2,a 3,a
3,b 4,a
3,c. 4,b
3,b

Motivation for Concurrency in software

Here are the Key motivations for embracing Concurrency.

① Performance Boost:
Multicore processors enables simultaneous execution of tasks, significantly enhancing Computational throughput and application speed.

② Scalability:
Adding more Cores allows for horizontal scaling, meeting increasing Computational demands more effectively than traditional single Core systems.

③ Resource Efficiency:
Concurrent execution optimizes hardware resource utilization like CPU cycle and memory bandwidth, improving overall efficiency.

④ Real time processing:
Enables handling of multiple I/P Concurrently, crucial for tasks such as multimedia Processing.

⑤ Energy Efficiency:
Multicore designs can potentially reduce power Consumption per task through dynamic power management techniques.

Parallel Computing Platform.

It refers to a hardware & software infrastructure designed to support the simultaneous execution of tasks across multiple processing units.

multiple instruction single data — MIMD
SISD — SIMD

Components

① Hardware Components :-

- Multicore processors :- CPU with multiple core on a single chip, allowing concurrent execution of tasks
- Accelerators :- specialized hardware like Graphics processing units (GPU) optimized for Parallel Computation
- Interconnects :- High speed Communication channels (buses, switches, network) that facilitate data exchange b/w processing units.

② Software Components :-

- Parallel Programming Model :- Abstractions & frameworks (MPI, openMP) that enable developers to express and manage parallel tasks effectively.
- OS support :- optimizations that enhance performance & resource utilization in parallel environments
- Middleware :- software layers providing services such as job scheduling, load balancing & fault tolerance to manage distributed Computation

Characteristics & benefits

- Performance Scalability :- Distributes tasks across cores or processors to handle larger workload efficiently.
- Improved Throughput :- overlaps computations to increase overall processing speed & reduce idle time
- Resource efficiency :-
- Support real time processing :-
- Fault tolerance :- Redundancy & error handling mechanism

Applications

① Scientific simulations:- for complex computations in physics chemistry & engineering.
② Big data Analytics:- Accelerates data processing for insights in finance, healthcare & marketing.
③ AI/ML:- facilitates training and inference tasks in deep learning models.
④ HPC (High Performance Computing):- Paves researches in weather forecasting & molecular dynamics.

Parallel Computing in microprocessors

It refers to the capability of modern processors to execute multiple tasks or instructions simultaneously thereby increasing overall computational throughput and performance.

- Multicore Architecture:
  Core operate independently, allowing concurrent execution of tasks.
- Simultaneous Multithreading (SMT):
  specialized units perform operations on multiple data elements simultaneously, beneficial for tasks like multimedia processing.

Applications:

① Real time system:- used in automotive system, robotics & interactive application.
② HPC:- support scientific simulations, numerical computations.

(5) Data Analytics & AI :- Accelerates data Processing & ML task through Parallel execution of algo.

(6) Server Application :- enhance Performance & scalability in web servers, db & Cloud Computing.

feature	Multicore Architecture	Hyper-Threading Technology
Definition	• Multiple Physical core on a single chip	• Intel's simultaneous multithreading technology.
Core Nature	• independent Physical core	• Single Physical core appears as multiple logical Processors.
Execution model	• Executes multiple thread simultaneously	• share resources b/w multiple threads on a single core
Scalability	• Scales linearly with additional cores	• improves utilization of existing core resources without additional cores.
Performance	• Provide significant Performance gains with each added Core	• increases core utilization, enhancing Performance but with diminishing returns Compared to Physical Core
Resource utilization	• Each Core has its own execution units & caches	• Threads share execution units & cache
Eg:	• Intel Core i7, AMD Ryzen Processor, Server grade CPU	• Intel Core i9, Xeon Processors

Threads : are the smallest unit of a Program.

- They are individual sequences of programmed instructions that can be executed independently by the OS scheduler.
- They enable concurrent execution within a program allowing multiple threads to run simultaneously and share the same memory space.
- Threads are lighter than ~~Processor~~ Processes and require fewer resources to create & manage.
- Threads can be categorized into user level threads (managed by the application) and Kernel level threads (managed by the OS)
- 1 Program ⇒ atleast 1 thread (main thread).

Thread lifecycle in OS

① New :-
The thread is in this state when it's created but hasn't yet started executing. At this point, the OS has allocated resources (like memory) but hasn't initiated execution.

② Runnable / Ready :-
Once the thread is ready to execute and waiting for the CPU time, it enters the runnable or ready state. This means the thread is eligible to run, but the CPU schedulers of the OS hasn't yet selected it for execution.

③ Running :-
In this state, the thread is actively executing its instructions on the CPU. Only one thread can be in this state per CPU core at any given time.

④ Blocked / waiting:-
A thread enters this state when it needs to wait for some event to occur or a resource to become available.

⑤ Dead / Terminated:-
A thread enters this state when it completes its execution or is explicitly terminated by the program. Once a thread is terminated, its resources are released by the OS.

Execution of threads in case of hardware.

① Processor Cores:-
Each Physical Core in a CPU is capable of executing instructions independently

② Thread Context:-
Hardware threads allows a Core to switch b/w different threads rapidly, appearing as if multiple threads are executing simultaneously.

③ Simultaneous Multi-Threading (SMT)
SMT technology allows each physical Core to handle multiple threads Concurrently by sharing certain resources, such as execution unit & cache.

