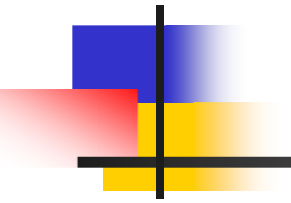
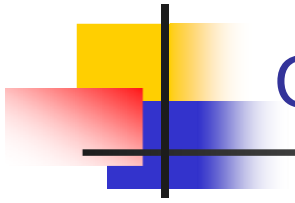


Operating Systems
(Threads & Concurrency)

Chapter 4

These lecture materials are modified from the lecture notes written by A. Silberschatz, P. Galvin and G. Gagne.

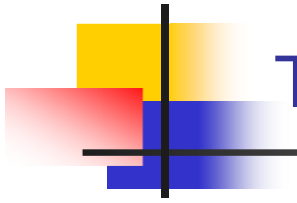
August, 2022



Outline

- Thread concept
- Multithreading
- Multithreading Models

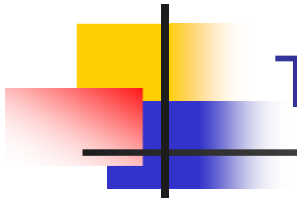
Thread concept



Thread concept

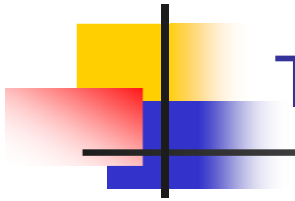
- 지금까지 소개한 프로세스 모델은 프로세스가 single thread를 가졌다고 가정.
- 그러나 실제로는 프로세스는 여러 개의 스레드를 포함.
- 스레드를 사용하여 병렬 처리를 수행하는 것은 최신 멀티코어 시스템에서 매우 중요함

- 이 장에서는 스레드의 개념 및 설계에 대하여 알아봄



Thread concept

- 프로세스 : 운영체제로부터 자원을 할당받는 작업의 단위
- 스레드 : 프로세스가 할당받은 자원을 이용하는 실행 단위이며, 프로세스 내에 여러 개가 생성됨.



Thread concept

- 스레드의 구성
  - Thread ID
  - Program counter(PC)
  - Register set
  - Stack space

- 프로세스처럼 스레드도 스레드 간에 context switch가 이루어짐
- 동일한 프로세스에 속한 다른 스레드들와 code section, data section, 그리고 open files, signals과 같은 운영 체제 리소스를 공유함.

Multithreading

Multithreading

- Single-threaded process는 한 번에 하나의 작업만 가능함.
- Multithreaded process는 한 번에 둘 이상의 작업의 수행이 가능함.

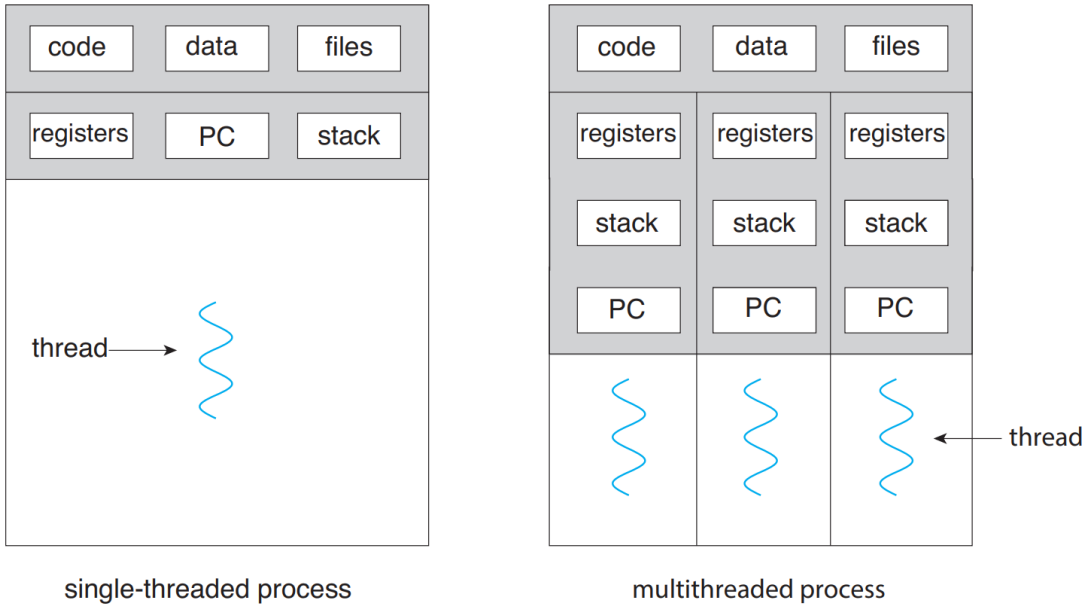
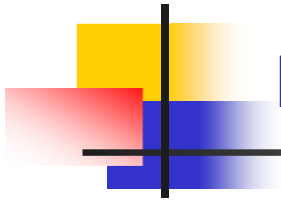


Figure 4.1 Single-threaded and multithreaded processes.



