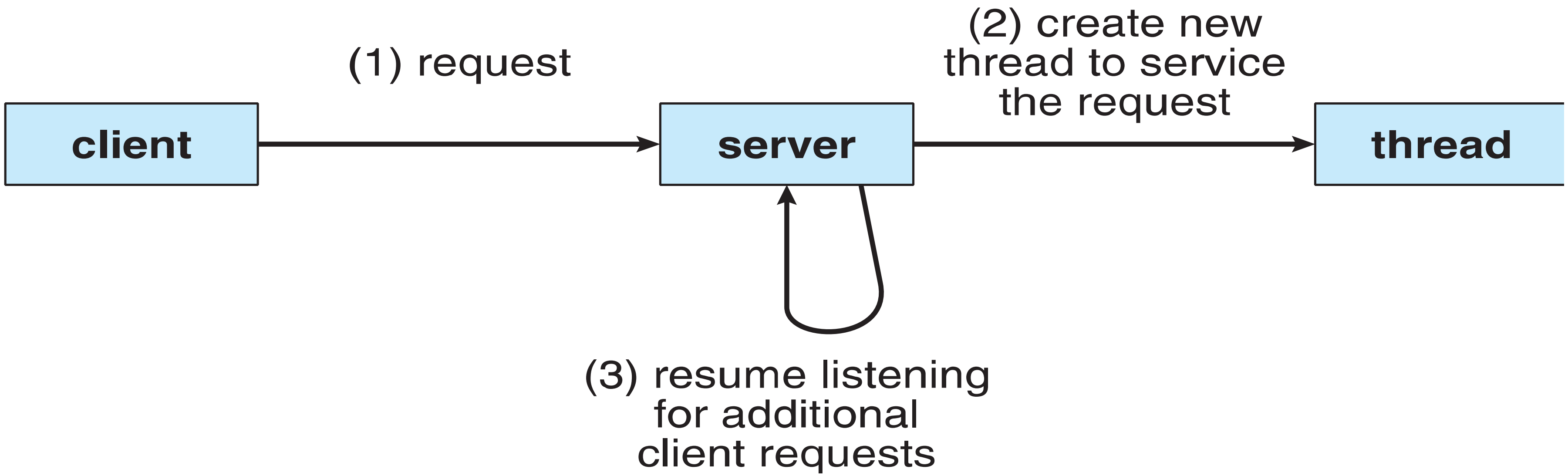
UNIT II

THREADS & CPU SCHEDULNG

Threads & CPU Schedulng

- **Threads**
  - Overview
  - Multicore Programming
  - Multithreading Models
  - Implicit Threading
  - Threading Issues
- **CPU Scheduling**
  - Basic Concepts
  - Scheduling Criteria
  - Scheduling Algorithms
  - Thread Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

Multithreaded Server Architecture



Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

Multicore Programming

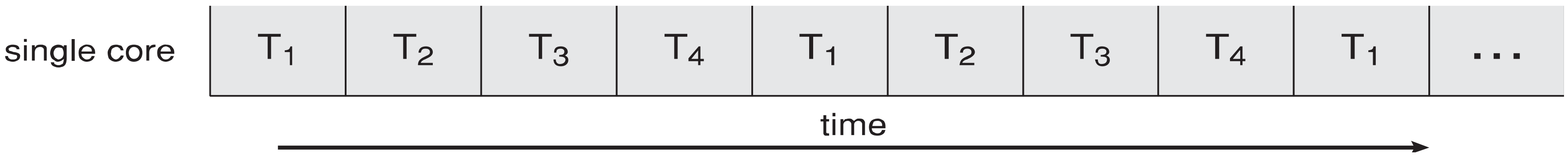
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

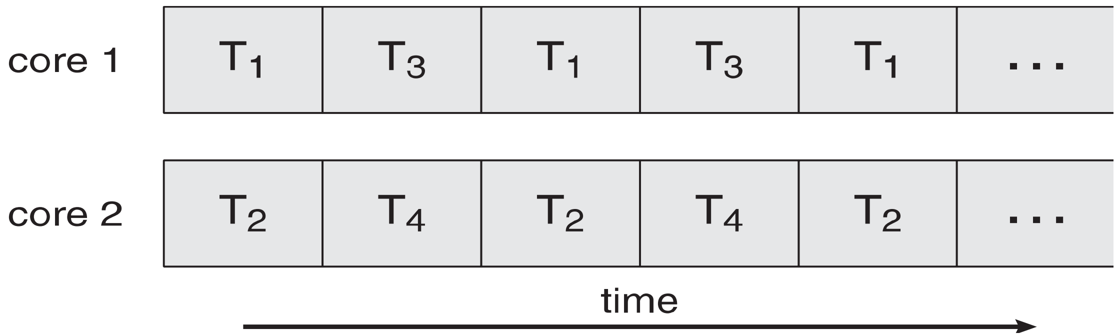
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

