



Processes

Presented by

Mr. T.M.Sonawane and Dr. G. D. Ramteke

Assistant Professor,
Institute of Management & Research,
Jalgaon (MS-India)-425 001

Content

A decorative horizontal border with a repeating geometric pattern of squares and lines.

- ✓ **Process Concept**
- ✓ **Process Scheduling**
- ✓ **Operation on Processes**
- ✓ **Inter-Process Communication.**
- ✓ **Communication in Client-Server Systems**

1.Process Concept :



- ✓ **Process/Job** is active entity with **Program Counter** ,
Specify
 - 1) Next instruction to be execute.
 - 2) Resource required for them for execution.
- ✓ **Process Means execute single/several program(like notepad ,web-browser etc.) at a Time.**

1. Process Concept :

- ✚ A **program which is in execution** is called **Process**.
- ✚ Each **process is identified by** process number is called **Process ID**.
- ✚ **Process Hierarchy or tree :**
Each **process can create another process** and these created process create another process and so on.



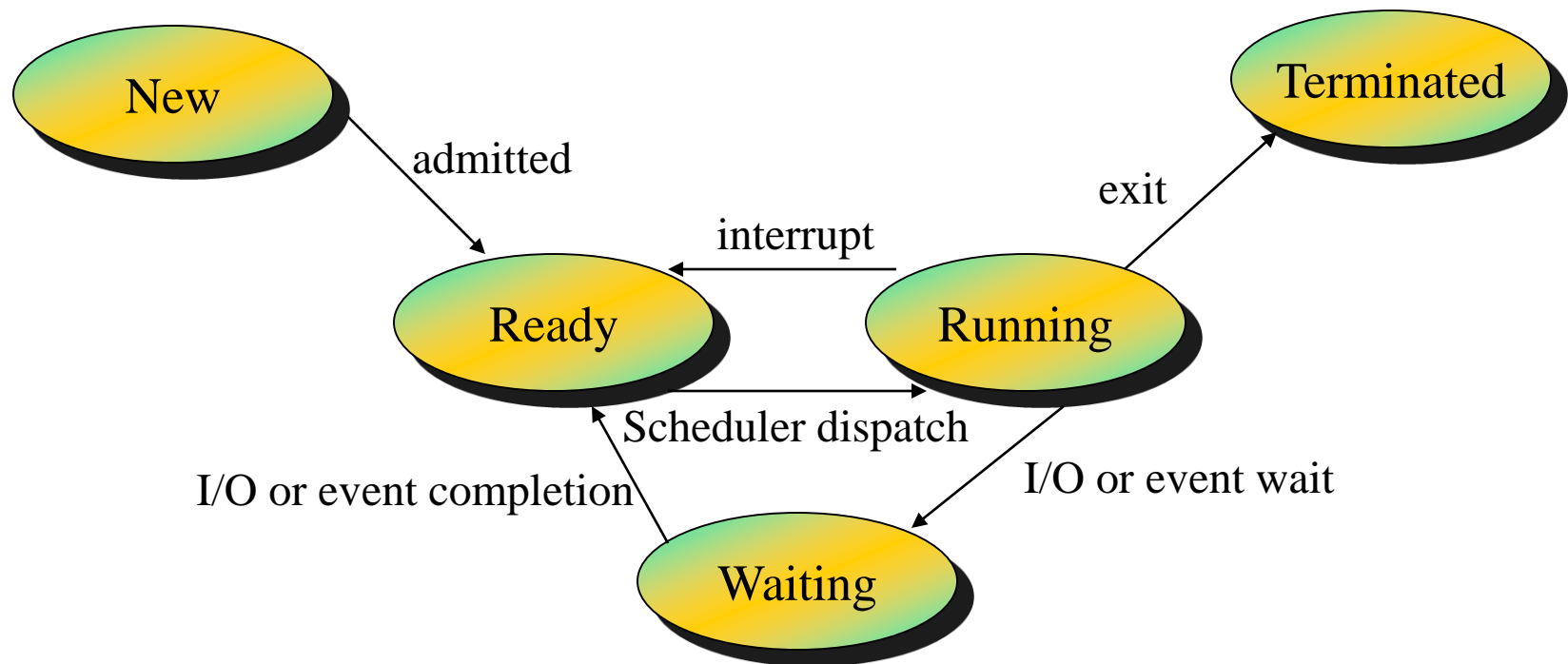
❑ Process Concept :


1.1 Process State :

- ✚ As a **process executes**, it **changes state**.
- ✚ It is used to **store the current activity/status** of any process.



