# *Multithreading Programming*

# *Some Examples of Thread : In General Layman terms*

- *Let us take an example of a human body. A human body has different parts having different functionalities which are working parallelly ( Eg: Eyes, ears, hands, etc.).*
- *Similarly in computers, a single process might have multiple functionalities running parallelly where each functionality can be considered as a thread.*

# *Some Examples of Thread : Google Chrome*

*Imagine you're using Google Chrome to open a few different websites at the same time. Each open website tab can be thought of as a separate thread. Each thread (tab) is responsible for loading and displaying the content of its specific website. When you're watching a video on one tab while reading an article on another, these actions are managed by different threads, like separate workers.*

*If one tab gets stuck or takes too long to load, it won't affect the other tabs. This is because each tab (thread) works independently. If Chrome didn't use threads and tried to do everything in one go, opening multiple websites could slow down the whole browser, and a problem with one tab could crash the entire browser.*

# *Some Examples of Thread : Microsoft Office*

*Now let's consider Microsoft Word from the Office suite. Imagine you're working on a document and, at the same time, you want to print it. Microsoft Word could use threads to handle these tasks separately.*

*One thread could be responsible for you typing and editing the document. Another thread could be in charge of formatting the document so it looks nice on the screen. Yet another thread could be handling the printing process.*

*Threads in Microsoft Office help different tasks within the software run smoothly and independently.*

*Just like with Google Chrome, if something goes wrong with one thread, it doesn't necessarily crash the entire program because threads keep things separate.*

# *Some Examples of Thread : Microsoft Office*

*Various threads are present in MS office like:*

- ***User Interface Thread:*** *It's responsible for displaying menus, buttons, text, and other visual elements.*
- ***Editing Thread:***
- ***Background Tasks Thread:***
- ***Printing Thread:***
- ***File Handling Thread:*** *Open save*
- ***Rendering Thread:*** *visual elements like slides and animation*

# *Some Examples of Thread : Conclusion*

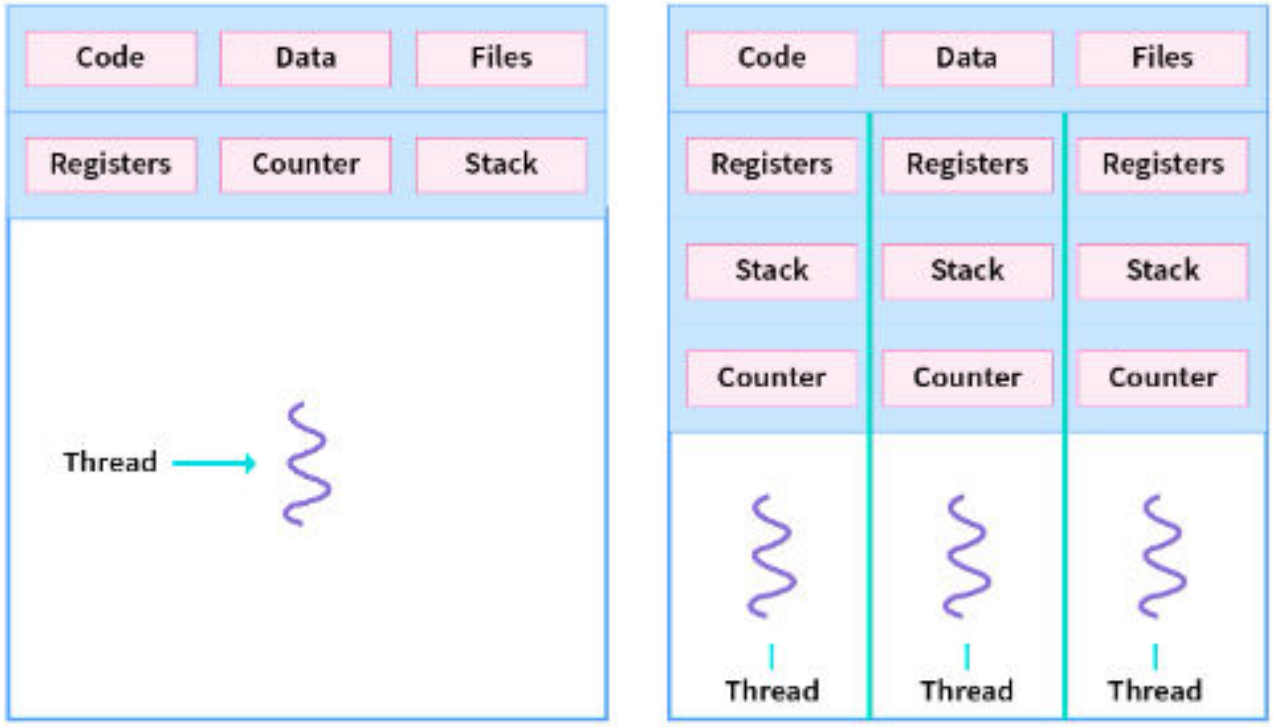
In both examples, threads allow different parts of the software to work together effectively without getting in each other's way. They help with multitasking, responsiveness, and overall smoother user experiences.

