- **멀티스레드 프로그래밍의 사용처**
  - 과학기술계산 (오래전부터)
  - 멀티미디어
    - Encoding & Decoding
    - 멀티스레드의 구현이 매우 쉬움.
      - 독립된 장면들 (key frame 단위)
  - 게임
    - 온라인 게임 서버 1997년부터
    - 3D 게임엔진 2000년대 중반부터

역사가 상당히 오래됐다. 게임하는 사람들이 신경 쓰게 된 건 90년대 후반, 리니지 1 때. 원래는 어디 대학에서 연구하는 연구논문이고 실생활과 관련이 없었다. 사실은 그 오래전부터 과학기술 계산용으로 많이 사용하고 있었고, 슈퍼컴퓨터에서 많이 쓰고 있었다. 우리나라 대학교에서는 슈퍼컴퓨터가 1대뿐이었다.

요새는 어디에 쓰느냐? 멀티미디어 분야에 많이 쓰이고 있다. 동영상 인코딩, 디코딩에 쓰인다. 요새는 자동으로 해주기 때문에 좋은 동영상을 다운로드하여 보면, 그냥 옮기면 안 되고 디코딩을 새로 해주어야 한다. 시간이 매우 많이 걸린다. 근데 코어의 개수가 많으면 많을수록 빨라진다. 코어의 개수에 비례한다.
그래서 인코딩 디코딩은 멀티코어가 짱이다. 또 동영상 하나 하는데도 많이 도움이 된다. 동영상은 프레임 단위로 짜여있기 때문에, 코어가 4개면 4개의 프레임을 한꺼번에 쓰는 것이다.

그리고 게임. 우리가 게임공학부라 게임을 이야기하는 게 아니고, 다른 운영체제 교과서를 봐도 예제로 드는 것이 게임이다. 게임 말고 멀티코어를 쓰는 곳은 거의 없다. 우리 컴퓨터를 쓰면서 멀티코어가 필요하다고 느끼는 것은 거의 없다. 웹 브라우저가 느려서 못써먹겠다 이런 거 없다. 서버가 버벅거리거나 네트워크가 느려서 문제가 되는 경우는 있어도.

멀티스레드 프로그래밍이라는 분야를 적극적으로 도입한 건 게임뿐이다. 데이터베이스 정도면 도입하고 있다. 과학기술 계산이나, 일반인들이 쓰는 건 게임 말고는 거의 볼 수 없다. 포토샵에서는 좀 쓰는지 모르겠다. 포토샵, 3D MAX 이런 건 멀티미디어 케이스이다. 그래서 게임을 예시로 든다.

엔씨가 지금까지 승승장구하는 것은 이 기술 때문이다. 리니지 이후 수많은 짝퉁이 나왔지만 모두 망했다. 콘텐츠가 부족해서? 그래픽이 리얼하지 않아서? 아니다. 모두 IOCP를 가지고 했지만 아무나 멀티스레드를 할 수 있는 것이 아니다. 서버가 불안하고, 동접이 적고. 렉 걸리고 하면서 망하게 되는 것이다. 그 이후로 수많은 게임들이 있었고, 괜찮게 살아남아 잘 나가는 게임들이 있다. 테라 같은 게임이다. 엔씨의 서버 프로그래머가 퇴사하고 그 기술로 만든 게임이기 때문이다. 그것 말고 살아남은 게 별로 없다. 10년 전까지만 해도 그 개발자가 아니면 만들 수가 없었다. 우리나라 게임 경쟁력이 저때 시작되었다.

서버는 게임회사에서 사서 설치하는 거니까 멀티코어 CPU가 아니라 멀티 CPU. 몇천만 원짜리 PC인데 서비스할 수 있었지만, 클라이언트는 아니다. 클라이언트는 멀티스레드로 짜여서 고성능이지만, 1000만 원짜리 PC를 집집마다 보급시켜줄 수 없었다. 할 수 있는 것은 PC방과 협력해서 최신 그래픽카드를 보급하는데 돈을 보태준 정도. 리니지 2 서비스하면서 GeForce 4를 pc방에 보급하는데 돈을 보탰다. 2002년도에. 2000년대 중반, 클럭 속도를 인텔이나 AMD가 높일 수 없어서 멀티코어 CPU를 팔기 시작했다. 그러니까 3D 게임엔진 만드는 회사들도 성능을 높이기 시작했다. 얼마나 화려하게 보이느냐 그건 쉽다. 화려하고 리얼해 보이지만 성능이 제대로 나오는 것 그게 어려운 것이다. 항상 CPU가 느려서 골치였는데, 멀티코어 CPU가 나오니까 멀티코어로 대거 전환하기 시작했다. 그러면서 멀티코어 게임엔진이 나오고 멀티스레드 게임엔진이 나오고 유명해졌다. 가장 잘 나가는 것은 언리얼 4이다.

