**Phần mở đầu**

Cùng với sự phát triển của kinh tế, kỹ thuật, y tế, giáo dục, máy tính cũng ngày càng được sử dụng phổ biến trong mọi lĩnh vực và đời sống để nhằm đáp ứng nhu cầu sử dụng của con người.

Một máy tính để có thể hoạt động được đều phải dựa trên một hệ diều hành nhất định. Có rất nhiều hệ điều hành được biết đến nhưng hệ điều hành được đa số mọi người biết tới và được sử dụng phổ biến đó là hệ điều hành Windows được phát triển bởi hãng Microsoft. Hệ điều hành là một phần mềm nhằm giúp cho con người giao tiếp với máy tính một cách đơn giản và hiệu quả bằng các thao tác trực tiếp với phần cứng của máy tính. Các phần mềm ứng dụng khác chạy trên Windows không thao tác trực tiếp với phần cứng, mà luôn phải thao tác trung gian qua hệ điều hành. Ví dụ khi gõ phím, hệ điều hành sẽ là nơi trung gian để nhận tín hiệu từ bán phím và xử lý xem đó là phím gì rồi xác định xem phím đó được gõ trên ứng dụng nào và gửi tín hiệu đến ứng dụng đó để nó xử lý thông qua các hàm được viết trong ứng dụng…

Windows API là một giao diện lập trình nằm trên nền windows, cung cấp các thao tác trực tiếp với hệ điều hành và phần cứng máy tính. Các ứng dụng windows sẽ thông qua Windows API để thao tác với máy tính.

Bởi vậy mà nhóm em đã quyết định chọn đề tài “Hệ điều hành mini trên nền tảng C# Winform và ứng dụng các hàm API Windows”, trong đó sử dụng các hàm API để giao tiếp trực tiếp với một số chức năng cơ bản được tích hợp sẵn trên máy như: Quản lý tiến trình(Tạo tiến trình mới, lấy thông số time process, xóa tiến trình đang hoạt động), quản lý bộ nhớ ngoài( Mở thư mục/ tệp, tạo thư mục, xóa thư mục), quản lý bộ nhớ( lấy thông tin về trạng thái bộ nhớ và hiệu suất), quản lý luồng(tạo, dừng, tiếp tục với từng thread).

Trong quá trình thực hiện đề tài tuy có nhiều cố gắng nhưng không thể tránh được sự sai sót, nhóm em mong sẽ nhận được những nhận xét của thầy để nhóm hoàn thành đề tài tốt hơn.

Chúng em xin chân thành cảm ơn!

1. Form Main



* 1. Giới thiệu:
     1. Form hiển thị giao diện ứng dụng
     2. Chức năng điều hướng tới 4 form tương ứng với 4 chức năng quản lý của hệ điều hành:
        1. Task (Tiến trình),
        2. Thread (Luồng),
        3. Disk (Bộ nhớ ngoài),
        4. Memory (Bộ nhớ trong).

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

{

Disk t = new Disk();

t.Show();

}

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

{

Task t = new Task();

t.Show();

}

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

{

Memory t = new Memory();

t.Show();

}

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

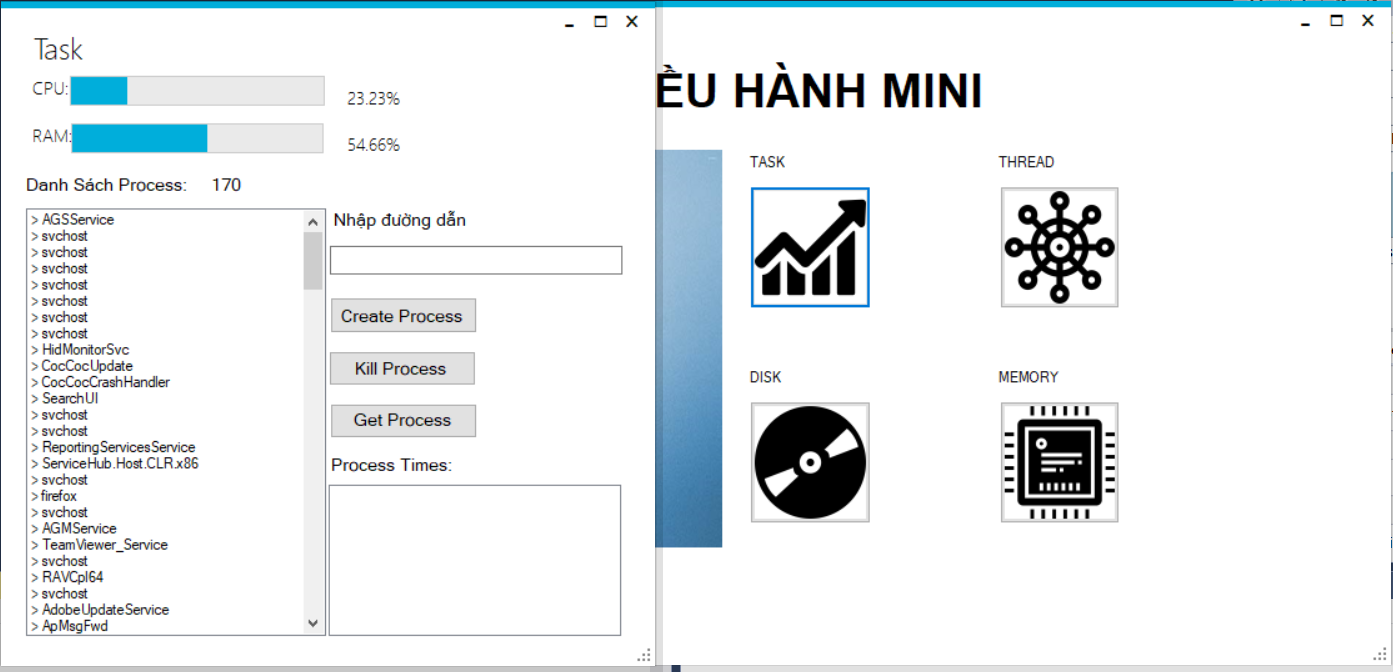
{

ThreadM t = new ThreadM();

t.Show();

}

1. Form Task



* 1. Giới thiệu:
     1. Form thực hiện quản lý tiến trình: đưa ra các tiến trình đang hoạt động, tạo tiến trình mới, lấy thông tin tiến trình đang hoạt động và xóa tiến trình đang hoạt động bất kỳ.
     2. Các thao tác: Create Process (tạo tiến trình mới), Get Process (lấy thông số time process), Kill Process (xóa tiến trình đang hoạt động).
  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;

