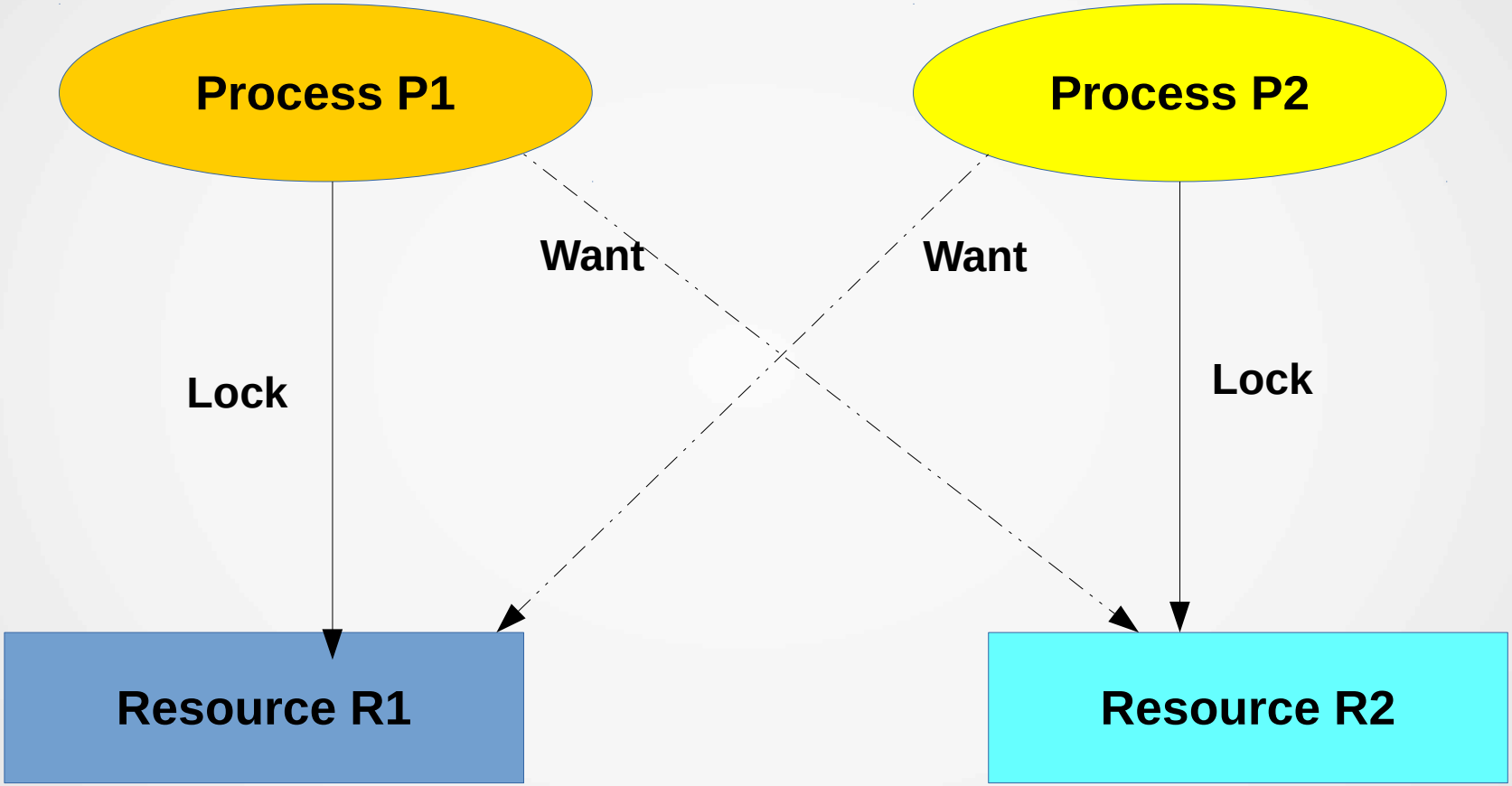
Deadlocks

- If there is no control on the competing processes for accessing multiple resources, then it can lead to a severe problem in the system.
- For example, a Process P1 is accessing a Resource R1 and needs another Resource R2 to complete its execution, but at the same time, another Process P2 is holding a Resource R2 and requires R1 to proceed.
- Here, both the processes are waiting for each other to release the resources held by them. This situation is called a deadlock.
- There can be many situations like this causing deadlock problems in the system

Deadlocks


Process P1
Process P2
Want
Want
Lock
Lock
Resource R1
Resource R2

DeadLock – Defining a DeadLock

- The computer system uses several types of resources, such as consumable or non-consumable, and pre-emptable or non-pre-emptable.
- In a multi-programming environment, the problem starts with non-pre-emptable resources. These resources have to be used in a mutually exclusive manner. But the problem is that resources are always limited, compared to the number of processes.
- In this case, when the concurrent processes request the resources, a deadlock occurs.