- **Unreal 4**
  - 3D 게임엔진
    - 3D 그래픽을 구현하는 라이브러리와, 콘텐츠 제작 툴들의 집합.
    - 비싼 엔진. 10억 정도
    - 플랫폼
      - PC, PS4, XBOX-One, IOS, Android
    - 2012년 공개
    - Heterogeneous 멀티스레딩

게임엔진은 웬만해서 언리얼을 쓴다. 유니티도 유명한데 이건 어디에 쓰느냐? 그래픽으로 승부하는 게 아니라 캐주얼 게임 그런데 쓴다. 본격적으로 실사풍의 대용량 사실적인 그래픽 고퀄리티 그래픽을 요구하는 건 언리얼을 쓴다. 왜? 유니티는 그런 걸 못하냐? 아니다 할 수 있다. 언리얼 뺨 때리는 그래픽, 만들 수 있지만 똑같은 성능이 안 나온다. 여기서 말하는 성능 차이는 멀티스레드에서 나온다. **언리얼은 멀티스레드로 구현되어있다. 그래서 코어가 늘어날수록 성능이 좋아진다**. 유니티는 조금 올라가긴 하는데 조금... 그래서 고사양 게임 시장에서는 유니티가 싹 망하고 언리얼이 대세다. 유니티가 우세할 때도 있었지만 역전되었다.

사실 지금 우리가 볼 수 있는 3d게임엔진 중에서 유니티 빼고 다 멀티스레드이다. 하프라이프, 헤일리, 아케이드 엔진 다 멀티스레드이고 유니티는 저사양 게임으로 버티고 있다. 그럼 왜 유니티는 멀티스레드를 안했냐? 어렵기 때문에.

언리얼 얘기를 하면 언리얼 4는 3D 게임엔진이고 그래픽을 구현하는 라이브러리와 콘텐츠들 스토어, 다 합쳐서 거대한 콘텐츠를 이루고 있다. 얼마나 그래픽 퀄리티가 높냐? 에픽 홈페이지에 가서 데모를 보면 알 수 있다.

그리고 무척 비싸다. 10억 정도 한다. 공짜로 쓸 수 있지만, 매출이 1년에 1000만 원이 넘어가는 순간부터 매출의 30%를 줘야 한다. 1천만원을 벌면 매달 3백만 원을 주어야 한다. 공짜로 쓰는 건 없다. 약관이 다 깨알같이 써있다. 서비스할 때 매출에는 제작비, 운영비 그런거 다 포함한건데 그 중 30%를 주려면 매우 뼈아프다. 출시 전 에필에 얘기해서 30% 안 떼줄테니, 사용권을 팔아라 할 수 있다. 그럼 10억원을 줘야한다. 10억에 팔지 말지는 에픽이 정한다. 그렇게 손해 본 게임이 배틀그라운드이다. 배틀그라운드가 잘 될지 아무도 몰랐고, 10억 안 내고 30%를 에픽에 계속 내고 있다. 30억 줄게 30% 없애자 했지만 에픽이 거절했다. 그래서 지금 배틀그라운드 2를 만들고 있다. 자체 엔진을 써서.

그래서 옛날에 보면 모바일 게임은 카카오 연동이 대부분이었지만 지금은 아니다. 매출의 30%를 카카오에게 줘야 하기 때문에. 처음에 애니팡 같은 개발비 안 드는 거 30% 아무것도 아니지만, mmorpg처럼 대작게임 만들고 30% 카카오에 주는 것 손해다. 그래서 요즘엔 카톡 연동을 잘 안 한다. 구글이나 애플에 떼주는 게 30% 아직 남아있다. 모바일 게임의 치명적인 약점이다. 피할 수가 없다. 안 떼주려면 앱스토어에서 내려가고 게임 망하니까. pc는 그런 거 없다. pc가 지금까지 살아남은 것도 30% 안 떼주기 때문이다.

또 특징은 플랫폼이 pc, 엑박, ios 다 돌아간다. 유니티도 마찬가지. 언리얼 4는 2012년에 공개되었고 처음부터 멀티스레드였다. 그때는 멀티코어 cpu가 없기 때문에. 언리얼 3은 싱글 스레드였다. 한 번 언리얼 엔진을 밑바닥부터 갈아엎었다. 2003년도에 1년 동안 아무것도 안하고 멀티스레드로 전환만 1년동안 했다. 그래서 전환을 해서 멀티코어에서 성능을 높여 출시를 했고, 3부터 유니티를 이기기 시작했다. 멀티스레드라 빨라요 끝. 이게 아니라 어떻게 구현되었나? 헤테로지니어스. 헤테로는 뭐냐? 여러 가지 서로 다른 스레드들이 멀티스레드니까 여러 개가 있는데 스레드마다 하는 일이 다 다르다. 반대는 호모 지니어스. 모든 스레드가 하는 일이 똑같다. 그래서 **언리얼 4는 헤테로 지니어스로 되어있다.**

그런데 앞에서 스레드 기능을 나누지 말라고, 모듈화를 하지 말라고 했었을 것이다. 그런데 언리얼에서 이렇게 한 건 옛날에 잘 모를 때 그렇게 해서 그렇다.