# Thread

- *A process can be further divided into independent units known as threads.*
- *A thread is a single sequence stream within a process so we can call it as light-weight process within a process.*
- *Thread is a basic unit of CPU utilization.*
- *Threads are popular way to improve application through parallelism.*
- *There can be multiple threads in a single process having the same or different functionality*

# Differences between process and thread

- *Threads are not independent from each other unlike processes. As a result, threads shares with other threads their code section, data section and OS resources like open files and signals.*
- *But, like processes, a thread has its own program counter (PC), a register set, and a stack space.*
- *If a process has multiple threads of control it can perform more than one task at a time.*

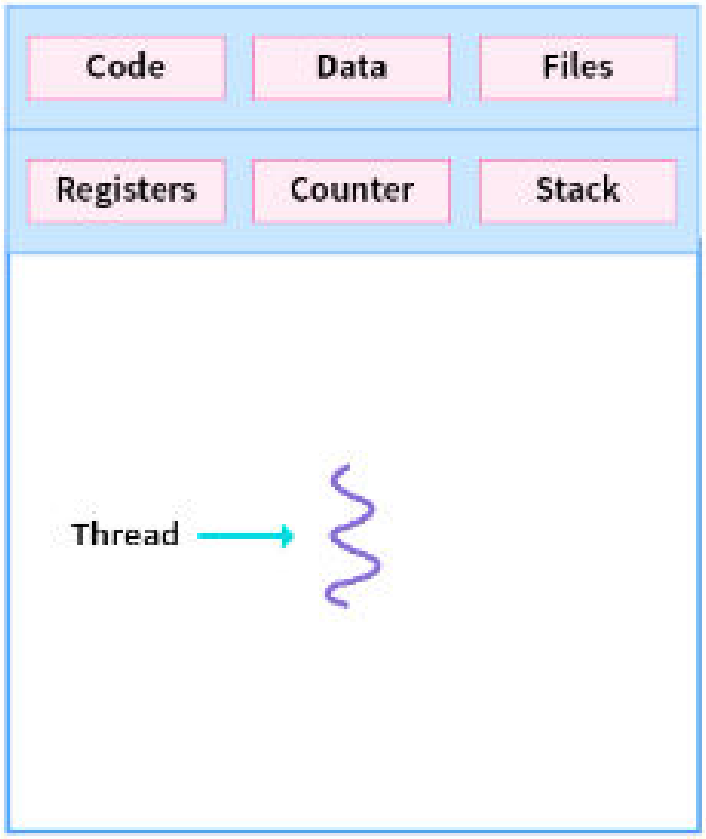



Single-threaded process

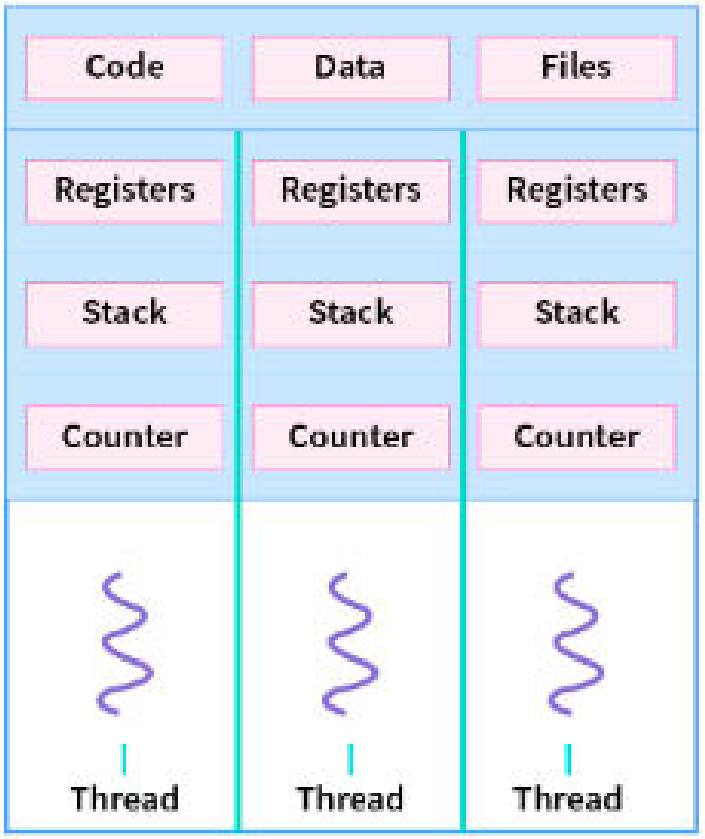
Multithreaded process

- *A traditional heavy weight process has a single thread of control.*

# Differences between process and thread




Single-threaded process




Multithreaded process

# *Why do we need Threads*

- *Since threads use the same data and code, the operational cost between threads is low.*
- *Creating and terminating a thread is faster compared to creating or terminating a process.*
- *Context switching is faster in threads compared to processes.*
- *Communication between threads is faster.*

# *Multithreading*

- *In Multithreading, the idea is to divide a single process into multiple threads instead of creating a whole new process.*
- *Multithreading is done to achieve parallelism and to improve the performance of the applications as it is faster in many ways.*
- ***Advantages:** Resource Sharing, Responsiveness, Economy*

