Chapter 04

스레드와 병행성

Contents

01.

Overview

3장에서 소개한 프로세스 모델은 한 프로세스가 하나의 제어 스레드로 프로그램을 실행한다고 가정했다. 그러나 거의 모든 현대 운영체제는 한 프로세스가 다중 스레드를 포함하는 특성을 제공한다.
다중 CPU를 제공하는 최신 다중 코어 시스템에서 스레드 사용을 통한 병렬 처리의 기회를 식별하는 것이 점차 중요해진다.

Overview

스레드

- CPU 이용의 기본 단위
- 스레드ID, 프로그램 카운터(PC), 레지스터 집합, 스택으로 구성되어있다.
- 같은 프로세스에 속한 다른 스레드와 코드, 데이터 섹션 그리고 열린 파일이나 신호와 같은 운영체제 자원들을 공유한다.
- 싱글 스레드 프로세스와 멀티 스레드 프로세스로 나뉜다.

Overview

Motivation

- 현대 컴퓨터와 모바일 장치에서 실행되는 대부분의 소프트웨어 애플리케이션은 멀티 스레드 방식으로 구현된다. 애플리케이션은 일반적으로 여러 제어 스레드를 가진 별도의 프로세스로 구성된다.
- 멀티코어 시스템에서는 이러한 APP이 여러 CPU 집약접 작업을 병렬로 수행 가능하다.
- 예를 들어, 웹 서버는 여러 클라이언트 요청을 동시에 처리해야 하며, 전통적인 단일 스레드 프로세스에서는 한 번에 하나의 클라이언트만 서비스할 수 있다.
- 웹 서버는 요청을 수신할 때 새로운 스레드를 생성하여 요청을 처리하고, 추가 요청을 계속 수신할 수 있다.

Overview

Motivation

- 대부분의 운영 체제 커널도 멀티스레드 방식으로 설계되어 있다.
- 예를 들어, 리눅스 시스템 부팅 시 여러 커널 스레드가 생성되어 장치 관리, 메모리 관리, 인터럽트 처리 등의 작업을 수행한다.
- 마지막으로 많은 APP이 기본 정렬, 트리 및 그래프 알고리즘과 같은 작업에서 여러 스레드를 활용할 수 있으며, 데이터 마이닝, 그래픽, 인공지능과 같은 CPU 집약적 문제(CPU-intensive problems)를 해결하기 위해 현태 멀티 코어 시스템의 병렬 처리 능력을 활용할 수 있다.

Overview

Benefits

- 멀티 스레드 프로그래밍의 이점은 네 가지 주요 범주로 나눌 수 있다.

1. Responsiveness(응답성)
   a. 대화형 APP을 멀티스레딩하면 프로그램의 일부가 차단되거나 긴 작업을 수행하더라도 프로그램을 계속 실행할 수 있으므로 사용자에 대한 반응성이 높아진다.
   b. 이러한 품질은 특히 사용자 인터페이스를 설계하는 데 유용한다.
2. Resource sharing(자원 공유)
   a. 프로세스는 공유 메모리 및 메시지 전달과 같은 기술을 통해서 만 리소스를 공유할 수 있다.
   b. 이러한 기술은 프로그래머가 명시적으로 배열해야 한다.
   c. 그러나 스레드는 기본적으로 자신이 속한 프로세스의 메모리와 리소스를 공유한다.
   d. 코드와 데이터를 공유하는 이점은 APP이 동일한 주소 공간 내에서 여러 다른 활동 스레드를 가질 수 있다는 것이다.

Overview

Benefits

1. Economy(경제성)
   a. 프로세스 생성을 위해 메모리와 리소스를 할당하는 것은 비용이 많이 든다.
   b. 스레드는 자신이 속한 프로세스의 리소스를 공유하기 때문에 스레드를 생성하고 Context를 전환하는 것이 더 경제적이다.
   c. 경험적으로 오버헤드의 차이를 측정하는 것은 어려울 수 있지만 일반적으로 스레드 생성은 프로세스 생성보다 시간과 메모리를 덜 소모한다. 또한 Context 전환은 일반적으로 프로세스 간보다 스레드 간에서 더 빠르다.
2. Scalability(확장성)
   a. 멀티 스레딩의 이점은 스레드가 다른 처리 코어에서 병렬로 실행될 수 있는 멀티프로세서 아키텍처에서 더 클 수 있다.
   b. 단일 스레드 프로세스는 사용 가능한 프로세서 수에 관계없이 단일 프로세서에서만 실행될 수 있다.

02.

Multicore Programming

초기 컴퓨터 설계에서 더 높은 컴퓨팅 성능에 대한 필요성으로 인해 단일 CPU 시스템이 다중 CPU 시스템으로 발전하였다.

Multicore Programming

OverView

- 멀티코어 시스템
  - 이후 시스템 설계에서 단일 처리 칩에 여러 컴퓨팅 코어를 배치하는 경향이 나타났다. 각 코어는 운영체제에 대해 별도의 CPU 로 인식된다
- 멀티 스레드 프로그래밍
  - 멀티코어 시스템에서 여러 컴퓨팅 코어를 효율적으로 사용하고 동시성을 개선하기 위한 메커니즘을 제공한다.

