게임 OS 및 DB 멀티쓰레드와 게임아키텍처

이준
junlee@hoseo.edu

목차

1

멀티 쓰레드 프로그래밍

필요성

- 한 프로세스에서 처리해야할 여러 로직들을 동시 다발적으로 처리하는 것이 필요
  - Ex : 게임에서 렌더링, AI, 시뮬레이션, 네트워크 등 다양한 부분들을 처리해야 함
- 컴퓨팅 자원의 발전
  - 서버 뿐만 아니라 클라이언트에서도 멀티 코어를 제공하고 있음

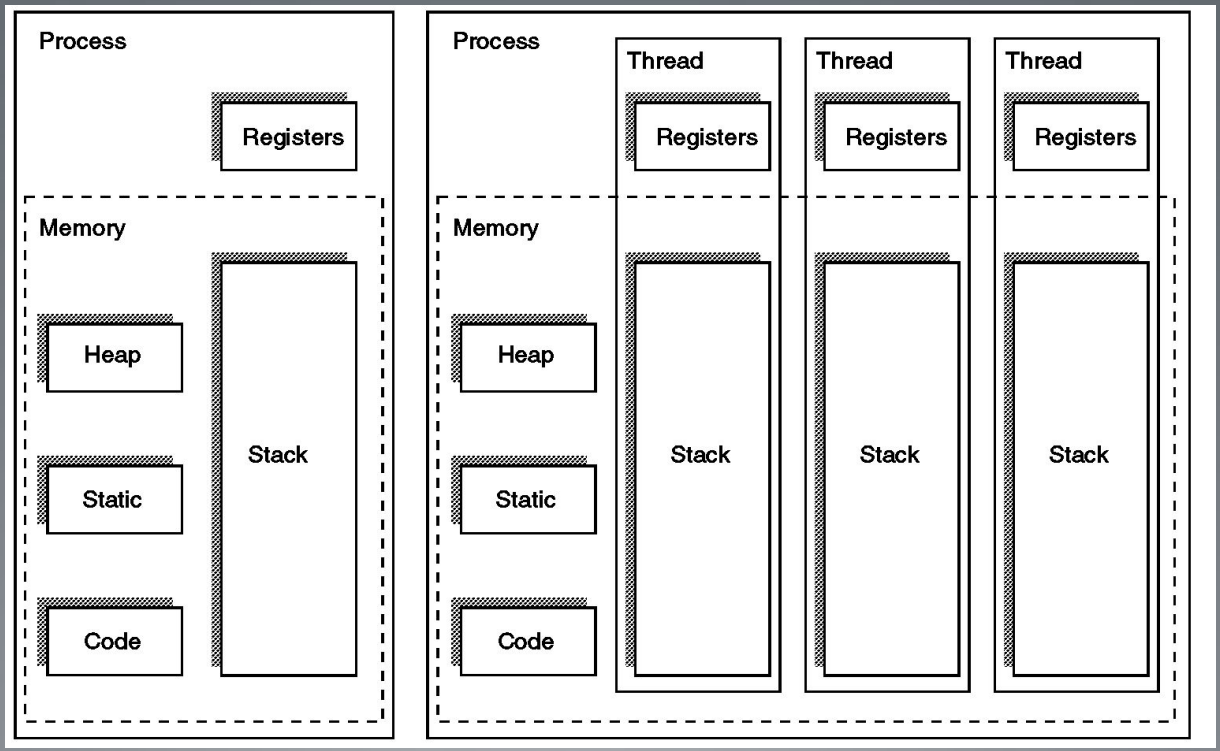
- 멀티 프로세스 프로그래밍 보다 멀티 쓰레드 프로그램이 더 관리하기 쉬움
  - 멀티 프로세스 프로그래밍 : IPC(Inter Process Communication : pipe, socket)
  - 멀티 쓰레드 프로그래밍 : 프로세스내의 전역 변수 혹은 클래스 멤버 변수를 통해 접근 가능

