

Threads

Dr. Tarunpreet Bhatia
Assistant Professor
CSED, TU

Disclaimer

THIS IS NOT A COPYRIGHT MATERIAL

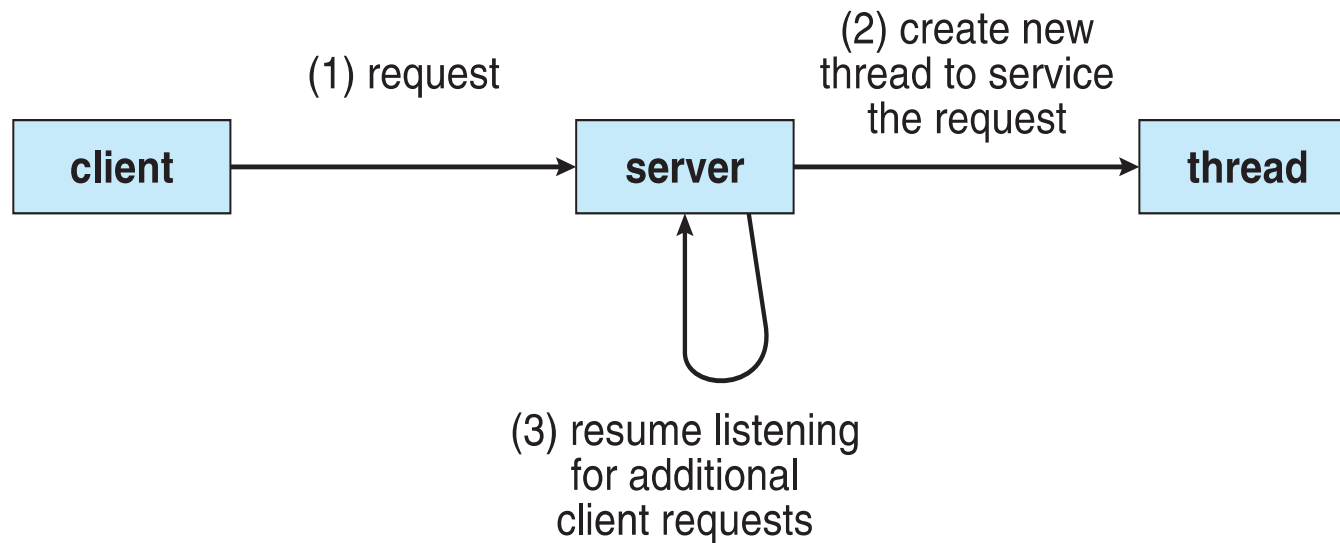
Content has been taken mainly from the following books:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings

Thread

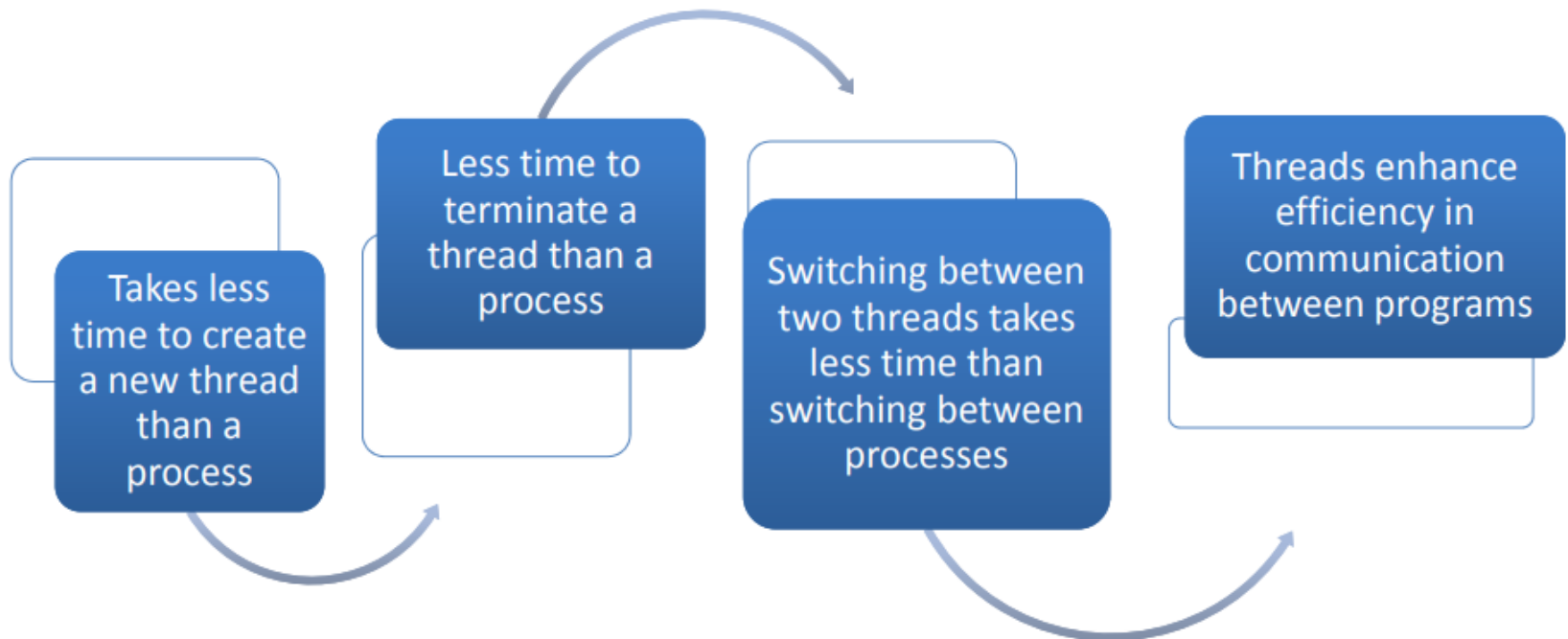
- Application - Set of Multiple Processes
- Process – An Executable File (.exe)
- Each **PROCESS** has its own Address Space, CPU Quota, Access to Hardware Resources and Kernel Resources
- **THREAD** – Function Present within Executable File
- Each **THREAD** has its own PC, Stack , Registers

