# Thread

2022년 4월 23일 토요일 오전 11:21

## Thread

한 프로세스 내에서의 실행 흐름의 단위 혹은 cpu가 처리할수있는 작업의 최소 단위 -> CPU에서 실행되는 단위

멀티 프로세스처럼 프로세스르 context switching 방식으로 처리 되면 overhead가 상당히 크다
-> 하나의 새로운 프로세스가 이미 존재하는 프로세스 위에서 동작한다면?==> 스레드의 발상 시작

- 하나의 프로세스는 최소 하나의 스레드로 구성
- 하나의 스레드르 실행하는데 cpu내의 하나의 코어 필요 (현대는 대부분 멀티 코어 사용[병렬처리 (Parallel 처리 ])
  - 최근에는 하나의 코어에서도 2개의 스레드 동시 실행 가능
  - 다수의 스레드 동시 실행 가능

**Thread : unit of scheduling**
**Process: unit of resource ownership**

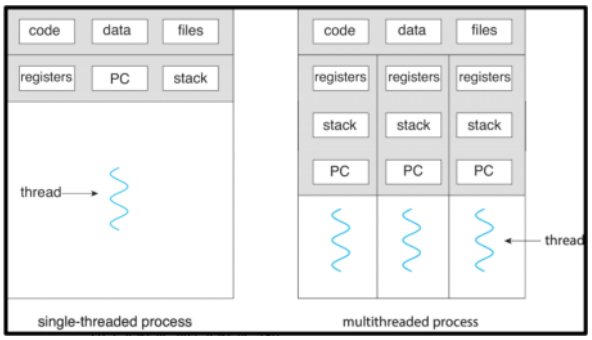
## 구성

스레드가 공유 자원으로 쓰는건 프로세스의 code,data,file이다

그 프로세스의 메모리 영역을 공유한다

- Tid
- Pc
- Register set
- Stack

-> Thread마다 고유한 값을 가지고 있다




화면 캡처: 2022-04-25 오전 11:02

## Multi Thread Pros and cons

- **Responsiveness(빠른 응답성)**
  - 연산을 진행하면서도 사용자와 상호작용가능한 상태로 유지가 되는 특성
    - **Single thread**
      - 다른 io 장비 쓸때 연산 끝날때 까지 사용자는 다른 io장비 쓸수 없음
    - **Multi thread**
      - 해당 연산을 진행함과 동시에 사용자는 사용가능
- **자원 공유로 인한 작업의 병렬화**
  - 같은 프로세스의 데이터 및 주요 자원들을 공유함으로써 해당 데이터들을 대상으로 동시에 서로 다른 작업을 할수있다
- **프로세스보다 높은 경제성**
  - 스레드간의 context swtching에 소요되는 시간 또한 프로세스보다 간단-> 공유 자원이 있기 때문
- **Scalability (확장성)**
  - 프로세스 안에 다수의 스레드가 존재하는 경우 멀티 코어/멀티 프로세스의 이점을 충분히 활용하여 하나의 프로세스의 각기 다른 부분이 다수의 코어에서 동시 실행

**Cons :** 하나의 스레드에 문제가 발생하면 프로그램 전체에 영향을 줄 수있다

멀티 프로세스와 멀티 스레드는 처리 방식의 일종일 뿐이다
크롬 같은 경우 멀티 프로세스이다 오버헤드가 크긴하지만 독립적인 메모리 가지고있어서 동기화 문제를 신경쓰지 않아도 됨
멀티 스레드 같은 경우 더 효율적으로 보이지만 하나의 스레드에 문제가 있다면 전체의 프로그램에 영향을 줄 수 있다
동기화 문제에 신경을 써야한다

## 동시성과 병렬성

**Concurrency**

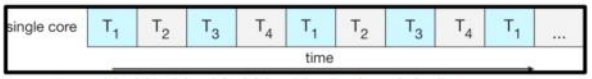
여러 작업을 같이 처리 하는 성격

- 한번에 하나의 작업을 하지만(time sharing) 처리량 늘리기 위해 다른 작업들도 함께 처리
  - => 프로세스간의 context switching
-

