**Multithreading**

We can open multiple applications at a time in Windows.

OS uses process to execute all the applications internally

Process: is a part of the OS, responsible for executing a program

We can see the processes in task manager( each process runs application) and each process runs parallely.

Behind the screen we even have background process(OS behind the screen, runs the background process)

Windows services : which run at the background ( found at the control panel services window, administrator tools)

Each process runs one application

OS uses process to run application, while under process, we have thread.

Thread : light weight process. It is a unit which executes a code under application. Every application has logic, and that logic is executed by thread.

Every application by default has a one thread to execute the program and that is called main thread.

Every program by default is single threaded model.

To get the default main thread :

Thread t=Thread.CurrentThread; // to get the current executing thread.

t.Name= “Main thread” // to get the main thread

The drawback of the single threaded model is : In single threaded model, the process i.e methods start one after the other. After completing the execution of one method, then only new method will be started to execute. So in this approach, the drawback is that if a particular method waits for another operation in it to respond(like the database operations) then the other methods has to wait until the execution of other thread.

To overcome this problem, we have multi threading

Multi threading : in one process we can have multiple threads. The execution takes place simultaneously. The os allocates some time to each thread to execute, based on the time sharing all the methods are given equal responsibility.

Multi threading works based on time sharing. The main advantage is maximum utilisation of CPU resources.

How multi threading works?

Thread1 -> method 1

Thread 2 -> method 2

Thread 3 -> method 3

There will be four threads in program along with main thread

The os gives time to each thread ( for example each thread is given like 2 seconds, then while executing thread1 if it is not done, also the control goes to thread2 and similarly the execution runs until, the execution of each method is done.)

Suppose, thread2 is not able to execute, then it would go to the next thread after its allocated time is done.

using System;

using System.Threading;

class ThreadTest{

static void Test1(){

Console.WriteLine(“Test 1:”+i);

}

static void Test2(){

Console.WriteLine(“Test 2:”+i);

}

static void Test3(){

Console.WriteLine(“Test 3:”+i);

}

static void Main(){

Thread t1=new Thread(Test1);

Thread t2=new Thread(Test2);

Thread t3=new Thread(Test3);

t1.start();

t2.start();

t3.start(); // all the three threads are runned parallely.

}}

**Constructors for the Thread Class**

1. ThreadStart obj=new ThreadStart(Test); // delegate instance created by user itself.

Thread t=new Thread(obj);

1. Thread t= new Thread(test); //clr creates the instance of delegate threadstart
2. Instantiation : binding method with delegate
3. ThreadStart obj=Test;

Thread t=new Thread(obj)

1. ThreadStart obj=delegate() {Test();};
2. For a parametrised method

ParameterizedThreadStart obj =new ParameterizedThreadStart(Test);

Thread t=new Thread(obj);

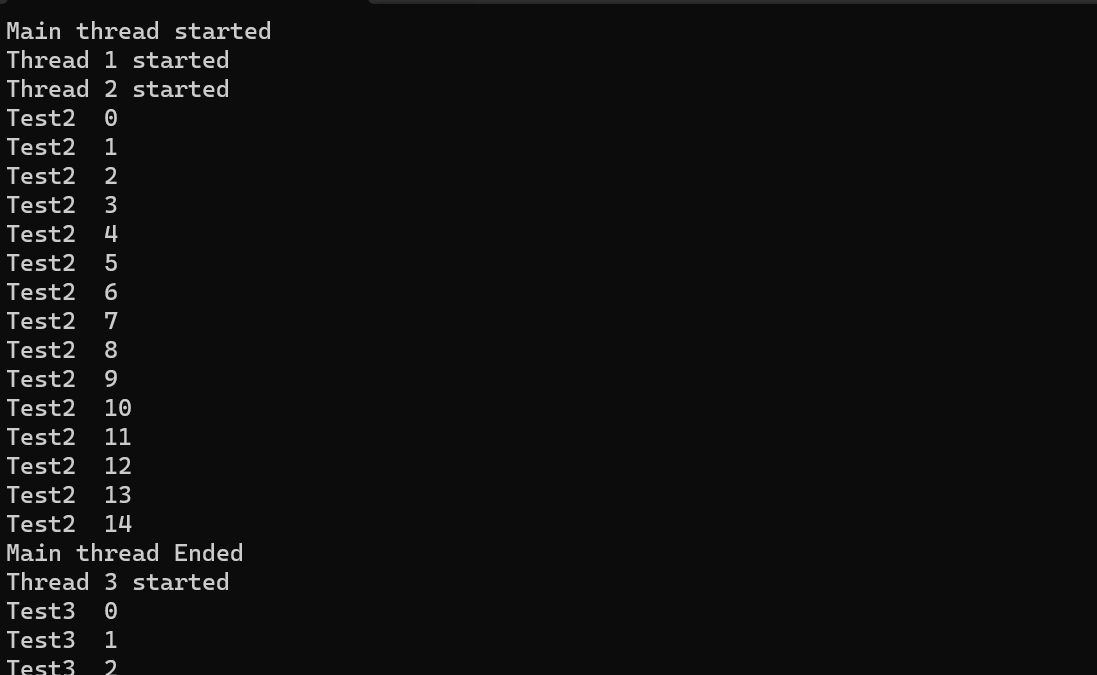
t.start(75);