언리얼로 만들면 굉장히 그럴듯하다. 언리얼로 만들어주니까. 그래서 졸업작품을 언리얼로 제작하게 되면 그래픽 퀄리티를 잘 안 보고 콘텐츠를 본다. 퀄리티는 언리얼이 해주니까 심사할 이유가 없다. 퀄리티만 보면 다들 박사 졸업 수준이다.

- **Unreal 4의 스레드**
  - 3개의 주 스레드
  - 많은 개수의 worker thread
    - 주 스레드가 생산한 작업을 병렬 수행.

스레드를 어떻게 하느냐? **헤테로지니어스(Heterogeneous) 방식으로 되어있다. 스레드마다 하는 일이 다 다르다. 세 개의 메인 스레드로 구성이 되어있다. 스레드는 수십 개가 돌아간다.** 수십 개를 쓰고 있다. 근데 다 바쁘게 돌아가는 게 아니고 바쁘게 돌아가는 건 3개의 스레드이다. **게임 스레드, 렌더스레드, 오디오 스레드. 이 3개가 기본적인 스레드이다.** 게임 스레드는 게임로직. 움직이고 애니메이션하고. 버튼누르면 문 열리고, 그런거 다 여기서 하는 것이다. 총알 나가고 데미지 깎이고. 게임로직은 다 게임스레드에서. 그래서 언리얼 3엔 게임스레드가 있고 혼자하는건 아니고 여러가지 부수적 작업이 있으면 작업들을 나누어서 다른 스레드에 넘겨준다. 게임스레드는 물리 계산이 메인이다. 총알 맞고 문 열고 하는 계산은 시간이 별로 안 걸린다. 이건 8비트 cpu로도 충분하다. 문제는 충돌처리. 총알이 가면서 맞았느냐 내가 가는데 좁은 문에 끼이느냐. 돌이 떨어지는데 내 발등이 찍혔나. 물리에서 시간을 굉장히 많이 쓴다.

랜더링 스레드는 게임 스레드 돌리면 나오는 결과 애니메이션을 화면에 어떤 이펙트로 표현해라. 비행기가 어디서 어디로 이행했다. 그 결과들을 가지고 랜더링을 하는 것이다. 랜더링은 뭐냐 이 게임의 수많은 오브젝트가 있는데 어떤 오브젝트가 카메라에 보이는가, 이 상태는 어떻게 되어있고 어떤 애니메이션 어떤 포즈를 취하는가 그리고 gpu에 떠넘긴다. command buffer 이런 거 다렉 12에서 썼을 텐데, 언리얼에서는 여러 버퍼 관리를 멀티스레드로 할 수 있다. 또 종료되길 기다렸다 다음 씬으로 넘어간다.

오디오 스레드. 백그라운드 뮤직, mp4로 되어있는 거 풀어넣고 총 사운드 넣어주고 발자국 사운드 3d사운드로 해야 하는 거 플레이어 위치 기준으로 왼쪽 오른쪽 위성 차이를 계산해주고 다 오디오 스레드에서 해준다. 오디오 디코딩 그런 거 하고, 오디오 하드웨어에 넘겨준다.

우직하게 루프를 도는 게 아니라 작업들을 잘게 쪼개서 실행하고 쪼개진 작업들은 다른 스레드에 떠넘기기도 한다. 내가 다 실행하지 않고 다른데 떠넘길 수 있는 거 워커 스레드에 떠넘긴다.

이렇게 3개의 스레드로 되어있고 얘네가 모든 걸 다 하는 게 아니라 작업을 나누어서 하고, 멀티스레드로 할 수 있는 건 작업 나눠서 쪼개고 그러니 스레드는 3개보다 더 많다. 그런데 한계가 있다. 이 3개가 보틀넥(bottleneck)이다. 싱글코어로 돌릴 때보다 듀얼코어가 2배 빠르고, 트리플코어에서 3배 빠르다. 쿼드코어에서는 4배까지 빨라진다. 왜? 나머지에서 하니까. **그런데 문제는 그 이상으로 코어를 8개 16개 늘렸을 땐 거의 빨라지지 않는다. 보틀넥이기 때문이다.** 아무리 편안하게 실행한다 하더라도 하나가 끝나지 않으면 다음으로 넘어갈 수 없다. 병목현상이 일어난다.