- In general, the OS follows a protocol to use non-pre-emptive resources. The protocol consists of the following three events:
- **1. Request**: Before using a resource, a process must request for it. This request is in the form of a system call, and it is passed on to the kernel. This uses a data structure resource table to store the information about all the resources in the system. When- ever the request is received, it checks the status of the resource. If the resource is free, the requesting process will get it. Otherwise, the requesting process will be blocked, and it must wait until the resource is free.

  2.Use: If a process gets the resource, the kernel changes the status of resource to allocated and the state of process to ready, if it has been blocked earlier. The process uses the resource in a mutually exclusive manner.

  3.Release: After its use, a process releases the resource again through the use of a system call. The kernel can now allocate this resource to any blocked process or change the status of the resource to free.

Conditions for Deadlock

- There are certain conditions that give rise to a dead lock. The following are the conditions:
  - Mutual Exclusion
  - Hold and Wait
  - No Pre-emption
  - Circular Wait

Conditions for Deadlock

- It is not necessary that a deadlock will always occur in concurrent process environments. In fact, there are certain conditions that give rise to a deadlock. The following are the conditions:

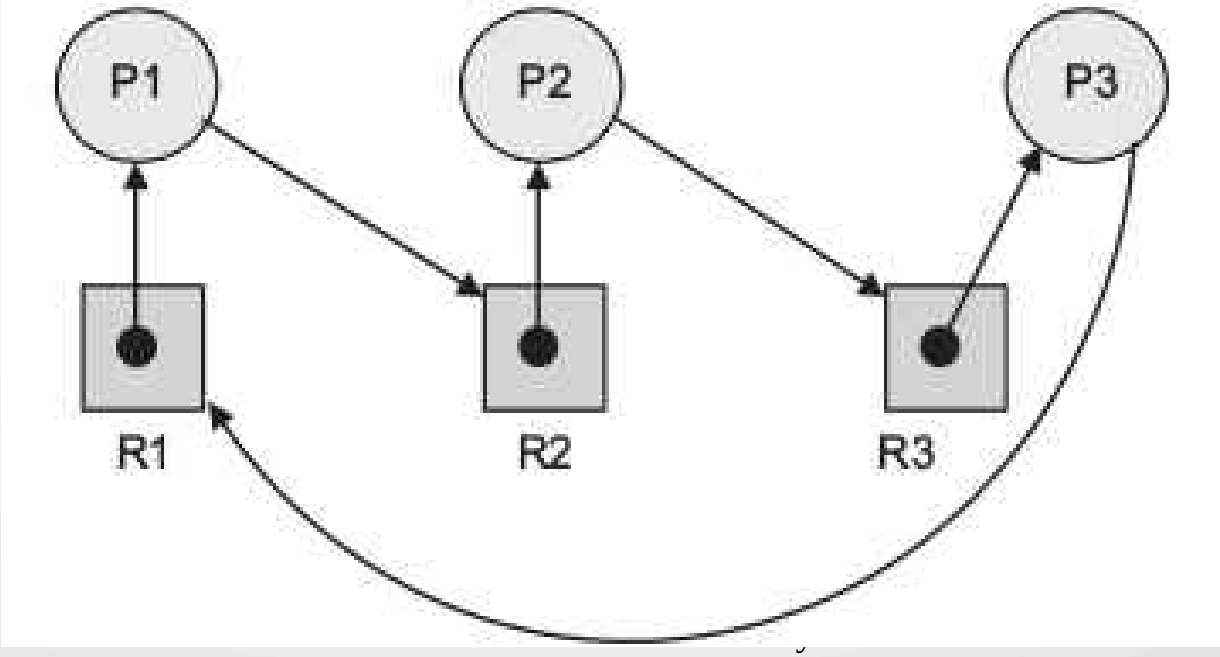
  1 Mutual Exclusion - The resources which require only mutually exclusive access, may give rise to a deadlock. These resource types cannot allow multiple processes to access it at the same time. For example, a memory location, if allocated to a process, cannot be allowed to some other process. A printer cannot print the output of two processes at the same time. Therefore, a printer, if allocated to a process, cannot be allocated to another.

  2 Hold and Wait - When all the processes are holding some resources and waiting for other resources, a deadlock may occur.

  3 No Pre-emption - The resources in a deadlock situation are of non-pre-emptable nature, as discussed above. If a resource cannot be pre-empted, it may lead to a deadlock.

