SNS COLLEGE OF TECHNOLOGY

(Autonomous)

COIMBATORE-35

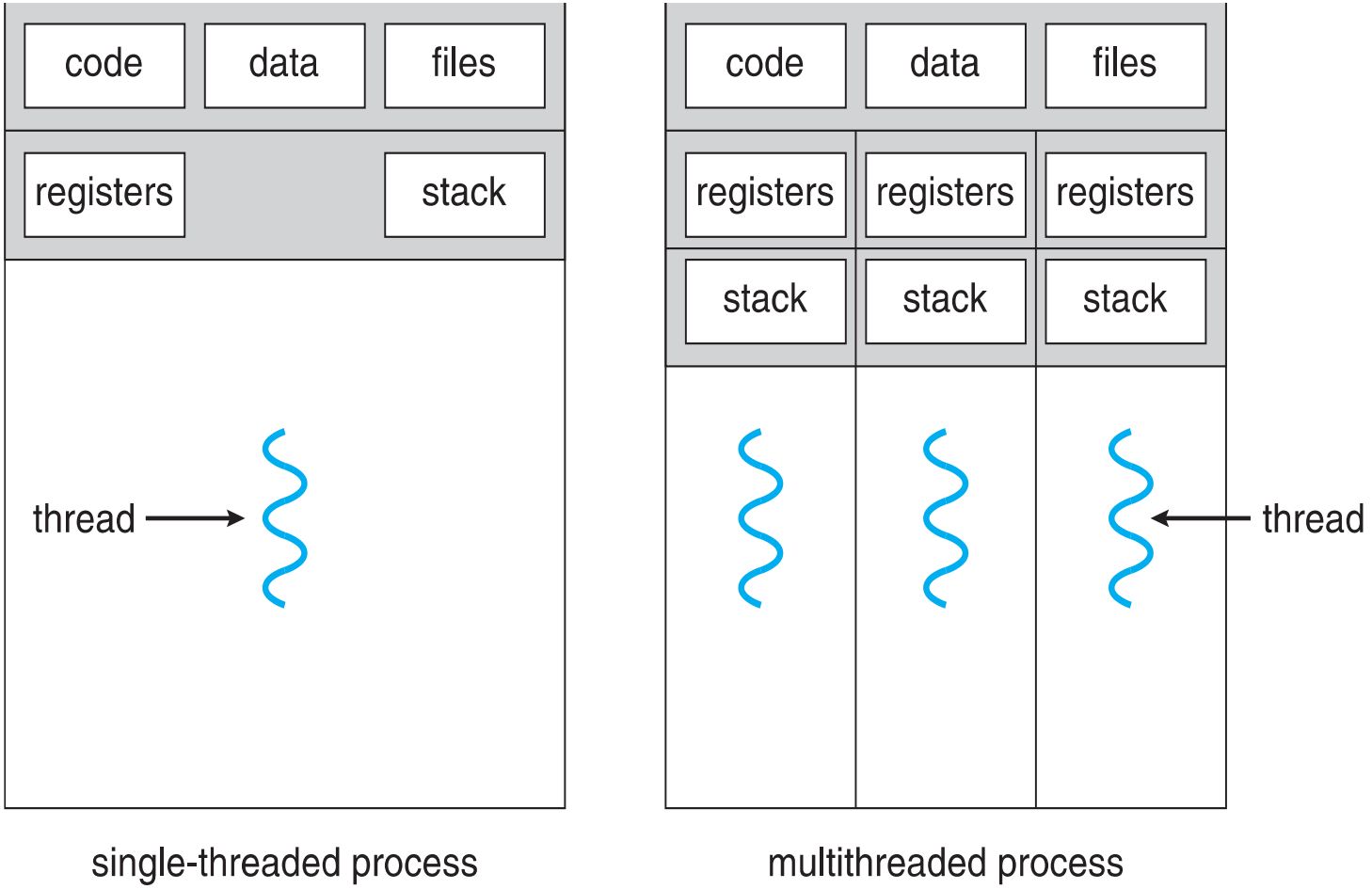
23CST201 Operating Systems

Multithreading Models

Threads

- A thread is a basic unit of CPU utilization; **it comprises a thread ID, a program counter, 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, such as open files and signals.
- A traditional (or heavyweight:) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