- 멀티 쓰레드 프로그래맹의 단점
  - 생성된 쓰레드들의 관리가 어려움
  - 동기화, 자원 관리 이슈들이 발생

1 멀티 쓰레드 프로그래밍

필요성

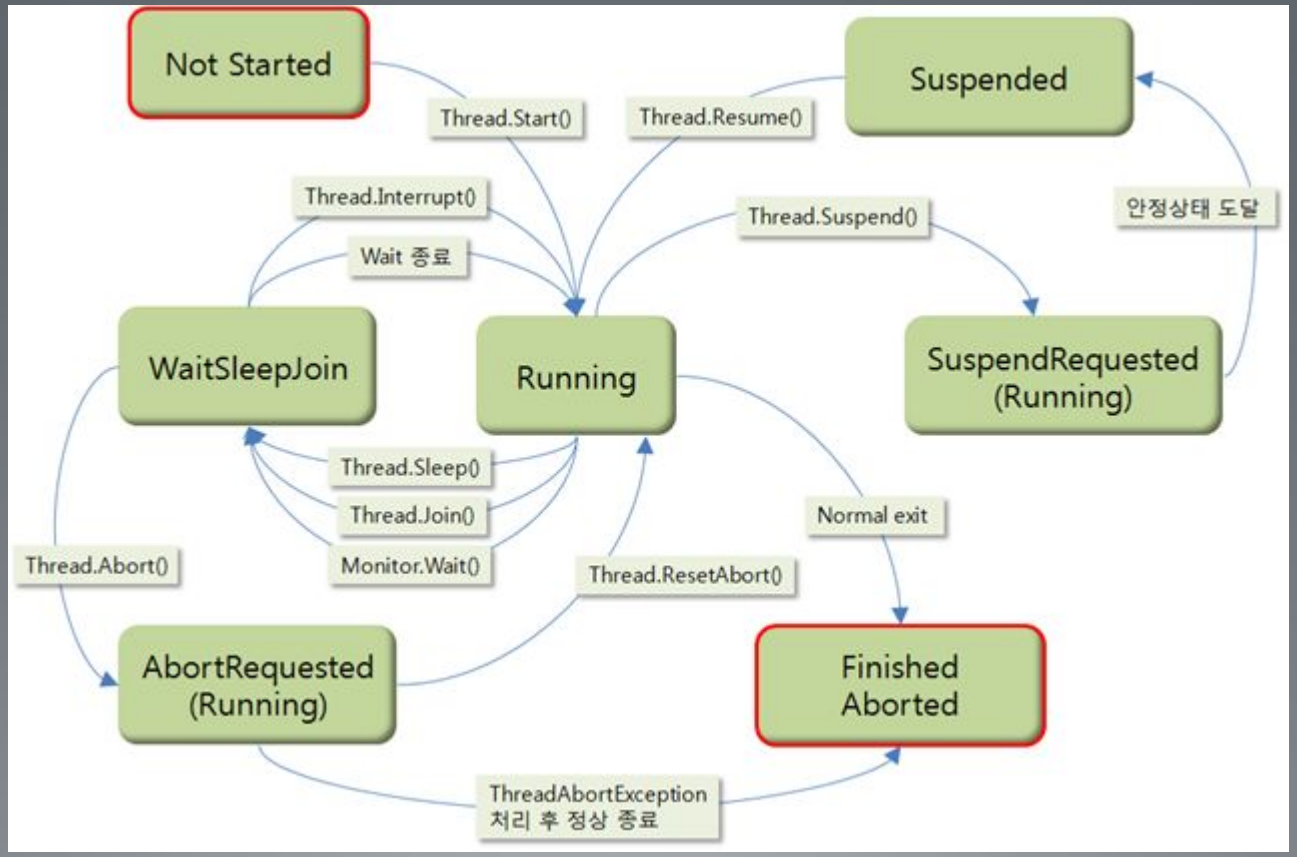
- 쓰레드는 프로세스처럼 스케줄링의 대상이다. 이 과정에서 컨텍스트 스위칭이 발생한다. 하지만 쓰레드는 공유하고 있는 메모리 영역 덕분에 컨텍스트 스위칭 때문에 발생하는 오버헤드(overhead)가 프로세스에 비해 작음



쓰레드의 생명 주기

쓰레드의 생명 주기

- 쓰레드의 생명 주기는 아주 민감 하므로 정말 잘 설계하고 써야 한다!



