1. Form Memory:



* 1. Giới thiệu:
     1. Form thực hiện quản lý bộ nhớ: lấy các thông tin về trạng thái bộ nhớ và hiệu suất
  2. Các API sử dụng:

[StructLayout(LayoutKind.Sequential)]

public struct PERFORMANCE\_INFORMATION

{

public uint cb;

public UIntPtr CommitTotal;

public UIntPtr CommitLimit;

public UIntPtr CommitPeak;

public UIntPtr PhysicalTotal;

public UIntPtr PhysicalAvailable;

public UIntPtr SystemCache;

public UIntPtr KernelTotal;

public UIntPtr KernelPaged;

public UIntPtr KernelNonpaged;

public UIntPtr PageSize;

public uint HandleCount;

public uint ProcessCount;

public uint ThreadCount;

}

[StructLayout(LayoutKind.Sequential)]

internal struct MEMORYSTATUSEX

{

internal uint dwLength;

internal uint dwMemoryLoad;

internal ulong ullTotalPhys;

internal ulong ullAvailPhys;

internal ulong ullTotalPageFile;

internal ulong ullAvailPageFile;

internal ulong ullTotalVirtual;

internal ulong ullAvailVirtual;

internal ulong ullAvailExtendedVirtual;

}

class API

{

// Hàm lấy thông tin trạng thái Memory

[return: MarshalAs(UnmanagedType.Bool)]

[DllImport("Kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern bool GlobalMemoryStatusEx(ref MEMORYSTATUSEX lpBuffer);

// Hàm lấy thông tin hiệu suất

[DllImport("psapi.dll", SetLastError = true)]

public static extern bool GetPerformanceInfo(out PERFORMANCE\_INFORMATION pPerformanceInformation, uint cb);

}

* 1. Chi tiết sử dụng:
     1. Khi Load form và timer chạy : Lấy thông tin cần thiết, gọi hàm GetMemoryInfo()

private void GetMemoryInfo()

{

// Lấy thông tin trạng thái Memory

MEMORYSTATUSEX statEX = new MEMORYSTATUSEX();

statEX.dwLength = (uint)Marshal.SizeOf(typeof(MEMORYSTATUSEX));

lbxInfo.Items.Clear();

if (API.GlobalMemoryStatusEx(ref statEX))

{

pbRAM.Value = (int)statEX.dwMemoryLoad;

lbRAM.Text = statEX.dwMemoryLoad + " %"; // 0-100

lbxInfo.Items.Add("Tổng kích thước (Vật lý): " +

statEX.ullTotalPhys + " bytes");

lbxInfo.Items.Add("Kích thước (Vật lý) có sẵn: " +

statEX.ullAvailPhys + " bytes");

lbxInfo.Items.Add("Kích thước (Ảo) đã sử dụng: " +

(statEX.ullTotalPageFile - statEX.ullTotalPhys) + " bytes");

lbxInfo.Items.Add("Kích thước (Ảo) khả dụng: " +

(statEX.ullAvailPageFile - statEX.ullAvailPhys) + " bytes");

}

// Hàm lấy thông tin hiệu suất

PERFORMANCE\_INFORMATION perfInfo = new PERFORMANCE\_INFORMATION();

uint input\_size = (uint)System.Runtime.InteropServices.Marshal.SizeOf(

typeof(PERFORMANCE\_INFORMATION));

if (API.GetPerformanceInfo(out perfInfo, input\_size))

{

lbxInfo.Items.Add("----------------------------------------------");

lbxInfo.Items.Add("Số trang tối đa hiện tại: " +

perfInfo.CommitLimit.ToUInt64() + " pages");

lbxInfo.Items.Add("Số trang tối đa đồng thời: " +

perfInfo.CommitPeak.ToUInt64() + " pages");

lbxInfo.Items.Add("Kích thước 1 trang: " +

perfInfo.PageSize.ToUInt64() + " bytes");

lbxInfo.Items.Add("Số handle: " + perfInfo.HandleCount);

lbxInfo.Items.Add("Số process: " + perfInfo.ProcessCount);

