**MULTITHREADING**

Day14:

Multiprocessing

Multithreading

./app& -- & mean it will run in background

If many applications are running, then command prompt will be parent and apps are child. The link between these apps is thread.(lwt – light weighted threads or processes).

A **thread** has its own stack, registers and control flow.

**Concurrency** – parallelly running multiple things at a time. (multiple tasks simultaneously)

*Necessity of concurrency*

1. Responsiveness – remains responsive while processing background tasks.
2. Utilizing Multi-core Processors – execution of multiple instructions at once.
3. Efficient Resource utilization – to better utilize available resources by avoiding idle times.
4. Scalability – multiple clients need to be handled simultaneously.
5. Improved throughput – the amount of work an application can process in each amount of time.

*Concurrency in programming*

1. Multithreading – multiple threads within a single process are created to perform different parts of a task concurrently. Threads share the same memory space, allows for efficient communication but requires careful synchronization to prevent race conditions.
2. Multiprocessing – Separate processes are created to handle different tasks concurrently. Each process has its own memory space. But requires inter-process communication (IPC) methods like pipe.
3. Asynchronous Programming – Non-blocking functions are used to handle tasks such as I/O operations without blocking.

Concurrency vs parallelism

Methods to Achieve concurrency

1. Multithreading using POSIX Threads
2. Using fork( ) for multiprocessing
3. Asynchronous I/O
4. Atomic operations
5. Semaphores
6. Message Queues

In threading – each thread has its own stack, registers and program counter

Traditional Process = process context + code, data and stack

Program context – Data registers, condition code, stack pointer, PC

Kernal context – VM Structures, Descriptor table, break pointer

Alternative Process = thread + code and data

**Threads vs Processes**

*Similarity*

1. Each has its own logical control flow
2. Each can run concurrently
3. Each is context switched

*Different*

1. Threads share code and data, processes do not
2. Threads are less expensive than processes

**To create a thread -**

pthread\_create(&thread\_name,thread\_atributes,function\_name,function\_arguments);

**Wait for threads -**

To get result of thread, in main, it should wait for some time. It can be done by **pthread\_join**.

pthread\_join(thread\_name,thread\_atributes);

**RACE CONDITIONS**

If two threads working to do same, it will get confused.

Ex: a =0, and there is a function to increment a value by 1.

Using 2 threads in main by calling same function. First thread will increment and waits if other thread works. Then first thread will get confused as who worked on it. This is known as race conditions.

To avoid race condition, make a lock of it. Use mutex for synchronization.

**Mutex for synchronization**

pthread\_mutex\_t lock;

pthread\_mutex\_init

pthread\_mutex\_destroy