1 멀티 쓰레드 아키텍처 설계하기

어떻게 좋은 멀티쓰레드 아키텍처를 설계해야 할까?

- 좋은 멀티쓰레드 아키텍처
  - 주어진 환경에서 최적의 성능을 낼 수 있음
  - 크라이시스 2, 메탈기어 솔리드 2등등

1

멀티 쓰레드 아키텍처 설계하기

어떻게 좋은 멀티쓰레드 아키텍처를 설계해야 할까?

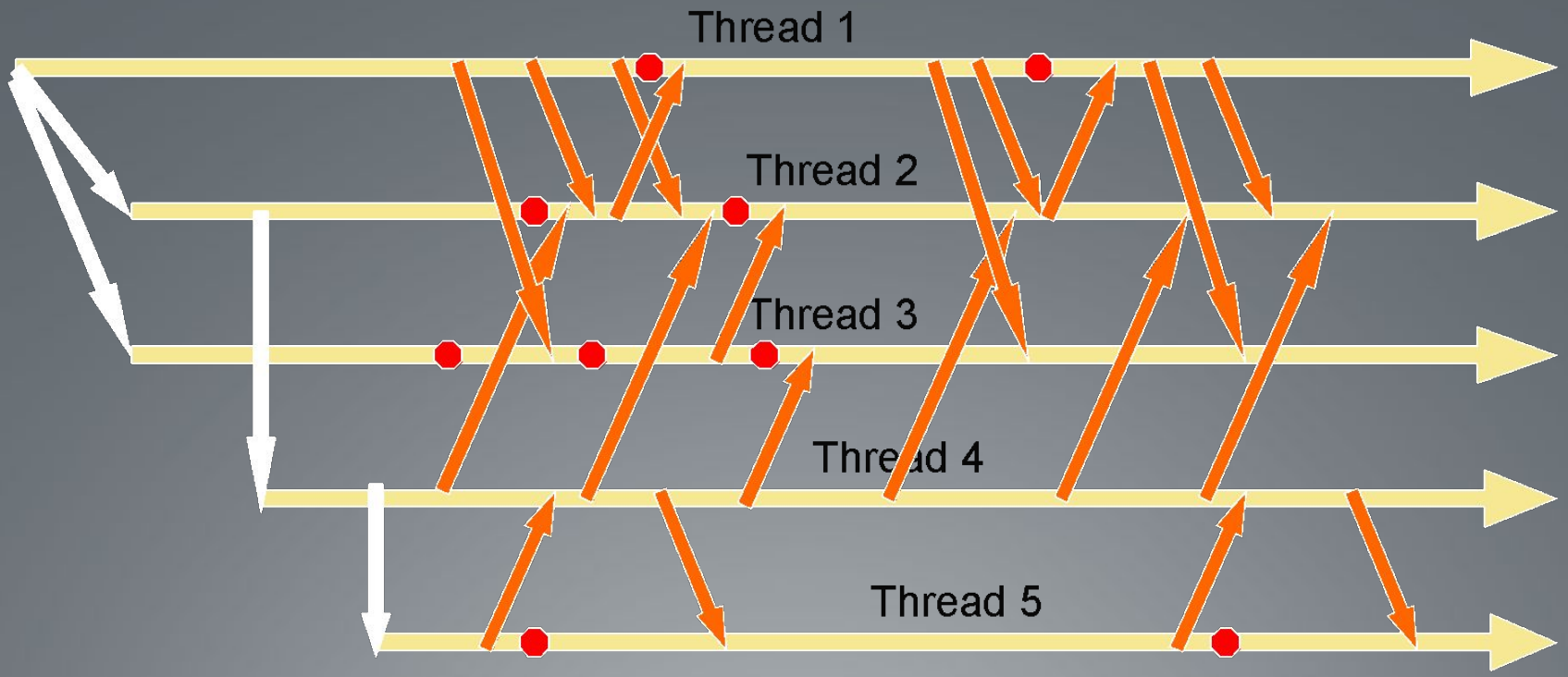
- 병목현상(Deadlock)
  - 쓰레드 간의 공동 자원을 처리하는 경우에, 자원에 대한 locking과 unlocking을 제대로 해주지 않는 다면, 쓰레드 간에 서로 locking을 걸어 동작이 중지되는 deadlock 상황이 발생하게 된다.
- 동기화버그(Synchronization Bugs)
  - 잘못된 시스템 구조는 쓰레드 간에 자원의 동기화를 제대로 해주지 못함으로 원하지 않은 결과를 얻을 수 있다.
- 낮은 퍼포먼스(Poor Performance)
  - 다중 쓰레드 간에 잦은 자원의 locking/unlocking 등을 처리하다 보면 오히려 단일 쓰레드를 사용하는 거에 비해서 매우 낮은 성능을 낼 수 있다.