}

public struct PROCESS\_INFORMATION

{

public IntPtr hProcess;

public IntPtr hThread;

public uint dwProcessId;

public uint dwThreadId;

}

public struct STARTUPINFO

{

public uint cb;

public string lpReserved;

public string lpDesktop;

public string lpTitle;

public uint dwX;

public uint dwY;

public uint dwXSize;

public uint dwYSize;

public uint dwXCountChars;

public uint dwYCountChars;

public uint dwFillAttribute;

public uint dwFlags;

public short wShowWindow;

public short cbReserved2;

public IntPtr lpReserved2;

public IntPtr hStdInput;

public IntPtr hStdOutput;

public IntPtr hStdError;

}

public struct SECURITY\_ATTRIBUTES

{

public int length;

public IntPtr lpSecurityDescriptor;

public bool bInheritHandle;

}

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 process theo đường dẫn tới ứng dụng

[DllImport("kernel32.dll")]

public static extern bool CreateProcess(

string lpApplicationName,

string lpCommandLine,

IntPtr lpProcessAttributes,

IntPtr lpThreadAttributes,

bool bInheritHandles,

uint dwCreationFlags,

IntPtr lpEnvironment,

string lpCurrentDirectory,

ref STARTUPINFO lpStartupInfo,

out PROCESS\_INFORMATION lpProcessInformation

);

// Hàm lấy thông số process (Time) theo process handle

[DllImport("Kernel32",

EntryPoint = "GetProcessTimes",

ExactSpelling = true,

CharSet = CharSet.Ansi,

SetLastError = true)]

public static extern bool GetProcessTimes(

IntPtr hProcess,

ref long lpCreationTime,

ref long lpExitTime,

ref long lpKernelTime,

ref long lpUserTime

);

// Hàm hủy Process

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

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool TerminateProcess(IntPtr hProcess, uint uExitCode);

// Hàm lấy thông số -- > có số lượng process

[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 GetProcesses()

Process[] procs;

private void GetProcesses()

{

procs = Process.GetProcesses(); //Trả về tất cả các tiến

trình đang chạy trên máy tính

if (Convert.ToInt32(lbProcess.Text) != procs.Length)

{

lbxProcess.Items.Clear();

for (int i = 0; i < procs.Length; i++)

{

lbxProcess.Items.Add("> " + procs[i].ProcessName); // Tạo danh

sách tương tự task manager

}

}

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))

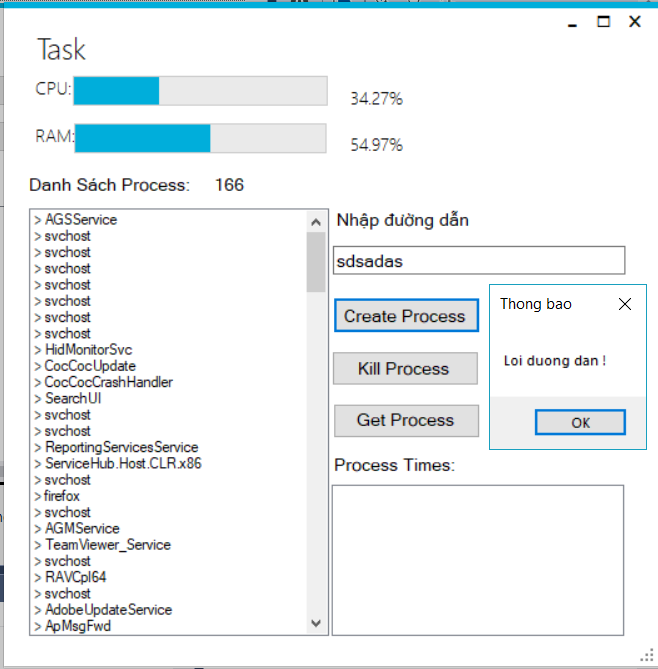
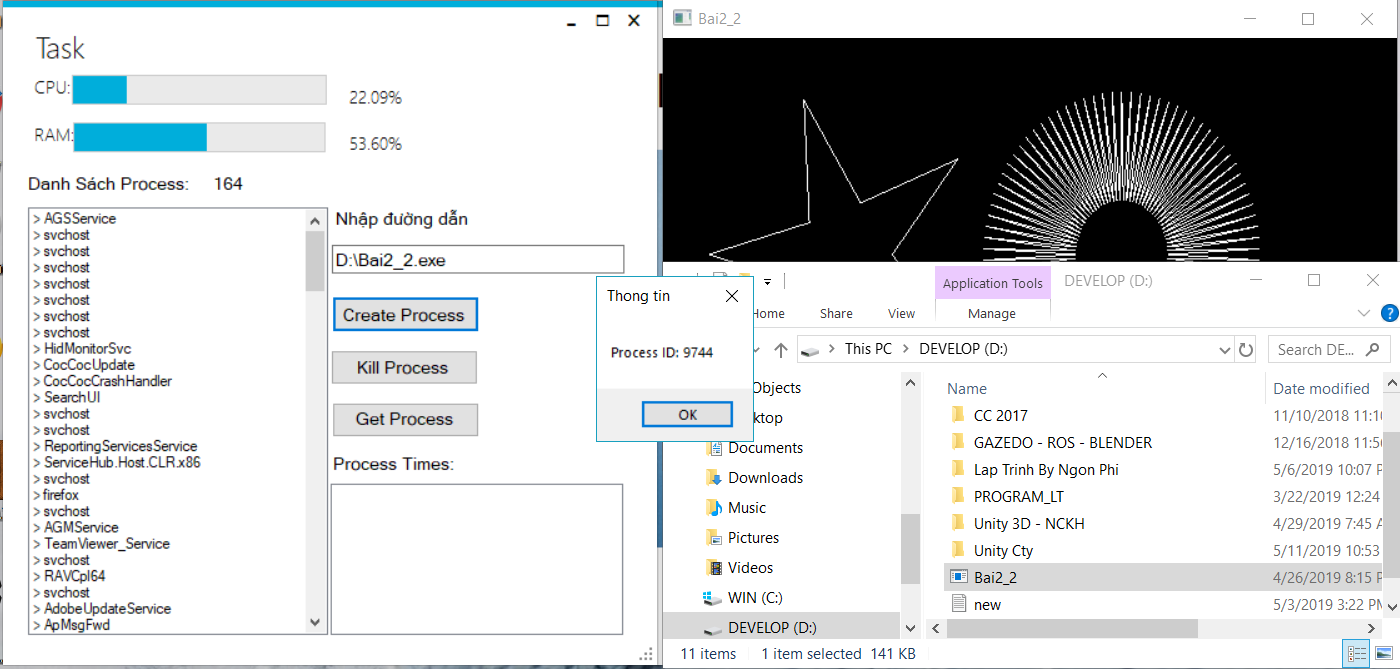
{

lbProcess.Text = perfInfo.ProcessCount.ToString();

