CSE308 Operating Systems

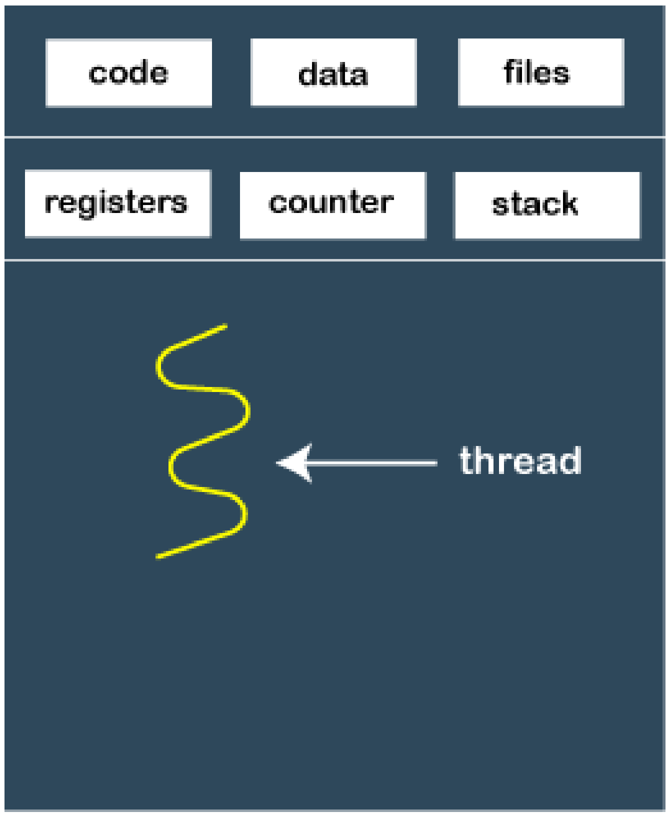
Threads

S.Rajarajan
SASTRA

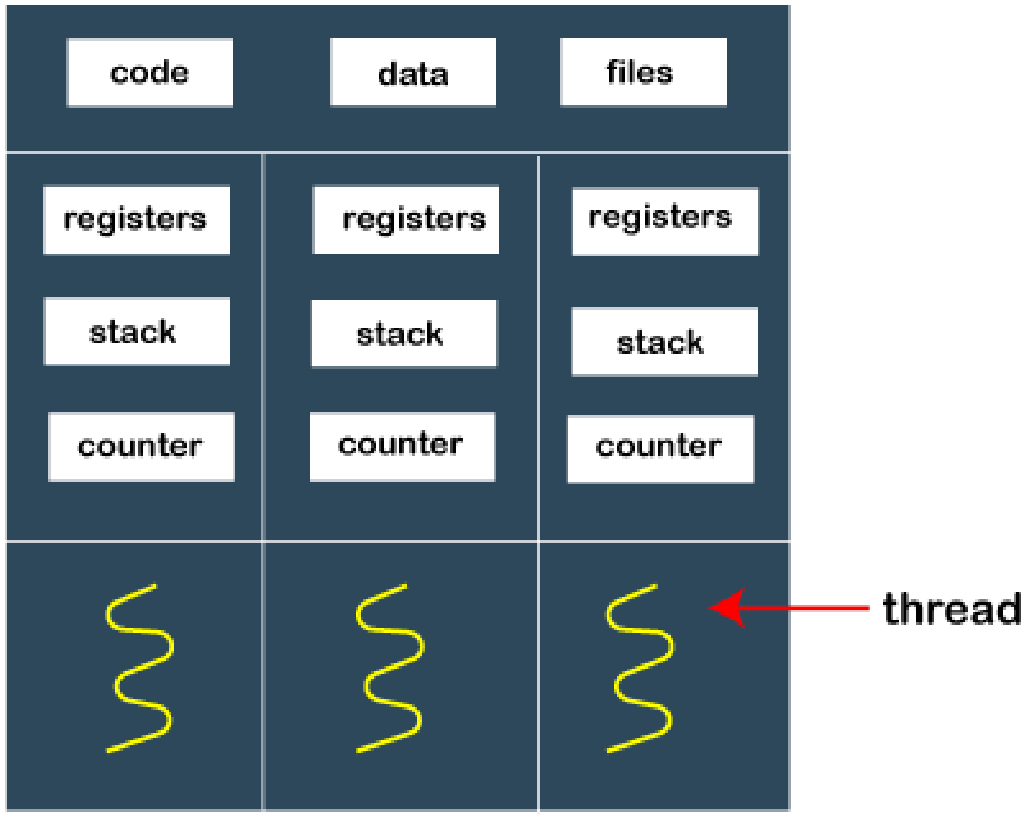
What is a thread ?