Multithreaded process의 예

- 웹 브라우저에는 한 스레드가 이미지나 텍스트를 보여주는 동안, 다른 스레드는 네트워크에서 데이터를 검색할 수 있음
- 워드프로세서에는 편집 화면을 보여주는 스레드, 사용자의 키 입력에 응답하는 스레드, 백그라운드에서 맞춤법 및 문법 검사를 수행하는 세 번째 스레드가 있음.
- 운영 체제 커널도 일반적으로 각 스레드는 장치 관리, 메모리 관리 또는 인터럽트 처리와 같은 작업을 따로 수행하는 멀티스레드임.

Multithreaded process의 예

- 웹 서버는 동시에 액세스하는 클라이언트가 여러 개 있을 수 있음. 여러 클라이언트의 요청을 처리하고 서비스하기 위해서는 각 클라이언트마다 스레드가 필요함.

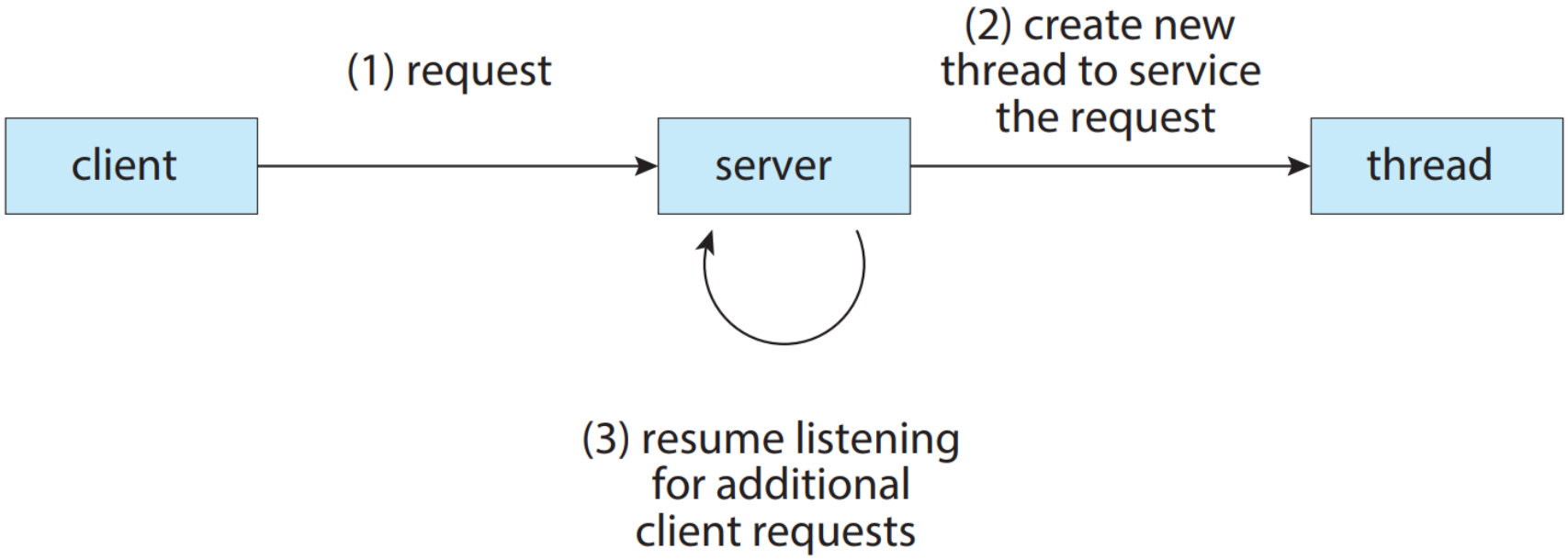
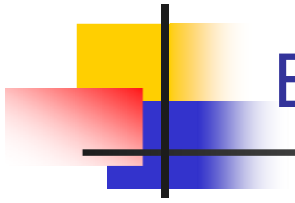
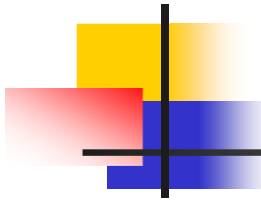


Figure 4.2 Multithreaded server architecture.



