April 28, 2020

Concept of Thread is a perfect construct or application of Parallelization. **MS WORD** is the most familiar example where thread is used.

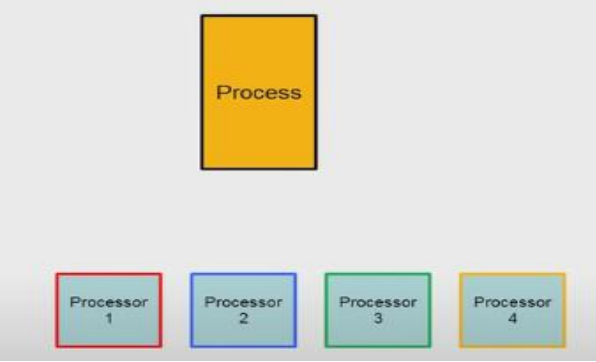
```c
#include <stdio.h>


unsigned long addall(){
  int i=0;
  unsigned long sum=0;

  while (i< 10000000){
     sum += i;
     i++;
  }
  return sum;
}


int main()
{
   unsigned long sum;
   srandom(time(NULL));
   sum = addall();
   printf("%lu\n", sum);
}
```

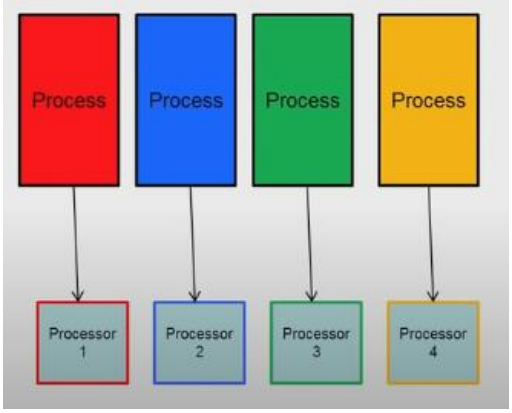
In this program, we have a function addall(). The function sums up the first 10000000 numbers. The sum is returned and printed inside the main function. Obviously this program could be written in N-many different ways other than this. To explain the need of thread this typical program is considered . After writing this program, it is compiled and executed hence process is created. Suppose the system is having four different Processors. But the program will still take too long to execute as the process will be allocated to one processor and the other processors will be idle for that time.

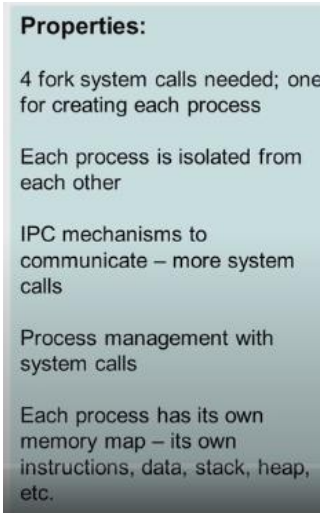


So, the above example is not a very ideal situation . So can be done is total four processes can be created . each loop does ¼ th of the whole work.

10000000 / 4 = 2500000

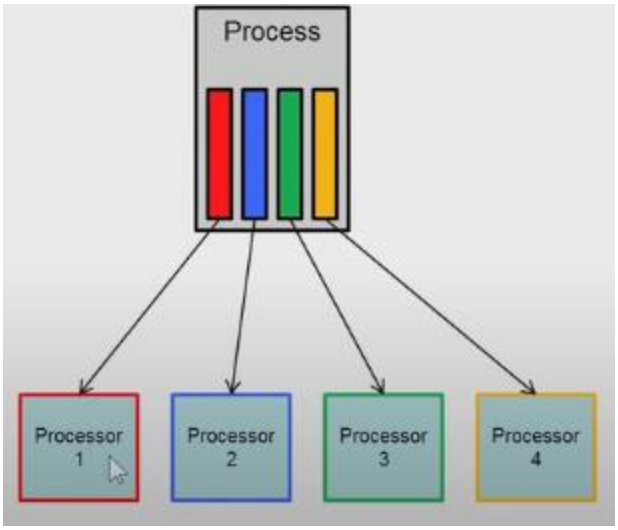
So, the scenario will be four time faster as given below :