- A process is a program in execution.
- A process can be further divided in to several units of execution called threads.
- Thread is also called as **light-weight process**.
- A thread is a basic unit of CPU utilization
- It comprises
  - a **thread ID**
  - **a program counter**
  - **a register set**
  - **and a stack**.

- Threads belonging to the same process share the following of the process:
  - **code section**
  - **data section**
  - **and other operating-system resources such as open files and signals**



Single-threaded process



Multi-threaded process

Thread creation Vs Process creation

- Process creation involves **allocating memory** for the process, **loading the program into memory**, creating the **context and PCB**, attaching the **process to a queue** and **scheduling the process**.
- Thread creation only involves **logically partitioning** the process into a number of threads **at run time** and creating **contexts for each thread**.

Motivation

- Most **software applications** that run on modern computers are **multithreaded.**
- An application typically is implemented as a separate **process** with **several threads of control**
- A **web browser** might have one thread that 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

- Applications can also be designed to leverage processing capabilities on **multicore systems**.
- Such applications can perform several CPU-intensive **tasks in parallel** across the multiple computing cores
- In certain situations, **a single application** may be required to perform **several similar tasks** e.g merge sort

Web Server

- A busy **web server** may have several (perhaps thousands of) **clients concurrently accessing it**.
- If the web server ran as a **traditional single-threaded process,** it would be able to service only one client at a time, and a client might have to **wait a very long time**
- 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
- In fact, this process-creation method was in common use before threads became popular

- Process creation is time consuming and resource intensive
- It is generally **more efficient** to use **one process** that contains **multiple threads**
- When a request is made, rather than creating another process, the **server creates a new thread** to service the request

RPC

- Threads also play a vital role in remote procedure call (RPC) systems
- RPCs allow **inter-process communication** by providing a communication mechanism similar to ordinary function or procedure calls.
- Typically, **RPC servers are multithreaded**.
- When a **server receives a message**, it **services the message** using a **separate thread.**
- This allows the server to service several concurrent requests

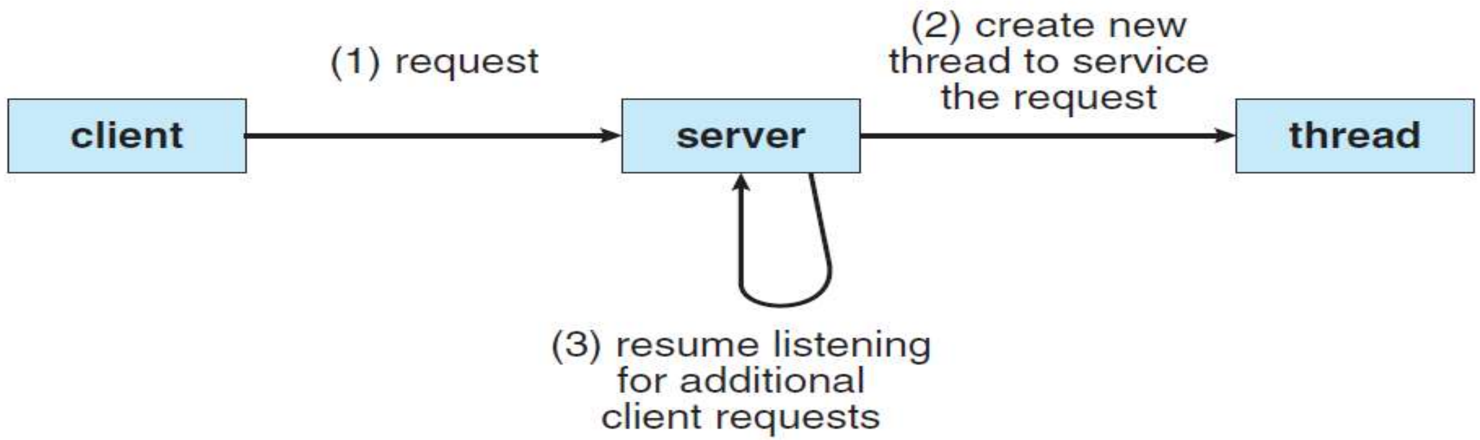


Figure 4.2 Multithreaded server architecture.