Single and Multithreaded Processes



Thread Control Block

Per Thread State

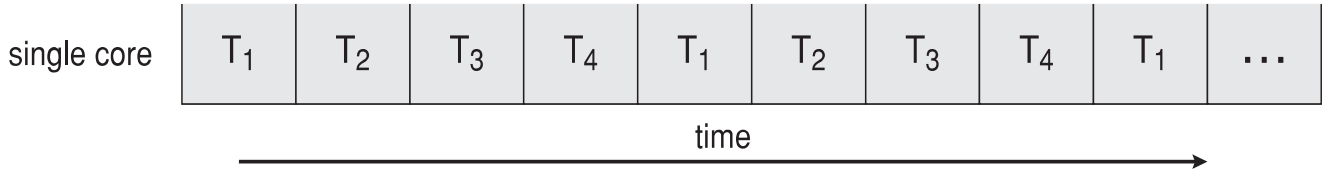
- Each Thread has a Thread Control Block (TCB)
  - Execution State: CPU registers, program counter, pointer to stack
  - Scheduling info: State (more later), priority, CPU time
  - Accounting Info
  - Various Pointers (for implementing scheduling queues)
  - Pointer to enclosing process (PCB)
  - Etc

Benefits

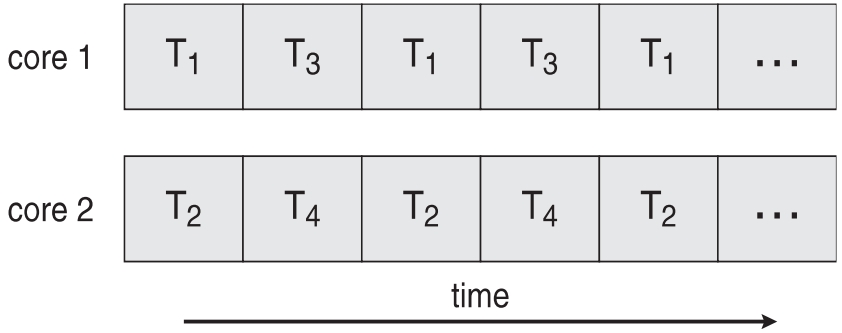
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower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

