 것 입니다.

저장되는 공간을 버킷 이라고 부릅니다.

* Unordered\_map은 탐색에 있어 항상 map보다 우수한가요?

아니라고 알고있습니다.

자세히는 모르지만 긴 문자열이 Key로 설정 되었을 때는 키를 해싱하는 작업이 map에서 문자열을 비교하는 것 보다 성능적으로 좋지 못하다고 알고있습니다.

그리고 일정 개수를 넘어갈 경우 해시 충돌이 자주 발생하여 map이 더 좋은 경우가 있습니다.

* Unordered\_map은 시간복잡도가 항상 상수시간인가요?

최악의 경우 체이닝 방식에서 n이의 시간복잡도가 나올 수 있습니다.

체이닝이란 해시 충돌이 발생하면 해당 인덱스에 추가로 이어 저장하는 방법인데 한 인덱스에 모든 데이터가 이어서 저장되어 있고 가장 끝에 찾으려는 값이 존재한다면 n이 될 수 있습니다.

* 충돌을 해결하는 다른 방식은 없나요?

개방 번지화 라는 방식이 있습니다.

해시 충돌이 발생하면 비어 있는 인덱스를 찾아 value를 할당하는 방법입니다.

* 개방 주소법의 종류에 대해 아시나요?

선형 탐사, 제곱 탐사, 이중 해싱이 있습니다.

선형 탐사란 충돌 시 다음 인덱스를 확인해 가며 비어있는 공간을 찾는 방식이고

제곱 탐사란 이차식을 통해 인덱스를 찾는 폭을 구해 비어있는 공간을 찾는 방식입니다.

이중 해싱이란 해시 충돌 발생 시 다른 해시 함수를 사용하여 증가 폭을 구하고 그만큼 인덱스를 증가시키며 비어있는 곳을 탐색하는 방식입니다.

* Map에 대해서 설명해보세요

Key와 Value로 된 쌍을 저장하는 컨테이너이며 자료를 자동 정렬하기 때문에 정렬이 필요한 경우 사용하며 키를 새로 추가할 때마다 정렬된 위치를 찾아 추가해야 하기 때문에 대용량 데이터를 저장하는 경우 시간이 많이 소요된다는 단점이 있습니다.

* 대용량 데이터를 저장하는 경우 시간이 많이 소요된다고 하셨는데 최악의 경우 시간복잡도는 어떻게 되나요?

Log n입니다.

* Map은 내부적으로 어떻게 구현되어 있는지 아시나요?

레드블랙트리로 구현되어 있습니다.

* 레드블랙 트리란 무엇인가요?

균형 이진 탐색 트리의 한 종류로 이진 탐색 트리에서 한쪽으로 데이터가 몰려 최악의 경우 n의 탐색시간이 걸릴 수 있다는 단점을 보완하기 위한 자료구조입니다.

* 좀 더 자세히 설명하실 수 있나요?

레드 블랙 트리는 5가지 조건을 가지고 구현합니다.

모든 노드는 블랙 또는 레드 색상을 가지며

루트 노드는 검정색이여야 하고 리프 노드는 검정색이여야 합니다.

그리고 빨간 노드의 자식은 모두 검정색이지만 검정 노드의 자식이 빨강인 필요는 없고

루트 노드에서 무든 잎 노드 사이에 존재하는 검은색 노드의 수는 모두 동일하다는 조건을 가지고 있습니다.

* 이진 탐색 트리란 무엇인가요?

노드의 Key는 중복이 없어야 하며 어떤 노드의 왼쪽 자신 노드는 나보다 작고

오른쪽 자식 노드는 나보다 커야한다는 조건을 만족하는 자료구조 입니다.

단점으로는 트리의 모양이 한쪽으로 치우쳐지게 되면 탐색 성능이 떨어지게 됩니다.

* 이진 탐색이란 무엇인가요?

데이터가 정렬되어 있는 배열에서 특정값을 찾는 알고리즘 입니다.

배열의 중간에 있는 임의의 값을 선택하여 찾고자 하는 값 과 비교하고 중간 값 보다 작으면 좌측의 데이터, 크다면 우측의 데이터들을 대상으로 다시 탐색하는 알고리즘 입니다.

* 트리와 그래프에 대해 아시나요?

