

Process - program in execution

**Thread** ---- path of execution in a process

Path ---- sequence of points , we follow one point after another in the sequence

Starting point

|  
|-  
|  
|-

Ending point

Thread ---- **sequence** in which a set of instructions will execute from start to end !!!

EVERY PROCESS **MUST** have at least one path of execution ( thread )

```
F1(){.....}  
F2(){.....}  
F3(){.....}  
....  
...f100(){.....}
```

---

DEFAULT **sequence** of instructions in the program is given in main  
MAIN is the DEFAULT THREAD in any PROCESS !!!

```
Void main()  
{ -----start point of the thread path  
    F33();  
    F44();  
    F45();  
} -----end point of the thread path
```

---

**One** process can have MANY THREADS ---- MULTITHREADING !!!!

WHY would we need many threads within one process ???

Assuming I have a single CPU ----

One process performs 4 tasks ---IN a **single threaded application**

```
T1 s-----eT2s-----eT3s-----eT4s-----e  
      10ms          7 ms          6ms          9ms  
0-----10,-----17,-----23,-----32
```

IN a **multithreaded application**

Ready Q -----T1,T2,T3,T4

These threads will compete with each other in RR

T1-----3T2-----6T3-----9T4-----12T1-----15T2-----18T3-----21T4

Threads will run one at a time on the CPU for the time slice

ALL the threads hit the CPU without waiting for other threads to finish  
they only wait for time slice ---which is very less }}} RESPONSE time improves

!!

---

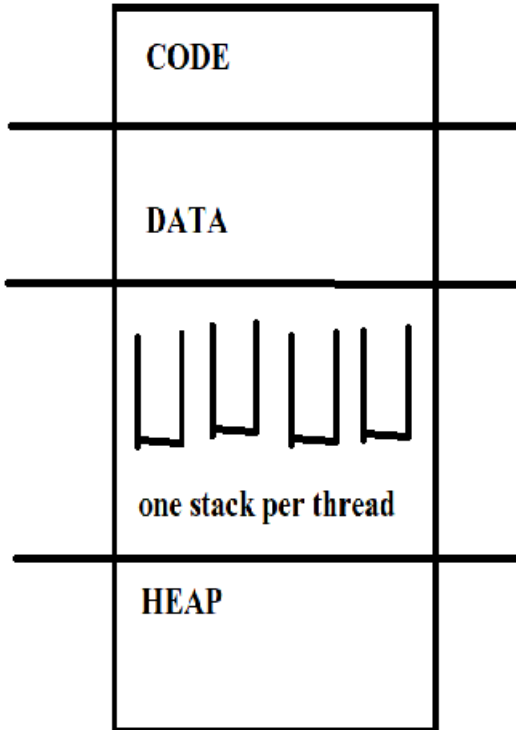
### Examples of multithreading

Process	Th1	Th2	Th3
One Note	Editor	spellchecker	Autosave
Browser	Gmail tab	Flipcart tab	Google tab
Eclipse	Editor	Compiler	Code assist
Game	Speaker	Graphics	Timer
Live Cricket	Score	Ad	Match
Zoom	Share screen	Chat	Video
Google maps	Computing path	Shows screen	Navigation instruction
Media player	Play the song	Change playlist	Control volume

---

### Life Cycle

	Created	Ready	Runnin g	IO wait	Termin ated
Process	Pid, pcb , addresss space( code, data ,stack, heap) loaded in RAM	Ready queue( l inux based OS process based ready queue	Get CPU for a time slice	Wait for IO to complet e or sleep to complet es	Last instruct ion execute d
Thread	Tid , TCB , ONLY the STACK is added to the process address space in RAM , other segments are shared by all threads in the process STACK is exclusive for each thread	Ready queue ( windo ws have thread based ready queue)	same	same	same

	 <p>The diagram shows a vertical stack of memory sections. From top to bottom: a box labeled 'CODE', a box labeled 'DATA', a box labeled 'one stack per thread' containing four small U-shaped icons representing individual thread stacks, and a box labeled 'HEAP'. Horizontal lines separate these sections.</p>				
	As thread creation involves only stack creation ===less effort as compared to process creation ==== LIGHT WEIGHT PROCESS				

---

Stack holds Activation Frame for a function call ( AF is pushed in the stack when function is called and it is popped from the stack when function returns )

	F4
F2	F3
F1 x=20 p1=addr of dynamic variable	F2
main	T1

---

Inter thread communication is very easy ( because code, data and heap areas are shared )  
Inter process communication is very difficult ( because two processes do not share any part of address space )

---

Kernel Threads  
User Threads

Kernel threads manage user threads !!!!!

One to one	One kernel thread per user thread
Many to one	One kernel thread all user threads
Many to many	A few kernel threads manage all user threads

---

To create threads in Linux ----- POSIX thread library !!! = pthread library has APIs for thread management!!!

Posix thread library API

1. Pthread\_t = struct that holds the thread info
2. Pthread\_create = this method is used to attach the thread with the path of execution
3. Pthread\_join = this method is used to block main thread till the editor thread is not complete

If main thread terminates then other threads created by main are forced to terminate!!!

---

2.30 please do the thread program shared with you!!!  
 3 pm onwards we will do shell script !!!

---

