# Threads

# Threads

- Introduction
- Multicore Programming
- Multithreading Models
- Benefits of Threads
- Thread Libraries

## Program and Process Concept

- When a software developer builds a solution, the set of capabilities it provides is usually static and embedded in the form of processed code that is built for the OS. This is typically referred to as the program.
- When the program gets triggered to run, the OS assigns a process ID and other metrics for tracking.
- At the highest level, an executing program is tracked as a process in the OS.
- Note that in the context of different operating systems, jobs and processes may be used interchangeably. However, ***process refer to a program in execution***.

## Context Switching

- The operating system may need to swap the currently executing process with another process to allow other applications to run, it does so with the help of context switching.
- When a process is executing on the CPU, the process context is determined by the program counter (instruction currently run), the processor status, register states, and various other metrics.
- When the OS needs to swap a currently executing process with another process, it must do the following steps:
  1. Pause the currently executing process and save the context.
  2. Switch to the new process.
  3. When starting a new process, the OS must set the context appropriately for that process.
- This ensures that the process executes exactly from where it was swapped.

## Process Control Block (PCB)

| | |
|---|---|
| **Pointer** | **Process state** |
| **Priority** | |
| **Program counter** | |
| **CPU registers** | |
| **Memory management info** | |
| **I/O status information** | |
| **Accounting Information** | |

Ref: https://www.javatpoint.com/os-attributes-of-a-process

## Process Control Block (PCB)

- The **process ID** is a unique identifier for the instance of the process that is to be created or currently running.
- The **process state** determines the current state of the process, described in the preceding section.
- The **pointer** could refer to the hierarchy of processes (e.g., if there was a parent process that triggered this process).
- The **priority** refers to the priority level (e.g., high, medium, low, critical, real time, etc.) that the OS may need to use to determine the scheduling.
- **Affinity and CPU register** details include if there is a need to run a process on a specific core. It may also hold other register and memory details that are needed to execute the process.

## Process Control Block (PCB)

- The **program counter** usually refers to the next instruction that needs to be run.
- The **I/O status information,** like which devices assigned, limits, and so on that is used to monitor each process is also included in the structure.
- The **accounting information** such as paging requirements from memory, timers, how many time unit remaining to finish, … etc

- There could be some modifications to how the PCB looks on different OSs. However, most of the preceding are commonly represented in the PCB.

