

Importance of Threads in OS

Why Thread?

- Multiple tasks with the application can be implemented by separate threads
 - Update display
 - Fetch data
 - Spell checking
 - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Process and Thread

- A process/task = a unit of resource ownership, used to group resources together
- Processes have two characteristics:
 - **Resource ownership** - process includes a virtual address space to hold the process image
 - **Scheduling/execution** - follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

What is Thread?

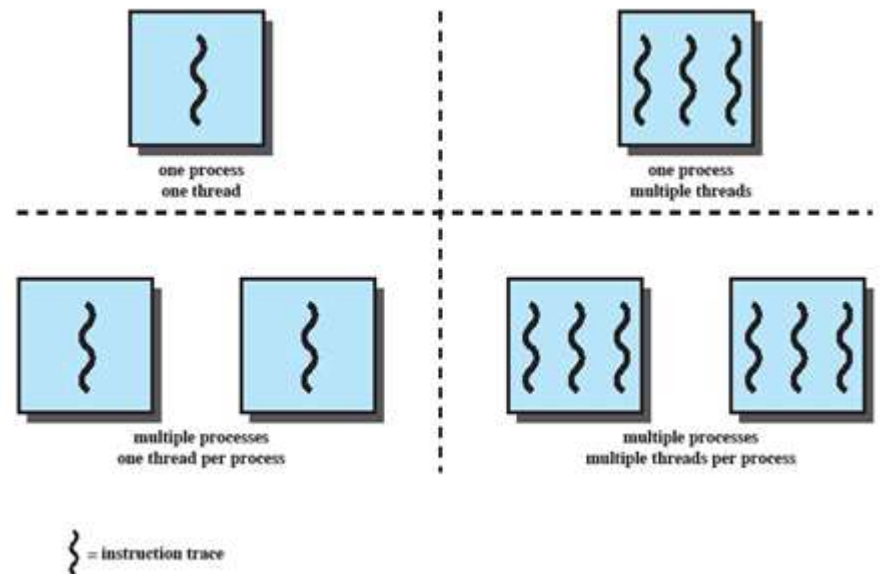
- Wikipedia: A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.
- A Thread = a unit of dispatching/scheduling, scheduled for execution on the CPU
- “Processes within processes”
- “Lightweight processes

Processes and Threads

- Processes share devices: CPU, Disk, Memory, Printer, etc.
- Threads share resources: Memory space, file pointers, etc
- Processes own: Threads, Memory space, file pointers, etc
- Threads own: PC, registers, Stack, etc.

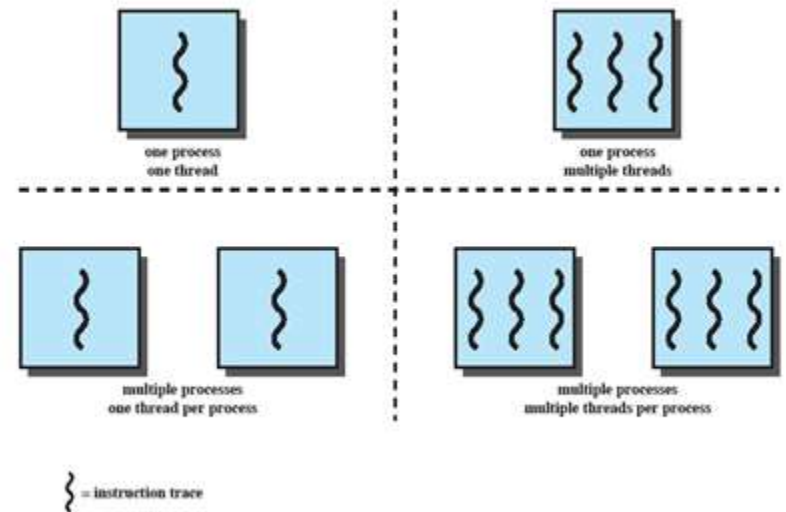
Multithreading

- The ability of an OS to support multiple, concurrent paths of execution within a single process.



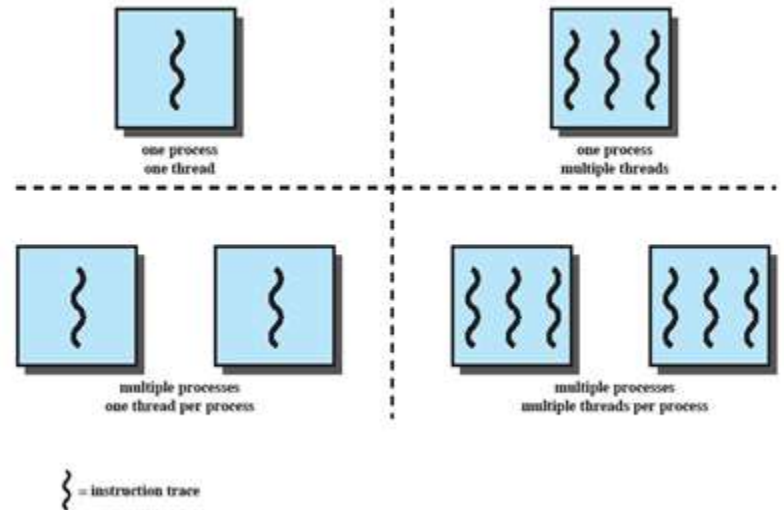
Single Thread Approaches

- MS-DOS supports a single user process and a single thread.
- Some UNIX, support multiple user processes but only support one thread per process



Multithreading

- Java run-time environment(JRE) is a single process with multiple threads
- Multiple processes *and* threads are found in Windows, Solaris, and many modern versions of UNIX