# Multithreading: Benefits

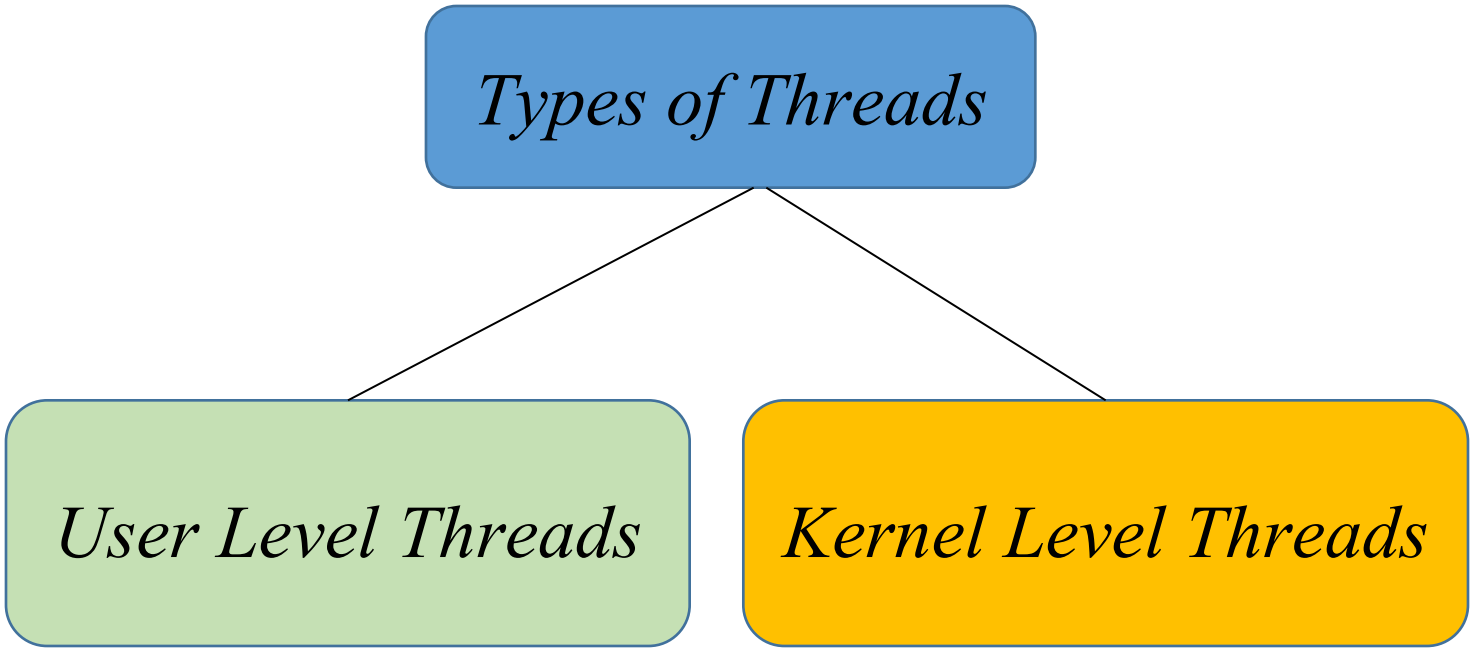
| | |
|---|---|
| **Responsiveness** | *One thread may execute while other threads are blocked or slowed down doing intensive calculations.* |
| **Resource sharing** | *By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.* |
| **Economy** | *Creating and managing threads ( and context switches between them ) is much faster than performing the same tasks for processes.* |

# *Multithreading: Benefits*

***Utilization of multiprocessor architectures***

*The benefits of multithreading can be greatly increased in multiprocessor architecture, where threads may be running in parallel on different processors. (True parallelism)*

# Types of Thread



# *Types of Thread: User Level Threads*

- *These are the threads that application programmers would put into their programs, managed entirely by the application or program without direct intervention from the operating system kernel.*
