고려대학교
KOREA UNIVERSITY

KU-The Future

Operating Systems (10th Ed., by A. Silberschatz)

Chapter 4 Threads & Concurrency

Heonchang Yu
Distributed and Cloud Computing Lab.

Contents

Objectives

– Identify the basic components of a thread, and contrast threads and processes.
– Describe the major benefits and significant challenges of designing multithreaded processes.
– Illustrate different approaches to implicit threading, including thread pools, fork-join, and Grand Central Dispatch.
– Describe how the Windows and Linux operating systems represent threads.
– Design multithreaded applications using the Pthreads, Java, and Windows threading APIs.

Overview

- Thread – a light-weight process 서로 다른 일들이 독립적으로 수행되는 것을 의미한다 동시다발적으로 수행
  - ✓ A basic unit of CPU utilization – It comprises a thread ID, a program counter (PC), a register set, and a stack.
  - ✓ It shares with other threads belonging to the same process its code section, data section, and other operating-system resources
  - ✓ An application is implemented as a separate process with several threads of control.
    - ❖ An application that creates photo thumbnails from a collection of images may use a separate thread to generate a thumbnail from each separate image.
    - ❖ A web browser might have one thread display images or text while another thread retrieves data from the network.
    - ❖ A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Single and Multithreaded Processes

프로세스들이 독립적으로 나뉘어 있으면 무거워질 수 밖에 없음
스레드는 아랫단에서 기본적인 것들을 공유하기 때문에 가볍다

프로세스에 공유하기 기능이 포함되는 것
스레드를 쓴다는 것은 공유할 수 있는 것은 최대한 공유하겠다는 의미다.