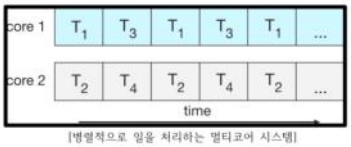
**Parallelism**

한번에 여러 작업을 동시에 처리 ==> 병렬성 , 하드웨어적 처리에 가깝다
동시성 ⊃ 병렬성

멀티 코어 같은 경우 병렬성인데 하드웨어적 처리 방식인 병렬성을 가지고 있다고 생각하면 될듯



[동시성을 띄기는 하나, 병렬성은 띄지를 않는 싱글 코어 시스템]

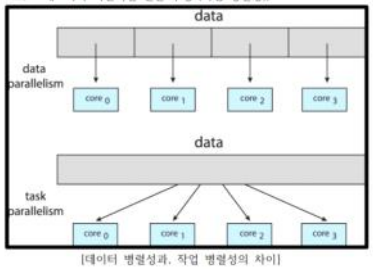


[병렬적으로 일을 처리하는 멀티코어 시스템]

화면 캡처: 2022-04-25 오전 11:18

1. **Data Parallelism (데이터 병렬성)**
   a. 프로세스 내에 공유되는 데이터들을 스레드 개수 만큼 나누어 각각의 스레드가 본인들에게 주어진 각각으 ㅣ데이터들의 부분에 대해 똑같은 작업을 수행하는것
2. **Task Parallelism (작업 병렬성 )**
   a. 각각의 스레드가 모두 같은 데이터들을 공유하지만 이들을 가지고 스레드는 각각의 고유한 연산을 진행
   b. 데이터들의 분할이 이뤄지지 않고 각 스레드는 모두 동일한 데이터 집합 사용 하지만 각 스레드는 각자의 독단적인 연산 작업을 수행

※ 2가지 병렬성은 완전 별개의 것이 아님(상호 배타적 ×). 예를 들어, 한 프로세스는, 데이터 병렬성과, 작업병렬성을 모두 필수 있음 (먼저, 데이터들을 분산하여 처리(데이터 병렬성),하고, 이후, 종합된 데이터를 대상으로, 쓰레드마다 독단적인 연산 수행 (작업 병렬성))



[데이터 병렬성과, 작업 병렬성의 차이]

화면 캡처: 2022-04-25 오전 11:27
2가지는 완전 별개의 것이 아니다 동시에 만족 될 수 있다

## 멀티 스레드 프로그래밍

2개 이상의 스레드 사용하는 프로그램 디자인

**고려 사항**

- **멀티 스레딩이 가능한 작업 규명 (dntifying Tasks)**
  - 하나의 거대한 작업을 서로 독립적인 하위 작업들로 나눌수있어야한다
  - 분리된 두 개 이상의 하위 작업들이 서로 종속적인 관계(하나의 스레드가 일을 어느정도 해야 다른 스레드가 일을 진행 약강 파이프라이닝때와 비슷 )에 있지 않아야한다
- **Balance**
  - 멀티 스레딩 처리로 인해 처리량이 1/n으로 가까워 질수록 균형이 좋다고 할 수 있다
- **Data Splitting and Dependency (데이터 분리 및 의존성)**
  - 일부 작업의 경우 데이터에 접근하는 스레드 간에 동기화 혹은 각 스레드만이 사용해야하는 고유한 데이터에 대한 명시 필요 하다
  - 데이터가 공유 되기에 한 스레드에서 데이터 일부 변경 시키면 이 변경된 데이터를 다른 스렏가 그대로 사용하게 되는일이 발생

## Multi Thread Model

**User thread**

- 사용자 및 프로그래머에 의해 응용 프로그램 영역에서 생성 /소멸 되는 스레드
- 각 언어별로 제공되는 API이용해서 프로그램 개발자가 필요에 의해 직접 생성하는 스레드
- 스레드 생성/연산/종료에 관하여 커널은 개입하지 않는다 -> 커널은 사용자 스레드에 대해서 존재 여부도 알지 못함