그래서 언리얼의 가장 큰 문제는 쿼드코어 이상에서는 성능 향상이 없다. 뭐가 문제냐? 게임기가 문제가 된다. 플스 4는 cpu가 옥타코어이다. 거기서 성능이 좋지 않다. 콘솔로는 언리얼로 만든 대작 게임이 별로 없다. 한계가 있기 때문이다. 헤일로 시리즈니 그런 건 자체 개발 엔진을 쓴다. 스레드가 3개가 아니라 4개 16개 사용하는 완전히 멀티스레드에 특화된.

- **3D API의 변천**
  - OpenGL -> Vulkan
  - D3D11 -> D3D12

그리고 이제 3d api도 많이 바뀌었다. 다이렉트x 오픈지엘. 모바일은 오픈지엘이다. 모바일에서는 다렉이 안 돌아간다. 모바일에서 3D를 하고 싶다, 그럼 오픈지엘을 써야 한다. 그러나 이건 과거의 유물이 되었다. 더 이상 서비스하지 않는다. 물론 **AS는 해주는데 새 버전이 더 안 나온다. 다음 버전인 Vulkan이 나왔다**. 다렉 9 이후에 다렉 12로 갈아타는 수준이다. 오픈지엘 쓰던 사람은 벌칸으로 갈아타야 한다. 오픈지엘 4, 5 안 나오고 왜 벌칸이 나왔냐? 호환성이 없어서 바닥부터 새로 짰다. 파라미터 위치 바꾸고 그렇게 하는 게 아니라 완전히 새로 짰고 그래서 새로 이름 붙였다. 왜 그런 미친 짓을 했을까? 업그레이드하고 성능 올리지 왜 전혀 생소한 인터페이스로 바닥부터 새로 삽질을 하느냐. 다양한 이유가 있지만 기존의 멀티코어에서 멀티스레드 가속하는 게 불가능했던 것이 가장 큰 이유이다.

다렉 12 더럽게 욕 많이 먹는다. api가 너무 로우 레벨이고 신경 쓸게 너무 많다. 처음부터 이걸 배웠다면 그냥 원래 어렵나 보다 할 텐데, 9에서 12로 넘어가는 사람들은 정말 어려워졌다고 느낀다. MS가 왜 이렇게 어렵게 만들었을까? 9 10 11은 조금씩 바뀌었지 이렇게까지 많이 바뀌지 않았었다. 파라미터 바꾸고 쉐이더 새로 바꾸고 그 정도였는데, 12는 근본적으로 달라졌다. 오픈지엘에서 벌칸으로 가는 것과 똑같다. 이게 벌칸인가 다렉 12인가 그런 얘기까지 나온다.

왜 다렉 12를 만들었나? 멀티코어 CPU에서 동시에 만들기 위해서. 멀티스레드 커맨드 버퍼 레코딩을 사용해서 성능 향상을 하려고. **왜 이렇게 함수를 잘게 쪼개 놓았냐?** 깔끔하게 함수 하나로 자료구조 하나로 하지 왜 이렇게 심하게 쪼개 놓았냐? **멀티스레드로 나눠서 쓰라는 이유다.** 그래서 이제 3D 클라이언트도 멀티스레드를 피해 갈 수 없다.

5년 전까지만 해도 플스 4도 없었고 멀티코어 CPU 쓰는데도 언리얼 쓰면 대충 다 됐다. 근데 지금은 해결되지 않고 멀티스레드 프로그래밍을 좀 복잡하게 해야 한다.

- **멀티스레드 프로그래밍의 종류**
  - Heterogeneous 멀티스레딩
    - 스레드마다 맡은 역할이 다르다
    - 다른 code part를 실행
    - 스레드 간의 load balancing이 힘들다
    - 병렬성이 제한된다.
  - Homogeneous 멀티스레딩
    - Data/Event Driven 프로그래밍
    - 모든 스레드는 Symmetric 하다.
      - avaliable 한 순서대로 input을 처리한다
    - 자동적인 load balancing, 제한 없는 병렬성
    - 작업분배 Queue를 비롯한 일반적인 병렬 자료구조