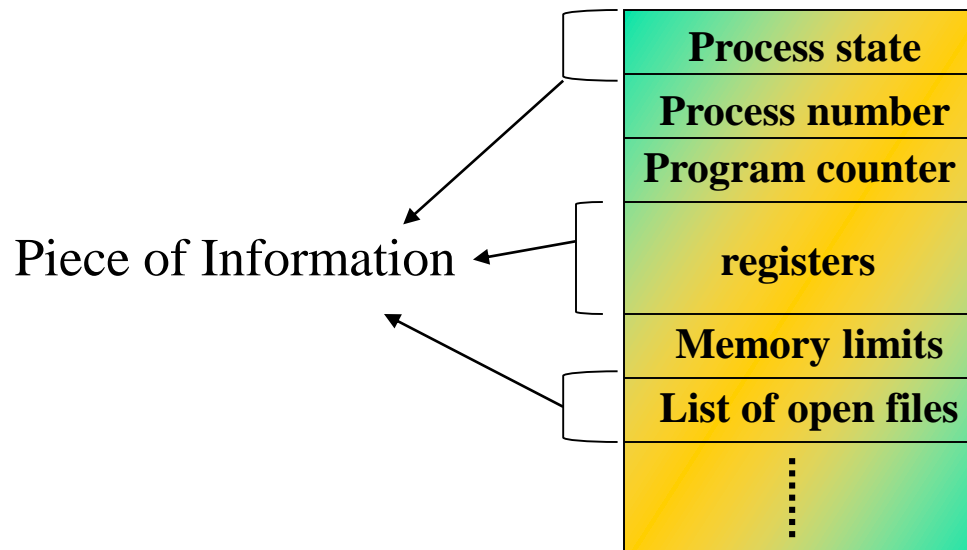
✚ Each process may be in one of the following states :



- 
- New.** The process is being created.
 - Running.** Instructions are being executed.
 - Waiting.** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
 - Ready.** The process is waiting to be assigned to a processor.
 - Terminated.** The process has finished execution.

1.2 Process control Block(PCB) :

- ✚ Each **process** is **represented** in operating system by a **PCB**.
- ✚ **PCB** is also called a *task control block*.
- ✚ PCB provides a **data structure** for process.



Data Structure : how data is stored actual.

Fig. Process Control Block

1. Process State

The state may be new, ready, running, waiting, halted, and so on.

2. Process Number : Process ID

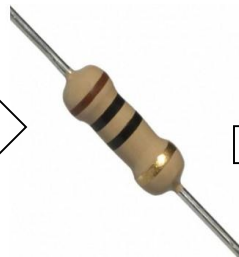
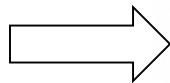
3. Program counter

The counter indicates the address of the next instruction to be executed for this process.

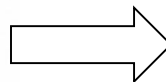
4. CPU registers :



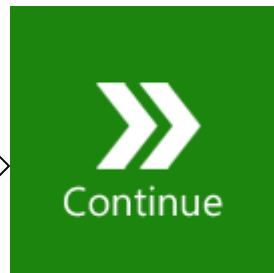
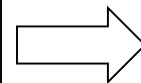
Process



register



Store prog-
counter and
info



Process



5. CPU-scheduling information

This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

6. Memory-management information

This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

7. Accounting information.

This information includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on.

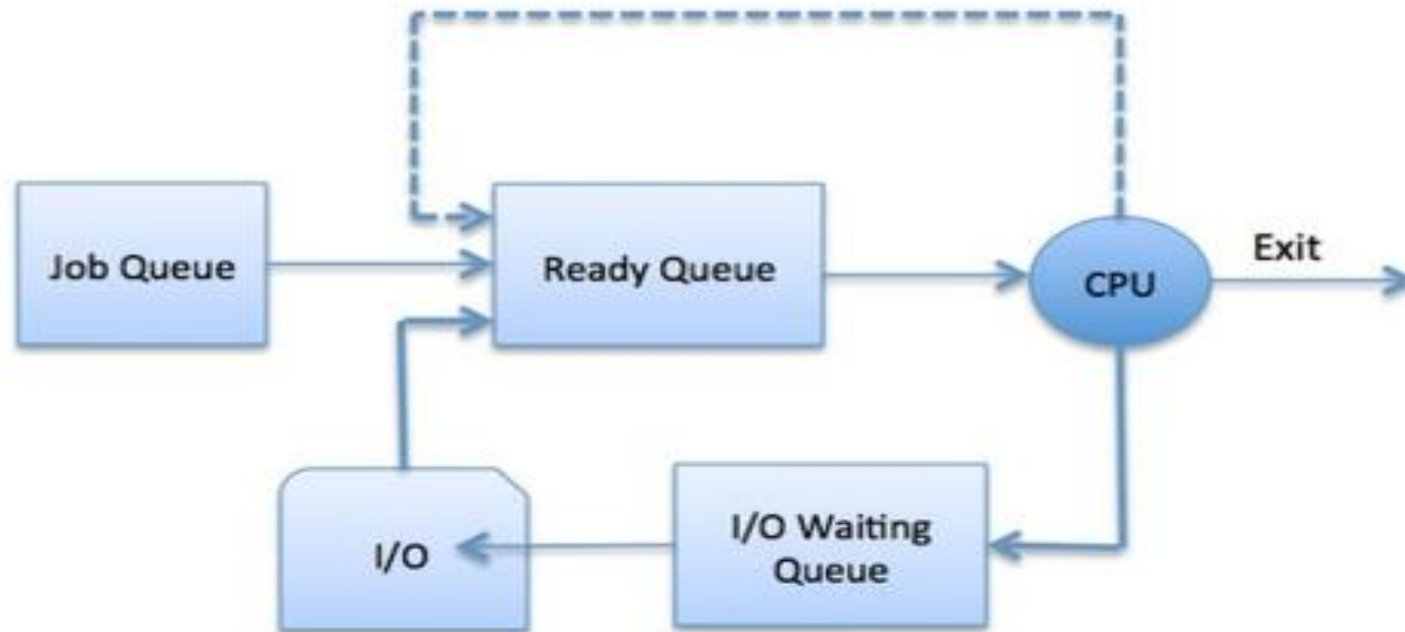
8. I/O status information

list of I/O devices allocated to the process, a list of open files.