- *The thread management functions (like creation, scheduling, synchronization) are provided by a user-level thread library, usually in the application code.*
- *Thread switching does not need to call OS and to cause interrupt to Kernel.*

# *Types of Thread: Kernel Level Threads*

- *Kernel-level threads are managed by the operating system kernel. The kernel provides thread management services and is aware of all the threads in the system.*
- *Provide system calls to create and manage threads from user space.*

# *Multithreading Models*

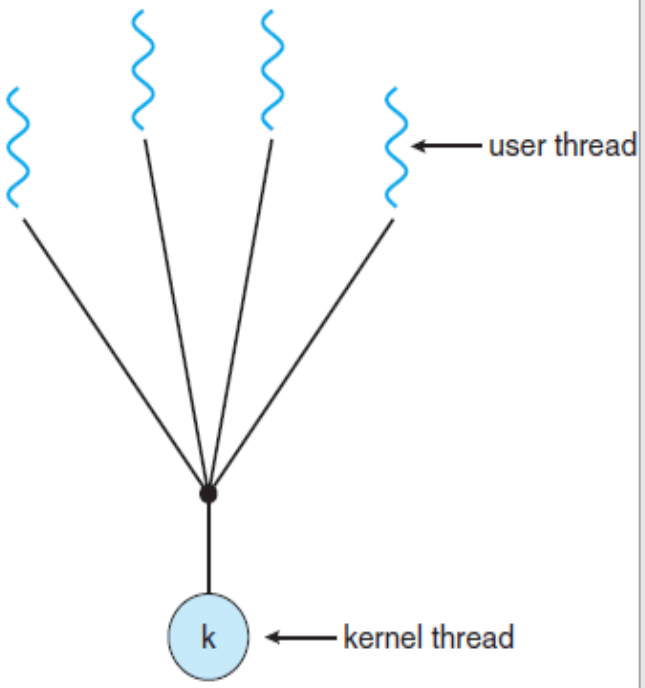
- *There must exist a relationship between user threads and kernel threads. The user threads must be mapped to kernel threads,*

