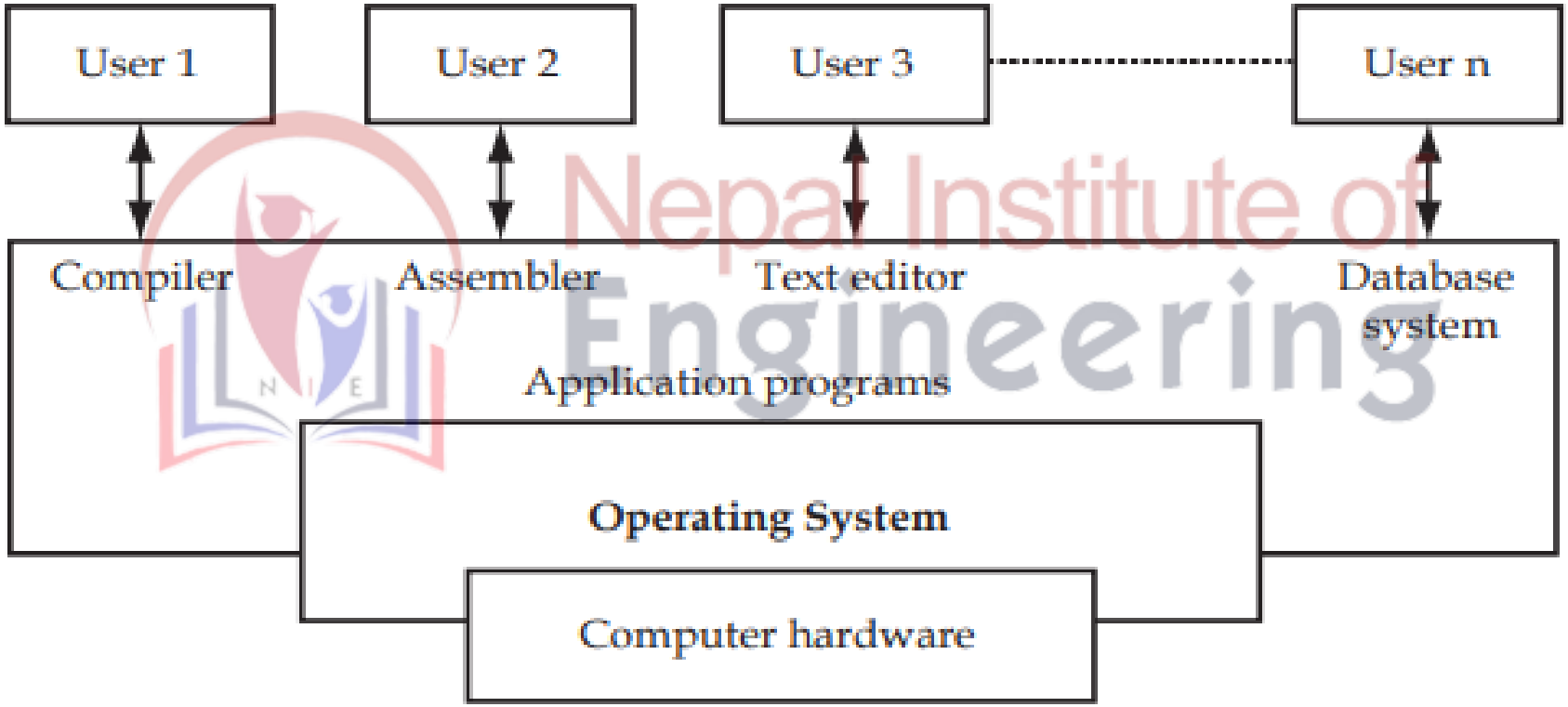
7.5 Introduction to Operating System and process management

- Evolution of Operating System,
- Type of Operating System,
- Operating System Components,
- Operating System Structure,
- Operating System Services,
- Introduction to Process,
- Process description,
- Process states, Process control, Threads,
- Processes and Threads,
- Types of scheduling,
- Principles of Concurrency,
- Critical Region,
- Race Condition,
- Mutual Exclusion,
- Semaphores and Mutex,
- Message Passing,
- Monitors, and Classical Problems of Synchronization

Operating system and functions

- An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs.
- Examples: Windows, Linux, Unix and Mac OS, etc.,
- In simple terms, an operating system is an interface between the computer user and the machine.
- An operating system acts similarly like government means an operating system performs no useful function by itself; though it provides an environment within which other programs can do useful work.