Multicore Programming

concurrency & parallelism

- 병행성(concurrency) : 모든 작업이 진행되도록 허용하여 여러 작업을 지원하는 시스템이다.
  - 동시에 진행할 수 있도록 허용하는 것이다.
- 병렬성(parallelism) : 여러 작업을 동시에 수행할 수 있는 시스템이다.
  - 여러 작업이 동시에 수행되는 상황을 의미한다.

Multicore Programming

concurrency & parallelism

단일 코어 시스템

단일 코어 시스템에서는 병행성이 스레드 실행이 시간에 따라 인터리브되는 것을 의미한다.

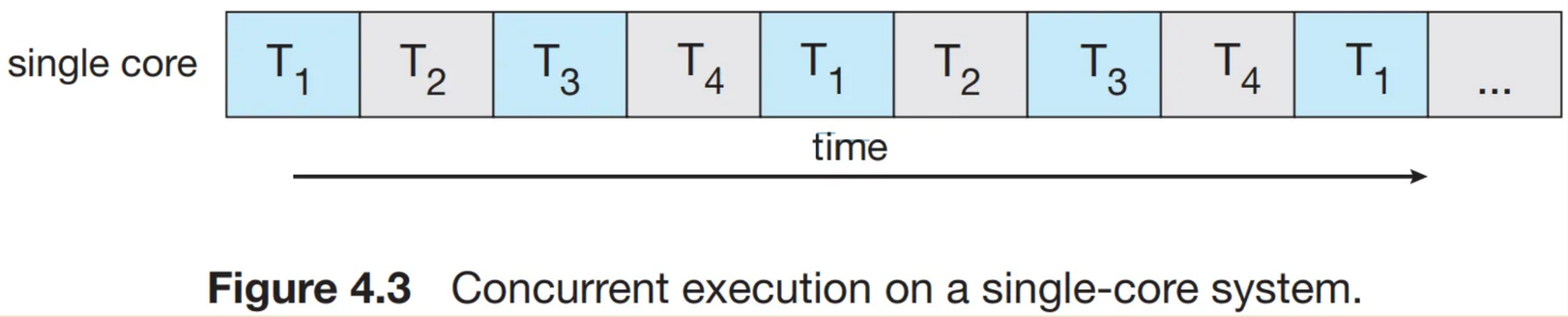


Figure 4.3 Concurrent execution on a single-core system.

Multicore Programming

concurrency & parallelism

멀티 코어 시스템

여러 코어가 있는 시스템에서는 일부 스레드가 병렬로 실행될 수 있다. 이는 시스템이 각 코어에 별도의 스레드를 할당할 수 있기 때문이다.

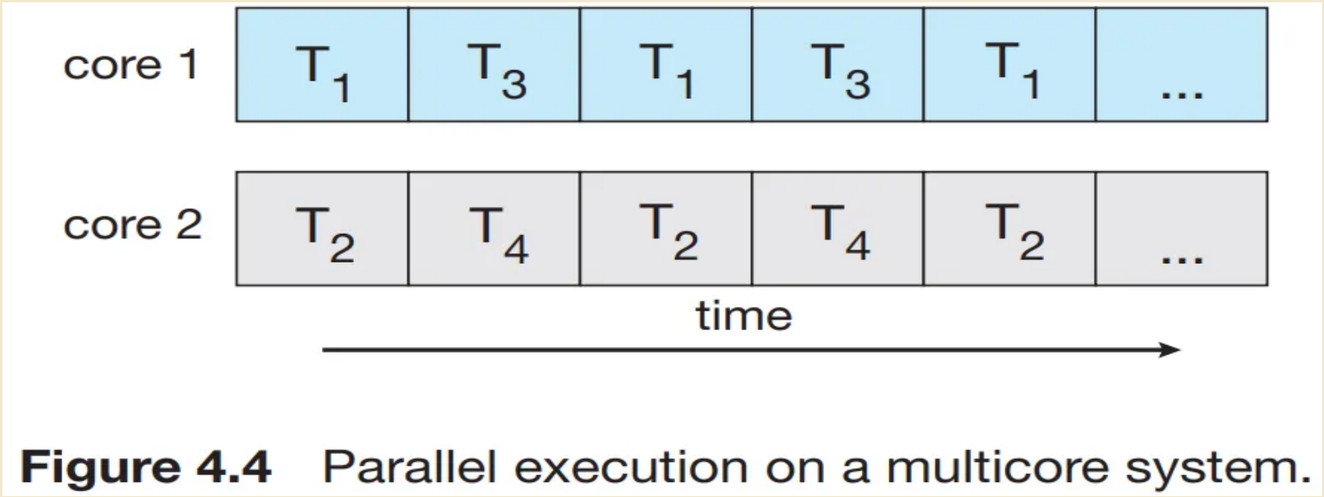


Figure 4.4 Parallel execution on a multicore system.

Multicore Programming

concurrency & parallelism

- **병행성(concurrency)** : 모든 작업이 진행되도록 허용하여 여러 작업을 지원하는 시스템이다.
  - 동시에 진행할 수 있도록 허용하는 것이다.
- **병렬성(parallelism)** : 여러 작업을 동시에 수행할 수 있는 시스템이다.
  - 여러 작업이 동시에 수행되는 상황을 의미한다.