Benefits

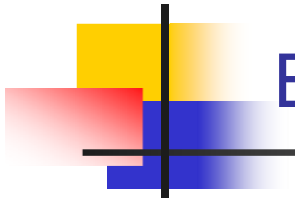
- 멀티 스레드 프로그래밍의 장점 4가지
  - 1. Responsiveness
  - 2. Resource sharing
  - 3. Economy
  - 4. Scalability



Benefits – 1. Responsiveness

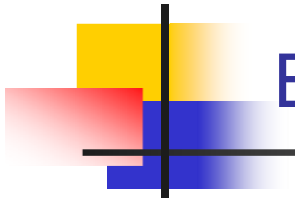
- interactive application의 멀티스레딩은, 일부가 차단되거나 긴 작업을 수행하는 경우에도 application이 계속 실행되도록 하여 사용자에 대한 응답성을 높일 수 있음.
- 사용자 인터페이스를 디자인할 때 특히 유용함.

- 예를 들어, 사용자가 버튼을 클릭하여 시간이 많이 소요되는 작업을 수행할 때:
  - single-threaded process는 작업이 완료될 때까지 사용자에게 응답하지 않음.
  - multithreaded process는 그 작업을 수행하는 스레드 외에는 모두 동작하므로, application은 사용자에게 계속 응답.



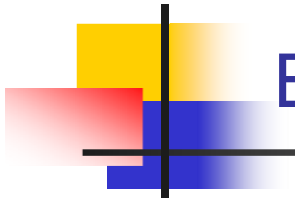
Benefits – 2. Resource sharing

- 프로세스는 shared memory, message passing과 같은 기술을 통해서만 리소스를 공유할 수 있음.
  - 이러한 기술은 프로그래밍이 필요함.
- 그러나 스레드는 기본적으로 자신이 속한 프로세스의 메모리와 리소스를 공유함.



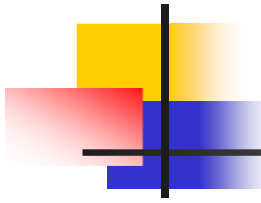
Benefits – 3. Economy

- 일반적으로 스레드 생성은 프로세스 생성보다 시간과 메모리를 덜 소모함.
- 일반적으로 프로세스 간보다 스레드 간의 Context Switch가 더 빠름.



Benefits – 4. Scalability

- single-threaded process는 사용 가능한 코어 수에 관계없이 하나만 실행 가능함.
- multithreaded process는 각 코어마다 다른 스레드가 병렬로 실행 가능함.
  - 같은 시간동안 더 많은 일을 수행할 수 있음.



Concurrency / Parallelism

- **Concurrency:**
  - 하나의 코어는 한 번에 하나의 스레드만 실행할 수 있기 때문에, 스레드들이 교차로 배치됨(interleaved)을 의미함.
  - 프로세스가 동시에 실행되지만 병렬로 실행되지는 않음.

- **Parallelism:**
  - 각 코어에 별도의 스레드를 할당할 수 있기 때문에 일부 스레드가 병렬로 실행될 수 있음을 의미함.
  - 멀티코어 시스템만 가능함.

Concurrency / Parallelism

- **Concurrency:**

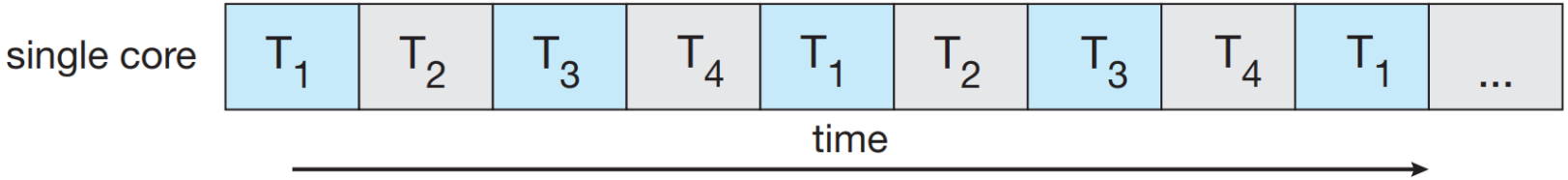


Figure 4.3 Concurrent execution on a single-core system.

