1. 개발 환경

- 멀티스레드 프로그래밍에서 운영체제의 역할은 거의 없다.
- 모든 것은 CPU에 달려있다.

게임회사들이 점점 더 리눅스를 많이 쓰고 있다.

C로 프로그래밍하는 것? 리눅스로 하는 것과 완전히 똑같다.
게임 서버 쪽은 예전엔 윈도우로 개발을 했었으나, 최근에야 굳이 그럴 필요가 없고 해외로 수출하기 위해 리눅스를 많이 쓴다. 윈도우 운영체제로 개발하려면 돈이 많이 들기 때문이다.

그렇다면 운영체제 별 성능은? 똑같다.
멀티스레드에서 운영체제의 역할은 없다. CPU가 알아서 하는 것.
똑같은 프로그램을 윈도우, 리눅스, OSX, 어디서 돌리나 성능의 차이가 없다.
~~(거의 없다나 없다나 같은 건가? PPT에는 차이가 거의 없다고, 설명에는 없다고 하심)~~

2. 컴퓨터

- 직렬 컴퓨터

-하나의 CPU(또는 core)만을 갖는 컴퓨터.
-듀얼코어가 대중화 되기 이전의 대부분의 컴퓨터.
-학생들이 배우는 자료구조/알고리즘은 전부 직렬 컴퓨터를 가정함.
-현재 멸종

- 병렬 컴퓨터

-여러개의 CPU(또는 core)가 동시에 명령들을 실행하는 컴퓨터.
-직렬 컴퓨터의 속도 제한을 극복하기 위해 제작.
-우리가 작성하는 프로그래밍 실행되는 컴퓨터.

컴퓨터는 원래 모두 직렬 컴퓨터였다. 사과라고 하지 과일 사과라고 하지 않는 것처럼, 컴퓨터도 처음에 직렬 컴퓨터라고 구분하지 않았다.

그러나 현재는 멸종되고 병렬 컴퓨터만 남게 되었다. 문제는 자료구조/알고리즘 프로그래밍 방법이 다 직렬 컴퓨터용 알고리즘이라는 점이다. 멀티코어 스레드 알고리즘이 아니라, 싱글코어 스레드에서 돌아가는 알고리즘. 그래서 싱글코어에서 멀티코어를 한다고 성능이 좋아지지 않는다.

병렬 컴퓨터가 나오게 된 배경? 싱글코어 컴퓨터가 너무 느려서 나오게 되었다.
얼마나 느렸냐? 컴퓨터를 개발한 폰노이만 아저씨가 컴퓨터와 2의 거듭제곱 중에 12번째 자리가 7이 나온 수가 무엇인지 계산을 했는데 컴퓨터가 졌다.
해결 방법은? 코어 속도를 높이거나 여러 개의 코어를 쓰거나. 여러개의 코어를 쓰는 것이 싸기 때문에 이렇게 했다. 우리가 짠 모든 프로그램이 병렬컴퓨터에서 실행된다. 싱글 컴퓨터에서는 돌아갈 리 없음. 이미 멸종되어서 없으니까.

3. 병렬 컴퓨터

- 병렬 컴퓨터의 사용 목적
  - 여러개의여러 개의 작업을 보다 빨리 실행하기 위해서 (그러나, 그냥 여러 대의 컴퓨터를 사용해도 됨)
  - **하나의 작업을 보다 빨리 사용하기 위해서**
- 병렬 프로그램
  - 기존의 프로그램을 병렬 컴퓨터에서 실행했을 경우 속도 증가는 0%
  - 동시에 여러개의 명령흐름이 실행되는 것을 가정하고 다시 프로그래밍 해야 한다. → 병렬 프로그래밍

위에서 말한 **여러개의 작업을 빨리하기 위해 병렬 컴퓨터를 사용했다는 말은 사실 정확한 말이 아니다.**
컴퓨터를 하나 더 산 다음 일을 시키면 되는 것을 굳이 병렬 컴퓨터가 필요 없기 때문이다. 병렬은 컴퓨터 한 대 안에 계산하는 CPU가 여러 개 있는 것을 의미한다.
싱글코어 컴퓨터를 2개 나란히 놓았다고 병렬이 되는 것이 아니다.