}

}

* Giải thích:
  + Sử dụng lớp Process trong C# lấy các tiến trình đang chạy trên máy tính để hiển thị thông tin và sử dụng các thao tác sau khi chọn 1 process.
  + Gọi hàm 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 lấy ra thông tin ProcessCount (đếm số process).
    1. Khi nhập đường dẫn tới process và nhấn Create Process



string path = tbPath.Text;

STARTUPINFO si = new STARTUPINFO();

PROCESS\_INFORMATION pi = new PROCESS\_INFORMATION();

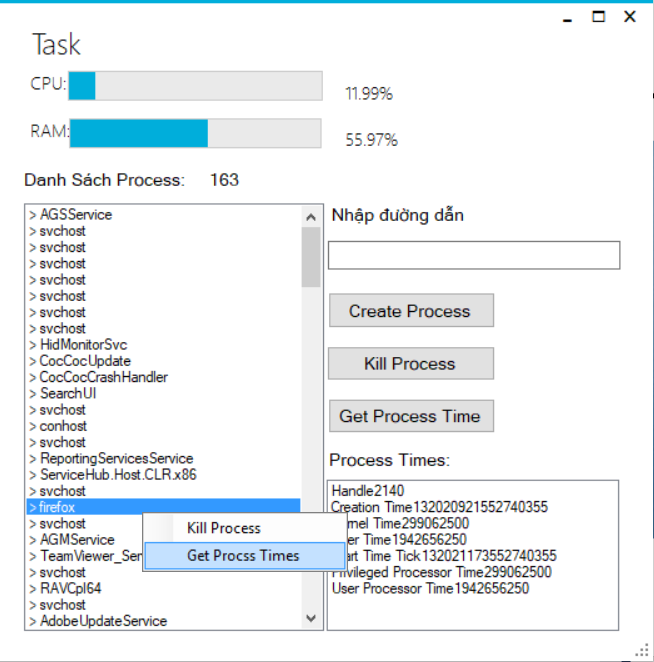
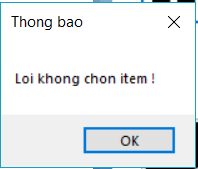
if (!API.CreateProcess(@path, null, IntPtr.Zero, IntPtr.Zero,

false, 0, IntPtr.Zero, null, ref si, out pi))

API.ShowMessage(0, "Loi duong dan !", "Thong bao", 0);

API.ShowMessage(0, "Process ID: " + pi.dwProcessId, "Thong tin", 0);

* Giải thích:
  + Gọi hàm API.CreateProcess(@path, null, IntPtr.Zero, IntPtr.Zero, false, 0, IntPtr.Zero, null, ref si, out pi) của thư viện kernel32 để tạo 1 process theo đường dẫn tuyệt đối @path truyền vào khi nhập.
  + Nếu thành công các thông tin được lưu vào biến si kiểu struct STARTUPINFO , pi kiểu struct PROCESS\_INFORMATION và từ biến pi lấy ra thông tin dwProcessId (Id của process) hiển thị thông báo thành công. Ngoài ra si, pi có thể lấy ra các thông tin khác như: lpTitle, dwFlags, hStdInput, hStdOutput, hProcess, hThread, dwThreadId ,...
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi duong dan !", "Thong bao", 0) của thư viện user32 để thông báo lỗi đường dẫn sai (type : 0 --> hiển thị thông báo).
    1. Khi lựa chọn 1 process và ấn Get Process Times

GetProcessTimes(lbxProcess.SelectedIndex);

private void GetProcessTimes(int index)

{

try

{

if (index < 0 || index >= procs.Length) //TH khi không chọn item

{

API.ShowMessage(0, "Loi khong chon item !", "Thong bao", 0);

return;

}

else

{

lbInformation.Items.Clear();

long lpCreationTime = 0;

long lpExitTime = 0;

long lpKernelTime = 0;

long lpUserTime = 0;

Process P = procs[index];

bool RetVal = API.GetProcessTimes(P.Handle, ref lpCreationTime,

ref lpExitTime, ref lpKernelTime, ref lpUserTime);

lbInformation.Items.Add("Handle" +

P.Handle.ToString() + " ");

lbInformation.Items.Add("Creation Time" +

lpCreationTime.ToString() + " ");

lbInformation.Items.Add("Kernel Time" +

lpKernelTime.ToString() + " ");

lbInformation.Items.Add("User Time" +

lpUserTime);

lbInformation.Items.Add("Start Time Tick" + (P.StartTime.Ticks –

DateTime.Parse("1/1/1601").Ticks).ToString() + " ");

lbInformation.Items.Add("Privileged Processor Time" +

P.PrivilegedProcessorTime.Ticks.ToString() + " ");

lbInformation.Items.Add("User Processor Time" +

P.UserProcessorTime.Ticks.ToString() + " ");

}

}

catch (Exception ex)

