The benefits of multithreaded programming can be broken into four major categories :

- **Responsiveness –** may allow a program to continue running even if part of it is blocked or is performing a lengthy operation , especially important for user interfaces thereby increasing responsiveness to the user
- **Resource Sharing –** By default threads share memory and resources of process to which they belong. The benefit of sharing code and data is that it allows an application to have several threads of activity within the same address space
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching. Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
- **Utilization of multiprocessor architecture–** The benefits of multithreading can be greatly increased in a multiprocessor architecture , where threads may be running in parallel on different processors. A single threaded process can only run on one CPU, no matter how many are available. Multithreading on multiple CPU machine increases concurrency.

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

There are three main thread libraries in use today:

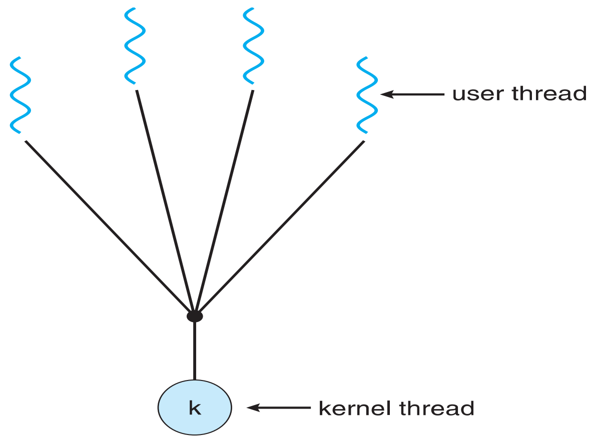
1. **POSIX Pthreads** - may be provided as either a user or kernel library, as an extension to the POSIX standard.
2. **Win32 threads** - provided as a kernel-level library on Windows systems.
3. **Java threads** - Since Java generally runs on a Java Virtual Machine, the implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either Pthreads or Win32 threads depending on the system.

Multithreading models :

I. Many to one –

- Multiple user-level threads mapped to single kernel level thread
- When a user thread makes a blocking system call entire process blocks.
- Multiple threads may not run in parallel on multi-core system because only one may be in kernel at a time
- Few systems currently use this model

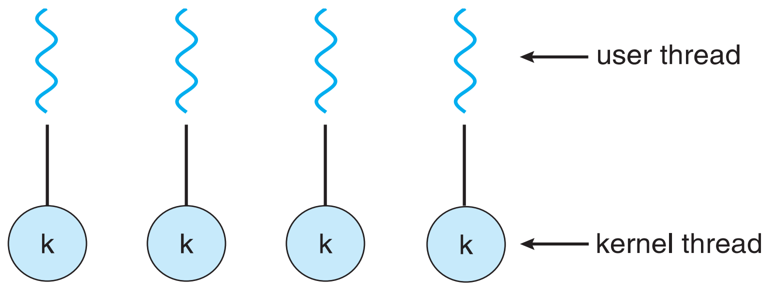
  Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



II. One- to –one

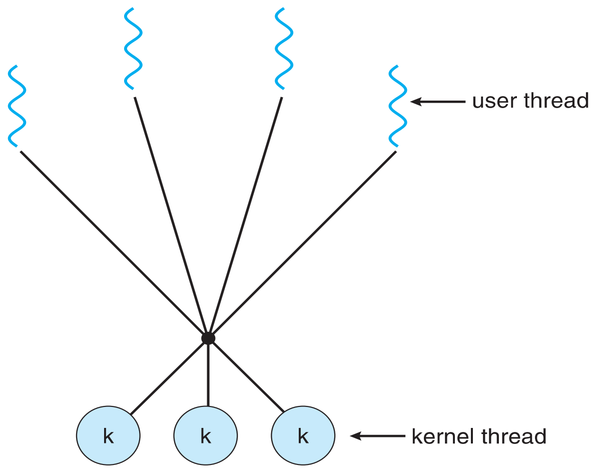
- Here there is a one to one relationship between kernel and user thread.

- Creating a user-level thread requires creating the corresponding kernel thread
  - In this model multiple threads can run on multiple processors.
    Examples- Windows, Linux



III. Many – to- many

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads


user thread
k
k
k
kernel thread