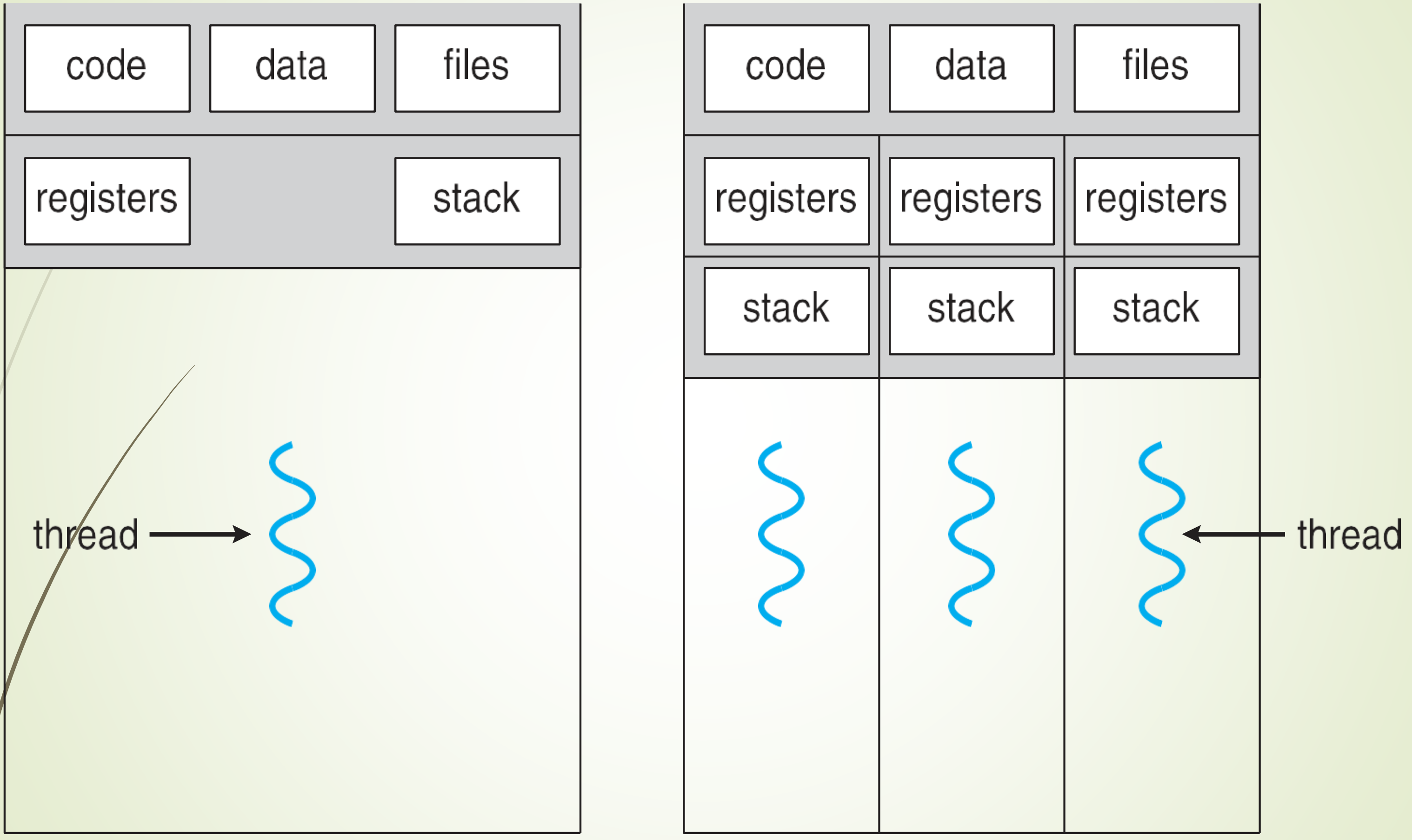
## Introduction

- A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- It shares with other threads belonging to the same process its **code section**, **data section**, and other operating-system **resources**, such as open files and signals.

## Introduction

- Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time.
- The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time. Threads allows us to do this.

## Introduction



single-threaded process

multithreaded process

## Introduction

- Most software applications that run on modern computers are **multithreaded**.
- An application typically is implemented as a separate process with several threads of control.
  - ➢ A web browser might have one thread display images or text while another thread retrieves data from the network, for example.
  - ➢ A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

## Thread Concepts

- Examples:

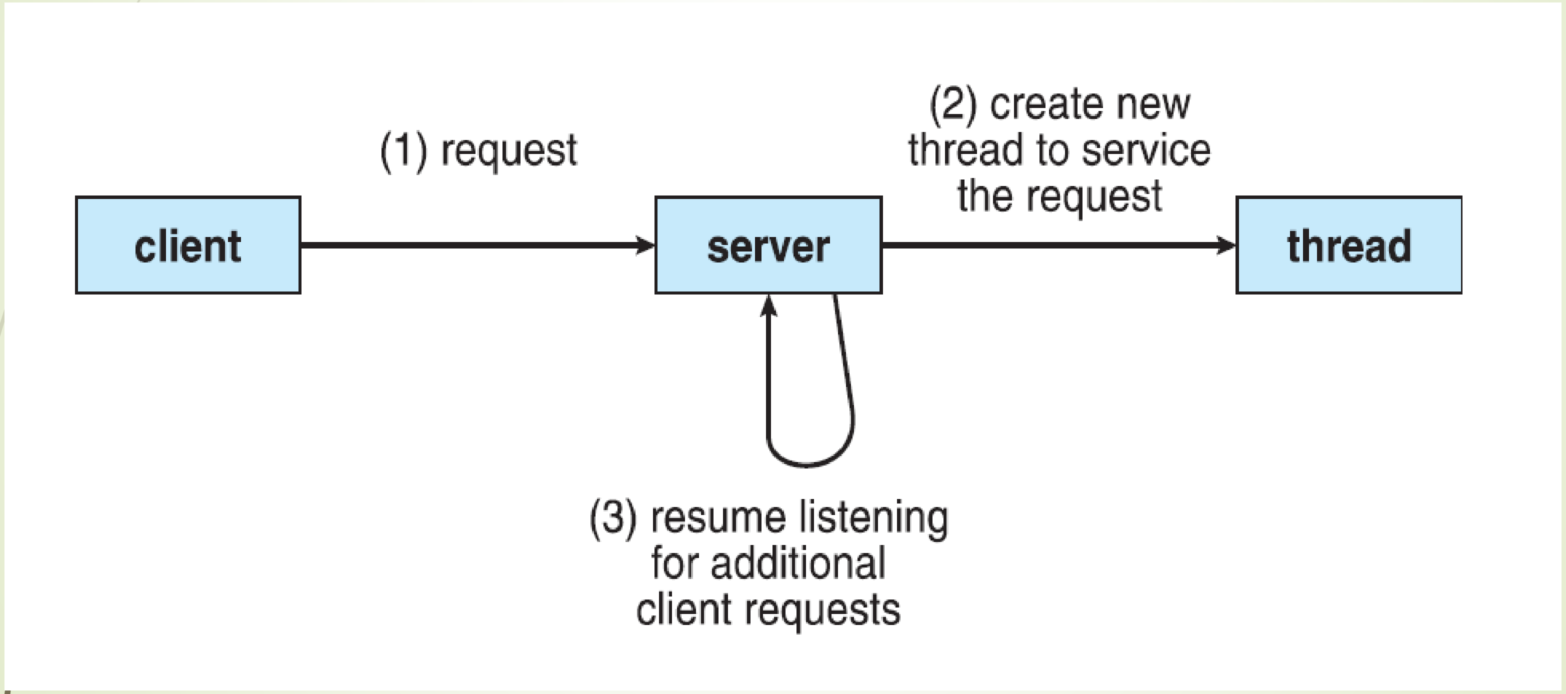
- Chatting program: the sending and receiving operations are independent, using threads