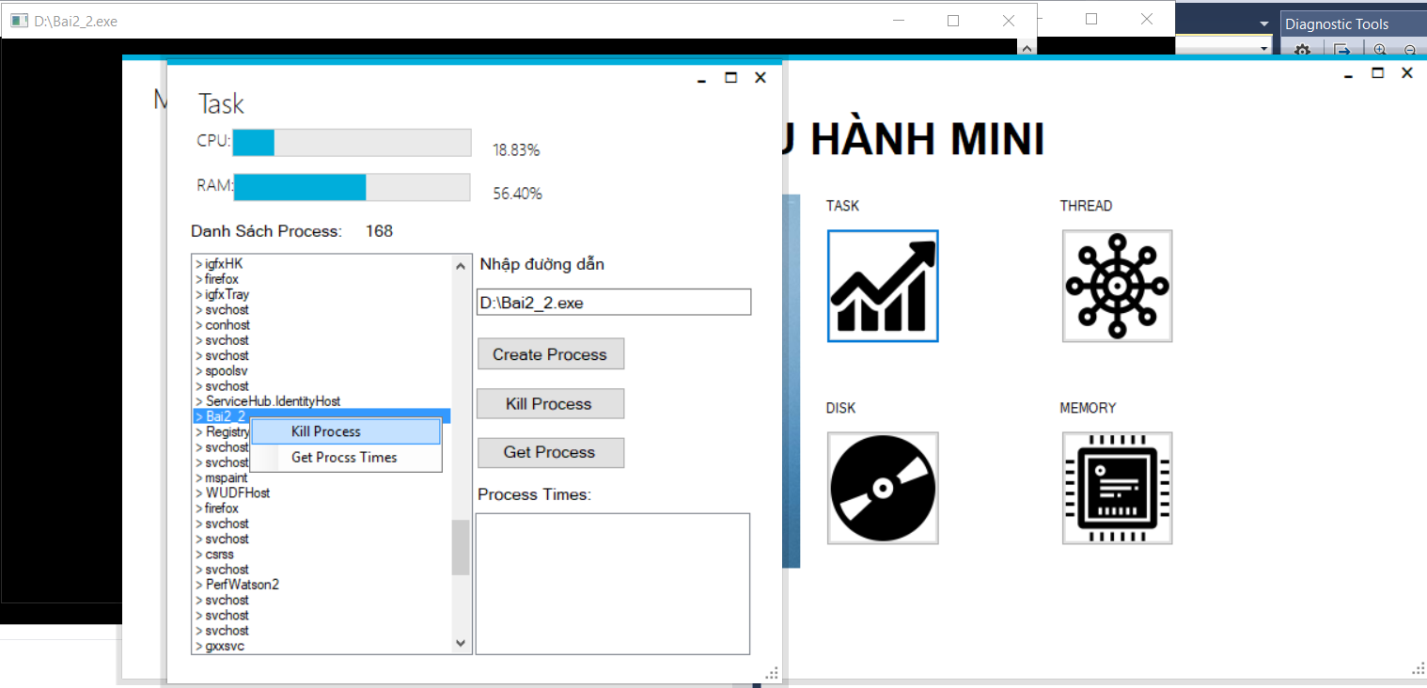
{

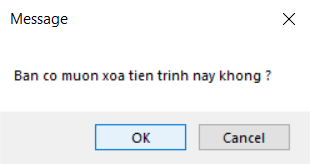
API.ShowMessage(0, "Loi: " + ex.Message, "Thong bao", 0);

}

}

* Giải thích:
  + Khi không chọn 1 process gọi hàm API.ShowMessage(0, "Loi khong chon item !", "Thong bao", 0) của thư viện user32 để thông báo lỗi không chọn item (type : 0 --> hiển thị thông báo).
  + Khi gọi hàm API.GetProcessTimes(P.Handle, ref lpCreationTime, ref lpExitTime, ref lpKernelTime, ref lpUserTime) của thư viện kernel32 truyền vào Handle của process thành công thông tin được lưu lại và hiển thị thông qua các biến: lpCreationTime, lpExitTime, lpKernelTime,...
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi: " + ex.Message, "Thong bao", 0) của thư viện user32 để thông báo lỗi (type : 0 --> hiển thị thông báo).
    1. Khi lựa chọn 1 process và ấn Kill Process





KillProcess(lbxProcess.SelectedIndex);

private void KillProcess(int index)

{

if (index < 0 || index >= procs.Length) //TH khi không chọn item

{

API.ShowMessage(0, "Loi khong chon item !", "Thong bao", 0);

return;

}

else

{

int result = API.ShowMessage(0, "Ban co muon xoa tien trinh nay khong ?",

"Message", 1);

if (result == 1)

{

try

{

API.TerminateProcess(procs[index].Handle, 1);

}

catch (Exception ex)

{

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

}

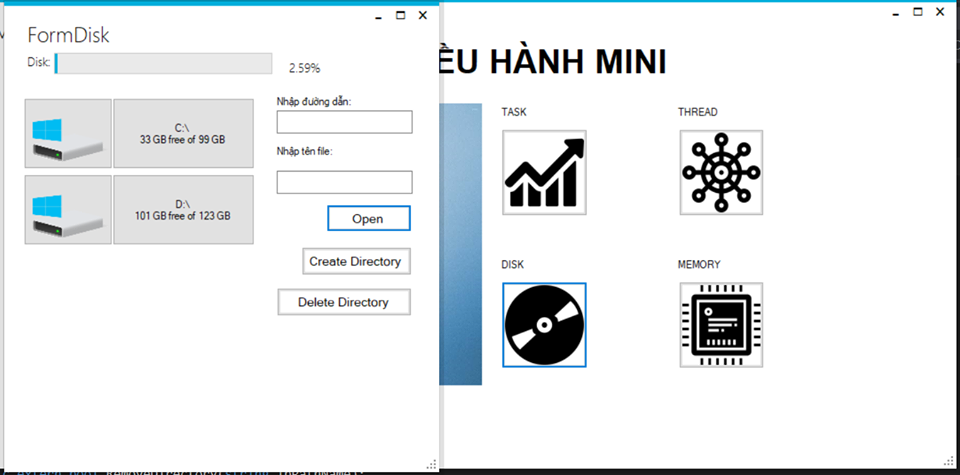
}

}

}

* Giải thích:
  + Tương tự khi không chọn 1 process
  + Hiển thị thông báo bằng cách gọi hàm int result = API.ShowMessage(0, "Ban co muon xoa tien trinh nay khong ?", "Message", 1) (type: 1 --> hiển thị lựa chọn OK / Cancel ).
  + Khi nhấn Kill Process và chọn OK gọi hàm TerminateProcess(procs[index].Handle, 1) sẽ thực hiện và xóa process theo handle của process[index] được truyền vào và mã ExitCode = 1.

1. Form Disk



* 1. Giới thiệu:
     1. Form thực hiện quản lý bộ nhớ ngoài (ổ đĩa): đưa ra danh sách các phân vùng ổ đĩa cùng thông tin chi tiết.
     2. Các thao tác Open(mở thư mục/ tệp), Create Directory(Tạo thư mục), Delete Directory(Xóa thư mục).
  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 lấy danh sách tên phân vùng ổ đĩa

[DllImport("kernel32.dll")]

public static extern uint GetLogicalDriveStrings(uint nBufferLength,

[Out] char[] lpBuffer);

// Hàm lấy thông số phân vùng ổ đĩa theo tên

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

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool GetDiskFreeSpaceEx(string lpDirectoryName,

out ulong lpFreeBytesAvailable,

out ulong lpTotalNumberOfBytes,

out ulong lpTotalNumberOfFreeBytes);

// Hàm tạo thư mục theo đường dẫn

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

public static extern bool CreateDirectory(string lpPathName,

IntPtr lpSecurityAttributes);

// Hàm xóa thư mục theo đường dẫn

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

public static extern bool RemoveDirectory(string lpPathName);

// Hàm mở file / thư mục / link web

[DllImport("Shell32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern bool ShellExecute(IntPtr hwnd, string lpOperation,

string lpFile, string lpParameters, string lpDirectory, int nShowCmd);