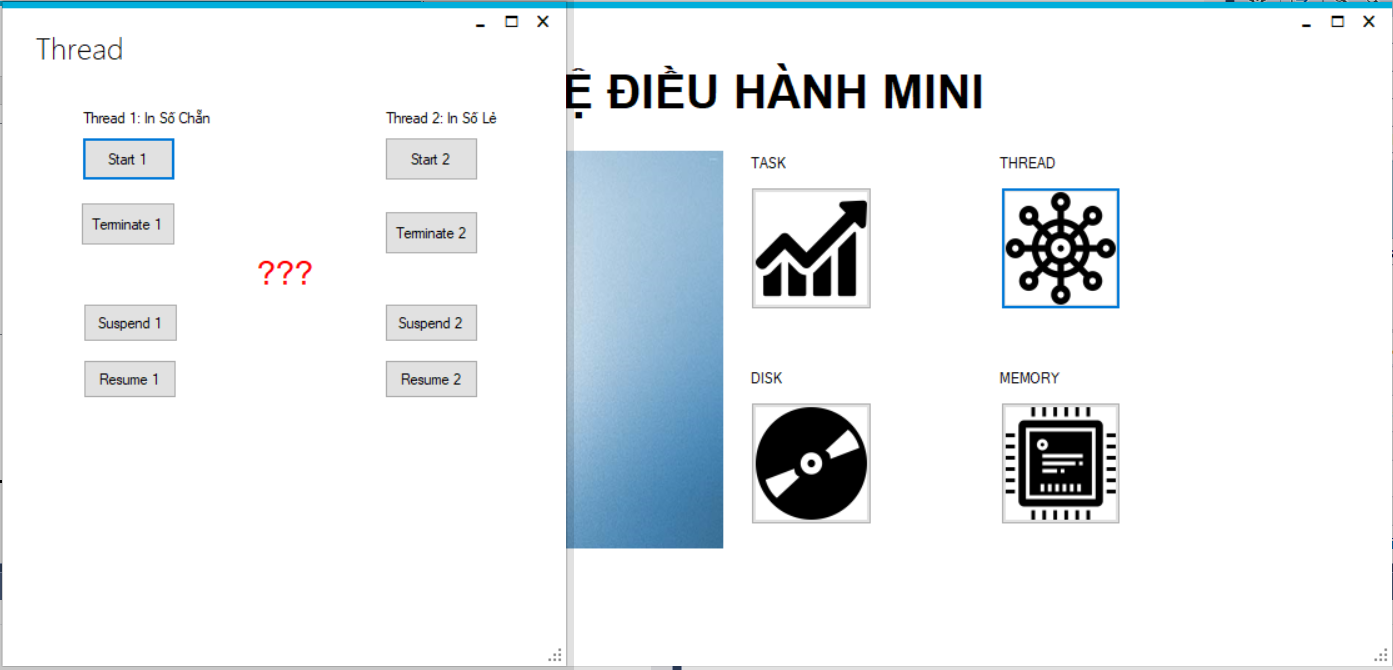
lbxInfo.Items.Add("Số thread: " + perfInfo.HandleCount);

}

}

* Giải thích:
  + Gọi hàm API.GlobalMemoryStatusEx(ref statEX) của thư viện Kernel32.dll thành công để lưu các thông số trạng thái memory vào biến statEX kiểu struct MEMORYSTATUSEX và từ biến statEX gọi ra các thông tin để hiển thị bao gồm: dwMemoryLoad, ullTotalPhys, ullAvailPhys, ullTotalPageFile, ullAvailPageFile,...
  + Gọi hàm A API.GetPerformanceInfo(out perfInfo, input\_size) của thư viện psapi.dll thành công để lưu các thông số hiệu suất vào biến perfInfo kiểu struct PERFORMANCE\_INFORMATION và từ biến perfInfo gọi ra các thông tin để hiển thị bao gồm: CommitLimit, CommitPeak, PageSize, HandleCount, ProcessCount, ThreadCount,...