Conditions for Deadlock

- These three conditions together give rise to a fourth condition. A chain of processes may be produced, such that every process holds a resource needed by the next process. In this way, a circular chain of processes is built up in the system, due to mutually exclusive and non-pre-emptable nature of resources.



Dealing With Deadlock

- Every system may be prone to deadlocks in a multi-programming environment, with few dedicated resources. To deal with the deadlock, the following three approaches can be used:
- **Deadlock Prevention:** This method is very idealistic. It is based on the fact that if any of the four necessary conditions is prevented, a deadlock will not occur.
- Although it is very difficult to achieve this kind of prevention, this method is better, in the sense that the conditions for deadlock will not be allowed to arise in the system. Moreover, a principle is established that a discipline is maintained, while the processes request for the resources.

Deadlock Avoidance

- **Deadlock Avoidance:** Deadlock prevention is not always possible. Another method to deal with the deadlock is to avoid it.
- But avoidance is possible when there is complete knowledge in advance, about which resources will be requested and used by the processes.
- At any instant of time, information such as, how much resources are available, how many are allocated to the processes, and what the future requests are, must be known.
- With this information, a process may be delayed, if its future request may produce a deadlock. In this way, the deadlock may be avoided.

Deadlock Detection & Recovery

- **Deadlock Detection and Recovery:** If a system does not employ either prevention or avoidance methods, then a deadlock may occur.
- In this case, it becomes necessary to detect the deadlock. At least, it must be known that a deadlock has occurred in the system.
- The processes,due to which the deadlock has occurred, should also be known.
- We should have a mechanism to detect the deadlock in the system and resolve it, that is, recover from the deadlock situation.

Thread

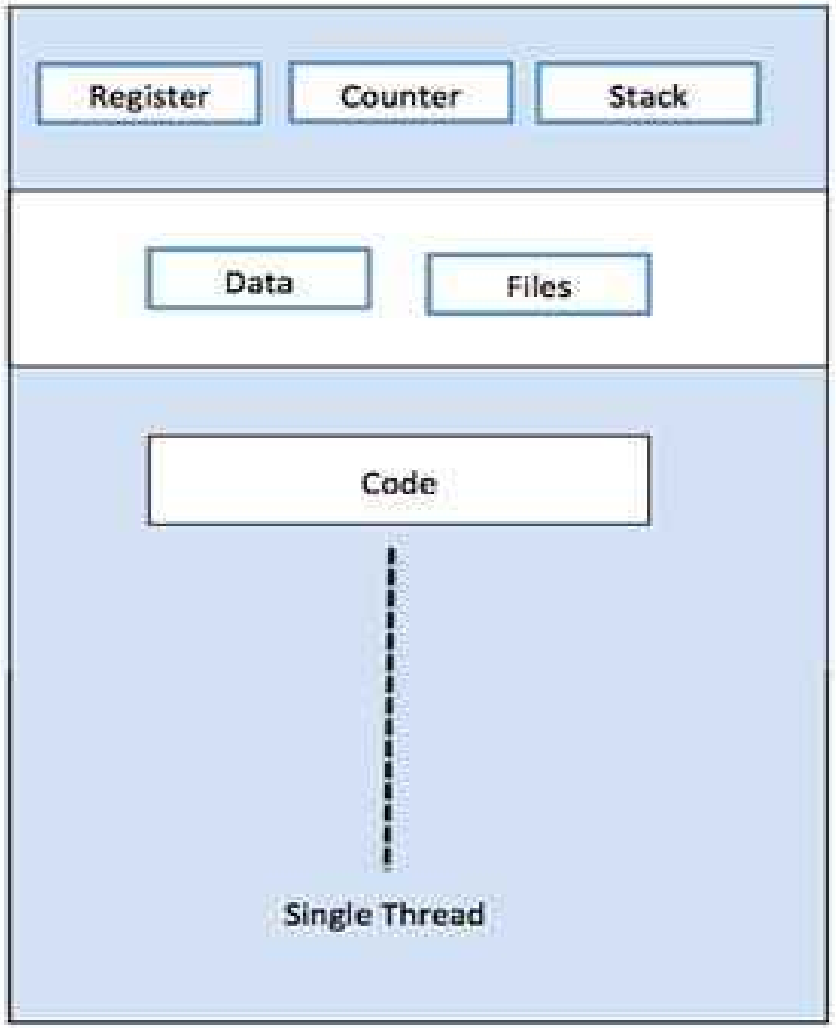
Thread : A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

