Process 2 & Process 3 (Thread)

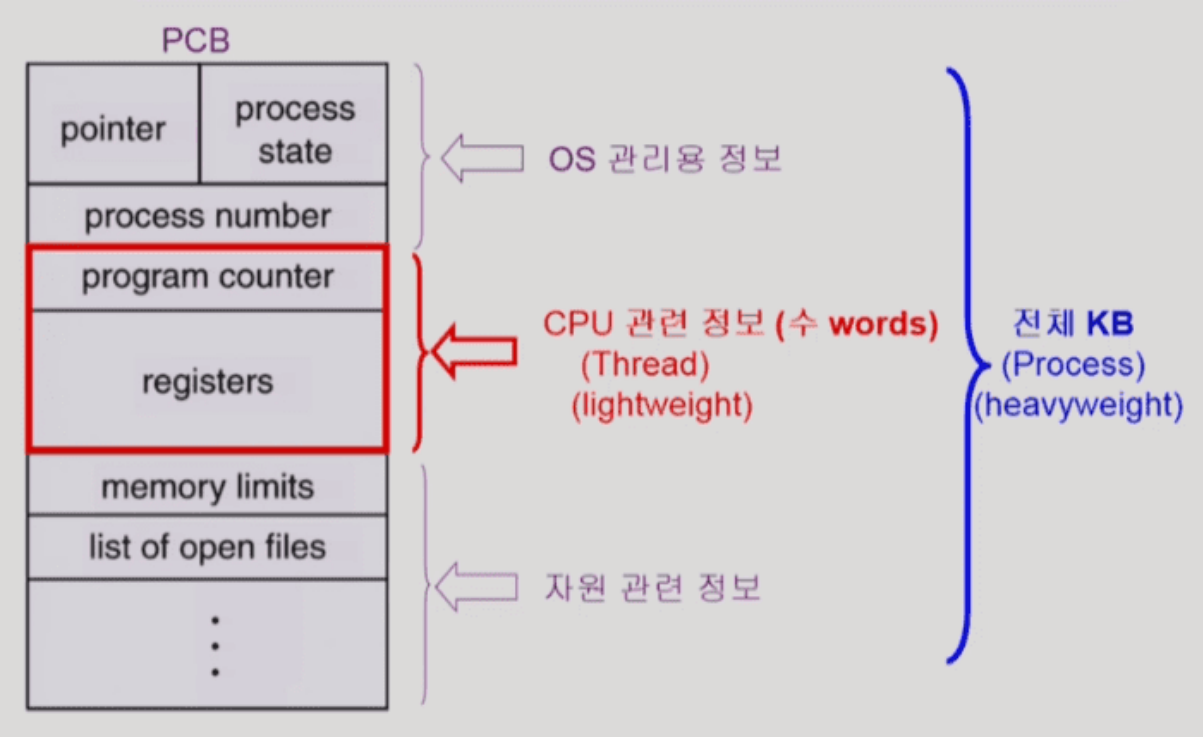
상태	완료
생성 일시	@2023년 7월 4일 오후 4:25
수업일	@2023년 7월 4일
유형	정리

Thread란

정의

프로세스 내에서 실행 되는 여러 흐름의 단위 혹은 프로세스가 할당 받은 자원을 이용하는 실행 흐름의 단위를 뜻한다.(or 스레드는 프로세스 내부의 CPU 수행 단위가 여러 개인 경우를 말함)

구성



- program counter
- register set
- stack space

쓰레드마다 CPU 관련 정보들을 각자 가지고 있어야 한다.

