OPERATING SYSTEMS

Subject code : CSC 404

Subject In-charge

Nidhi Gaur
Assistant Professor
email: nidhigaur@sfit.ac.in

THREADS

Threads

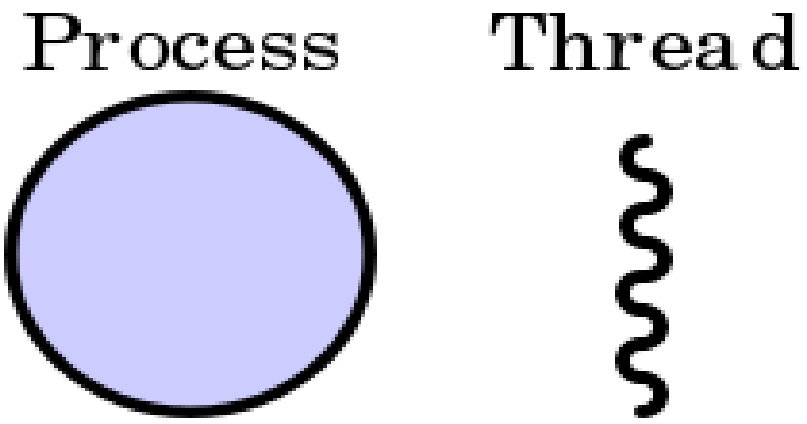
- Process and thread
- Concept of Multithreading
- Types of threads
- Multithreading models

Process

The process can be referred to as a program in execution.

Processes require more time for context switching as they are heavier.

Processes possess their individual program counter (PC), stack space and register set.



Process

- Individual processes are independent of each other.
- In case any process gets blocked, the remaining processes carry on with their work.
- Processes require more resources than threads
- Processes require more time for termination.

Process

Context switch time is proportional to the frequency of interrupted process, resulting in high overhead.

The state saved in PCB is related to resource ownership increases the size of PCB, hence increasing the overhead.

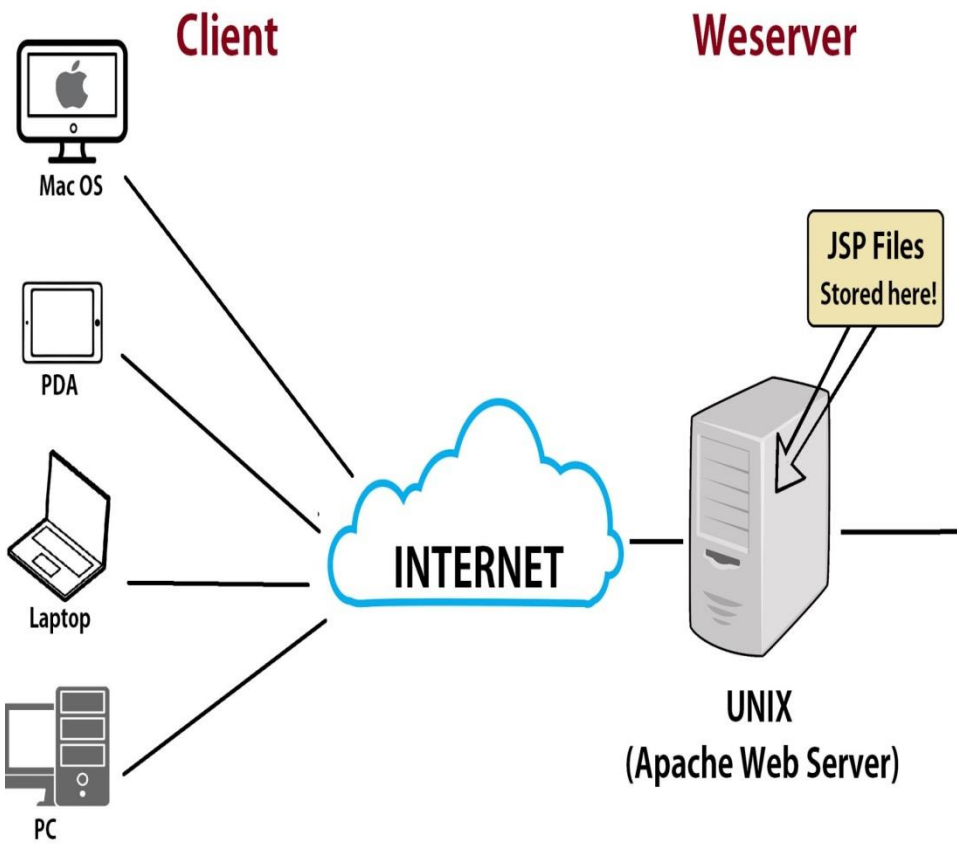
More the information in PCB, more time in saving and restoring.

Hence more context switch time.

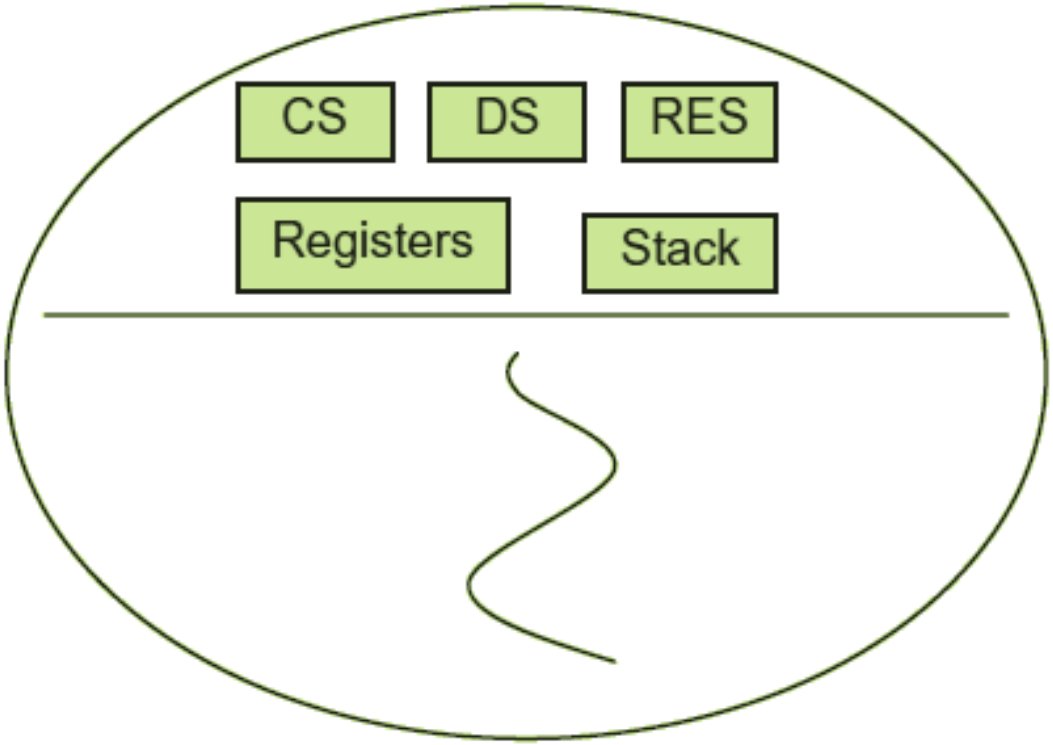
Process and thread

Example of web server

- Web server runs a single process and then creates multiple processes that accepts client's request.
- Whenever a request is received by server, it creates a new process.
- Increasing context switch overhead.