Public void Test(object max){ // the thread points to this method itself

Console.WriteLine(max);

}

**Join in Threads**



Main thread started and ended, before all the threads completed for the code as:

Console.WriteLine("Main thread started");

Thread t1 = new Thread(Test1);

Thread t2 = new Thread(Test2);

Thread t3 = new Thread(Test3);

t1.Start();

t2.Start();

t3.Start();

Console.WriteLine("Main thread Ended");

But, the main thread cannot exit the program until all the child threads are finishing the job.

So, to ensure that we use the join

After using the join , main thread ended last

Console.WriteLine("Main thread started");

Thread t1 = new Thread(Test1);

Thread t2 = new Thread(Test2);

Thread t3 = new Thread(Test3);

t1.Start();

t2.Start();

t3.Start();

t1.Join();

t2.Join();

t3.Join();

Console.WriteLine("Main thread Ended");



Join tells that the calling thread cant exit from the program until all the called threads(child threads) are executed and terminately.

If we use thread.sleep for a particular child thread, then also the main thread executes.

T1.join(3000) // the parent thread waits for 3000ms to complete the execution of thread, if it doesn’t complete then after the 3000ms the main thread gets terminated.

**Thread locking**

The context switching occurs here,

public void Display()

{

Console.Write("C Sharp is an");

Thread.Sleep(5000);

Console.WriteLine("Object oriented language");

}

static void Main(string[] args)

{

Program p=new Program();

Thread t1 = new Thread(p.Display);

Thread t2 = new Thread(p.Display);

Thread t3 = new Thread(p.Display);

t1.Start();

t2.Start();

t3.Start();

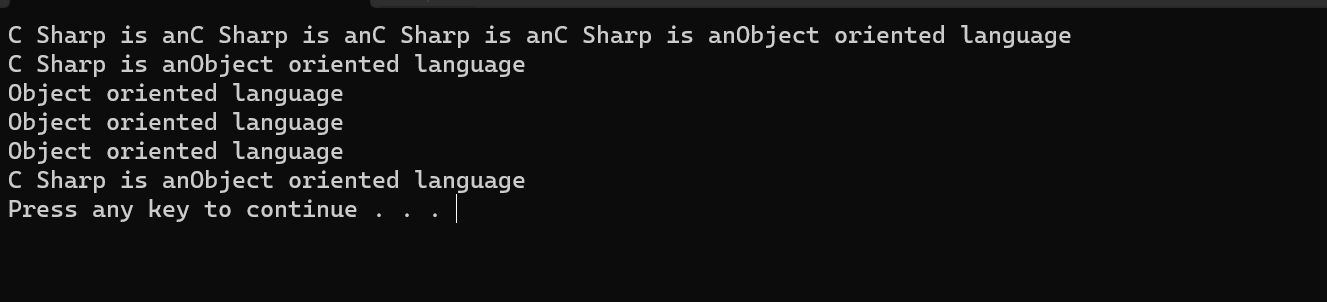
p.Display();

p.Display();

p.Display();

}

The threads are simultaneously shared among themselves based on the os time sharing,



This is a normal c# program, if in real time one thread tries to access the same resource and another thread is trying to modify the same record, it will be a problem

To resolve this, we have a locking mechanism

The code that is being used by multiple threads can be locked, then multiple threads cant access the same code simultaneously and after one thread executes the code then only it goes to the 2nd thread.

lock (this)

{

Console.Write("C Sharp is an");

Thread.Sleep(5000);

Console.WriteLine("Object oriented language");

}

Access to resource is given to one thread only. When multiple threads access a same code, then we can keep it in the lock so that single thread can use them.

**Thread Priorities**

There are five priorities for a thread :

1. Lowest
2. Below normal
3. Normal - default
4. Above Normal
5. Highest – uses more cpu resouces

By default, every thread has Normal priority.

Thread t1 = new Thread(IncrementCount1);

Thread t2 = new Thread(IncrementCount2);

// if we make the main thread goes to sleep then in that time the t1, t2 threads can be executed

t1.Start();

t2.Start();

t1.Priority= ThreadPriority.Lowest;

t2.Priority= ThreadPriority.Highest; // count 2 value will be higher

Console.WriteLine("Main thread to sleep");

Thread.Sleep(2000);

Console.WriteLine("Main thread wokeup");

t1.Abort(); // terminates the thread

t2.Abort();

t1.Join();

t2.Join();

Console.WriteLine("Count 1:" + count1);

Console.WriteLine("Count 2:" + count2);

A black screen with white text

Description automatically generated