Multithreaded kernel

- Most **operating-system kernels** are now **multithreaded**.
- Several threads operate in the kernel, and each thread performs a specific task, such as **managing devices**, **managing memory**, or **interrupt handling**
- Solaris has a set of threads in the kernel specifically for interrupt handling;
- **Linux** uses a **kernel thread** for managing the **amount of free memory** in the system.

Benefits of multithreaded programming

- **Responsiveness.**
  - Multithreading an interactive application may allow a program to continue running even **if part of it is blocked**, or is performing **a lengthy operation**, thereby **increasing responsiveness** to the user.
  - A single-threaded application would be unresponsive to the user until the operation had completed.
- **Resource sharing / Communication.**
  - Processes can only share resources or communicate through techniques such as **shared memory** and **message passing**.
  - Such techniques **must be explicitly arranged** by the programmer.
  - However, threads **share the memory** and the **resources of the process** to which they belong by default. So **shared memory** is used for inter-thread communication

- The benefit of sharing code and data is that it allows an application to have **several different threads of activity** within the **same address space**.

- **Economy.**
  - Allocating **memory and resources** for process creation is costly
  - Because **threads share the resources** of the process to which they belong, it is more economical to create and **context-switch** among **threads**.

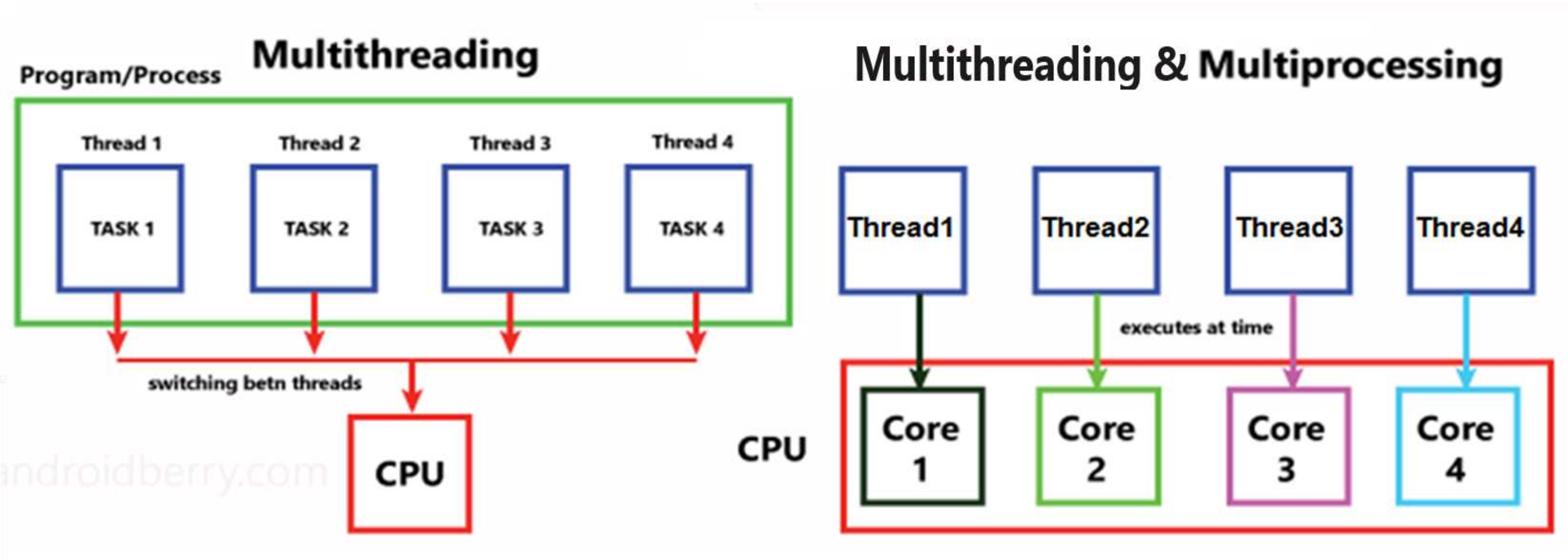
- **Scalability.**
  - The benefits of multithreading can be even greater in a **multi-core architecture**, where **threads may be running in parallel** on different processing cores.
  - A **single-threaded** process can run on **only one processor**, regardless how **many processors are available**.