- Strategic games: there are many actions happened at the same time

independently, using threads

## Introduction

- Multithreaded server architecture

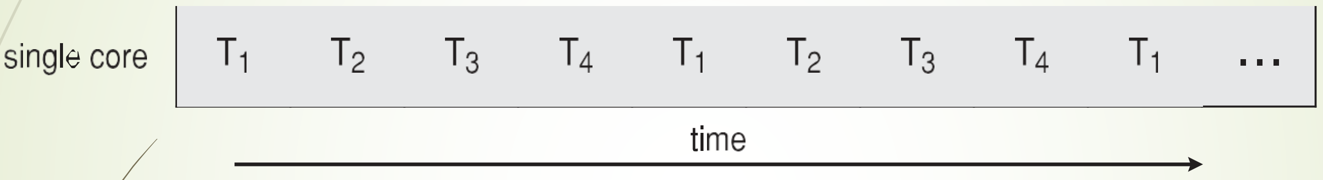

client
(1) request
server
(2) create new thread to service the request
thread
(3) resume listening for additional client requests

## Introduction

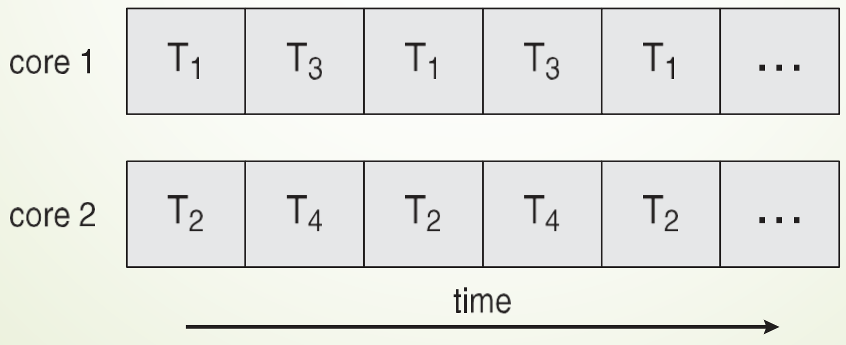
- Finally, most operating-system kernels are now multithreaded. Several threads operate in the kernel, and each thread performs a specific task, such as managing devices, managing memory, or interrupt handling.

## Multicore Programming

- **Multithreaded** programming provides a mechanism for more efficient use of these **multicore** or **multiprocessor** systems.



Concurrent execution on single-core system



Parallelism on a multi-core system

## Multithreading Models

- Support for threads may be provided either at the user level, for **user threads**, or by the kernel, for **kernel threads**.
- User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.

## Thread Concepts

- *The OS may employ different types of threads, depending on whether they are run (**user-mode** threads), or (**kernel-mode** threads).*

## User Mode vs Kernel Mode

- There are two modes of operation in the operating system to make sure it works correctly. These are **user mode** and **kernel mode**.