*There are three common ways of establishing this relationship.*

***1. Many to One Model*** ***2. One to One Model***

***3. Many to Many Model***

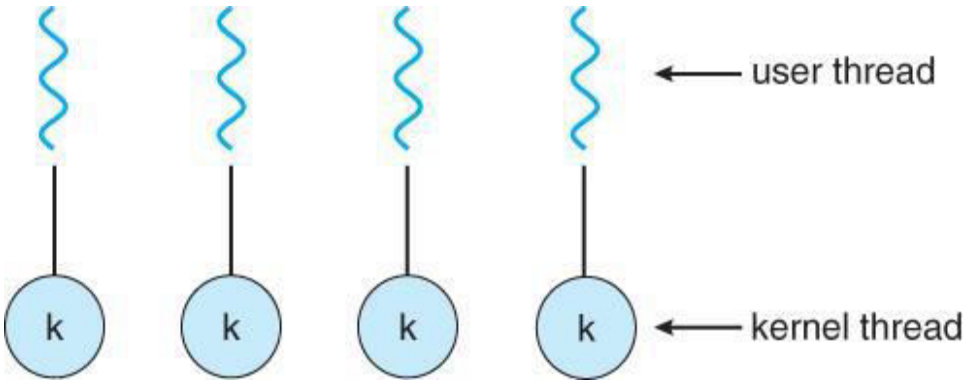
# Many to One Model




- *Maps many user-level threads to one kernel level thread.*
- *Thread management is done by the thread library in user space so it is efficient.*
- *The entire process will block if a threads makes a blocking system call.*
- *Because only one thread can access the kernel at one time multiple threads are unable to run in parallel on multiprocessors.*

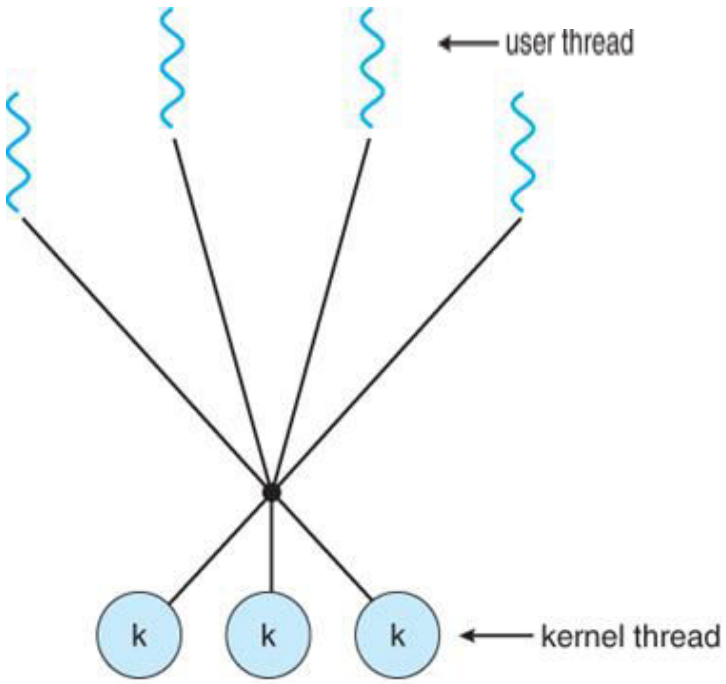
# One to One Model

- *Maps each user-level-thread to a kernel- level-thread.*
- *Provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call.*
- *Allow multiple threads to run in parallel on multiprocessors.*




- *Creating a user thread requires creating the corresponding kernel thread involving more overhead and slowing down the system*
- *Place a limit on how many threads can be created.*