Multicore Programming

- A recent trend in system design is to place **multiple computing cores** on a single chip.
- Each core appears as a separate processor to the operating system.
- **Multithreaded programming** provides a mechanism for more **efficient use of the multiple computing cores** and improved concurrency

- Consider an **application** with **four threads**.
- On a system with a **single computing core**, concurrency merely 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**, however, concurrency means **overlapping the threads** so that they can **run in parallel**, because the system can assign a separate thread to each core

Interleaving Vs Overlapping



Concurrency Vs Parallelism

- A system is parallel if it can perform **more than one task simultaneously**.
- In contrast, a concurrent system supports more than one task by allowing **all the tasks to make progress (by switching).**
- Thus, it is possible to have concurrency without parallelism.
- Before the advent of **SMP** and **multicore architectures**, most computer systems had only a **single processor**.
- **CPU schedulers** were designed to provide the **illusion of parallelism** by **rapidly switching** between processes in the system, thereby allowing each process to make progress.

Amdhal's Law on Performance Enhancement by Multiple cores

- Amdahl's Law is a formula that identifies potential performance gains from adding additional computing cores to an application that has both **serial (nonparallel)** and **parallel** code.
- If ***S*** *is the portion of the application* that must be **performed serially** on a system with ***N processing cores****, the* formula appears as follows:

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

- As an example, assume we have an application that is **75 percent parallel** and **25 percent serial**. If we run this application on a system with **2 processing cores**, we can get a speedup of **1.6 times**.
- 1 / (0.25 + (1- 0.25)/2) =1 /(0.25 + 0.375) = 1/ 0.625 = 1.6 %
- If we add 2 additional cores (for a total of 4), the speedup will be
  – 2.28

Programming Challenges

1. **Identifying tasks.** This involves examining applications **to find areas** that can be **divided** into **separate, concurrent tasks**. Ideally, that tasks are **independent** of one another and can **run in parallel** on individual **cores.**
2. **Balance.** While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform **equal work** of equal value.
3. **Data splitting.** Just as applications are divided into separate tasks, **the data accessed** and manipulated by the tasks **must be divided** to run on separate cores.

4. **Data dependency.** The data accessed by the tasks must be examined for dependencies between two or more tasks. When **one task depends on data from another**, programmers must ensure that the execution of the tasks is **synchronized** to accommodate the data dependency.
5. **Testing and debugging.** When a program is running in parallel on multiple cores, many **different execution paths** are possible. Testing and debugging such concurrent programs is **inherently more difficult**

Types of Parallelism

- There are two types of parallelism:
  - **data parallelism** and **task parallelism.**
- Data parallelism focuses on **distributing subsets of the same data** across **multiple computing cores** and performing the **same operation** on each core.
- Consider, for example, summing the contents of an array of size *N.*
- *On a* single-core system, one thread would simply sum the elements [0] . . . [*N − 1].*
- On a dual-core system, however, thread *A, running on core 0, could sum the* elements [0] . . . [*N/2 − 1] while thread B, running on core 1, could sum the* elements [*N/2] . . . [N − 1]. The two threads would be running in parallel on* separate computing cores.

- **Task parallelism** involves **distributing** not data but **tasks (threads) across** multiple computing cores.
- Each thread is performing a **unique operation**
- Different threads may be operating on the **same data**, or they may be operating on **different data**.

