Chapter : Threads

Chapter : Threads

- Overview
- Benefits of Threads vs Processes
- Single and Multithreaded Processes
- Types of Threads
- Multithreading Models

Threads

- It is single sequential (flow of) execution of tasks of process
- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:
  - program counter
  - Thread id
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating-system resources

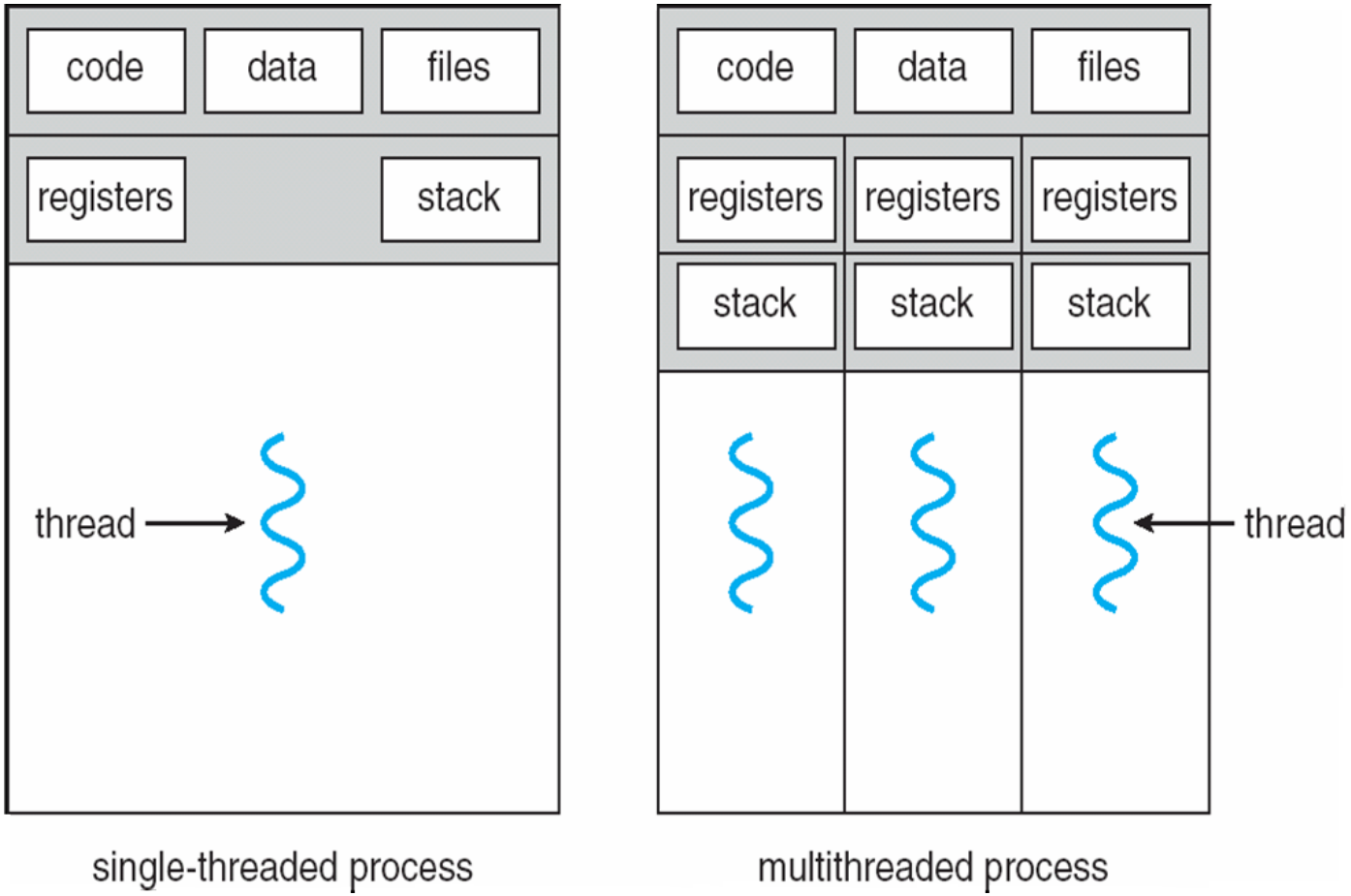
Threads (Cont.)