Components of Computer System



Evolution of Operating System

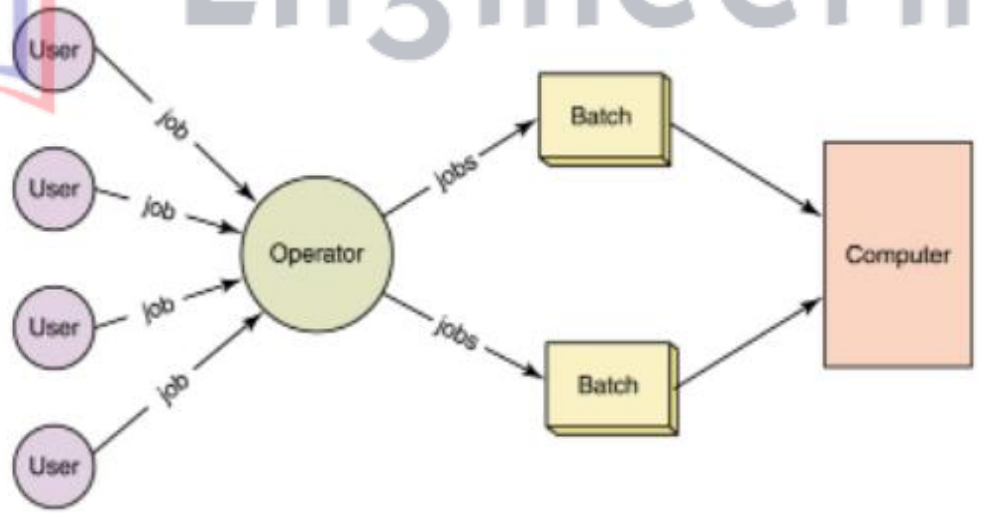
Generations	Years	Electronic devices used	Types of OS and devices
First	1945-55	Vacuum tubes	Plug boards
Second	1955-65	Transistors	Batch systems
Third	1965-80	ICs	Multiprogramming
Fourth	Since 1980	LSI	Personal Computers

Types of Operating System

- **Batch Processing System**
- **Multiprocessing System**
- **Time sharing System**
- **Real time operating system**
- **Network Operating Systems**
- **Distributed Operating System**

Batch Processing System

- In this type of system, there is no direct interaction between user and the computer. It requires the grouping of similar jobs which consists of programs data and system commands.
- **Examples of Batch Processing System: Payroll system, Bank statements**



Multiprocessing System

- A computer's capability to process more than one task simultaneously is called multiprocessing.
- A multiprocessing operating system is capable of running many programs simultaneously, and most modern network operating systems (NOSs) support multiprocessing.
- These operating systems include Windows NT, 2000, XP, and UNIX.
- A multiprocessing system uses more than one processor to process any given workload, increasing the performance of a system's application environment beyond that of a single processor's capability.

Time sharing System

- Time sharing is a technique which allows people sitting at various terminals to access particular computer system at the same time.
- A time sharing operating system is a multi-tasking operating system in which the CPU time is divided equally among all the jobs waiting for the time on the CPU. Normally this time slice is 10 to 100 microseconds
- **Example of Time sharing system:** Time sharing systems are all around us. For example, while we are using our smartphone we can listen songs, search something, view pictures and various notifications appear side by side. All this is enabled by time sharing operating system only.

Real time operating system

- It comprises of two terms: **Real time system** and **operating system**. **Operating system** is simply a program which provides an interface between the hardware and the applications run by the user. **Real-time systems** are those systems which are defined not just by the accurate computation of result but also by the specified time constraints and deadlines within which the processing ought to be completed.
- It is an operating system that supports real-time applications by providing the correct result and, that too, within the deadline required. It is just like a general OS but it provides some additional mechanisms for the real-time scheduling of tasks.
- Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behavior.

Hard real-time operating systems

- These operating systems are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable.
- These are mainly present in the life saving systems like automatic parachutes or air bags which are required to be readily available in case of any accident.

Soft real-time operating systems

- These kinds of operating systems serve the real-time applications in which the time constraints are somewhat less strict and a little amount of delay is acceptable.
- For example, Live TV broadcast, Video Conferencing etc. Even if some delay is there, nothing catastrophic is going to happen.