- **User mode** −Cannot access any hardware resources, which perform only the user operations.
- **Kernel-mode** −Can access hardware resources like RAM, Printer.

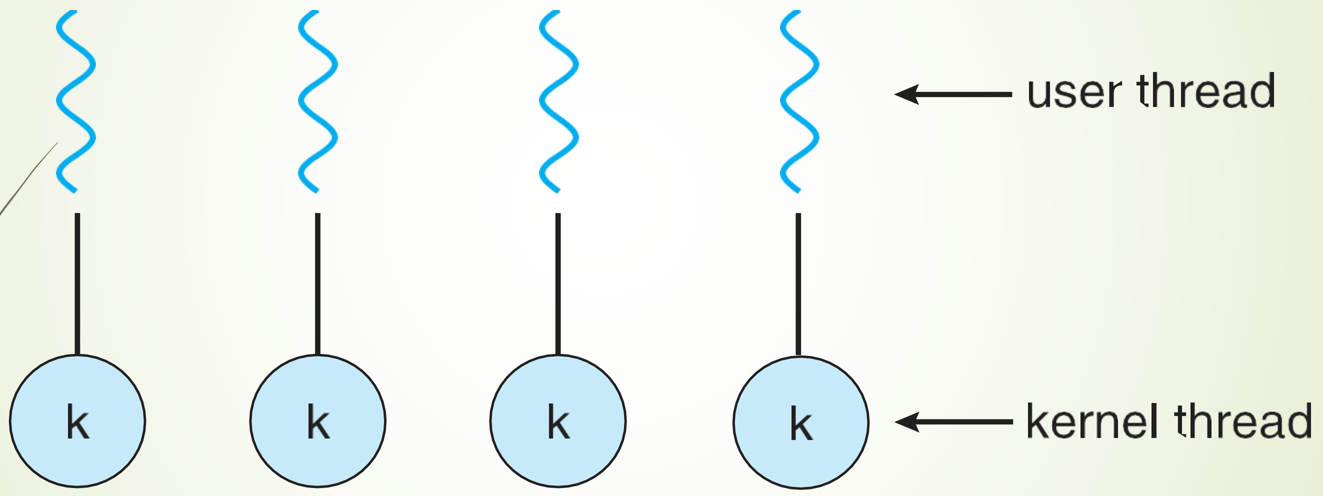
The system is in user mode when the operating system is running a user application such as handling a text editor. **The transition from user mode to kernel mode occurs when** the application requests the help of operating system or **an interrupt or a system call occurs.**

## Multithreading Models

- Ultimately, a relationship must exist between user threads and kernel threads.
- We look at three common models:
  - ➢ the one-to-one model,
  - ➢ the many-to-one model, and
  - ➢ the many-to-many model.

Multithreading Models

## One-to-One model (1/3)



## Multithreading Models

### One-to-One model (2/3)

- The one-to-one model maps each user thread to a kernel thread. It also allows multiple threads to run in parallel on multiprocessors.