헤테로지니어스(Heterogeneous)는 스레드마다 역할이 다르다. 물리엔진 데이터 로딩 스레드마다 다 다른 역할을 하고 커뮤니케이션하는 것을 이거라고 한다. 스레드마다 역할이 다르다. 스레드마다 시작 함수가 다 다르다.
문제는 스레드 간의 로드밸런싱이 힘들다. 다 역할이 다르기 때문에 누군 할 일이 많고 누군 적다. 누구는 놀고, 누구는 바빠서 보틀넥이 일어난다. 그런데, 내가 할 일을 옆에 넘겨주지 못한다. 게임 스레드에서 랜더링 스레드로 넘겨주지 못한다. 그리고 이런 식으로 했다는 건 게임 로직 자체는 싱글 스레드로 돌아간다는 것이다. **커뮤니케이션은 멀티스레드 고려하고 lock을 거는데, 정작 게임 로직 안에서는 싱글스레드로 돌아간다. 그러니까 로드밸런싱이 안되고, 병렬성이 제한이 된다.** 랜더링 스레드가 보틀넥이네? 그럼 2개로 나누어서 2배 성능을 내자! 이게 안된다는 것이다. 2개로 자르는 순간 data race가 막 생겨버린다. 그걸 수습하려고 lock을 걸고 한다고 성능이 올라가지 않는다.

왜 이 방법을 쓰는가? 싱글 스레드를 멀티스레드로 바꾼다고 하면 이 방법이 편하기 때문이다. 게임 로직, 랜더링 로직 놔두고, 서로 주고받는 그 커뮤니케이션만 신경 써서 멀티스레드로 만들면 되기 때문에. 변환이 쉬워서 초창기에 나온 게임엔진들은 헤테로지니어스 멀티스레드 프로그래밍 방식을 많이 쓴다. 에픽의 언리얼 4의 문제가 바로 이거다. 쿼드코어까지는 성능이 올라가는데, 그 이상은 성능이 안 올라간다. 콘솔게임에서 그래서 힘들다. 옥타코어인 콘솔게임에서는 안된다. 언리얼 5 정도는 되어야 콘솔에서 제대로 성능이 나올 것이다.

그러면 이 한계를 어떻게 극복할 것인가? 호모 지니어스(Homogeneous)로 해야 한다. 스레드마다 맡은 역할이 같은 경우가 이것이다. 스레드마다 똑같은 일을 한다. 계산해야 할 작업이 생겼다, 그럼 아무 스레드나 갖다가 실행한다. 처리할 이벤트가 생겼다, 그럼 놀고 있는 아무 스레드나 가져다 실행한다. 헤테로지니어스는 마우스 버튼을 누가 눌렀는지 게임 스레드에서 처리해야 한다. 사운드 스레드에서 하지 못한다. 그런데 호모 지니어스에서는 가능하다. 실행하는 코드가 똑같고 어떤 작업 어떤 이벤트가 발생하더라도 아무 스레드나 다 처리 가능하다. 가능한 순서대로 input을 처리한다. 작업을 묻지도 따지지도 않고 실행한다. 어떤 스레드가 놀고 있나 보고, 무조건 작업을 맡긴다. 헤테로지니어스처럼 '게임 스레드는 날 처리를 못하고 랜더링 스레드는 바쁘네? 기다려야지' 이런 게 없다. 로드밸런싱이 문제가 없다. 스레드가 할 일이 없어서 논다? 그럴 수 없다. 다 일을 던져주니까**. 제한 없는 병렬성, 코드코어에서는 4배 핵사 코어에선 16배 성능 향상을 얻을 수 있다. 그래서 게임 서버는 호모 지니어스 방식으로 한다.**