그래프란 노드 간을 연결하는 간선으로 구성된 자료구조 이며 계층이 없는 네트워크 모델입니다. 객체간의 관계를 표현할 수 있는 자료구조라는 특징이 있습니다.

순환 사이클 가능

그래프의 종류 중 하나로 노드와 간선으로 이루어진 자료구조 입니다.

계층적인 모델이며 노드간에 단 하나의 간선으로 연결되어 있다는 특징이 있습니다.

순환 사이클 불가능

* 트리와 그래프의 차이점에 대해 아시나요?

트리와 다르게 그래프는 루트 노드나 부모 자식 관계라는 개념이 없고 노드 간에 2개 이상의 경로가 가능하다는 차이점이 있습니다.

공통점으로는 노드 사이를 연결하는 간선으로 구성된 자료구조라는 점이 있습니다.

* 깊이 우선탐색

루트 노드에서 시작하여 다음 분기로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법으로 모든 노드를 방문하고자 하는 경우에 이 방법을 사용합니다.

특징으로는 자기 자신을 호출하는 순환 알고리즘의 형태를 지닌 다는 것과

방문 여부를 반드시 검사해야 한다는 점이 있습니다.

* 왜 검사해야 하나요?

방문 여부를 검사하지 않는다면 무한루프에 빠질 위험이 있기 때문입니다.

* 시간복잡도는 어떻게 되나요?

인접 리스트로 표현된 그래프의 경우 n+e

인접 행렬로 표현된 그래프는 n2입니다.

* 왜 그런지 아시나요?

인접 행렬은 이차원 배열로 구현 되는데 행과 열이 노드의 개수이기 때문입니다.

인접 리스트는 벡터의 배열로 나타내는 방식이며 노드의 개수 만큼의 벡터에 간선의 정보를 저장하므로 n+ 간선의 개수 입니다.

* 너비 우선 탐색에 대해 아시나요?

루트 노드에서 시작하여 인접한 노드를 먼저 탐색하는 방법입니다.

보통 두 노드 사이의 최단 경로 혹은 임의의 경로를 찾고 싶을 때 이 방법을 선택 합니다.

특징으로는 재귀적으로 동작하지 않으며 그래프 탐색의 경우 방문여부를 검사해야하고

방문한 노드를 차례로 저장한 후 꺼낼 수 있는 자료 구조인 큐를 사용합니다.

* 시간 복잡도에 대해 아시나요?

인접 행렬의 경우 n2

인접 리스트로 구현한 경우 n + 간선의 개수입니다.

* 두가지는 어떨 때 사용하나요?

경로의 특징을 저장해 줘야하는 문제는 DFS를 사용하고

최단거리를 구하거나 검색 시작 지점부터 원하는 대상 사이의 거리가 멀지 않다면

BFS를 사용합니다.

* Mmo Rpg 개발을 위해 IOCP를 사용하셨다고 했는데 어떤 점에서 좋다고 생각했나요?

IOCP 객체 내부의 스레드 풀을 사용하기 때문에 스레드 생성 파괴 오버헤드를 줄여 효율적인 스레드 관리가 가능하고 그렇기 때문에 적은 수의 스레드로 많은 연결을 할 수 있기 때문입니다.

스레드가 적다는 것은 Context switching 비용이 적다는 것을 의미하기도 합니다.

* GQCS에서 어떤 내용이 있는지에 대해 어떻게 판단하셨나요?

오버렙드 구조체를 확장하여 사용하였습니다.

Send recv를 판단하는 정보와 buffer의 위치를 추가하였습니다.

* ID는 필요하지 않았나요?

아니요 필요했지만 확장하여 사용하지 않았습니다.

IOCP에서 오고 가는 completion\_key에 id를 담아 사용했습니다.

* Buffer의 위치는 왜 알아야 하나요?

SEND일 경우 어떤 버퍼에서 완료되었는지 알 수 없는데 재사용을 위해서는 위치를 알아야 합니다.

* 좀더 자세히 설명해 주실 수 있나요?

기본적으로 Overlapped IO (callback)으로 동작합니다.

그리고 IOCP 객체를 통해 관리되는데 이는 커널에서 관리하며 소켓을 IOCP에 등록하여 사용합니다.

전반적인 실행의 순서는

비동기 함수(AcceptEX, WSASend, WSARecv)를 실행하여 비동기 입출력이 종료되면 완료된 내용을 IOCP객체에 쌓게 됩니다.