2. Process Scheduling :



2. Process Scheduling :



- ✓ The OS maintains all PCBs in Process Scheduling Queues.



✓The Operating System maintains the following important process scheduling queues :-

- 1) **Job queue** – This queue keeps all the processes in the system.
- 2) **Ready queue** – This queue keeps a set of all processes residing in main memory. Every new process Put in this queue.
- 3) **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.

3. Operations on Processes : Concurrent Process :

✚ Processes executing concurrently in the operating system.

✚ There are **two types** of concurrent process,

1. Independent Process



Person

Don't
Share Info



Unknown Guy

2. Cooperating process

Share Info
with friends



3. Operations on Processes : Concurrent Process :

- ✚ Processes executing concurrently in the operating system.
- ✚ There are two types of concurrent process,
 1. Independent Process
 2. Cooperating process
- ✚ A process is **independent** if it cannot affect by the any other processes executing in system.(i.e **processes can't share data with one another**)
- ✚ A process is **cooperating** if it can affect by the any other processes executing in system. (i.e **processes can share data with one another**)

4. Inter-Process Communication(IPC) :

✚ Cooperating processes require an inter-process communication.



Fig: Admission in Foreign college

- ✚ It's mechanism that allow processes to exchange data and info.
- ✚ It is a method which allow processes to communicate and to synchronize there actions .

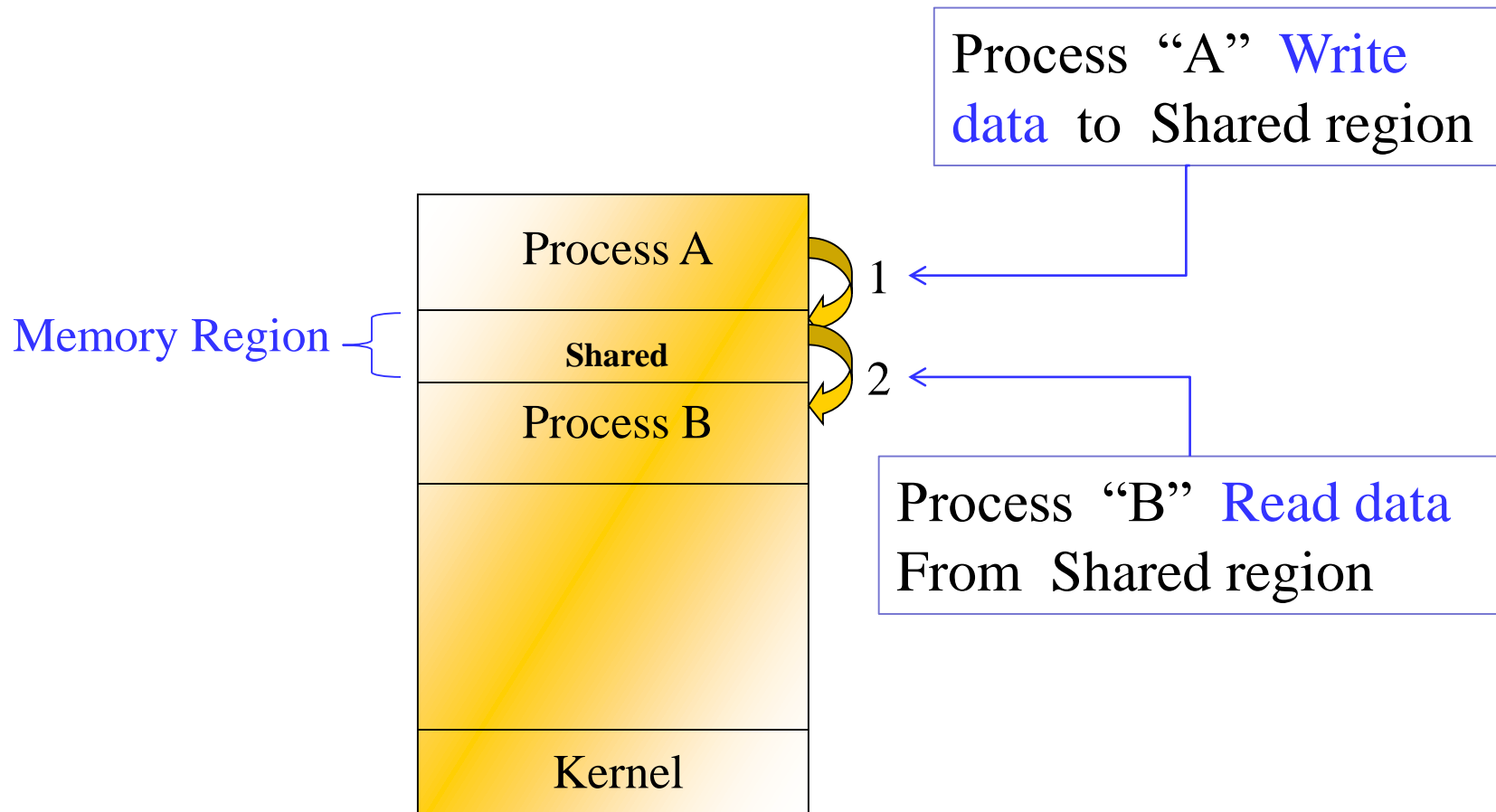
4. Inter-Process Communication(IPC) :