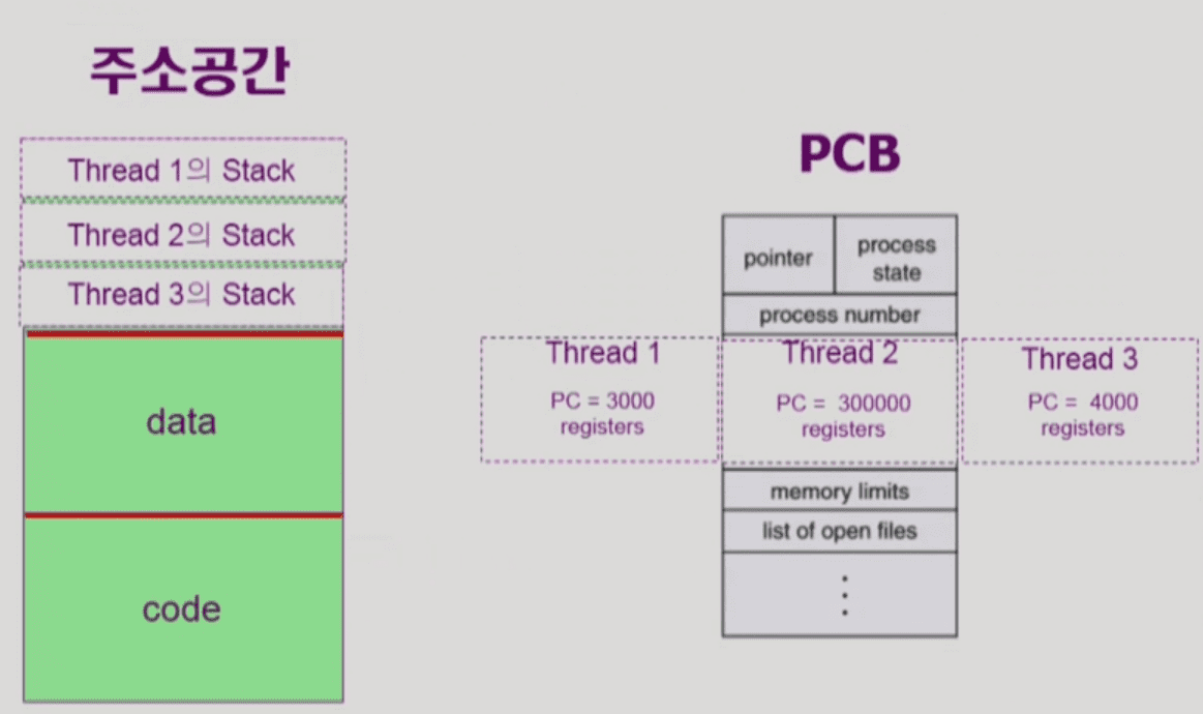
▼ Lightweight"와 "heavyweight"

Lightweight"와 "heavyweight"는 소프트웨어나 컴퓨팅 시스템에서 사용되는 용어로, 프로세스, 스레드, 컴포넌트 등의 관점에서 서로 다른 무게 또는 부담을 나타냅니다.

1. Lightweight (가볍다):
    - 경량화된 리소스나 처리 방식을 가지고 있는 것을 의미합니다.
    - 작은 크기와 적은 리소스를 요구하며, 빠르게 시작하고 실행되는 특징이 있습니다.
    - 예를 들어, 경량 스레드(lightweight thread)는 스레드 관리와 전환에 작은 오버헤드를 가지며, 더 빠르게 생성되고 소멸될 수 있습니다. 경량 컴포넌트(lightweight component)는 단순한 기능을 수행하며, 부가적인 기능이나 의존성이 적습니다.
2. Heavyweight (무거움):
    - 더 많은 리소스와 복잡한 처리 방식을 필요로 하는 것을 의미합니다.
    - 큰 크기와 높은 리소스 요구량을 가지며, 시작과 실행에 더 많은 시간과 노력을 필요로 합니다.
    - 예를 들어, 무거운 프로세스(heavyweight process)는 자체 주소 공간과 운영체제 리소스를 할당받아 실행되며, 더 많은 메모리와 계산 능력을 필요로 합니다. 무거운 컴포넌트(heavyweight component)는 다양한 기능과 의존성을 가지며, 다른 서비스와 상호 작용하는 더 복잡한 구조를 가지고 있을 수 있습니다.

일반적으로, "가볍다"와 "무거움"은 리소스 사용, 처리 시간, 복잡성 등의 측면에서 차이를 나타냅니다. 경량화된 구조는 보다 빠르고 효율적이며, 작은 규모의 작업에 적합합니다. 반면, 무거운 구조는 더 많은 기능과 처리 능력을 제공하지만, 초기화 및 실행에 더 많은 시간과 리소스를 필요로 하며, 대규모 및 복잡한 작업에 적합합니다.