그러면 작업이 끝난 스레드에서 GetQueuedCompletionStatus를 호출하여 완료된 내용을 가져와 사용하게 됩니다.

* 데이터가 잘리는 경우가 있는가?

네 있습니다 (부하가 적으면 데이터가 띄엄띄엄 가고 부하가 커지면 잘려오는 등 제대로 동작하지 않는 경우가 있습니다.)

* 그럴 땐 어떻게 처리했는가?

패킷 재조립 방식을 사용했습니다.

* 패킷 재조립이란 무엇인가요?

패킷이 잘못 올 경우를 대비하여 클라이언트 각 객체는 보관용 버퍼와 받은 패킷의 사이즈를 저장하는 변수를 가지고 있으며 패킷이 도착했을 때

받은 패킷의 사이즈와 버퍼의 내용을 담아 놓고 패킷을 완성할 수 있다면 완성 후 패킷을 처리하는 함수로 넘어가는 방식입니다.

완성할 수 없다면 버퍼에 복사해 두고 다음 처리할 데이터에 접근합니다.

* Overlapped io 모델에 대해 아시나요?

Overlapped io는 입출력 함수를 호출한 후 작업의 완료 여부와 무관하게 다른 작업을 진행하고 있다가 입출력 작업이 끝나면 입출력 결과를 처리하는 방식입니다.

* 두가지 모델이 있는데 혹시 아니나요?

이벤트 모델과 콜백 모델이 있습니다.

이벤트 모델이란 작업이 완료되면 이벤트 객체의 상태를 바꿔줌 으로써 작업 완료를 판단하여 결과를 처리하는 방식이고

콜백 모델은 작업이 완료되면 등록해둔 함수를 호출하여 처리하는 방식입니다.

* 오버랩드 io 콜백 모델에 대해 자세히 말씀해 주실 수 있나요?
* 다른 모델에 대해 아시는 게 있나요?
* Iocp쪽 코드에 모르는 내용들 정리
* 효림이한테 usb 받아서 추가 작성
* 왜 스레드의 개수가 이렇게 많나요?

거의 개발이 완료되었을 때 스레드의 개수를 달리하여 성능 테스트를 진행했고 가장 많은 동시접속자를 확보했기 때문에 사용했습니다.

* 성능 테스트는 어떻게 진행했나요?

더미 클라이언트를 만들어 테스트를 진행 했는데 8코어 PC에서

1초에 한번 Move/Stop 패킷을 랜덤으로 전송하여 확인했습니다.

* 쿼드트리 구현하셨는데 쿼드트리에 대해 설명해 주실 수 있으신가요?

자식 노드가 4개 이기 때문에 쿼드트리라고 불리며 불필요한 큰 영역을 한번의 검사로 제외 시킬 수 있기 때문에 시야처리에서 유리한 방식이라고 생각합니다.

* 부모 노드의 데이터를 자식 노드에 넘겨주는 방식을 채택한 이유가 있나요?

부모 노드에 데이터를 남기고 검사해야 한다면 최악의 경우 전체 맵의 꼭지점에 있는

플레이어 끼리도 검사를 하는 일이 생길 수 있어서 비효율 적이라고 생각했습니다.

그래서 자식 노드로 데이터를 넘겨주고 시야범위에 포함된 노드에 대해서만 검사를 진행하려고 사용했습니다.

* 그렇게 하면 단점은 없나요?

아무래도 더 많은 노드가 생성되기 때문에 메모리 측면에서 불리하다고 생각합니다.

* 쿼드트리의 Search 알고리즘에서 포함되는 경우를 여러 개로 나누어 검사하게 된 이유가 있나요?

목표했던 동접치를 달성하지 못해서 성능 프로파일러를 사용해 많은 부하를 차지하는

부분이 Search함수라는 것을 알게 되었고 1차적으로 완전 포함되는 객체의 자식 노드가 가진 플레이어 까지도 시야범위 포함여부에 대한 검사를 하는 연산을 줄이기 위해 경우를 나누어 검사하게 되었습니다.

* 1차적으로라고 하셨는데 다른 곳에서의 최적화도 진행하셨나요?

네 맞습니다.

Search함수를 호출하지 않고 기능을 구현할 수 있는 부분에 대해 다른 방식을 채택하여 구현했습니다.