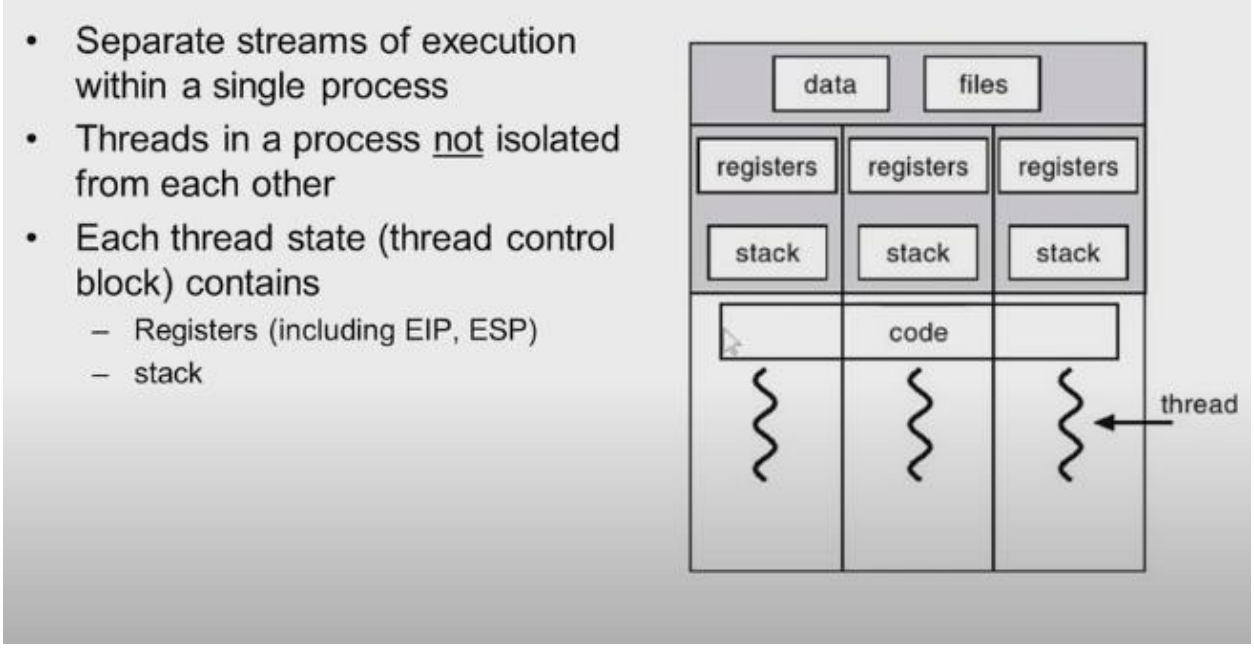


So, the above mentioned mechanishm incurs a huge amount of overhead in our work. So can we do better ?

The answer is yes, using threads.



So the properties of thread is given below:



From the abovementioned digarm it can be understood that threads have their own context for execution. Execution context of each thread is having their own register. It helps them in the time of context switch,to run it in an error free manner.

- **A simple comparison between processes and threads.**

Similarities

- Like processes threads share CPU and only one thread active (running) at a time
- Like processes, threads within a processes, threads within a processes execute sequentially.
- Like processes, thread can create children.
- And like process, if one thread is blocked, another thread can run.

Differences

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task .
- Unlike processes, thread are design to assist one other.
- As processes may originate from different users they may or may not assist one another.

THREAD	*PROCESS*
• A thread has no data segment or heap	• A process has code, heap, stack, other segments
• A thread cannot live on its own. It needs to be attached to a process	• A process has at-least one thread.
• There can be more than one thread in a process. Each thread has its own stack	• Threads within a process share the same code, files.
• If a thread dies, its stack is reclaimed	• If a process dies, all threads die.