④ OS Interaction
The OS schedules software threads onto available hardware threads. It manages thread execution, ensuring that threads receive CPU time based on their priority or the system's scheduling policies.

Thread Creation

① Thread Creating Mechanism.

Threads are typically created using system calls or library fun provided by the OS or Programming lang. runtime.

② Thread Attributes:

When creating a thread, attributes such as stack size, priority, and initial state can often be specified. These attributes determine how the thread behaves and interacts with ~~other~~ other thread and the system.

③ Memory Allocation:

Upon creation, the OS allocate necessary resources for the thread, including memory space for stack and other thread specific data structure.

④ Thread Entry Point:

Threads have a starting point, typically a fun or method that defines the initial code to execute when the thread begain execution. This entry point is specified during thread creation.

Thread Management

① Scheduling:

The OS scheduler decides when and for how long each thread runs on the CPU.

② State transitions:

Threads transition b/w diff states (eg. running, ready, blocked) based on their execution needs and interaction with system resources.

③ Concurrency Control:
Mechanism such as mutexes, semaphores, and condition variables are used to synchronize access to shared resources among threads, preventing data corruption and ensuring thread safety.

④ Thread Termination:
Thread can terminate either voluntarily by completing their task or involuntarily due to errors or explicit termination requests. Resources allocated to the thread are released upon termination.

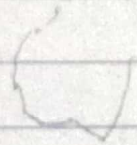
Threads inside OS

- Threads inside an OS refers to the fundamental units of execution managed directly by the OS Kernel.
- Threads are basic Kernel manages thread execution, scheduling them on CPU cores, and provides mechanisms for synchronization & Communication b/w threads.
- The OS Kernel manages thread execution, scheduling them on CPU cores, and provides mechanisms for synchronization & Communication b/w threads.

PAGE NO.
DATE / /

Hardware threads

- They are virtual execution units within a processor core that allow multiple threads to execute simultaneously on a single core.
- Achieved through technologies like hyper-threading or simultaneous multithreading, which duplicate core components to handle multiple instruction streams concurrently.
- Improve CPU utilization by reducing idle time on cores, enhancing multitasking performance across application.



Process Synchronization.

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

Type of Process on the basis of synchronization :-

- **Independent process:-**
  The execution of one process does not affect the execution of other process.
- **Cooperative process:-**
  A process that can affect or be affected by other processes executing in the system.

Process Syn Problem occurs in the case of cooperative processes because resources are shared in this

Race Condition

- when more than one process is executing the same code or accessing the same memory or any shared variable In that condition there is a possibility that the o/p is wrong. So for that all the process doing the race to say that my o/p is correct this condition is known as race condition.
- It is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in the the Critical Section differs according to the order in which the threads execute.

Critical Section

- It is a code segment that can be accessed by only one process at a time.
- It refers to a part of a program's code that access shared resources such as variable, DS or files, which must be accessed in an exclusive manner to avoid race condition and maintain data integrity.
- Only one process or thread is allowed to execute within the critical section at any given time to prevent conflicts and ensure that the shared resources are not accessed simultaneously by multiple entities.

Binary Semaphore

It is a type of semaphore that can have only two states: locked or unlocked. It acts as a simple flag that controls access to a shared resource or critical section by allowing only one process or thread to access it at a time. When a process acquires the semaphore (locks it), it changes its state to locked, preventing other processes from accessing the resources. When the process releases the semaphore (unlock it) the stage changes back to unlocked, allowing other process to acquire it.

Semaphore

It is an integer variable which is used in mutual exclusive manner by various Concurrent Cooperative process in order to achieve synchronization.

Semaphore

Counting Sema
- ∞ to ∞

Binary Sema
(0, 1)

Counting Semaphore

It is like a counter that controls access to multiple instances of a resources. It can have a range of values, typically starting from a positive integer. Each time a process or thread want to access the resources, it decrement the semaphore value. If the value is greater than or equal to zero, access is granted, and the process continues. However, if the value is zero or negative, the process is forced to wait until the semaphore value increases again other process can increment the semaphore value when they are done using the resource. This way, the semaphore effectively count the available instances of the resource and regulate access accordingly

Binary Semaphore

P(S) → wait, sleep, down. → Entry code
V(S) → signal, wake-up, up → Exit code
~~Code~~ → Code

```
P(Semaphore S)
{
  while (S == 0);
    S = S-1;
}
→ Critical section {
V(Semaphore S)
{ S = S+1;
}
→ remaining Section code
```

Counting Semaphore

value of S is the count of resources available

→ Code

```
P( Semaphore S)
{  S = S-1;
   if (S < 0)
     { Put Process in suspended list sleep();
     }
   else { return };
}
Critical section
V(Semaphore S)
{ S = S+1;
  if ( S <= 0)
  { Select Process from suspended list wakeup();
  }
}
```

mutex	Semaphore
• it is an obj	• It is an Integer.
• It works upon the locking mechanism	• It uses signaling mechanism.
• operation :- Lock, unlock	• operation :- wait, signal.
• It doesn't have any subtype	• Types :- Binary, Counting
• A mutex can only be modified by the process that is requesting or releasing a resource	• It works with two atomic operations (wait, signal) which can modify it.
• If the mutex is locked then the process needs to wait in the process queue and mutex can only be accessed once the lock is released	• If the process needs a resource, and no resource is free. So, the process needs to perform a wait operation until the semaphore value is greater than zero.