**Threading**

**1. What is mean by process?**

A process is a collection of threads that share the same virtual memory. A process has at least one thread of execution, and a thread always run in a process context.

**2. What is Thread?**

Threads are the basic unit to which an operating system allocates processor time, and more than one thread can be executing code inside that process.

**3. What is Multi-threading?**

Multi threading is the collection of thread executed with in the same program to generate the output.

**4. What is Multi-tasking?**

It's a feature of modern operating systems with which we can run multiple programs at same time example Word, Excel etc.

**5. What is the namespace used for threading?**

System.Threading

Note: - .NET program always has at least two threads running one is the main program and second is the garbage collector.

**6. What is mean by AppDomain?**

Operating system process are subdivides into lightweight managed sub processes called application domains. One or more managed threads (represented by System.Threading.Thread) can run in one or any number of application domains within the same managed process. Although each application domain is started with a single thread, code in that application domain can create additional application domains and additional threads.

**7. Can you explain in brief how can we implement threading?**

This sample explains about the implementation of the threading. Let start this example by creating a class with two methods called "Thread1()", "Thread2()"

class TestClass

{

public void Thread1()

{

int index = 0;

for (index = 0; index < 100; index++)

{

Console.WriteLine("This is from first thread: {0}", index.ToString());

}

}

public void Thread2()

{

int index = 0;

for (index = 0; index < 100; index++)

{

Console.WriteLine("This is from second thread: {0}", index.ToString());

}

}

}

Create a new console application; in the main method we will be creating the new instance of the Thread and pass the address of the TestClass method as constructor parameter to the Thread class.

Start the Thread by calling the Thread.Start() method.

class Program

{

static void Main(string[] args)

{

TestClass \_objTestClass = new TestClass();

Thread th1 = new Thread(new ThreadStart(\_objTestClass.Thread1 ));

Thread th2 = new Thread(new ThreadStart(\_objTestClass.Thread2));

th1.Start();

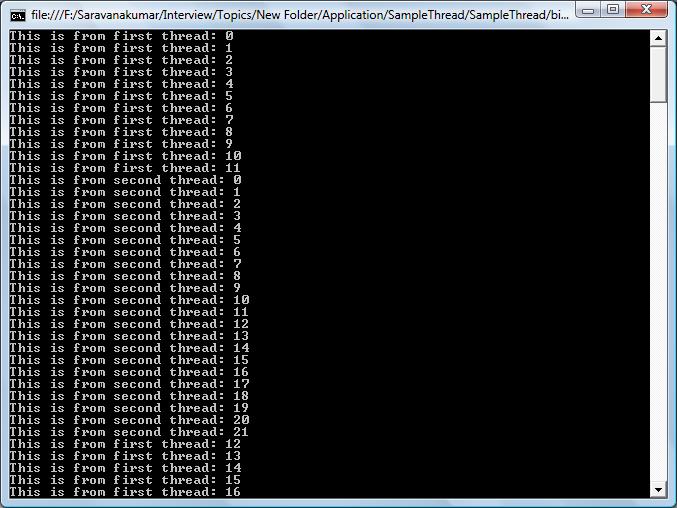
th2.Start();

Console.ReadLine();

}

}

Output of the windows is shown below. In which we can identify that Thread1 and Thread 2 are called simultaneously. We cannot define when Thread1 or Thread2 is called.



**8. What are different levels of threading priorities are available?**

Priority of the thread execution can be changed by using the "Priority" property of the thread instance.

Example:

ThreadName.Priority = ThreadPriority.BelowNormal

Following are the different level of the Thread priority available

* ThreadPriority.Highest
* ThreadPriority.AboveNormal
* ThreadPriority.Normal
* ThreadPriority.BelowNormal
* ThreadPriority.Lowest

**9. What is mean by Theard.Sleep()?**

The *Thread.sleep()* method effectively "pauses" the current thread execution for a given period of time. This method takes an integer value as parameter that determines how long the thread needs to be paused.

Example:

System.Threading.Thread.Sleep(4000);

**10. How can we make a thread sleep for infinite period?**

You can also place a thread into the sleep state for an indeterminate amount of time by calling *Thread.Sleep (System.Threading.Timeout.Infinite)*. To interrupt this sleep you can call the *Thread.Interrupt* method.

**11. What is mean by Thread.Suspend and Resume?**

* **Thread.Suspend()** - this method is used to suspend the thread execution. If the method is already suspended, it does not have any effect.
* **Thread.Resume()** - Suspended thread can be resumed using this method call.

**12. What is difference between Thread.Sleep and Thread.Suspend()?**

* Thread.Sleep() method will immediately place the thread under wait state, where as
* Thread.Suspend() method will not go into wait state until .net determines that it is in a safe place to suspend it.