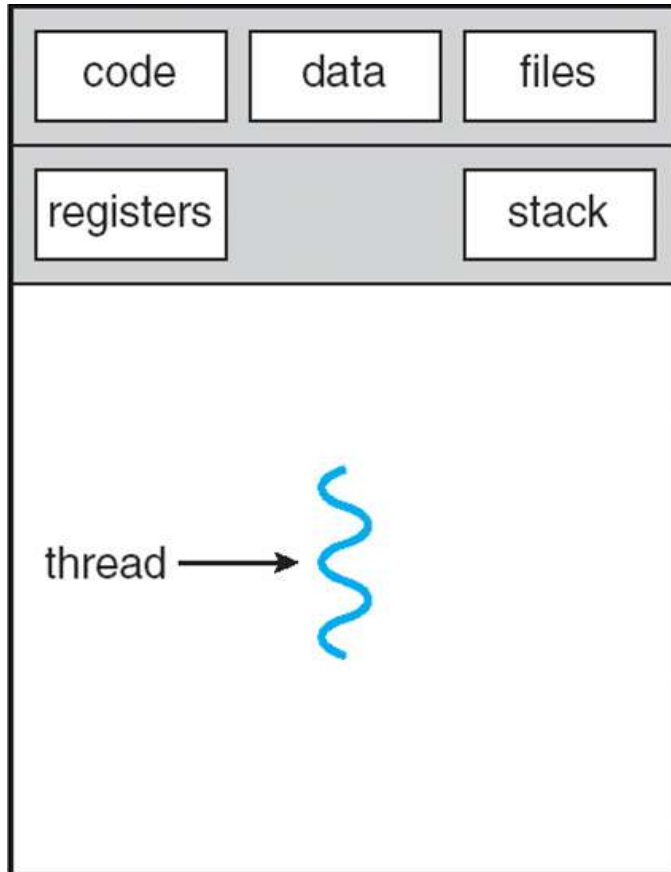
Threads in Process

- Each thread has
 - An execution state (running, ready, etc.)
 - Saved thread context when not running
 - An execution stack
 - Some per-thread static storage for local variables
 - Access to the memory and resources of its process
(all threads of a process share this)

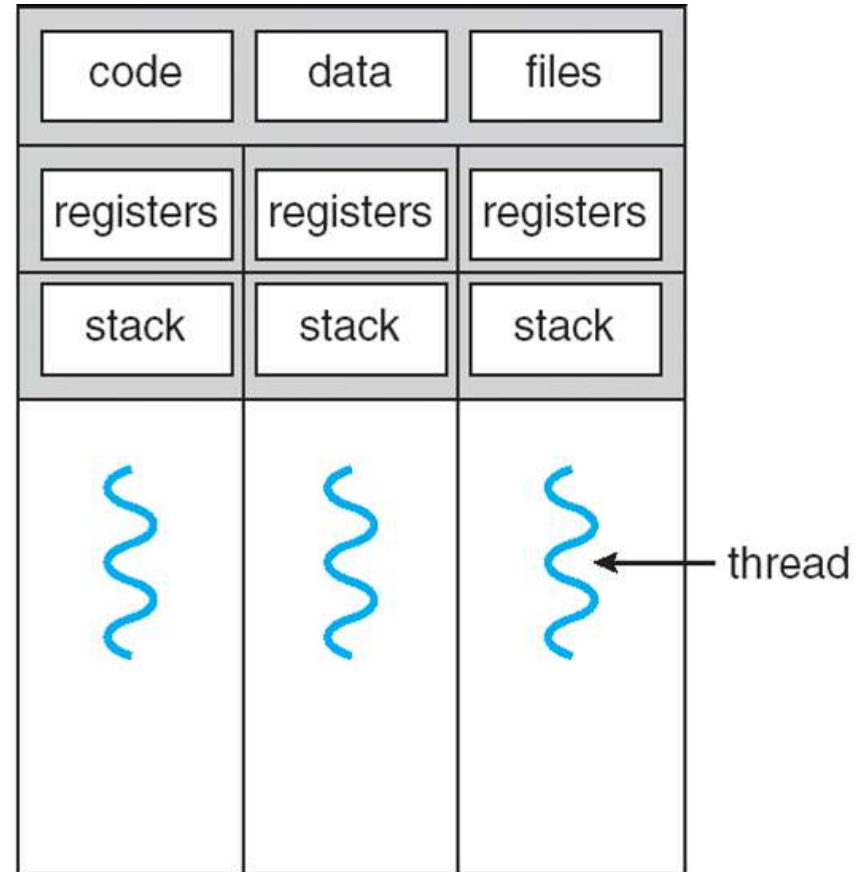
One view...

- *One way to view a thread is as an independent program counter operating within a process.*