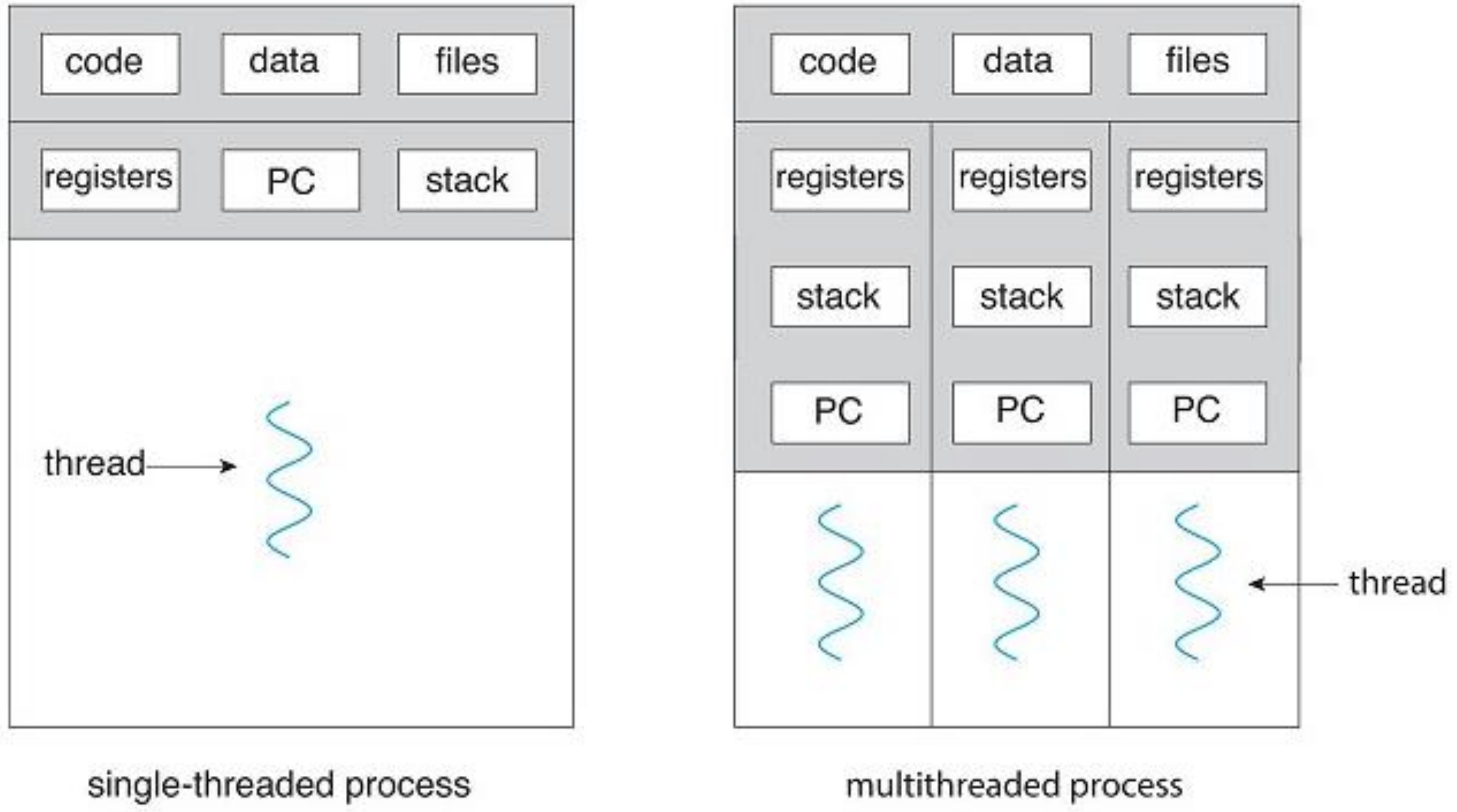


Figure 4.1 Single-threaded and multithreaded processes.

Overview

- ✓ A single application may be required to perform several similar tasks.
    - ❖ A web server accepts client requests for Web pages, images, sound, and so forth.
- ✓ One solution is to have the server run as a single process that accepts requests. When the server receives a request, it creates a separate process to service that request. – Process creation is time consuming and resource intensive.
- ✓ If the web-server process is multithreaded, the server will create a separate thread that listens for client requests.
- ✓ When a request is made, rather than creating another process, the server will create a new thread to service the request and resume listening for additional requests

Multithreaded Server Architecture

- Most operating system kernels are typically multithreaded.
  - ✓ As an example, during system boot time on Linux systems, several kernel threads are created. Each thread performs a specific task, such as managing devices, memory management, or interrupt handling.