Network Operating Systems

- These systems run on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions.
- These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a small private network.
- Mainly there are two types of network operating.
  - **Peer-to-peer network operating systems**
  - **Client/server network operating systems**

Peer-to-peer network operating systems

- It allow users to share resources and files located on their computers and to access shared resources found on other computers.
- In a peer-to-peer network, all computers are considered equal; they all have the same privileges to use the resources available on the network.
- Peer-to-peer networks are designed primarily for small to medium local area networks. Windows for Workgroups is an example of the program that can function as peer-to-peer network operating systems.

Client/server network operating systems

- It allow the network to centralize functions and applications in one or more dedicated file servers.
- The file servers become the heart of the system, providing access to resources and providing security. The workstations (clients) have access to the resources available on the file servers.
- The network operating system allows multiple users to simultaneously share the same resources irrespective of physical location.

Distributed Operating System

- A distributed operating system is an operating system that runs on several machines. Its purpose is to provide a useful set of services, generally to make the collection of machines behave more like a single machine.
- Various interconnected computers communicate with each other using a shared communication network.
- Distributed operating systems typically run cooperatively on all machines whose resources they control. These machines might be capable of independent operation, or they might be usable merely as resources in the distributed system.
- **Examples of Distributed operating systems:** Telephone and cellular networks, Internet, etc.

Operating System Components

- An operating system is a large and complex system created by combining small pieces. These pieces are well-defined parts of a system that have carefully defined inputs, outputs, and functions. Though most OSs differ in structure, most of them have similar components.
- However, many modern operating systems share the system components outlined below.
  - Process management
  - I/O management
  - Main Memory management
  - File & Storage Management
  - Protection
  - Networking
  - Protection
  - Command Interpreter

Process

- A process is defined as an entity which represents the basic unit of work to be implemented in the system i.e. a process is a program in execution.
- The execution of a process must progress in a sequential fashion.