- ✦ There are two fundamental models of interprocess communication,
 1. Shared memory
 2. Message Passing

1. Shared memory system:

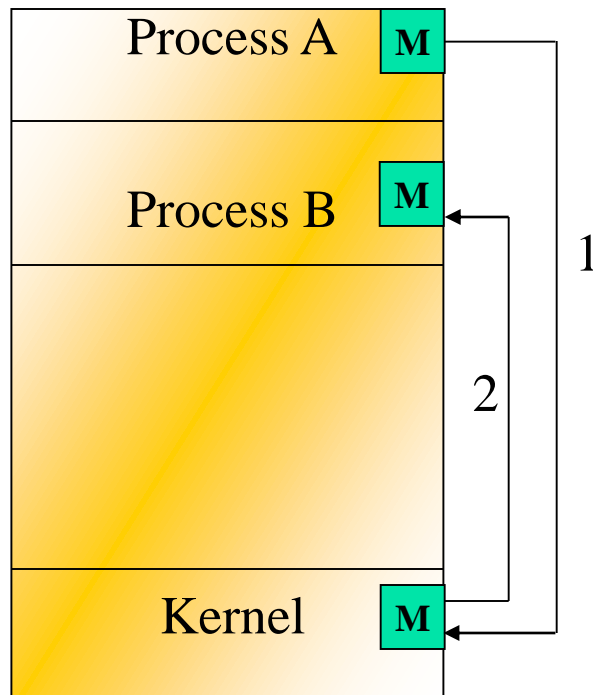
- ✦ Region of memory is shared by cooperating processes.
- ✦ process's can exchange information by reading and writing data to the shared region.
- ✦ Shared memory is faster than message passing.
- ✦ No kernel is required for memory accesses.

+ processes can exchange information by reading and writing data to the shared region.



2. Message passing system:

- ✓ A **communication** takes place **by** means of **messages** exchanged between the cooperating processes.
- ✓ Message passing is **useful for exchanging smaller amounts of data.**



Message passing facility **provides** at least **two operations**,

1. **Send(message)**
2. **Receive(message)**

- Message passing require more time consuming task of kernel intervention.

5. Communication in Client – Server System :



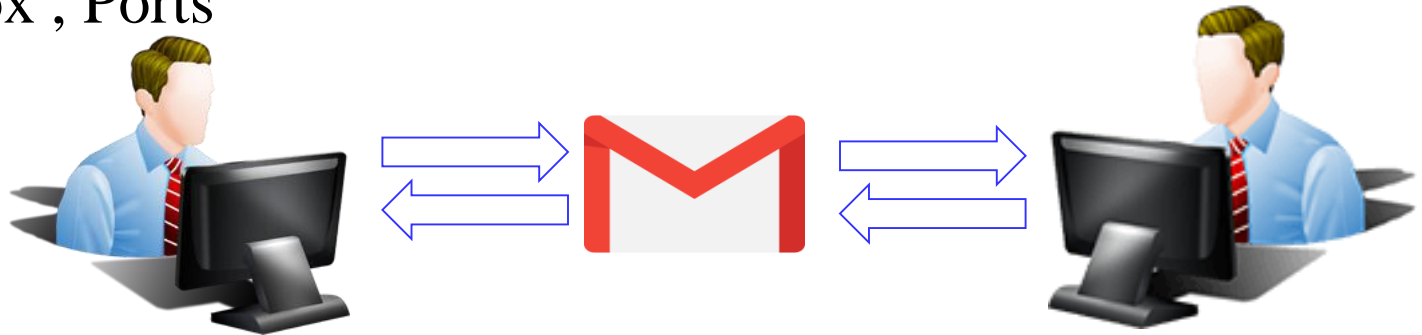
1. Direct or Indirect communication

✓ Under **direct communication**, each **process** that wants to communicate must **explicitly name the recipient or sender** of the **communication**.



Here **friends** are **directly call to each other**

- ✓ Under **indirect communication**, the **messages** are **sent to and received**
- ✓ E.g. Mailbox , Ports



2. Synchronous or Asynchronous communication

- Communication between processes takes place through calls `Send()` and `receive()` primitives.
- Message passing may be either synchronous and asynchronous.