Concurrency vs. Parallelism

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



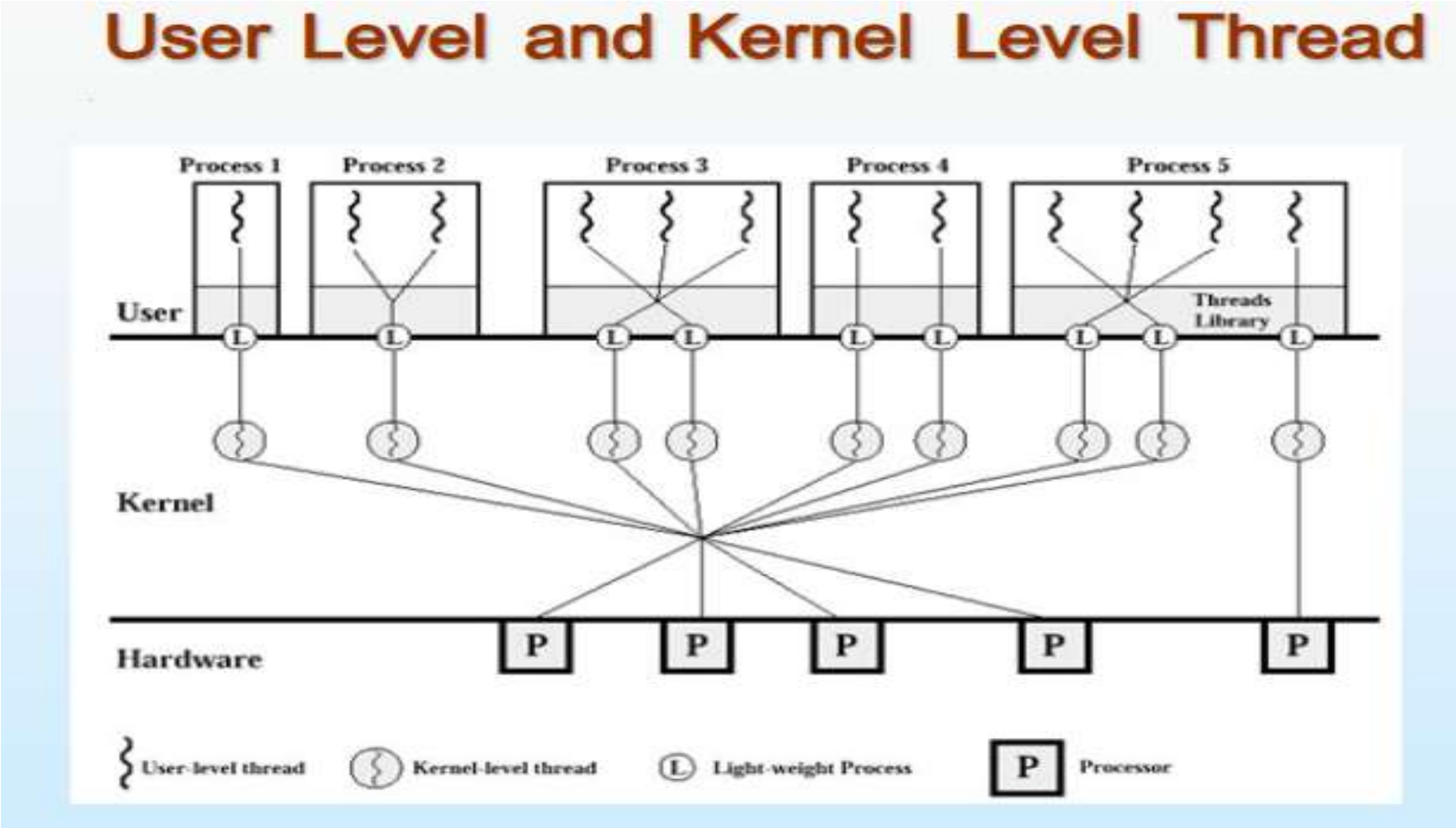
- **Parallelism on a multi-core system:**



User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Thread library provides support for the thread creation, Scheduling and management with no support from the kernel
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported directly from the operating system
- The kernel performs thread creation, Scheduling and management in the kernel space.
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

User Threads and Kernel Threads



User Threads and Kernel Threads

Advantages & Disadvantages

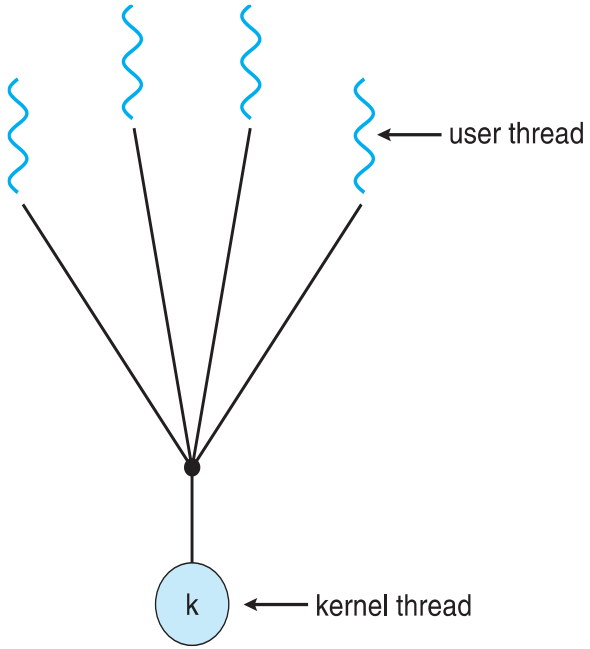
- **Advantages:**
  - User-level threads does not require modification to operating systems.
  - Thread switching does not involve the kernel -- no mode switching
  - Scheduling can be application specific -- choose the best algorithm.
  - User-level threads can run on any OS -- Only needs a thread library
- **Disadvantages:**
  - Most system calls are blocking and the kernel blocks processes -- So all threads within the process will be blocked
  - The kernel can only assign processes to processors -- Two threads within the same process cannot run simultaneously on two processors