**13. What the way to stop a long running thread?**

Thread.Abort() stops the thread execution at that moment itself.

**14. How will we get current thread?**

Using *System.Threading.Thread.CurrentThread* we will be able to get the current thread instance.

**15. How we can make the thread run in background?**

By setting *ThreadName.IsBackground = true* will run the Thread in background process.

Example:

TestClass \_objTestClass = new TestClass();

Thread th1 = new Thread(new ThreadStart(\_objTestClass.Thread1 ));

th1.IsBackground = true;

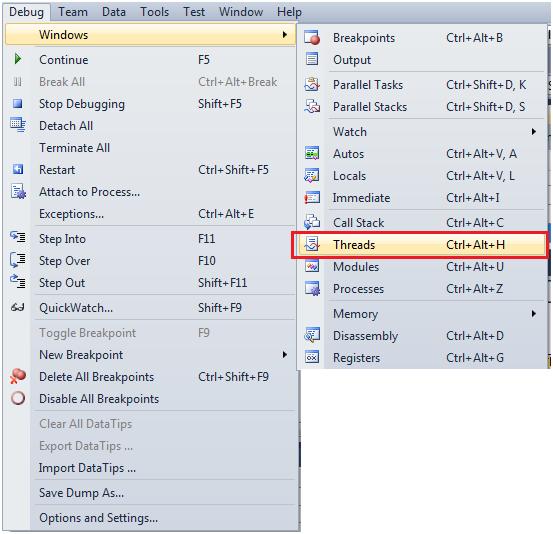
th1.Start();

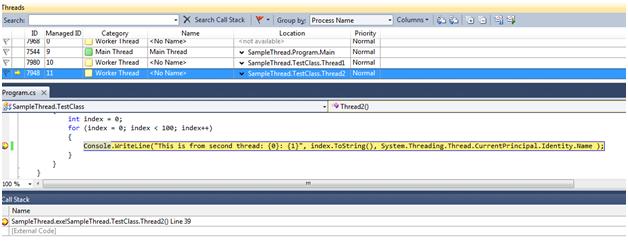
**16. What are Daemon threads and how can a thread is created as Daemon?**

Daemon thread's are threads run in background and stop automatically when nothing is running program. Example of a Daemon thread is "Garbage collector". Garbage collector runs until some .NET code is running or else it's idle. Thread can be made as Daemon by *Thread.Isbackground=true*

**17. How to debug the thread?**

Threading application can be debugged using **Debug->Windows->Threads** or **"Ctrl+Alt+H"**





**18. How we can use the same variable by multiple thread or Thread safe?**

In certain scenario, multiple threads must need to access the same variable at the same time; this will leads to some other problem. This can be avoided by using SynchLock. So until first thread released its variable, other thread will not be able to access the variable.

Example:

SyncLock (X)

'some operation with "X"

End SyncLock

**19. What are different states of a thread?**

Thread status can be known by using *ThreadName.ThreadState* property. *ThreadState* enumeration has all the values to detect a state of thread. Some sample states are Aborted, Running, Suspended, etc

Followings are the list of state of a thread.

* ThreadState.Aborted
* ThreadState.AbortRequested
* ThreadState.Background
* ThreadState.Running
* ThreadState.Stopped
* ThreadState.StopRequested
* ThreadState.Suspended
* ThreadState.SuspendRequested
* ThreadState.Unstarted
* ThreadState.WaitSleepJoin

**20. Can we use events with threading?**

Yes, we can use events with thread; this is one of the techniques to synchronize one thread with other.

**21. What is Event Wait Handle in threading?**

Event Wait Handle allows threads to communicate with each other by signaling and by waiting for signals. Event wait handles are wait handles that can be signaled in order to release one or more waiting threads.

**22. What is difference between Monitor object and EventWaitHandle?**

Both EventWaitHandles and Monitors are used to synchronize activities But Named event wait handles can be used to synchronize activities across application domains and processes, whereas monitors are local to an application domain.

**23. What is mean by ManualResetEvent and AutoResetEvent?**

Threads that call one of the wait methods of a synchronization event must wait until another thread signals the event by calling the Set method. There are two synchronization event classes. Threads set the status of ManualResetEvent instances to signaled using the Set method. Threads set the status of ManualResetEvent instances to no signaled using the Reset method or when control returns to a waiting WaitOne call. Instances of the AutoResetEvent class can also be set to signaled using Set, but they automatically return to nonsignaled as soon as a waiting thread is notified that the event became signaled.

**24. What is mean by Deadlock in Threading?**

Dead lock issue will be raised when multi thread try to access the same variable. Example when both the threads try to hold and monitor the variable at same time. Each thread monitor and wait for another thread to release the variable. Since no one is hold the variable and both the threads are waiting for the other thread to release each other, at last application hangs. This is called as Deadlock.