}

* 1. Chi tiết sử dụng:
     1. Khi Load form: Lấy thông tin các phân vùng ổ đĩa

private void Disk\_Load(object sender, EventArgs e)

{

const int size = 512;

char[] buffer = new char[size];

uint code = API.GetLogicalDriveStrings(size, buffer);

if (code == 0)

{

API.ShowMessage(0, "Loi lay danh sach ten phan vung o dia !", "Thong bao", 0);

return;

}

StringCollection list = new StringCollection();

int start = 0;

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

{Nếu

if (buffer[i] == 0)

{

string s = new string(buffer, start, i - start);

list.Add(s);

start = i + 1;

}

}

int t = 1;

Image bkg = Image.FromFile(@"C:\hard-drive-disk-icon.png");

foreach (string s in list)

{

ulong FreeBytesAvailable;

ulong TotalNumberOfBytes;

ulong TotalNumberOfFreeBytes;

if (!API.GetDiskFreeSpaceEx(s, out FreeBytesAvailable,

out TotalNumberOfBytes, out TotalNumberOfFreeBytes))

throw new System.ComponentModel.Win32Exception();

Button btnx = new Button();

btnx.BackgroundImage = bkg;

btnx.Margin = new System.Windows.Forms.Padding(0, 3, 0, 3);

btnx.BackgroundImageLayout = System.Windows.Forms.ImageLayout.Stretch;

btnx.Name = "btnx" + t;

btnx.Size = new System.Drawing.Size(100, 80);

pnlDisk.Controls.Add(btnx);

Button btn = new Button();

btn.Name = "btn" + t;

btn.Size = new System.Drawing.Size(160, 80);

btn.Margin = new System.Windows.Forms.Padding(0, 3, 20, 3);

btn.Text = s + "\n" + TotalNumberOfFreeBytes / (1024 \* 1024 \* 1024) + " GB free of " + TotalNumberOfBytes / (1024 \* 1024 \* 1024) + " GB\r\n";

btn.Click += btn\_Click;

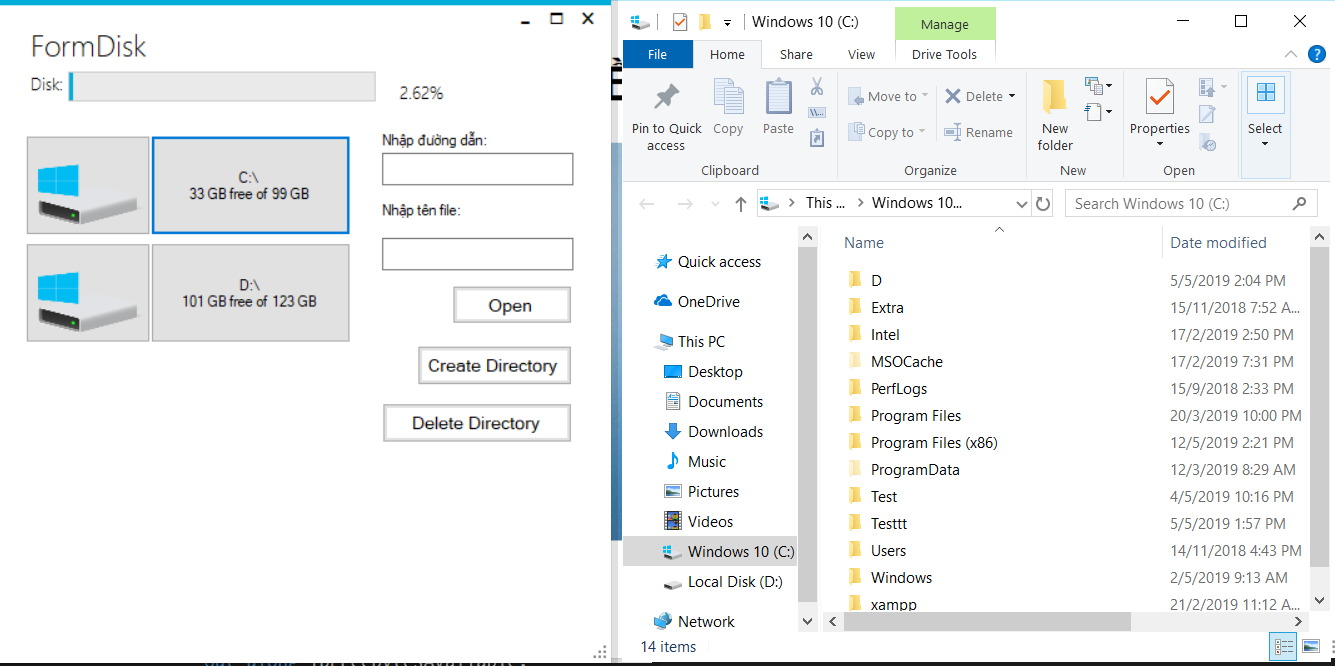
pnlDisk.Controls.Add(btn);

t++;

}

}

* Giải thích:
  + Gọi hàm API GetLogicalDriveStrings(size, buffer) của thư viện kernel32.dll thành công để lấy ra danh sách tên phân vùng ổ đĩa
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi lay danh sach ten phan vung o dia !", "Thong bao", 0) của thư viện user32 để thông báo lỗi không lấy được danh sách phân vùng ổ đĩa;
  + Chuyển mảng buffer thành mảng string rồi gọi hàm API.GetDiskFreeSpaceEx(s, out FreeBytesAvailable, out TotalNumberOfBytes, out TotalNumberOfFreeBytes) của thư viện kernel32 để lấy thông tin phân vùng theo tên, thông tin gồm: TotalNumberOfFreeBytes, TotalNumberOfBytes, .. nếu sai (trả về false) ném vào ngoại lệ throw new System.ComponentModel.Win32Exception().
    1. Khi nhấn vào các Nút (phân vùng ổ đĩa): mở đến đường dẫn tới các phân vùng tương ứng



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

{

Button btn = sender as Button;