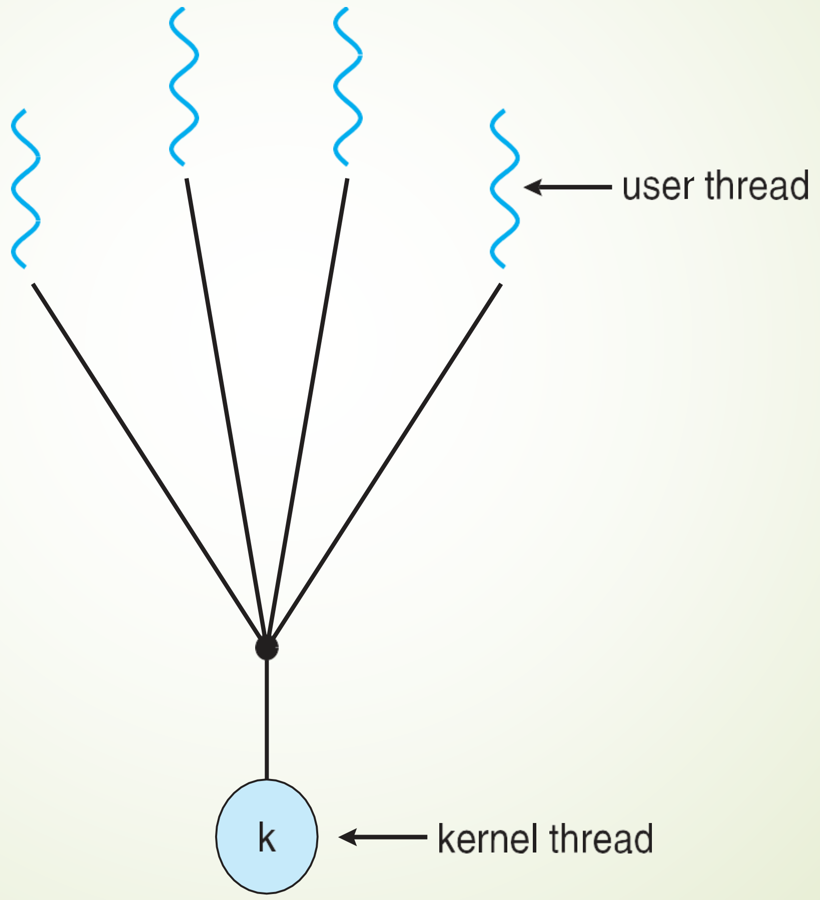
## Multithreading Models

### One-to-One model (3/3)

- The only **drawback** to this model is that **creating a user thread requires creating the corresponding kernel thread**. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model **restrict** the number of threads supported **by the system**. Linux, along with the family of Windows operating systems, implement the one-to-one model.