Concurrency vs. Parallelism

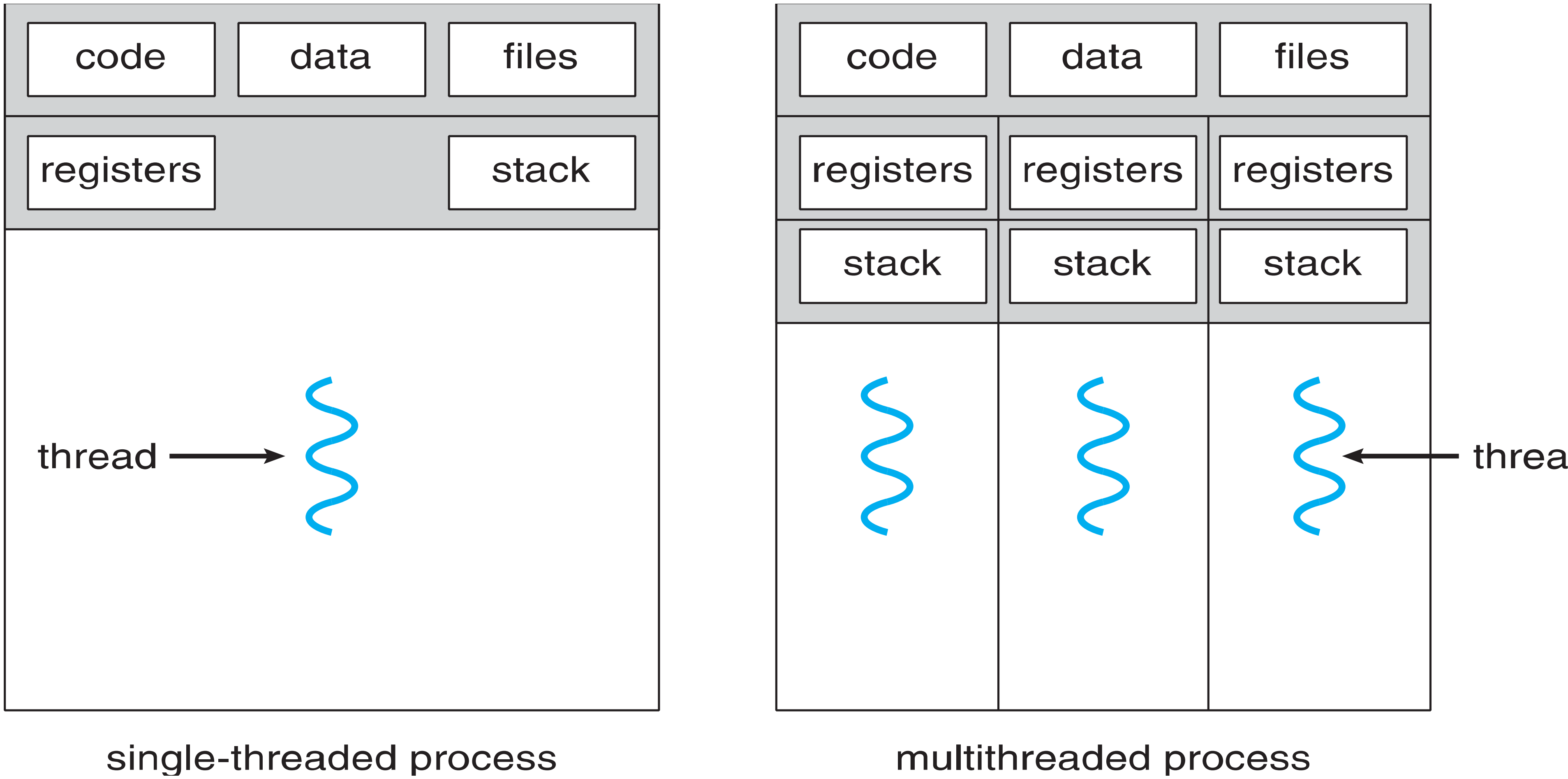
- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Single & Multithreaded Processes



Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is serial portion
- *N* processing cores

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

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As *N* approaches infinity, speedup approaches 1 / *S*

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

User Threads and Kernel Threads

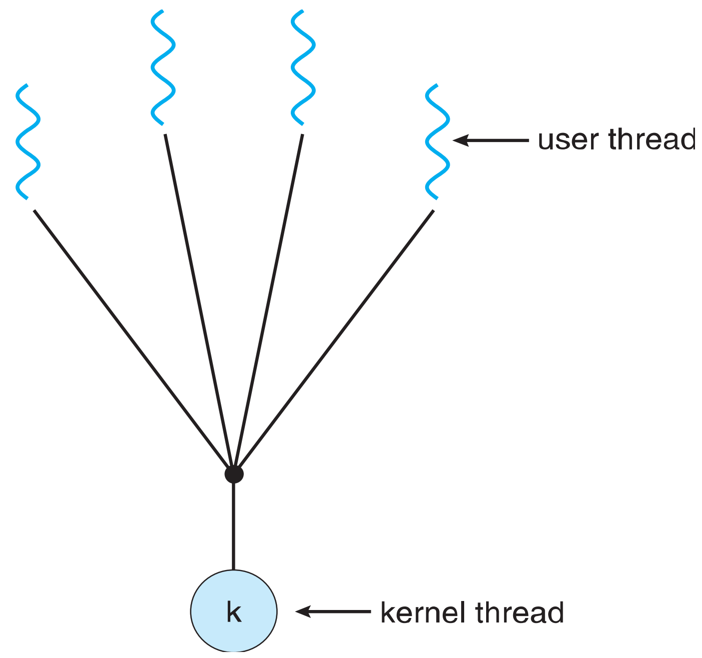
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows , Solaris , Linux ,Tru64 UNIX ,Mac OS X

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

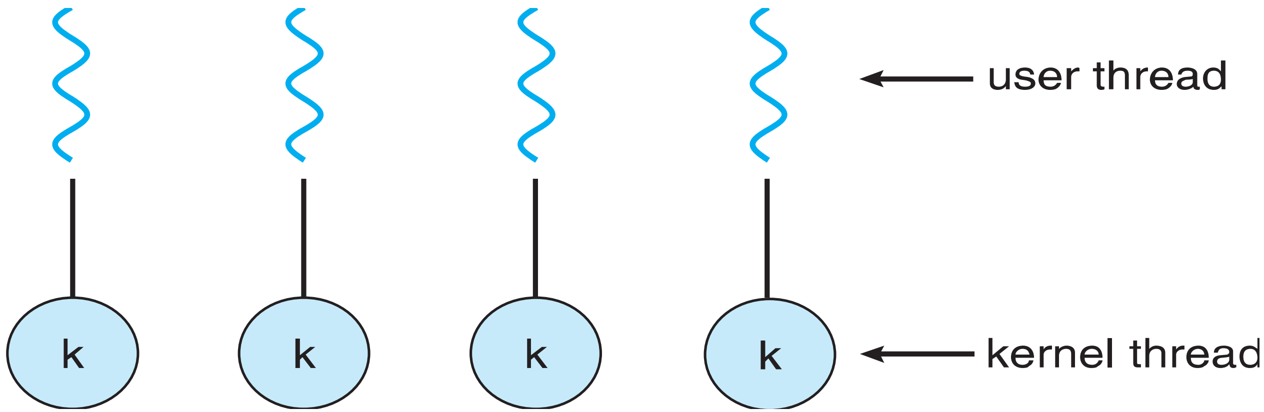
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



