Multithreading

A Process

- A <u>Process</u> is a running application
- A Process is composed of <u>Threads</u>
  - e.g. a process may have:
    - A GUI thread
    - Several computational threads
    - A file I/O thread
    - A print thread

Multithreading

- Thread
  - A block of code that is the fundamental unit of execution to which processor allocates processor time
  - Threads run concurrently and share the cpu(s)
    - OS manages running threads with scheduling algorithms
    - Switches processor time between threads
    - Done so fast and efficiently that it appears all threads are running simultaneously
  - A .NET managed application begins as a single thread
    - Can spawn additional threads to partition its tasks
  - On multi-cpu system, applications can run faster since different threads can run on different processors

Concurrent Execution

- Threads run concurrently and asynchronously with respect to each other
  - Example: a GUI application that enters into a long computational loop
  - Running as a single thread:
    - While application's single thread is computing, messages on the message queue are ignored
      - So the application's user interface is frozen until computation finishes
  - Running as two threads:
    - Relegate the computational work to a background thread
    - Now the primary thread is free to service the message queue
      - App is now responsive to user input while computation is occurring

Multithreading Complexities

•Since threads run asynchronously, multithreaded applications are hard to write & debug

 –For example, write then read a data structure

   •If both are in a single thread, we know write will occur first

   •But if in separate threads, we don't know in advance when each thread is going to run

    –Read first then write ✍ old (wrong) data will be read

   •Threads need to be synchronized

Threads in .NET

•Threading classes are in namespace:
        <u>System.Threading</u>

 –Most important class: <u>Thread</u>

   •Represents a thread of execution

   •Implements properties and methods that allow programmer to launch and manipulate concurrently-running threads

Some Thread Class Properties

- bool   <u>IsBackground</u>
  - false (default) means thread runs in foreground
  - An application doesn't end until all its foreground threads have finished
    - Background threads are terminated by the CLR
- string   <u>Name</u>
  - Retrieve/change a thread's name
- Thread   <u>CurrentThread</u>
  - Static property returning a reference to the calling thread
  - Use result to get or change properties of currently-running thread
- <u>ThreadPriority</u> <u>Priority</u>
  - <u>ThreadPriority</u> is an enumeration:
    - Highest, AboveNormal, Normal (default), BelowNormal, Lowest
  - Determines relative amount of processor time allotted to the thread
  - Can be changed:
    - Thread myThread = Thread.CurrentThread;
    - myThread.Priority = ThreadPriority.AboveNormal;

Starting Threads

- Instantiate a <u>Thread</u> object
  - Give constructor a new "thread method"
    - This is the method the thread executes when it starts
    - Must be "wrapped" in the <u>ThreadStart</u> delegate
- Then use the thread's <u>Start( )</u> method
- Example:
  - Thread myThread = new Thread (new ThreadStart (myThreadMethod));
  - myThread.Start();
    - Starts the thread and causes myThreadMethod( ) to run
    - Your application must implement this method:
      - void myThreadMethod( ) { // code to run };
  - Thread is now "alive" and remains alive until it terminates
  - When the "thread method" returns, the thread ends

Threads-One & Threads-Two Example Programs

- Form has "Toggle Background Color" & "Start Computation" buttons and a label
  - First button handler toggles background between red and green
  - Second button handler starts a long, nested-loop computation
    - When computation is done, label control is turned blue and displays an "All Done" message
- Running as a single thread (as usual):
  - After "Start Computation" button is clicked
    - Program does not respond to "Toggle Background Color" button until computation is done (seems to be dead)
- Running in two threads:
  - Foreground thread starts a background thread to do the computation when user clicks the "Start Computation" button
  - Now the program responds to the "Change Background Color" button while the computation is being done

Suspending & Resuming Threads

- Suspend( ) method temporarily suspends a running thread
  - Any thread can call Suspend( ) on any other thread
- Resume( ) method starts it running again
  - If a thread suspends itself, some other thread must call Resume( ) on it to start it again
- Static method Sleep(int iMilliseconds)
  - Suspend current thread for a specified number of milliseconds
  - A thread can only call Sleep( ) on itself

Terminating a Thread

- Abort( ) method terminates a running thread

  myThread.Abort();   //terminates myThread

- Many times a thread should pause until the thread it is trying to abort terminates

  - Join( ) method does that

    otherThread.Abort( ); // ask the other thread to finish

    otherThread.Join( );   // "joins" the other thread

    - Pauses (sleeps) until other thread finishes

Other Thread Complexities

- Starting and stopping threads is easy

- Making them work cooperatively with shared data is not -- Thread synchronization is difficult

- One way of synchronizing threads:

  - Use Monitors (System.Threading.Monitor class)

    - Use "locks" so that only one thread can access data at a time

      - Monitor.Enter(obj) static method acquires a lock – Thread can then manipulate the object's data

        » All other threads are blocked from acquiring the lock and accessing the data

        » If another thread has executed Enter(), our thread blocks

      - Monitor.Exit(obj) static method releases the lock

        » Blocked thread can now acquire the lock and manipulate the data

  - See Chapter 15 of the Deitel text book for details

Starting Processes on the System

•A <u>Process</u> component provides access to processes that are running or can run on the system

–In <u>System.Diagnostics</u> namespace

•To run a process:

–Instantiate a new <u>Process</u> object

–Set its <u>StartInfo.FileName</u> property to the name of the executable file

–Invoke its <u>Start( )</u> method

```
Process myProc = new Process( );
myProc.StartInfo.FileName = "c:\\Windows\\Notepad.exe";
myProc.Start( );
```

•StartProcess example program

–Allows user to start any application program on the system