원래는 이동 시 시야처리에 관한 부분에서 원래 좌표, 새로 이동한 좌표에 대해 Search함수를 실행하여 그에 따른 패킷을 전송하는 방식으로 구현 했습니다.

이 부분에서 플레이어에게 시야에 들어와 있는 객체의 ID를 미리 저장해 두어 좌표가 바뀐 후에만 Search함수를 호출하는 구조로 바꿨습니다.

* 코드를 보면 이동시에 쿼드트리에서 삭제 삽입을 통해 업데이트 해주시고 계신데 다른 효율적인 방법은 없었을까요?

쿼드트리 내에서 객체의 좌표를 이동 시켜주고 그에 따라 속한 노드의 데이터를 변경해 주는 방법과 지금의 방식에 대해서 고민을 했었습니다.

전자의 경우 코드가 너무 복잡해 질 수도 있고 노드가 바뀌게 된다면 바뀐 노드를 찾아가는 과정이 생기게 되는데 최악의 경우 Delete Insert를 호출하는 경우와 별반 다를 것이 없다고 생각해서 후자를 택하게 되었습니다.

* 동기화 기법으로 뮤텍스를 사용하셨는데 뮤텍스의 역할은 뭔가요?

멀티스레드 환경에서 여러 스레드가 하나의 자원에 접근하여 결과값이 달라지는 것을 막는 역할을 합니다.

* 단점은 없나요?

Lock을 사용 시 속도 측면에서 불리합니다. 자세히는 모르지만 lock 호출 자체가 큰 부하를 요구한다고 알고 있습니다.

그렇기에 lock으로 보호하는 영역이 너무 작다면 lock의 잦은 호출로 성능이 저하되고

반대로 너무 크다면 다른 스래드가 대기하는 시간이 떨어지기에 성능이 저하됩니다.

* Data Race란 무엇인가요?

여러스레드가 공유자원에 동시 접근할 때 경쟁 상태가 되는 것을 의미 합니다.

* 데이터레이스가 생기는 변수들엔 뭐가 있나요?

전역변수에 대해 발생하는 것으로 알고 있습니다.

스레드를 생성 후 업데이트 되는 변수 정도로 알고 있습니다.

* 뮤텍스 말고 다른 방법은 없는가?

세마포어가 있습니다.

공유자원에 접근할 수 있는 스레드의 개수를 두어 여러 개의 동기화 대상을 둘 수 있고

세마포어를 소유하지 않는 스레드가 해제할 수 있다 라는 특징이 있습니다.

* 데드레커닝 모델을 사용해서 구현하셨는데 어떻게 생각하게 되셨나요?

처음에는 예를 들어 1초에 한번 이동 패킷을 보내서 방향과 좌표를 보내 주는 방식이었는데 이 방식을 사용했을 때 1초 안에 방향이 크게 바뀌게 된다면 해당 내용이 동기화 되지 않는다고 생각했고 실제로도 끊겨 보이는 현상이 발생했습니다.

그래서 전송주기를 짧게 조정하다 보니 비효율 적이라고 생각했습니다.

그러던 중 관련 서적에서 데드레커닝 모델을 통한 방법에 대해 알게 되었고 전송주기를 통한 방식의 문제점들을 해결할 수 있는 방식이라 생각해서 사용하게 되었습니다.

* 정확성을 위해 2초마다 동기화를 위한 패킷을 보낸다고 하셨는데 동기화는 왜 필요한가요?

데드레커닝 모델과 실제 모델은 직선으로 이동할 때 차이가 발생할 수 없습니다.

그렇기 때문에 패킷을 보내지 않게 되는데 서버프로젝트와 클라이언트 프로젝트 간의

Delta Time 연산은 각자 해주고 있었기 때문에 좌표가 서서히 틀어질 것이라고 생각했습니다.

실제로 오랫동안 직진을 하다가 방향을 바꿀 경우 좌표가 틀어지는 상황이 발생했고 이를 위해 리프래쉬 해주는 패킷을 2초마다 전송하게 되었습니다.

* 포폴을 보니까 좌표차이가 발생했을 때 데드레커닝 모델을 객체와 동기화 해주면서 Speed변수는 동기화 해주지 않았는데 이유가 따로 있나요?

졸업작품 기획 상으로는 대쉬 기능이 있었는데 걸을 때는 문제가 없었지만 대쉬를 하면 속도와 애니메이션이 따로 노는 듯한 현상을 해결하지 못했고 그에 따라 대쉬 기능을 보류해 두며 speed관련 연산을 제외 했었습니다.