1

멀티 쓰레드 아키텍처 설계하기

보통 우리는...

- 처음에 동작하고 있던 1, 2, 3등의 쓰레드가 필요에 의해서 여러 개의 쓰레드를 개별적으로 생성하고 생성된 쓰레드가 원래 쓰레드와 서로 자원을 공유하게 되는 경우 많은 부분에서 문제점이 발생



1 멀티 쓰레드 아키텍처 설계하기

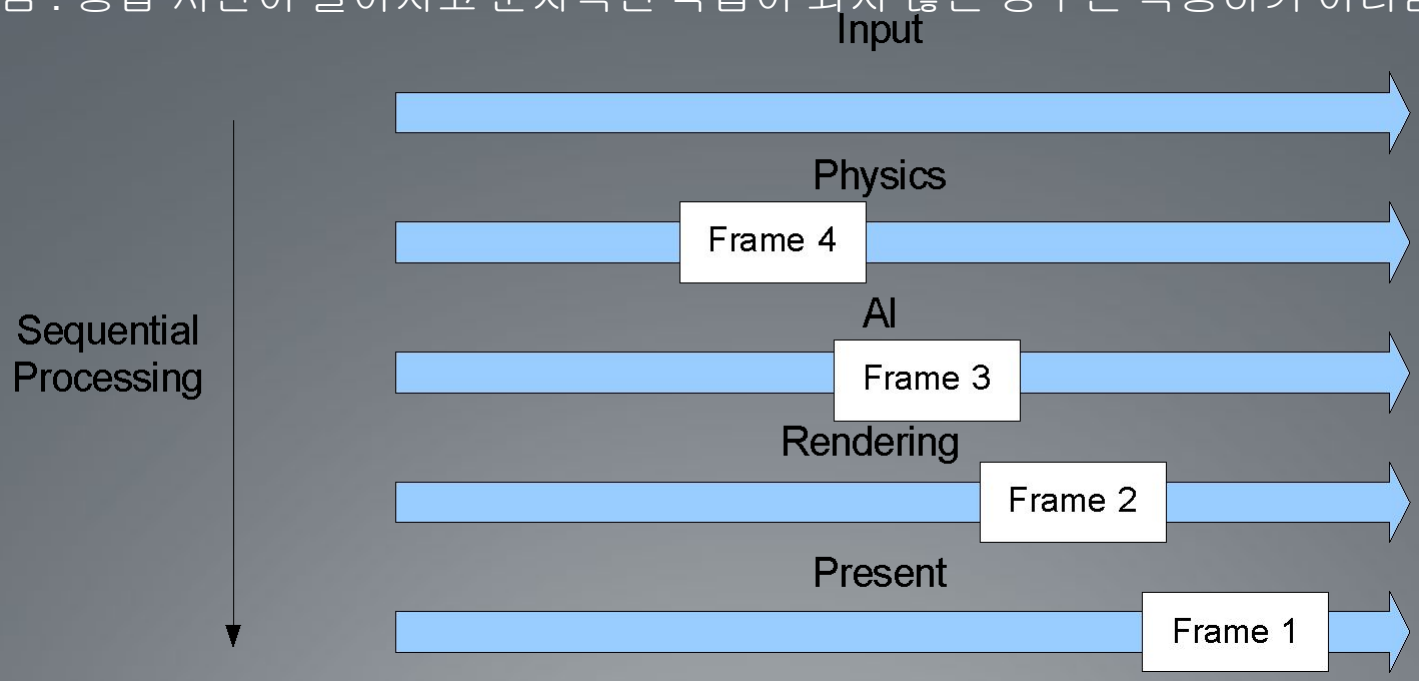
효율 적인 멀티 쓰레드 아키텍처

- 메인 쓰레드에서 작업(Task) 별로 쓰레드를 분기 시킴
  - 렌더링, 게임 작업
- 작업에 시간이 많이 걸리는 작업의 경우 별로도 동기화때만 처리
  - File I/O, 네트워크 (쓰레드 및 비동기 처리)
- 각 작업 중 더 복잡한 작업이 필요한 경우..
  - 게임 시뮬레이션 쓰레드에서는 가상 객체의 움직임, 물리엔진 계산, 충돌 처리등이 필요 -> 게임 시뮬레이션 쓰레드 내에서 여러 개의 쓰레드를 생성하여 처리
  - 렌더링 쓰레드 내에서도 애니메이션 작업, 파티클 처리 -> 여러 개 쓰레드로 처리

멀티 쓰레드 아키텍처 설계하기

순차적인 작업을 처리할때 필요한 아키텍처

- 순차적입 작업을 처리할때 이들을 고려한 쓰레드를 설계 함으로 병목현상 해결 가능
- 단점 : 응답 시간이 길어지고 순차적인 작업이 되지 않는 경우는 적용하기 어려움



1

멀티 쓰레딩이 필요한 작업들

주요 작업들

- 파일 입출력