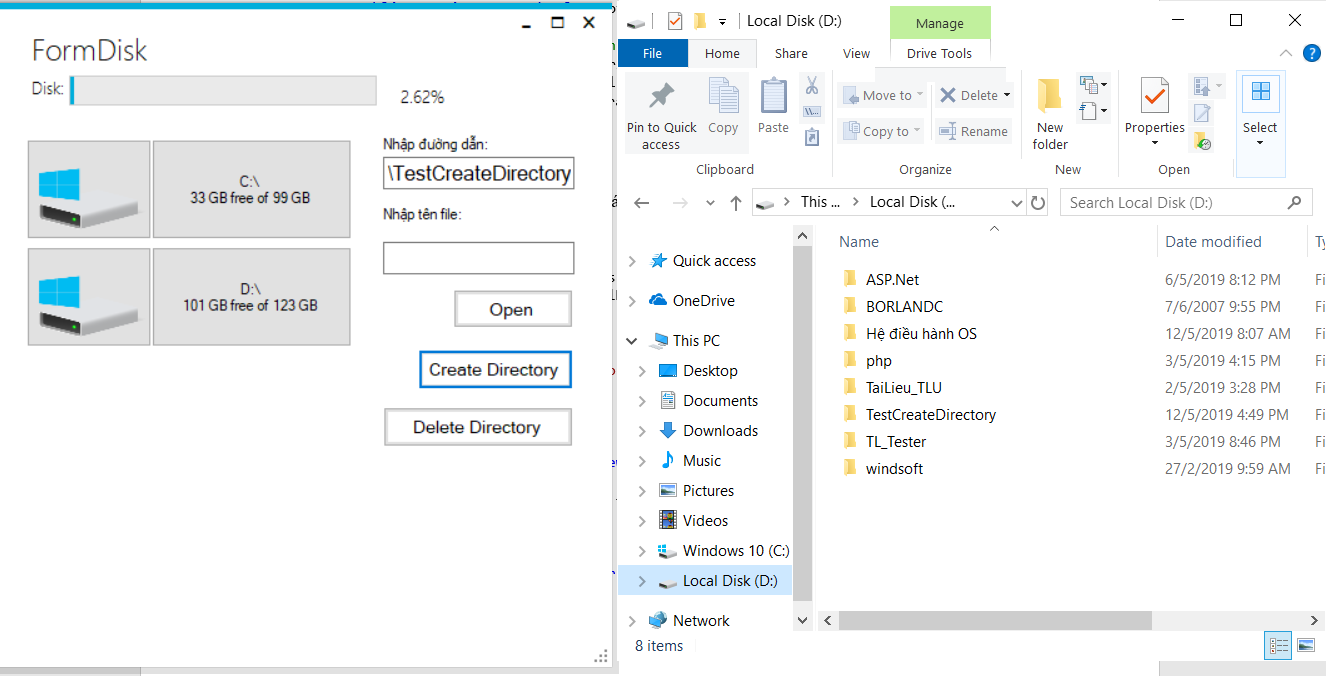
string path = btn.Text.Substring(0, 3);

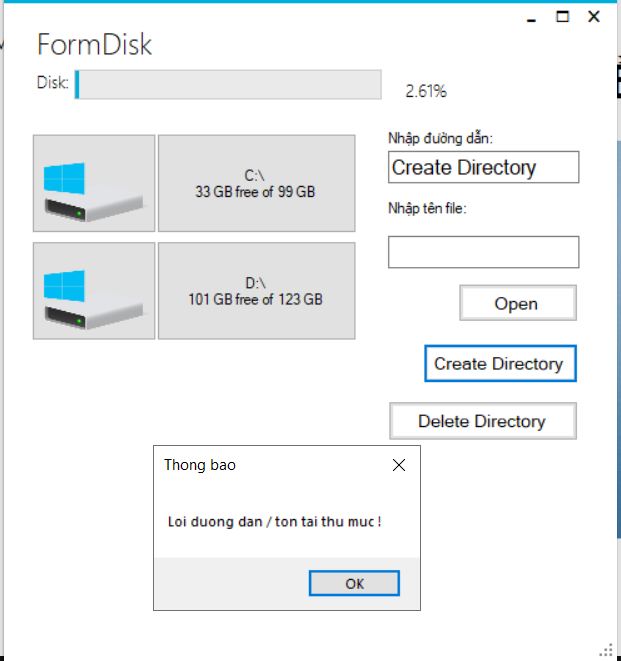
if (!API.ShellExecute(IntPtr.Zero, "open", null, null, @path, 1))

API.ShowMessage(0, "Loi duong dan !", "Thong bao", 0);

}

* Giải thích:
  + Gọi hàm API ShellExecute(IntPtr.Zero, "open", null, null, @path, 1) của thư viện Shell32.dll thành công để mở file/ thư mục/ link web.
  + Nếu thành công sẽ mở ra phân vùng ổ đĩa theo @path đường dẫn tới phân vùng tương ứng
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi duong dan !", "Thong bao", 0) của thư viện user32 để thông báo lỗi đường dẫn sai (type : 0 --> hiển thị thông báo).
    1. Khi nhập đường dẫn và nhấn Create Directoty: Tạo thư mục mới tại đường dẫn chỉ định





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

{

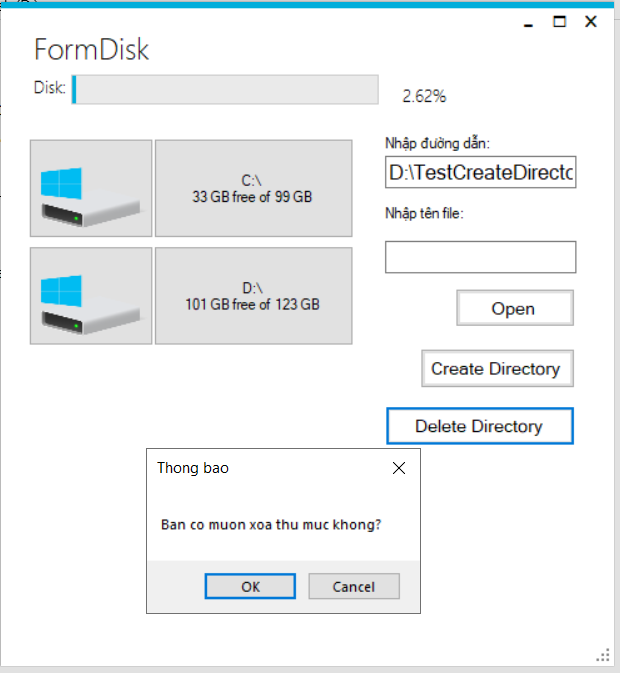
string path = @txbDD.Text;

if (!API.CreateDirectory(@"\\?\" + path, IntPtr.Zero))

API.ShowMessage(0, "Loi duong dan / ton tai thu muc !", "Thong bao", 0);