Multithreaded Server Architecture

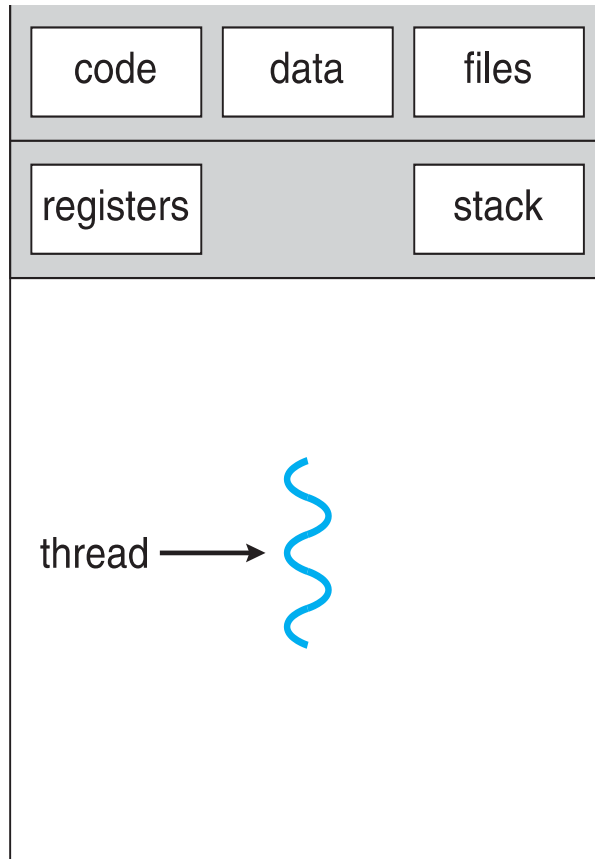


Benefits

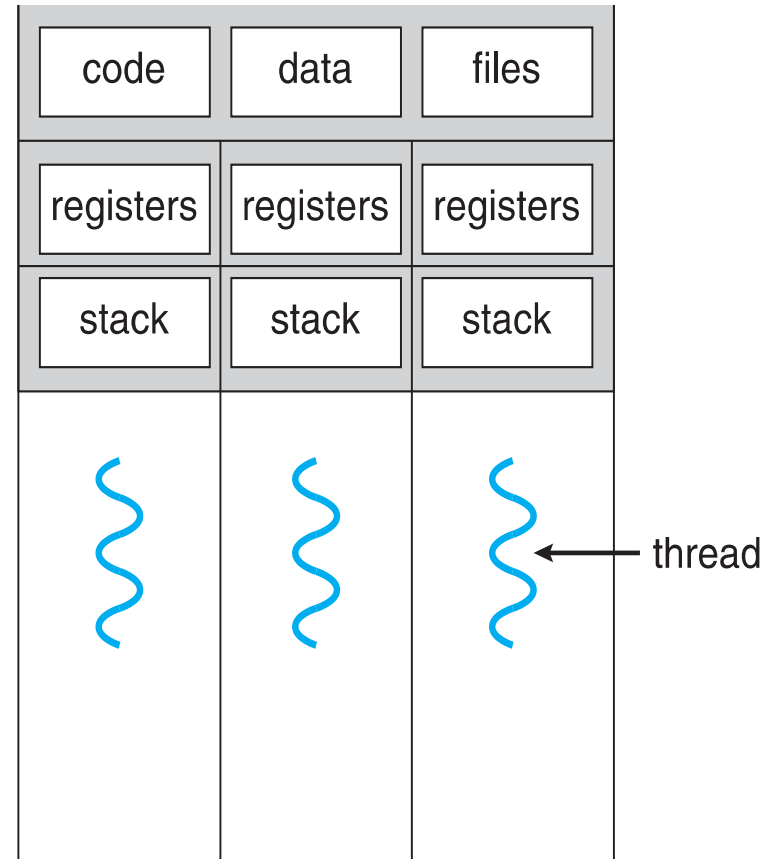
- Responsiveness
- Resource Sharing
- Economy
- Scalability



