Java Multithreading: Synchronization, Locks, Executors, Deadlock, Countdown Latch, CompletableFuture

Basics:

Central Processing Unit (CPU):

- Brain of computer
- Responsible for executing instructions from programs
- Performs basic ALU and I/O operations (using instructions)
- Examples: Intel Core i7 8, AMD Ryzen 7

Core:

Individual processing unit within a CPU

Quad-core processor (4 cores) → 4 tasks simultaneously

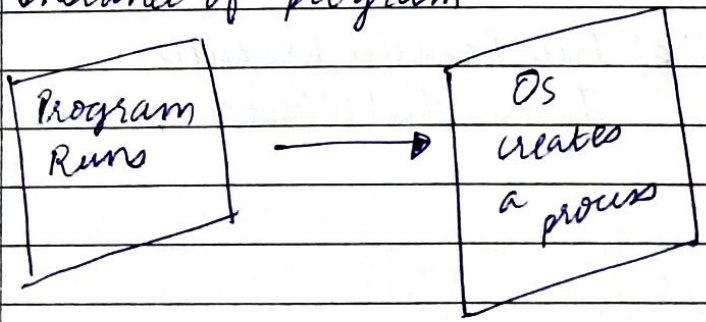
- Core 1: Web Browser
- Core 2: Music Player
- Core 3: Download Manager
- Core 4: Background System Update

Program:

Set of instructions. Example: Microsoft Word

Process:

Instance of program



→ This process manages execution of program

__/__/__

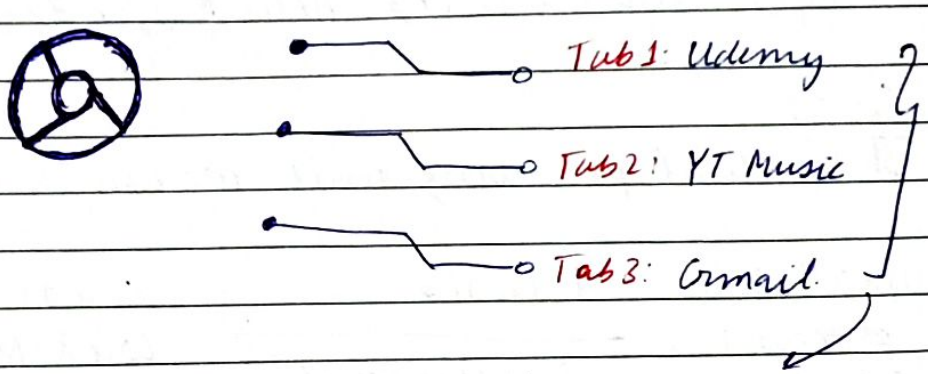
Thread:

Smallest unit of execution within a process.

{Process में जो दौड़े दौड़े काम हैं, those are threads}

Ex: In MS-Word, text predictions, spell check, auto-save takes place simultaneously.

Google Chrome



All are independent tasks and can execute concurrently.

Multitasking

- Allows OS to run multiple processes simultaneously

Single Core CPU's: Time sharing: Rapidly switching between tasks

Multi Core CPU's: True Parallel Execution: Tasks distributed across cores.

OS Scheduler

Balances the load, ensuring efficient and responsive system performance.

Ex: Browsing the internet while listening to music and downloading a file.

While performing multitasking, OS can assign different tasks to different cores (instead of single core)

Efficiency: Multiple cores >>> Single core

Multithreading

Ability to execute multiple threads within a single process concurrently.

Multithreading for Web-browser

Web browser has separate threads for: → Makes browser responsive and efficient

- Rendering the page
- Running Javascript
- Managing user inputs

→ Multithreading enhances efficiency of multitasking

Ex: Downloading multiple files from browser:
Single-Threaded Approach (download each file one after the other)

1. Download File : 5 min
2. Download File : 4 min
3. Download File : 6 min

Total-time : 15 minutes

__/__/__

Multithreaded Approach: (All 3 files at the same time)

1. Thread 1 (File 1): 5 min
2. Thread 3 (File 2): 1 min
3. Thread 2 (File 3): 6 min

] occurs simultaneously

Total Time : 6 min

[More efficient]

Time Slicing

- Dividing CPU time in small intervals (time slices/ quanta)
- Function:
  OS scheduler allocates time slices to different processes and threads, ensuring each gets fair share of CPU time.
- Purpose:
  * Preventing any single process/thread from monopolizing the CPU.
  * Improving responsiveness and enabling concurrent execution.

Context Switching

- Process of saving the state of a currently running process/thread and loading the state of next one to be executed.

__/__/____

Function:
When a process / thread's time slice expires, OS scheduler performs a context switch to move the CPU's focus to another process/thread.

Purpose:
* Allows multiple processes and threads to share the CPU, giving the appearance of simultaneous execution on a single-core CPU or improving parallelism on multi-core CPUs.