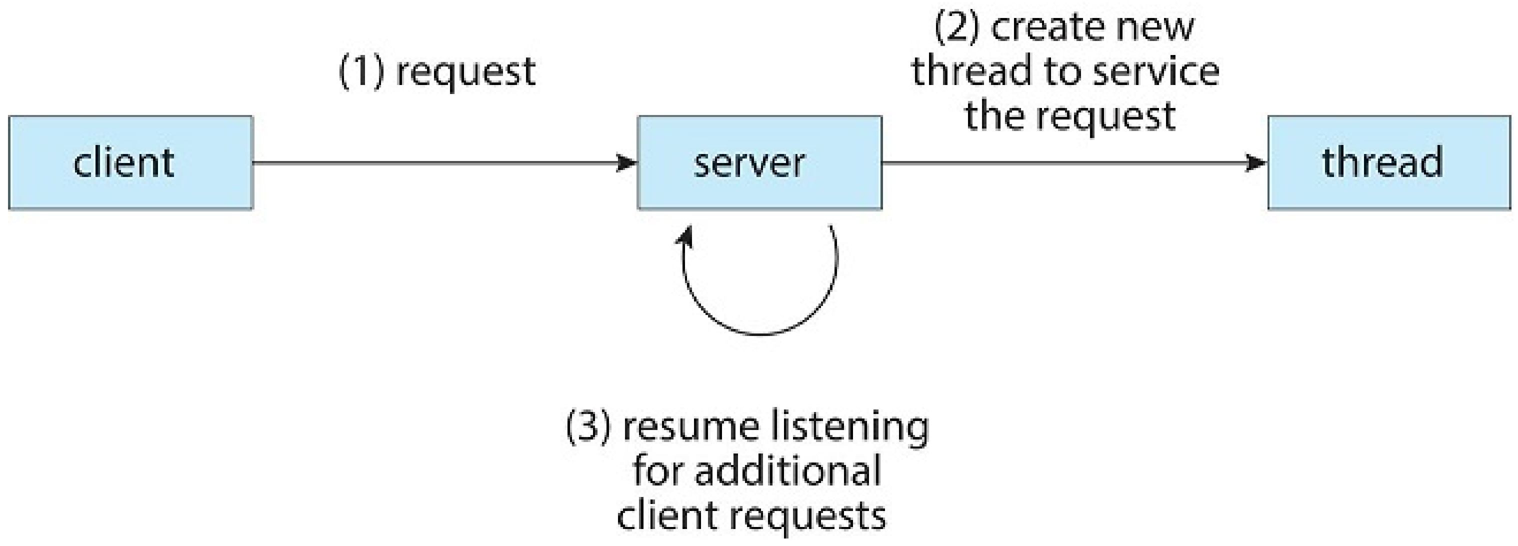

(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Figure 4.2 Multithreaded server architecture.

Overview

- Benefits
  - Responsiveness
    - ✓ To continue running even if part of it is blocked or is performing a lengthy operation
  - Resource Sharing
    - ✓ Processes can share resources only through techniques such as shared memory and message passing.
  - Economy
    - ✓ Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads
  - Scalability 늘어나고 줄어드는 속성이 자유로움
    - ✓ Can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores

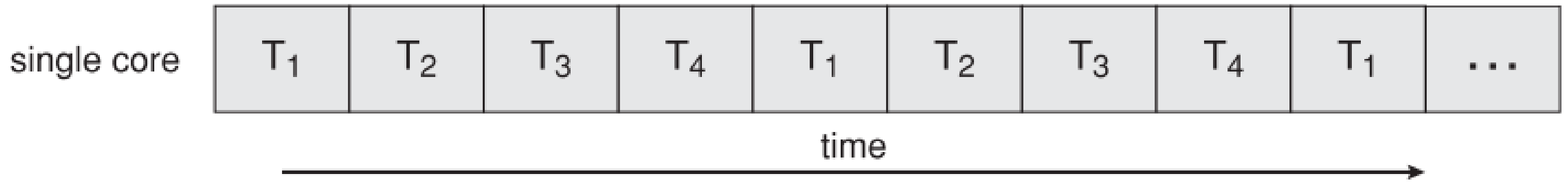
Multicore Programming

– Multicore Programming

- Multithreaded programming provides a mechanism for more efficient use of multiple computing cores and improved concurrency.
- On a system with a single computing core
  - ✓ Concurrency means that the execution of the threads will be interleaved over time, because the processing core is capable of executing only one thread at a time.
- On a system with multiple cores
  - ✓ Concurrency means that some threads can run in parallel, because the system can assign a separate thread to each core
- ***Parallelism*** implies a system can perform more than one task simultaneously 하나의 CPU에 코어가 여러개 있을 때 서로 다른 독립적인 작업을 수행 가능하다는 것
- ***Concurrency*** supports more than one task by allowing all the tasks to make progress 하나의 코어가 있을 때도 동시 수행은 가능하다.

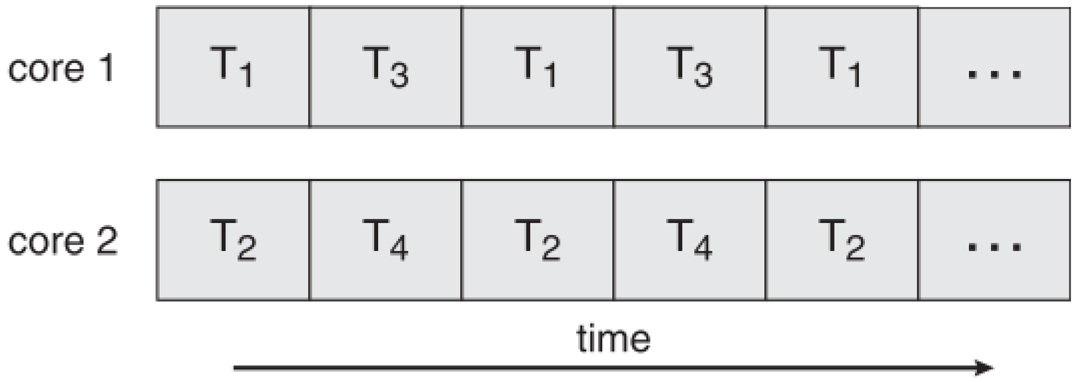
Multicore Programming

아래의 간트 차트가 interleaved되었다는 것을 의미함



코어가 하나일 때는 패래럴이라는 표현을 쓰지 않는다

Figure 4.3 Concurrent execution on a single-core system.