Single and Multithreaded Processes

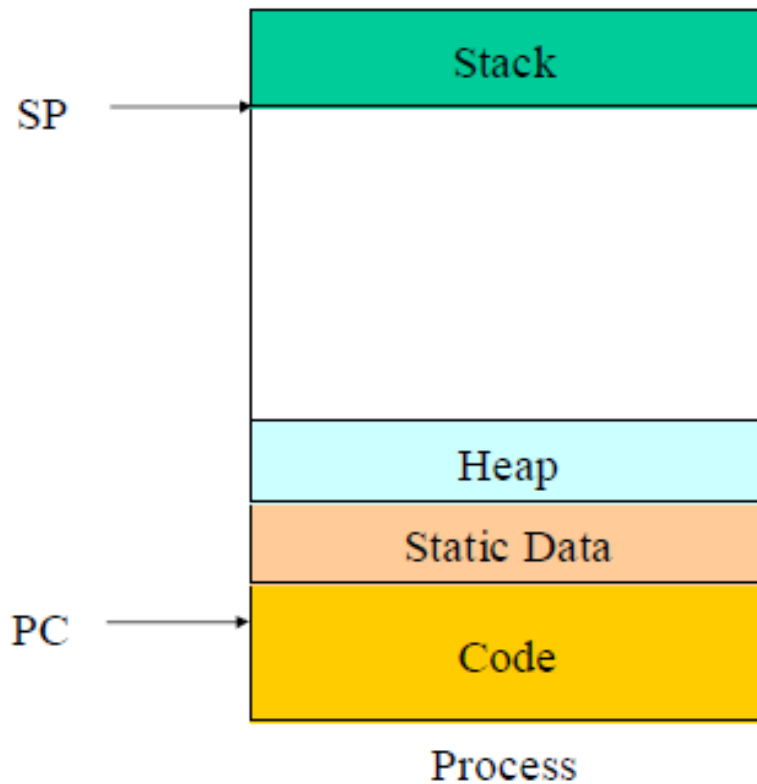


single-threaded process

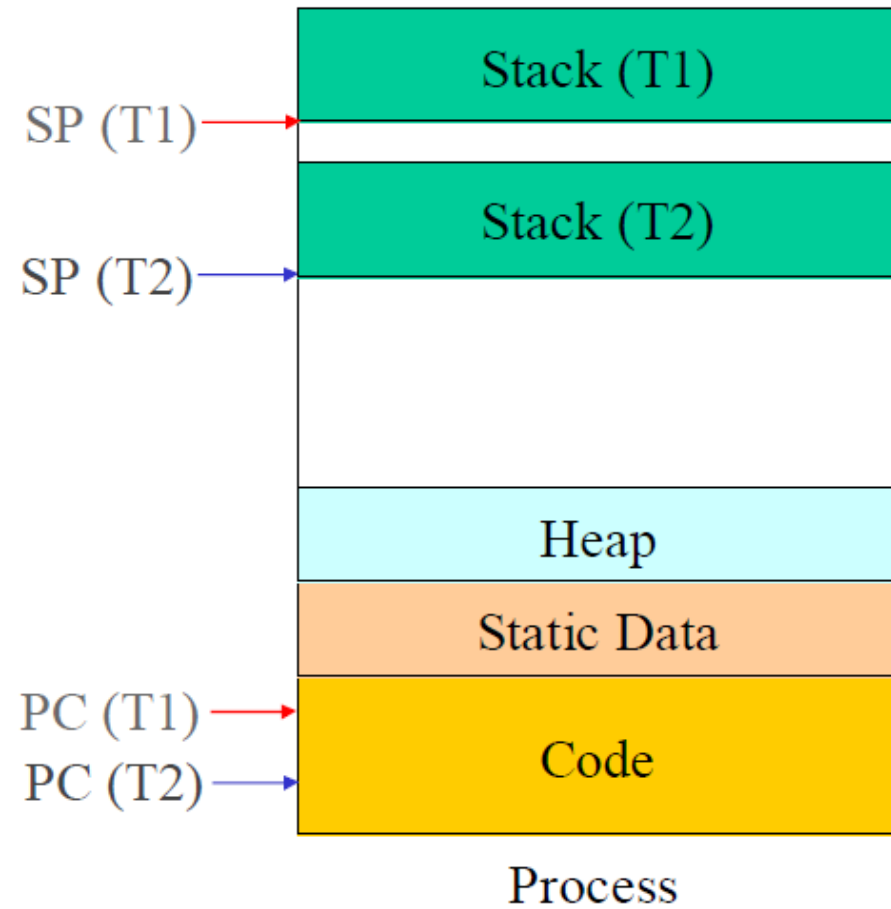


multithreaded process

Process View



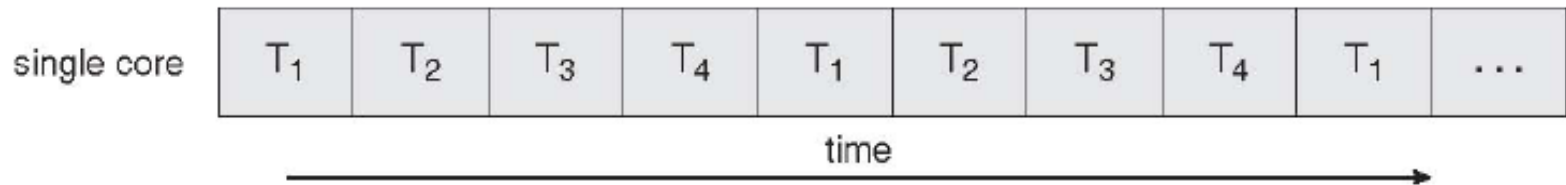
Thread view



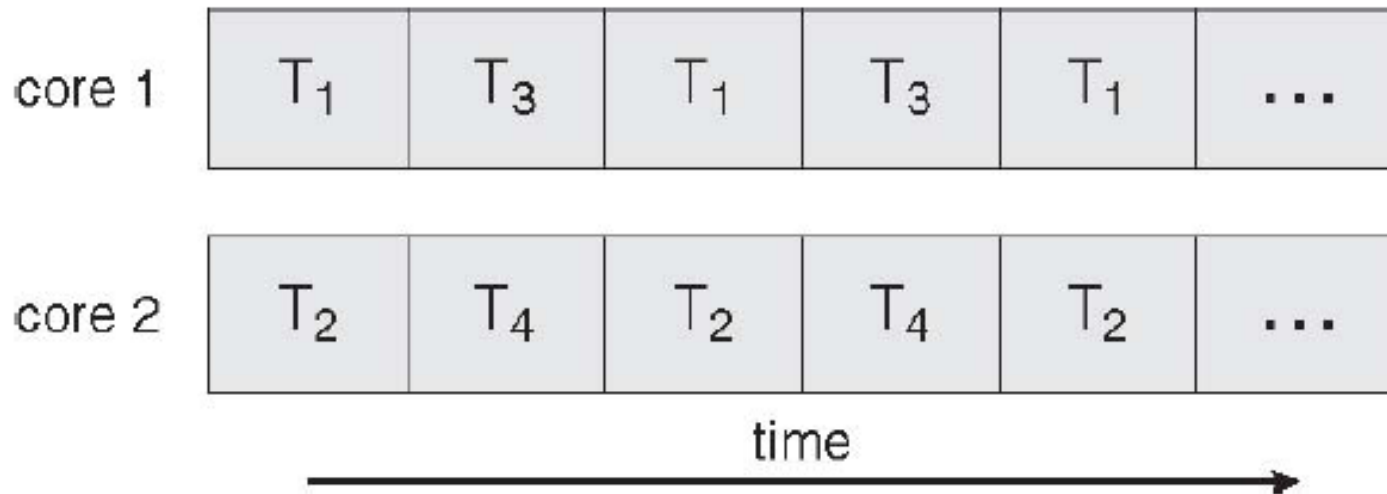
Process Vs Thread

Process	Thread
Process means any program is in execution.	Thread means segment of a process.
Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.
Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Processes require more time for creation and termination.	Threads require less time for creation and termination
Processes are totally independent and don't share memory.	A thread may share some memory with its peer threads.
If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.

Concurrent Execution on a Single-core System



Parallel Execution on a Multi-core System



Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- S is serial portion
- N processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As N approaches infinity, speedup approaches $1 / S$
Serial portion of an application has disproportionate effect on performance gained by adding additional cores

Multicore Programming

- Multi-Core Systems putting pressure on programmers, challenges include
 - Dividing activities
 - Balance
 - Data splitting
 - Data dependency
 - Testing and debugging

Types of Threads

User-level threads

All **code** and **data structures** for the library exist in user space.

Invoking a function in the API results in a **local function call** in user **space** and not a system call.