}

* Giải thích:
  + Gọi hàm API.CreateDirectory(@"\\?\" + path, IntPtr.Zero) của thư viện kernel32 để tạo 1 thư mục theo đường dẫn tuyệt đối truyền vào khi nhập.
  + Nếu thành công thư mục sẽ được tạo ra tại đường dẫn path
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi duong dan / ton tai thu muc !", "Thong bao", 0) của thư viện user32 để thông báo lỗi đường dẫn hoặc đường dẫn đã tồn tại.
    1. Khi nhập đường dẫn và nhấn Delete Directoty: Xóa thư mục tại đường dẫn chỉ định



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

{

string path = @txbDD.Text;

int rs = API.ShowMessage(0, "Ban co muon xoa thu muc khong?", "Thong bao", 1);

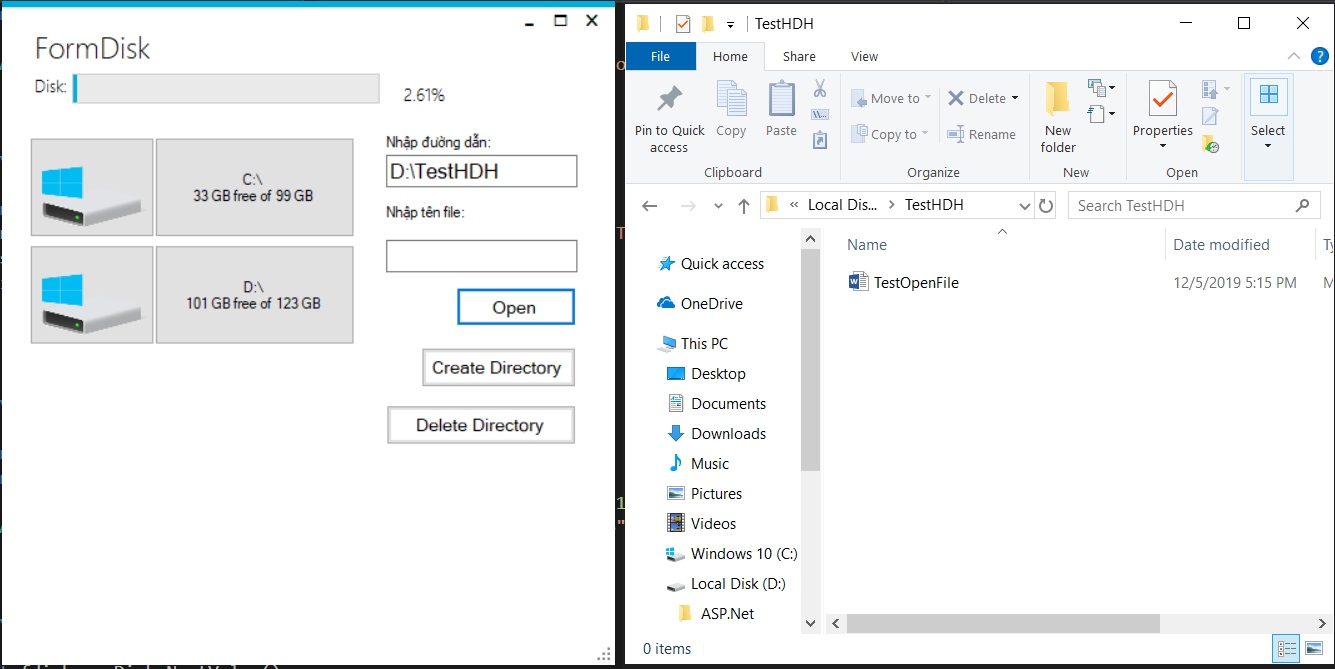
if(rs == 1)

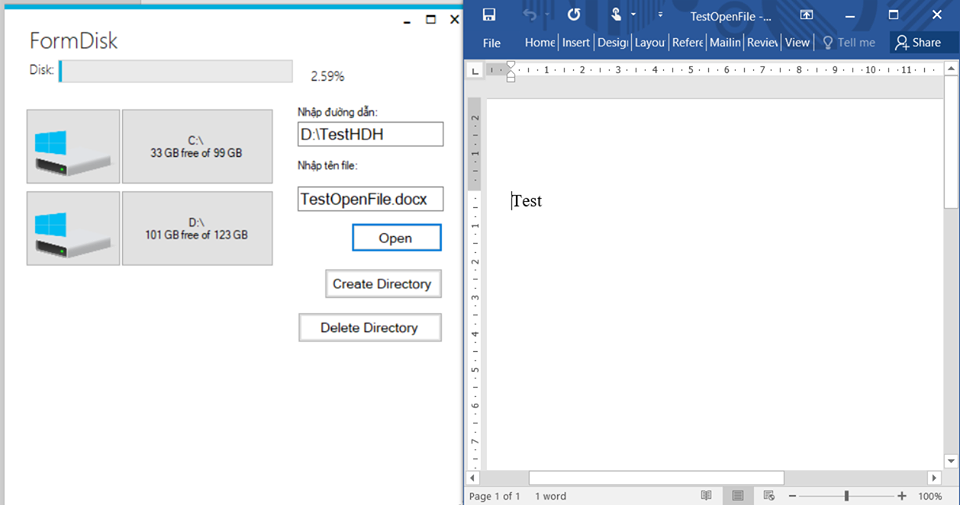
if (!API.RemoveDirectory(@"\\?\" + path))

API.ShowMessage(0, "Loi duong dan / thu muc ton tai!", "Thong bao",

}

* Giải thích:
  + Nhập vào đường dẫn thư mục cần xóa và chọn Delete Directory sẽ hiển thị thông báo gọi hàm int rs = API.ShowMessage(0, "Ban co muon xoa thu muc khong?", "Thong bao", 1) (type: 1 --> hiển thị lựa chọn OK / Cancel) của thư viện user32, nếu chọn OK thì thư mục sẽ được xóa khỏi phân vùng ổ đĩa bằng hàm API.RemoveDirectory(@"\\?\" + path) của thư viện kernel32 đúng theo đường dẫn path chỉ định
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi duong dan / ton tai thu muc !", "Thong bao", 0) của thư viện user32 để thông báo lỗi đường dẫn hoặc đường dẫn đã tồn tại.
    1. Khi nhập đường dẫn, tên file (có đuôi) và nhấn Open : Mở file tại đường dẫn chỉ định bằng các ứng dụng mặc định





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

{

string path = @txbDD.Text;

string nameF = txbFile.Text;

if (!API.ShellExecute(IntPtr.Zero, "open", nameF, null, @path, 1))

API.ShowMessage(0, "Loi duong dan / ten file !", "Thong bao", 0);

}

* Giải thích:
  + Gọi hàm API.ShellExecute(IntPtr.Zero, "open", nameF, null, @path, 1) của thư viện Shell32 để mở thư mục/file theo đường dẫn tuyệt đối truyền vào path và tên file (có đuôi) nameF tương ứng.
  + Nếu thành công sẽ mở ra thư mục/ file theo đúng đường dẫn và file bằng ứng dụng mặc định.
  + Nếu thất bại gọi hàm API.ShowMessage(0, "Loi duong dan / ten file !", "Thong bao", 0) của thư viện user32 để thông báo lỗi đường dẫn hoặc tên file.