- A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

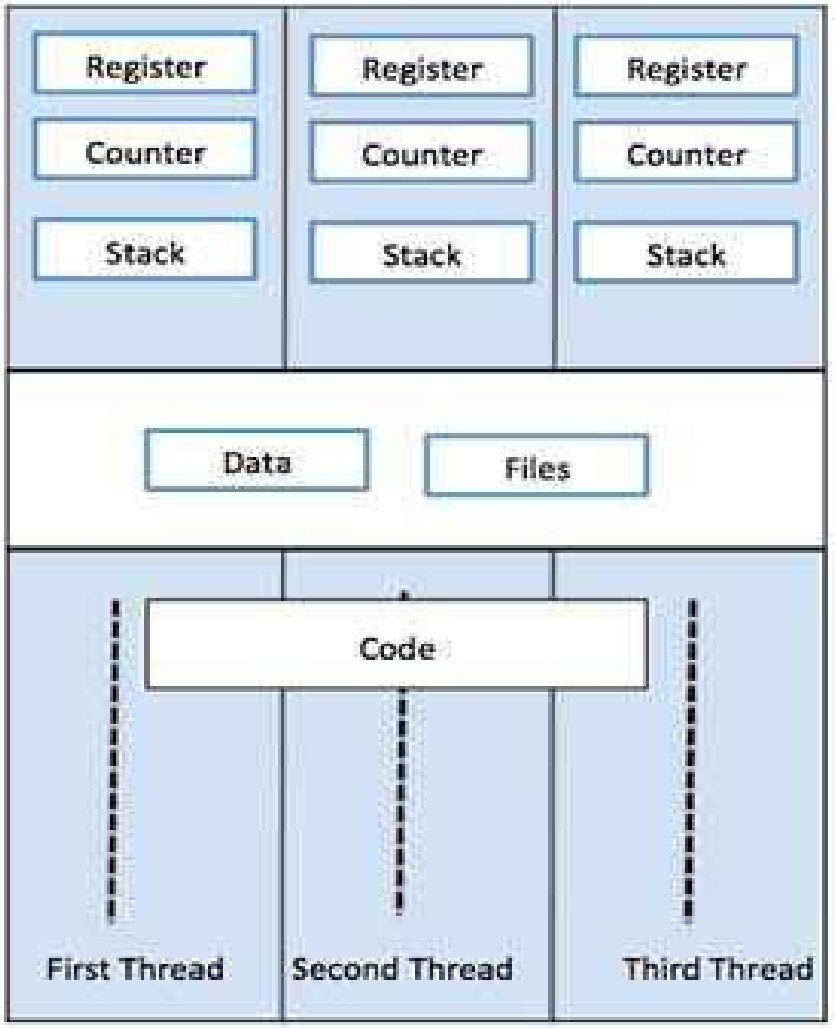
Thread

- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

Thread



Single Process P with single thread



Single Process P with three threads

Difference : Process -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.

