# DIPLOMA WALLAH

(Your One Stop Hub For Diploma Resources)

# OPERATING SYSTEM AND ADMINISTRATION

Complete Notes Based on Full Syllabus

- Diploma Engineering

4th Semester



www.diplomawallah.in

Notes prepared by **Sangam**

Diploma watch

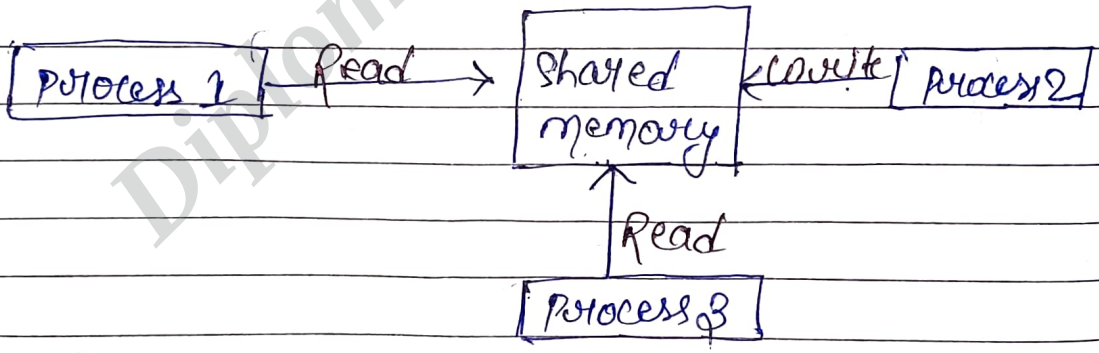
Sangam

Unit-05

# Process Synchronization

Process Synchronization is used in a Computer System to ensure that multiple processes or threads can run concurrently without interfering with each other.

The main purpose or objective of process Synchronization is to ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access.




Critical Section

* A critical section is the part of code where a shared resources is accessed.
problems: Ensure that only one process enters the critical section at a time.

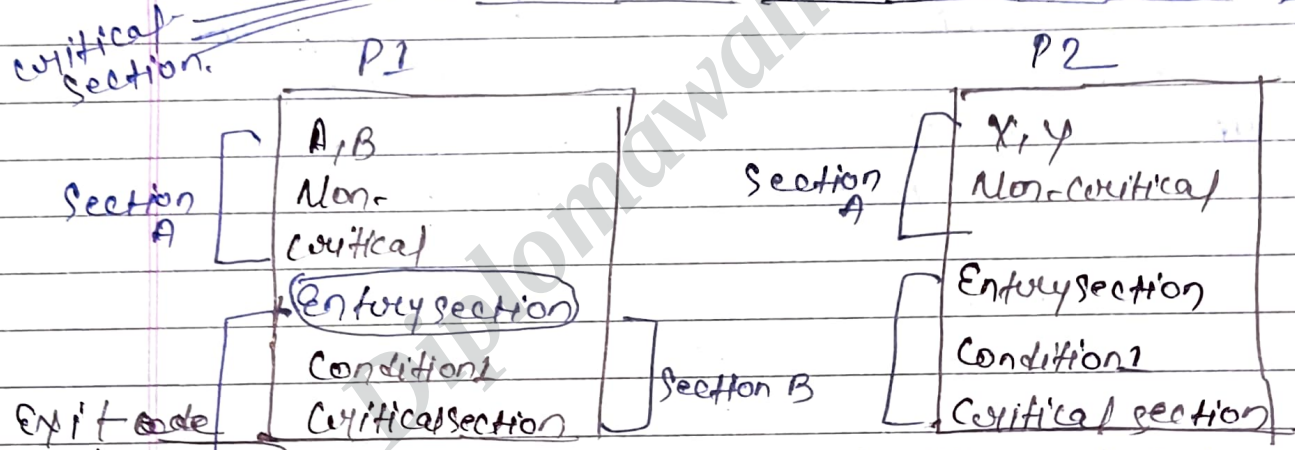
* Condition/Rules to solve critical section problem:-

when we editing the same file in two interface it will corrupt.

Primary rules (must be follow)
1. Mutual Exclusion :- Only one process in critical section at a time.
2. progress :- If no process is in critical section, one should be allowed to enter.

Secondary rule
3. Bounded waiting :- No process should wait forever.
4.) No assumption related to the speed. (Hardware are not give any special favour all are equal).

* Semaphores
+ It is a integer variables that are used to solve the critical section problem.




These entry section in both P1 and P2 have different code which help to execute 1 at a time.