**Kernel thread**

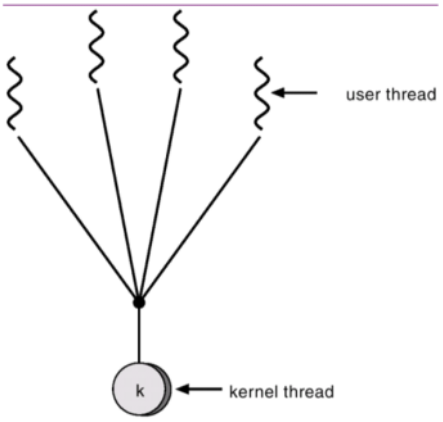
- 운영체제에 의해 커널 영역에서 생성/소멸되는 스레드
- 개발자의 요청에 따라 커널이 직접 스레드를 생성해줄 경우 해당 생성된 스레드를 커널 스레드라고 한다
- 커널에 의해 직접 관리 된다

- 사실상 스레드가 cpu 코어에 할당되는것은 운영체제가 해당 사용자 스레드와 매핑된 커널 스레드를 cpu 코어에 할당하는 것 이다

### 3가지 모델

#### Many to one

- 여러 개의 사용자 스레드들이 생성되면 이들중 하나만 커널 스레드로 매핑되는 방식
- -> 커널은 매핑 된 하나의 스레드 이외의 사용자 스레드에 대해서는 존재자체를 모름 즉 커널 입장에서는 하나의 스레드만 돌고있는 것 이다
- **Pros**
  - 사용자가 원하는 만큼 사용자 스레드 생성 가능 이에 대해 커널은 개입하지 않는다 -> overhead X
  - 다수의 스레드 간의 스케쥴링이 필요한 경우 커널 영역의 스케줄러가 아닌 스레드 관련 라이브러리에서 코드로 구현된 스케줄러를 사용하게 된다 이때 사용자가 만든 스케줄러를 사용하기 때문에 커널에 진입안해도됨
- **Cons**
  - 다수의 사용자 스레드를 만들어도 하나의 스레드밖에 커널은 인식하지 못하기 때문에 멀티 코어 프로세서 시스템을 잘 활용하지 못한다
  - 커널은 1개의 스레드만 있다고 생각하기 때문에 해당 스레드가 대기 상태로 context switching 호출되면 나머지 스레드들도 대기 상태가 된다 -> 한번에 하나밖에 수행 못한다는 이야기 그래서 멀티 프로세서 시스템 잘 활용못한다고 한것이다
  - 이러한 이유로 현대에서 이 모델은 쓰지 않는다

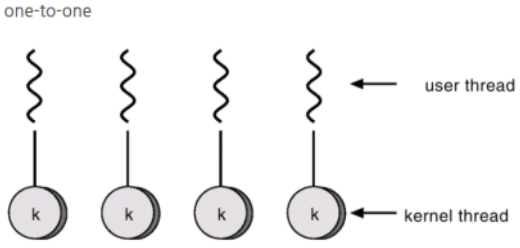



화면 캡처: 2022-04-25 오전 11:50

#### One to One

- 사용자 스레드들과 커널 스레들이 1:1 매핑 되는 방식
- **Pros**
  - Many to one model보다 동시성이 보장되고 각각의 사용자 스레드에 대한 독립적인 커널 스레드들이 매핑 되어 각각의 매칭된 사용자 스레드들이 연결된 커널 스레드와 함께 각각의 cpu 코어에 할당 -> 멀티 코어시스템을 충분히 활용 할 수 있다
  - 한 사용자 스레드가 시스템 호출 해도 다른 사용자 스레드도 사용가능

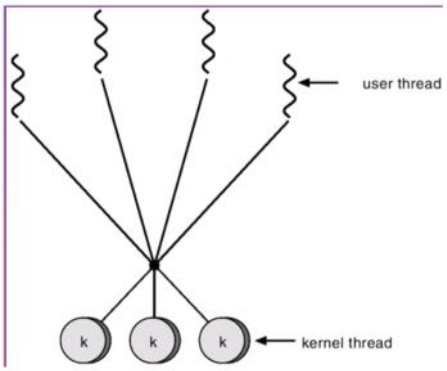
- **Cons**
  - 사용자 스레드 만들때 마다 해당하는 커널 스레드도 만들어야한다 -> 다수의 커널 스레드 생성은 overhead
  - 스레드간의 스케쥴링위해서 커널을 직접 호출하게 되면 오버헤드