* 그러면 스피드를 define이나 전역으로 선언해서 사용해도 되는 것 아닌가요? 굳이 객체나 패킷에 speed를 넣어야 했나요?

작업 속도 문제로 구현하지 못한 부분이 많아서 졸업작품이 끝난 후에 추가 작업을 하려했기 때문에 확장성을 위해 빼지 않았습니다.

* 추가 작업은 했나요?

아쉽게 졸업작품 끝나자마자 네오플 공고가 올라와서 작업하지 못했습니다.

* 왜 보간 알고리즘으로 큐빅 스플라인 보간법을 사용하셨나요?

(말했다면: 아까 말씀드렸던 보완이 필요한 부분이 이 부분입니다.)

인접한 자료점 들 간에 매끄러운 곡선을 그리는데 사용되는 방법으로

* 왜 큐빅 스플라인 보간법을 사용하셨나요?

사실 졸작을 진행할 당시에는 보간을 하면 좀더 매끄러운 움직임이 있겠구나!!

라고 생각해서 사용했습니다.

그런데 공부를 하다 보니 사실 불필요하지 않았나? 라고 생각했는데 그 이유는

애초에 서버와 클라이언트 간의 좌표차이가 눈으로 판별하지 못할 만큼 적은 값 이었는데 이를 아주 작은 시간인 RTT를 나눠 보간을 하면 의미가 없다 라고 생각했습니다.

지금 해 놓은 것은 보간이 아니라 단순 패킷이 송수신되는 시간만큼 좌표차이가 발생하는 것을 보완해주는 보정이라고 봐야 하는 게 맞지 않을까? 라고 생각해서 입니다.

그리고 결정적으로 큐빅 스플라인에 대해 제대로 모르고 사용했지 않나 생각합니다

지금 사용한 방식은 보간된 점을 구하여 이동을 진행하는 것인데

그게 아니라 점을 연결하는 그래프의 방정식을 도출해 내어 그래프에 따라 이동해야 제대로 된 보간이 아니었을까? 생각합니다.

* Monster의 이동에 대해 자세히 설명해 주실 수 있나요?

우선 모든 몬스터를 아무 행동도 하지 않는 Sleep상태로 만들어 놓은 뒤

시야에 플레이어가 들어오게 된다면 Wake\_Up\_Monster 함수를 호출합니다.

호출된 함수에서는 CAS함수를 통해 상태에 대한 검사를 진행 한 후

Timer 스레드에서 해야 하는 작업내용을 저장하는 timer\_queue에 넣어주게 됩니다.

Timer queue란 방향을 갱신해 줘야 하는 주기가 가까워진 순 정렬된 우선순위 큐 인데

작업이 있다면 PostQueuedCompletionStatus 함수를 사용해 Worker\_Thread를 깨워주게 됩니다.

워커스레드에서 해당 작업을 받게 되면 직접 연산을 수행하지 않고 연산에 필요한 데이터를 업데이트 해 주고 실제 움직임은 Processing\_Thread에서 진행 합니다.

* PostQueuedCompletionStatus는 어떤 함수인가요?

Completion Port객체를 생성한 후 PQCS함수에 핸들을 인자로 넣어주면 객체의 스레드에서 GQCS함수 호출을 통해 패킷을 전달 받아 처리할 수 있게 합니다.

* 우선순위 큐 사용하셨는데 우선 순위 큐가 뭔지 설명해 주세요

선입선출의 방식이 아니라 우선순위가 높은 데이터가 먼저 나가는 형태의 자료구조 이며 일반적으로 힙을 사용하여 구현합니다.

힙이란 완전이진트리 형태이며 우선순위가 가장 높은 데이터가 루트에 위치한다.

* 완전이진트리는 뭐죠?

노드를 삽입시에 왼쪽부터 차례로 삽입하는 트리입니다.

* 왜 힙으로 구현하나요? 다른 방법은 없나요?

배열로 구현한다면 삽입과정에서 뒤의 데이터까지 인덱스를 옮기는 단점이 있고 최악의 경우 모든 인덱스를 탐색해야하는 단점이 있고

연결리스트로 구현 시에 삽입과정에서 위치를 찾는데 n의 시간복잡도가 나옵니다.