Threads Vs Processes

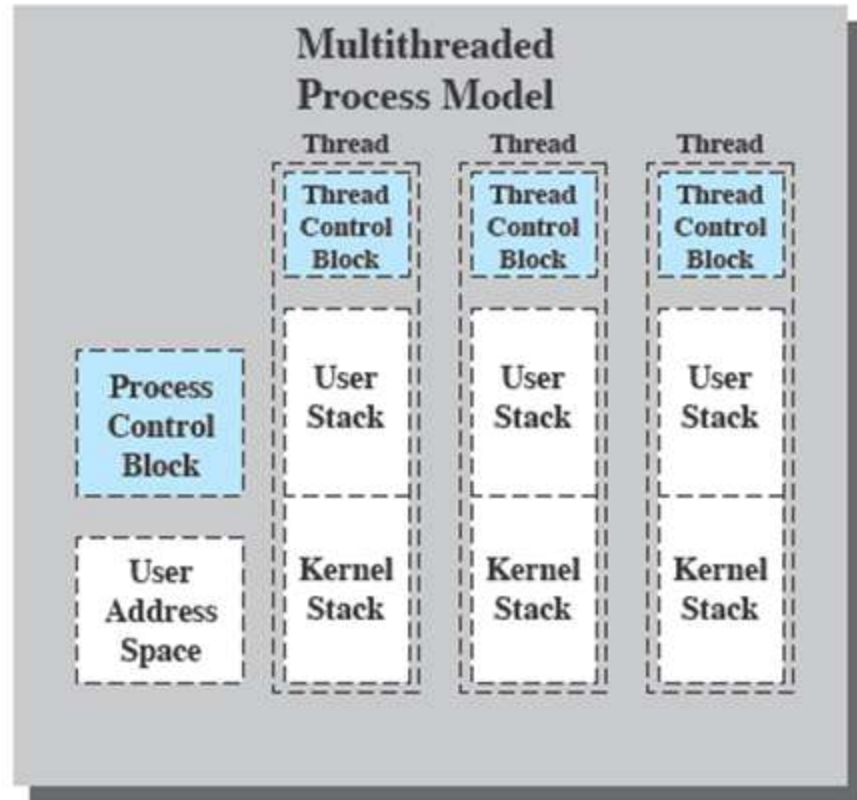
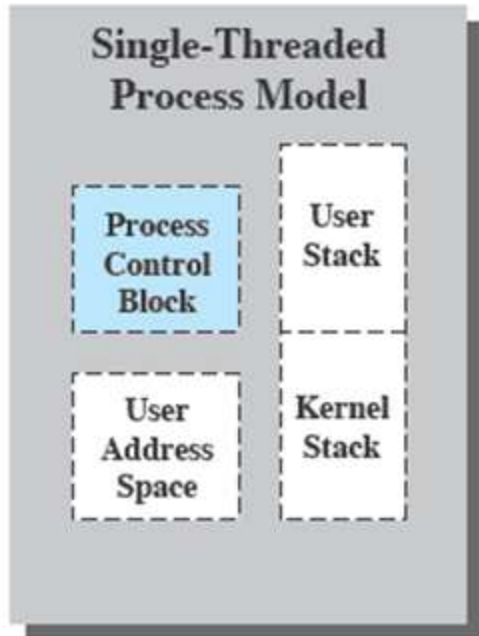


single-threaded process



multithreaded process

Threads vs. processes



Threads vs. processes

- **In a single-threaded process model, the representation of a process includes**
 - its process control block
 - user address space,
 - user and kernel stacks to manage the call/return behaviour of the execution of the process.
- **In a multithreaded environment,**
 - single process control block and user address space associated with the process,
 - separate stacks for each thread,
 - as well as a separate control block for each thread containing register values, priority, and other thread-related state information.

Thus, all of the threads of a process share the state and resources of that process.

- They reside in the same address space and have access to the same data.
- When one thread alters an item of data in memory, other threads see the results if and when they access that item.
- If one thread opens a file with read privileges, other threads in the same process can also read from that file.

Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Switching between two threads takes less time than switching processes
- Threads can communicate with each other
 - without invoking the kernel
- If there is an application or function that should be implemented as a set of related units of execution, it is far more efficient to do so as a collection of threads - rather than a collection of separate processes.
- Responsiveness
- Resource Sharing
- Economy
- Scalability

Thread use in a Single-User System

- Foreground and background work – In spreadsheet, display/update result of formula
- Asynchronous processing – To protect against power failure in word processor, write from memory to disk/input
- Speed of execution – divide work
- Modular program structure – various sources of input/output

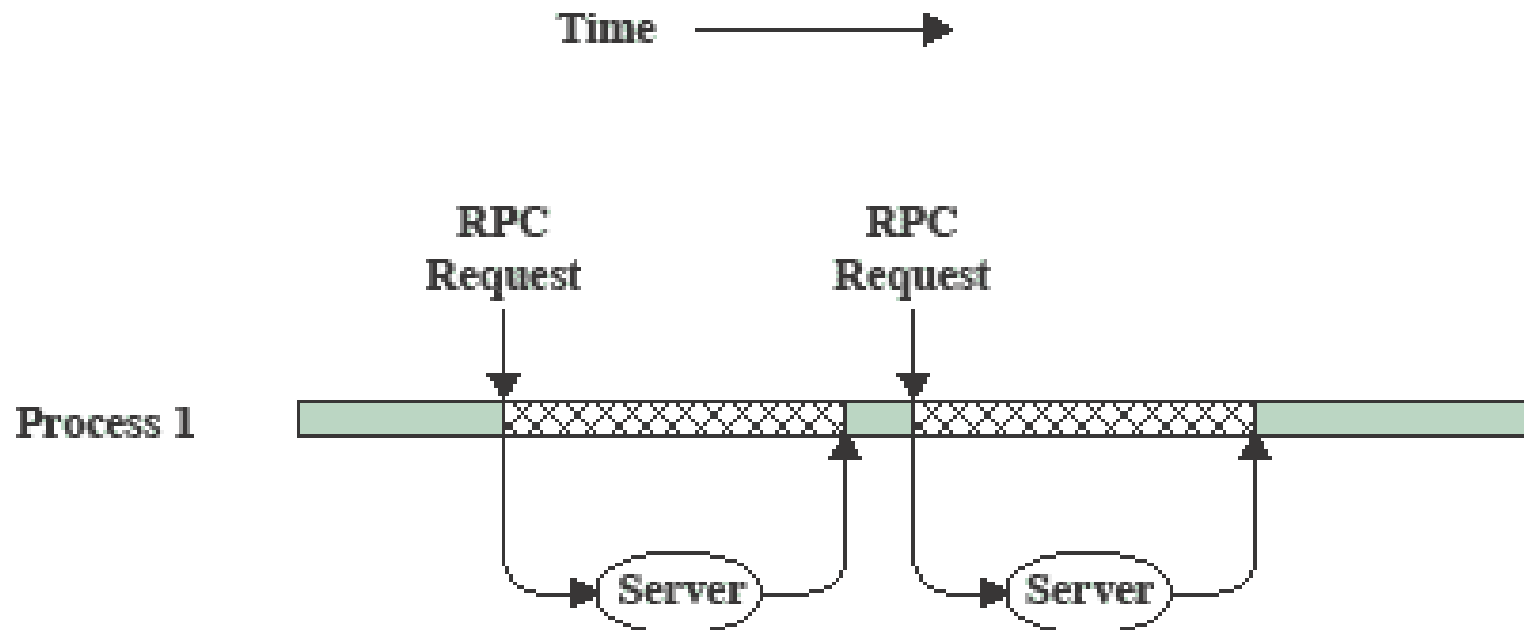
Thread Execution States

- States associated with a change in thread state
 - Spawn (another thread)
 - Block
 - Issue: will blocking a thread block other, or *all*, threads
 - Unblock
 - Finish (thread)
 - Deallocate register context and stacks