Single threaded process



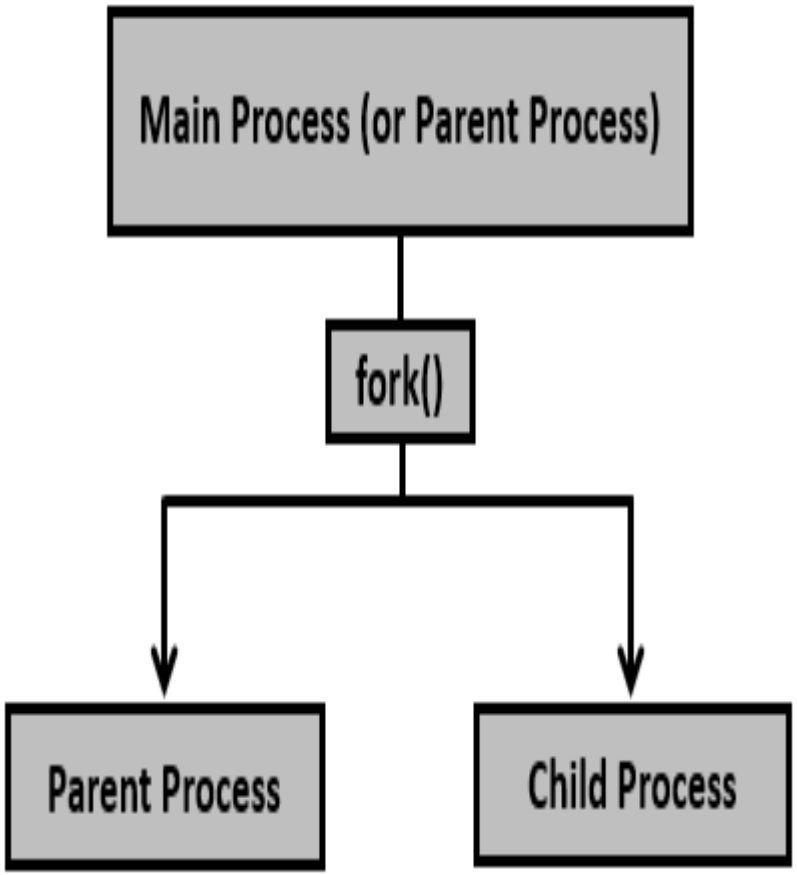
CS: Code section
DS: Data section
RES: Resources

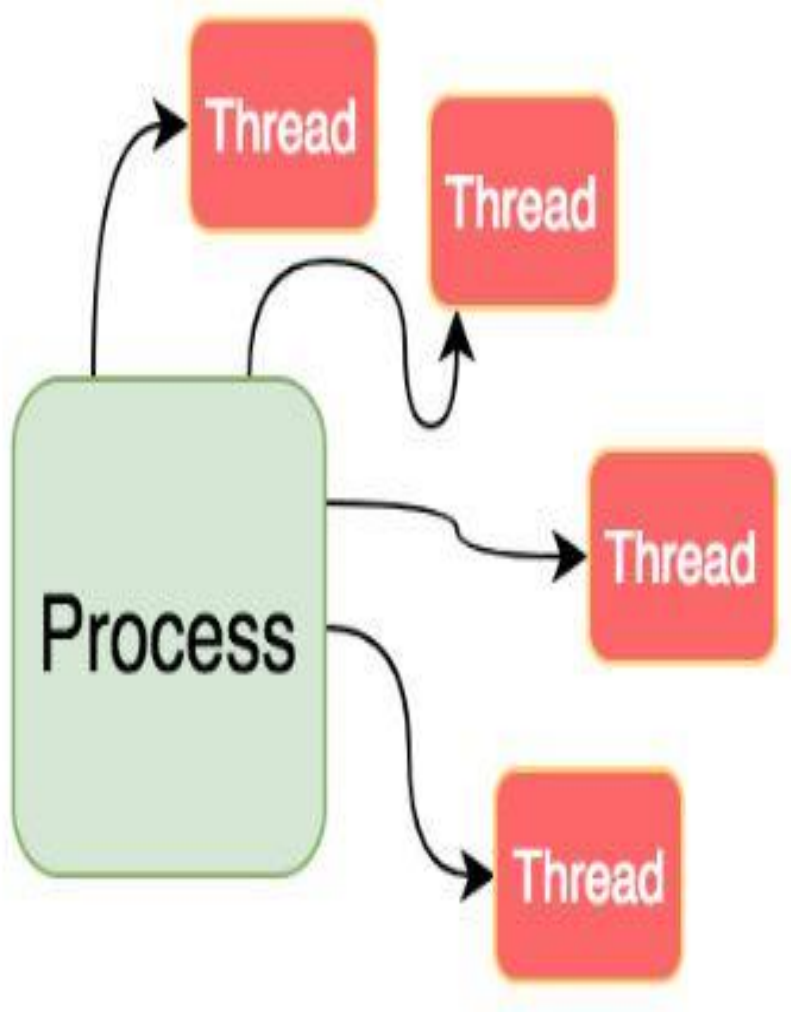
Process as a single thread of execution

Process and thread

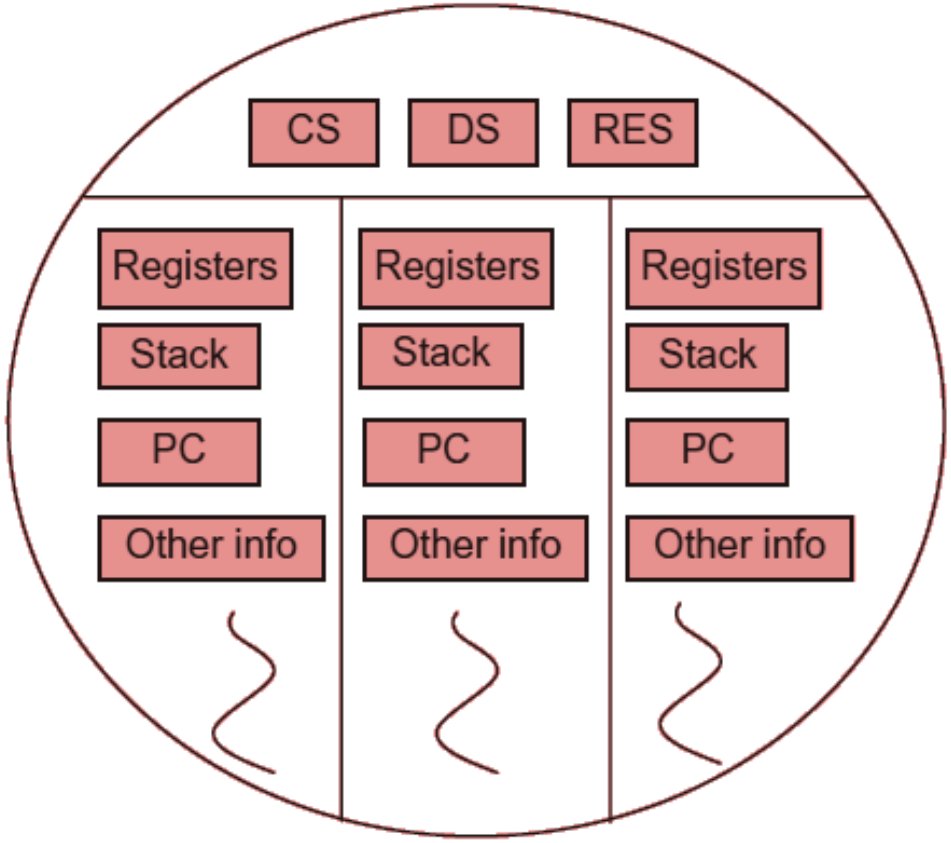
- The current process and next scheduled process have very less difference
- Only PC value, CPU registers and stack differ.
- All processes have been created by server process and share code, data and other resources.
- If this redundancy can be eliminated, context switch time reduces.