Multitasking vs Multi threading

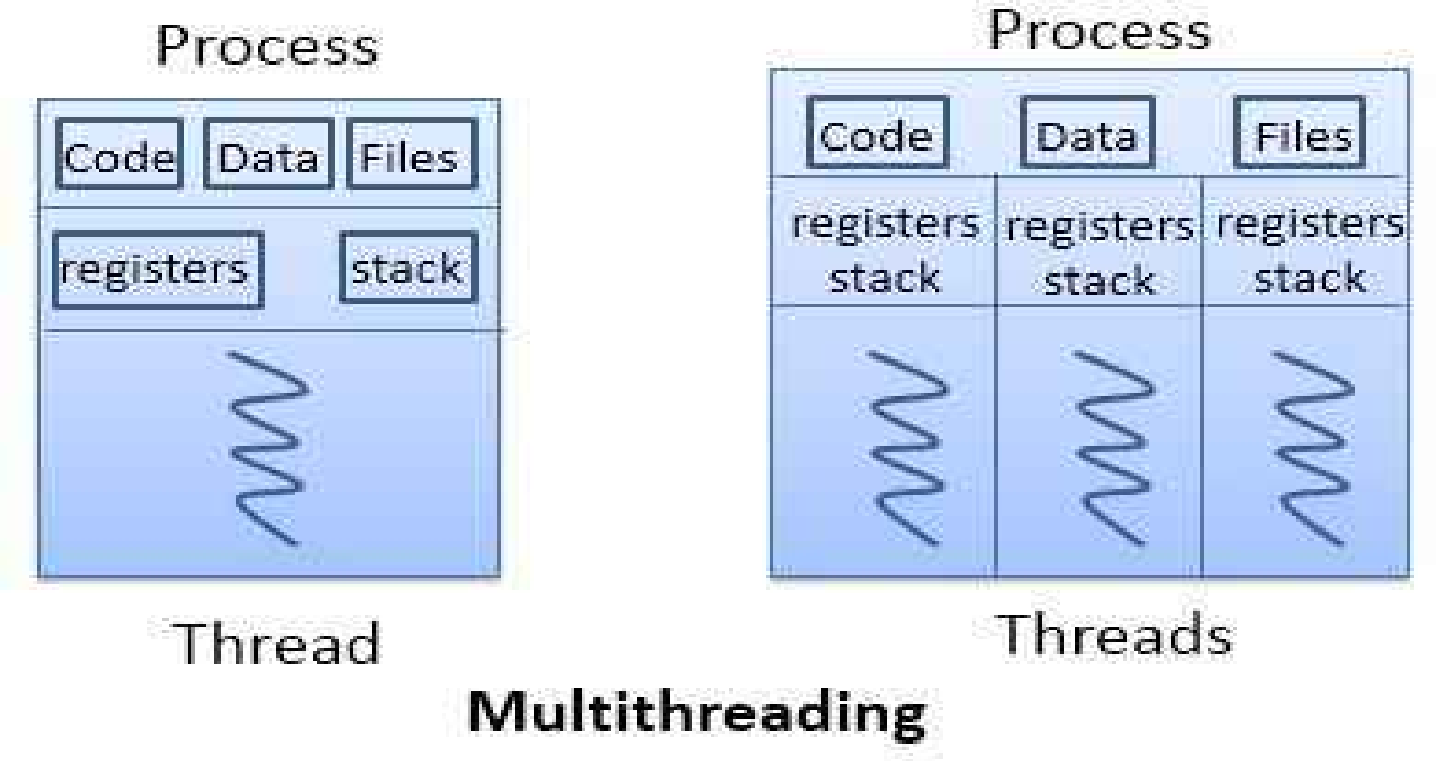
- *"The concept of implementing multiple threads to achieve concurrency within a single process is known as multi-threading."*
- Multi threading is implemented at thread level.
- Multi tasking or Multi user is implemented at process level.