Example: Remote Procedure Call

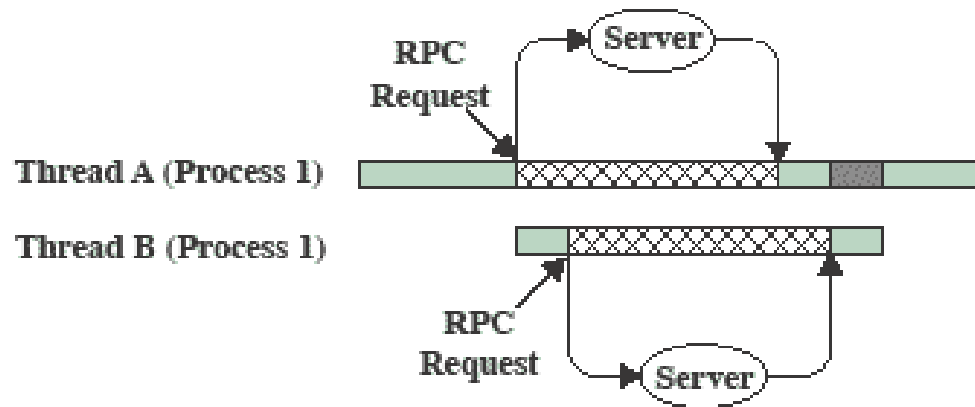
- Consider:
 - A program that performs two remote procedure calls (RPCs)
 - to two different hosts
 - to obtain a combined result.

RPC Using Single Thread






(a) RPC Using Single Thread

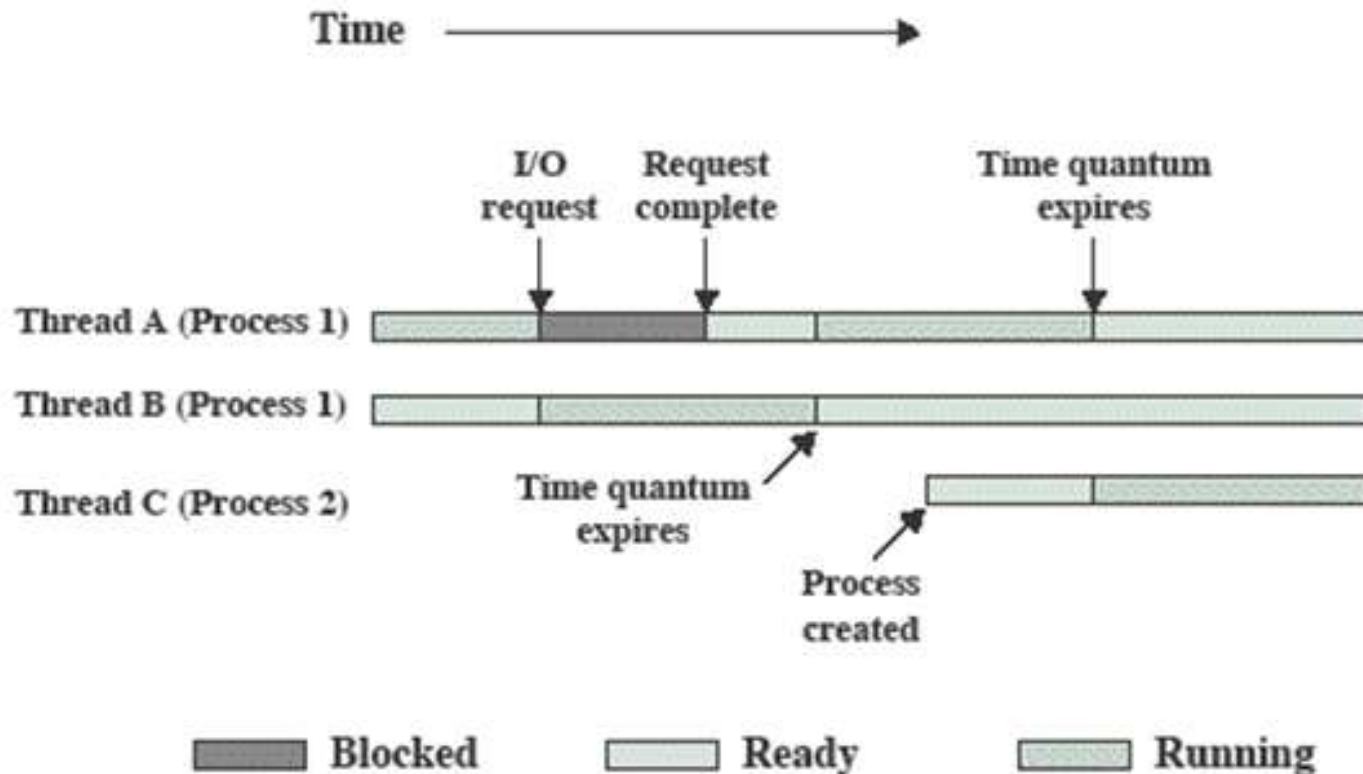
RPC Using One Thread per Server



(b) RPC Using One Thread per Server (on a uniprocessor)