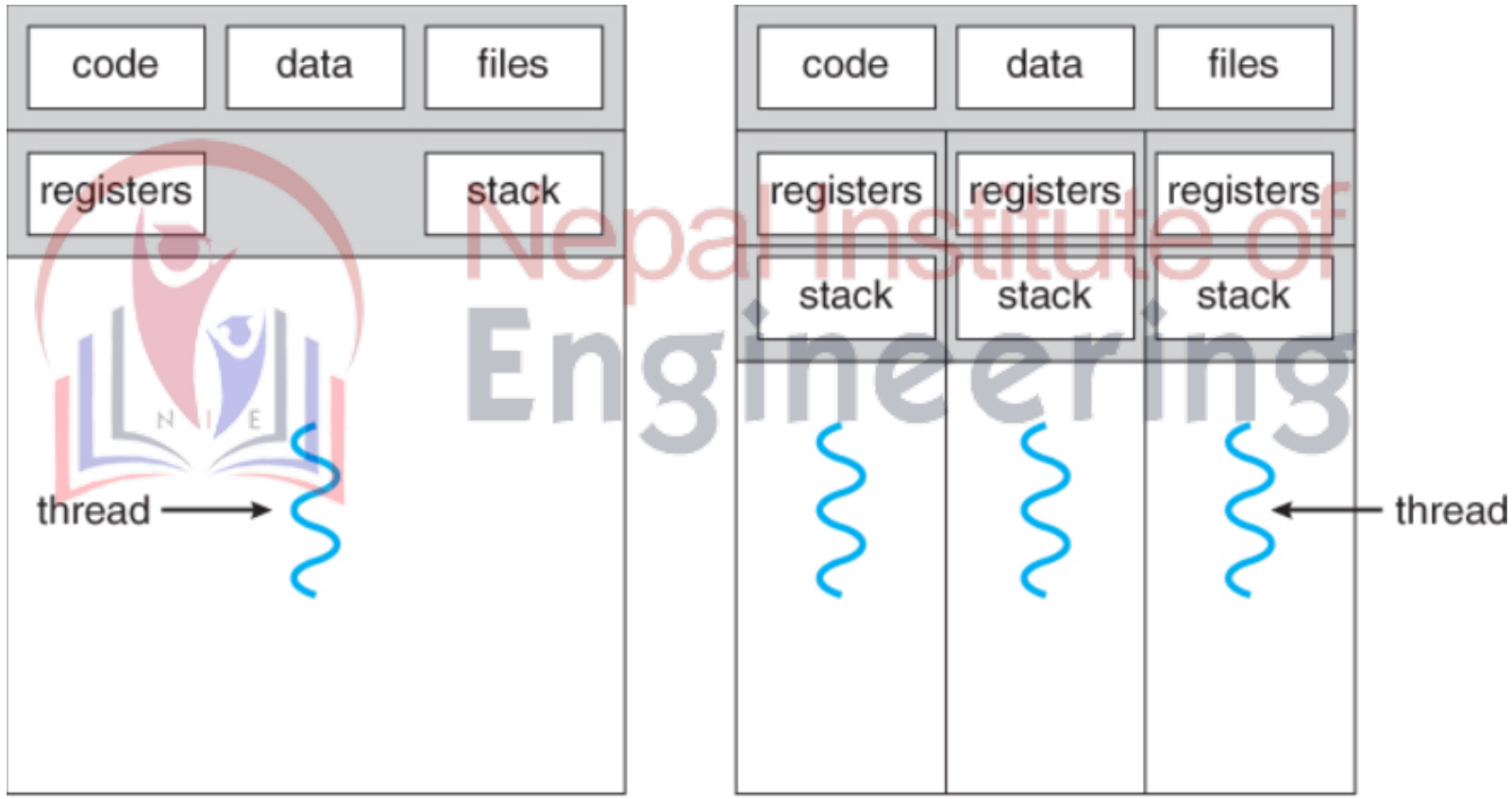
Program VS process

Program	Process
1. It is a set of instructions.	1. It is a program in execution.
2. It is a passive entity.	2. It is an active entity.
3. It has longer lifespan.	3. It has limited lifespan.
4. A program needs memory space on disk to store all instructions.	4. A process contains many resources like a memory address, disk, printer, etc.
5. Program is loaded into secondary storage device.	5. Process is loaded into main memory.
6. It is a static object existing in a file form.	6. It is a dynamic object (i.e., program in execution)
7. No such duplication is needed.	7. New processes require duplication of parent process.

Threads

- Thread is a single sequential flow of execution of tasks of a process. It is also known as thread of execution or thread of control.
- It is the fundamental unit of CPU utilization consisting of a program counter, stack and set of registers.
- Threads have some of the properties of processes, they are sometimes called lightweight processes.
- Each thread belongs to exactly one process and outside a process no threads exist.
- The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.
- Like a traditional process i.e., process with one thread, a thread can be in any of several states (Running, Blocked, Ready or terminated).

Threads



single-threaded process

multithreaded process

Similarities between process and thread

- Like processes, threads share CPU and only one thread active (running) at a time.
- Like processes, threads within processes execute sequentially.
- Like processes, thread can create children.
- Like process, if one thread is blocked, another thread can run

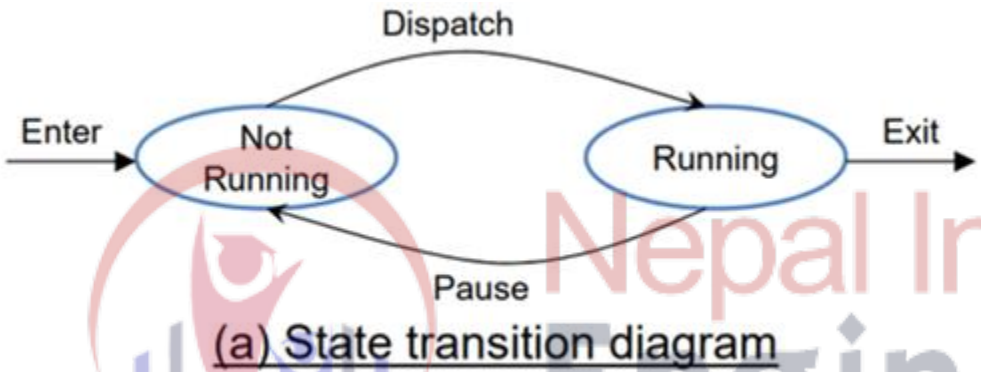
Difference between Process and thread

Process	Thread
1. heavy weight	1. light weight
2. It takes more time to terminate.	2. It takes less time to terminate.
3. More resources are required.	3. Less resources are required.
4. More time required for creation.	4. Less time required for creation.
5. System call is involved in process.	5. System call not involved in threads.
6. If one server process is blocked then other server processes cannot execute until the first process is unblocked.	6. If one thread is blocked and waiting then the second thread in the same task can run.
7. Slow Execution	7. Fast Execution
8. Doesn't share memory (loosely coupled)	8. Shares memories and files (tightly coupled)
9. Takes more time to switch between processes.	9. Takes less time to switch between threads.