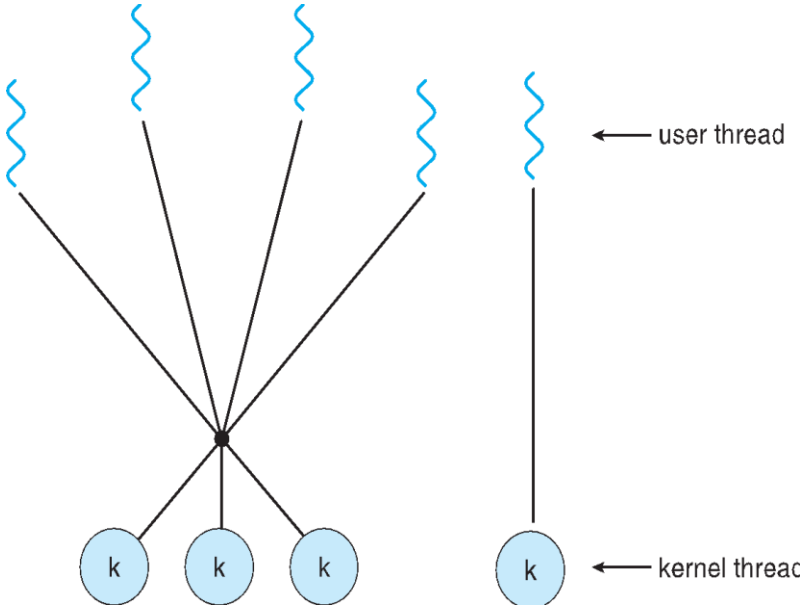
# Many to Many Model




- *Multiplexes any number of user threads onto an equal or smaller number of kernel threads.*
- *The number of kernel level threads may be specific to either a particular application or a particular machine.*
- *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 make a blocking system call kernel can schedule another thread for execution.*

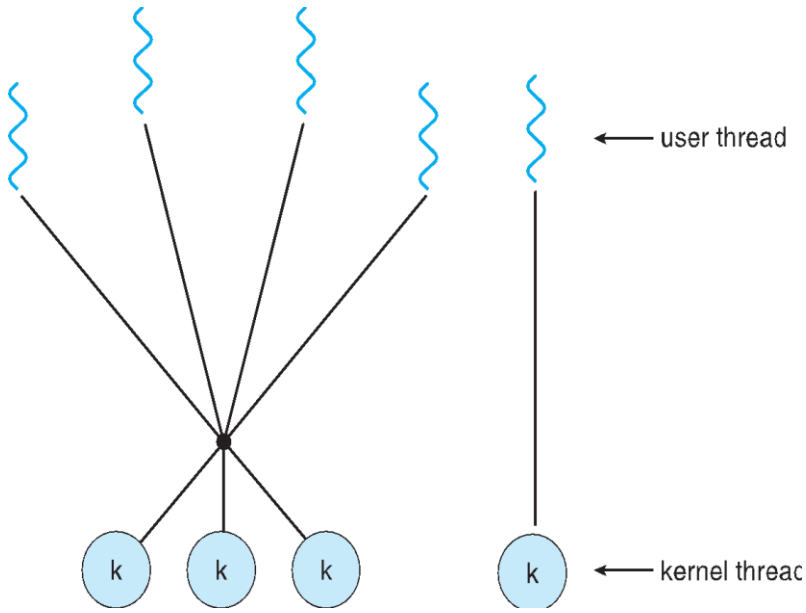
# Two-level Model

- 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




# 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



# *Example of Multithreading in C*

```
#include <stdio.h>
#include <pthread.h>    //POSIX Library

void *thread_function1(void *arg) {sleep(3)
     printf("\n Hello from thread Function1");
    return NULL;
}

void *thread_function2(void *arg) {
     printf("\n Hello from thread Function2");
    return NULL;
}

int main() {
    pthread_t thread1, thread2;  // Thread identifiers

    // Create two new threads
    pthread_create(&thread1, NULL, thread_function1, NULL);
    pthread_create(&thread2, NULL, thread_function2, NULL);

    // Wait for both threads to finish
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("\n Both threads have finished.\n");
    return 0;
}
```