Multithreading Models

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

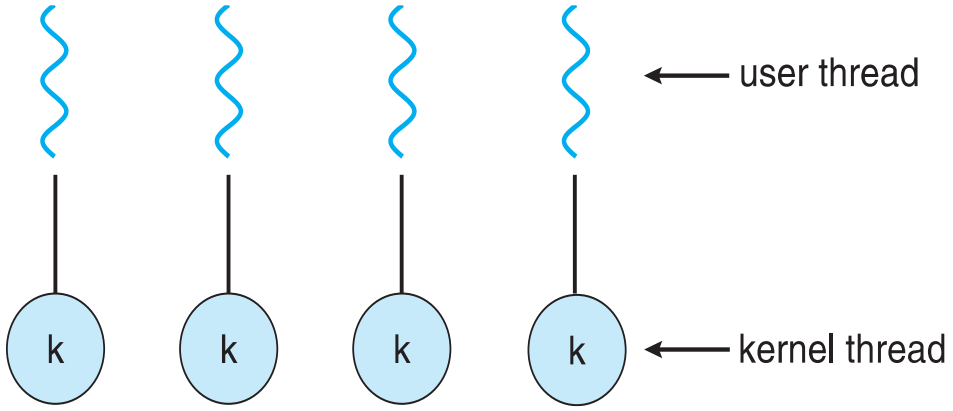
Many-to-One

- Many user-level threads mapped to single kernel thread
- Thread management is done by the thread library in user space, so it is efficient
- 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
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



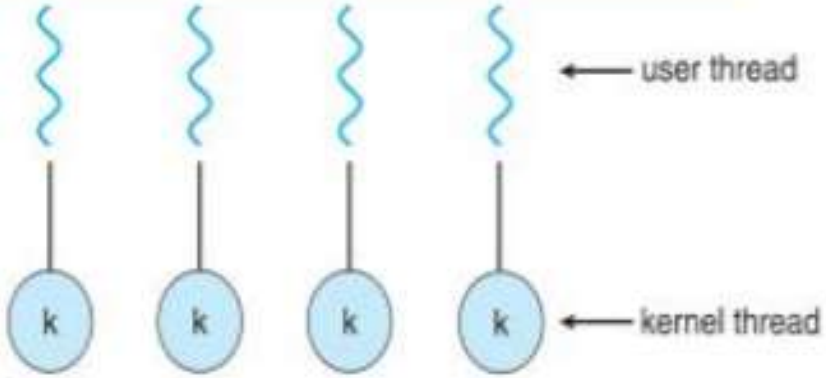
One-to-One

- 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



One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- **Advantages:**
  - More concurrency than many-to-one
- **Disadvantages:**
  - High overhead of creating kernel threads
  - Hence, number of threads per process sometimes restricted
- Examples
  - Windows
  - Linux

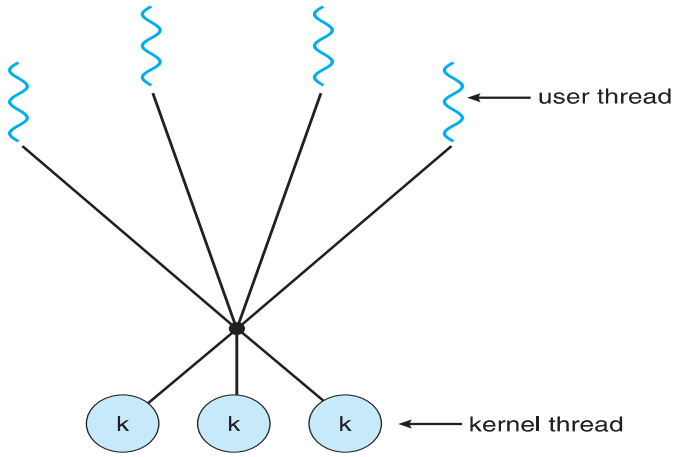


Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- The number of kernel threads may be specific to either a particular application or a particular machine