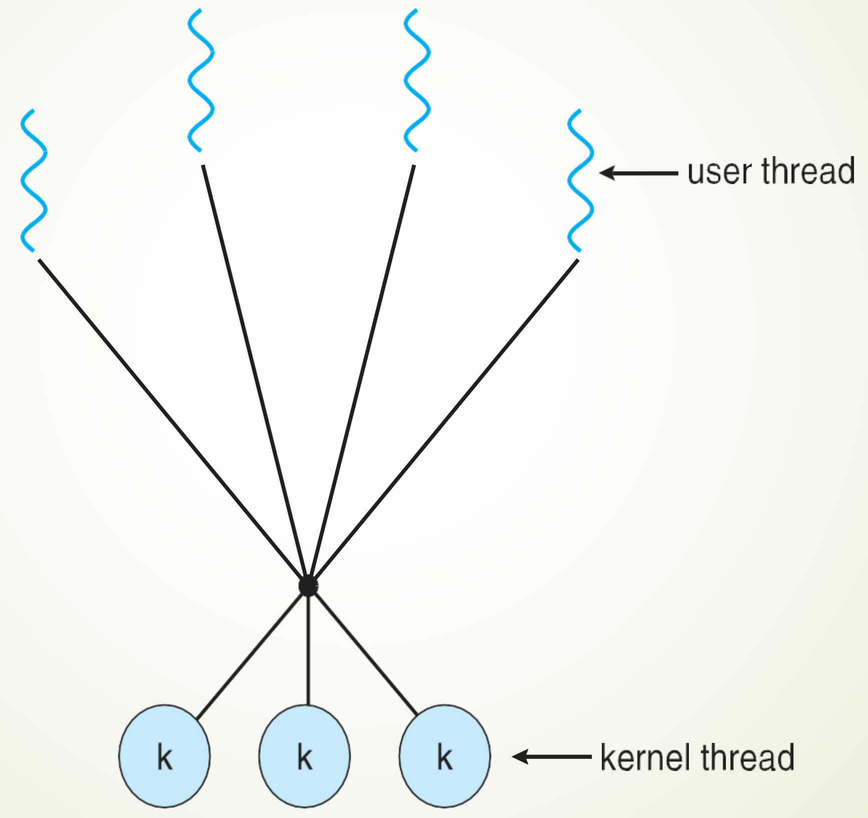
## Multithreading Models

### Many-to-One model



# Multithreading Models

## Many-to-Many model (1/2)



## Multithreading Models

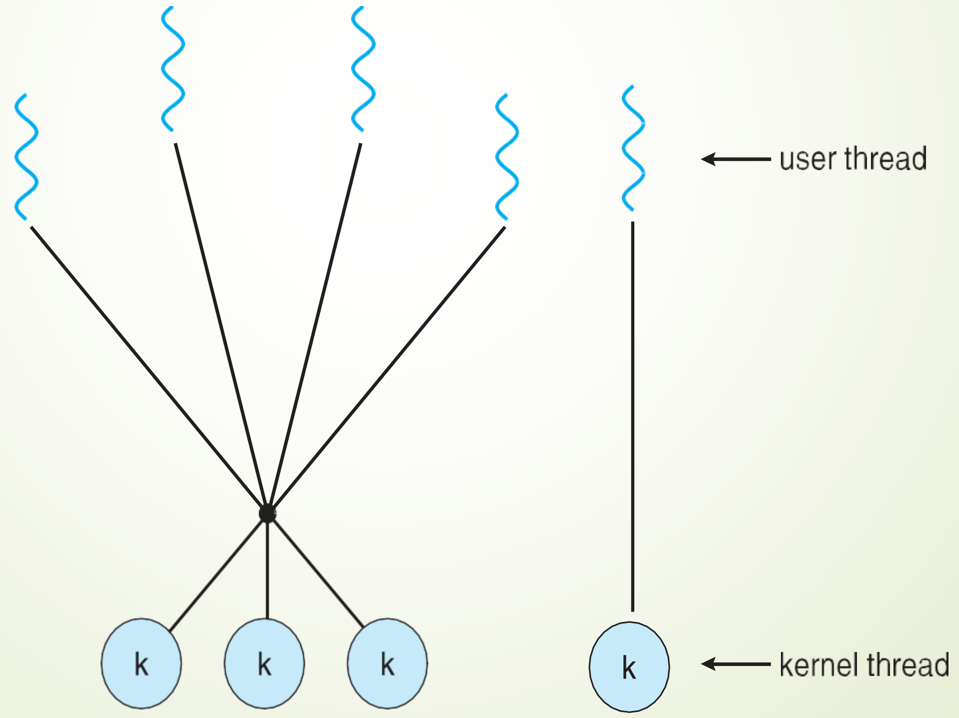
### Many-to-Many model (2/2)

- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- Allows the developer to create as many user threads as she wishes, it does not result in true concurrency, because the kernel can schedule only one thread at a time.

# Multithreading Models

## Two-level model

- Similar to Many-to-Many, except that it allows a user thread to be **bound** to kernel thread.



## Benefits of Threads

- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Communication:** Multiple-thread communication is simpler than process communication because the threads share the same address space, while in process.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files.

Ref: https://www.javatpoint.com/threads-in-operating-system

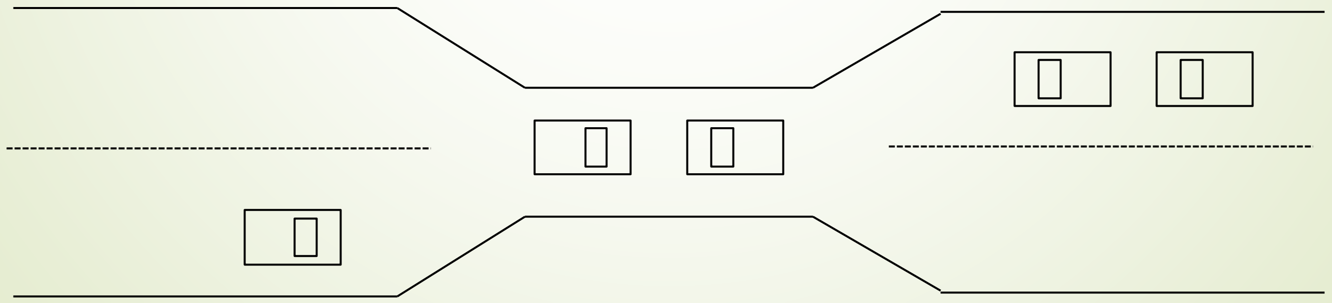
# Thread Libraries

- **Thread library** provides programmer with API for
  - creating and managing threads.
  - Three main thread libraries are in use today:
    - POSIX Pthreads,
    - Windows, and
    - Java.

| S/N | Process | Thread |
|---|---|---|
| 1 | Process is heavy weight or resource intensive. | Thread is light weight, taking lesser resources than a process. |
| 2 | Process switching needs interaction with operating system. | Thread switching does not need to interact with operating system. |
| 3 | In multiple processing environments, each process executes the same code but has its own memory and file resources. | All threads can share same set of open files, child processes. |
| 4 | If one process is blocked, then no other process can execute until the first process is unblocked. | While one thread is blocked and waiting, a second thread in the same task can run. |
| 5 | Multiple processes without using threads use more resources. | Multiple threaded processes use fewer resources. |
| 6 | In multiple processes each process operates independently of the others. | One thread can read, write or change another thread's |