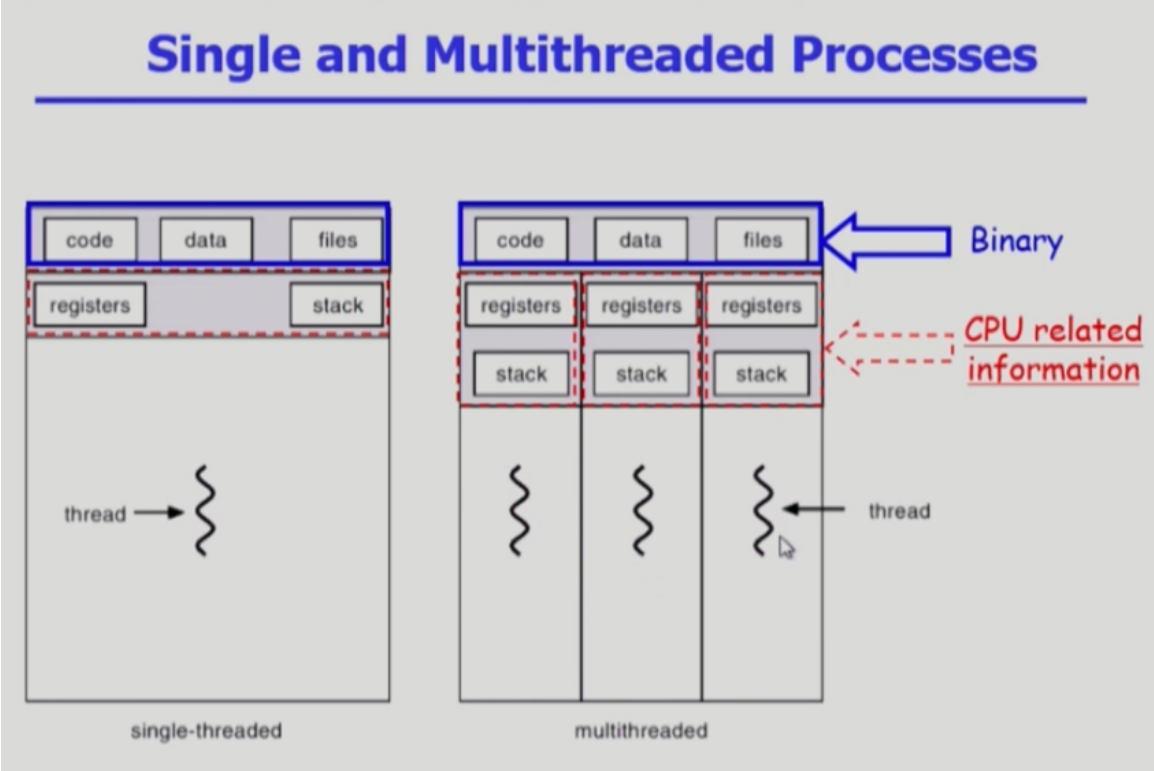
Thread가 다른 Thread와 공유하는 부분



- code section
- data section
- OS resources

Thread끼리 Address space의 data, code 영역은 공유하되 stack은 별도로 할당받는다. PCB에서는 program counter와 register set을 제외한 프로세스 관련 정보 및 자원을 모두 공유한다. 여기서 thread들이 공유하는 부분인 code section, data section, OS resources를 task라고 한다. 그래서 전통적인 개념의 heavyweight process는 하나의 스레드를 가지고 있는 task로 볼 수 있다. (하나의 프로세스의 여러 스레드가 있다가 해도 task는 하나만 있는 것임 일반적으로 프로세스 안에 여러 스레드를 두는 것을 lightweight process라고 함)

(프로세스가 만들어지면 각 프로세스별로 코데힙스 영역이 만들어지고 PCB에서 현재 메모리에 어느 부분을 실행하고 있는지 프로그램 카운터가 가리키고 있음 : 동일한 일을 하는 프로세스가 여러 개 있다고 하면 각각의 프로세스를 메모리에 다 올리게 되었을 때는 메모리 낭비가 심할 것임 그래서 같은 일을 하는 프로세스를 여러 개 띄우고 싶다고 하면 주소공간을 하나만 띄우고 스레드마다 다른 코드 영역을 처리하게 만들면 될 것임 그리고 PCB에 프로그램 카운터에서 어느 부분을 실행하고 있는지 기록을 해주면 됨(각 스레드마다) 즉, 스레드마다 CPU 레지스터의 어떤 값을 넣고 메모리의 어느 부분을 가리키며 실행하고 있는지 별도로 기록하는 구조로 되어있음. 스레드도 별도의 스택으로 관리 해야 함 스레드끼리는 자원을 대부분 공유하고 고유하게 가지는 것은 자신의 스택 영역과 PC값뿐임)



장점

응답성 (Responsiveness)

다중 스레드로 구성된 Task 구조에서 하나의 서버 스레드가 blocked(waiting) 상태인 동안에도 동일한 Task 내의 다른 스레드가 실행되어 빠른 처리가 가능하다. 예를 들어 웹 브라우저가 스레드를 여러 개 가지고 있을 때, 하나의 스레드는 이미지를 비롯한 추가 데이터를 받기 위해 서버에 요청을 걸어서 blocked 상태가 된 후(처음 HTML 문서가 날아오고 그다음 웹브라우저가 해석해서 다시 이미지나 리소스들을 웹서버에 요청함), 다른 스레드가 이미 받아 놓은 HTML 텍스트를 화면에 출력할 수 있다. 이러한 비동기식 입출력을 통해 응답성을 높일 수 있다.

