**1. Thread pool**

public class TaskQueue

{

private List<Thread> threads;

private Queue<Action> tasks;

public TaskQueue(int threadCount)

{

tasks = new Queue<Action>();

threads = new List<Thread>();

for (int i = 0; i < threadCount; i++)

{

var t = new Thread(DoThreadWork);

threads.Add(t);

t.IsBackground = true;

t.Start();

}

}

public void EnqueueTask(Action task)

{

lock (tasks)

{

tasks.Enqueue(task);

Monitor.Pulse(tasks);

}

}

private Action DequeueTask()

{

lock (tasks)

{

while (tasks.Count == 0)

Monitor.Wait(tasks);

return tasks.Dequeue();

}

}

private void DoThreadWork()

{

while (true)

{

Action task = DequeueTask();

try

{

task();

}

catch (ThreadAbortException)

{

Thread.ResetAbort();

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

**2 Mutex**

public class Mutex

{

private Thread thread;

public void Lock()

{

Thread t = Thread.CurrentThread;

while (Interlocked.CompareExchange(ref thread, t, null) != null)

Thread.Yield();

Thread.MemoryBarrier();

}

public void Unlock()

{

Thread t = Thread.CurrentThread;

if (Interlocked.CompareExchange(ref thread, null, t) != t)

throw new SynchronizationLockException();

Thread.MemoryBarrier();

}

}

**3 Action runner**

public class ActionRunner

{

int runningCount;

object sync = new object();

public void RunAndWaitAll(Action[] actions)

{

runningCount = actions.Length;

foreach (Action action in actions)

ThreadPool.QueueUserWorkItem(ExecuteAction, action);

lock (sync)

if (runningCount > 0)

Monitor.Wait(sync);

}

private void ExecuteAction(object state)

{

var action = (Action)state;

action();

lock (sync)

{

runningCount--;

if (runningCount == 0)

Monitor.Pulse(sync);

}

}

}

**4 Native Buffer**

public class NativeBuffer : Object, IDisposable

{

private IntPtr handle;

private bool disposed;

public NativeBuffer(int size)

{

handle = Marshal.AllocHGlobal(size);

}

~NativeBuffer()

{

Dispose(false);

}

public IntPtr Handle

{

get

{

if (!disposed)

return handle;

else

throw new ObjectDisposedException(ToString());

}

}

public void Dispose()

{

if (!disposed)

{

Dispose(true);

GC.SuppressFinalize(this);

disposed = true;

}

}

protected virtual void Dispose(bool disposing)

{

if (handle != IntPtr.Zero)

Marshal.FreeHGlobal(handle);

}

}

**5 Assembly**

private static void ListTypesInAssembly(Assembly assembly)

{

var types = assembly.GetTypes().Where(t => t.IsPublic).ToList();

types.Sort();

foreach (var type in types)

{

Console.WriteLine(type.FullName);

}

}

public static void Main(string[] args)

{

var assemblyPath = Assembly.GetExecutingAssembly().Location;

var assembly = Assembly.LoadFrom(assemblyPath);

ListTypesInAssembly(assembly)

Console.ReadKey();

}