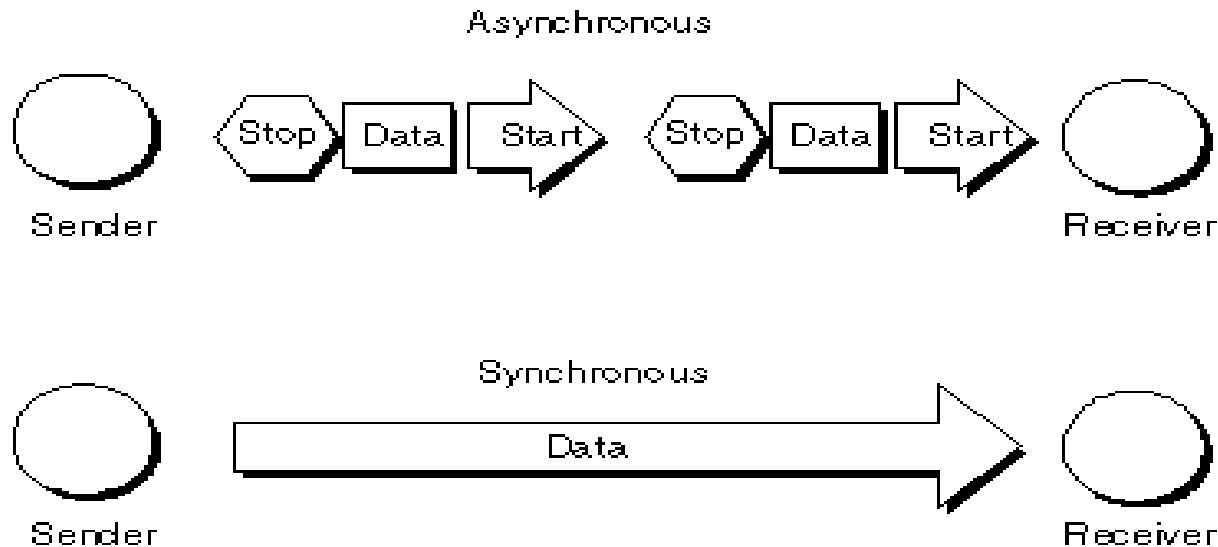
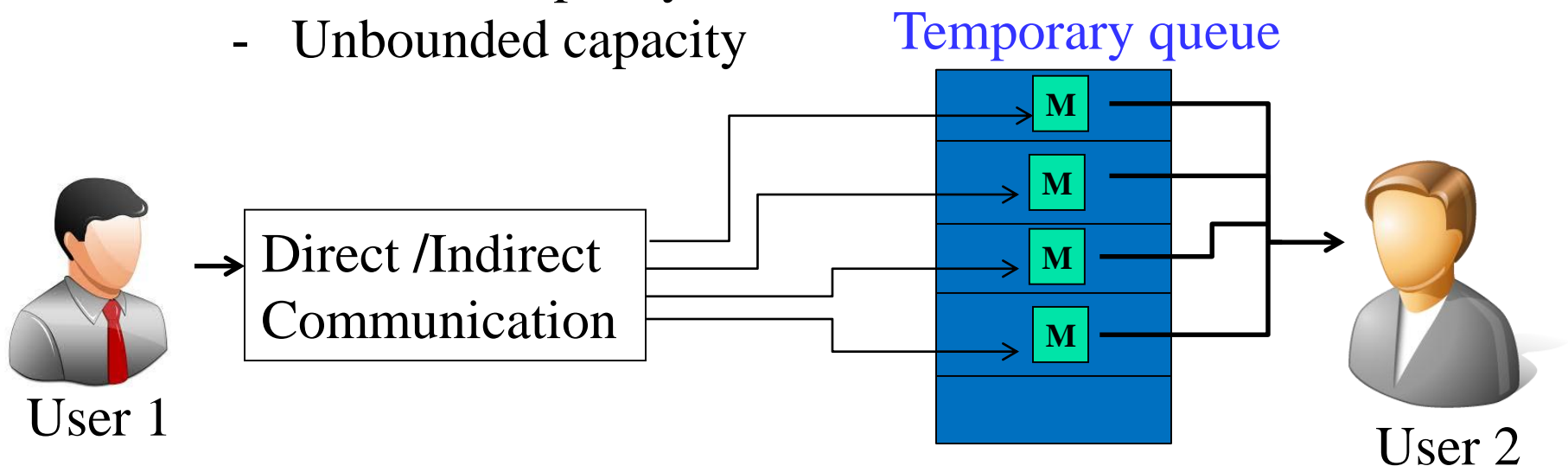


Fig : Communication Synchronous and Asynchronous


3. Automatic or explicit buffering

Whether **communication** is direct or indirect, messages exchanged by communicating processes **reside in a temporary queue**. This **temporary queue** is called **buffering**. This **queue** can be **implemented** in the **three ways**.

- Zero capacity
- Bounded capacity
- Unbounded capacity



Thread :




✚ It is nothing but a mini-process. *A thread is a single sequence stream within in a process.*

✚ A thread is basic unit of CPU utilization, it comprises a thread ID, a program counter, a register set and a stack.

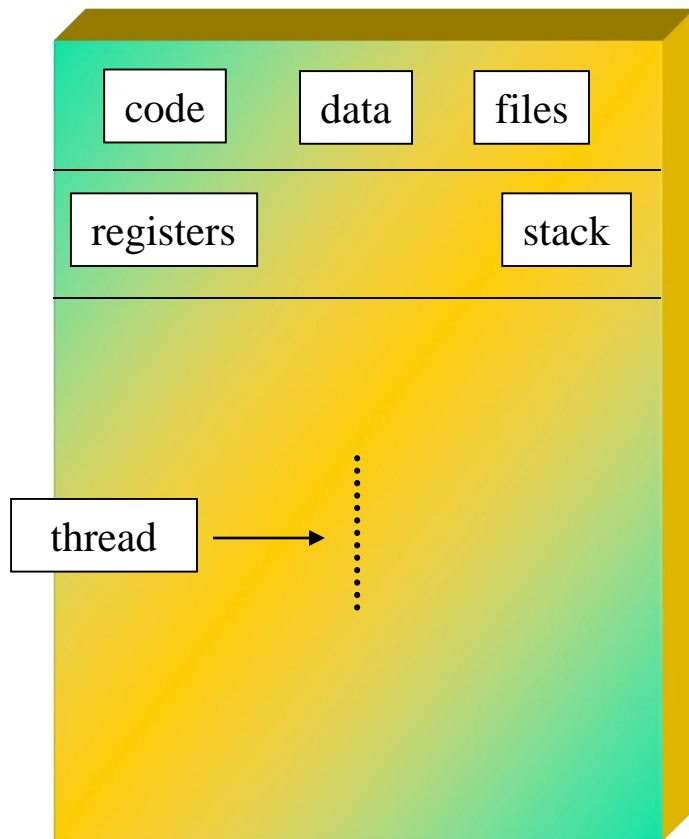
✚ For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time.