코어가 2개 이상이면 패래럴 실행이 가능하다

Figure 4.4 Parallel execution on a multicore system.

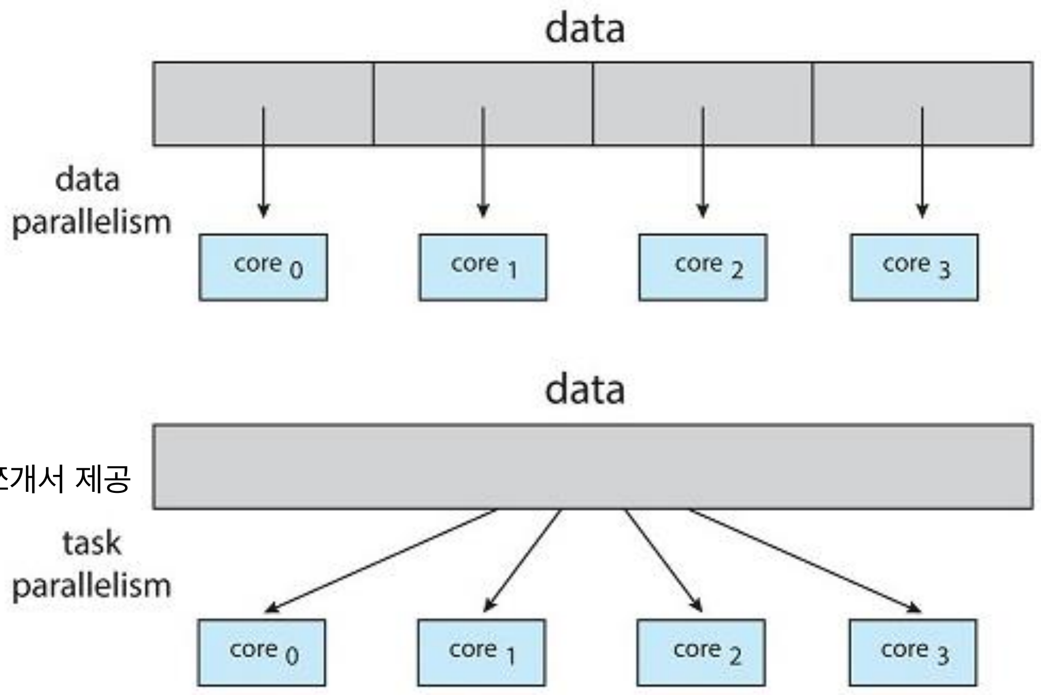
독립적으로 동시에 수행 가능

Multicore Programming

- Five areas present challenges in programming for multicore system
  - Identifying tasks
    - ✓ To find areas that can be divided into separate, concurrent tasks
  - Balance 특정 코어에 작업이 몰리는 것을 어떻게 균형을 맞출 것인가?
    - ✓ Ensure that the tasks perform equal work of equal value
  - Data splitting 어떻게 나눠서 처리할 것이냐?
    - ✓ The data accessed and manipulated by the tasks must be divided to run on separate cores
  - Data dependency
    - ✓ The data accessed by the tasks must be examined for dependencies between two or more tasks
  - Testing and debugging
    - ✓ Testing and debugging such concurrent programs is inherently more difficult than testing and debugging single-threaded applications

Multicore Programming – Types of Parallelism

- Data parallelism – distributing subsets of the same data across multiple computing cores and performing the same operation on each core
- Task parallelism – distributing not data but tasks (threads) across multiple computing cores



커다란 작업을 쪼개서 제공

Figure 4.5 Data and task parallelism.