Multicore Programming

Programming Challenges

- **시스템 설계자와 프로그래머의 압력**
  - 멀티 코어 시스템에 대한 추세는 시스템 설계자와 APP 프로그래머에게 여러 컴퓨팅 코어를 더 잘 활용하라는 압력을 가하고 있다.

1. **운영 체제 설계자**
   a. 병렬 실행을 허용하기 위해 여러 처리 코어를 사용하는 스케줄링 알고리즘을 작성해야 한다.
2. **애플리케이션 프로그래머**
   a. 기존 프로그램을 수정하고 멀티스레드인 새 프로그램을 설계하는 과제가 있다.

Five challenges

1. **작업 식별**
   a. APP을 검사하여 별도의 동시 작업으로 나눌 수 있는 영역을 찾는 과정이다. 이상적으로 작업은 서로 독립적이어야 하며, 개별 코어에서 병렬로 실행될 수 있어야 한다.
2. **균형**
   a. 병렬로 실행할 수 있는 작업을 식별하는 동안, 작업이 동일한 가치의 동일한 작업을 수행하도록 해야 한다.
   b. 특정 작업이 다른 작업만큼 전체 프로세스에 기여하지 못할 수 있다.
3. **데이터 분할**
   a. APP이 별도의 작업으로 나뉘는 것처럼, 작업에서 액세스하고 조작하는 데이터도 나누어야 한다.
4. **데이터 종속성**
   a. 작업에서 액세스하는 데이터 간의 종속성을 검사해야 한다.
   b. 한 작업이 다른 작업의 데이터에 의존하는 경우 프로그래머는 작업 실행이 데이터 종속성을 수용하도록 동기화 해야 한다.
5. **시험 및 디버깅**
   a. 프로그램이 다중 코어에서 병렬로 실행될 때, 다양한 실행 경로가 존재할 수 있다.
   b. 그런 병행 프로그램을 시험하고 디버깅하는 것은 상대적으로 단일 스레드의 경우보다 어렵다.

Types of Parallelism

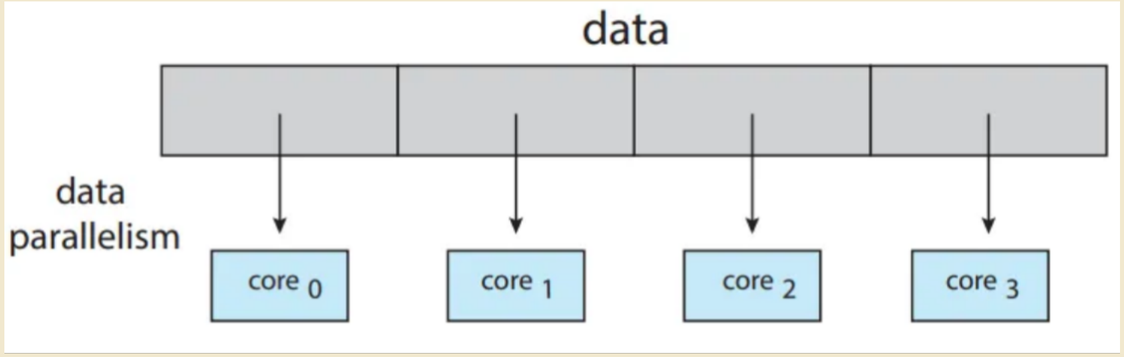
- 일반적으로 병렬 처리에는 데이터 병렬 처리와 테스크 병렬 처리의 두 가지 유형이 있다.

1. **데이터 병렬 처리**
   a. 동일한 데이터의 하위 집합을 여러 컴퓨팅 코어에 분산하고 각 코어에서 동일한 작업을 수행하는 방식
   b. 크기가 N인 배열의 내용을 더하는 경우
      i. 단일 코어 시스템에서는 한 스레드가 모든 요소를 더한다.
      ii. 듀얼 코어 시스템에서는 코어 0에서 실행되는 스레드 A가 배열의 절반을 더하고, 코어 1에서 실행되는 스레드 B가 나머지 절반을 더할 수 있다.
2. **테스크 병렬 처리**
   a. 데이터가 아닌 테스크(스레드)를 여러 컴퓨팅 코어에 분배하는 방식, 각 스레드는 고유한 작업을 수행한다.
   b. 두 개의 스레드가 각각 고유한 통계적 테스크를 수행하는 경우