✚ Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.

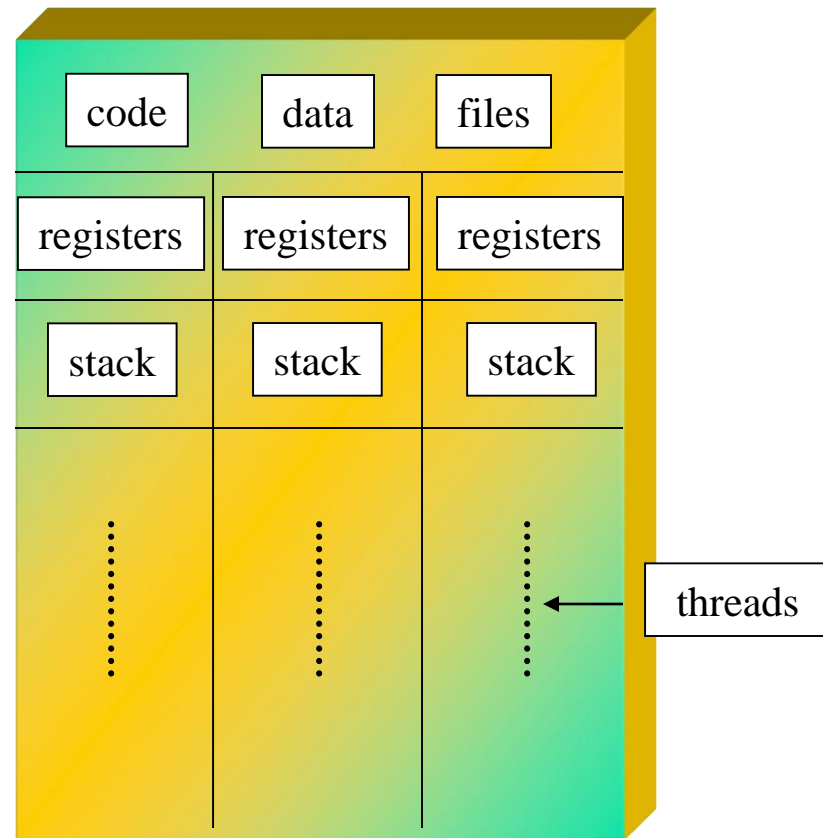
- 
- ✚ In any operating system each process has an address space of single thread or multiple threads of control.
 - ✚ Thread can create child threads also.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.



Single-threaded process



multi-threaded process

Types of Thread :

Threads are implemented in following two ways –


1. User Level Threads –

User managed threads.

2. Kernel Level Threads –

Operating System managed threads acting on kernel, an operating system core.

1. User Level Threads – User managed threads.

A decorative horizontal border consisting of a repeating geometric pattern, possibly a Greek key or meander design, in a dark color.

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Advantages


- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.



Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

2. Kernel Level Threads –

A decorative horizontal border consisting of a repeating geometric pattern, resembling a Greek key or meander design, in black and gold colors.

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages


A decorative horizontal border with a repeating geometric pattern of squares and lines.

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

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models :

A decorative horizontal border consisting of a repeating geometric pattern of squares and lines.

Some operating system provide a combined user level thread and Kernel level thread facility.

Solaris is a good example of this combined approach.