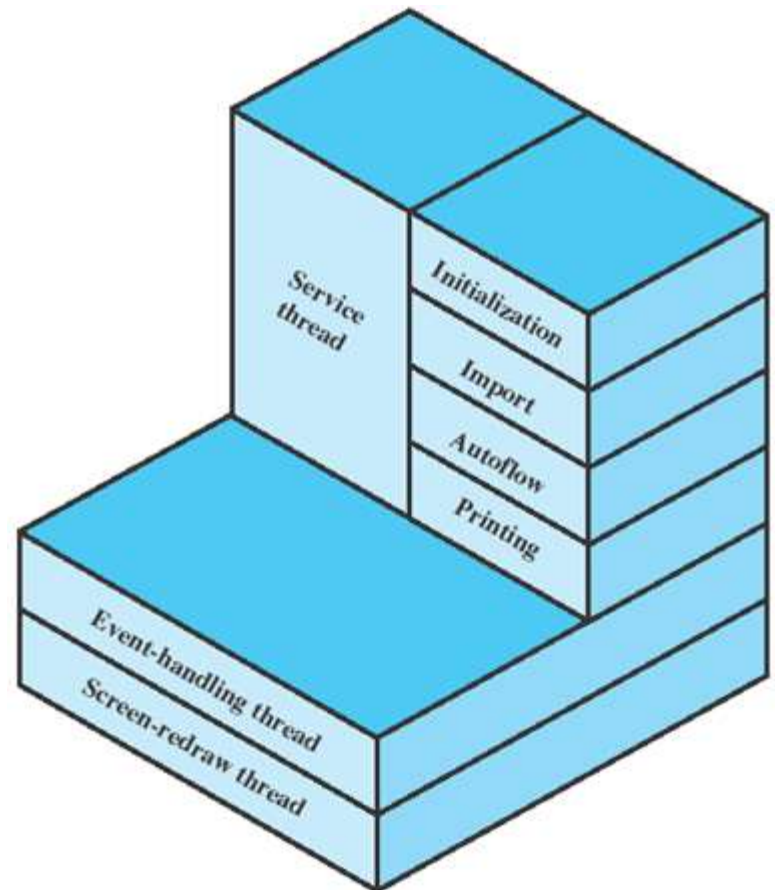
-  Blocked, waiting for response to RPC
-  Blocked, waiting for processor, which is in use by Thread B
-  Running

Multithreading on a Uniprocessor



Example: Adobe PageMaker

- an event-handling thread
- a screen-redraw thread
- a service thread

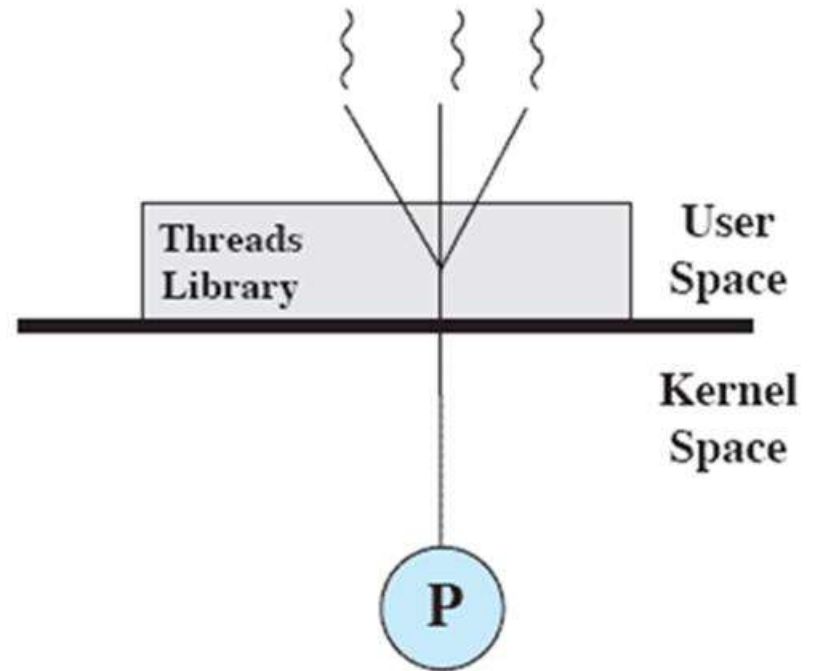


Thread Implementation

- User Level Thread (ULT)- Thread management done by user-level threads library:
 - POSIX Pthreads
 - Win32 threads
 - Java threads
- Kernel level Thread (KLT) also called kernel-supported threads/lightweight processes
 - Windows XP/2000
 - Solaris
 - Linux
 - Tru64 UNIX
 - Mac OS X