Race Condition

- A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.
- When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions.

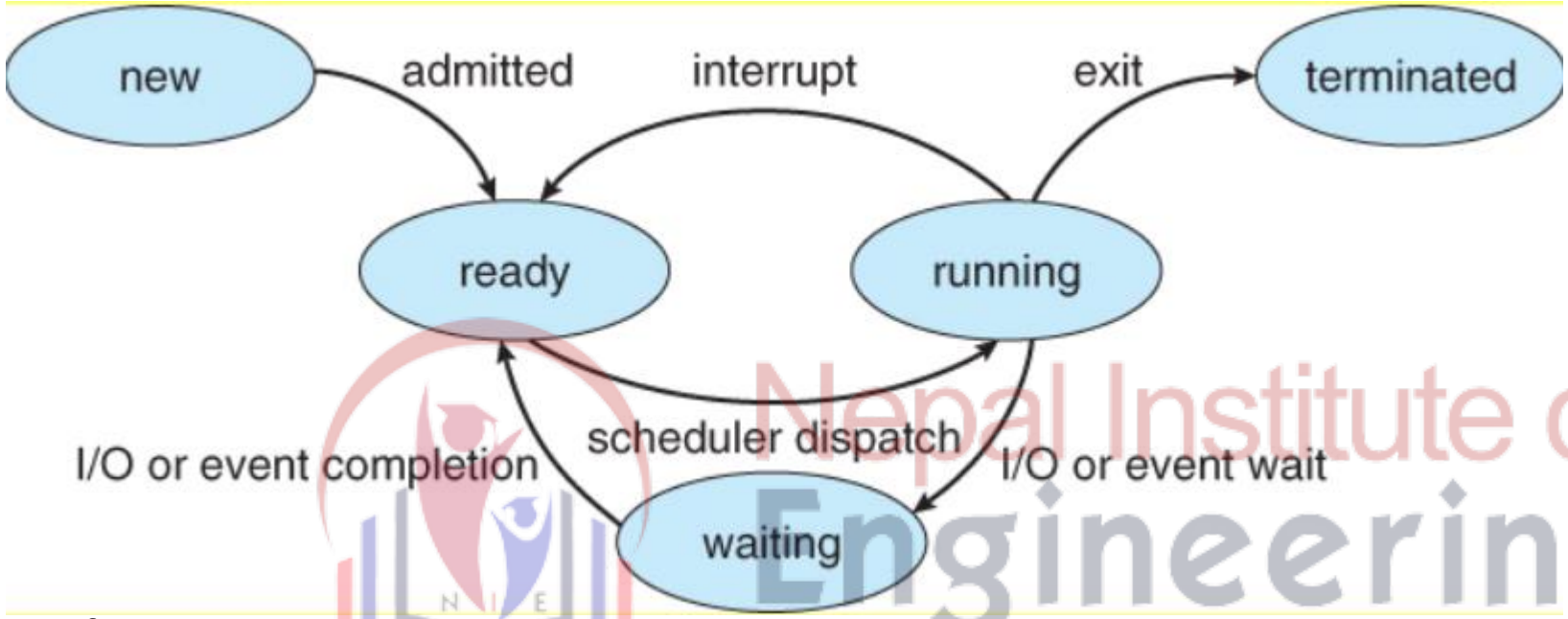
Process State (Two state model)



(a) State transition diagram

- Two State Process Model consists of two states:
- **Not-running State:** Process waiting for execution.
- **Running State:** Process currently executing.

Process State (Five State Model)



The five states in this model are:

- **New**: A newly created process that has not yet been admitted to the pool of executable processes by the OS.
- **Ready**: Processes that prepared to execute when given opportunity.
- **Running**: The process is currently being executed.
- **Waiting**: The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
- **Terminated:** Process has been completed.

Race Condition

Example:

- Consider two cooperating processes P1 and P2 that updates the balance of account in a bank.

Process P1	Process P2
Read Balance	Read Balance
Balance= Balance + 1000	Balance= Balance -400

- Suppose that the balance is initially 5000, then after the execution of both P1 and P2, it should be 5600. The correct result is achieved if P1 and P2 execute one by one in any order either P1 is followed by P2 or P2 followed by P1.
- However if the instructions of P1 and P2 are interleaved arbitrarily, the balance may not be 5600 after the execution of both P1 and P2. One possible interleaving sequence for the executions of instructions of P1 and P2 is given in below table.

Race Condition

Process P1	Process P2	Balance
Read balance		5000
	Read balance	5000
Balance= Balance + 1000		6000
	Balance = Balance -400	4600

- The above interleaved sequence results in an inconsistent balance that is 4600. If the order of the last two instructions is interchanged, the balance would be 6000 (again, inconsistent).

Note that race condition is a term for a situation where several processes sharing some data execute concurrently, and where the result of execution depends on the order in which the shared data is accessed by the processes.

Mutual Exclusion

- To avoid race conditions, we need to ensure mutual exclusion. That means if a process P1 is manipulating shared data, no other cooperating process should be allowed to manipulate it until P1 finishes with it. In the above example, since mutual exclusion was not ensured while accessing the shared data, it yielded an inconsistent result.
- Mutual exclusion is the way of making sure that if one process is executing critical section, other processes will be excluded from executing critical section.
- It is the responsibility of operating system to assure that no more than one process is in its critical section simultaneously.

Critical Section Problem

- In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed need to be protected in ways that avoid the concurrent access. This protected section is the critical section or critical region. It cannot be executed by more than one process at a time.
- **Four conditions to avoid critical region problem:**

1. No two processes may be simultaneously inside their critical regions.
2. No assumptions may be made about speeds or the number of CPUs.
3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region.

Critical Section Problem

Example:

Here process A enters its critical region at time T1. A little later, at time T2 process B attempts to enter its critical region but fails because another process is already in its critical region and we allow only one at a time. Consequently, B is temporarily suspended until time T3 when A leaves its critical region, allowing B to enter immediately. Eventually B leaves (at T4) and we are back to the original situation with no processes in their critical regions.

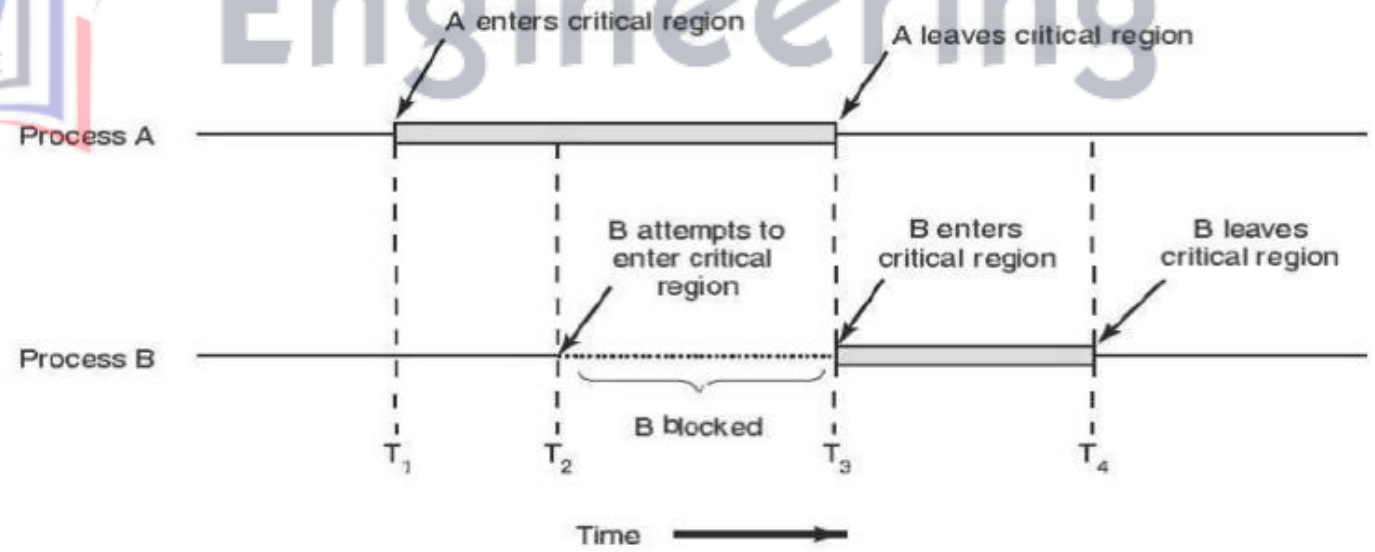


Figure 1. Mutual exclusion using critical regions.