- In a multiple threaded task, while one thread is blocked and waiting, a second thread in the same task can run.
  - In web browser, one thread displays images, other text, other fetches data from network.
  - Word processor, graphics, response to keystrokes, spell check.
  - **Process = [Code + Data + Heap + Stack + PC + PCB + CPU Registers]**
  - **Thread = [Parent Code, Data and Heap + Stack, PC and CPU Registers]**

Benefits of Threads vs Processes

- **Less time to create a new thread** than a process, because the newly created thread **uses the current process address space**.
- **Less time to terminate a thread**
- **Less time to switch between two threads** - newly created thread uses the current process address space.
- Less communication overheads - **threads share everything**: address space, So, **data produced by one thread is immediately available to all the other threads.**

Single and Multithreaded Processes



Threads States

- Three key states: running, ready, blocked
- Termination of a process, terminates all threads within the process

Types of Threads

1. **User Level Threads**: (ULT)

- Threads of user application process.
- ULT are supported above the kernel and managed without kernel support
- These are implemented in user space in main memory. And managed bu user level library.
- The kernel is not aware of the existence of threads
- User Level Library is used for – Thread Creation, Scheduling and Management

User Level Threads

Advantages

•Thread switching does not require Kernel mode privileges.

•User level thread can run on any operating system.

•Scheduling can be application specific in the user level thread.

•User level threads are fast to create and manage.

Disadvantages

•In a typical operating system, most system calls are blocking.

•Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads(KLT)

- Threads of processes defined by OS itself
- KLT are supported and managed directly by OS.
- Kernel performs Thread Creation, Scheduling and Management in kernel space.
- No thread library but system calls to the kernel facility exists.

Kernel Level Threads(KLT)

- Kernel level threads are managed by the OS
- thread operations (ex. Scheduling) are implemented in the kernel code.
- Kernel level threads may favor thread heavy processes.
- they can also utilize multiprocessor systems by splitting threads on different processors or cores.
- If one thread blocks it does not cause the entire process to block.
- KLT are not portable because the implementation is operating system dependent.

Kernel Level Threads(KLT)

Advantages

•Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

•If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

•Kernel routines themselves can be multithreaded.

Disadvantages

•Kernel threads are generally slower to create and manage than the user threads.

•Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

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

One-to-One

- Each user-level thread maps to one kernel thread
- The one-to-one model associates a single user-level thread to a single kernel-level thread.
- kernel level threads follow the one to one model