화면 캡처: 2022-04-25 오후 12:50

#### Many to Many

- 다수의 사용자 스레드를 다수의 커널 스레드에 매핑 시키는 방식
- 다대일,일대일 모델의 단점을 보완한 모델
- 커널 스레드수는 사용자 스레드의 수와 같거나 작을수있다
- 한 사용자 스레드가 대기 상태로 전환시키는 시스템 호출 발생시켜 해당 사용자 스레드가 대기 상태가 되면 대기 상태에 놓인 사용자 스레드에 연결된 커널 스레드는 다른 사용자 스레드에 매핑되어 해당 스레드에게 cpu코어 할당해 실행 시킴



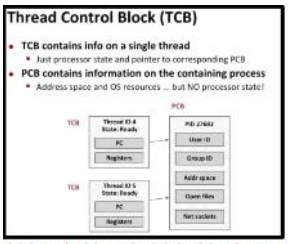

화면 캡처: 2022-04-25 오후 12:54

## TCB context switching

스레드에 대한 정보들은 tcb에 저장된다

**저장 되는 것 들**

- **Tid**
- **Stack pointer**
- **Program counter**
- **Register**
- **PCB pointer**
- **Thread state**




하나의 PCB와, 해당 프로세스에 속한 2개의 쓰레드들(TCB)

화면 캡처: 2022-04-25 오후 12:56

기존의 context switching은 프로세스가 단위였다 이렇게 되면 매번 문맥 전환 할때마다 주소공간의 대체가 발생하고 overhead가 커진다는 문제 발생

Thread 문맥전환시에는 같은 공간을 공유하기때문에 overhead가 작다

단, 다른 프로세스에 속한 스레드끼리 문맥전환 발생하면 기존의 프로세스 문맥전환과 마찬가지로 주소공간이 대체 된다

## 스레드 기법

**스레드 생성**

**동기적 멀티 스레딩(synchronous threading)**

- 부모 스레드와 자식 스레드들 혹은 한 프로세스 내에서 스레드들이 종속적인 관계에 있는 경우 해당 스레드들끼리는 독단적으로 수행 될 수없으며 어떤 경우에는 해당 스레드들끼리는 동시 실행이 불가능하다
- 보통 부모 스레드가 자식 스레드의 특정 연산 및 종료까지 기다리는 형태가 된다

**비 동기적 멀티 스레딩(asynchronous threading) -> 서버에서 쓰이는 모델**

- 부모 스레드부터 자식 스레드가 생성된 직후 각자의 스레드들이 독립적으로 동시에 실행되며 일반적으로 이 스레드들간의 관계는 독립적이다
- -> 각 스레드들간의 공유되는 데이터가 거의 없거나 작고 각자 독단적인 연산을 수행하는 환경
- -> 주로 다양한 프로토콜을 사용하는 하나의 서버에 사용되는 운영체제는 비동기적 멀티스레딩을 사용

**스레드에 대한 작업 할당 및 스레들의 개수 관리를 누가 하는가?**

**명시적 스레딩(Explicit threading)**

- 프로그래머에게 직접 API를 줘서 프로그래머가 직접 스레드에게 작업 할당/시작 /스레드 개수 관리를 하도록 하는 방법 이다.

**암묵적 스레딩(Implicit Threading )**

- 스레드에게 작업 /할당/ 실행/ 스레드 개수 관리를 프로그래머가 아닌 해당 런타임 라이브러리가 자동으로 하도록하는 방법

- **쓰레드 풀(Thread Pool)**
  - 해당 프로세스에서 쓰일 다수의 스레드들을 사전에 생성해 풀에 위치시킨 다음 스레드가 필요할때마다 풀에 꺼내서 사용
  - 프로그래머는 스레드 풀 라이브러리에서 제공하는 함수를 통해 스레드 풀 생성하고 작업만 전달하면 됨
  - 프로그래머가 전달한 작업을 스레드에 할당할지 말지는 관련된 런타임 라이브러리가 결정
  - 가용할수있는 스레드가 풀에 없을 경우 하나의 스레드가 작업을 마칠때까지 해당 작업은 대기 상태