1. Form Thread:



* 1. Giới thiệu:
     1. Form thực hiện quản lý luồng: mô phỏng các luồng chạy song song trong ứng dụng (Luồng main và 2 luồng con Thread 1, Thread 2).
     2. Các luồng này sử dụng chung tài nguyên là lable “???”, Thread 1 để in các số chẵn 1 lần / s, Thread 2 để in các số lẻ 2 lần / s.
     3. Các thao tác: Start (tạo), Terminate (hủy), Suspend(dừng), Resume(tiếp tục) 1 Thread
  2. Các API sử dụng:

class API

{

// Hàm hiển thị thông báo : type : 0 (OK) / 1 (OK - Cancel)

[DllImport("user32.dll", EntryPoint = "MessageBox")]

public static extern int ShowMessage(int hWnd, string text, string caption, uint type);

// Hàm tạo thread --> handle

[DllImport("kernel32")]

public static extern IntPtr CreateThread(

IntPtr lpThreadAttributes,

UInt32 dwStackSize,

ThreadStart lpStartAddress,

IntPtr param,

UInt32 dwCreationFlags,

UInt32 lpThreadId

);

// Hàm dừng thread bằng handle

[DllImport("kernel32.dll", SetLastError = true)]

public static extern int SuspendThread(IntPtr hThread);

// Hàm tiếp tục thread bằng handle

[DllImport("kernel32.dll", SetLastError = true)]

public static extern int ResumeThread(IntPtr hThread);

// Hàm hủy Thread bằng handle

[DllImport("Kernel32.dll", CharSet = CharSet.Auto)]

public static extern bool TerminateThread(IntPtr hThread, uint dwExitCode);

}

* 1. Chi tiết sử dụng:
     1. Khi nhấn Start1



IntPtr th1;

int dem1 = 0;

IntPtr th2;

int dem2 = 1;

private void btnStart1\_Click(object sender, EventArgs e)

{

dem1 = 0;

ThreadStart ThreadFunc = new ThreadStart(Action1);

th1 = API.CreateThread(IntPtr.Zero, 0, ThreadFunc, IntPtr.Zero, 0, 0);

}

private void Action1()

{

while (true)

{

ChangeText(label1, dem1.ToString());

dem1 += 2;

Thread.Sleep(1000);

}

}

* Giải thích: Gọi hàm API.CreateThread(IntPtr.Zero, 0, ThreadFunc, IntPtr.Zero, 0, 0); tham số truyền vào là 1 ThreadStart ThreadFunc chạy Action1 (in số chẵn) trả về 1 Thread để thực hiện các thao tác điêù khiển bên dưới
  + 1. Khi nhấn Suspend1

if (API.SuspendThread(th1) == -1)

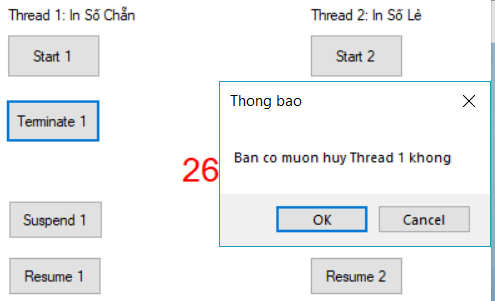
API.ShowMessage(0, "Loi khong ton tai Thread / Thread dang dung!", "Thong bao", 0);

* + 1. Khi nhấn Resume1

if (API.ResumeThread(th1) == 0)

API.ShowMessage(0, "Loi khong ton tai Thread / Thread dang chay!", "Thong bao", 0);

* + 1. Khi nhấn Terminate1



int result = API.ShowMessage(0, "Ban co muon huy Thread 1 khong", "Thong bao", 1);

if(result == 1 )

if(!API.TerminateThread(th1, 1))

API.ShowMessage(0, "Loi khong ton tai Thread", "Thong bao", 0);

* + 1. Tương tự đối với Thread 2
    2. Khi đóng form:

try

{

API.TerminateThread(th1, 1);

API.TerminateThread(th2, 1);

}

catch(Exception ex)

{

API.ShowMessage(0, ex.Message, "Thong bao", 0);

}

* Giải thích: Tránh đóng form mà các thread vẫn chạy nên phải đóng các thread th1 và th2
  + 1. Lưu ý:

// Fix cross thread ( Dùng chung 1 tài nguyên )

private void ChangeText(Label lb, string s)

{

if (lb.InvokeRequired)

{

lb.BeginInvoke(new MethodInvoker(delegate () { ChangeText(lb, s); }));

}

else

{

lock (lb)

{

lb.Text = s;

}

}

}