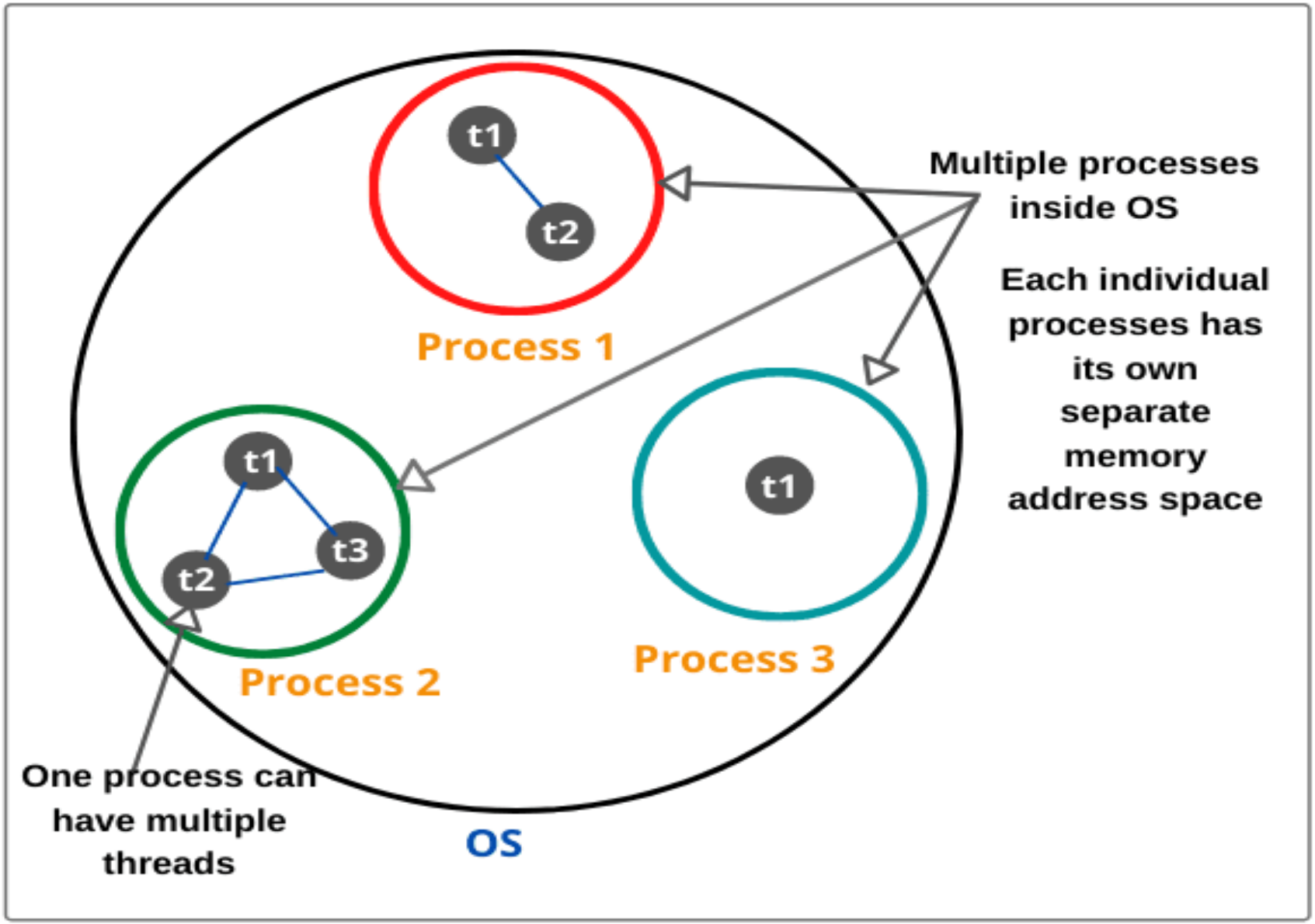
Process divided into child processes or threads





A process divided into multiple threads of execution



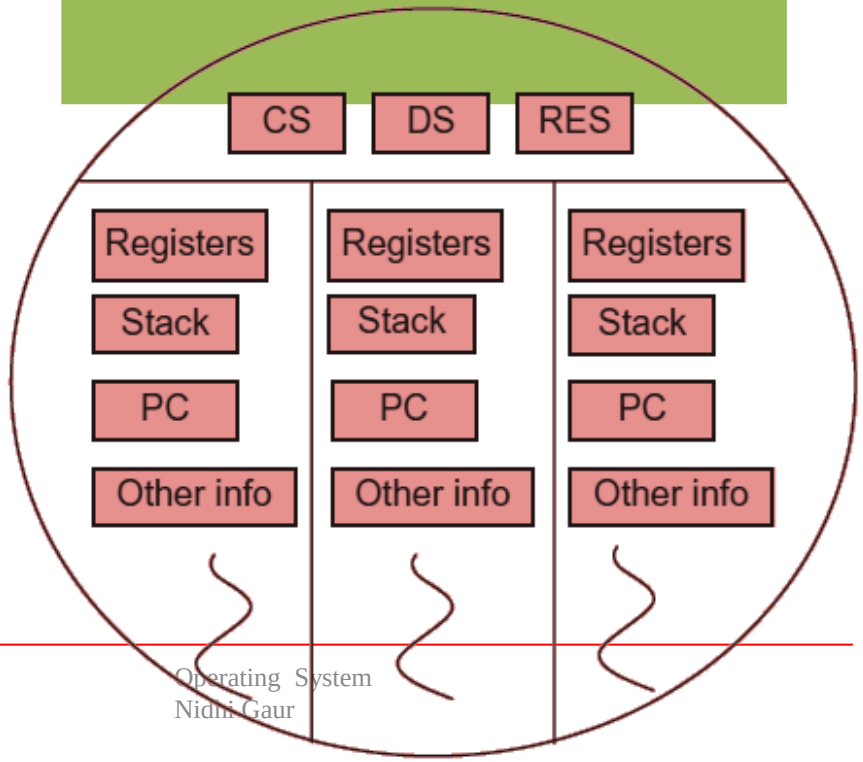

t1
t2
Process 1
Multiple processes inside OS
Each individual processes has its own separate memory address space
t1
t2
t3
Process 2
t1
Process 3
One process can have multiple threads
OS