* A Semaphore is a synchronization tool used in OS to control access to shared resources among multiple processes or threads.
SRTF It is basically an integer variable (S) that help in avoiding race conditions.
Types :- (i) Binary Semaphore
(ii) Counting Semaphore

Counting Semaphore – $-\infty$ to $\infty$
Binary Semaphore – 0, 1

0 → busy
1 → available

## * Semaphores

Semaphores is a method or tool which is used to prevent race condition.

Semaphore is an Integer variable which is used in mutual exclusive manner by various concurrent & cooperative process in order to achieve Synchronization.

Operation perform in Semaphores.

1) P(), down, wait → Entry section.
→ Decrease value, blocks if $<0$.

* V(), up, Signal(S) → Increase value, wakes up waiting process.

## * Counting Semaphore

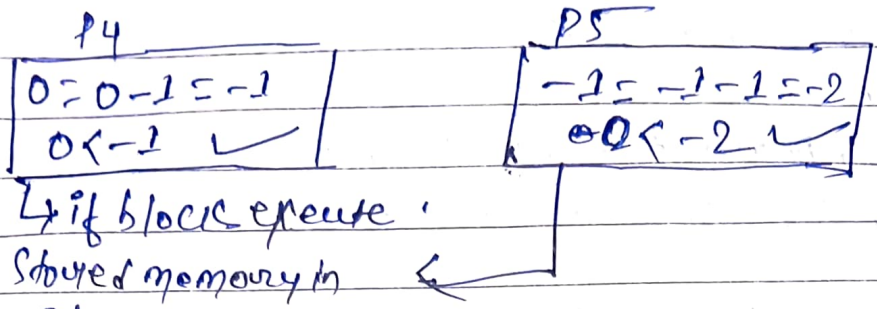
1) Entry Section.

```
down (Semaphore S)
  S.value = S.value - 1
  if (S.value < 0)
  {
     put process (PCB) in suspended
  list, sleep()
  }
  else
     return;
}
```

Ex. S = 3

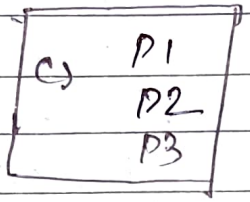
| P1 | P2 | P3 |
|---|---|---|
| 3 = 3-1 = 2 | 2 = 2-1 = 1 | 1 = 1-1 = 0 |
| 2 < 0 X | 1 < 0 X | 0 < 0 X |

↳ stored in the Critical section.




↳ if block execute. Stored memory in Sleep mode.

P4, P5 - Suspended block.




ii) Exist Section

```
up(Semaphoure S)
{
  S.value = S.value + 1;
  if (S.Value <= 0)
  {
    Select a process from
    Suspended list
    wake up();
  }
}
```

In this code we use `S.value + 1` which will up the value means the value which is in critical section will be outside.

* If Semaphoure value is 0 then there is no any suspended list operation perform.

# Binary Semaphoure.

Operation

Down, P, wait - for entry

up, V, Signal - for exit.

```
down(semaphoure S)
{
    if (S.value == 1)
    {
        S.value = 0;
    }
    else
    {
        Block this process and place in
        suspend list, sleep();
    }
}

up(semaphoure S)
{
    if (suspend list is empty)
    {
        S.value = 1;
    }
    else
    {
        select a process from suspend
        list and wake up()
    }
}
```

P1

// if S = 1 then convert it into 0.

P2 will block because it neither be 1 or 0.

// if S = 0 then convert it in S = 1 process

// Same here all other block or suspend.

* Deadlock:
If two or more process are waiting on happening of some event, which never happens, then we say these process are involved in deadlock. Than that state is called deadlock.

* Necessary Conditions (Coffaman's Conditions):

1. Mutual Exclusion.
- A resource can be used by only one process at a time.
- Ex- A printer cannot shared by two process at the exact same amount.