Amdahl's Law

코어가 2개 이상인 경우가 많은데, 이 때 패래럴하게 처리하지 못하고 Serial하게 처리하는 경우가 있다

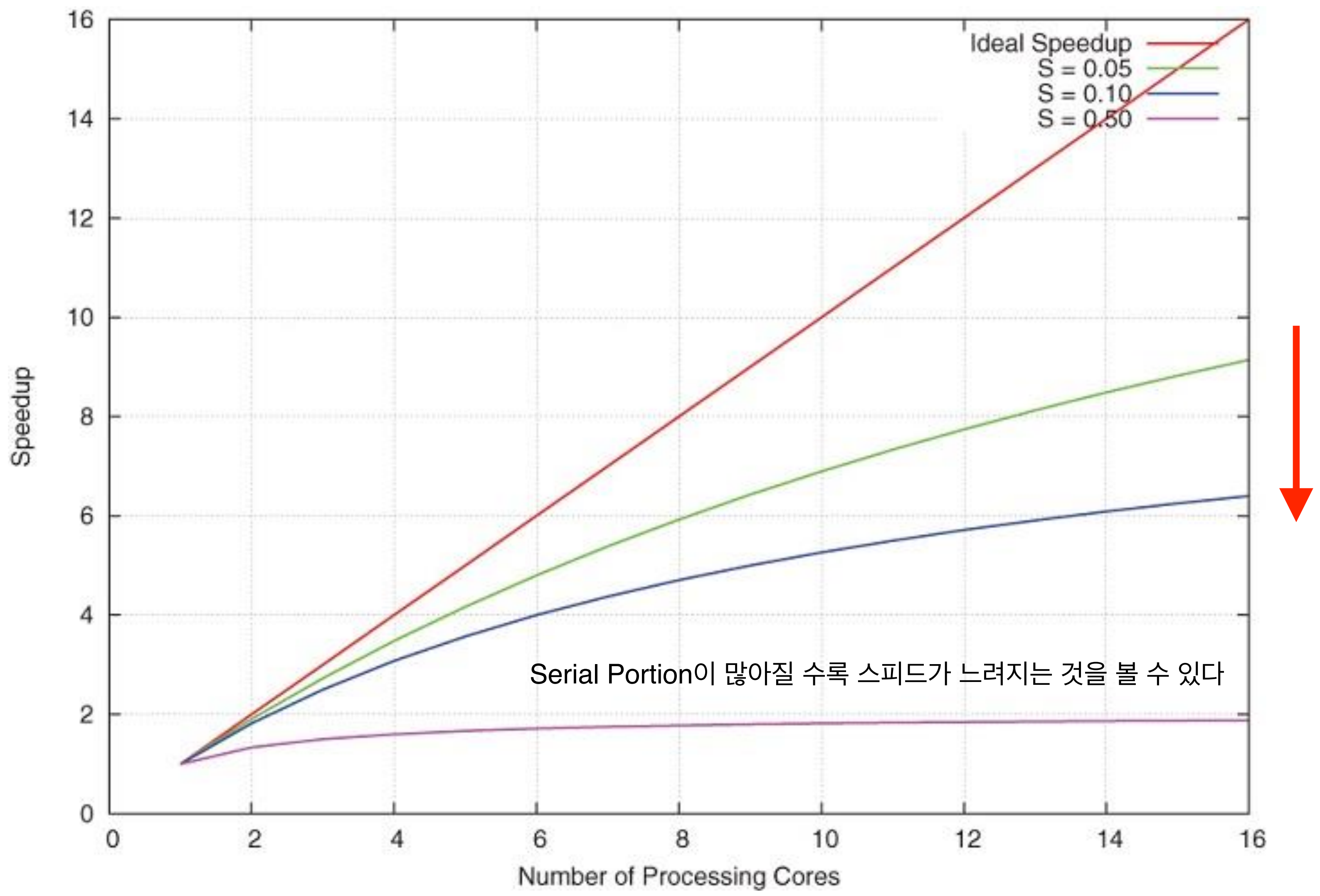
- Identifies performance gains from adding additional computing cores to an application that has both serial and parallel components
- *S* is the portion of the application that must be performed serially on a system with *N* processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

1-S : portion
N : 코어의 갯수

- If application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As *N* approaches infinity, the speedup converges to 1 / *S*.
  Serial portion of an application can have a disproportionate effect on the performance gained by adding additional computing cores.
- Some argue that Amdahl's law does not take into account the hardware performance enhancements used in the design of contemporary multicore systems.

