헤테로지니어스 / 호모지니어스

헤테로지니어스 : 여러 다른 종류들로 이루어진거 . 기능별로 스레드 분리.

호모지니어스 : 같은일을 하는 스레드 여러 개. 각자 정해진 일을 전담해서 막 하는 것. 이론적으로 스레드 개수를 늘리고 코어 개수를 늘리면 성능이 올라간다는 상황 .

지금 우리가 스레드 풀 만든건 호모지니어스. 근데 데이터베이스 저장같은건 헤테로지니어스로 정해진 것. 얜 두개이상 늘려서도 안된다. 그러면 게임 컨텐츠에 대한 변화된 값이 저장되어야 하는데 그러면 순서가 맞게 차곡차곡 똑 같은 결과를 반영해서 저장해야한다. 서로 뽑아서 쓰면 순서보장이 안된다. 그래서 전담하는 스레드가 단독으로 하나 나와야 한다.

로그인스레드 같은 경우 메시지 처리에 대한 워커스레드라면 일단은 메시지 수신 순서 보장만 되어있다면 네트워크 수신받는 스레드 자체가 여러 개가 되면 그냥 코어 늘리고 스레드 30 40 50개 늘리면 성능은 그냥 좋아진다. 이렇게 설계할수도 있는거다.

게임서버는 서로 경합이 많고 동기화가 많이 들어가서 마냥 스레드가 늘어난다고 좋은게 아니다. 그래서 헤테로지니어스 방식이 더 많이쓰이긴 한다.

과제 32에서 종료할 때 종료 메시지를 스레드 수만큼 enq할수도 있고, 한 스레드가 빼서 보고 종료하면서 또 종료메세지 넣고 끌수도 있고. 아니면 아예 종료메세지면 빼지 않고 다른스레드도 보게 해도 되고.

그리고 과제에서 job이 쌓여있는거 다 처리해야함. 한번 깨어났을 때. 하나라도 블락되어서 기다리고 있으면 상관없는데, 다 뺑뺑 돌고있으면 job이 남아있는데도 일을 안하게 됨. 그래서 한번 깨어나면 일을 다 하고 가야함.

그래서 이 워커스레드 풀을 효율적으로 관리해주는 iocp가 나옴. 자동으로 하나의 메시지당 하나의 workerthread가 불리게. 큐방식을 쓴다는 점에서 위험성이 있다. 우리가 아예 제어를 못한다. 지금은 개수나 상태를 파악할수있ㅈ미ㅏㄴ iocp의 큐는 커널 내부에 있다. 몇 개 있는지도 모르고 모니터링도 못한다. 미친듯이 많으면 그냥 non-paged memory만 엄청 뜬다.

waitformultipleObject 에 결과가 여러 개가 들어오면 앞쪽에 있는 시그널부터 뽑아냄. 그래서 exit를 앞에 배치해서 이게 왔다면 accept가 있어도 exit에 우선순위를 주기위해 핸들 배열 앞쪽에 이벤트를 둔다.

Accept랑 disconnect가 같이 있는건 나중에 lock을 없애고 동기화를 할 때 이점이 있음. 따로따로 하는것보다 만든애가 없애는게 낫다.

그리고 잠갔다 풀었다 여러 번 하는것보다, 경합이 발생할만한 일을 모아서 한방에 처리하는게 낫다. 그래서 겸사겸사 이런 구조를 사용한거다.

과제로 나올건 문제는 11개, 이중에 문제가 안되는건 3개. 데드락 같은 문제는 아닌데 좋지 않은 방식인거.

서버의 구조

1. Update 스레드를 여러 개
2. …
3. …

위는 같은 컨텐츠 로직인데 컨텐츠별로 동기화를 걸었으면 이제는 컨텐츠… 읭..>???