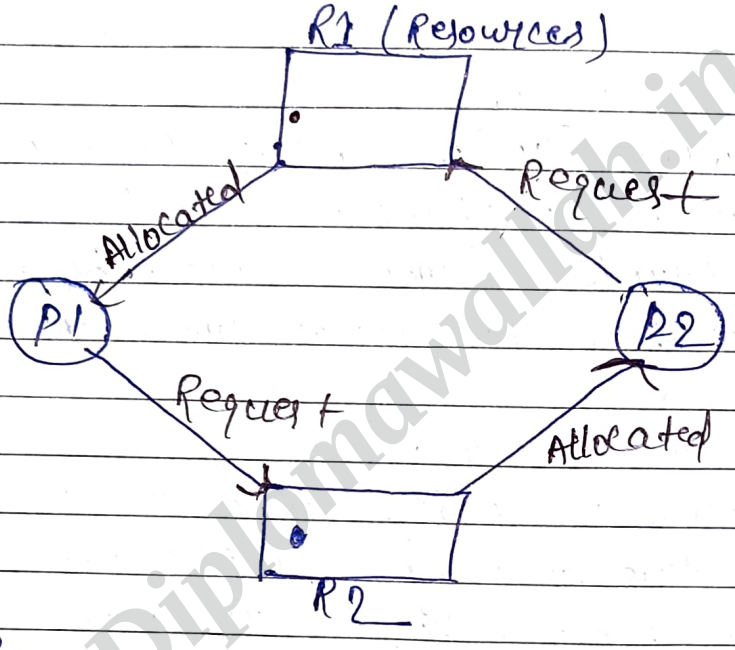
2. Hold and wait
A process is holding atleast one resource and waiting to acquire additional resources.
Ex- process A is holding CPU but waiting for I/O device.

3. No preemption
- A process cannot be forcibly taken away from a process, it must be released voluntarily.
Ex- We can't snatch the printer from a process until it finishes printing.

4. Circular wait
A circular chain of process exist, where each process is waiting for a resource held by the next.

Ex- P1 → waiting for resource held by P2, P.
P2 → waiting for P3, Pn → waiting for P1.

→ If these all four condition true deadlock will happen.
If one condition is broken, deadlock cannot happen.




Resource 1 allocate P1 and P1 wants R2, but Resource 2 allocate P2 and P2 want R1.

* Methods for handling Deadlocks

1. Prevention:-
- Ensure that at least one of the four necessary conditions for deadlocks (mutual exclusion, Hold and wait, No preemption, Circular wait) never occurs.

Eg -
- Do not allow "Hold and wait" → a process must request all resource at once.
- Break "circular wait" → assign a strict ordering of resources (e.g. CPU first, then I/O).

2. Deadlock Avoidance :-
OS check system and allocates resources safely so deadlock never occurs.

- The most famous algorithm = Banker's Algorithm (like a bank does not given loan if it may lead to bankruptcy.)
- Require advance knowledge of maximum resources a process may need.

Banker Algorithm :-
The Banker's Algorithm is based on the concept of resource allocation graph. (visual way to understand how resources are assigned in OS.) A resource allocation graph is a directed graph where each node represents a process, and each edge represent a resource.

RAG → Resource Allocation graph
WFG → wait for graph

3. Deadlock detection
Deadlock detection is used by employing an algorithm that detect circular waiting and kills one or more processes so that the deadlock is removed.
- This technique does not limit resource access or restrict process action.
- Requested resources are granted to processes whenever possible.
- It never delays the process initiation and facilates online handling.

4. Deadlock Ignorance:
In the deadlock Ignorance method the OS acts like the deadlock never occurs and completely ignore it even if the deadlock occurs.
- Killing processes involved in deadlock.
- Preempting resources (take back pointer, memory etc from some process.)
- Rollback (returns a process to some previous safe state and restarts).

Ex
If a deadlock is detected in a database system, OS may terminate one transaction (process) and roll it back, releasing resources for others.