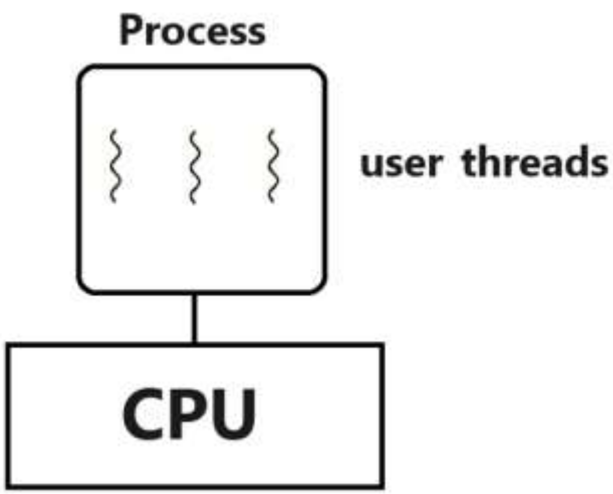
Multithreading Models

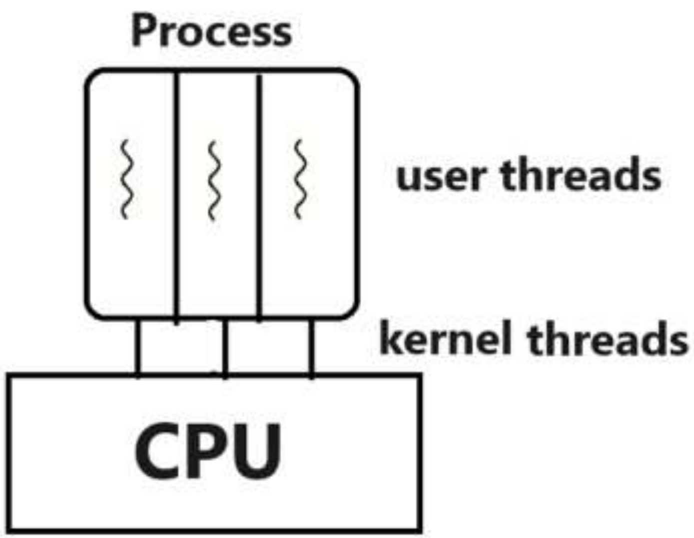
- <u>User threads Vs Kernel threads</u>
- Support for threads may be provided either at the user level, (**user level threads),** or by the kernel (**kernel threads).**
- **User threads** are created above the kernel and are managed **without kernel support.**
- **User threads** are created with the help of thread libraries available in programming languages.
- **Kernel threads** are created and managed **directly by the operating system kernel**.
- Virtually all contemporary operating systems—including Windows, Linux, Mac OS X, and Solaris— support kernel threads.
- Pthread, Java multithreading are examples of **user threads**

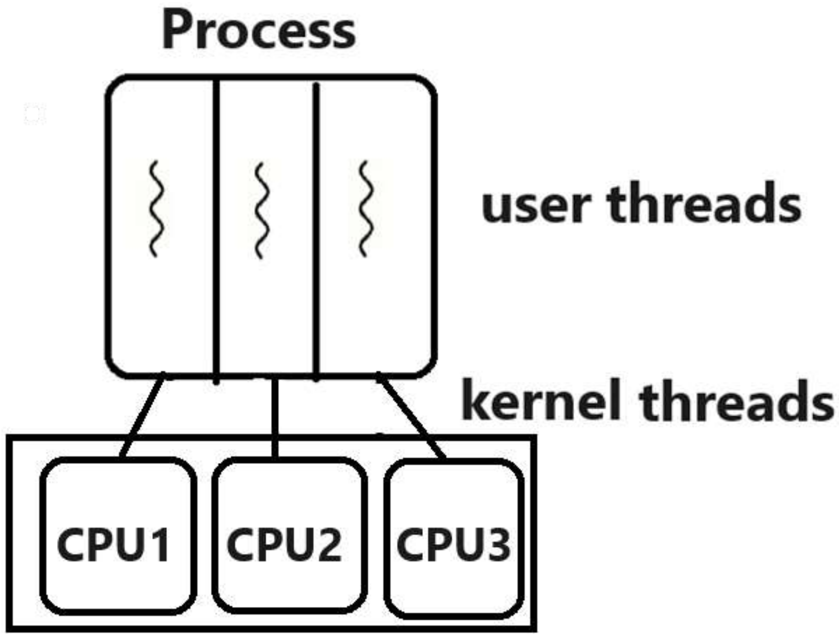
ULTs Vs KLTs

- ULTs are **created by users** with the help of thread libraries. KLTs are **created by Kernels**.
- Kernel is **unaware of ULTs**. Kernel is **fully aware of KLTs.**
- ULT Thread management is **done by the programming language**. KLT thread management is **done by Kernel**.
- In ULT, when a thread **invokes IO operation**, entire process will be blocked by kernel. Only that thread will be blocked and another thread in the same process will be scheduled on CPU.