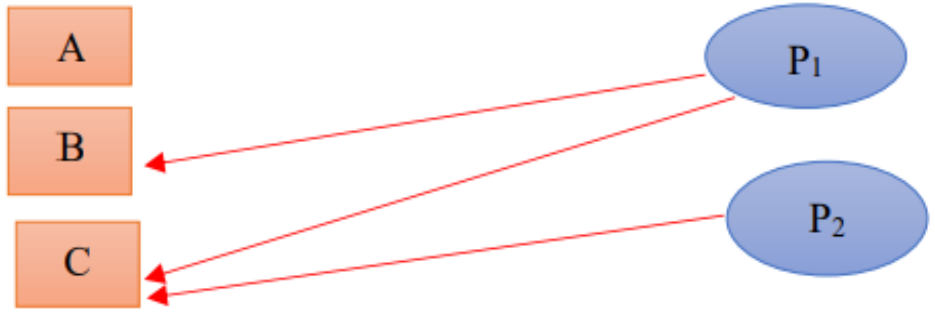
Semaphores

- Semaphore is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multiprocessing environment.
- It was proposed by Edsger Dijkstra. It is a technique to manage concurrent processes by using a simple integer value known as semaphore.
- The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1 only.
- A semaphore can only be accessed using the following operations:
- **wait ()** :- called when a process wants access to a resource
- **signal ():-** called when a process is done using a resource

Mutex

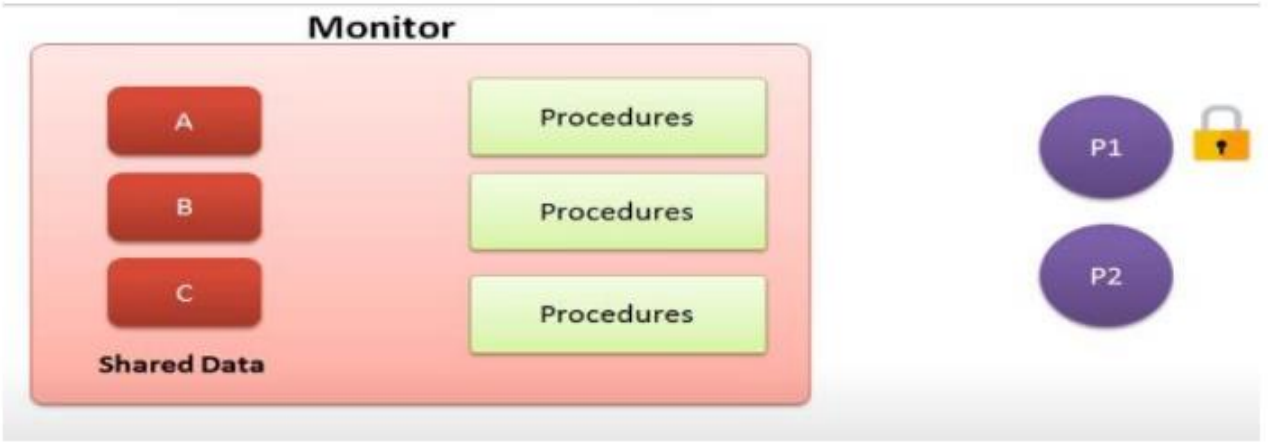
- When the semaphore's ability to count is not needed, a simplified version of the semaphore, called a mutex, is sometimes used.
- Mutexes are good only for managing mutual exclusion to some shared resource or piece of code. They are easy and efficient to implement, which makes them especially useful in thread packages that are implemented entirely in user space.
- A mutex is a variable that can be in one of two states: unlocked or locked.
- Consequently, only 1 bit is required to represent it, but in practice an integer often is used, with 0 meaning unlocked and all other values meaning locked.