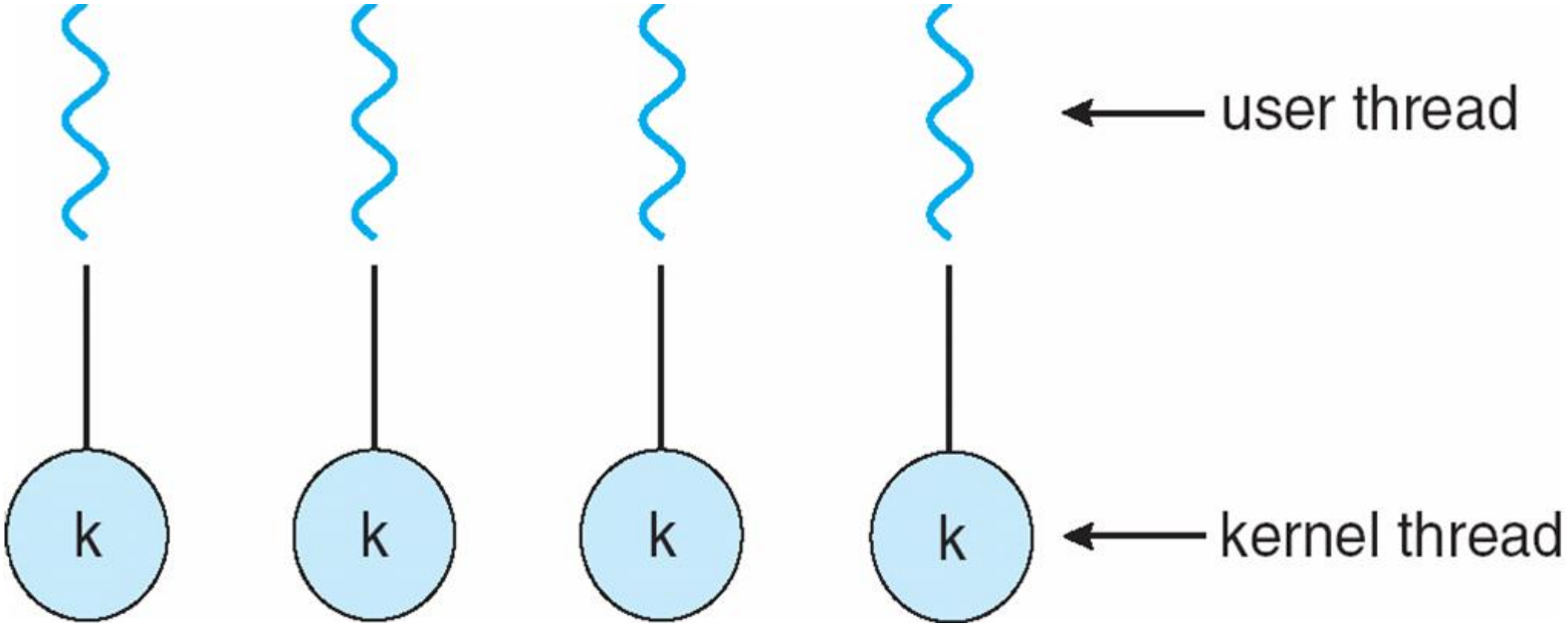
Advantage:

facilitates the running of multiple threads in parallel.

Drawback:

Generation of every new user thread must include the creation of a corresponding kernel thread causing an overhead

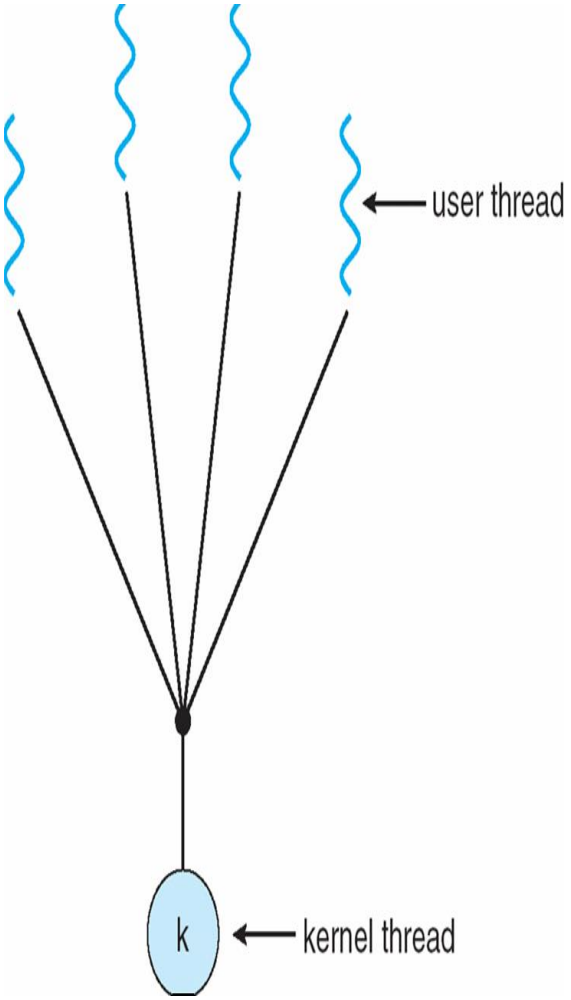
One-to-one Model



The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application

Many-to-One

- The many-to-one model associates all user- level threads to a single kernel-level thread.
- User level threads follow the many to one threading model.
- This means multiple threads managed by a library in user space but the kernel is only aware of a single thread of the process owning these threads.