자원 공유 (Resource Sharing)

하나의 프로세스 안에 CPU 수행 단위인 쓰레드를 두게 되면 code, data, resource 자원을 공유하여 효율적으로 자원 활용이 가능하다.

경제성 (Economy)

- 동일한 일을 수행하는 다중 쓰레드가 협력하여 높은 처리율(throughput)과 성능 향상을 얻을 수 있다.
- 새로운 프로세스 하나(여러개)를 만드는 것 보다 기존의 프로세스에 쓰레드를 추가하는 것이 오버헤드가 훨씬 적다.
    - 프로세스의 Context Switching과 달리 캐시 메모리를 초기화할 필요가 없고, cpu 관련 정보 저장하는 작업 등 필요 없이 Code, Data, Resource 영역을 공유하므로 Stack 영역만 처리하면 되기 때문이다.

멀티 프로세서(CPU가 여러개) 아키텍처에서의 이용성 (Utilization of MP Architectures)

각각의 쓰레드가 서로 다른 CPU를 가지고 병렬적으로 작업을 진행해서 훨씬 효율적으로 작업을 수행할 수 있다.

단점

- 자원의 공유로 인한 동기화 문제(여러 개의 실행 스레드나 프로세스가 공유된 자원에 동시에 접근할 때)가 발생할 수 있다.
- 디버깅이 까다롭다.

▼ 동기화

동기화는 여러 스레드 또는 프로세스 사이에서 실행 순서나 동작을 조정하여 상호간의 일관성을 유지하는 메커니즘입니다. 동기화는 동시에 실행되는 여러 개의 스레드 또는 프로세스가 공유 자원에 안전하게 접근하고 조작할 수 있도록 하는 중요한 개념입니다.

일반적으로 병렬 컴퓨팅이나 동시성 프로그래밍에서 여러 스레드 또는 프로세스는 하나 이상의 공유된 자원(예: 메모리, 파일, 데이터베이스 등)에 동시에 접근할 수 있습니다. 이러한 동시 접근은 경쟁 상태나 일관성 문제를 야기할 수 있습니다. 예를 들어, 한 스레드가 공유 변수의 값을 변경하는 동안 다른 스레드가 동시에 해당 변수에 접근하여 값을 읽는 경우, 예측할 수 없는 결과가 발생할 수 있습니다.

동기화는 이러한 문제를 해결하기 위해 공유 자원에 대한 접근과 조작을 조정하는 메커니즘을 제공합니다. 동기화 메커니즘을 사용하면 여러 스레드 또는 프로세스가 공유 자원을 안전하게 사용할 수 있도록 실행 순서를 조절하거나 접근 권한을 관리할 수 있습니다. 이를 통해 데이터 일관성을 유지하고, 경쟁 상태를 방지하며, 예측 가능한 실행 결과를 보장할 수 있습니다.

동기화 메커니즘에는 상호 배제(mutual exclusion), 임계 구역(critical section), 세마포어(semaphore), 락(lock) 등이 있습니다. 이러한 메커니즘을 사용하여 스레드 또는 프

로세스 간의 동기화를 구현할 수 있습니다.

구현 방식

Kernel Threads (커널이 thread가 **있다는 것을 알고 있음)**

여러 개의 thread가 있다는 사실을 운영체제가 알고 있어서 thread 간의 CPU 교환을 커널이 CPU 스케줄링하듯이 관리한다.

- Windows 95, 98, NT
- Solaris
- Digital UNIX, Mack

User Threads (사용자 수준)

여러 개의 thread가 있다는 사실을 운영체제가 모르는 상태에서 사용자 프로그램이 라이브러리의 지원을 받아 스스로 여러 개의 thread를 관리한다. 커널이 thread의 존재를 모르기 때문에 구현에 제약이 있을 수 있다.

- POSIX Pthreads
- Mach C-threads
- Solaris threads

Real-Time Threads

리얼 타임 기능을 지원하는 thread가 있다는 것만 알자