- **데이터 병렬성과 테스크 병렬성은 상호 베타적이지 않으며, 실제로 APP은 이 두 가지 전략의 하이브리드를 사용할 수 있다.**

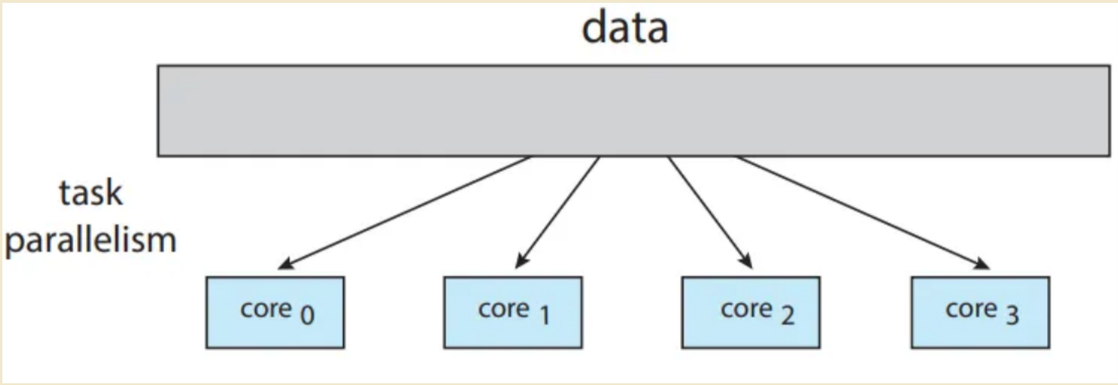
Multicore Programming

concurrency & parallelism



데이터 병렬 처리

여러 코어에 분산하고 각 코어에서 동일한 작업을 수행하는 것을 확인할 수 있다.



테스크 병렬 처리

테스크(스레드)를 여러 코어에 분배하는 것을 확인할 수 있다.

03.

Multithreading Models

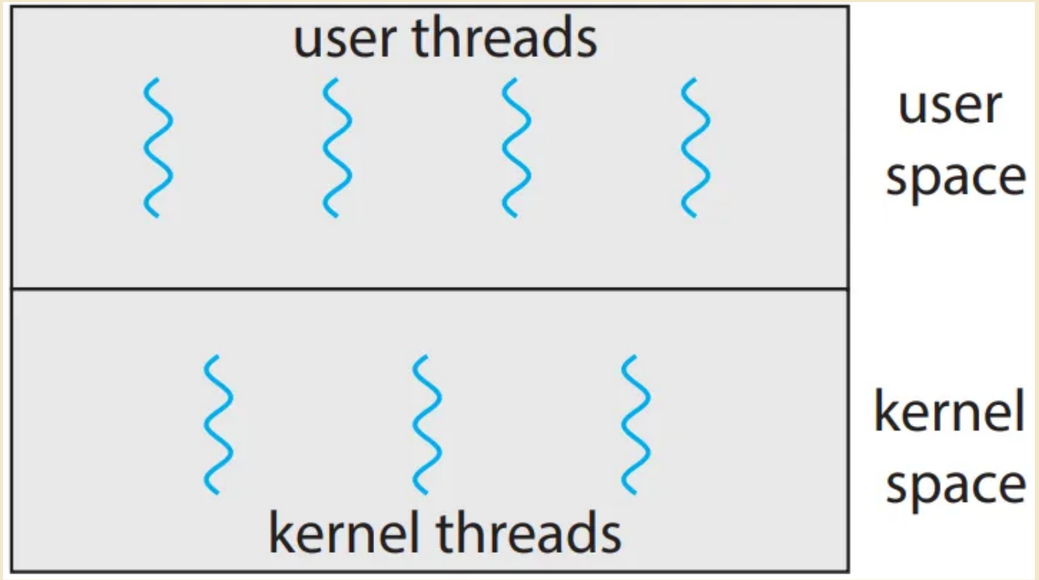
스레드는 컴퓨터 프로그램 내에서 실행되는 경량 프로세스이다.
스레드는 병렬 처리를 가능하게 하여 여러 작업을 동시에 수행할 수 있도록 한다.

Multithreading Models

OverView

- 스레드에 대한 지원은 두 가지 방식으로 제공된다.
- **사용자 스레드** : 사용자 수준에서 관리된다.
- **커널 스레드** : 운영 체제의 커널에서 직접 지원하고 관리된다.
- 현대 운영 체제(Windows, Linux, MacOS)는 커널 스레드를 지원한다.

Multithreading Models



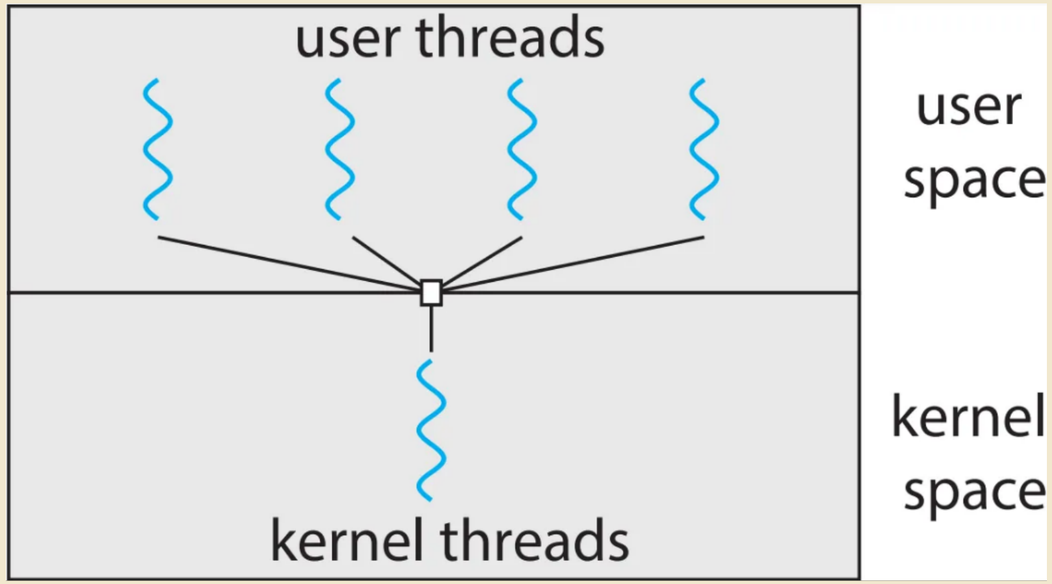
사용자 스레드와 커널 스레드 사이에는 관계가 존재해야 한다.