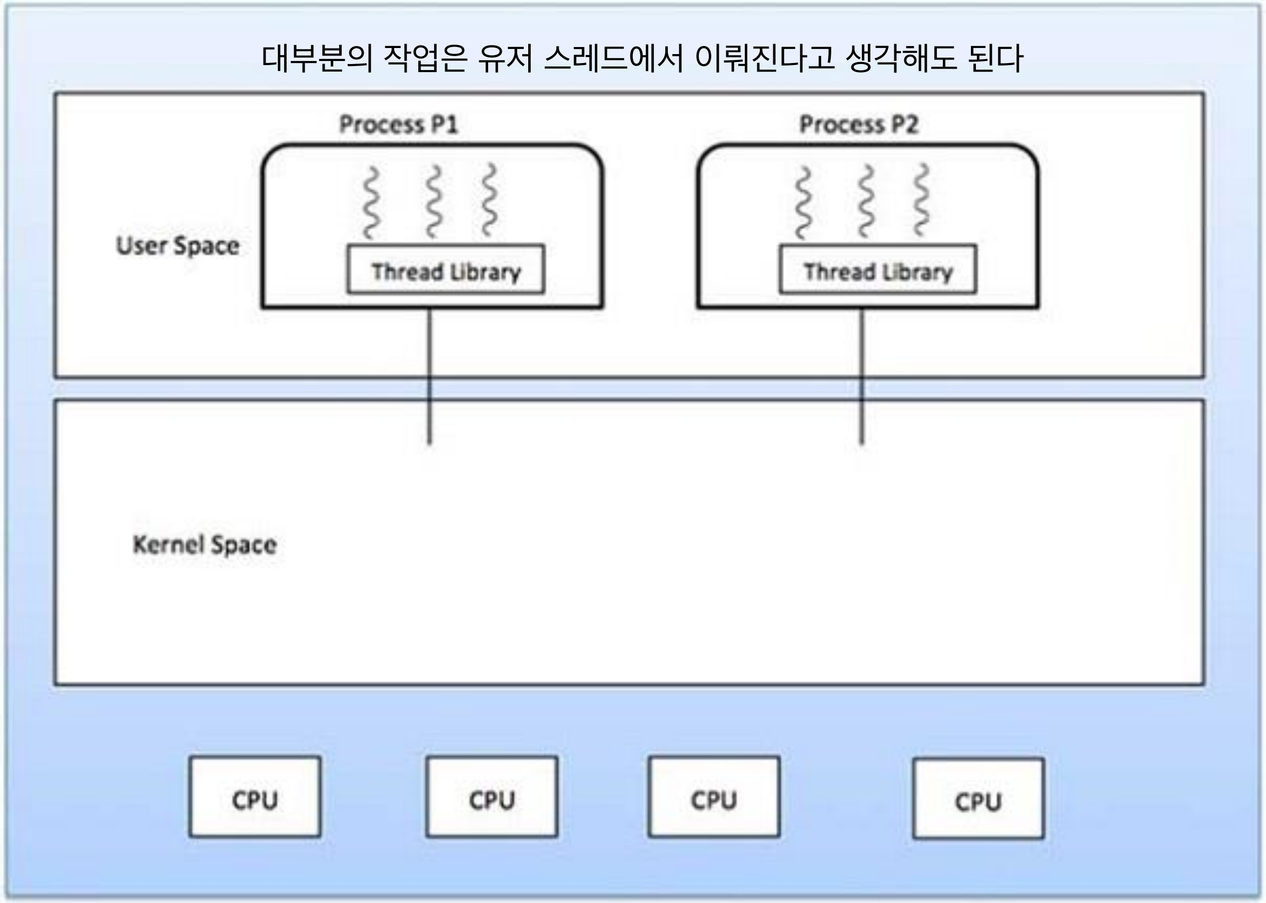
Amdahl's Law



Serial Portion이 많아질 수록 스피드가 느려지는 것을 볼 수 있다

Multithreading Models

- User threads
  - Supported above the kernel and are managed without kernel support



Multithreading Models

- User threads
  - Thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.
- – Advantages
  - Thread switching does not require kernel mode privileges.
  - User threads can run on any operating system.
  - User threads are fast to create and manage.
- – Disadvantages
  - The entire process is blocked if one user thread performs blocking operation. I/O 가 발생하면 전부 Blocking된다
  - Multithreaded application cannot take advantage of multiprocessing.