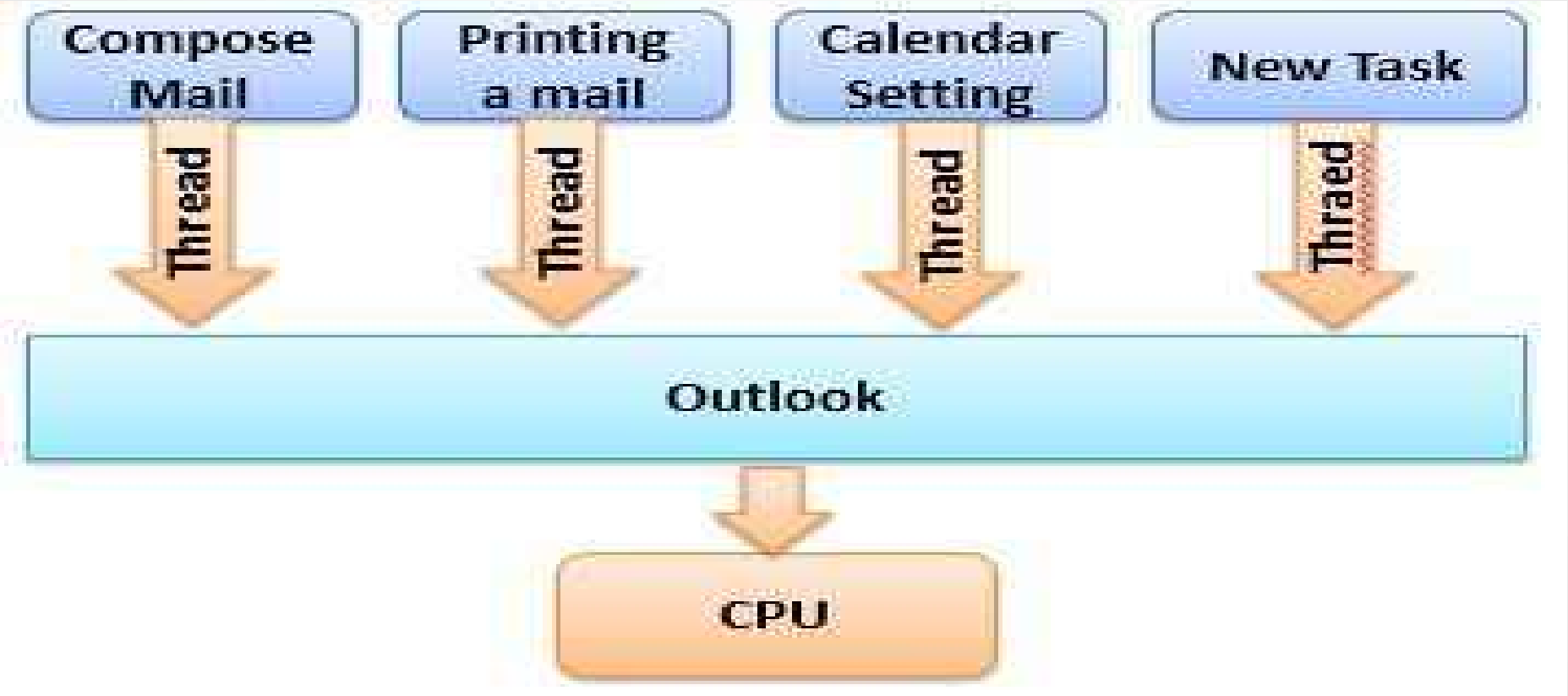
Multitasking vs Multi threading

- Multitasking is when a single CPU performs several tasks (program, process, task, threads) at the same time. To perform multitasking, the CPU switches among theses tasks very frequently so that user can interact with each program simultaneously.

Multitasking

- Multithreading is different from multitasking in a sense that multitasking allows multiple tasks at the same time, whereas, the Multithreading allows multiple threads of a single task (program, process) to be processed by CPU at the same time.




Compose Mail
Printing a mail
Calendar Setting
New Task
Thread
Thread
Thread
Thraed
Outlook
CPU

BASIS FOR COMPARISON	MULTITASKING	MULTITHREADING
Basic	Multitasking let CPU to execute multiple tasks at the same time.	Multithreading let CPU to execute multiple threads of a process simultaneously.
Switching	In multitasking CPU switches between programs frequently.	In multithreading CPU switches between the threads frequently.
Memory and Resource	In multitasking system has to allocate separate memory and resources to each program that CPU is executing.	In multithreading system has to allocate memory to a process, multiple threads of that process shares the same memory and resources allocated to the process.

Thread Control Block

- Just like the PCB, in multi-threading, there is a thread control block (TCB) to save and restore the context of a thread in case of thread switching.
- The only difference is that the information in the TCB is lesser as compared to the PCB. The fields associated with a TCB are

1. Thread ID (TID) : It is a unique identification number of the thread.
2. PC : Indicates the address value at which the next instruction of the thread will be executed by the processor.
3. Registers : CPU registers are used for the execution of a thread. While the thread is in execution, data registers, address registers, control registers, and status registers are used for executing and controlling the process. The information in registers must be saved when there is a change in the state of the threads so that it may resume its execution in its next turn.

TCB

4. State : A thread also has a number of states just like a process. For scheduling the threads, the current state of a thread must be known.

5. Priority : A priority number may be assigned to a thread as provided to a process.

6. Event information : This is the event for which a blocked thread is waiting. If the awaited event is over, the information regarding this event must be stored in this field so that the status of the blocked thread is changed to ready.

7. Information related to scheduling : The information related to scheduling of a thread, such as the waiting time and the execution span of the thread the last time it was running, is also stored.

TCB

8. Pointer to owner Thread : The thread will be created within a thread and it needs to access the execution environment of its owner/ parent thread. Thus, the TCB contains a pointer to this information.

9. TCB pointer : This is a pointer to another TCB that is used to maintain the scheduling list.

UNIT – 3 Usage of Multi-Threading

- **Low Context Switch Overhead**
  - Switching the inforamtion to be ***save and retrieved is less as compared to the process swithcing.***
- **High Computatation Speed up**
  - Due to less information in the TCB and low context switch overhead, there is a **tremendous increase in computation speed.**
- **No Need of IPC**
  - The threads in a multithreaded process do n**ot require system calls** to communicate with each other.

UNIT – 3 Usage of Multi-Threading

- **Decreased Response time**
  - As a consequence of the **increased computation speed,** the response time to a user decrease inspite of the execution of many other tasks.
- **Sharing of Resources**
  - All the **threads share the resources allocated to their owner process.**
- **Efficient Management**
  - It is **easy and economical** in terms of memory and speed to **create and destroy a thread as compared to a process.**

UNIT – 3 When to use Multi Threading instead of Multi Processing

- If the application need to divided into multiple prcesses.
- If process have several tasks inside.
- If application are highly interactive.