  - 일반적으로 파일 입출력에는 CPU 리소스의 소모가 거의 들지 않지만 이를 다중 쓰레드로 처리함으로써 보다 더 빠른 성능 향상 가능
- 렌더링
  - 일반적으로 렌더링을 수행하는 과정과 데이터를 업데이트 하는 과정, 폭포형 아키텍처의 과정 처럼 데이터를 업데이트 한 후 렌더링을 처리하는 것으로 설계 함으로써 성능 향상 가능
  - 렌더링 과정에서는 항상 고정된 작업량을 제공 해야함!
- 그래픽 시뮬레이션
  - 순차적으로 증가하는 구름 텍스처 표현
  - 옷감 애니메이션
  - 동적인 주변광 처리
  - 새로 자라나는 식물의 처리과정
  - 파티클 표현 등
- 물리 엔진
  - 폭포형으로 해결하기에는 응답 시간이 많이 걸림
  - 최대한 많은 쓰레드를 사용하여 해결할 수 있어야 함

효율적인 멀티 쓰레딩을 위한 고려 사항

쓰레드가 쓸 수 있는 자원은 무한 하지 않다!

- 멀티 쓰레드는 결국 하나의 컴퓨터 자원을 공유
  - L1/2 캐쉬를 공유하고 실행하지만, 독립적인 레지스터 셋을 가지고 있음
- 하나의 쓰레드가 매우 빡빡한 스케쥴로 리소스를 쓰고 있다면
  - 다른 쓰레드는 많은 일들을 할 수 없고, 캐시의 용량이 넘쳐나고 최악에는 쓰레드 성능이 떨어짐
- 하나의 쓰레드를 설계할때 적절한 스케쥴로 작업을 할 수 있어야 함
  - Sleep 함수들을 사용하여 유후 상태로 넘어가는 것이 좋음
- 단 쓰레드의 변환이 너무 많이 일어나는 경우는 피해야함