이 관계를 설정하는 세 가지 일반적인 모델이 존재한다.

1. 다대일 모델
2. 일대일 모델
3. 다대다 모델

Multithreading Models

Many-to-One Model

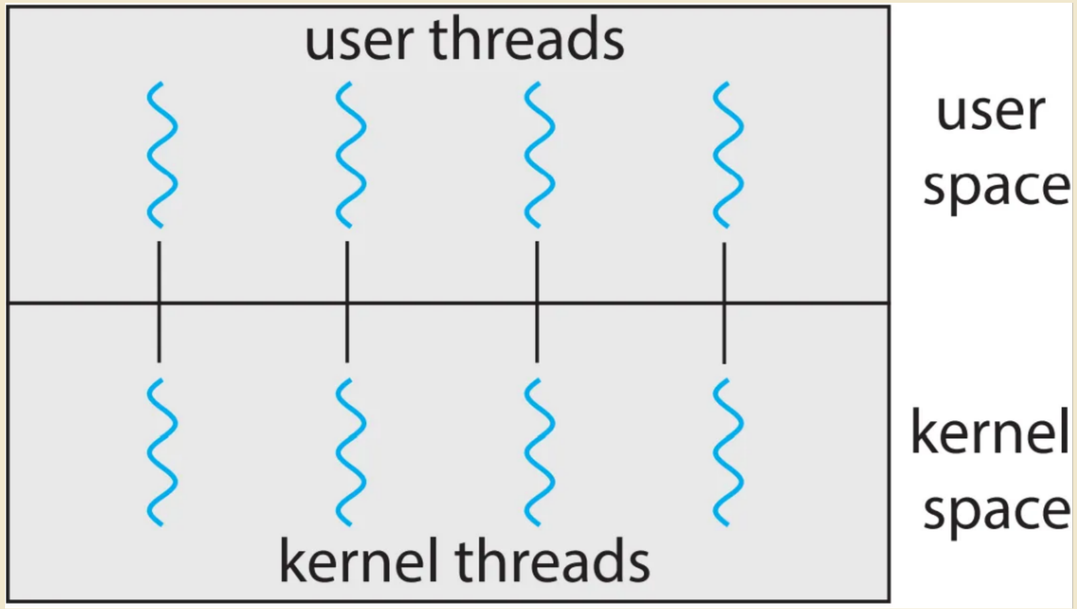


다대일 모델은 여러 사용자 수준 스레드를 하나의 커널 스레드에 매핑하는 방식이다.

1. 스레드 관리가 사용자 공간에서 스레드 라이브러리에 의해 수행된다.
2. 효율적이지만, 스레드가 차단 시스템 호출을 하면 전체 프로세스가 차단된다.
3. 멀티 코어 시스템에서 병렬 실행이 불가능하다.

Multithreading Models

One-to-One Model



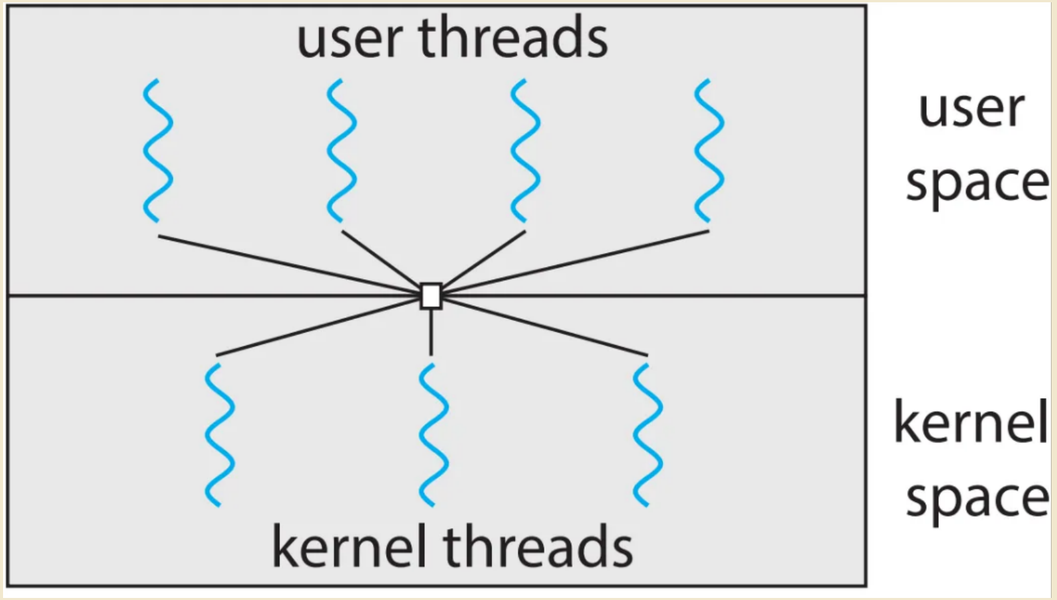
일대일 모델은 각 사용자 스레드를 커널 스레드에 매핑하는 방식이다.