UNIT – 3 Types of Threads

- Threads can be of TWO types:
  - User Threads
  - Kernel Threads

UNIT – 3 User Threads

- The user threads are **managed by the application in user space.**
- By default, an **application starts running with a single thread of control,** that is as a process managed by the kernel.
- **User Threads are created and managed** through a **Thread library** provided by the Os.

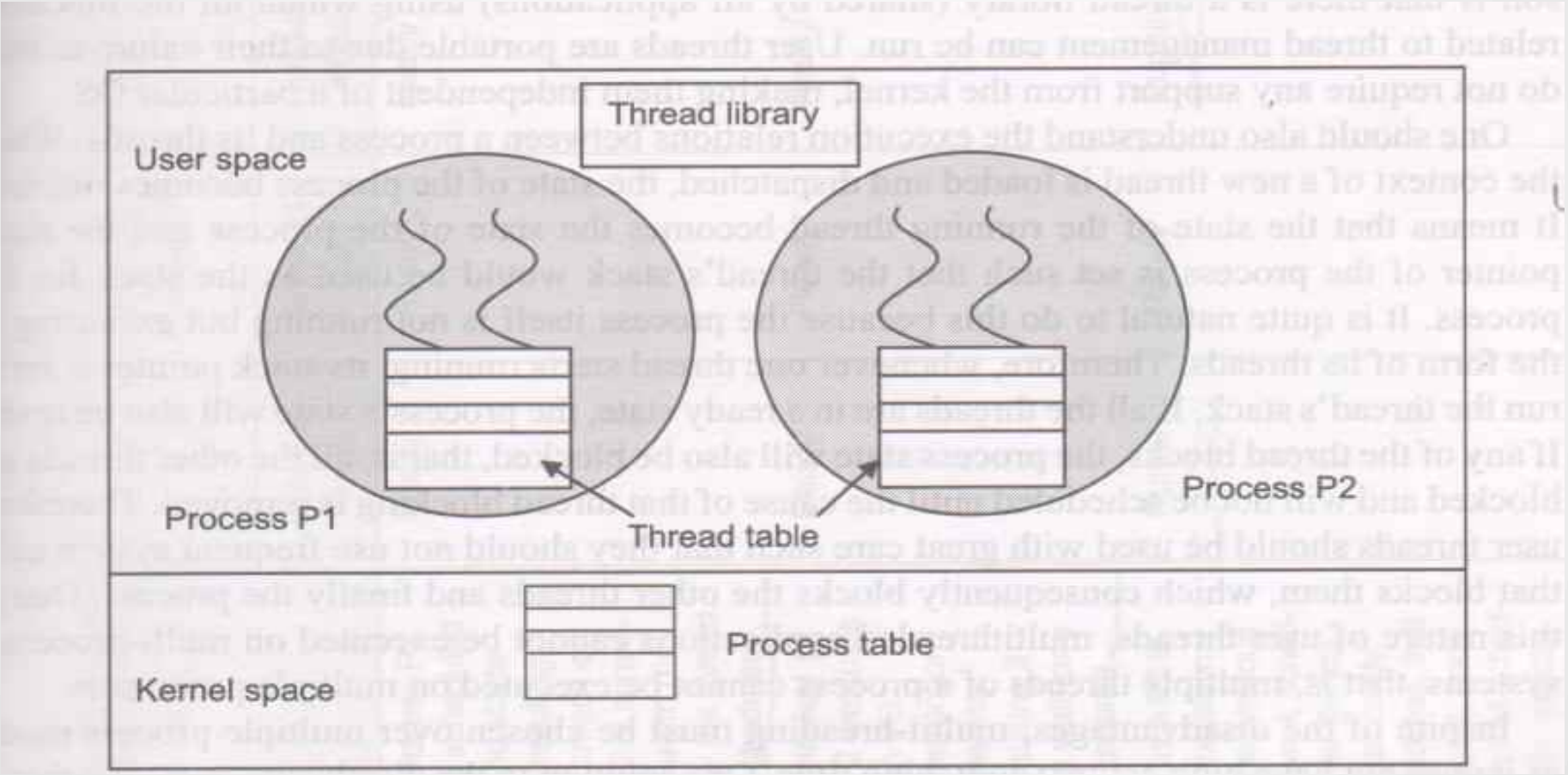
UNIT – 3 User Threads

- Thread **Library / Package** has **following functions:**
  - Thread creation and deletion
  - Assigning priorites to the threads
  - Thread Scheduling
  - Thread Synhronization
  - Communication between the threads
  - Saving and restoring contexts of the threads
  - Blocking and resuming the threads

UNIT – 3 User Threads

- A user thread, is managed through the library functions.
- First of all a **process spawns a thread by calling a procedure.**
- The **thread related data structures** are then **stored in the TCB.**
- The process needs to keep track of all the threads within it. For **that process used Thread table.**
- **Thread table** contains entries as a **pointer to the TCB** of each thread.