- **Parallelism:**

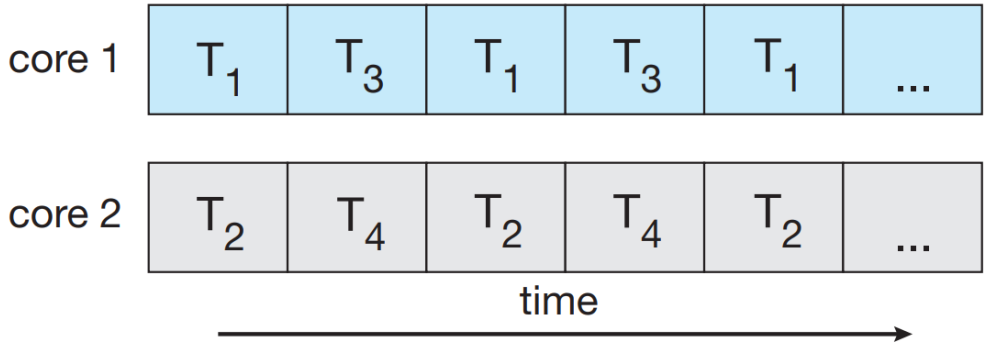
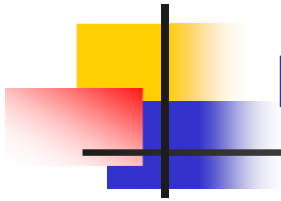


Figure 4.4 Parallel execution on a multicore system.



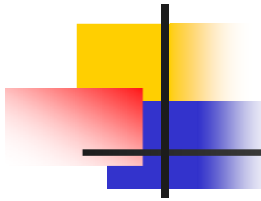
Programming Challenges

- 개발자가 멀티 프로그래밍에서 고려할 것 5가지

- **1. Identifying tasks:** 별도의 동시 task으로 나눌 수 있는 영역을 찾기 위해 application을 체크할 필요가 있음.

- **2. Balance:** 개발자는 병렬로 실행할 수 있는 task의 영역을 나눌때, task가 동일한 가치인지 확인이 필요함.
  별도의 실행 코어를 사용하여 task를 사용하는 것이 비용의 측면에서 가치가 있는지 확인해야 함.

- **3. Data splitting:** 각 task에서 액세스하고 조작하는 데이터는 별도의 코어에서 실행되도록 분할되어야함.



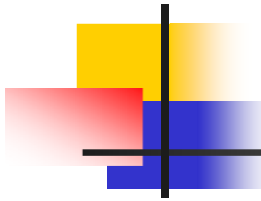
Programming Challenges

- 개발자가 멀티 프로그래밍에서 고려할 것 5가지

- **4. Data dependency:** 한 task이 다른 task의 데이터에 의존하는 경우, 개발자는 task가 Data dependency를 수용하도록 동기화되었는지 확인해야함.

- **5. Testing and debugging:** 멀티스레딩 프로그램을 테스트 및 디버깅하는 것은 본질적으로 싱글스레드 프로그램보다 어려움.

Multithreading Models

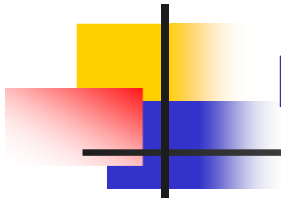


Multithreading Models

- 스레드는 유저 스레드와 커널 스레드로 나뉨.

- **유저 스레드:** 스레드의 기능을 제공하는 라이브러리를 활용하는 방식, 커널 지원 없이 관리
  - 장점: 성능 저하가 없음.
  - 단점: 안정성이 떨어짐. 스레드 하나에 문제 발생시, 같은 프로세스의 전체 스레드가 중단됨.

- **커널 스레드:** 운영 체제에서 직접 지원 및 관리됨.
  - 장점: 안정적임
  - 단점: 유저 모드에서 커널 모드로 context switching이 필요하여 성능이 저하됨.



Multithreading Models

- 유저 스레드와 커널 스레드 간의 관계를 설정해야함
- 일반적인 방법은 다음과 같음
  - Many-to-One Model
  - One-to-One Model
  - Many-to-Many Model
  - Two-level Model

Many-to-One Model

- 하나의 커널 스레드에 여러 개의 유저 스레드를 연결
- 한 번에 하나의 유저 스레드만 커널에 접근할 수 있기 때문에 멀티코어 시스템에서 병렬적인 수행을 할 수가 없음.