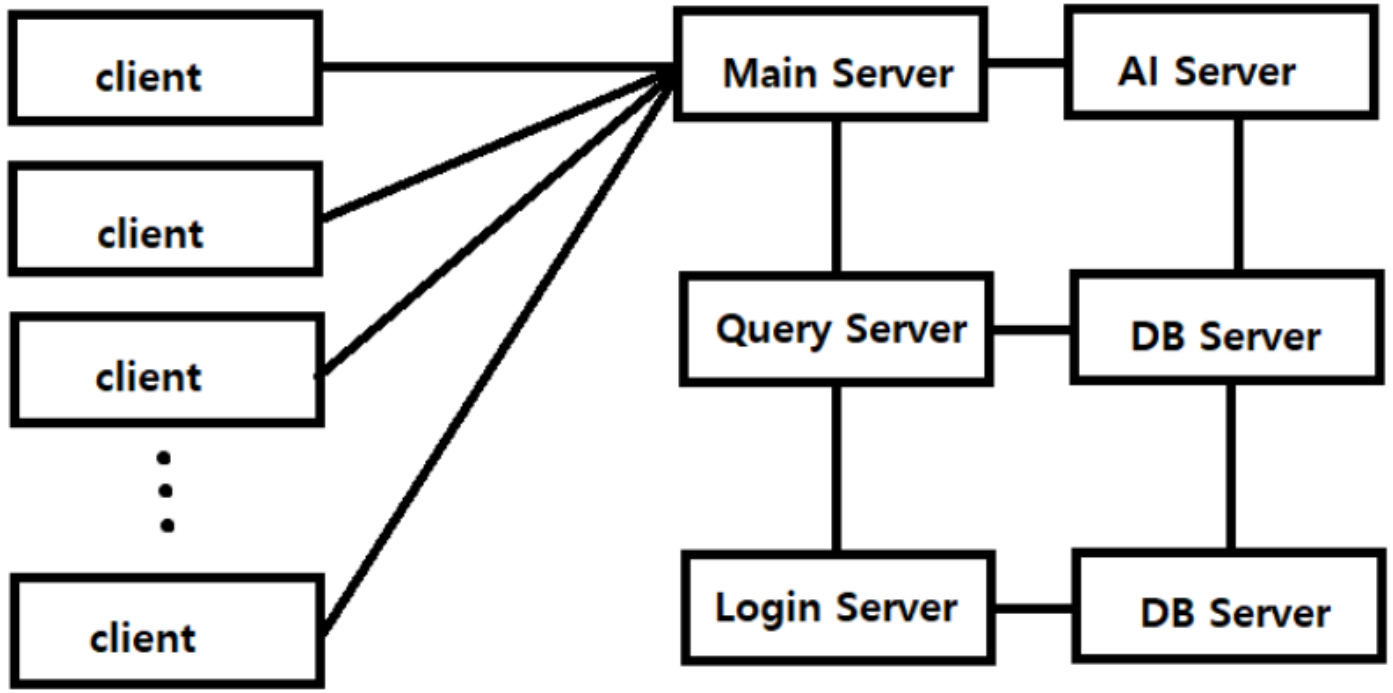
게임엔진이 왜 /헤테로지니어스 방식이냐? 옛날에 만든 방식이기 때문이다. 사실 언리얼 4는 언리얼 3의 껍데기만 바꾼 것이다. 옛날 코드를 들어내지 못하고 둔 것. 에픽이 새로 짜지 않으면 미래가 없다. 다른 게임회사에서 호모 지니어스 방식을 만들어 성능이 나온다면 언리얼도 망하게 될 것이다.

IOCP로 하다 보면 어쩔 수 없이 이렇게 할 수밖에 없다. 제한 없는 병렬성, 다른 회사에선 동접 500 600 1000에서 허덕일 때 리니지는 동접 15000까지 갔다. 코어가 더 많은 2배 비싼 컴퓨터를 사면 그냥 동접이 올라간다. 헤테로지니어스로 짠 회사에서 호모 지니어스로 되어있는 회사를 성능으로 당할 수가 없다. 쿼드코어 옥타코어 성능향상이 없는데, 호모지니어스는 2배 뻥튀기 하니까 당할 수 없다.

처음부터 짜면 무조건 호모지니어스로 짜야한다. 우리도 그 방식을 배울 거고 뭐가 필요한가? 데이터나 이벤트를 모두 모아 두고 여러 스레드에게 떠넘기는 자료구조가 필요하다. 작업분배 큐를 비롯한 여러 자료구조가 필요하다.

- **MMORPG Game Server**
  - Massively Multiplayer Online Role Playing Game
  - 한 개의 서버 프로세스에 5000명 이상의 사용자가 동시 접속해서 게임 실행
  - 게임 서버는 가상 환경을 시뮬레이션하는 네트워크 이벤트 시뮬레이터
    - 5000명 플레이어 (소켓)
    - 수십만 몬스터 (AI)
    - 20km X 20km 월드 (충돌 체크, 길 찾기)

mmorpg에서 사용. 그냥 이동, 그러면 이동하고 끝나는 게 아니다. 이동했다고 하면 주위의 npc가 반갑다고 하면서 ai가 돌아가야 하고 파이어볼 날리고 칼질에 거리게 산도 해야 하고, 가상 환경을 시뮬레이션을 해줘야 한다. 그래서 cpu 오버헤드가 매우 크다. 스크립트 엔진 cpu가 남아나질 않는다. 그래서 멀티코어 프로그래밍을 해야 한다. 5천명의 플레이어? 5천 개의 소켓 날라오고, 몬스터 수십마리 ai로 돌아가야한다. 월드 충돌체크, 길찾기 해야한다. 플레이어가 알아서 길찾는거 아니다. 몬스터 뚫고 갈 수 없으니 길찾기 해야한다. 그래서 멀티스레드 프로그래밍 하지 않으면 버틸수가 없다.



5천개 이상의 클라이언트가 메인서버에 접속해있고, 날아오는 패킷을 받아서 해야 하고. 메인에서 모든 걸 해야 하나? 아니다. 데이터베이스, 쿼리, ai, 분할해서 작업을 덜어내긴 하는데 그래도 메인서버가 보틀넥이다. 그래서 얘가 성능이 좋아야 하고, 멀티스레드로 처리해야 한다.