user
mode

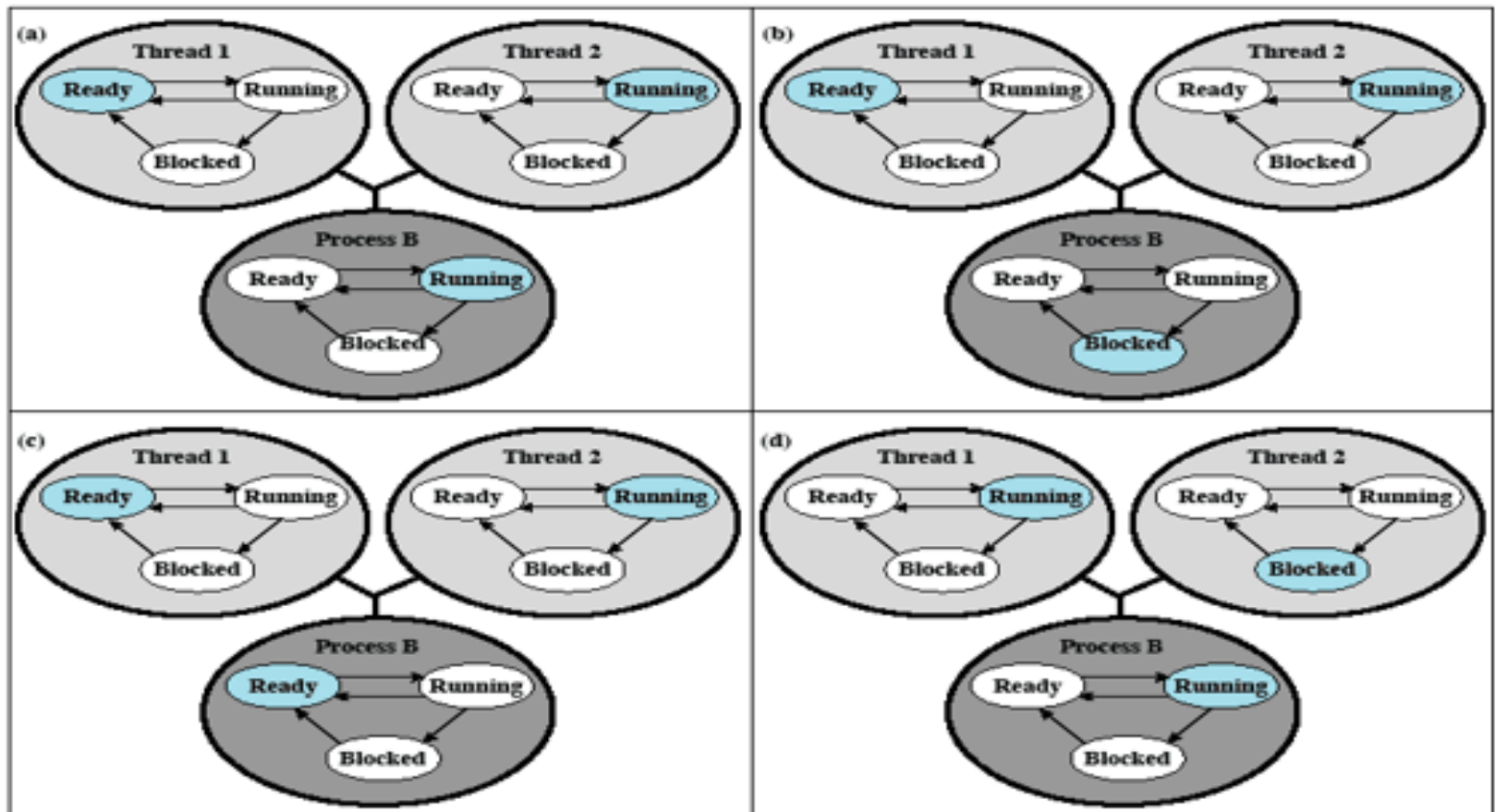
Kernel-level threads

All **code** and **data structures** for the library exists in **kernel space**.

Invoking a function in the API typically results in a **system call** to the kernel.

kernel
mode

Relationships between User Level Thread and Process States



Colored state
is current state

User-level threads

Advantages

- Thread switch does not require kernel-mode.
- Scheduling (of threads) can be application specific.
- Can run on any OS.