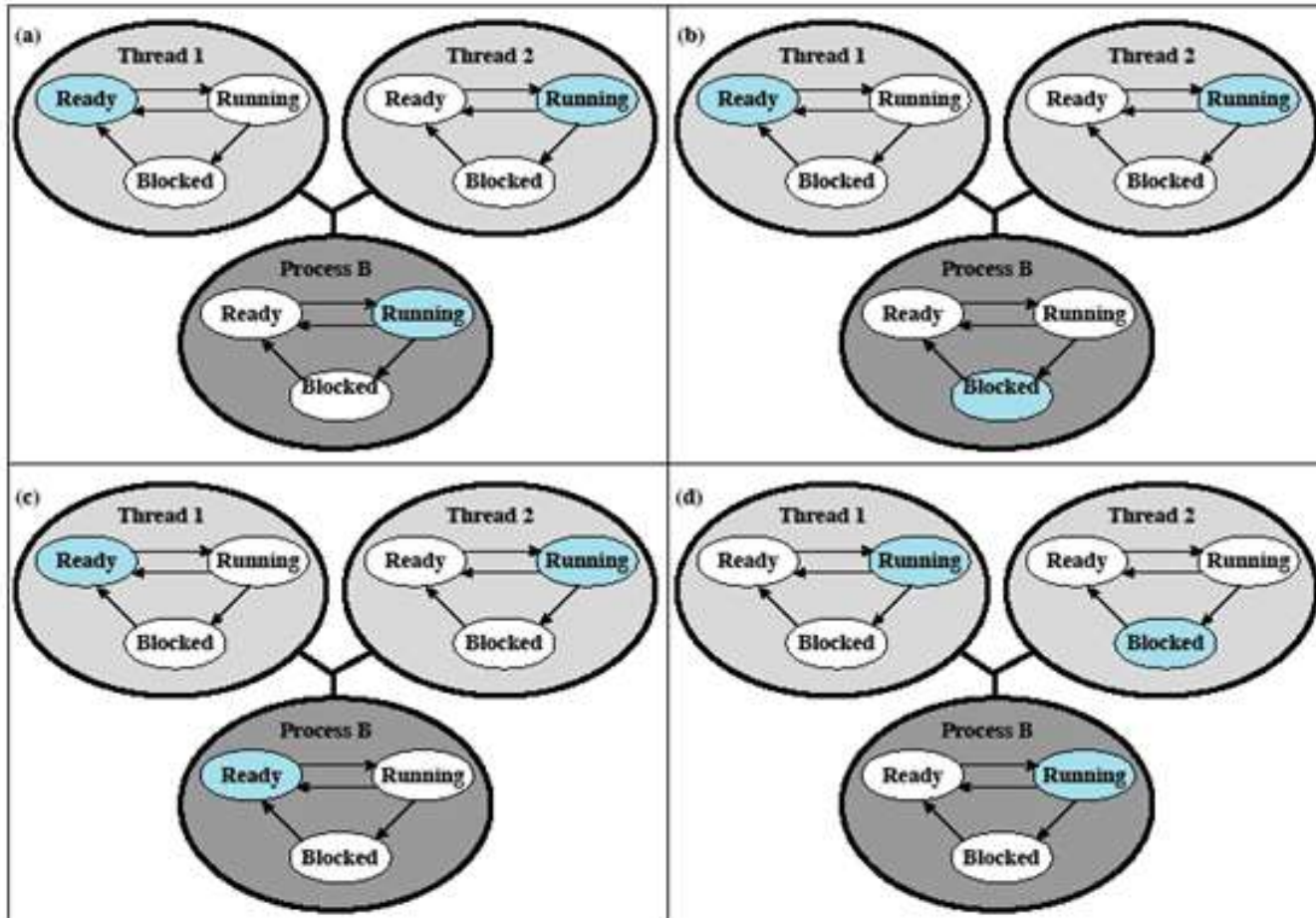
User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads



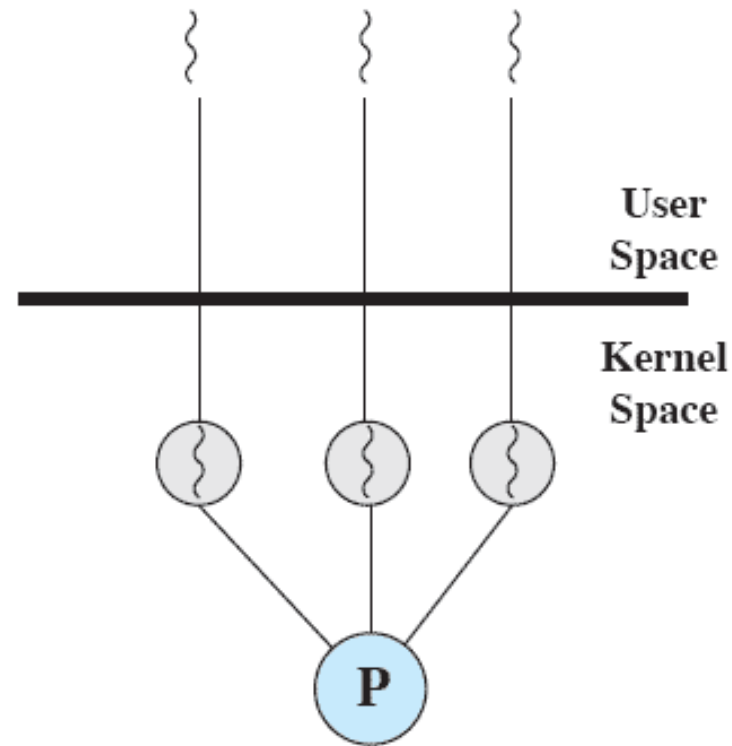
(a) Pure user-level

Relationships between ULT Thread and Process States



Colored state
is current state

Kernel-Level Threads



(b) Pure kernel-level

- Kernel maintains context information for the process and the threads
 - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach

Advantages of KLT

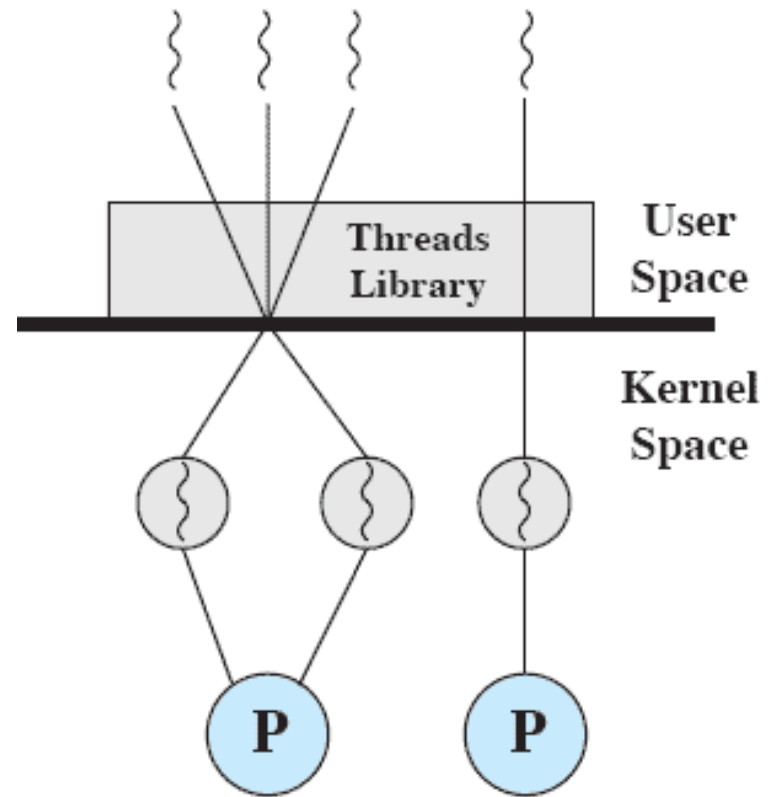
- The kernel can simultaneously schedule multiple threads from the same process on multiple processors.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantage of KLT

- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

Combined Approaches

- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads by the application
- Example is Solaris



(c) Combined

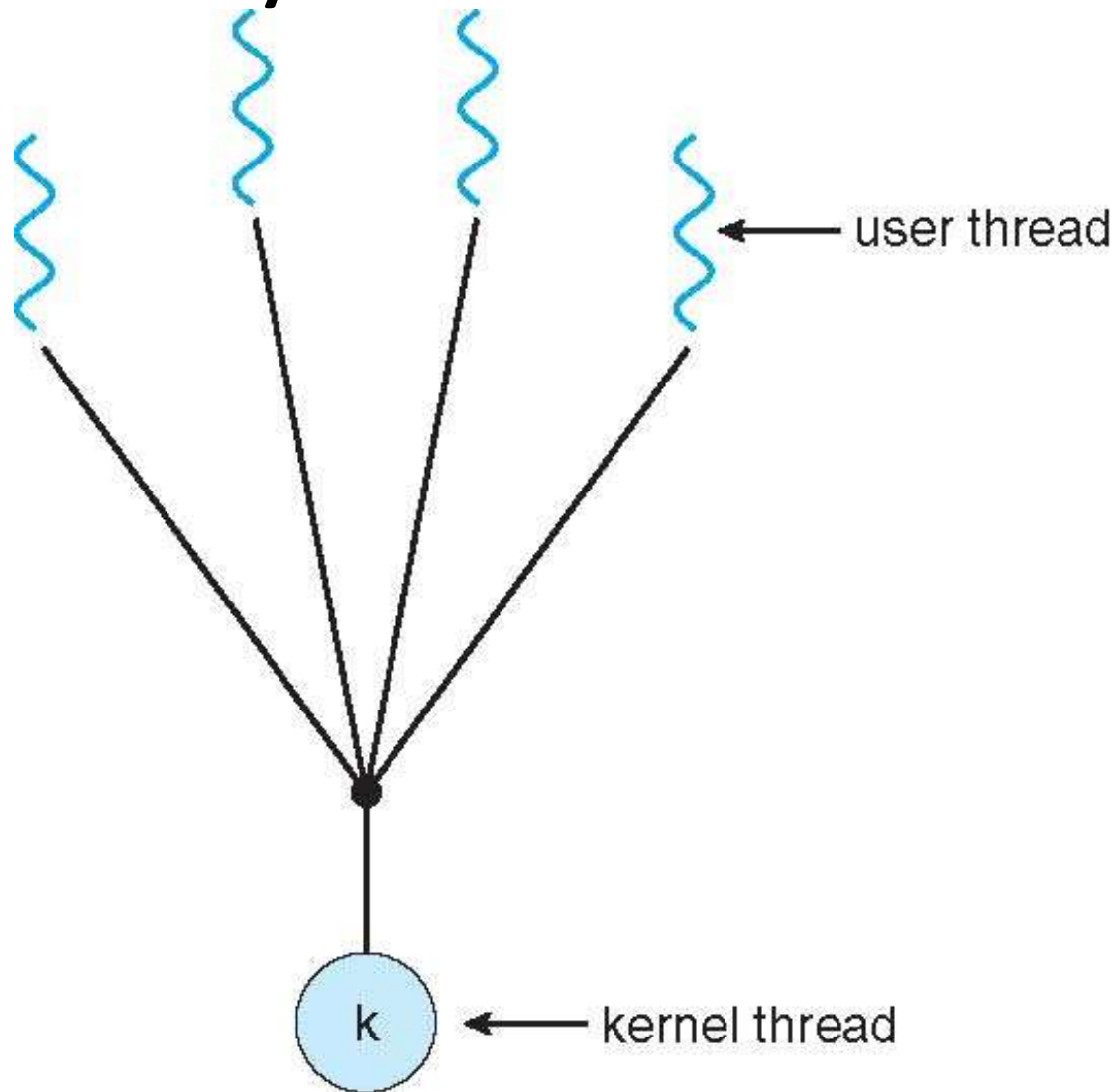
Multithreading Models

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

Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
 - **Solaris Green Threads**
 - **GNU Portable Threads**