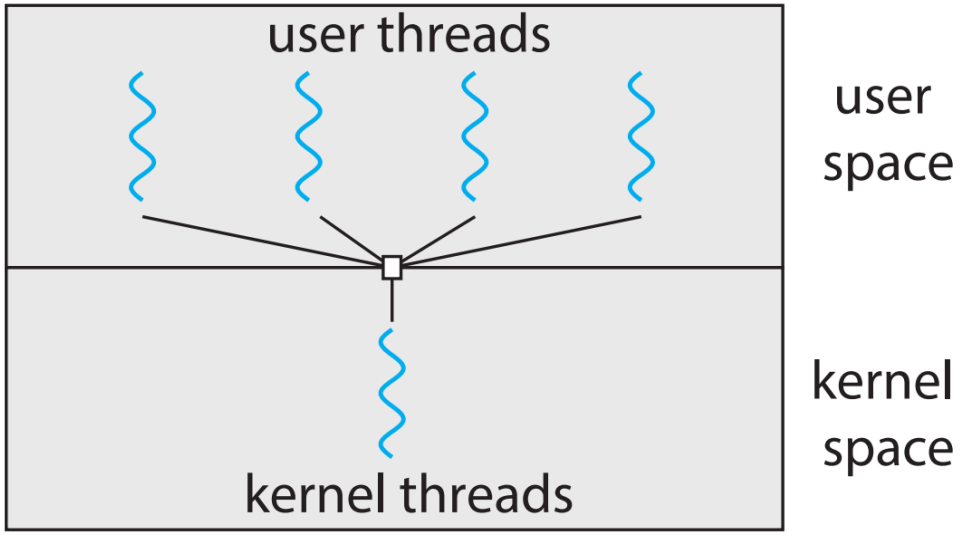


Figure 4.7 Many-to-one model.

One-to-One Model

- 하나의 유저 스레드에 하나의 커널 스레드가 연결
- 가장 일반적인 모델
- 그러나 유저 스레드를 늘리면 커널 스레드도 똑같이 늘어나는데, 커널 스레드를 생성하는 것은 오버헤드가 큰 작업이기 때문에 성능 저하가 발생할 수 있음.

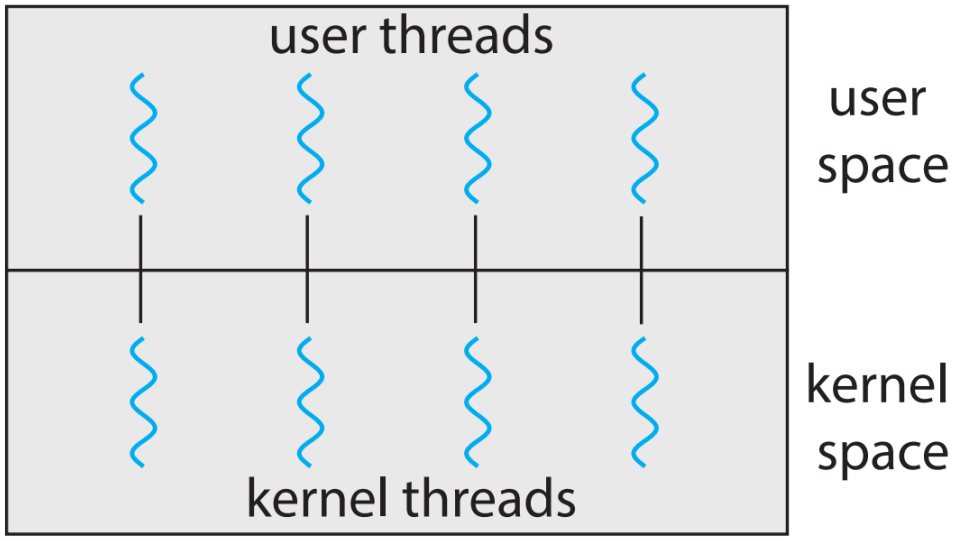


Figure 4.8 One-to-one model.

Many-to-Many Model

- 여러 개의 유저 스레드에 더 적거나 같은 수의 커널 스레드가 연결
- 성능을 유지하는데 충분한 수의 커널 스레드만 만들 수 있도록 하여 성능 저하를 막음.