- **No mode switching** required in ULT for thread switching. Mode switching required in KLT.
- ULT can be **implemented on any OS**. KLT can be implemented only on OS that supports multi-threading.
- ULT do not benefit from **multi-core systems**. KLTs benefit from multi-core systems by running on different cores.







ULT Vs KLT

ULT	KLT

ULT	KLT

- **Many-to-One Model**
- The many-to-one model maps **many user-level threads to one kernel thread**
- Thread management is done by the **thread library in user space**, so it is efficient.
- However, the **entire process will block** if a thread makes a **blocking system call**.
- Also, because only one thread can access the kernel at a time, multiple threads are **unable to run** in parallel on **multicore systems**.
- **Green threads—**a thread library available for Solaris systems the many-to-one model.
- However, very few systems continue to use the model.

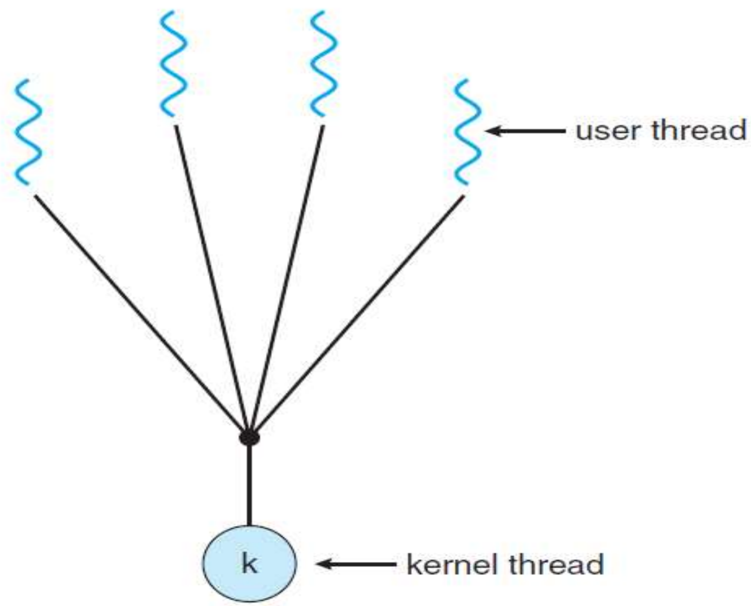
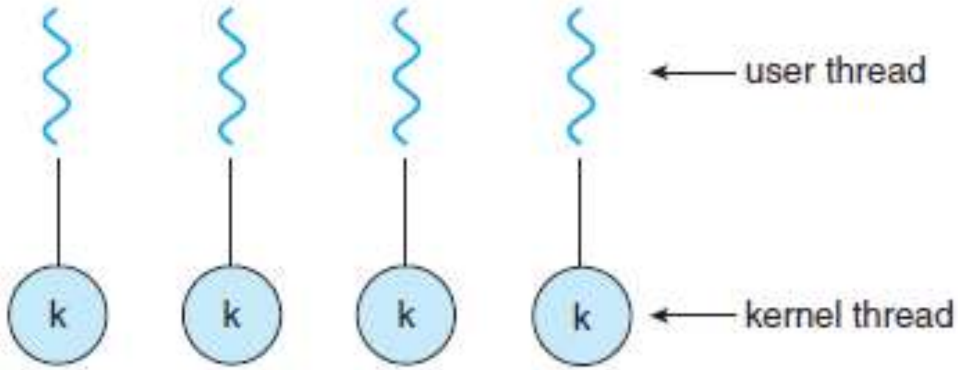