## Introduction

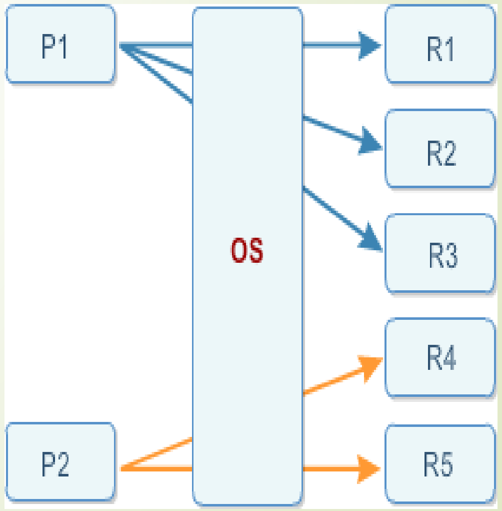
- Generally speaking, deadlock, involves conflicting needs for resources by two or more request orders. A common example is a traffic deadlock.
  - ➢ If deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
  - ➢ Several cars may have to back up if deadlock occurs.
  - ➢ Starvation is possible.



# Introduction

## Deadlock in computer system (1/2)

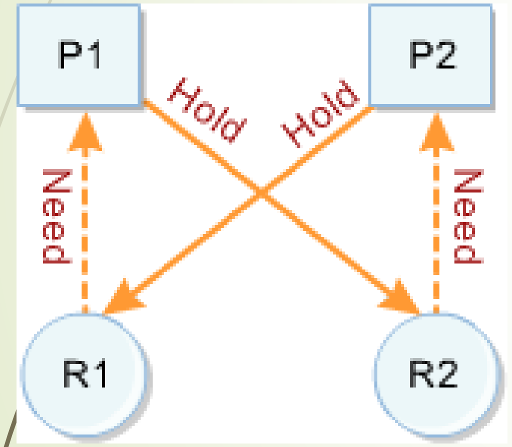
- A computer system consists of a **finite number of resources** to be distributed among a number of competing processes.
- An operating system is a **resource allocator** i.e., there are many resources that can be allocated to only one process at a time.
- Each process utilizes a resource as follows :
  - **request**
  - **use**
  - **release**

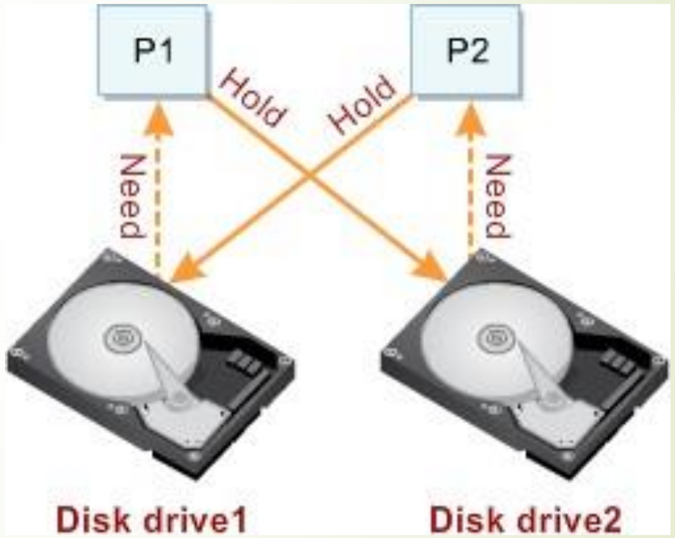


# Introduction

## Deadlock in computer system (2/2)

- General example of deadlock in a computer system:





## Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion.**
- **Hold and wait.**
- **No preemption.**
- **Circular wait.**

# Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

1. **Mutual exclusion**: only one process at a time can use a resource.
2. **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes.

# Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

3. **No preemption**: a resource can be released only willingly by the process holding it, after that process has completed its task.

4. **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that $P_0$ is waiting for a resource that is held by $P_1$, $P_1$ is waiting for a resource that is held by $P_2$, …, $P_{n-1}$ is waiting for a resource that is held by $P_n$, and $P_n$ is waiting for a resource that is held by $P_0$.

Thank you