멀티스레드 하고 네트워크 api 결합해서 사용한다. 윈도우에서는 iocp, 리눅스에서는 epoll. 근데 리눅스에서 사용할 땐 멀티스레드와 관련이 없지만 엮어서 프로그래밍해야 한다.

- **MMORPG Game Server의 구현**
  - Windows에서 제공하는 멀티스레드 + Network I/O API인 IOCP 사용.
    - 일반적인 Select() 함수로는 몇천 개의 socket을 관리할 수 없음.
    - socket 하나당 하나의 스레드는 운영체제의 과부하 Linux나 IOS는 epoll 또는 kqueue를 사용.
  - Time Consuming 작업 및 Blocking 작업들의 재배치.
    - Throughtput과 함께 response time도 중요.

5천 개의 소켓을 만들어서 소켓 하나당 하나 하려고 했는데, 운영체제의 과부하로 버틸 수 없다. 그래서 이런 적절한 개수의 스래드로 다중접속을 컨트롤할 수 있는 api를 써야 하고, 그게 IOCP이다.

그러면서 모든 스레드가 모든 작업을 할 수 있어야 한다. 시간이 많이 걸리는 작업, 데이터베이스에서 데이터를 읽어와야 한다, 그럼 멈춰야 한다. 안 그럼 스레드 자체가 멈추니까 코어 하나가 놀고 성능이 떨어진다. 그러니까 그런 작업은 재배치하고, Throughtput을 신경 써야 한다.

- **MMORPG Game Server**
  - Homogeneous Multithreading
    - worker thread의 pool 사용
    - socket을 통해 packet이 올 때마다, OS가 thread pool의 thread를 하나 깨워서 packet의 처리를 맡김.
    - HW Core개수 1.5배의 worker threads
    - Network 데이터 처리뿐만 아니라 AI 루틴도 worker thread에서 처리
      - 모든 time consuming 작업은 병렬로 처리

워커 스레드는 모든 스레드. 무슨 일이든 다 하는 것이다. 소켓을 통해 패킷을 이룰 때마다 스레드 처리. 끝나면 다시 큐에 가서 자고, 패킷 날아오면 깨워서 일 처리하고. 스레드 풀에 스레드가 모자라다? 패킷이 날아오는데 놀고 있는 스레드가 없어서 렉 걸린다면 컴퓨터를 업그레이드해야 한다.

10 코어 cpu면 20 코어 cpu로 바꾸어야 한다. 가격은 4배 정도 차이나지만. 리니지 M 1년에 1조 번다. 그냥 새로 사도 된다.

워커 스레드는 5천만 개 만드는 게 아니라, 하드웨어 코어의 개수의 1.5배 정도 만든다. 16 코어다 그럼 스레드 16개만 돌리면 되는데 이게 게임 서버는 네트워크 I/O를 해야 한다. send/recv 해야 하지만 딜레이가 있다. 그래서 1.5배 정도로 하는 게 밴치마크하다 보니 적절하다. 옥타코어 12개. 이렇게 만들어서 돌린다. 프로그래밍을 잘할수록 줄어든다. 1.0배 정도가 제일 좋은데 그 정도는 아직 안된다.

클라 패킷 날아오면 워커 스레드 깨워서 처리한다. 근데 ai 돌려야 하는데 어떤 스레드에서 돌려야 하느냐? ai 스레드에? 아니다. 그럼 호모 지니어스 방식이 아니다. 다 워커 스레드가 처리할 수 있도록 프로그래밍해야 한다. 걔네들은 하는 일이 없고 모든 cpu 작업은 워커 스레드에서 해야 하고, 그래야지 제한 없는 성능 향상이 있다.

- **MMORPG Game Server**
- **멀티스레드 트릭 #1 : Database query**
    - 문제 : data base 쿼리는 스레드의 blocking을 초래
    - 해결 : blocking 전용 스레드를 따로 두어서, 작업 전달

블로킹 작업 디비 왔을 때, 스레드가 멈춰있으면 안 된다. 코어 하나가 노는 것이기 때문이다. **그래서 워커 스레드는 DB를 호출하지 않고, 이벤트 큐에 넣어놓고 실행한다.** 걔는 블로킹하는데 CPU를 쓰지 않는다. **그래서 워커 스레드가 돌아가는 것을 훼방 놓지 않는다.**

- **멀티스레드 트릭 #2 : NPC AI**
    - 문제 : NPC가 너무 많다.
    - 해결 : Time thread를 사용하여 Active 한 NPC의 Active한 event만 처리.