Monitor



- Monitor is programming language construct that control access to shared data.
- It is a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package.
- Monitor is a module that encapsulate:
  - Shared data structures.
    - Procedures that operates on shared data structure and
  - Synchronization between concurrent procedure invocations.
- In Monitor when the process wants to access shared data, they cannot access directly. Process have to call procedure and those procedure in turn will allow access to shared data.
- Only one process can enter into monitor at a time.

Monitor



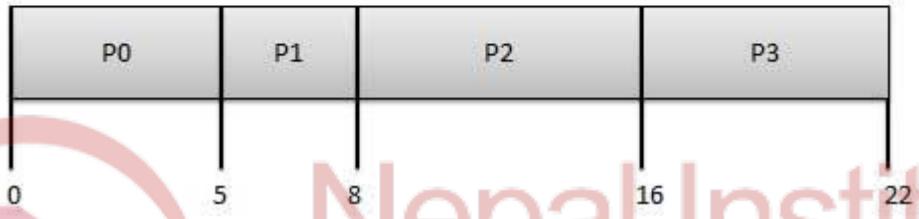
Example:

- Here when process P1 calls the procedure in order to access shared data, at first it is checked if any other process is in monitor or not.
- If no other process is in the monitor then P1 acquires lock and enters the monitor.
- When process P2 calls the procedure, then it gets block and will be placed in the queue of monitor.
- In monitor Conditional variable provides the synchronization between the processes.
- Three operations can be performed on conditional variables:
- **wait ():** When a process call wait () operation, that process is placed on a queue to enter to the monitor.
- **signal ():** When a process wants to exit from monitor it calls signal () operation.
- When a process calls a signal () operation, it causes one of the waiting processes in the queue to enter monitor.
- **broadcast ():** It signals all the waiting processes in the queue to wait.

Classical Problems of Synchronization

- **Producer Consumer Problem**
- **Dining Philosophers Problem**
- **Reader writer problems**
- **Sleeping Barber Problem**

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Waiting time

P0= tat-bt=0-0=0, p1=5-1=4, p2=8-2=6 and p3=16-3=13

avg wt=(0+4+6+13)/4=5.75

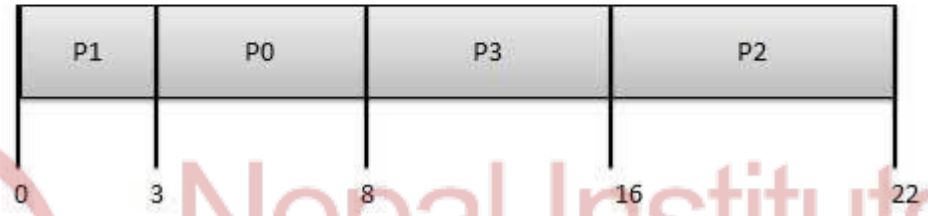
turn around time

p0=ct-at=5-0=5, p1=8-1=7 p2=16-2=14 and p3=22-3=19

avg tat=(5+7+14+19)/4=11.25

asda

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Gantt chart

P0(0-5)->P1(5-8)->P3(8-14)->P2(14-22)

TAT

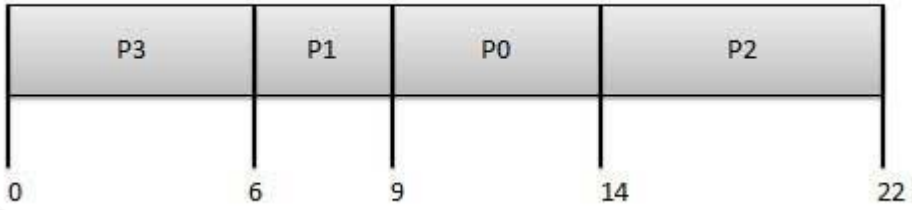
P0=ct-at=5, p1=8-1=7, p2=22-2=20, p3=14-3=11=> avgTAT=43/4=10.75

WT

P0=tat-bt=5-5=0, p1=7-3=4, p2=20-8=12, p3=11-6=5=> avg wt=21/4=5.25

asd

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



P0(0-5)p3(5-11)p1(11-14)p2(14-22)

Tat=ct-at

P0=5, p1=14-1=13, p2=22-2=20, p3=11-3=8=>avg tat=46/4=11.5

Wt=tat-bt

P0=5-5=0, p1=13-3=10, p2=20-8=12, p3=8-6=2=>avg wt=24/4=6