Examples

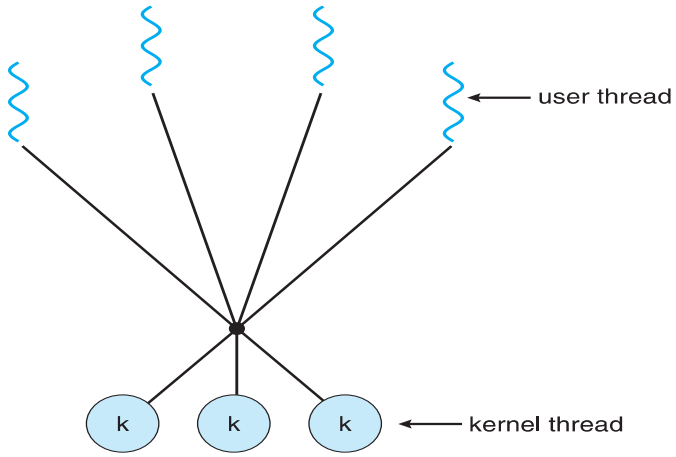
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



- Many-to-one
  - Any number of threads is allowed, but low concurrency
- One-to-one
  - Great concurrency, but the number of threads is limited
- Many-to-many
  - Gets rid of the shortcomings of the precious two

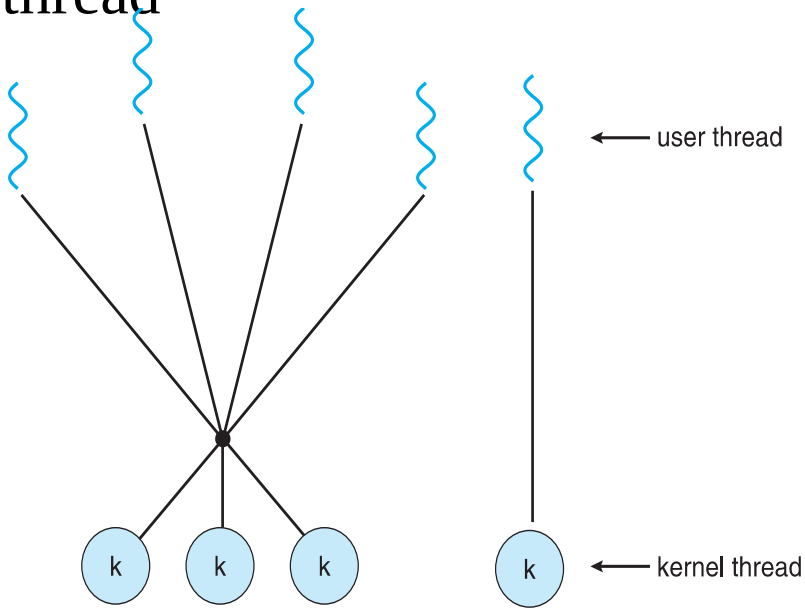
Many-to-Many Model

- In this model, the library maps all threads to a single lightweight process
- Advantages:
  - totally portable
  - easy to do with few systems dependencies
- Disadvantages:
  - cannot take advantage of parallelism
  - may have to block for synchronous I/O
  - there is a clever technique for avoiding it
- Mainly used in language systems, portable libraries



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

References

1. Silberschatz, Galvin, and Gagne, “Operating System Concepts”, Tenth Edition, Wiley India Pvt Ltd, 2018
2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2010.
3. William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011

Summarization