Multithreading Models

- Kernel threads 유저 스레드와 반대 개념으로 이해하면 매우 빠를 것이다
  - Supported and managed directly by the operating system
- – Advantages
  - Multiple threads of the same process can be scheduled on different processors in kernel threads.
  - If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- – Disadvantages
  - A mode switch to kernel mode is required to transfer control from one thread to another in a process.
  - Thread operations are hundreds of times slower compared to user threads.

Multithreading Models

– Many-to-One Model

- Maps many user-level threads to one kernel thread
- Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multicore systems
- Examples:
  - ✓ Solaris Green Threads
- Very few systems continue to use the model because of its inability to take advantage of multiple processing cores.

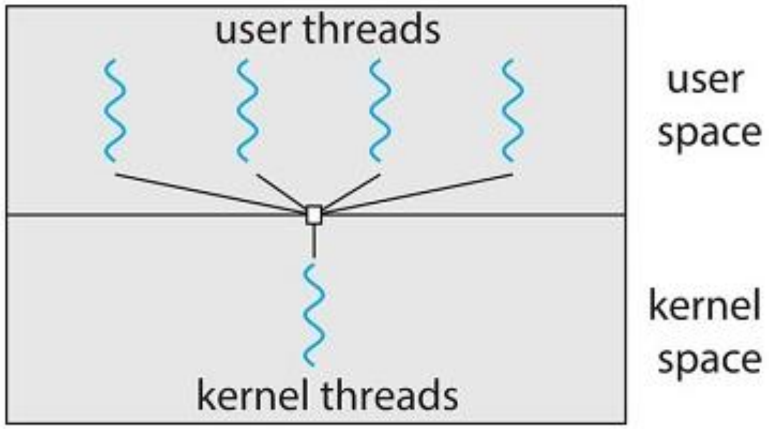


Figure 4.7 Many-to-one model.

커널 쪽으로 컨트롤이 넘어가는 경우
: System call이 발생할 때

Blocking System Call

Multithreading Models

– One-to-One Model 커널 스레드

- Maps each user thread to a kernel thread
- It provides more concurrency than the many-to-one model
- To run in parallel on multiprocessors
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread, and a large number of kernel threads may burden the performance of a system.
- Examples
  - ✓ Family of Windows OS
  - ✓ Linux

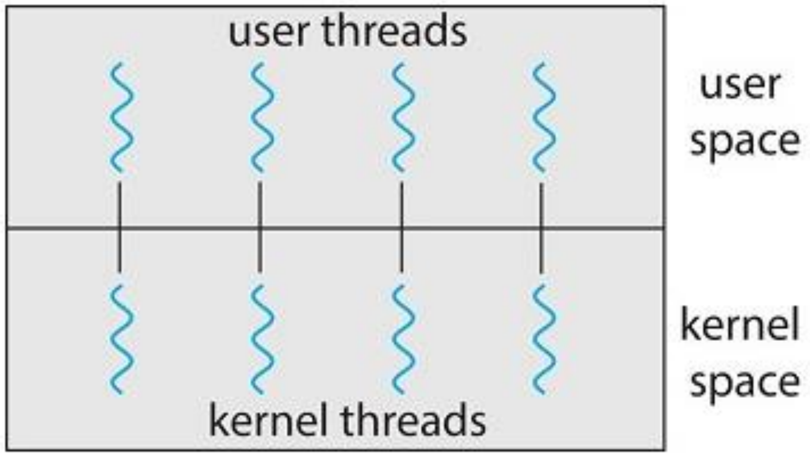


Figure 4.8 One-to-one model.

Multithreading Models

– Many-to-Many Model 하이브리드 모델

- Multiplexes many user-level threads to a smaller or equal number of kernel threads
- The number of kernel threads may be specific to either a particular application or a particular machine.