그래서 병렬 컴퓨터가 무엇이냐? **하나의 작업을 보다 빨리 실행시키기 위한 것이다.**
게임 로스트아크를 한다고 치자. 나는 멀티를 하기 위해 2개의 클라이언트를 돌려야겠다. 한 컴퓨터에 2개를 돌리니까 느려질 것이다. 그럼 컴퓨터 2대를 사서 1개씩 돌리면 빨라진다. 그러나, 빨라진 것이 아니라 한 컴퓨터에서 돌려 속도가 떨어졌던 것이다.

병렬 컴퓨터의 목적은 프로그램 하나를 빨리 돌리기 위해 사용하는 것.
클라이언트 2개, 10개를 동시에 돌리는데 느려지지 않게 하는 것이 아니다.
1개를 돌렸을 때 그 속도를 높이기 위해 쓰는 것, 성능을 업그레이드하기 위해 쓰는 것이다.

우리가 작성해왔던 프로그램을 싱글코어, 쿼드코어, 핵사코어 어디서 돌리든 활용할 수 없는 구조이기 때문에 속도의 증가가 없다. 2배 4배 속도를 업그레이드시키고 싶다면 다시 프로그래밍해야 한다.
작업을 CPU에게 하나씩 나누어 주는 작업이 필요하다. 작업분량이 1000개가 있으면 250개씩 코어 하나씩에 할당해주어야 한다.

4. 병렬 프로그램

- 병렬 프로그램의 특징
  - 실행된 프로세스의 내부 여러곳이 동시에 실행됨.
  - 병렬로 실행되는 객체(Context로 불림) 사이의 협업(동기화 또는 synchronization)이 필수.
  - 크게 공유메모리(Shared Memory) 모델과 메시지 패싱(Message Passing) 모델이 있다.
  - 우리가 다루는 것은 공유 메모리 모델

우리가 여태까지 작성해온 프로그램은 실행되면 메인부터 쭉 차례차례 실행되었다. 루프가 나오면 돌고, 함수 호출이 나오면 함수에 갔다 리턴되고.

그런데 그 개념이 바뀐다. 프로그래밍 여러군데가 동시에 실행이 된다. 프로그램을 작성했을 때, 싱글은 위에 적은 순서대로 실행이 될 것이다. 그런데 병렬은 어떻게 되는가? 실행되다가 가지를 친다. 가지 하나하나가 프로그램 카운터이다. 카운터가 명령을 캐치해서 실행하고, 또 캐치해서 실행한다. 병렬 프로그램은 프로그램 카운터가 여러 개 있어서 동시에 실행된다.

이게 무슨이야기냐? 멀티코어 컴퓨터이기 때문에 코어가 4개면 프로그램 카운터가 4개이다. 그럼 4 군데가 동시에 실행될 수 있다는 것이다. 실행하는 프로세스의 흐름이 바뀌는 현상이 콘텍스트 체인지(context change)라고 한다. 하나하나가 다 따로따로 옮겨 다니고 동시에 실행이 되는 것.

또 특징 중 하나. 서로 가지를 쳐서 쭉 실행되다가 서로 다른 가지 끝에 있는 두개는 완전히 독립적이고 따로따로 동작할까? 그렇지 않다. **스레드끼리 데이터를 주고받고, 실행순서를 맞추고, 협동을 해서 실행한다. 이것을 동기화라고 한다.** 멀티스레드 프로그래밍을 할 때 어려운 점이 바로 이것이다. 서로 다른 두 개의 콘텍스트를 맞춰서 잘 작성해야 한다는 것.