In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

Multithreading models are three types

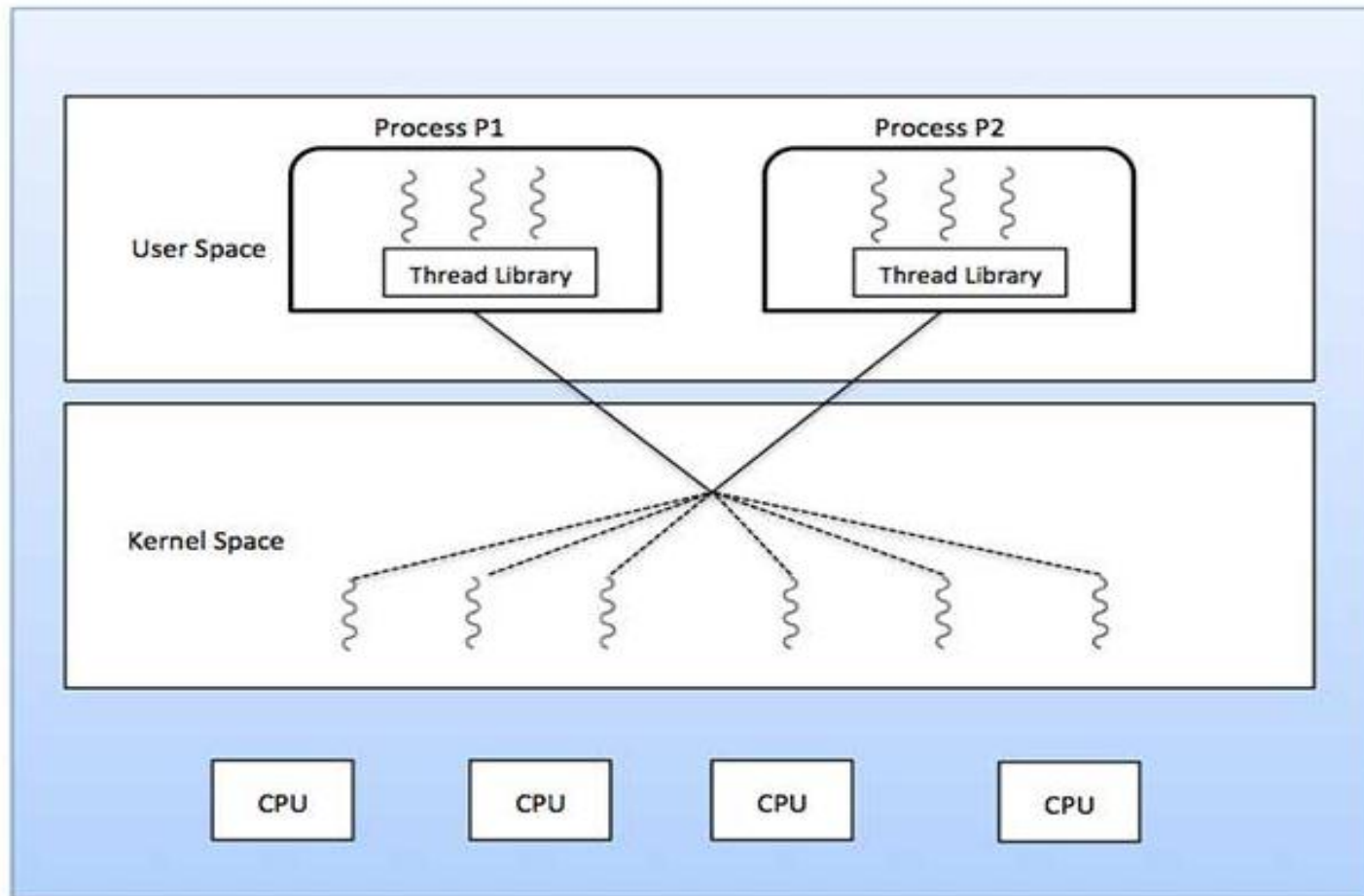
- Many to many relationship.
- Many to one relationship.
- One to one relationship.

1. Many to Many Model

A decorative horizontal border with a repeating geometric pattern of squares and lines.

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Many to Many Model



2. Many to One Model

A decorative border consisting of a repeating geometric pattern, resembling a Greek key or meander design, in black and gold.

Many-to-one model maps many user level threads to one Kernel-level thread.

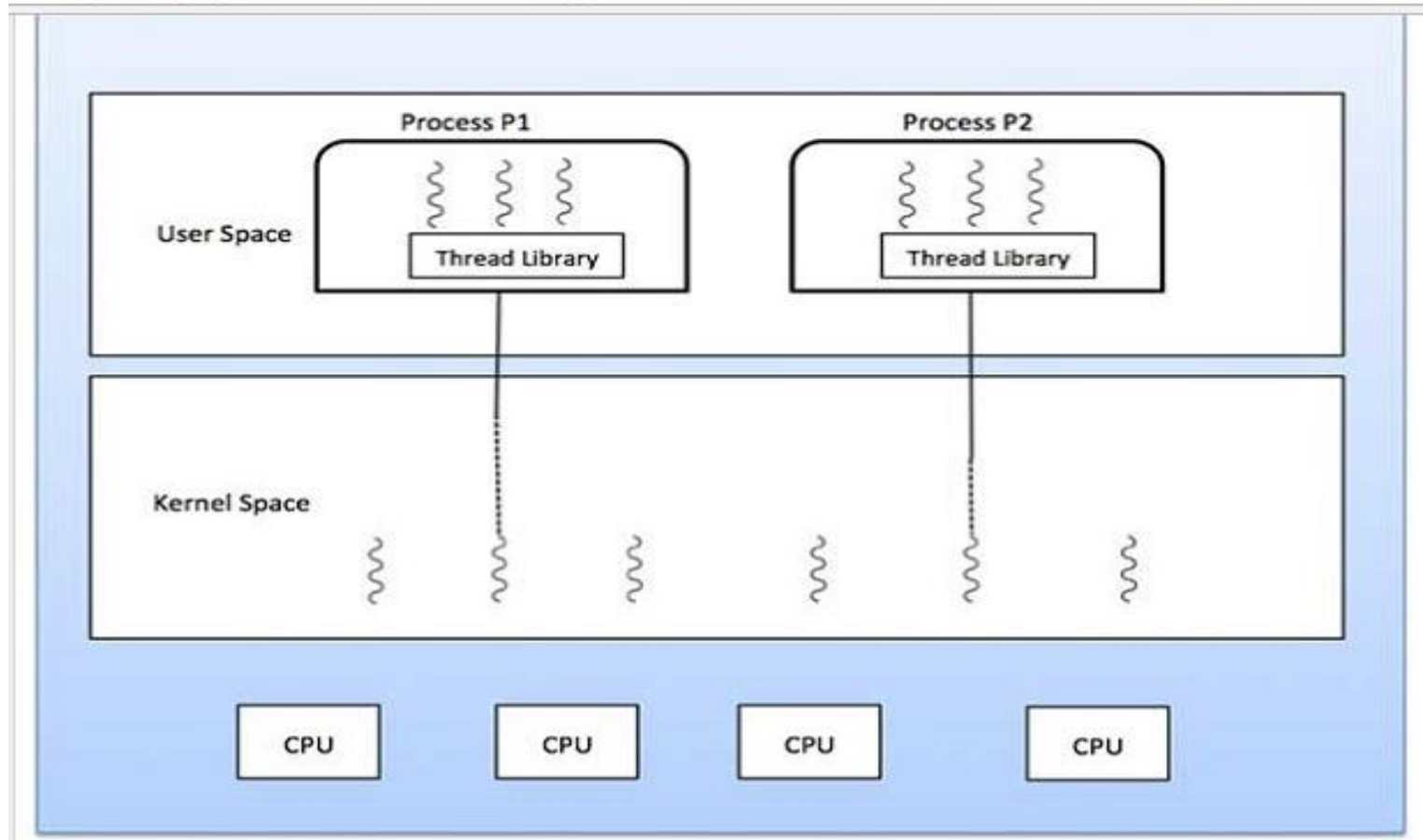
Thread management is done in user space by the thread library.