Thread

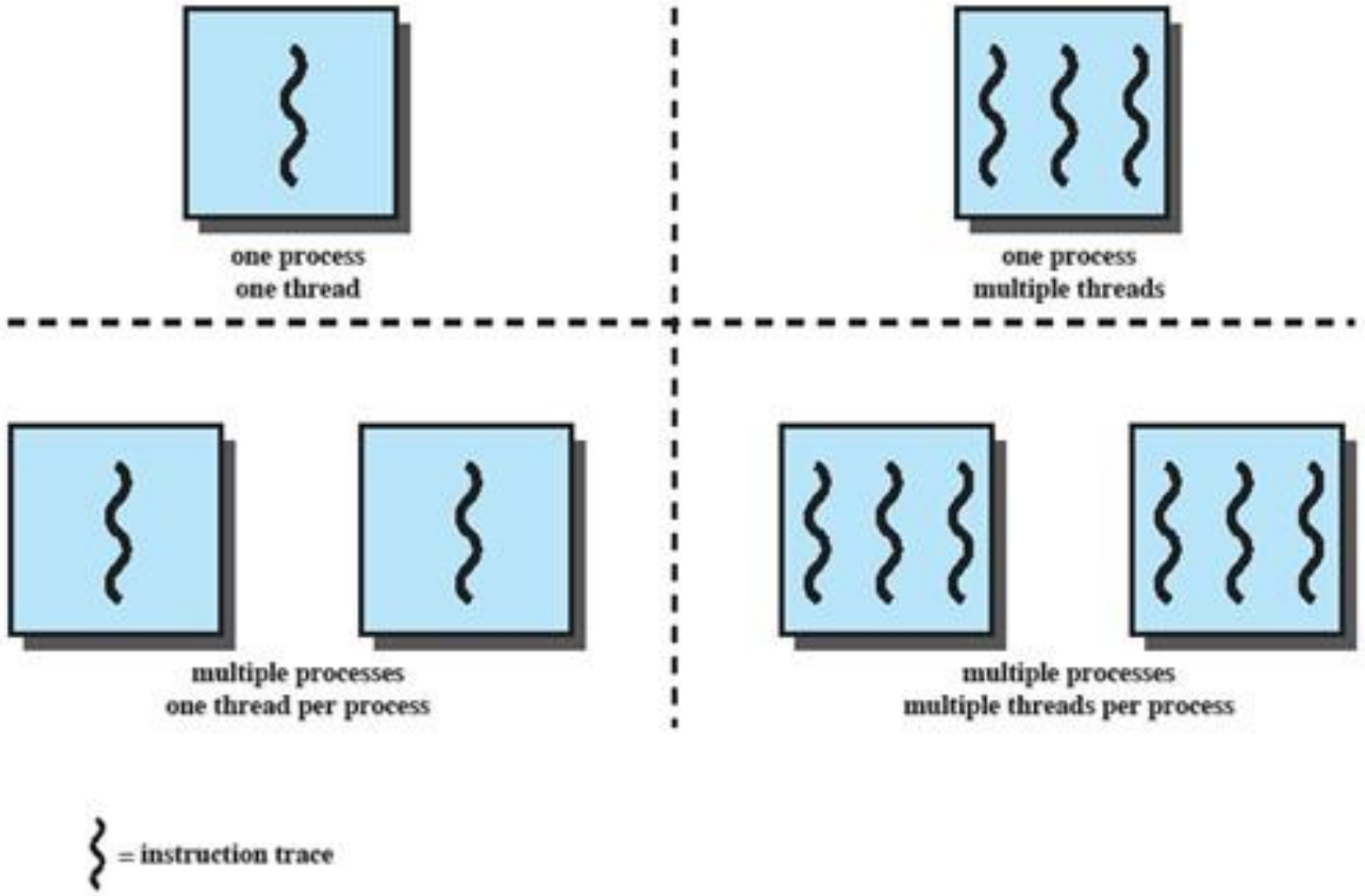
A thread is a basic unit of CPU utilization

It comprises of thread ID, a register set, a program counter and a stack.

It shares with other threads belonging to the same process its code section, data section and other OS resources such as opened files and signals.



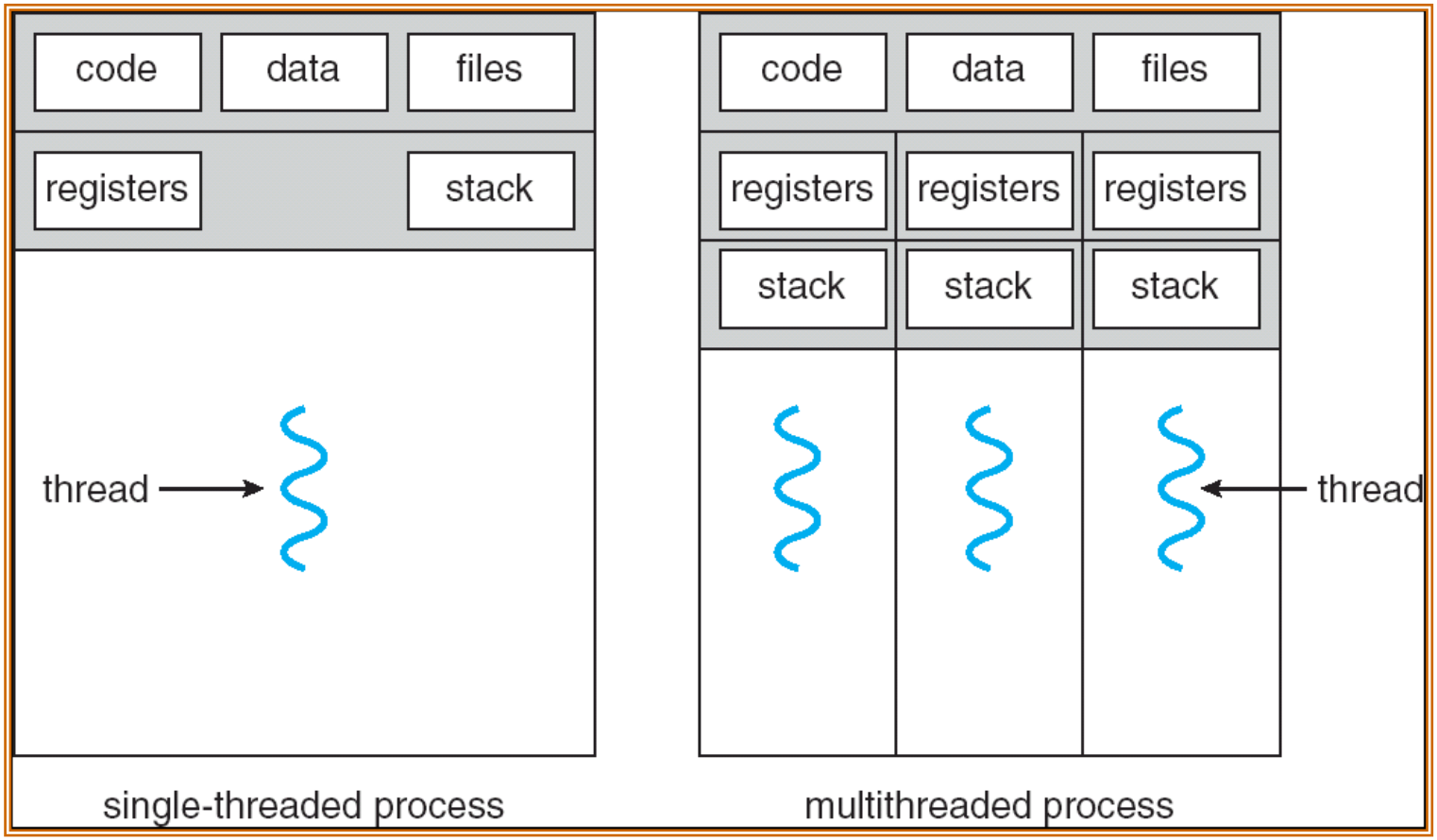
Threads



Thread

- A traditional process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.
- For example a word processor may have different threads for displaying graphics, responding to keystrokes and a thread for performing spelling check.