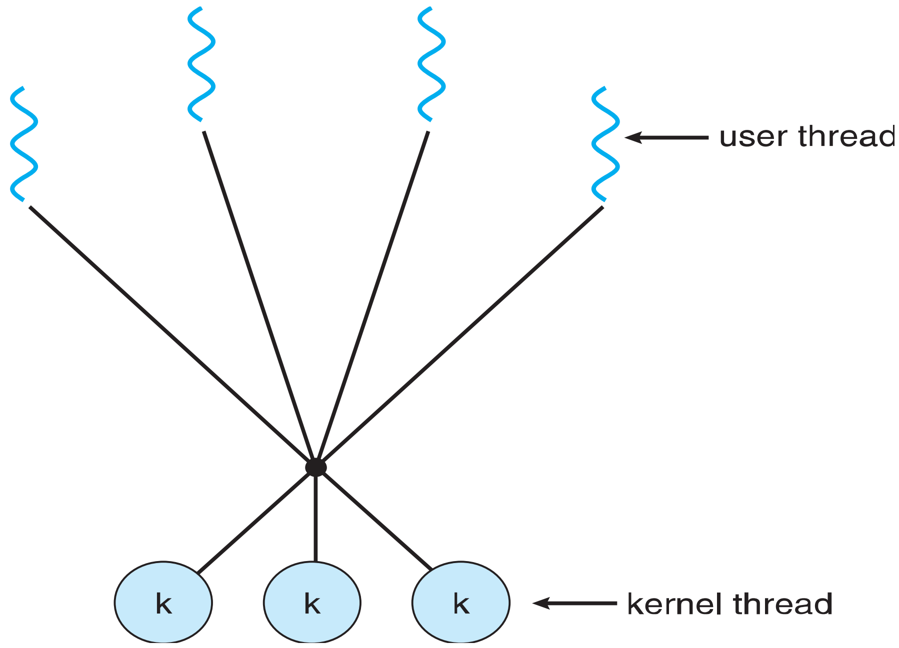
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
    - Windows
    - Linux
    - Solaris 9 and later



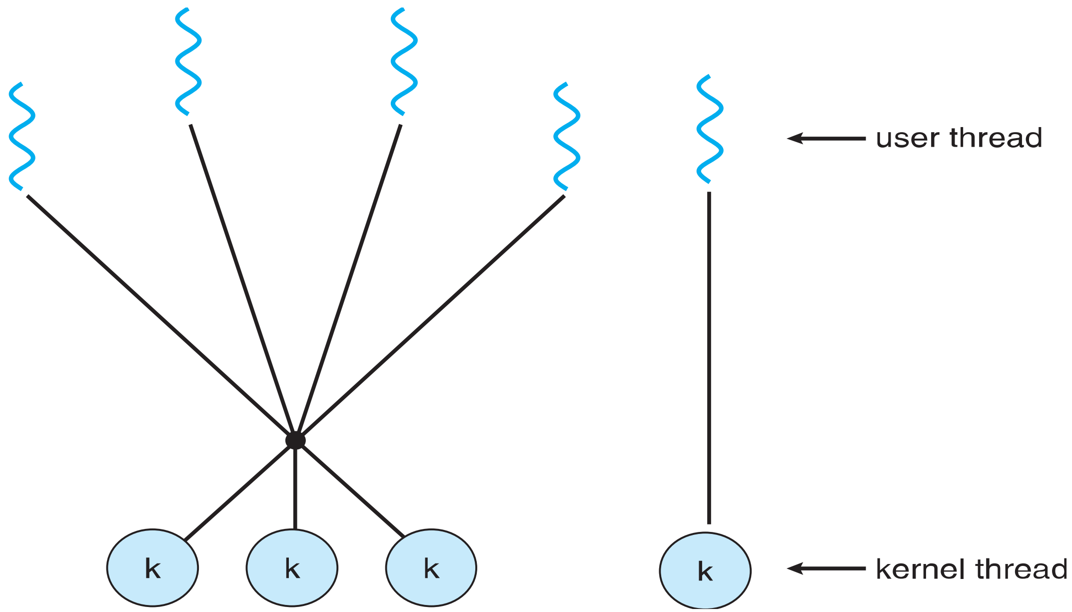
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the *ThreadFiber* package



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:

```
public interface Runnable
{
    public abstract void run();
}
```

  - Extending Thread class
  - Implementing the Runnable interface

TEXT BOOK

1. Abraham Silberschatz, Peter B. Galvin, “Operating System Concepts”, 10th Edition, John Wiley & Sons, Inc., 2018.

2. Jane W. and S. Liu. “Real-Time Systems”. Prentice Hall of India 2018.

3. Andrew S Tanenbaum, Herbert Bos, Modern Operating Pearson , 2015.

REFERENCES

1. William Stallings, “Operating Systems: Internals and Design Principles”,9th Edition, Prentice Hall of India., 2018.

2. D.M.Dhamdhere, “Operating Systems: A Concept based Approach”, 3rd Edition, Tata McGraw hill 2016.

3. P.C.Bhatt, “An Introduction to Operating Systems–Concepts and Practice",4th Edition, Prentice Hall of India., 2013.

THANK YOU