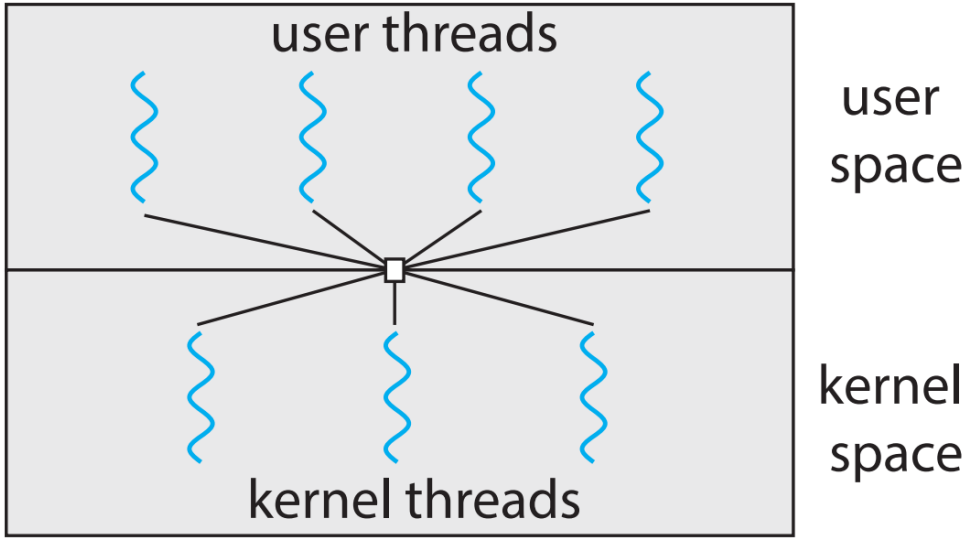


Figure 4.9 Many-to-many model.

Two-level Model

- Many-to-Many Model의 변형.
  특정 유저 스레드를 커널 스레드를 따로 제공함
- 특정 유저 스레드를 더 빠르게 처리하는 것이 가능.

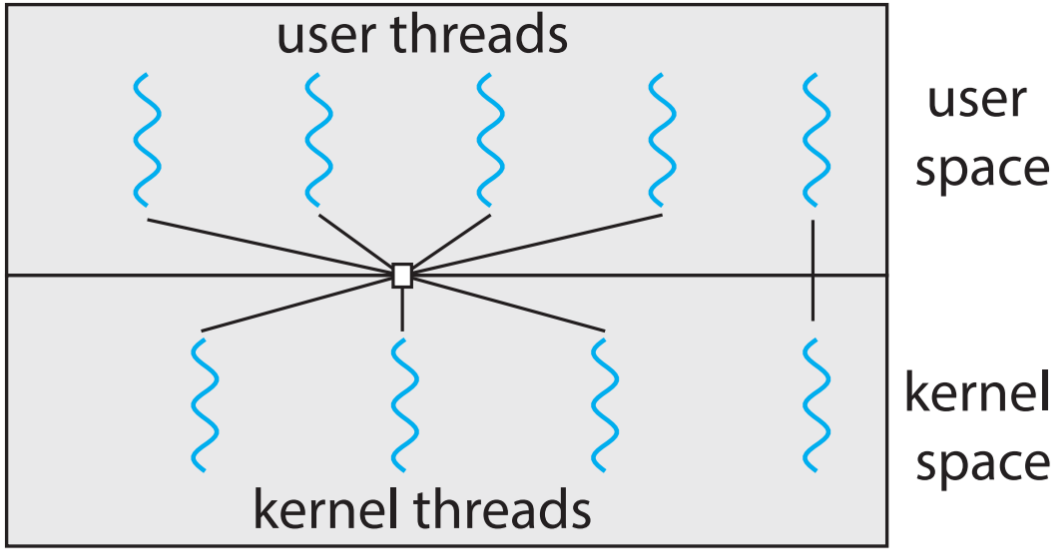
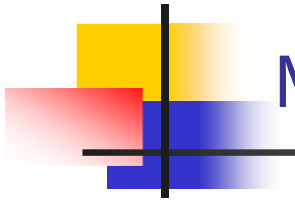


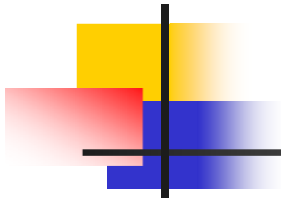
Figure 4.10 Two-level model.



Multithreading Models

- 유저 스레드와 커널 스레드 간의 관계를 설정해야함
- 일반적인 방법은 다음과 같음
  - Many-to-One Model
  - One-to-One Model
  - Many-to-Many Model
  - Two-level Model

- **이 방법들은 전부 개발자에 의해 생산, 관리되는 explicit thread**



Implicit Threading

- 직접 개발자가 스레드를 제대로 컨트롤 하기는 매우 어려움.
- Implicit Threading이란 개발자가 직접 스레드를 생산, 관리하는 것이 아니라 **운영체제에게 스레드 생산, 관리를 맡기는 것**.
  - 1) Thread Pools
  - 2) Fork-Join model
  - 3) OpenMP
  - 4) Grand Central Dispatch (GCD)

Thread Pools

- 멀티스레딩을 이용할 때, 무제한으로 스레드를 생성할 경우 시스템 리소스를 고갈시킬 수 있음.
  - 한 가지 해결책은 스레드 풀을 사용하는 것.
- Thread Pools: 시작 시 여러 스레드를 미리 만들고 풀에 배치해 둔 후, 필요 시 할당하는 것.

