Concurrency → 4.

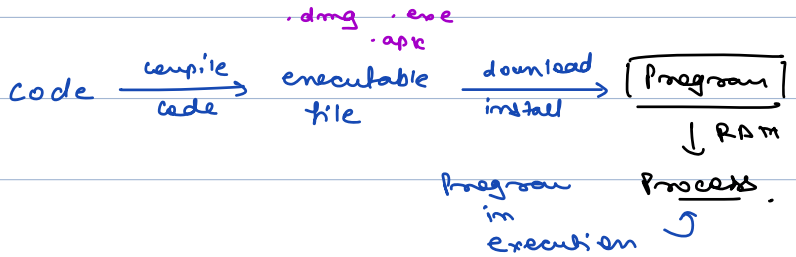
Agenda :-

Process vs Threads

Concurrency vs Parallelism

Multicores

& Multithreaded Program



for(i=0; i<=9; i++){

2.2 GHz

↓

2.2×10^9 instructions/sec.

Program in Execution

↓

Process.

Process Control Block

pid

list<variable>

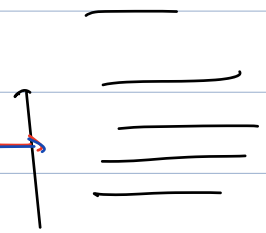
registers

priority

memory details

Program Counter

call stack



MS Word

Spell check → P_1

auto save → P_2

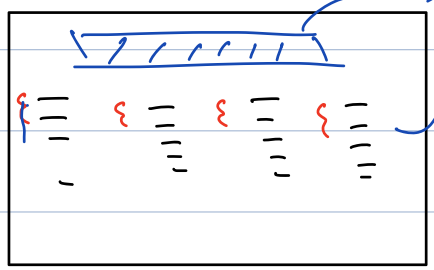
O/P display → P_3

auto correct / autosuggest → P_4

Thread

↳ unit of cpu execution.

Process



→ shared memory.

→ a thread is something which is actually executed by CPU

IPC

Process Control Block

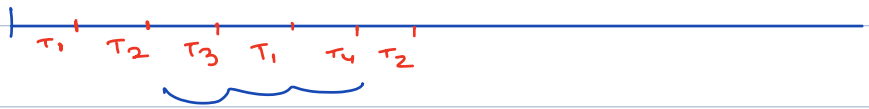
pid
list<variable>
registers
priority
memory details

Thread

Program Counter
call stack
↓
save fixed memory if traversal only

1. Data sharing is easier among threads.
2. creation of threads is easier.
3. Threads are light weight.

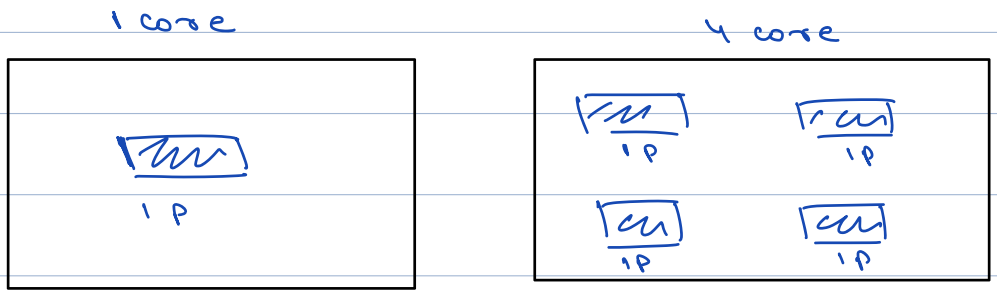
Context Switching



multiple processes.

Single core vs Multi-Core

int x = 10; int y = 10;



1 core = 1 thread at a time.

4 core = 4 threads.

i3 → dual core

i5 → quad core

i7 → quad core → hyperthreading.

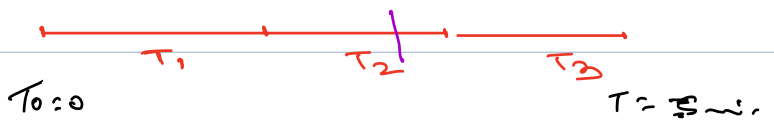
1 core = 2 threads.

Concurrency & Parallelism

Case-1

→ Single Core

→ Until one thread completes, it won't move to the next one.



① How many threads are in Partial state at the same time ?.

1

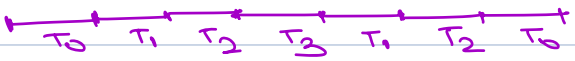
② How many threads are executing at the same time ?.

1

Case-2 → concurrency.

→ Single core

→ no necessity of completing 1 thread.



① How many threads are in partial state at the same time?

Many.

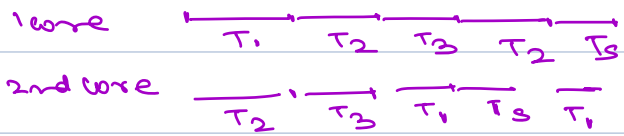
② How many threads are executing at the same time?

1

Case- 3

→ Multiple Cores

→ Context Switching



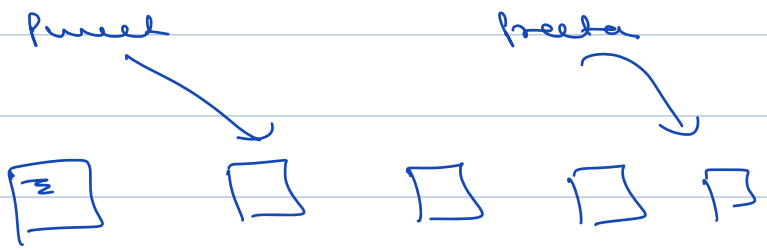
① How many threads are in Partial state at the same time ?.

many

② How many threads are executing at the same time ?.