병렬 컴퓨터는 크게 두종류로 나눈다. 공유 메모리와 메시지 패싱. 이 두 개는 완전히 다른 프로그래밍 방법이다. 공유 메모리와 메시지 패싱은 시작점부터 다른 이야기. 따로 배워야 하는데 여기서는 공유 메모리만 다룰 것이다. 왜냐하면, 멀티스레드 프로그래밍은 공유 메모리 모델이기 때문이다. 메시지 패싱은 분산처리 프로그래밍 모델이 이것을 사용함. 게임에서도 메시지 패싱을 쓰기 위한 시도가 계속 있었지만, 실패함. 달빛조각사가 현재 이 모델로 제작 중에 있고 듀랑고가 이 모델로 제작되었다. 마비노기 영웅전도 메시지 패싱을 시도했다가 공유 메모리 방식으로 바꾸었다.

5. 병렬 프로그램 요구사항

- 정확성
  - 여러 흐름(Context)에서 동시다발적으로 호출해도 문제 없이 실행되는 알고리즘이 필요.
  - 오류가 발생하면 모든 것이 의미 없음.
- 성능
  - Context 증가에 따른 성능 향상이 높아야 한다.
  - 기존 직렬 프로그램보다 느려질 수 있다.

병렬 프로그램에서는 정확성과 성능을 고려해야 한다.

정확성? 프로그램에는 오류가 있으면 안된다. 하지만 적어놓은 이유는 정확성을 지키는 것이 어렵기 때문이다.
성능? 왜 멀티스레드를 써야하는가. 성능 때문에 써야 한다.
CPU가 느리기 때문에 멀티스레드 프로그래밍을 해야한다. CPU가 느린 것과 프로그램하고 무슨 상관이냐? 게임은 성능이 중요하기 때문에 상관이 있다. 게임이 느리거나, 프로그램이 느리다면 어떡해야 하는가? 멀티스레드 프로그래밍을 한다? 아니다. **멀티스레드 프로그래밍은 최후의 마지막 수단이다.**

성능 개선을 먼저 해야한다. 자료구조에서 배운 알고리즘을 봐서 성능개선을 하고, 원하는 속도가 나오면 거기서 멈추고 다른 것을 하면 된다. 그런데, 그래도 안 됐을 때 멀티스레드를 시도할지 고민해야 한다. 이때 무엇을 보아야 하는가?

느린 원인이 I/O 때문에 느릴 수 있다. 특히 게임 서버는 네트워크 I/O를 하고, 3D 그래픽은 GPU I/O를 한다. 이렇게 I/O 때문에 성능이 안 나온다? 그렇다면 멀티스레드가 아닌 I/O최적화를 해야 한다.
하드디스크 액세스 때문에 성능이 안 나온다? 그렇다면 하드디스크 접근 횟수를 줄여야 한다. 그리고 하드디스크로 주고받는 데이터의 사이즈를 줄여야 한다.
네트워크 때문에 느리다? 그럼 I/O 최적화를 해야 한다. 여기서 많이 쓰는 기법이 비동기 I/O이다.

느린 원인이 CPU다? 그러면 이제 멀티스레드 프로그래밍을 해야 한다.

느린 원인이 CPU다? 그러면 이제 멀티스레드 프로그래밍을 해야 한다.

그런데, 느린 원인이 I/O인지, CPU인지 어떻게 아는가?
작업 관리자를 켰을 때 세부 정보에서 실행파일이 CPU를 몇 % 나 활용하나 보자. 100% 활용하고 있다? 그럼 CPU가 문제이다. CPU는 놀고 있는데 컴퓨터가 느리다면, 프로그램에 버그가 있거나 I/O에 문제가 있는 것이다.

100%라는 것은 i7이면 4개의 코어를 다 쓰고 있다는 것인데 이게 아니다. 싱글코어 프로그램이 4개의 코어를 다 쓸 리가 없다. 아무리 많이 써도 25%. CPU 사용량이 25%라면 그것은 100% 쓰고 있다는 것이다.

정확성과 성능. 이 둘 중 하나라도 안되면 멀티스레드 프로그래밍을 하는 이유가 하나도 없다.

멀티스레드를 잘못하면 더 느려지는데 그 이유는 뒤에 나오게 된다.

6. 멀티스레드 프로그래밍

- 멀티스레드 프로그래밍
  - 병렬 프로그래밍의 유일한 구현 수단.
  - 하나의 프로세스 안에서 여러 개의 스레드를 실행시켜 병렬성을 얻는 프로그래밍 방법
  - Windows, Linux, Android, iOS에서 기본으로 제공하는 유일한 병렬 프로그래밍 API
    - HW와 운영체제가 직접 지원하는 것인 이것뿐.
    - 다른 API로는 GPGPU가 있음.