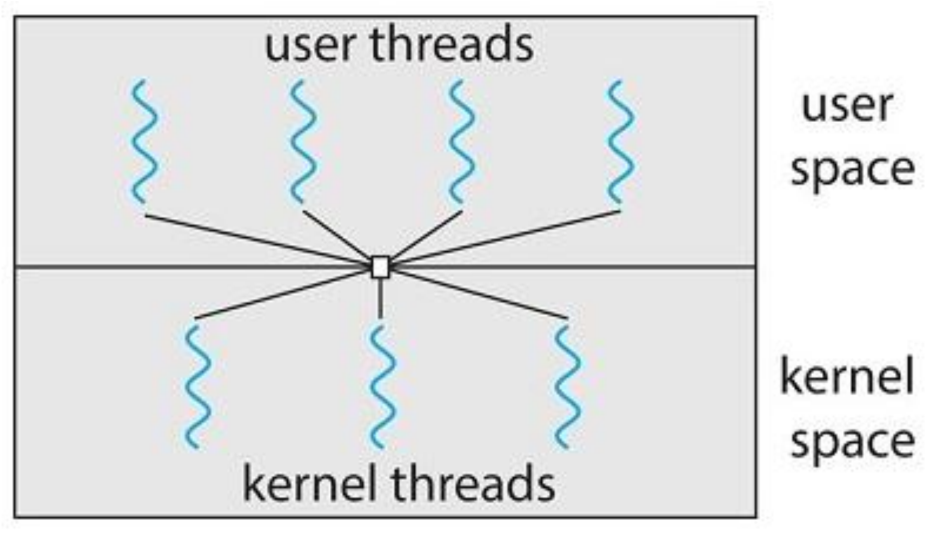


Figure 4.9 Many-to-many model.

Multithreading Models 설명 거의 안하고 스킵

– Many-to-Many Model

- Shortcomings of other models
  - ✓ Whereas the many-to-one model allows the developer to create as many user threads as she wishes, it does not result in parallelism, because the kernel can schedule only one thread at a time.
  - ✓ The one-to-one model allows greater concurrency, but the developer has to be careful not to create too many threads within an application
- The many-to-many model suffers from neither of these shortcomings:
  - ✓ Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
  - ✓ Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Multithreading Models 설명 거의 안하고 스킵

– Two-level Model

- Multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread
- Example
  - ✓ Solaris 8 and earlier

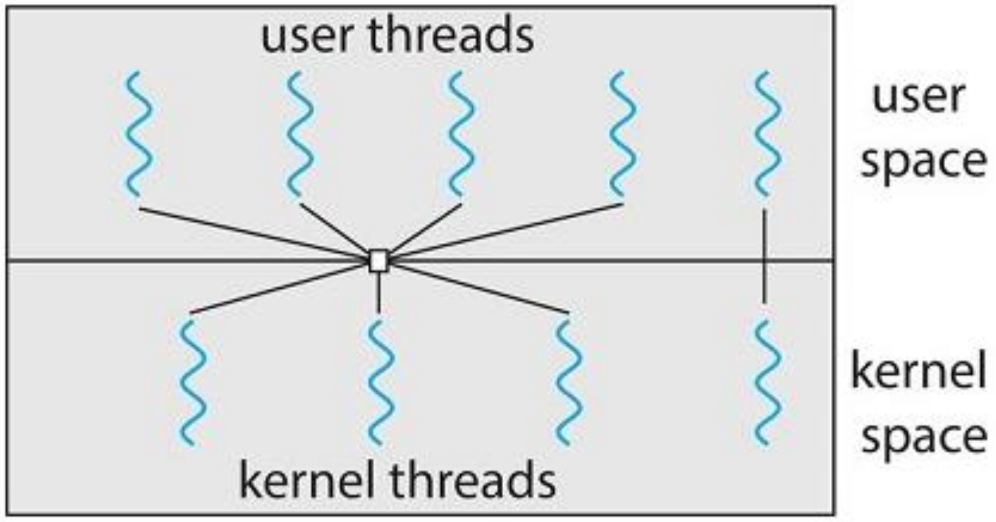


Figure 4.10 Two-level model.