# * Thread

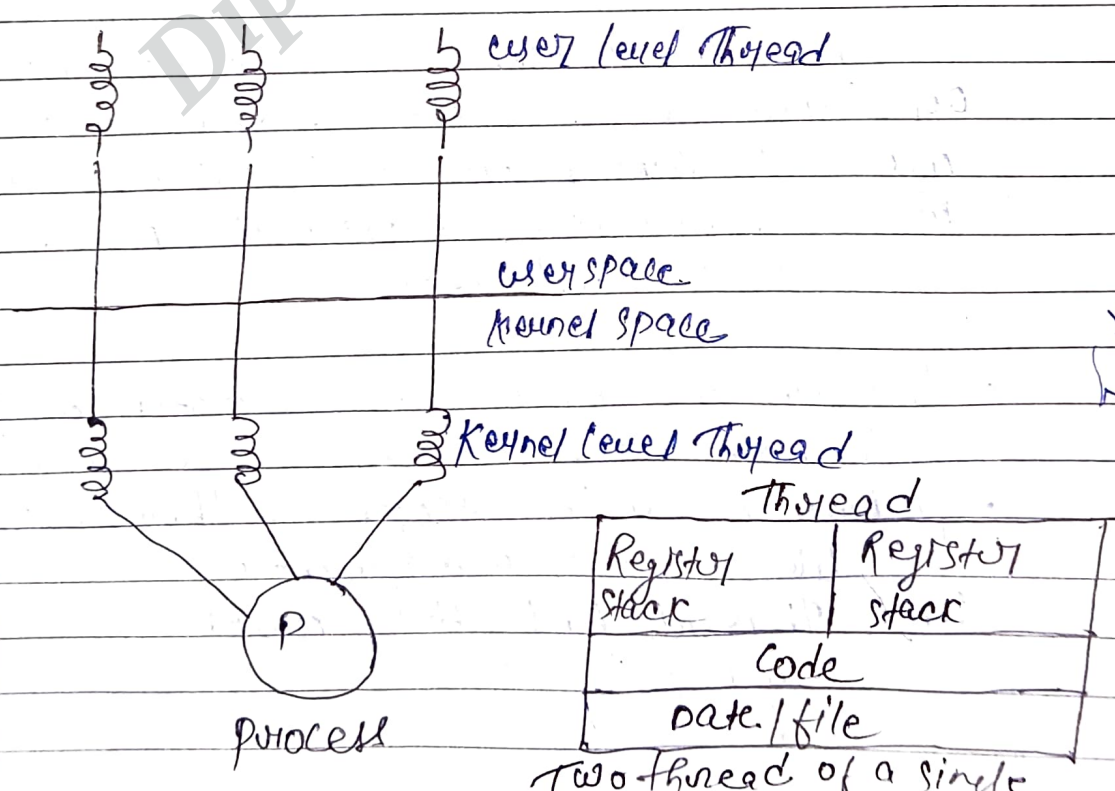
→ A thread is the smallest unit of execution within a process.

→ A process can have one or more threads.

→ Threads inside the same process share:
- Code (some program instruction)
- Data (global variable, heap memory).
- Resources (like files, memory space)

→ But each thread has its own:
- program counter — keeps instruction which execute next.
- Stack — local variable function call like.
- Registry — temporary storage

Ex — A web browser process has:
- one thread to render the page.
- Another thread to download files.
- Another thread to play music.




Two thread of a single process.

- Each Thread belongs to exactly one process.
- In an OS that supports multithreading, the process that can consist many threads. But thread can be effectively only if the CPU is more than 1 otherwise two threads have to context switch for the single CPU.
- All threads belonging to the same process share - code section, data section, and OS resources (eg - open files and signals).
- Each Thread has their own thread ID, program counter, register set and a stack.

Ex- When we work on microsoft word or Google docs we notice that while we are typing, multiple things happen together, formatting is applied, page is changed and auto save happen.

- Threads can share common data so they do not use inter-process communication.
- priority can be assign to thread just like process management.
- Each Thread had their own Thread control Block (TCB) → store the important information of thread.

Thread Identifier, Program Counter, Registers, Thread Stack, Stack pointer, priority, Pointer to process, Resource info (if needed).

## Types of Thread.

1. User Level Thread.
- It is a type of Thread that is not created using system calls.
- The kernel has no work in management of user level threads.
  User level thread can be easily implemented by a user.

In case when user-level threads are single handed processes, Kernel level thread manages them.

Advantage
→ Easier than Kernel-level thread.
→ Context Switch Time is less in user level.
→ It is more efficient than Kernel-level.

Disadvantage
→ The OS is unaware of user-level threads, so Kernel-level optimization like load balancing across CPUs are not utilized.
→ If a user-level thread makes a blocking system call, entire process (and all its thread) is blocked.

OS thread schedule them individually.

2. Kernel level Thread

→ Kernel level is a type of thread that can recognize the OS.
→ It has its own thread table where it keep track of the system.
→ The OS Kernel help in managing threads.
→ It has longer context switching time.
→ Each thread treated like a schedulable unit.
→ Kernel helps in management of threads.
→ Each thread has their own TCB inside thread.