- 장점
  - 스레드가 차단 시스템 호출을 할 때 다른 스레드가 실행될 수 있어 더 많은 동시성을 제공한다.
  - 여러 스레드가 멀티 프로세서에서 병렬 실행될 수 있다.
- 단점
  - 사용자 스레드를 만들기 위해 커널 스레드를 생성해야 하며, 많은 커널 스레드는 시스템 성능에 부담을 줄 수 있다.

Linux와 Windows 운영 체제는 이 모델을 구현하고 있다.

Multithreading Models

Many-to-Many Model

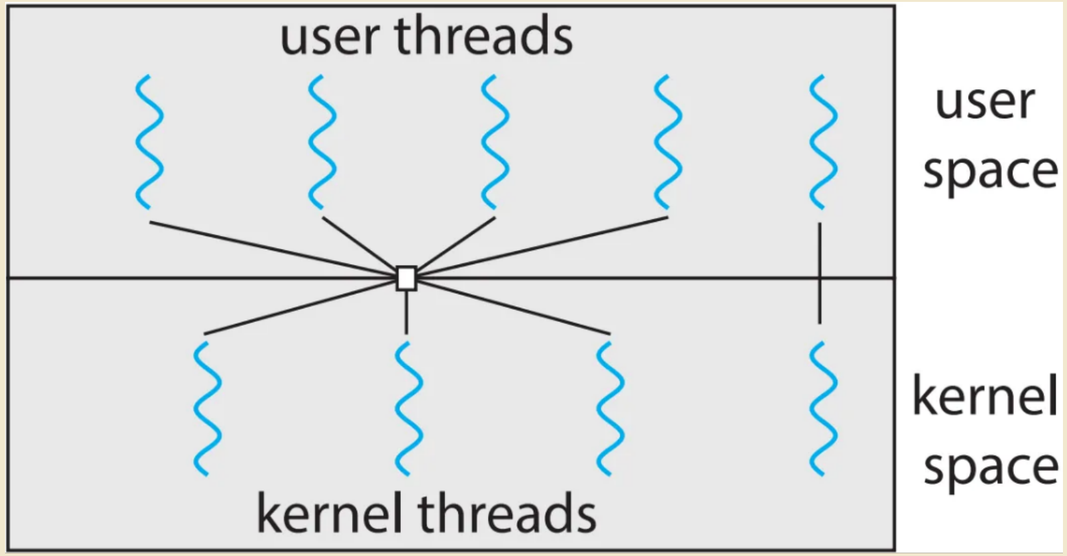


다대다 모델은 많은 사용자 수준 스레드를 더 적거나 같은 수의 커널 스레드로 멀티 플렉싱하는 방식이다.

- 커널 스레드의 수는 특정 APP이나 머신에 따라 다를 수 있다.
- 개발자는 원하는 만큼 많은 사용자 스레드를 만들 수 있으며, 커널 스레드는 멀티프로세서에서 병렬 실행될 수 있다.
- 스레드가 봉쇄형 시스템 콜을 발생시켰을 때, 커널이 다른 스레드의 수행을 스케줄링할 수 있다.

Multithreading Models

Two-level model



- 다대다 모델의 변형은 사용자 수준 스레드를 커널 스레드에 바인딩할 수 있는 방식이다.
- 이 변형을 2단계 모델이라고 부른다.
- 이 다대다 모델은 유연성이 뛰어나지만, 구현하기 어렵다는 단점이 있다.

Multithreading Models

- 대부분의 운영 체제는 이제 일대일 모델을 사용한다.
- 커널 스레드 수를 제한하는 것이 덜 중요해진 이유는 처리 코어의 수가 증가하고 있기 때문이다.
- 일부 현대적 동시성 라이브러리는 개발자가 작업을 식별한 후 다대다 모델을 사용하여 스레드에 매핑하도록 한다.

04.

Thread Libraries

스레드 라이브러리는 프로그래머에게 스레드를 만들고 관리하기 위한 API를 제공하는 것이다.

Thread Libraries

OverView

- 스레드 라이브러리는 커널 지원 없이 사용자 공간에서 완전히 제공되거나, 운영 체제에서 직접 지원하는 커널 수준 라이브러리로 구현될 수 있다.

- 주요 기능
  - 스레드 생성
  - 스레드 관리
  - 스레드 동기화

- 스레드 라이브러리를 사용하면 병렬 처리를 통해 프로그램의 성능을 향상 시킬 수 있다.