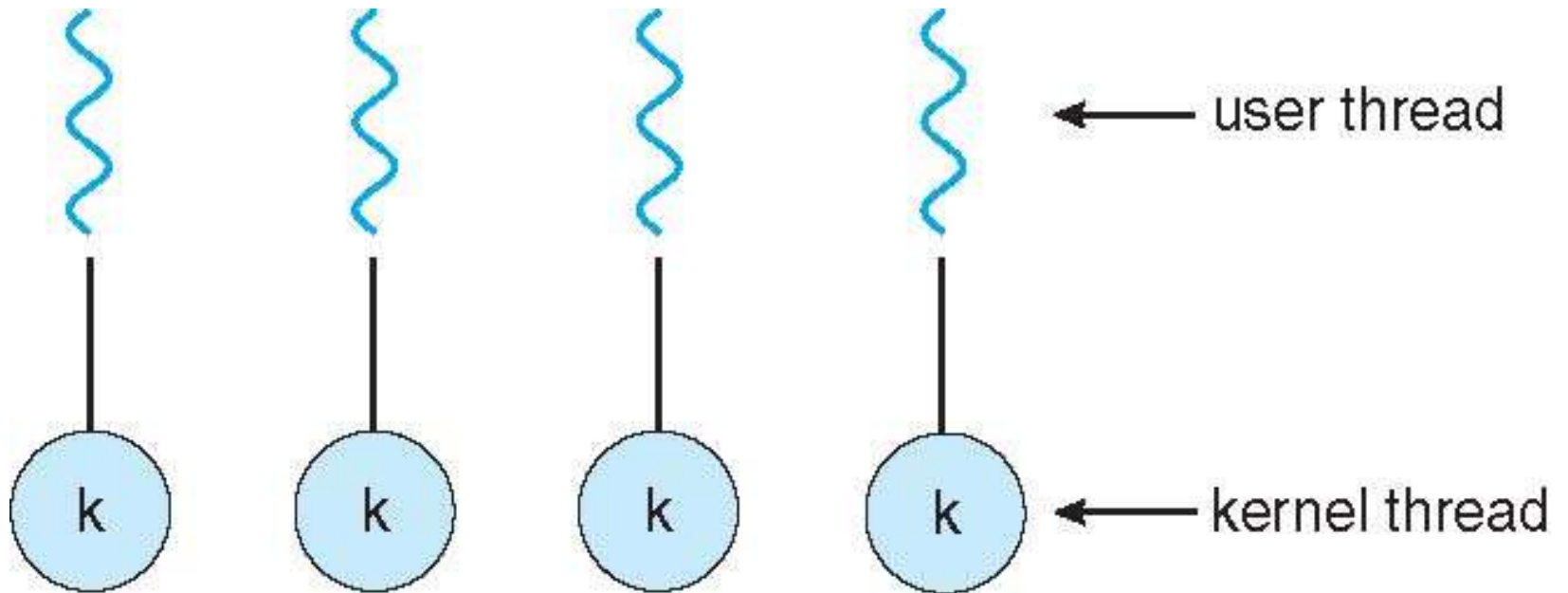
Many-to-One Model



One-to-One

- Each user-level thread maps to kernel thread
- Examples
 - Windows NT/XP/2000
 - Linux
 - Solaris 9 and later

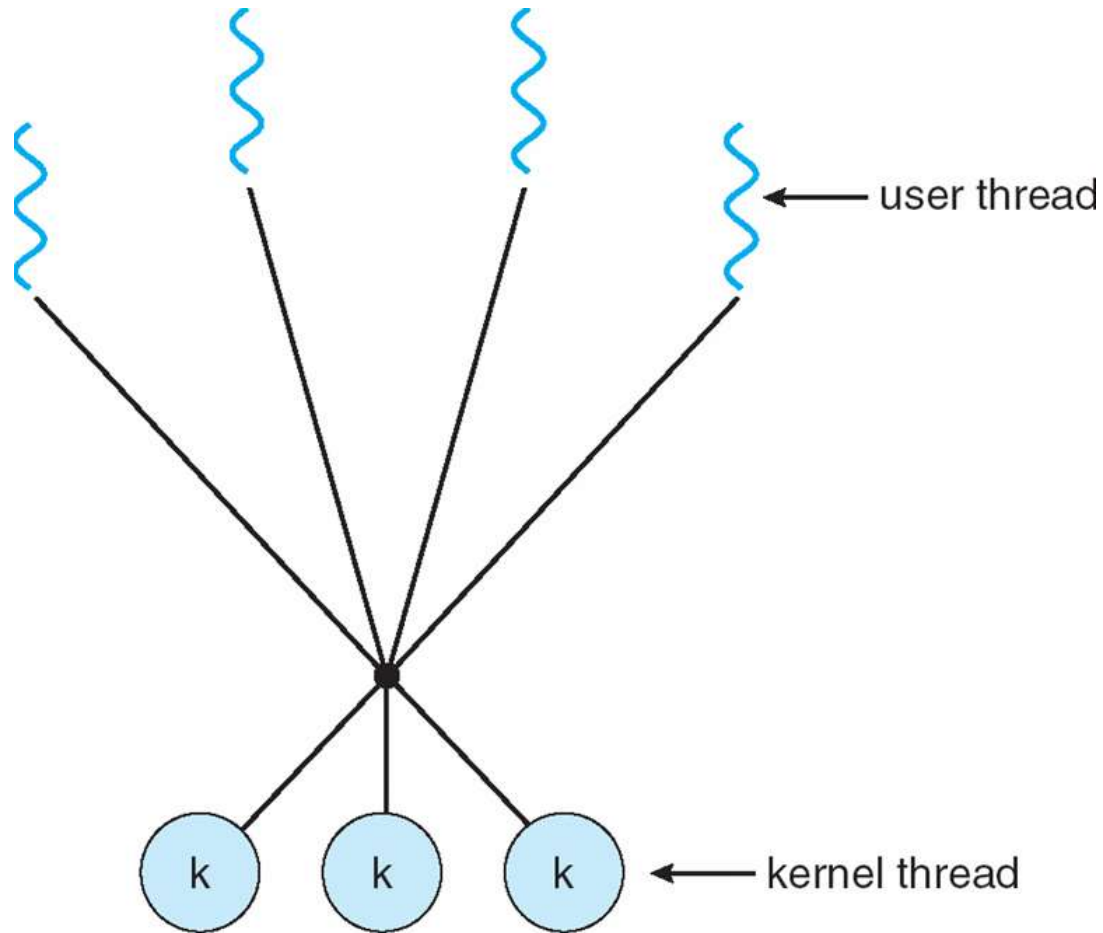
One-to-one Model



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
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber* package

Many-to-Many Model



ULT Vs KLT

ULT	KLT
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
User level threads are designed as dependent threads.	Kernel level threads are designed as independent threads.
Example : Java thread, POSIX threads.	Example : Window Solaris.

Relationship Between Thread and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Example- thread in C using pthread

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Header file for sleep(). man 3 sleep for details.
#include <pthread.h>

// A normal C function that is executed as a thread when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Printing GeeksQuiz from Thread \n");
    return NULL;
}

int main()
{
    pthread_t thread_id;
    printf("Before Thread\n");
    pthread_create(&thread_id, NULL, myThreadFun, NULL);
    pthread_join(thread_id, NULL);
    printf("After Thread\n");
    exit(0);
}
```

```
abcd@ubuntu:~/$ gcc multithread.c -lpthread
abcd@ubuntu:~/$ ./a.out
Before Thread Printing
GeeksQuiz from Thread
After Thread
abcd@ubuntu:~/$
```