병렬 프로그래밍의 유일한 구현 수단이다. 프로그램 실행 속도를 2배 4배 높이는 방법은 이것뿐이다. 멀티스레드 프로그래밍을 하지 않았는데, 게임엔진 언리얼 4를 사용했는데 만약 듀얼코어보다 쿼드코어가 조금 더 빠르다면? 왜? 언리얼 엔진이 내부적으로 멀티스레드 프로그래밍을 구현해 두었다.

구현하지 않고 openMP, 이런 걸 써서 했더니 빨라졌다? 역시 내부적으로 멀티스레드로 구현이 되어있음.
유일한 예외는 쿠다(CUDA) 그래픽카드 가속이 있다.
멀티코어가 100% 활용되었다면 내부적으로 멀티스레드가 사용된 것.

멀티스레드 외에 다른 방법으로 병렬로 짜겠다면 운영체제에 CPU를 뺏어서 써야 하는데 그럴 수는 없다. 유일한 예외는 플스 3가 운영체제와 상관없이 CPU를 사용했지만 현재는 망했다.

그런 현재 멀티코어를 활용하는 유일한 방법은 멀티스레드. 최근 운영체제는 다 멀티스레드 프로그래밍을 제공한다. 그리고 유일하고.
CPU를 처음 만들 때부터 멀티스레드를 고려해서 만들어졌다. CPU를 공부해보면 알겠지만 멀티스레드와 관련한 CPU 명령어가 있다. 기계어로 스레드를 만들고 마이크로 코드로 코딩되어 있다.

7. 프로세스

- 운영체제는 사용자의 프로그램을 프로세스 단위로 관리한다.
- 실행파일의 실행 → 운영체제가 파일 내용을 메모리에 복사 후 시작 주소로 점프하는 것.
    - 읽어 들일 때 여러 가지 초기화가 필요하다
- 시분할 운영체제는 여러 프로세스를 고속으로 번갈아가면서 실행한다.
    - 실행 중인 프로세스의 상태를 강제로 준비로 변경 가능.

스레드를 알려면 프로세스를 알아야 한다.
프로세스란? 운영체제가 관리하는 프로그램의 단위.

실행파일을 실행한다? 프로세스를 만든다랑 같은 의미이다. 내부적으로 실행파일의 실행은 운영체제가 실행파일을 읽어서 메모리에 복사해두고, 시작 주소로 점프하는 것. 여러 초기화가 필요하다. 멀티코어가 아니더라도 여러 개의 프로그램이 동시에 실행됨.

싱글코어에서 프로그램 여러 개가 동시에 실행되는 것을 볼 수 있다. 컴퓨터가 쿼드코어다? 프로세서 4개밖에 실행 안될 것 같은데 사실 프로세서 수십 개가 나온다. 그게 왜 가능할까? 왜냐하면 **시분할 운영체제이기 때문**이다. 코어가 1개여도 **번갈아가면서 실행**시킬 수 있기 때문. 그렇기 때문에 코어가 하나라도, 코어가 4개라도 수십 개의 프로세스가 동시에 실행이 된다.

- 프로세스의 메모리 구조는 다음과 같다.
    - Code : 실행될 명령어가 들어가는 구역
    - Data : 전역 변수가 들어가는 구역
    - Stack : 지역변수와 함수 리턴 주소가 들어가는 구역
    - Heap : malloc이나 new로 할당받은 메모리가 들어가는 구역
    - PCB : Process Control Block

실행파일을 읽고 닥치고 실행하는 것이 아니라, 프로세스는 내부적으로 성격에 따라 여러 개의 구역으로 나누어서 관리를 한다. 이것을 세그먼트라고 한다.

기계어가 들어가는 부분이 코드, 데이터는 전역 변수. 스택은 지역변수와 함수 리턴 주소, 힙은 할당받은 메모리가 들어간다.