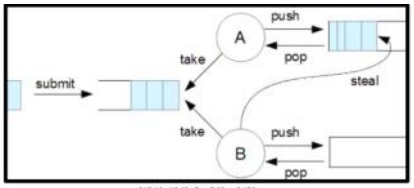
**Pros**

- 필요할때마다 새로운 스레드를 생성하는 것 보다 시간 절약된다
- 최대 생성가능한 스레드를 정할수있어 시스템 자원을 과도하게 사용하는거 방지한다
- 스레드의 숫자에 대한 제한을 바꿔가면서 스레드 수의 제한으로 인해 발행하는 작업 수행 상의 지연을 이용하여 전체적인 작업들의 처리 방식을 유연하게 할수있다.
- 예 )
  만약, 한 프로세스 내에 5 개의 부분 작업들이 있고, 이중 4개는 독단적인 연산
  을 수행하고, 나머지 1개의 작업은, 4개의 서로 다른 연산들의 결과물을 똑같은
  방법으로 처리하는 작업이라고 가정하자. 이때, 우리는, 굳이 8개의 쓰레드 (4개
  의 독단적 연산들에 해당하는 쓰레드 4개 + 연산 결과들을 처리하는 쓰레드 4개)
  가 필요없다는 것을 알 것이다. 대신, 4개의 쓰레드들만 생성하여, 첫 번째로 연
  산이 끝나면, 종료된 쓰레드에게, 바로, 연산 결과에 대한 처리 작업을 요청하면 되기 때문.. (어차피 연산 작업 동안, 처리 작업은 계속 지연되기 때문...)

- **포크 조인 기법 (fork join)**
  - 자바에서 제공하는 멀티 스레딩중 하나 , 분할 정복 알고리즘을 효과적으로 수행하기 위해 하나의 작업을 다수의 하위작업들로 분할 처리 이후 이들의 결과들을 병합하여 하나의 최종적인 결과물 생성
  - 역시 스레드 개수와 작업 할당에는 런타임 라이브러리에서 직접 결정한다

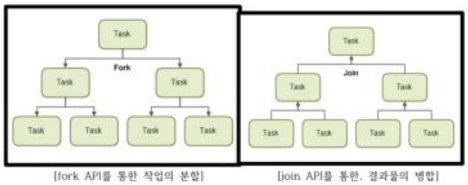
  - **작업 탈취(work stealing)**
    - 각각의 스레드들이 최대한 놀지 않고 작업을 수행토록하기 위해, 하나의 스레드가 더 이상 처리할 작업이 없는경우 다른 스레드의 남은 작업을 가져와서 처리해주는 방법
    - 각각의 스레드는 내부적으로 각자의 Queue를 가지고있다
    - 각각의 스레드는 push/pop 을 통해 Queue작업을 추가 / 꺼내기 수행 하게 된다

    - 놀고 있는 스레드 B가 A의 작업을 가져올려 하는 경우
      - A는 B에게 가져가도 되는 작업을 찾는다



[작업 탈취에 대한 이해]

화면 캡처: 2022-04-25 오후 1:31

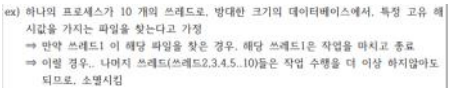


[fork API를 통한 작업의 분할] [join API를 통한, 결과물의 병합]

화면 캡처: 2022-04-25 오후 1:31

## 스레드 취소

스레드가 작업을 마치고 소멸 되는게 아니라 작업 도중에 소멸되거나 소멸 시키는것을 의미한다


ex) 하나의 프로세스가 10 개의 쓰레드로, 방대한 크기의 데이터베이스에서, 특정 고유 해
시값을 가지는 파일을 찾는다고 가정
⇒ 만약 쓰레드1 이 해당 파일을 찾은 경우, 해당 쓰레드1은 작업을 마치고 종료
⇒ 이럴 경우.. 나머지 쓰레드(쓰레드2,3,4,5..10)들은 작업 수행을 더 이상 하지않아도
되므로, 소멸시킴