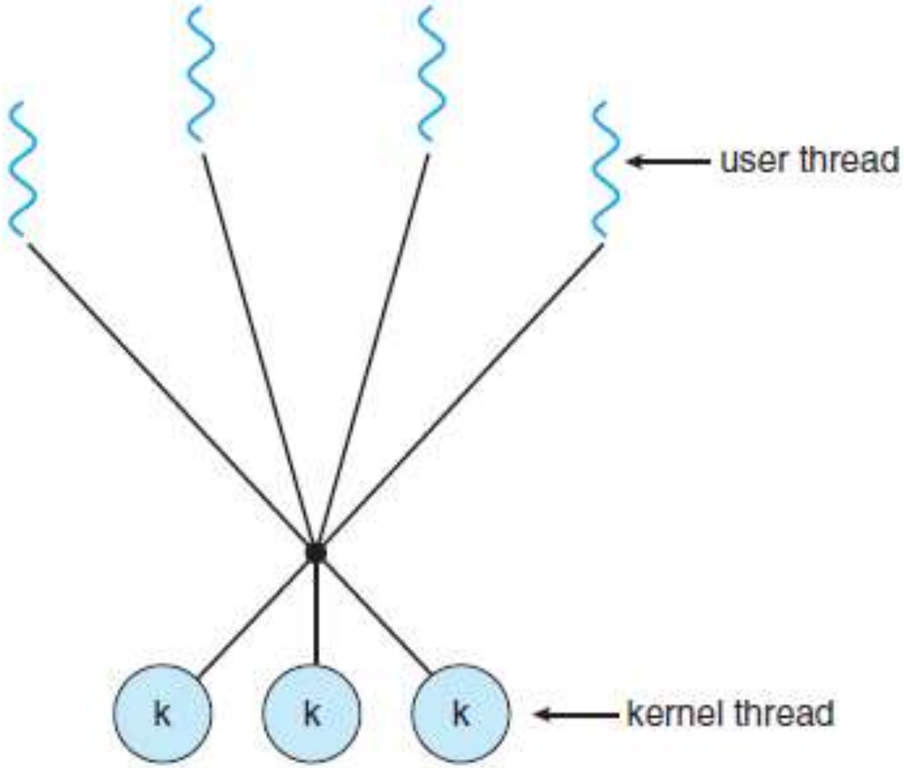
Figure 4.5 Many-to-one model.

- **One-to-One Model**
- The one-to-one model maps each user thread to a kernel thread.
- It provides **more concurrency** by allowing another thread to run when a thread makes a **blocking system call**.
- It also allows multiple threads to **run in parallel** on **multiprocessors**.
- The only **drawback** to this model is that creating a user thread requires **creating the corresponding kernel thread**.
- Because the **overhead of creating kernel threads** can burden the performance of an application, most implementations of this model **restrict the number of threads** supported by the system


user thread
k
k
k
k
kernel thread

- **Many-to-Many Model**
- The many-to-many model **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.
- **Advantage**
- Whereas the **many to-one model**, **does not** result in **true concurrency**, 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.


user thread
k
k
k
kernel thread

Process	Thread

Context switching involves mode switching.	**Context switching may or many not involve mode switchin g.**

Thread Libraries

- A thread library provides the programmer with an **API** for **creating and managing threads**
- There are **two primary ways** of implementing a thread library
- The first approach is to provide a library **entirely in user space** with **no kernel support**
- All code and data structures for the library exist in user space
- This means that invoking a function in the library results in a **local function call** in user space and **not a system call**.

- The second approach is to implement a kernel-level library supported directly by the operating system
- In this case, code and data structures for the library exist in kernel space
- Invoking a function in the API for the library typically results in a system call to the kernel
- Three main thread libraries are in use today:
  - POSIX Pthreads
  - Windows,
  - Java. Pthreads

- The **Windows thread library** is a **kernel-level** library available on Windows systems.
- The **Java thread API** allows threads to be created and managed directly in Java programs.
- However, because in most instances the JVM is running on top of a host operating system, the Java thread API is generally implemented using a thread library available on the host system
- This means that **on Windows systems**, **Java threads** are typically implemented **using the Windows API**
- **UNIX and Linux** systems often use **Pthreads.**

Pthreads

- Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation
- Operating-system designers may implement the specification in any way they wish.
- Numerous systems implement the Pthreads specification; most are UNIX-type systems, including Linux, Mac OS X, and Solaris.n and synchronization

Summary

- What are threads ?
- Differences between threads and processes
- Advantages of threads
- Characteristics of threads
- Types of threads