Thread Libraries

구현 방법

1. **사용자 공간에서의 구현**
    a. 모든 코드와 데이터 구조가 사용자 공간에 존재한다.
    b. 시스템 호출이 아닌 사용자 공간에서 로컬 함수 호출이 발생한다.
2. **커널 수준 라이브러리**
    a. 라이브러리의 코드와 데이터 구조가 커널 공간에 존재한다.
    b. API에서 함수를 호출하면 커널에 대한 시스템 호출이 발생한다.

Thread Libraries

- POSIX Pthreads
  - 사용자 수준 또는 커널 수준 라이브러리로 제공될 수 있다.
- Windows
  - Windows 시스템에서 사용할 수 있는 커널 수준 라이브러리이다.
- Java
  - Java 프로그램에서 직접 스레드를 만들고 관리할 수 있는 API를 제공한다.
  - Java 스레드 API는 JVM이 호스트 운영 체제 위에서 실행되기 때문에, 일반적으로 호스트 시스템에서 사용할 수 있는 스레드 라이브러리를 사용하여 구현된다.

Thread Libraries

스레드 생성 전략

- 비동기 스레딩
  - 부모가 자식 스레드를 생성한 후, 부모가 실행을 재개하므로 부모와 자식은 동시에 독립적으로 실행된다.
  - 일반적으로 데이터 공유가 거의 없다.

- 동기 스레딩
  - 부모 스레드가 자식을 만든 후, 모든 자식이 종료될 때까지 기다려야 재개할 수 있다.
  - 부모는 자식 스레드가 작업을 완료할 때까지 대기하며, 자식 스레드가 종료되면 부모와 결합된다.

Thread Libraries

Pthreads

- Pthreads는 스레드 생성 및 동기화를 위한 API를 정의하는 POSIX 표준(IEEE 1003.1c)이다.
- 운영 체제 설계자는 원하는 대로 사양을 구현할 수 있다.
- 많은 시스템이 Pthreads 사양을 구현하고 있으며, 대부분은 Linux 및 MacOS를 포함한 UNIX 유형 시스템이다.
- 기본 구조
  - pthread.h 헤더 파일을 포함해야 한다.
  - pthread_t tid 문을 사용하여 생성할 스레드의 식별자를 선언한다.
  - pthread_attr_t attr 선언을 통해 스레드의 속성을 나타낸다.
  - pthread_create() 함수 호출로 별도의 스레드를 생성한다.

Thread Libraries

Windows Threads

- Windows 스레드 라이브러리를 사용하여 스레드를 만드는 기술은 Pthreads 기술과 유사하다.
- Windows API를 사용할 때는 windows.h 헤더 파일을 포함해야 한다.
- 특징
  - 전역적으로 선언된 데이터는 개별 스레드에서 공유된다.
  - CreateThread() 함수를 사용하여 스레드를 생성한다.
  - 스레드의 속성 집합은 보안 정보, 스택 크기 등을 포함한다.
- 부모 스레드는 WaitForSingleObject() 함수를 사용하여 자식 스레드가 완료될 때까지 기다린다.

Thread Libraries

Java Threads

- Java 스레드는 프로그램 실행의 기본 모델이며, Java 언어와 해당 API는 스레드 생성 및 관리를 위한 풍부한 특성을 제공한다.

- 기본 구조
  - 모든 Java 프로그램은 최소한 하나의 제어 스레드로 구성된다.
  - 스레드를 명시적으로 생성하는 두 가지 기술이 있다.
    - Thread 클래스에서 파생된 새 클래스 생성 : run() 메소드를 재정의 한다.
    - Runnable 인터페이스 구현 : public void run() 의 서명을 가진 단일 추상 메소드를 정의한다.

Thread Libraries

Java Lambda 표현식

- Java 1.8 버전부터 Lambda 표현식을 도입하여 스레드를 작성하는 데 훨씬 더 명확한 구문을 허용한다.
- Runnable을 구현하는 별도의 클래스를 정의하는 대신 Lambda 표현식을 대신 사용할 수 있다.

```
Runnable task = () -> {
    System.out.println("I am a Thread.");
};
Thread worker = new Thread(task);
worker.start();
```

Thread Libraries

Java Lambda 표현식

- Java에서 스레드를 생성하려면 Thread 객체를 생성하고, Runnable을 구현하는 클래스의 인스턴스를 전달한 후 start() 메소드를 호출해야 한다.

```
Thread worker = new Thread(new Task());
worker.start();
```

Thread Libraries

스레드 완료 대기

- Pthreads와 Windows 라이브러리의 부모 스레드는 각각 pthread_join() 과 Wait-ForSingleObject() 를 사용하여 자식 스레드가 완료될 때까지 기다린다.
- Java의 join() 메소드는 비슷한 기능을 제공하며, 다음과 같은 구조를 갖는다.
- 부모가 여러 스레드가 완료될 때까지 기다려야 하는 경우, join() 메소드는 for 루프에 포함될 수 있다.

```
try {
    worker.join();
}
catch (InterruptedException ie) { }
```

Thread Libraries

Java Executor Framework

- Java의 스레드 생성 : Java는 다양한 접근 방식을 통해 스레드 생성을 지원해왔다.
- 1.5 버전의 변화 : Java 1.5부터는 개발자에게 스레드 생성 및 통신에 대한 더 큰 제어력을 제공하는 여러 동시성 기능이 도입되었다.
- java.util.concurrent 패키지 : 이러한 새로운 도구들은 java.util.concurrent 패키지에서 사용할 수 있다.
- Executor 인터페이스 : 스레드 객체를 명시적으로 생성하는 대신, 스레드 생성은 Executor 인터페이스를 중심으로 구성된다.