	Single-core	Multi-core
○	Both threads & processes are managed by OS scheduler.	○ Both threads & processes can run in true parallel on different cores.
○	Uses time-slicing & context-switching to create illusion of simultaneous execution.	○ OS scheduler distributes tasks across the cores to optimize performance.

Multitasking: ○ Running of multiple applications
○ Operates at level of processes

Multithreading: ○ More granular, dealing with multiple threads within same application / process.
○ Operates at level of threads (smaller units within process)

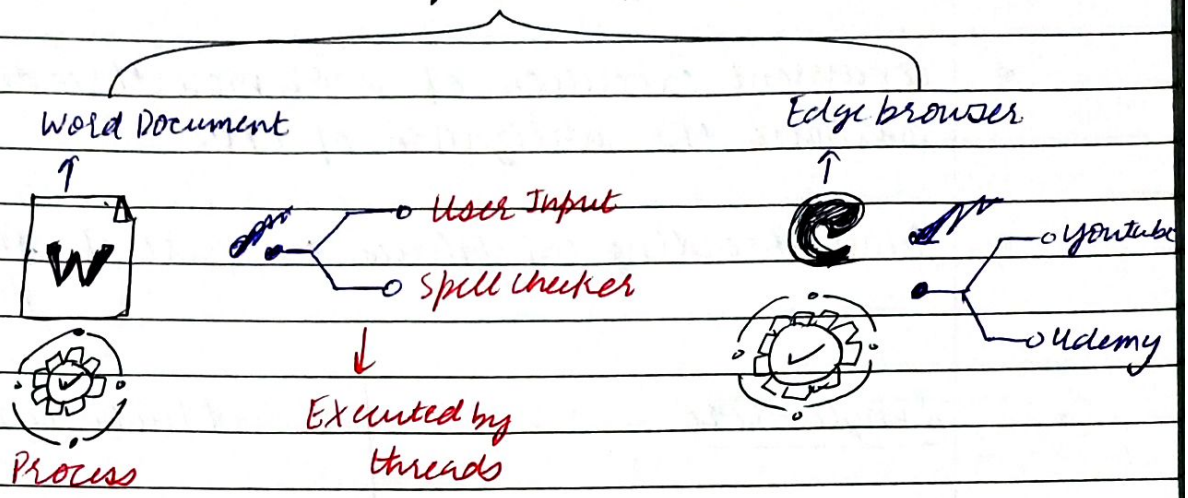
__/__/____

Multitasking (Achieved through → Multithreading)	
○ Running of multiple applications / processes within an OS. (simultaneously)	○ Running of multiple threads within same application / process
○ Operates at the level of processes.	○ Operates at the level of threads.
○ Involves managing resources b/w completely separate programs, which may have independent memory spaces & system resources.	○ Involves managing resources within a single program, where threads share same memory & resources.
~~Performance~~	
○ Improves productivity & utilization.	○ Single application performs multiple tasks at the same time. Improves application performance & responsiveness.
Ex: Office manager (OS) assigns different employees (processes) to work on different projects (applications) simultaneously.	Ex: Within single project (application), a team (process) of employees (threads) work on different parts of project at the same time, collaborating & sharing resources.
Each employee works on different project independently	

__/__/____

Ex:

Multitasking (Managed by OS)



__/__/____

Multithreading in Java

- Concurrent execution of 2 or more threads to maximize the utilization of CPU.
- Multithreading capabilities are part of java.lang package.

Single-Core	Multi-Core
○ Multithreading managed by JVM & OS	○ JVM distributes threads across multiple cores, allowing true parallel execution of threads.
○ Threads share single core & time slicing is used to manage thread execution	

- A thread is a lightweight process, the smallest unit of processing.
- Java supports multithreading through java.lang.Thread class & java.lang.Runnable interface
- When java program starts, one thread begins running immediately (Main thread)
- Main thread is responsible for executing main method of program.

__/__/__

- Create New Thread in Java
  - Extend the thread class
  - Implement the Runnable interface.

Thread Lifecycle

The lifecycle of a thread in java consists of several states, which a thread can move through during its execution.

New:
Thread is created but not yet started

Runnable:
After the start method is called, the thread becomes runnable. It's ready to run & waiting for CPU time.

Running:
The thread is in this state when it's executing

Blocked / Waiting:
Thread is waiting for a resource or another thread to perform an action.

Terminated:
Thread in this state has finished executing

Locks

1. Intrinsic : These are built into every object in Java. We don't see them, but they're there. When we use a synchronized keyword, we're using these automatic locks.

2. Explicit : These are more advanced locks which we can control ourselves, using the Lock class
   java.util.concurrent.locks.
   We explicitly say when to lock & unlock, giving more control.

Lock is an interface.
We create an object of Lock as:

```
Lock lock = new (Implementation class);
```

lock ↓ object

since Lock is an interface we cannot create its object

Methods: lock.lock() - Thread trying to acquire lock
lock.tryLock()
lock.unlock() - Thread which runs this method is trying to release the lock.

lock.lockInterruptibly(); → Thread is in blocked waiting state, here it's indefinitely.