AI 돌아가는 거 1초마다 몬스터를 힐링해야 한다. 그럴 때 루프돌면서 모든 몬스터 힐링하고 그럼 오버헤드가 너무 크다. ai 스레드를 쓰면 헤테로 지니어스 방식이 된다. 힐링을 해야한다 하면, 힐링 이벤트를 타이머에 등록해놓고 타이머 스레드에서 시간 이벤트들을 꺼내서 Post evented worker thread에게 일을 맡겨야 한다. 이 타이머 스레드는 하는 일이 하나도 없다. 시간이 됐나 안됐나 보고 처리 이벤트가 있나 없나 보고. 이벤트 없으면 자고, 이벤트 있으면 떠넘기고.

호모 지니어스 스레드는 절대 멈추면 안되고 호모지니어스 말고 다른 곳에서 cpu를 쓰면 안 된다. 로드밸런싱 문제가 생기고 확장성에 문제가 생기기 때문이다.

- **MMORPG Game Server**
- **Worst case Tuning**
    - 평상시의 cpu낭비 OK, 최대 부하일 때 잘 버티는 프로그램이 최고
- **컨테이너 자료구조를 사용한 객체 간 동기화**
    - Queue, Priority Queue, Set
    - Custom 자료구조
        - 1 write/multi reader Queue
        - multi writer/1 reader Priority Queue
        - Set with copy method
    - Thread & Cache affinity

제대로 된 프로그램이다 한다면, 할 일이 적으면 cpu도 놀아야 한다. 중요한 건 최대 동접이다. 동접 10000을 버티냐 안 버티냐가 문제다. 1000일 때 cpu 사용량이 10% 줄여서 전기를 아낀다? 그런 거 없다. 동접 1000으로 했는데, cpu 80% 쓴다? 낭비 아니냐? 아니다. 1만만 버티면 됐지 cpu가 낭비되는 비효율적인 프로그램이 나오기도 한다. 최대 동접만 버틸 수 있다면.

헤테로 지니어스처럼 쉽게 할 수 없다. 언리얼 엔진은 싱글로 돌고, 데이터 넘길 때만 멀티스레드인데 호모 지니어스는 모든 게 멀티스레드로 돌아가야 한다. 그래서 컨테이너를 많이 쓴다. 다 멀티스레드 자료구조를 써야 한다. 새로 만들어서 써야 한다 자료구조를.

멀 테스 레드에서 잘 돌아가게 만든다? 그럴 수 있는데 그럼 오버헤드가 생겨서, 제한조건을 걸어서 쓸 수밖에 없다. queue인데 멀티스레드에서 읽고 쓰고 할 수 있는데, enqueue를 하는 건 하나의 스레드에서 하고 dequeue는 여러 스레드에서 하는 그런 큐. 그런 큐를 사용한다 해야 되면 multiple reader, multiple writer보다 효율적인 자료구조를 만들 수 있다. 그래서 동접을 올리기 위해서는 이런 제한조건을 잘 계산해서, 성능을 높이고 해야 한다. 또 내가 만든 set은 insert count erase 3개는 되는데, copy도 되어야 한다? 그럼 또 새로 만들면 된다. 기존 걸로 안되면 수정하고, 새로 만들고 할 수 있다. 근데 그게 어렵다. 한 학기 동안 해볼 것이다.

그리고 멀티스레드 프로그래밍을 하다보면 이런 문제가 생긴다. 스레드 돌리는데 우리가 사용하는 운영체제가 시분할 운체이다. **프로세스가 여러 개였으면 한 프로세스는 한코에어서만 실행되지 않는다.** 콘텍스트 스위치 돼서 1번 코어에서 쓰던거 옆 코어에서 옮겨서 실행하고 할 수 있다. **근데 그게 성능에 안좋다.** 하나가 00 코어에서 실행하면, 옮기는 것보다 계속 가는 게 좋다. 왜? 0번 코어에 캐시, 그 프로세스가 사용하는 데이터가 올라와있는데, 캐시 적중률이 높은데 옆 코어로 올라가면 캐시 미스 쫙 뜨면서 다시 캐시 로드해야한다. **자기가 하던 일 같은코어에서계속하는 게 좋다.** 스레드마다 어떤 코어에서 해라 그게 어피니티(Affinity)라고 한다. 이건 나중에 가면 해주어야 한다. 리눅스 윈도우 코어스케줄링이 아직까지 썩 좋지 않다. 100% 만족이 안된다. 우리가 해주어야 하는데 수업시간엔 다루지 않는다. 굉장히 복잡한 문제고 대학원에서나 하는거지 0번 스레드 0번코어0번 코어, 1번 스레드 1번코어 이렇게 하면 성능이 안나온다안 나온다. 신경쓸게 굉장히 많다. 인터립트처리, 쭉 해서 신경쓸게 많아 주의깊게 해야한다. 초보자들이 건드릴게 아니다. 웬만한 게임회사에서도 이거 건드리는 회사가 없다. 엔씨에서만 건드리고 있다. 다른회사에서는건드리지 못하고, 건들여봤자 성능이 더 안나온다.