Executor 인터페이스 소개

```
public interface Executor {
    void execute(Runnable command);
}
```

- **execute() 메소드**
  - 이 인터페이스를 구현하는 클래스는 execute() 메소드를 정의해야 하며, 이 메소드는 Runnable 객체를 전달 받는다.
- **Thread 객체 생성 대체**
  - Java 개발자는 별도의 Thread 객체를 생성하고 해당 start() 메소드를 호출하는 대신 Executor를 사용할 수 있다.

- **Executor 사용 예시** : Executor는 다음과 같이 사용된다.

```
Executor service = new Executor;
service.execute(new Task());
```

Thread Libraries

Executor 사용법

- Executor 프레임워크
  - Executor 프레임워크는 프로듀서-소비자 모델을 기반으로 한다.

- Runnable 인터페이스
  - Runnable 인터페이스를 구현하는 작업이 생성되고, 이러한 작업을 실행하는 스레드가 이를 소비한다.

- 장점
  - 이 접근 방식의 장점은 스레드 생성을 실행에서 분리할 뿐만 아니라, 동시 작업 간 통신을 위한 메커니즘을 제공한다는 것이다.

- 데이터 공유
  - 동일한 프로세스에 속하는 스레드 간의 데이터 공유는 전역적으로 선언된 공유 데이터를 통해 쉽게 발생한다.

프로듀서 - 소비자 모델

- Java의 전역 데이터 개념
  - 순수한 객체 지향 언어인 Java에는 이러한 전역 데이터 개념이 없다.
- Runnable의 매개변수 전달
  - Runnable을 구현하는 클래스에 매개변수를 전달할 수 있지만, Java 스레드는 결과를 반환할 수 없다.
- Callable 인터페이스의 도입
  - 이러한 요구 사항을 해결하기 위해 java.util.concurrent 패키지는 추가로 Callable 인터페이스를 정의한다.
- Callable의 기능
  - Callable 인터페이스는 결과를 반환할 수 있다는 점을 제외하면 Runnable과 유사하게 동작한다.

Callable 인터페이스의 필요성

- **Future 객체**
  - Callable 작업에서 반환된 결과는 Future 객체라고 하며, 결과는 Future 인터페이스에 정의된 get() 메소드에서 검색할 수 있다.

- **합산 프로그램 예시**
  - p193. 그림 4.14에 표시된 프로그램은 이러한 Java 기능을 사용하는 합산 프로그램을 보여준다.

- **Summation 클래스**
  - Summation 클래스는 call()메소드를 지정하는 Callable 인터페이스를 구현한다.

- **스레드에서의 실행**
  - 이 call() 메소드의 코드는 별도의 스레드에서 실행된다

Future 객체와 결과 처리

- **ExecutorService 사용**
  - 이 코드를 실행하려면 ExecutorService 유형의 newSingleThreadExecutor 객체를 만들고 submit() 메소드를 사용하여 Callable 작업에 전달한다.

- **execute() 와 submit() 의 차이**
  - execute() 메소드와 submit() 메소드의 주요 차이점은 전자는 결과를 반환하지 않는 반면 후자는 결과를 Future로 반환한다는 것이다.

- **결과 대기**
  - 호출 가능한 작업을 스레드에 제출하면 반환하는 Future 객체의 get() 메소드를 호출하여 결과를 기다린다.

- **스레드 종료 대기**
  - 결과를 검색하기 전에 스레드가 종료될 때까지 기다리는 대신, 부모는 결과가 사용 가능해질 때까지만 기다린다.

Thread Libraries

스레드 생성 모델의 복잡성

- 복잡성 인식 : 처음에는 이 스레드 생성 모델이 단순히 스레드를 생성하고 종료시에 조인하는 것보다 더 복잡해 보인다는 것을 쉽게 알 수 있다.
- 복잡성의 이점 : 그러나 이렇게 적당히 복잡하게 만드는 것이 이점이 있다.
- 결과 반환 가능성 : Callable과 Future를 사용하면 스레드가 결과를 반환할 수 있다.
- 스레드 생성과 결과 분리 : 이 접근 방식은 스레드 생성과 스레드가 생성하는 결과를 분리한다.

Thread Libraries

결과 검색과 스레드 관리

- 결과 사용 가능 대기
  - 결과를 검색하기 전에 스레드가 종료될 때까지 기다리는 대신, 부모는 결과가 사용 가능해질 때까지만 기다린다.
- 강력한 도구
  - 이 프레임워크는 다른 기능과 결합하여 많은 수의 스레드를 관리하기 위한 강력한 도구를 만들 수 있다.
- 스레드 관리의 중요성
  - 많은 수의 스레드를 효과적으로 관리하는 것은 성능 최적화 자원 관리에 있어 매우 중요하다.

Thank You