Advantage

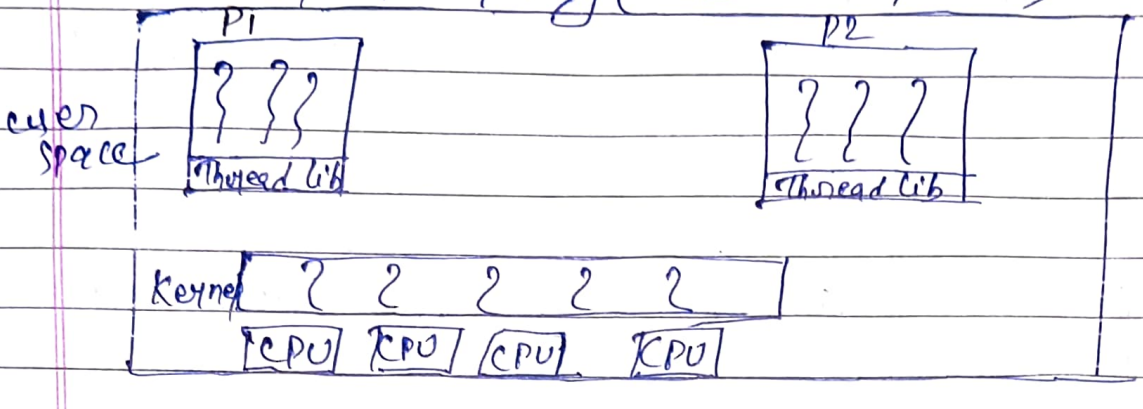
→ It can run on multiple processors or cores simultaneously.
→ Kernel is aware of all threads, allowing it to manage and schedule them effectively across available resources.
→ Application that block frequency are to be handled by the kernel-level threads.

Disadvantage

→ Context switching b/w kernel level threads is slower compared to user-level threads because it requires mode switching b/w user and kernel space.
→ managing Kernel level threads involves frequent system calls and kernel interactions, leading to increased CPU overhead.
→ Large number of threads may overload the Kernel scheduler.
→ It is little more complex than user-level.

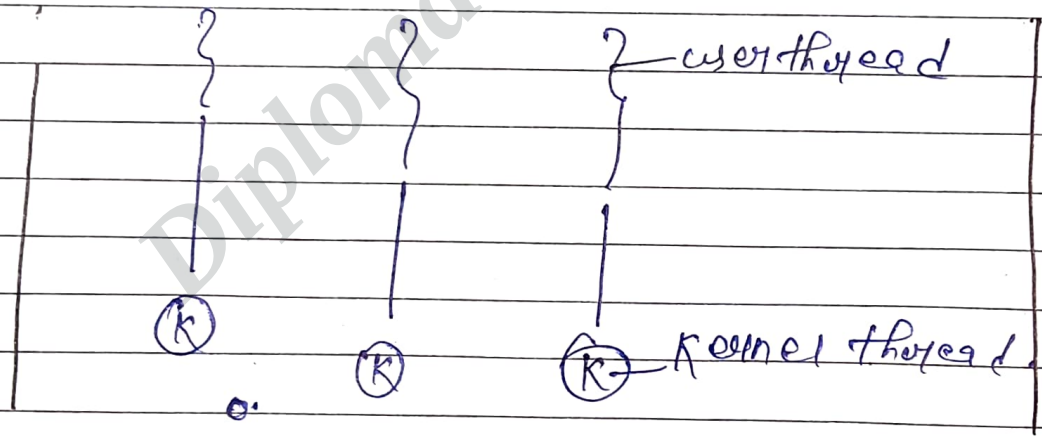
# Models of thread

## multithreading (many to one)




- Many user level thread map to one kernel level thread.
- If one thread blocks then all another will blocks automatically.

## One to one multithreading.



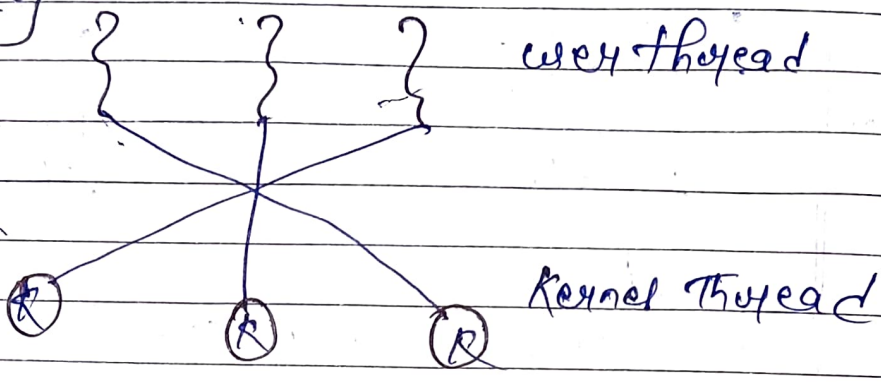

- Each user thread map with one kernel thread.
- Two parallism on multicore.
- Creating many kernel thread is expensive.

moreover overhead.

Sagar Sangam

Date
Page

many to many




- Six user level thread are connect with 6 Kernel level thread.
- This kernel level provide best accuracy and Concurrency.

Diploma wallah
Sharing/selling not allowed.