하지만 힙은 삽입 삭제 시에 부모와 자식 간의 비교만 이루어져 log2n의 시간 복잡도를 가집니다.

* CAS 함수란 무엇이고 왜 atomic\_compare\_exchange\_strong 함수를 사용해서 구현했나요?

CAS알고리즘 이란 Compare and Swap의 약자로

현재 현재 값과 기대값이 일치한다면 새로운 값으로 변경하고 true를 반환 일치 하지 않는다면 false를 반환하는 알고리즘으로 논 블로킹 알고리즘 입니다.

해당 함수는 lock의 내부에서 작동하는 함수로 알고 있습니다.

* Cas를 사용한 코드가 wait free가 맞나요?

그렇게 알고있었는데 100퍼센트 맞는 말은 아닌 것 같습니다.

* 왜그렇게 생각하시죠?

(Atomic\_compare\_exchange\_strong함수는 mutex의 lock 내부에서도 사용되며 lock내부에서 성능이 떨어지는 것을 막기 위한 안전장치가 있다는 있기 때문에

해당함수만 사용한다면 lock보다 성능이 떨어진다고 알고 있습니다.)

CAS알고리즘을 사용하여 기존의 자료구조를 wait free 자료구조로 변환할 수 있는 거지 CAS 자체가 wait free를 보장하는 알고리즘은 아닌 것 같습니다.

* 그럼 lock free 와 wait free는 뭔가요?

Lock free는 lock을 사용하지 않고 멀티 스레드에서 동시에 호출해도 정확한 결과를 보증하며 논블로킹으로 동작합니다.

그리고 스레드간의 충돌이 일어났을 경우 적어도 하나의 승자가 있으며 승자는 딜레이 없이 완료되는 알고리즘 입니다.

Lock는 기아현상 발생 가능성

Wait free는 lock -free 알고리즘 중 하나이며 lock -free가 가지는 조건을 만족하며 다른 스레드와의 충돌에도 모두 delay 없이 완료되어야 한다는 조건이 붙는 알고리즘입니다.

* 논 블로킹과 블로킹 알고리즘에 대해 아나요?

블로킹 알고리즘이란 스레드 간에 충돌이 일어난 경우 승자가 작업을 마치고 다음 스레드가 작업을 하게 되는데 승자가 작업을 마쳤는데 컨텍스트 스위치가 일어나 다음에 작업해야할 스레드가 실행하지 못하는 경우가 생기는 것을 블로킹 알고리즘이라 하고

논 블로킹 알고리즘에서는 승자가 작업을 하더라도 유한한 단계까지만 기다리고 그 시간이 지나면 작업을 진행하도록 하는게 논블로킹 알고리즘 입니다.

* 모든 논블로킹은 락프리 인가?

아니라고 알고있는데 자세히는 모르겠습니다.

* 락을 사용하지 않으면 전부 락프리 인가요?

그건 아니고 다른 스레드의 행동을 기다리지 않고 실행하는 것을 락프리라고 합니다.

* Wait – free는 lock free 인가?

ㅇㅇ

* 데이터 베이스쪽 다루신거 보니까 저장 프로시져를 사용하셨는데 사용했을 때 장 단점이 있나요?

DB서버에 저장해 두고 저장된 이름을 호출해서 접근 하는 방식입니다.

성능, 보안 측면에서 장점이 있습니다.

하나의 요청으로 여러 SQL문을 실행 할 수 있기 때문에 네트워크 소요 시간을 줄일 수 있어 성능적인 측면에서 뛰어나고 소스만으로 데이터 베이스의 구조를 알 수 없기 대문에 보안적인 측면에서도 좋습니다.

하지만 문자나 숫자 연산에 저장 프로시저를 사용하면 성능이 떨어진다는 단점과 디버깅이 얼렵다는 단점이 있습니다.

* Upsert 구문을 사용한 이유가 있을까요?
* Recvn을 사용한 이유가 있나요?

수신 버퍼에 받은 데이터가 보낸 데이터 만큼 다 오지 않는 상황에서 recv함수가 불리면 데이터가 다 받아지지 않기 때문에 대기 시간이 생기게 됩니다.

이런 상황을 방지 하기 위해 받을 데이터의 크기를 미리 알 수 있게 하고 recv함수를 여러 번 호출해 대기 시간을 줄일 수 있기 때문에 사용했습니다.