Single and Multithreaded Processes



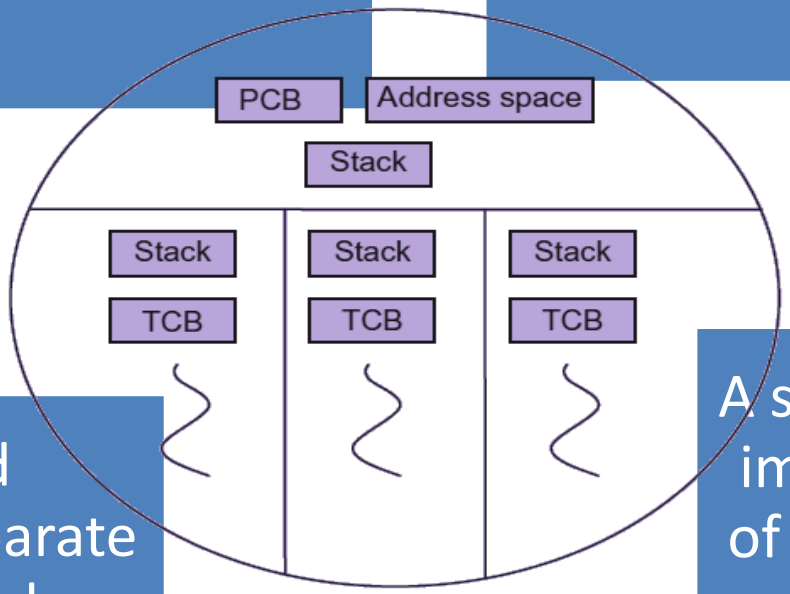
Thread Control Block

- Thread ID (TID)
- PC
- Registers
- State
- Priority
- Event information
- Scheduling related information
- Pointer to owner process
- TCB Pointer

Concept of multithreading

Threads provide a way to improve application performance through parallelism.

Each thread belongs to exactly one process and no thread can exist outside a process.

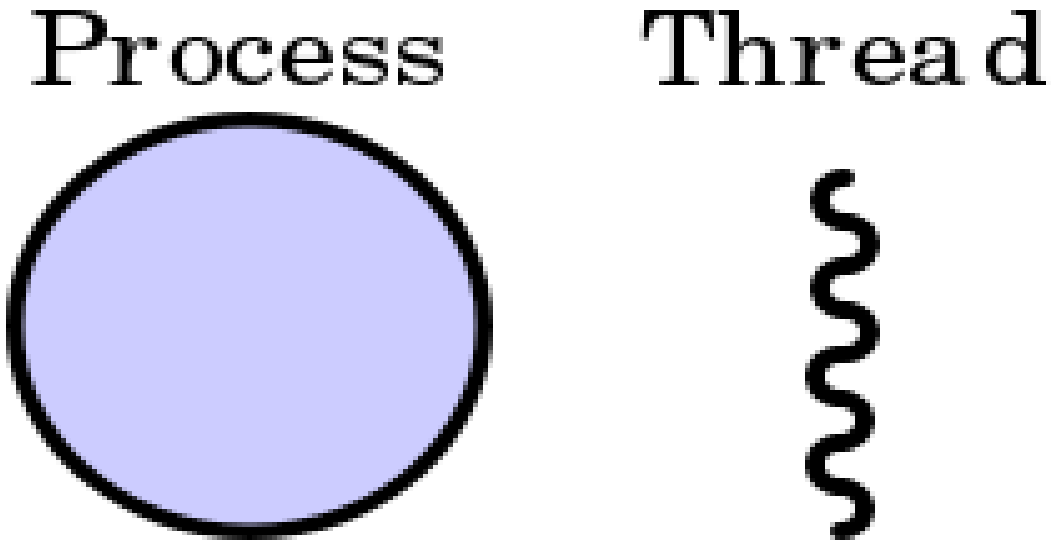


Each thread represents a separate flow of control.

A software approach to improve performance of operating system by reducing the overhead.

Process vs. thread

- **Thread:** The unit of dispatching is usually referred to as a thread or lightweight process.
- **Process:** The unit of resource ownership is usually referred to as a process.



Benefits

Responsiveness

an interactive application may allow program to run even if a part of it is blocked in lengthy operation.

Resource Sharing

threads share memory and resources of process to which they belong by default. It allows application to perform different activities within the same address space.

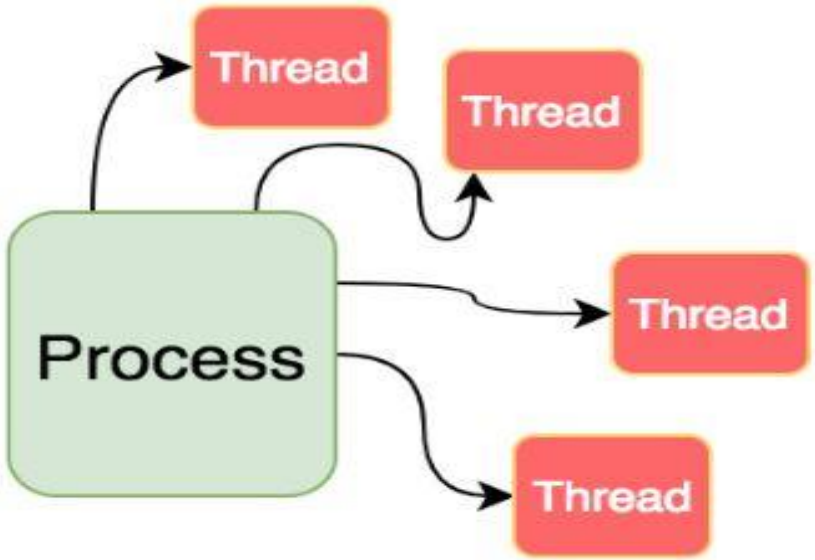
Benefits

Economy:

allocating memory and resources for process creation is costly. It is more economical to create and context switch threads. It is much more time consuming to create and manage processes than threads.

Scalability:

benefit of multithreading increases in multiprocessor architecture, threads may run in parallel on different processor.



Process vs. thread

BASIS OF COMPARISON	PROCESS	THREAD
Description	The process can be referred to as program in execution.	Thread is unit of execution within a process.
Context Switching Time	Require more time for context switching as they are heavier.	Require less time for context switching as they are lighter than processes.
Sharing Of Components	Processes possess their individual program counter (PC), stack space and register set.	A thread shares the code section, address space, data section with the other threads.
Independence	Processes are wholly independent and run in a different memory space than the thread.	A thread runs in the same memory space as the process it belongs to.
Dependence	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.

Types of threads

- User level threads (ULT)
- Kernel level threads (KLT)

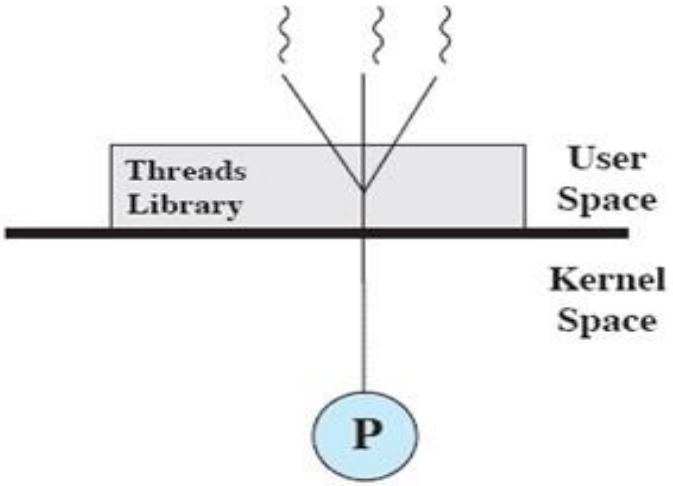
User threads

- The user threads are those threads which are managed by the application in user space only.
- User threads are created and further managed through a thread library provided by the OS.
- A thread table is maintained for user threads. The thread table has the entries to point the TCB of each thread.