Disadvantages

- A system-call by one thread can block all threads of that process.
- In pure ULT, multithreading cannot take advantage of multiprocessing

Kernel-level threads

Advantages

- 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 can be multithreaded

Disadvantages

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

Thread Library

- Three Primary Thread Libraries:
 - Java Threads
 - Pthreads
 - Win32 Threads

Multi Threading Models

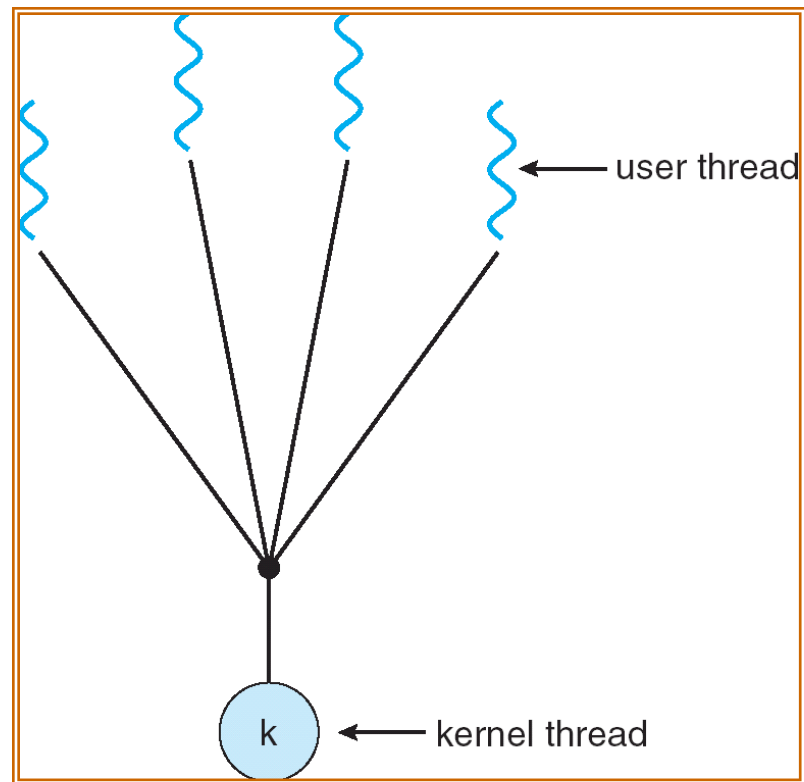
- Many-to-One
- One-to-One
- Many-to-Many
- Two-level

Many-to-One Model

- Many User-Level Threads mapped to Single Kernel Thread

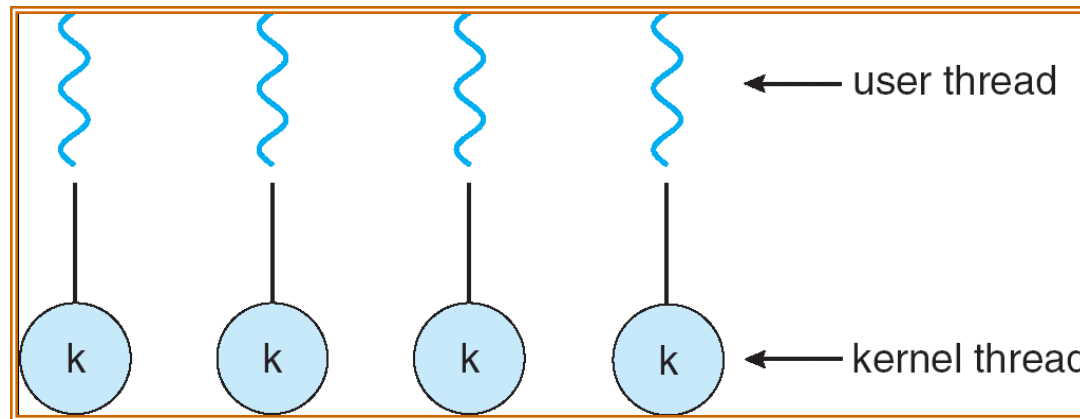
- Examples:

- Solaris Green Threads



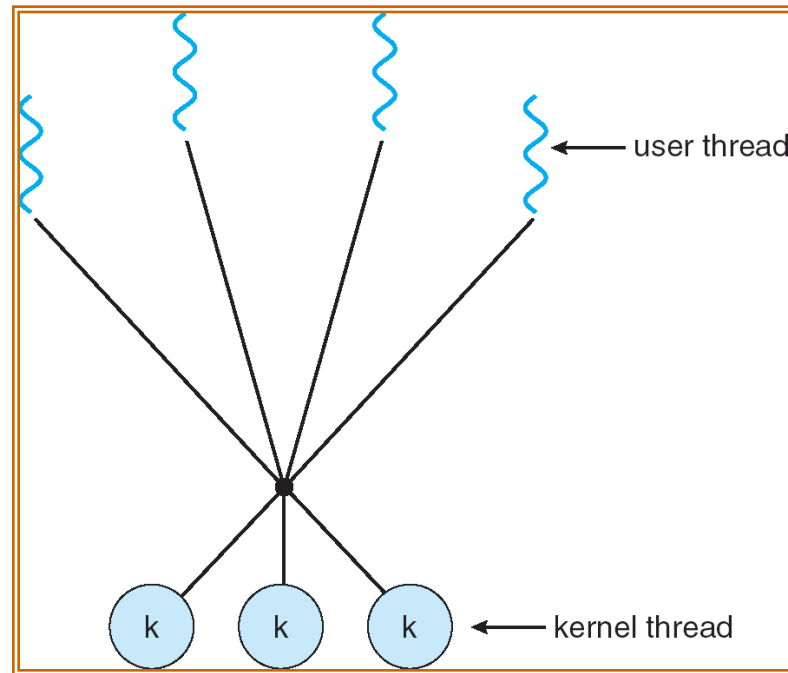
One-to-One Model

- Each User-Level Thread maps to KERNEL THREAD
- Examples
 - Linux
 - Solaris 9 and later
 - Windows NT/XP/2000



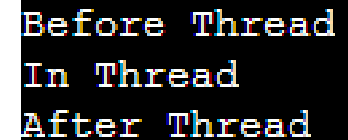
Many-to-Many Model

- Allows many User Level Threads to be mapped to many Kernel Threads
- Allows the Operating System to create a sufficient number of Kernel Threads
- Windows NT/2000



Multithreading example 1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Header file for sleep()
#include <pthread.h>
void *myThreadFun(void *param){
    sleep(1);
    printf("In Thread \n");
}
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");
}
```



Before Thread
In Thread
After Thread

Multithreading example 2

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int g = 0;
void * PrintHello(void * data){
    g++;
    int l = 0;
    l++;
    printf("Hello from thread %u - I was created in iteration %d, g
= %d and l = %d!\n", (int)pthread_self(), (int)data,g,l);
    pthread_exit(NULL);
}
```

```
int main(){
    int rc, i;
    pthread_t thread_id[5];
    for(i = 0; i < 4; i++)    {
        rc = pthread_create(&thread_id[i], NULL, PrintHello, (void*)i);
        if(rc) {
            printf("\n ERROR: return code from pthread_create is %d \n", rc);
            exit(1);    }
        printf("\n I am thread %u. Created new thread (%u) in iteration %d
        ...\n", (int)pthread_self(), (int)thread_id[i], i);
    }
    pthread_exit(NULL);
}
```


Output

```
I am thread 971499328. Created new thread (971495168) in iteration 0 ...  
I am thread 971499328. Created new thread (963102464) in iteration 1 ...  
I am thread 971499328. Created new thread (954709760) in iteration 2 ...  
Hello from thread 971495168 - I was created in iteration 0, g = 1 and l = 1!  
  
I am thread 971499328. Created new thread (946317056) in iteration 3 ...  
Hello from thread 946317056 - I was created in iteration 3, g = 2 and l = 1!  
Hello from thread 954709760 - I was created in iteration 2, g = 3 and l = 1!  
Hello from thread 963102464 - I was created in iteration 1, g = 4 and l = 1!
```