Many-to-Many Model

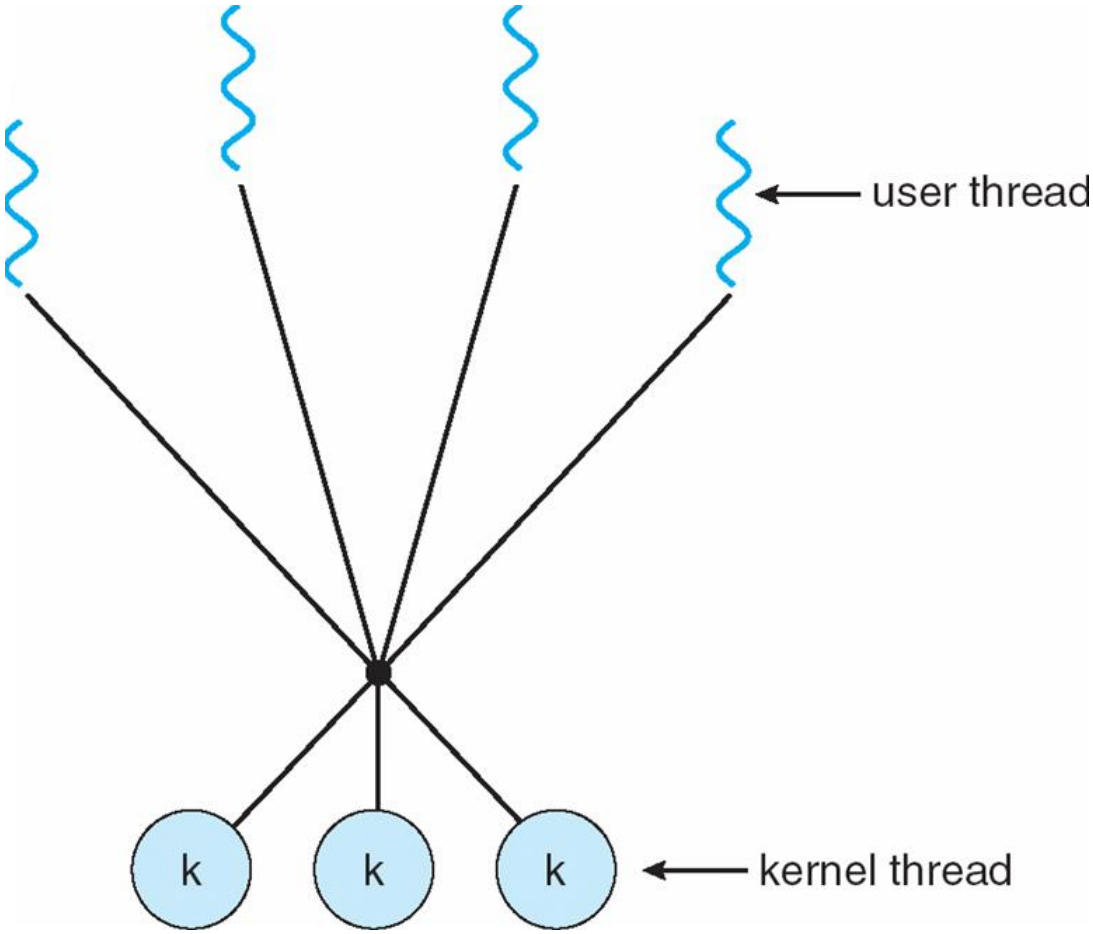
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Many-to-Many Model

- A number of user-level threads are associated to an equal or smaller number of kernel-level threads.

- Allows many user level threads to be mapped to many kernel threads

Many-to-Many Model



Thread Cancellation

- **Thread cancellation** is the task of terminating a thread before it has completed.
  - For example, if multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

End of Chapter