UNIT – 3 User Threads

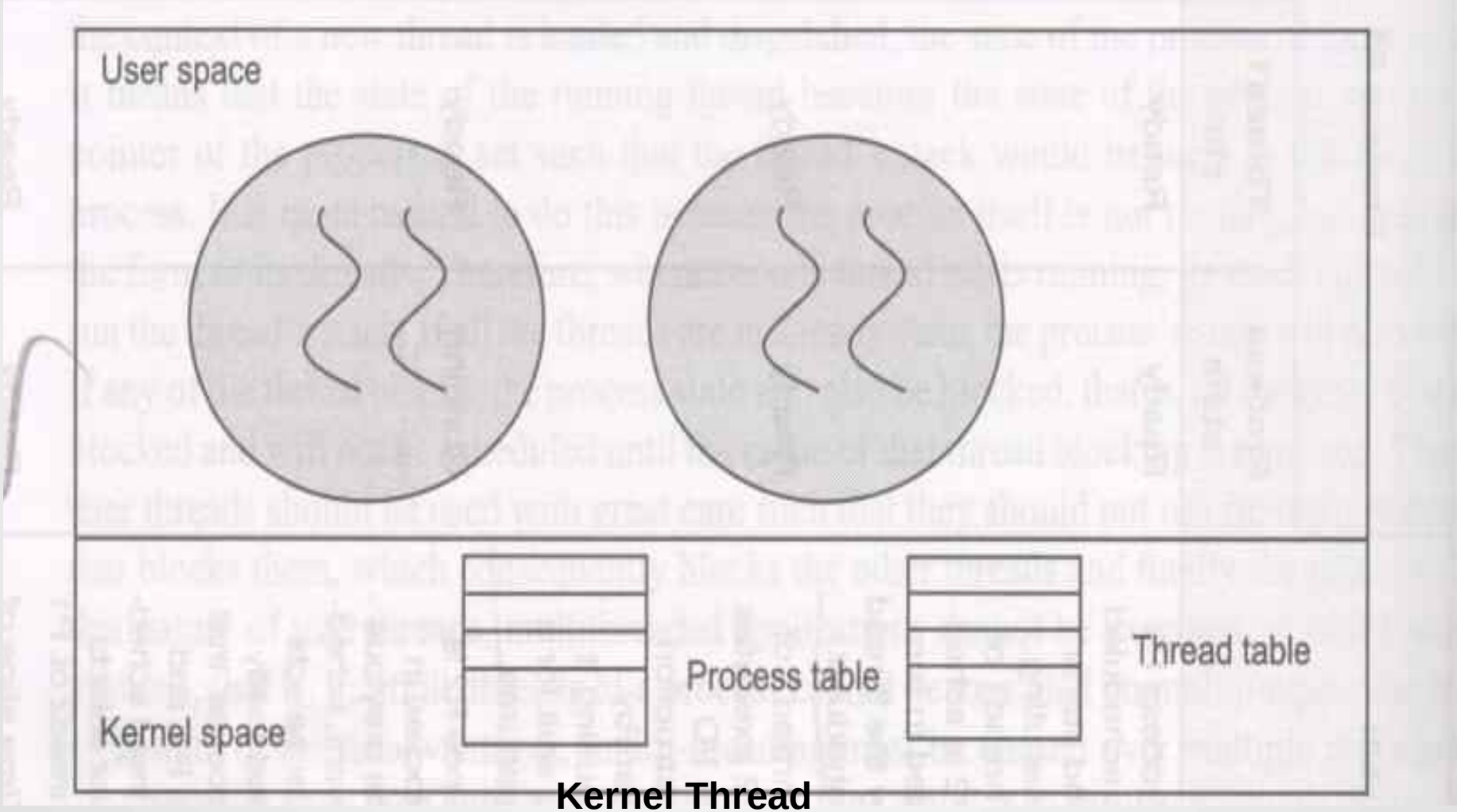


User Thread

UNIT – 3 Kernel Threads

- The kernel threads are **managed by the kernel.**
- Kernel threads, **implemented in the kernel space, are managed through system call.**
- Operation such as **thread creation, delection, context saving, and restoring, synchronization** and so on, are implemented through **system call.**
- There is **no seperated thread table corresponding to each thread** as in user threads.
- **All the thread are managed through a single thread table** maintained in the kernel space.

UNIT – 3 Kernel Threads



Kernel Thread