User Threads

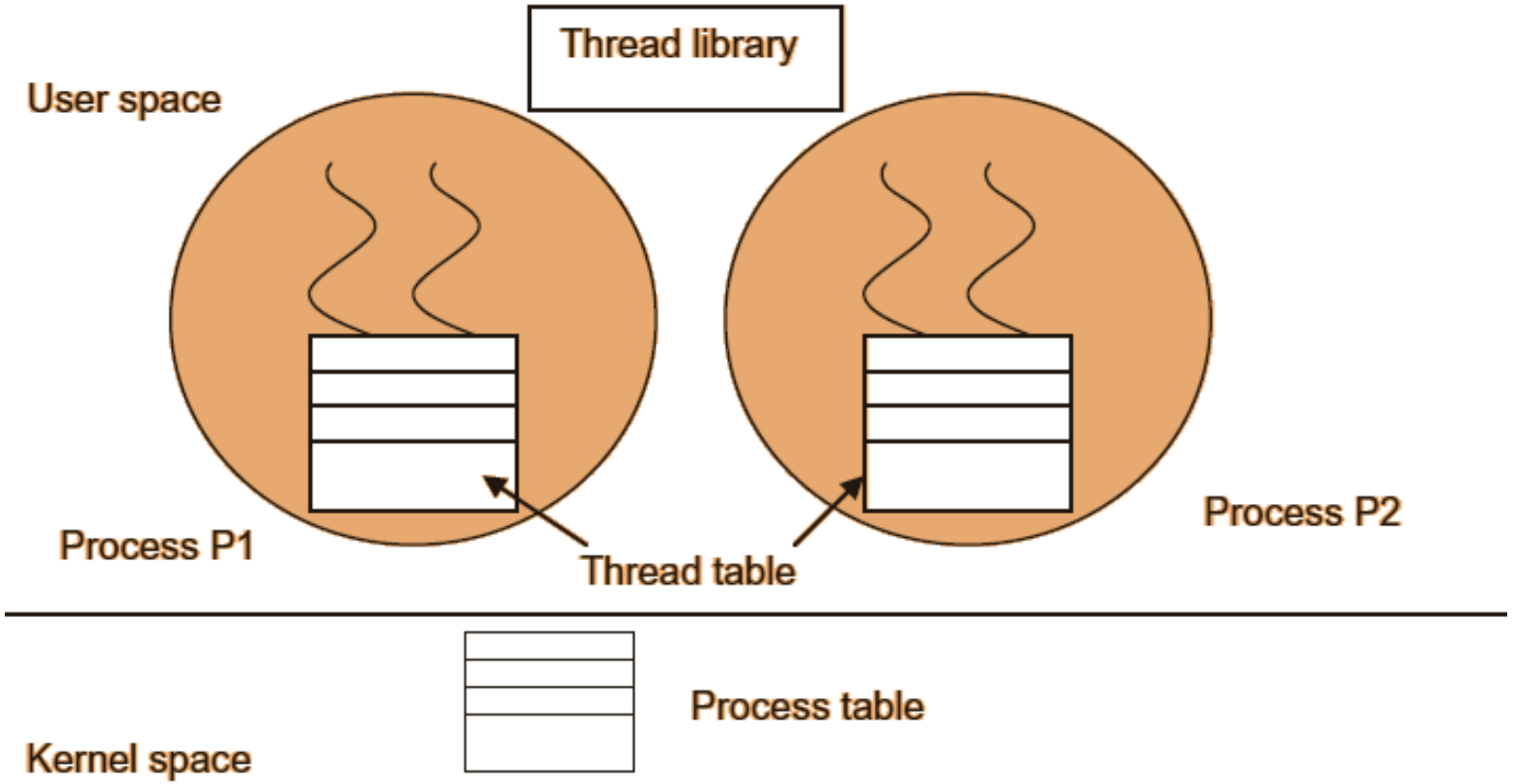
User level threads (ULT)

- All the work of thread management is done by the application and the kernel is not aware of the existence of threads.
- Thread library executed in user mode
- Kernel is not involved.
- Threads library contains code for creating and destroying threads, passing messages and data between threads, scheduling execution, restoring context.



(a) Pure user-level

User threads



User Threads

- By default an application begins with a single thread.
- At any time, the application may spawn a new thread to run within the same process.
- Spawning is done by invoking the spawn utility in the threads library.
- The thread library creates data structure for new thread and then passes control to one of the threads in process that is in ready state using some scheduling algorithm.
- The kernel is unaware of this activity.

User Threads

- Creating and managing the user threads is easy and much faster as compare to a process and a kernel thread.
- User threads are more portable also due to their nature as they do not require the support from kernel making them independent of a particular operating system.
- User level thread implementations are known as many-to-one mappings as the operating system maps all threads in a multithreaded process to a single execution context.

User Threads

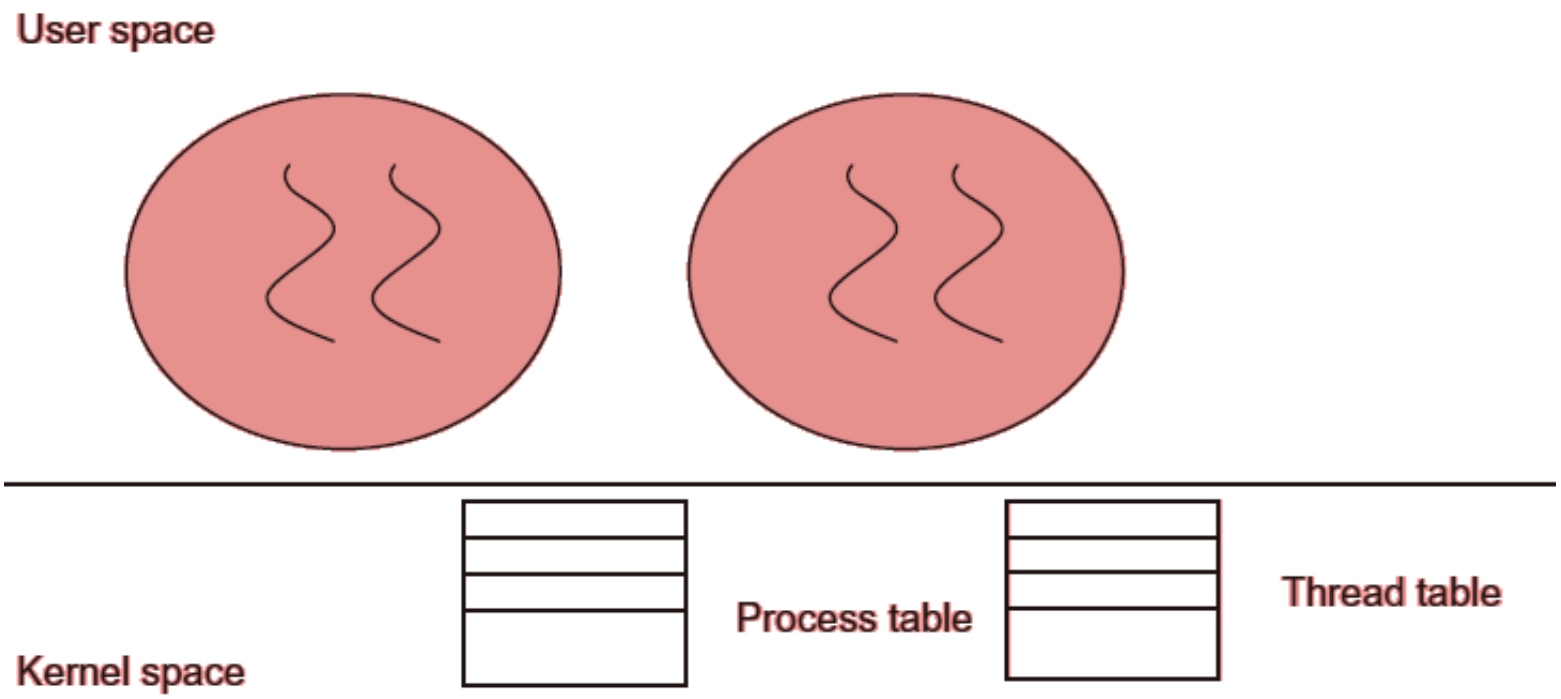
- Thread management done by user-level threads library

- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

Kernel Threads

- Kernel threads implemented in the kernel space are managed through system calls.
- Kernel threads are managed through a single thread table maintained in kernel space.
- There is one-to-one mapping in kernel threads that provides each user thread with a kernel thread that it can dispatch.