When thread makes a blocking system call, the entire process will be blocked.

Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

Many to One Model



3. One to One Model



There is one-to-one relationship of user-level thread to the kernel-level thread.

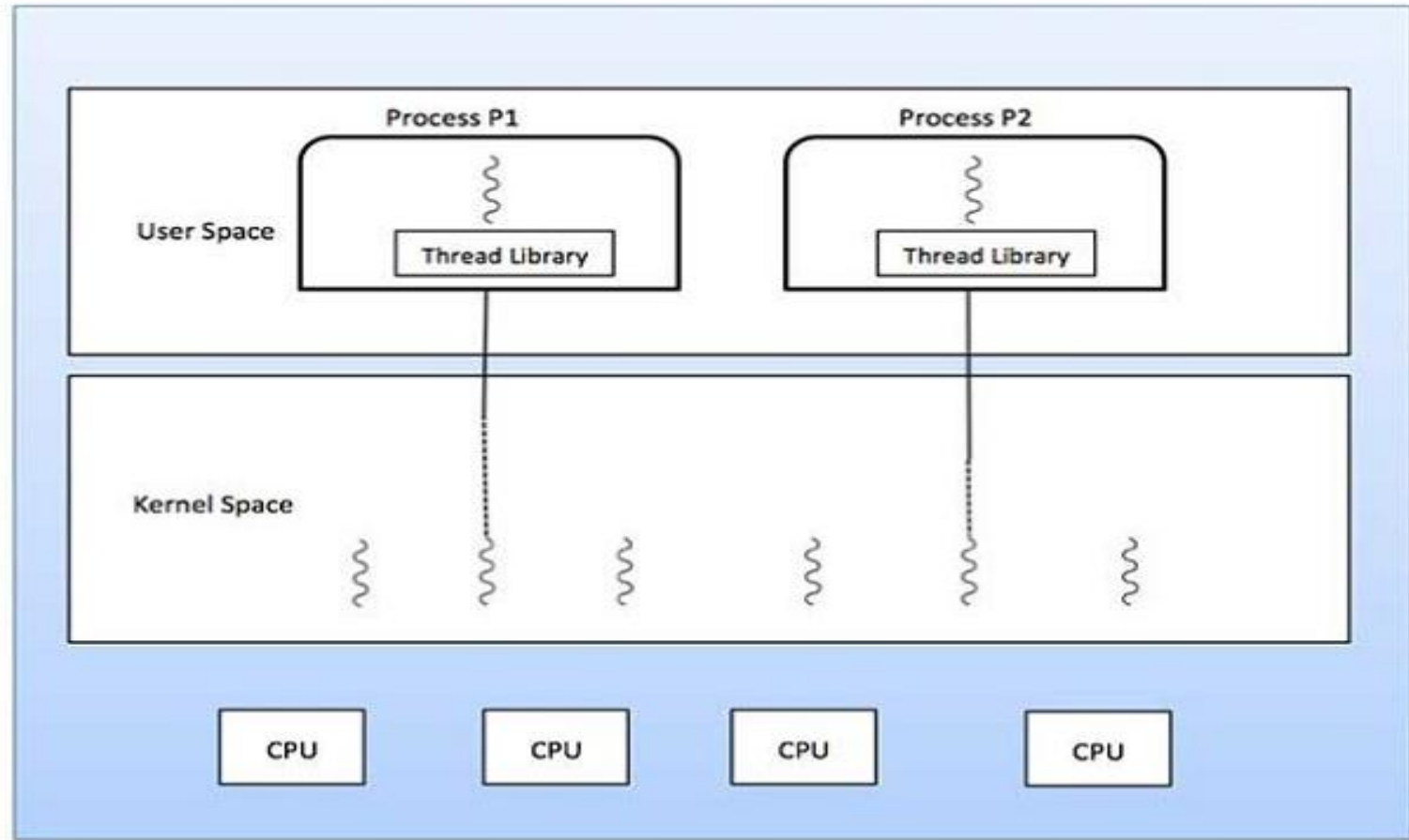
This model provides more concurrency than the many-to-one model.

It also allows another thread to run when a thread makes a blocking system call.

It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

One to One Model



Difference between Process and Thread

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 data.