화면 캡처: 2022-04-25 오후 1:33

**Target Thread**

취소시키고자 하는 스레드

- **비동기적 취소**
  - 다른 스레드에서 타겟스레드를 즉시 취소시키는것
  - 많이 사용하지는 않음 왜>
    - **메모리 누수**
      - 타겟 스레드에서 힙영역에 메모리 동적할당한 경우 스레드 취소후 메모리 누수
    - **데이터 회복 불가능**
      - 타겟스레드에서 공유되는 데이터를 갱신하는 도중 해당 타겟 스레드가 취소되는 경우 강제 종료시켜버리는 비동기적 취소방법에서 자주 발생한다
- **지연취소**
  - 다른 스레드에서 타겟 스레드에 대한 취소 요청이 들어오면 타겟 스레드 스스로가 취소가능한 상태인지 검사하여 취소가능 상태라면 해당 타겟 스레드는 취소됨 그렇지 않다면 취소가능 상태가 될때까지 타겟 스레드는 실행

☞ POSIX에서 정의하는 쓰레드 취소 관련 API는, 타겟 쓰레드가, 지연 취소 방식으로 취소되고, 취소
가능 상태를 갖는다하더라도, 타겟 쓰레드의 취소는, 취소지점 까지 대기됨

☞ 보통 취소지점은, 해당 쓰레드를 Block 혹은, Wait(대기 상태)로 전환시키는, 시스템 호출들이 해당
(read, write, ...)

☞ 취소 지점 도달 시, 타겟 쓰레드는 종료되기 전에, 클린업 핸들러를 호출하여, 아래의 메모리 누수나,
데이터 회복 불능 상태가 되지 않도록, 뒤처리를 시작

## Thread Scheduling

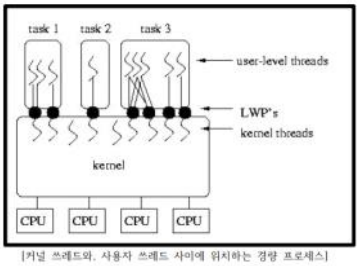
**Lightweight process LWP**

- Many to many모델에서 사용자 스레드와 커널 스레드 사이에 위치하는 자료구조로 각각의 커널 스레드는 하나의 LWP를 가진다
- LWP 는 커널 내부에 존재
- 커널 스레드와 사용자 스레드를 연결하는 역할
- 사용자 영역에서도 LWP접근이 가능하며 해당 LWP영역에 커널 스레드와 사용자 스레드가 데이터를 쓰고 이로부터 읽어옴으로써 소통이 가능해진다
- 커널 스레드의 상태에 따라 해당 커널 스레드 소유의 LWP 상태 또한 변하게 되고 이에 따라 해당 LWP에 매핑된 사용자 스레드들도 매핑된 LWP와 똑같은 상태로 변하게 된다

ex) write 시스템 호출이 발생하면, 커널 모드에서, write 시스템 호출 함수가 실행되면서, 출력에 대한 행동의 제어권을, 해당 출력 장치의 IOP에게 넘기게 되는데, 이 시기부터, 출력이 완료될 때까지, 해당 커널 쓰레드는, 대기 상태(Block)가 되고, 해당 커널 쓰레드 소유의 LWP도 대기 상태가 된다. 이윽고, 해당 LWP에 매핑된 모든 사용자 쓰레드들도, 대기 상태가 된다 .

⇒ 각각의 LWP는, 하나의 커널 쓰레드와 매핑되어, 커널에서 쓰레드 스케줄링의 단위가 됨
= 각각의 LWP는 하나의 스케줄링 단위로, 그 수가 많을수록, 동시(병렬성)에 실행될 수 있는 쓰레드의 수가 많아짐..
= 마치, LWP가 프로셋처럼 보이는 착각
= ∴ LWP를 가상 프로세서 라고함
⇒ 일반적으로, I/O 연산이 상당히 많은 멀티 쓰레드 프로세스일수록, 많은 수의 LWP가 필요

화면 캡처: 2022-04-25 오후 1:44



[커널 쓰레드와, 사용자 쓰레드 사이에 위치하는 경량 프로세스]

화면 캡처: 2022-04-25 오후 1:45

## 멀티 태스킹,멀티 프로세스, 멀티 스레드 차이

멀티 태스킹 : 하나의 CPU에 두개의 프로세스가 경합이 될때
멀티 프로세싱: 두개이상의 CPU 코어와 프로세스를 활용하는 시스템
멀티 쓰레딩 : 프로세스 안에 여러 개의 스레드가 있다