Kernel threads



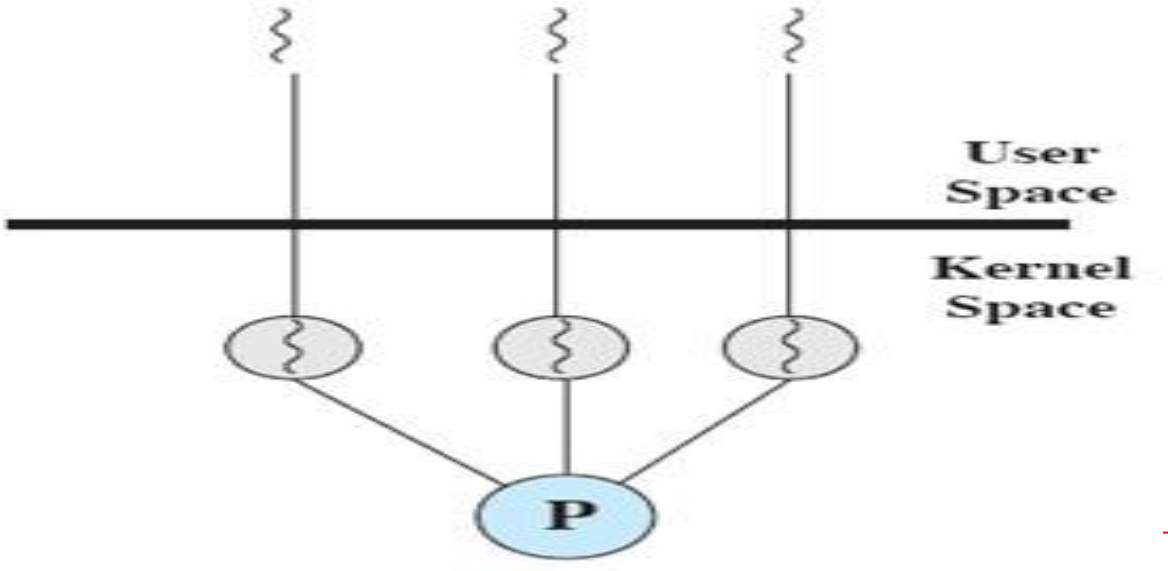
Kernel Threads

Kernel level threads (KLT)

Kernel is aware of and schedules threads

No thread management at application level

A blocking system call will not block all peer threads.



(b) Pure kernel-level

Kernel Threads

- Kernel maintains context information of process.
- Scheduling by kernel is done on thread basis.
- The kernel can schedule multiple threads from one process on multiple processors.
- Also if one thread is blocked, kernel can process another thread from the same process.

Kernel Threads

- The disadvantage of KLT is that the transfer of control from one thread to another within the same process requires a mode switch to kernel.

Kernel Threads

- Supported by the Kernel

- Examples
  - Windows XP/2000
  - Linux
  - Tru64 UNIX
  - Mac OS X

Advantages of ULTs over KLTs

- Thread switching does not require kernel mode privileges. This saves overhead of two mode switches.
- The scheduling algorithm can be tailored to application without disturbing OS scheduler.
- ULTs can run on any OS without changes to underlying kernel.

Disadvantages of ULTs compared to KLTs

- When ULT executes a blocked system call, not only that thread but all the threads within the process are blocked.
- In pure ULT, multithreaded can't take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Only single thread within a process can execute at a time.

Both problems can be overcome by writing an application as multiple processes.

Combined approach

- Some OS provides combined ULT/KLT approach facility.
- Solaris is good example of such OS.
- The current Solaris limits the ULT/KLT relationship to be one to one.

End of Chapter

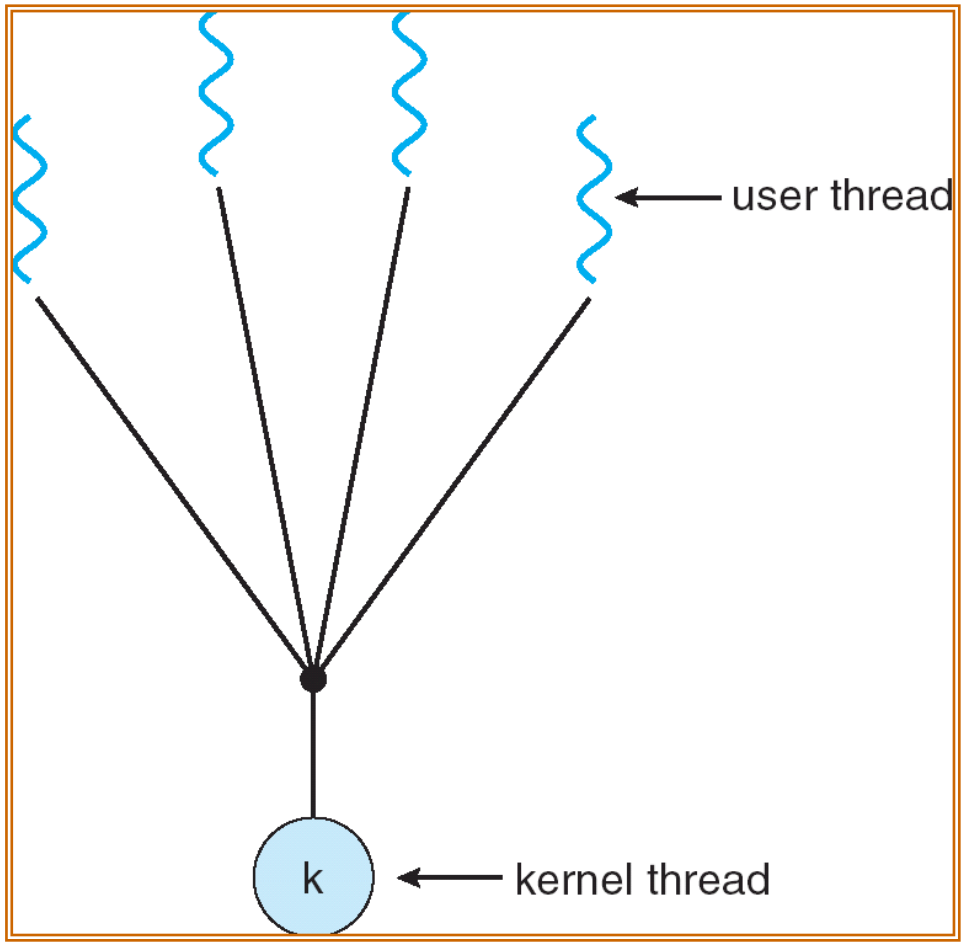
Multithreading Models

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

Many-to-One

- Many user-level threads mapped to single kernel thread.
- Thread management by thread library in user space, so efficient but entire process will block if thread makes blocking system call.
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