2

→ Parallel



$$\frac{a}{T_1} \quad \cdots \quad \frac{b}{T_1}$$

$$\frac{c}{T_1}$$

1) $a + b = c$

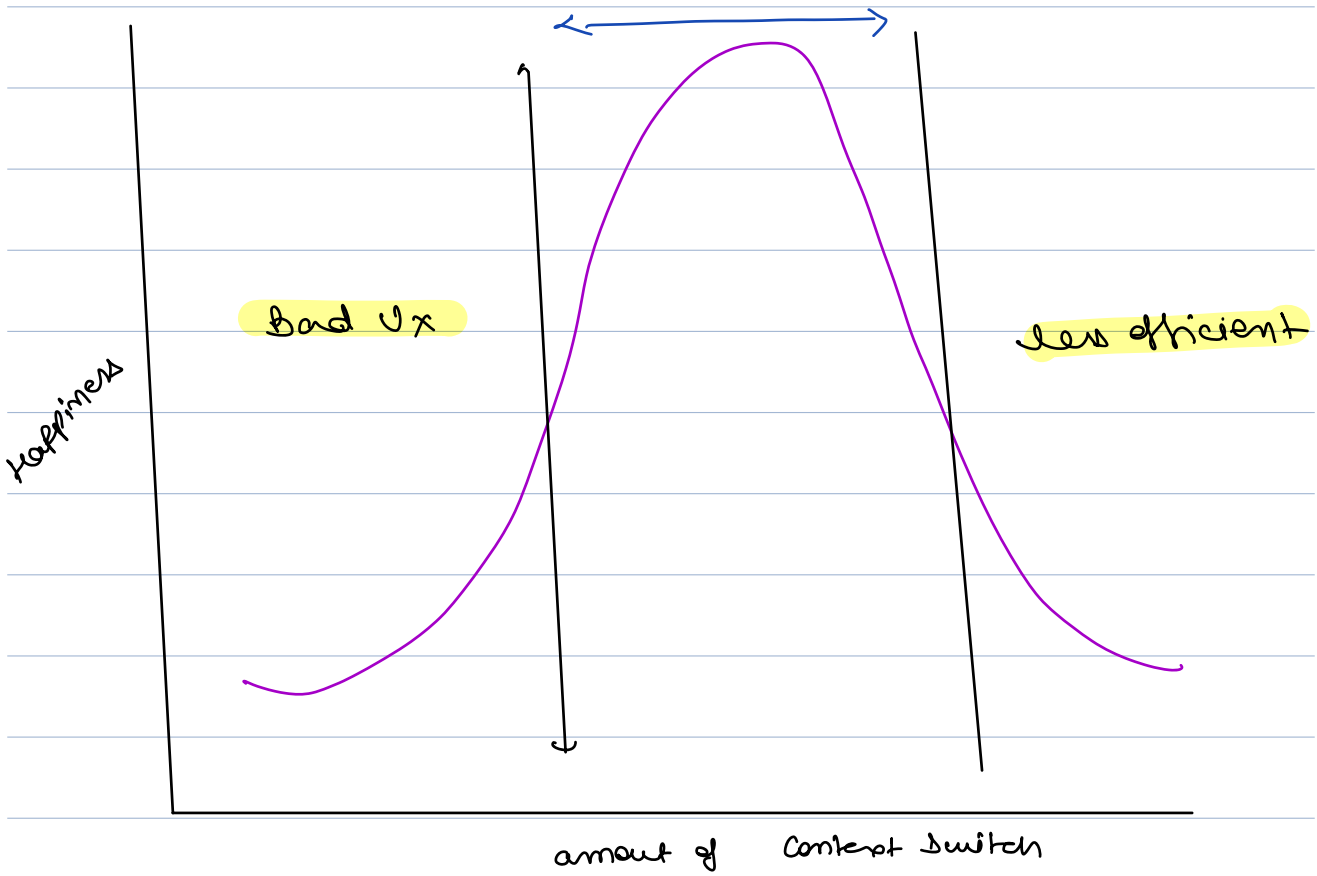
2) $a + b > c$

3) $a + b < c$

$T_1 \longrightarrow T_2$

① Save current state of T_1

② Load the previous state of T_2.



2 codes → hello world

→ 1-100 → multithreading

10:20pm - 10:30pm

- **Process**: The kitchen as a whole, responsible for preparing and serving meals.
- **Threads**: The individual chefs working in the kitchen. Each chef (thread) can work on a different dish (task) at the same time, sharing the same kitchen resources (oven, ingredients, utensils).

- **Process**: The web browser application itself.
- **Threads**:
  - One thread handles user interactions (clicks, typing).
  - Another thread loads and displays web pages.
  - Another thread runs background tasks like preloading content or running extensions.

Analogy: Imagine a single chef (CPU) in a kitchen preparing multiple dishes (tasks). The chef can start chopping vegetables for one dish, then move to stirring a pot for another, and then check the oven for a third dish. The chef is not cooking all dishes simultaneously but is managing and progressing on all of them. → concurrency

Analogy: Imagine multiple chefs (CPU cores) in a kitchen, each preparing a different dish at the same time. Each chef works independently on their own dish, so all dishes are being prepared simultaneously. → Parallelism.

Java : multithreaded program

① define a task

↓

create a class for the task

```
class HelloWorldPrinter {

}
```

② implement Runnable → run().