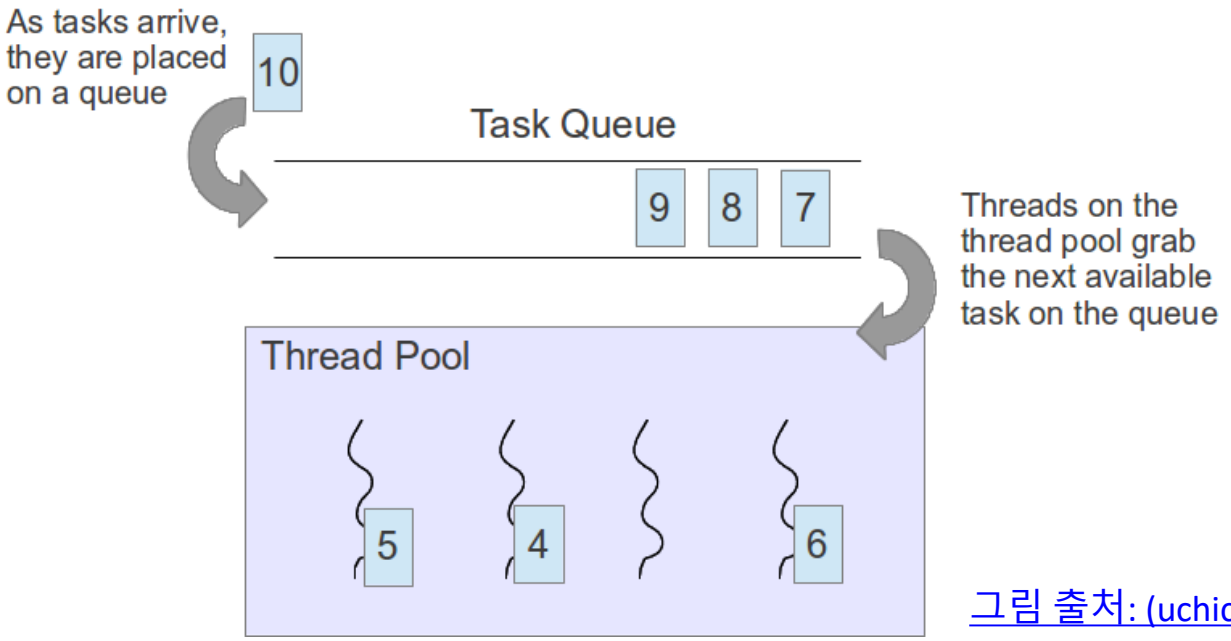
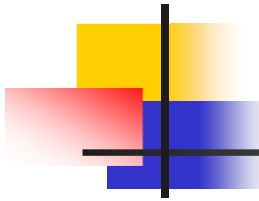


그림 출처: (uchicago.edu)



Thread Pools

- **장점:**
  - 1. 기존에 이미 생성된 스레드로 요청을 처리하는 것이, 요청마다 새로 스레드를 생성하는 것보다 빠름.
  - 2. 한 지점에 존재하는 스레드 수를 제한할 수 있음.
  - 3. 태스크를 시간 지연 후에 실행시키거나, 주기적으로 실행되도록 예약할 수 있음

- Thread Pool의 스레드 수는 시스템의 CPU 수, 메모리 양, 예상되는 동시 클라이언트 요청 수와 같은 요소를 기반으로 경험적으로 설정할 수 있음.

Fork-Join model

- 여러 개로 나누어 계산한 후 결과를 모으는 작업
  - 메인 스레드가 여러개의 자식 스레드들을 만들고(fork)
  - 자식 스레드들이 일을 마치고 다시 합쳐짐(join)

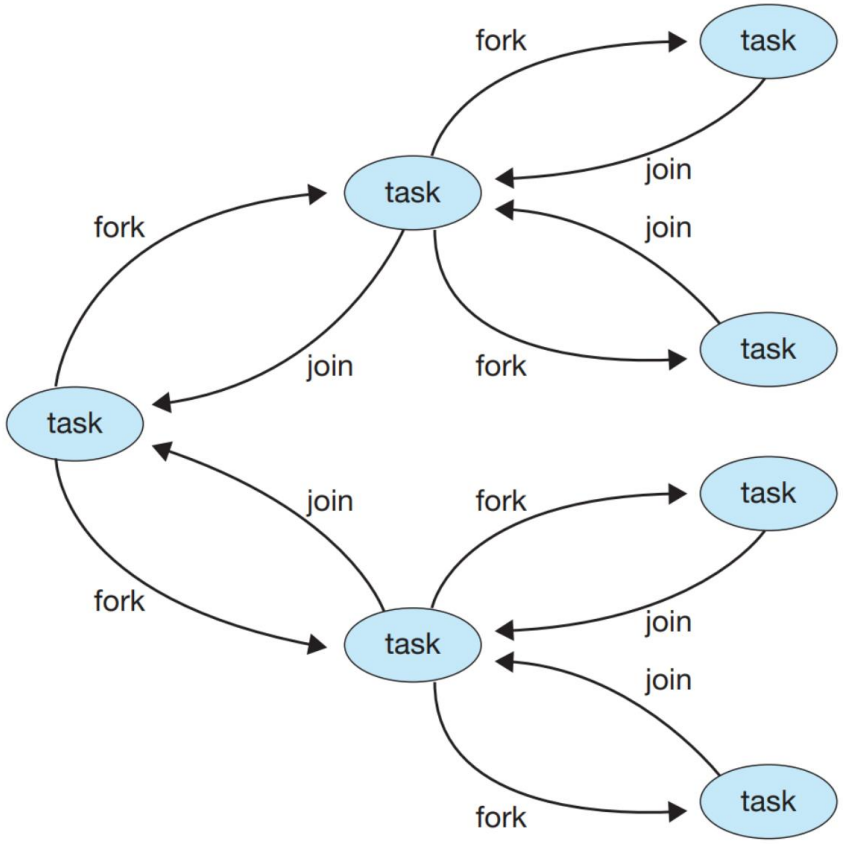
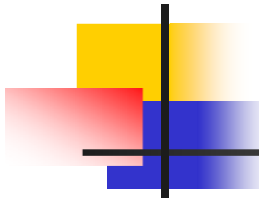


Figure 4.17 Fork-join in Java.



OpenMP

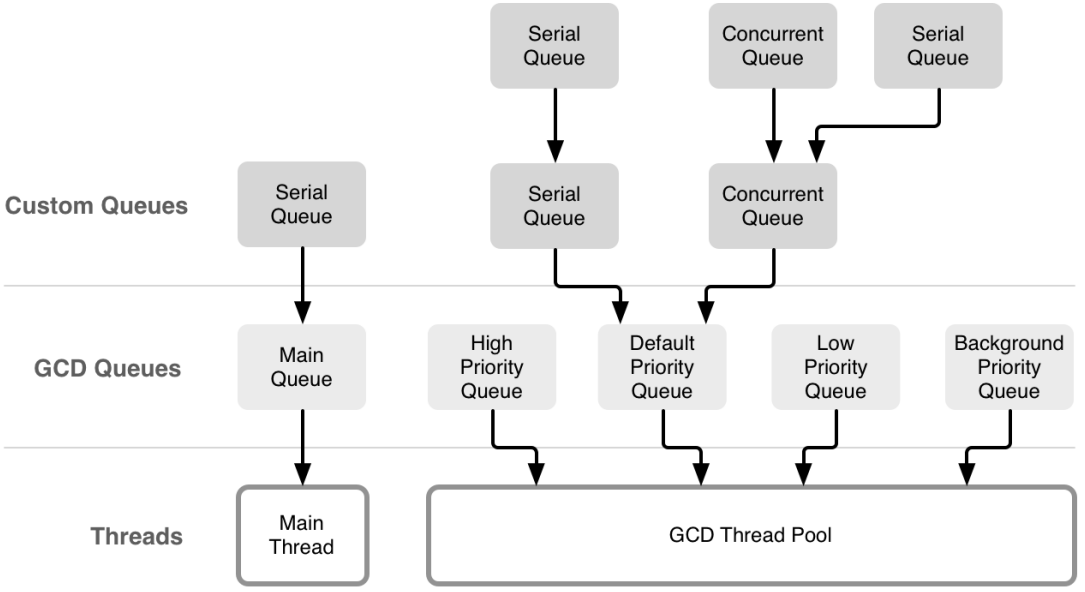
- **compiler directives(컴파일러가 스레드사용을 도와줌)**

- 스레딩을 language 수준에서 미리 기술하는 방법.
- Parallel regions를 미리 정하는 형태로 동작
- 주로 #progma 코드를 이용하여 parallel regions를 정의해줌

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
   c[i] = a[i] + b[i];
}
```

Grand Central Dispatch(GCD)

- Parallel regions를 미리 정하는 형태로 동작
  - 1. 정한 영역을 dispatch queue에 넣음
  - 2. 가용 스레드가 있다면 그 가용 스레드에 할당됨
- 다양한 dispatch queue가 제공됨



- Mac OS X 와 IOS에서 주로 사용

그림 출처 : https://www.objc.io/

Chapter 4
Finish