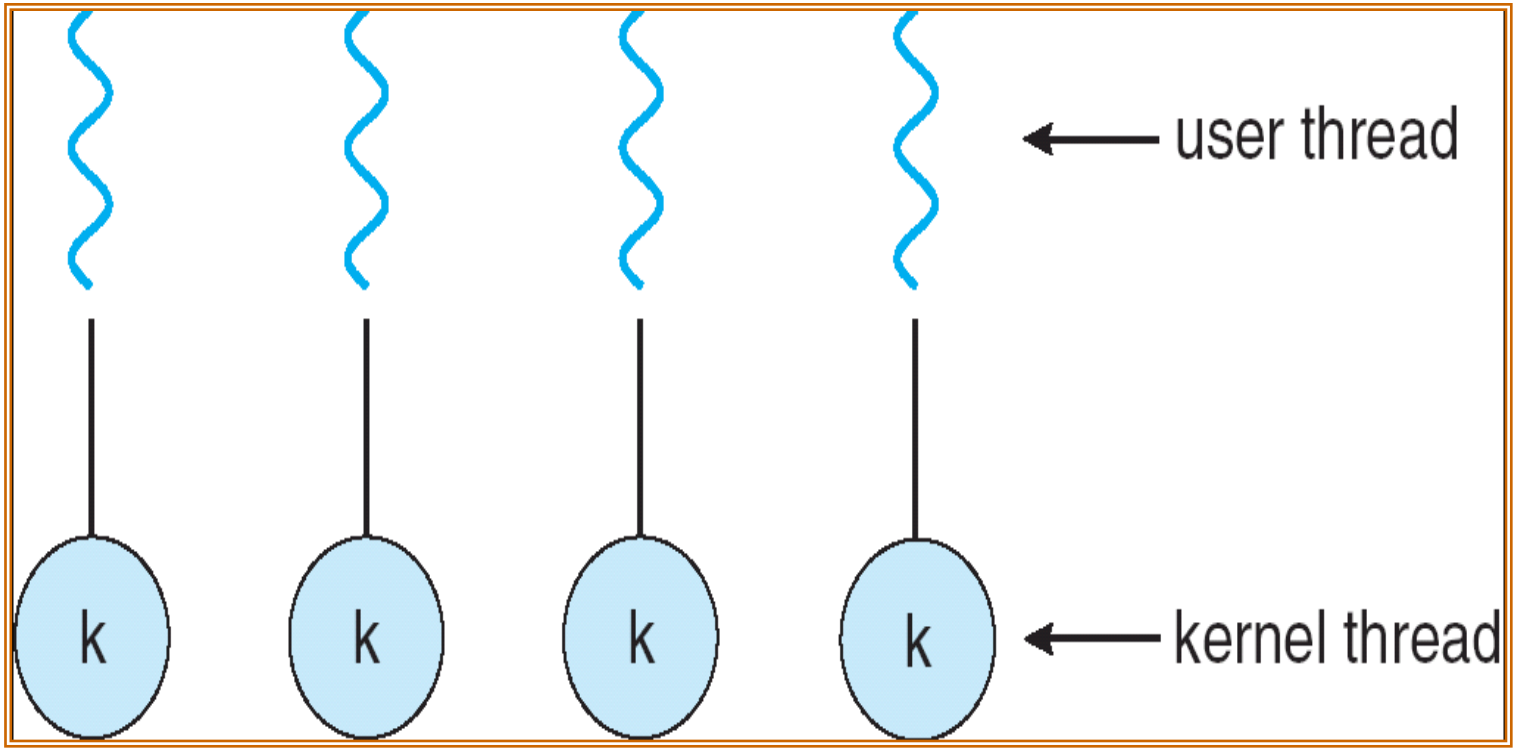
Many-to-One Model



One-to-One

- Each user-level thread maps to kernel thread
- More concurrency than many to one.
- It allows multiple threads to run in parallel on multiprocessors.
- Drawback is creating a user thread requires creating corresponding kernel thread.
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

One-to-one Model

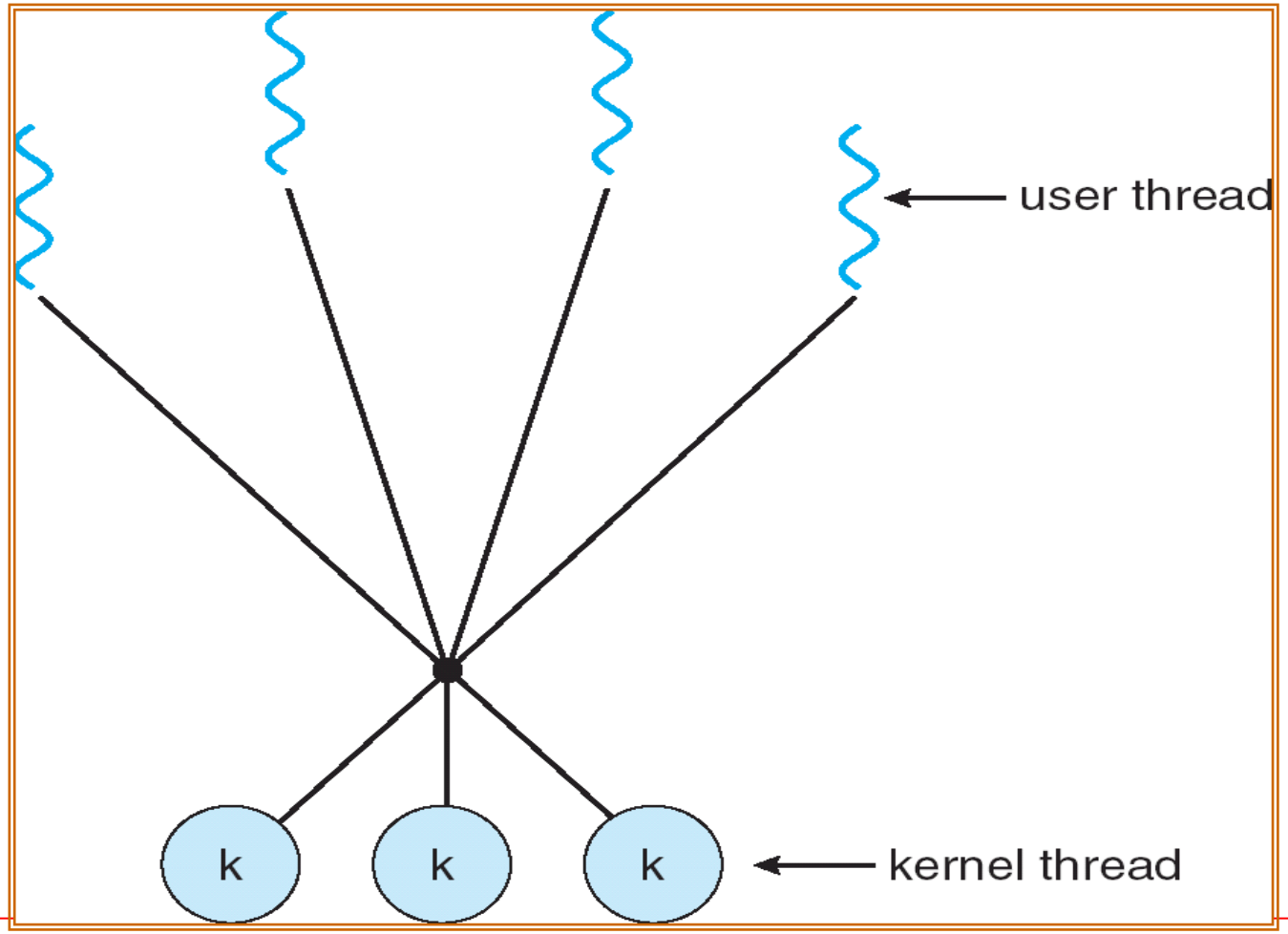


Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Developers can create as many user threads as necessary and corresponding kernel thread can run in parallel on multiprocessor.
- Examples are Solaris prior to version 9

  Windows NT/2000

Many-to-Many Model



Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface