로직과 랜더를 분리시키면 클라에서 동기화는 어떡하나? 로직도는동안 락걸고 랜더하는동안 락걸면 멀티쓰레드가 쓸모없어진다. 그래서 이때 활용한는것도 사본을 만들어서 랜더하는방식. 실제 로직에서는 로직 수행. 당연히 로직쪽이 가벼운 상황이니까 로직에서 사본까지 만들어주는게 가능. 랜더링에 필요한 재료까지 로직스레드에서 마무리하고 랜더링스레드에서는 그것만 대상으로 락걸고 그걸 랜더링하는것도 방법. 근데 하다보면 사본을 만드는데 더 시간이 오래걸릴수있다. 거의 대 부분이 멀티스레드 만들 때 더 역효과가 난다. 병렬로 돈다 해도 불필요한 작업이 있다. 사본만들고 그런 오버헤드가 너무 크다면 안하는게 나을수있다는거. 스레드로 분리시킨다는 그 행동이.

지금 과제 같은 경우에서 스레드들이 많고 cpu사용률이 높은 상황에서는 그냥 통째로 락걸고 파일쓰는게 나을수있따. 지금은 경합이 발생할만한 데이터가 list 하나인데 다른스레드에서 락걸면 안쓰고 끝이다. 근데 컨텐츠가 많으면 그냥 단시간내에 사용하는게 나을수도있따는거. cpu사용률이 높다는건 다른 로직들이 못돈다는거. 여러스레드가 동시에 막 여러 데이터의 동기화 객체가 도는 상태에서 하나의 로직과 하나의 컨텐츠 데이터만 봤을 때 하나는 락걸고 하나는 도는게 병렬로 도는 의미가 없어보이지만 전혀다른 컨텐츠를 돌게된다면 상황에 맞게 설계해야한다는거.

액터패턴

한쪽에서 신호를 주면 뭘 하는 방식 지금도 하고있었는데 이제 좀 더 일감을 생산하는쪽 처리하는쪽 스레드를 나눠서 생각해보자. 서버에서는 이 구조를 버릴수 없다. 스레드 하나가 일을 만들어서 던져주는거. 이게 액터패턴. 생산자 소비자 패턴으로 부르기도 한다. 근데 일을 어떻게 던져주나? 지금은 set event같은걸로 깨운다 정도였는데. 글머 이건 정말로 job덩어리들을 쌓아둬야 하는 상황. Workerthread에 할일들을 쌓아두고 일을 뽑아 쓰는 것. 그럼 job qeueue가 나와야 한다. 그럼 할일이 없으면 쉬고 할일이 있으면 하는게 이게 액터패턴이다.

리액터는 일하고 마무리까지 다 하는거, 프로액터는 누군가 일을 하고 결과를 받는거. 오버랩io를 쓰면 다 액터패턴이 된다. 이미 오버랩io에 비동기입출력에 iocp를 쓰면 이미 액터패턴이다. 일을 하고 마무리까지 하면 리액터. 근데 누가 일을 하고 workerthread에 완료만 주고 그 일에대한 마무리만 하면 그건 프로액터라고 한다. 사실 별차이는 없다. 마무리만 한다? 액터패턴을 이런식으로 만들었다고 하자

Job이 길찾기고, workerthread는 열심히 길을 찾았다. 그럼 얜 뭘 해야하나? 길찾기의 결과를 누가 쓸건지. Workerthread는 길찾는 목적이고 사용은 컼ㄴ텐츠는 원래 메인 로직스레드. 근데 그럼 동기화 이슈가 생긴다. 길찾기 스레드를 따로 뒀다는건 원래 로직스레드와 경합요소가 없으니 분리해뒀던건데, 완료 결과를 준다면 로직스레드에서 완료를 해야되는상황. 그래서 일 단은 이런 멀티스레드 환경에서 일감을 스레들도 분리시켜서 이렇게 일을 주는상황이 되면 일에대한 완료처리는 요청자가 마무리까지 하게되는게 일반적인 상황. 이 느낌으로 완료를 보면 이건 또 프로액터다.

지금은 스레드마다 일감을 분리해서 이름을 붙여서 스레드를 분리했었는데, 그냥 job을 던져주면 workerthread가 깨어나서 그 업무를 진행하는 방식으로 하자. 그럼 큐가 필요함.

그리고 이제 이 큐에 뭘 넣어야 하냐? Job의 덩어리.

DB도 나중에 이런 방식으로 할건데 읽기는 왜 비동기가 어려울까? 아직 읽어지지 않은 상황에서 리턴하는거. 비동기라는 함수 자체가 액터패턴이다. 나중에 알려줄게. 데이터가 지금 필요해서 읽으라고 했는데 결과가 언제나올지 모른다. 그럼설계가 복잡해진다. 그래서 읽기는 어차피 로그인할때 STATEFUL은 그러니까 읽어올떄 동기로 하고, 저장은 비동기로 100%설계할수있으니까. 저장은 결과가 필요없으니까 그냥 잘 되면 된다. 저장이 안되면 그냥 난리난거니까 꺼야하고. 저장이 안되는 상황은 그냥 고려를 안한다. 저장하다 실패했으면 그 다음것도 저장하면 안된다. 애초에 DB저장은 오류가 전제되지 않는다. 피드백이 필요가 없는 구조다 .그래서 스레드로 빼기가 깔끔하다.

그럼 저장을 한다 치면 저장 할 건덕지를 JOB으로 던지는데 이 JOB에대한 요소는 뭐가 될까? ITEM이면 능력치랑 아이템 ID랑 있는데 뭘 던지냐? 실제 데이터 모든걸 다 보내야한다. 객체를 보내는게 아니라 데이터를 하나하나 다 보내야한다. 그럼 진짜로 memcpy해서 데이터를 싹 넣고 workerthread에서는 그 일을 뽑아서 한다. 그래서 이런 구조라면 큐가 10m이상이다. 성능도 떨어질건데 개선시켜야한다.

그래서 메시지 타입과 길이를 전달하는 모양새가 나온다.

읽는 스레드 1개, 쓰는 스레드 1개라면, 락이 필요한가? 쓰는곳과 읽는곳의 위치가 다르다. Front는 읽기만 rear는 쓰기만. 그래서 잘만 만들었으면 전혀 상관이 없다. 근데 workerthread가 여러개면 동기화를 걸어야 한다.

그래서 링버퍼 안에 그냥 동기화 객체를 갖고 있어야 한다. 그리고 링버퍼에 그냥 lock과 unlock을 넣을거다.

enq안에다가 락언락을 바로 넣지 않은게, 지금은 header랑 msg를 따로 넣는데 그럼 다른스레드가 치고오면 짤릴수있다. 그래서 사용자측에서 함수를 호출해서 락으로 감싸게 했다. Tcp가 아니니까 우리가 넣고 빼니까 원래 헤더길이 이런거 체크 안해도 된다.

Workerthread pool에는 recv에대한 완료 통지가 온다. 내부에서 recv를 하고 완료통지를 해줌. 다 받았다고. 그럼 workerthread pool을 여러 개를 만든다. 원래 구조라면 하나씩하나씩 완료통지가올때마다 각자 받은메세지에 대한 패킷에 대한 packetproc를 각자해야한다. 맵이나 플레이어 리스트, 몬스터리스트, 아이템정보들, 퀘스트 등 컨텐츠데이터가 있을것. 원래 iocp 목적은 workerthread 여러 개 두고, 메시지가 완료통지떠서 받았으면 직접 컨텐츠를 처리하는거. 그럼 이론적으로는 workerthread를 늘리고 코어를 늘리면 병렬처리로 빨라진다는거. 근데 실제로는 모든 컨텐츠가 동기화 이슈가 이루어진다는게 문제. 웹서버라면 그냥 그 packet proc 하나 안에서 뭐 만들어서 보내주고 하면 되는거. 근데 게임이라 안된다,. 그래서 대부분 mmo rpg에서 하는건 별도의 업데이트 스레들르 만들어두고, 받은 데이터를 큐잉만 함. 그리고 업데이트 스레드 하나가 단일로 로직을 짠다는거. 그래서 iocp의 구조가 실제로는 io thread가 된다. 실질적인 로직을 수행하는게 아니라는거. 모든 부담은 update thread하나가 전담하는 꼴이 되었다. 안타깝지만 가장 무난함. 업데이트 싱 글스레드 하나 주고 모든 일을 얘 혼자함. 그럼 동기화도 필요없고 편하게 짤수있다.

도저히 이렇게 update스레드 하나로 가는거 이건안되고 멀티스레드로 가야겠다면?

1. Update 스레드를 여러 개 두고 큐를 하나로 두고 여러 스레드가 뽑아서 처리 🡪 하나의 유저가 보낸 데이터의 순서가 뒤집히는 경우가 있을수있다. 그렇다고 유저당 1스레드씩 가는 구조는 안된다. 웹서버는 그렇게 가고있고. 그럼 이런 구조에서는 어떻게 해야할까. 이런 구조라면 업데이트 스레드마다 큐를 둬야한다. 서로 들이대서 뽑아오는건 말이 안되고. 그러면 스레드마다 큐가 나오면 ? 그럼 순서가 뒤집히는 문제를 해결하기 위해서는 넣을 때 유저별로 스레드가 고정이 되어야한다는거. 유저마다 지정형태. 근데 이 구조도 주장하는 사람들이 많다. 그사람들의 이론은 일단은 메시지 수신받은걸 병렬로 처리하고자 하는 문제. 근데 일단 메시지가 온건데 왜 큐에 넣어서 update thread에 전달하고 유저마다 스레드 정해서 던지냐는거. Wortker thrad 가 그냥 하면 안되냐? 이 스레드 자체는 io에대한 send recv에 대한 완료처리가 너무 중요하니까 그 일을 전담하는걸로 살려두자는거. 얘네가 각자 다 io가 아닌 실제 일을 하면 send recv가 안될수있다는거. 그럴싸한가? 어차피 그러면 update스레드 큐도 밀릴건데 그게 그거 아닌가 싶기도. 이럴바엔 workerthread가 하는게 낫지 않을까 싶고.

그럼 이 다음 job을 던져줄떄 동기화 이슈를 어떻게 할거냐. Map이 있다면 map manager가 있고. 플레이어 등등… 덩어리가 많다. 우리가 이거 하나하나 동기화 객체만들어서 이거쓸 때 락걸고 저거쓸때 락걸고 하지 않는다. 동기화는 상호작용을 기준으로 본다. **맵이 나누어져있으면 그게 동기화의 단위가 된다**. 퀘스트나 아이템 이런게 다 플레이어 안으로 들어가거나 단위단위로 들어갈거. **게임 컨텐츠상 물리적으로 떨어져있는 구역이라면 동기화를 걸어도 된다는거**. 맵 별로 분리. 모든 유저가 한 맵에 몰려있지 않는 이상 멀티스레드로 가능. mo게임이면 방개념이면 방 단위로 락을 건다. 이렇게 설명하면 또 잘못이해해서 맵별로 스레드가 가는게 아니다. 그냥 락을 여러 개 스레드가 맵 별로 걸고 들어간다는거. 인던 들어가도 다른 던전애들끼리 경합이 없으니 락 걸고 들어가는거.

여기서 더 들어간다면? 원필드, 오픈월드 이런건 맵이라는 구분이 없다면? 섹터단위로 해도되는데 너무 자잘하기도 하고. 이동이라는 단위의 개념이 맵으로 분리된거랑 느낌이 다르다. 맵이나 던전은 아예 분리되어있어서 주변을 못봄. 근데 섹터는 주변이 보인다. 그럼 락을 주변 9개씩 건다. 이렇게 오픈월드라면, 플레이어가 느끼지 못하게 구역이 분리되어있는 경우가 많다. 바다라던가 묘한지역. 섹터 외에 개념적으로 분리된 구역들이 있다. 하나의 월드처럼 느껴지지만 맵이나 던전을 이동한다는 개념이 들어가는 구역ㅇ l있다. 탈것을 타고 이동한다거나. 수영한다거나. 그럼 분리가 된걸로 봐도 된다. 근데 그런게 없다? 검은사막, 리니지 그런데서 쓰는건 오브젝트 단위의 동기화. 캐릭터 단위의 동기화. 정말 멀티스레드로 동시에 캐릭터 단위로 락을 건다. 각자 읽기용 쓰기용 lock이 있다. 얘에 대한 행위를 변경하려면 lock걸고 한다. 옆에서 한다면 읽기용으로 걸든 쓰기로 걸든. 리니지에서 쓰는 건 오브젝트 하나하나마다 메시지큐방식. 해야하는걸 큐잉한다. 그리고 병렬로 들어가서 메시지 처리. 경합이 발생하면 한놈이 포기하고 빠짐. Try critical section같은 느낌.