이것들 말고 헤테로 방식으로 한다고 하면

1. 컨텐츠적인 입장에서 유저별로 모으는 방식이 있다.
   1. 로그인을 전담
   2. 로비를 전담
   3. 필드를 전담
   4. 던전을 전담
   5. 얘네들끼리 서로 상관이 없으니까 자기 안에 있는 유저들끼리만 처리한다고 하면 동기화가 없어도 된다. 위에 내용들은 어떤 데이터에 들어가기위해 스레드가 들어가기 위해 동기화를 거는거. 근데 이제는 스레드가 있으면 유저들이 들어가는 입장. 이렇게 나누어져있으면 어쨌든 플레이어를 일괄 관리하는애가 필요하다 . 그럼 로그인스레드다 로비다 그러면 로그인부터 할거. 그럼 로그인성공하면 로비로 가고, 필드갈땐 넘어가는거. 그럼 필드에 대한 컨텐츠만 필드 스레드가 전담. 그럼 얘네는 각자 자기의 루프를 뺑뺑이 돌면서 나한테 들어오는 플레이어를 처리해야하는거. 그러면 각 스레드별로 나한테 들어온 유저들을 관리해야한다. 그럼 거기에도 로그인스레드에 들어와있는 list가 따로 나온다. 통합으로 관리하는 것 외에. 그럼 로그인스레드는 자기 list만 보면서 순회한다.
   6. 문제는 한 장소에 계속 있으면 상관없지만, 스레드간 이동을 해야한다면. 들락날락을 하면 동기화 문제가 필요하다. 플레이어들이 필드에서 로비로 나갈수있는거니까. 그럼 어떡하지?? 로그인에서 로비로 옮기려면 각자 리스트 두개 다 락을 걸어야 한다. 그러면 다른 스레드가 또 나를 락걸려고 할수있으니까 데드락이 충분히 나올수있다. 이런 구조는 되지 않게 해야한다. 100% 데드락이 바로 나온다.
   7. 그럼 여기서의 가장 큰 문제는 동기화 이슈 어쩔수없다고 뺀다 하더라도 데드락의 이슈를 해결해야한다. 그러기 위해선 결국 자기껀 자기가 건들여야한다. 그 규칙을 세워야 한다. 내가 관리하는 리스트는 나만 관리하게 해야한다. 이게 중첩으로 일어나니까 문제가 일어나고 있는거. 그럼 push나 erase 행위는 무조건 나만이 할 수 있는거. 그럼 lock이 없어도 된다. 그럼 리스트간에 이동을 하려면 어떡하나? 내가 직접 건들이는게 아니라 해달라고 요청해야한다. 별도의 연결할 수 있는 스레드간에 서로 데이터를 (플레이어를)전달할 수 있는 매개체가 필요하다.
   8. 그럼 그 매개체는 어디에 어떻게 어떤개념으로 둬야하냐? 각각의 스레드마다 큐를 갖는다. 누군가 들어오고 싶어한다면 큐까지에 넣고 빠진다. 그럼 큐 자체에 동기화는 필요하겠지만 괜찮다. 그 큐만 잠그는거면. 그럼 큐에 들어있으면 판단해서 내가 뽑아내고. 그럼 이제 동기화 건덕지가 없다. 단점은 스레드가 정해져있기떄문에 성능 향상의 한계치가 있다. 그래도 이정도 설계의 분산이어도 충분하다.
   9. 이런 방식이 일반적인 방식이다. 스레드마다 자기가 처리한 애들을 리스트로 관리하고, 삽입과 삭제를 위해 전달매개체를 통해 전달받고. 이게 가장 일반적인 방법
   10. 근데 게임서버를 만드는 구조는 선생님이 만든 구조로 하니까 논란의 여지가 있다. 왜냐면 실제로 유저들은 필드에 몰려있다. 로비나 로그인에 전혀 몰려있지않다. 그래서 한두개 스레드에 동접자 전체가 쏠리면 의미가 없어진다. 불필요하게 lock 처리때문에 성능만 떨어질거같고. 그래서 여기서 이제 로그인 스레드는 어쩔수없이 필요하다. DB를 쭉 읽어올 때 시간도 느리고 로그인 스레드는 그래서 꼭 있어야함. 그래서 보통은 로비와 실제 게임으로 나온다. 결국 따지고보면 모든 유저는 실제게임스레드에 있는거고. 여기에 로그인까지 하면 너무 오래걸려서 느려서 서버가 멈춘다. 그래서 별로 로그인은 떼는거다. 사용자가 많지 않더라도.
   11. 이런 상황이면 실제로 5천명이 다 들어있는데 리스트를 순회한다. 그럼 큐에대한 확인도하고. 그럼 어떡하지? 리스트를 우리가 락 거는 이유는 뭐지? 무조건 한 스레드만 접근해야하는 상황이 왜냐? 삭제가 문제다 없어져버려. 그래서 이걸 원천적으로 해결하려면 배열로 하면 된다. 메모리가 없어지지 않으면 된다. 근데 걔가 걔인지는 모르겠지만 일단 데이터는 있으니까 메모리 참조 오류는 나지 않는다. 찾아갔는데 걔가 아닐 수는 있지만. 그럼 굳이 리스트로 가야하냐는거. 어차피 동접자수는 MAX치를 정해서 고정으로 가니까. 원활하게 플레이할 수 있는 최대치를 정해서 가는거. 7천명이라고 하면 7천개 만들어서 가는거. 굳이 리스트로 할 이유가 없음. 동접이 꽉찼을때 MAX일 때 가장 빠른선택을 봐야하는거. 그럼 굳이 LOCK걸면서 리스트로 갈 필요는 없어보임. 근데 넣었다 뺐다 배열에 그러는것도 이상하니까. 전체 관리 데이터를 LIST말고 배열로 가자는거.
   12. 각자 스레드에 리스트랑 배열 아무것도 쓰지말고 처음에 전체 플레이어 관리하는 배열만 쓰자는거. 그럼 락을 걸지 않아도 된다. 모든 스레드가 다 들어와서 데이터를 읽고 쓰고 한다. CRASH는 나지 않음. 그래도 여전히 문제는 있다. 데이터가 엉망진창이 되는거. 내가 어떤 소속인지를 플레이어가 알고있으면 된다. FLAG 같은 값으로. 그럼 하나의 스레드가 한명의 플레이어만 쓴다.