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(

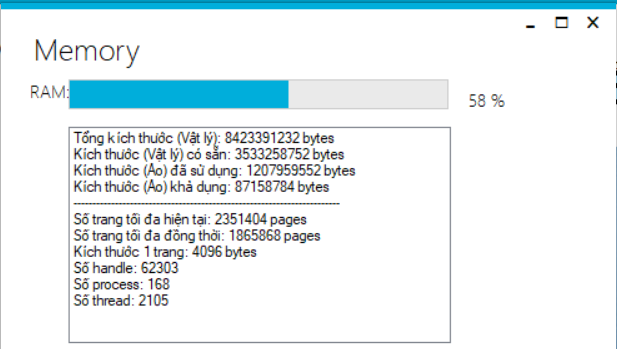
out PERFORMANCE\_INFORMATION pPerformanceInformation,

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);

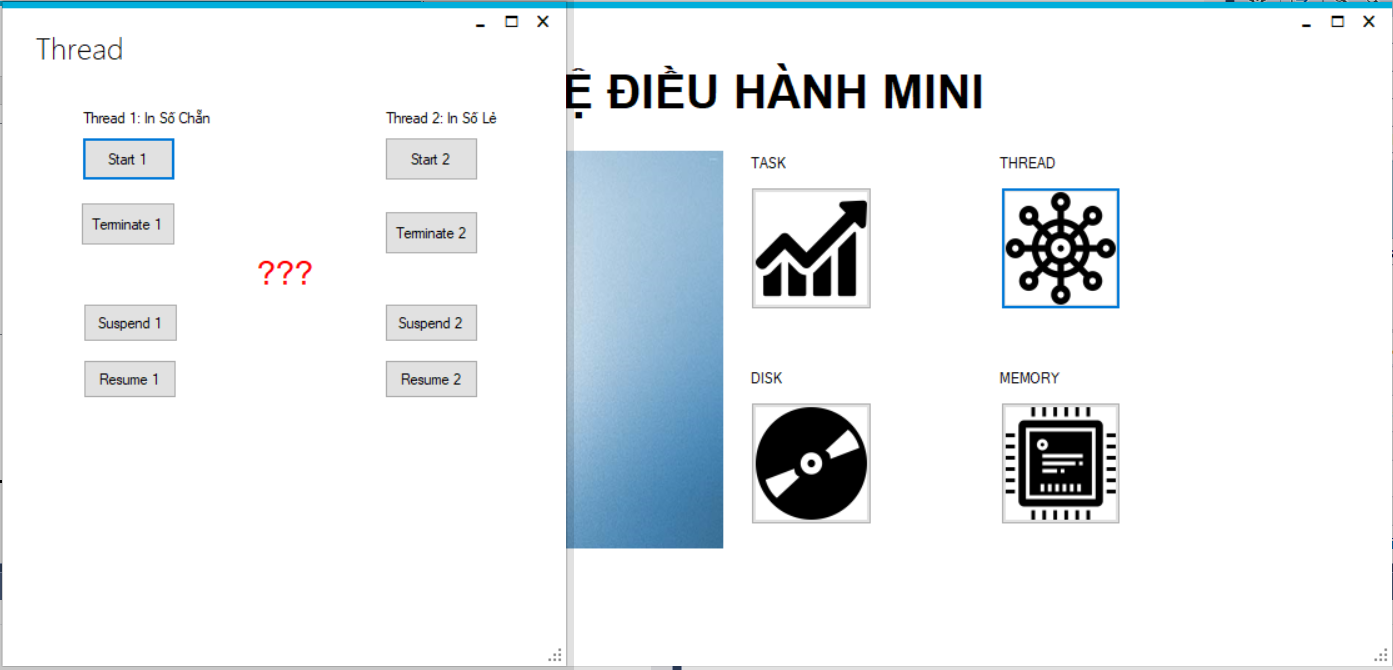
lbxInfo.Items.Add("Số thread: " + perfInfo.HandleCount);

}

}

* 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 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) đối với từng Thread th1, th2
  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.dll")]

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) của thư viện kernel32 tham số truyền vào là 1 ThreadStart ThreadFunc chạy Action1 (in số chẵn) trả về 1 con trỏ tới Thread để thực hiện các thao tác điêù khiển bên dưới.
  + 1. Khi nhấn Suspend1

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

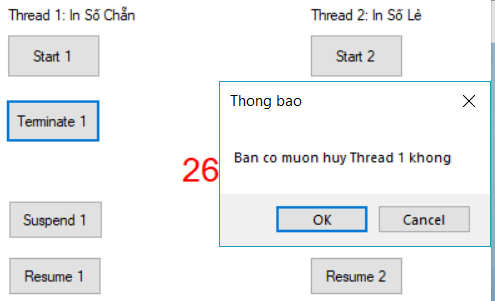
API.ShowMessage(0, "Loi khong ton tai Thread / Thread dang dung!", "Thong bao", 0);

* Giải thích: Gọi hàm API.SuspendThread(th1) của thư viện kernel32 tham số truyền vào là con trỏ Thread th1 để dừng Thread th1 đang chạy, nếu lỗi (trả về -1) thì gọi hàm API.ShowMessage(0, "Loi khong ton tai Thread / Thread dang dung!", "Thong bao", 0) của thư viện user32 để in ra thông báo (type:0 in thông báo).
  + 1. Khi nhấn Resume1

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

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

* Giải thích : Gọi hàm API. ResumeThread (th1) của thư viện kernel32 tham số truyền vào là con trỏ Thread th1 để tiếp tục Thread th1 đang dừng, nếu lỗi (trả về 0) thì gọi hàm API.ShowMessage(0, "Loi khong ton tai Thread / Thread dang chay!", "Thong bao", 0) của thư viện user32 để in ra thông báo (type:0 in thông báo).
  + 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);

* Giải thích: Hiển thị lựa chọn OK / Cancel bằng hàm API.ShowMessage(0, "Ban co muon huy Thread 1 khong", "Thong bao", 1) của thư viện user32 (type: 1 lựa chọn ) nếu OK thì tiến hành hủy Thread, Gọi hàm API.TerminateThread(th1, 1) của thư viện kernel32 tham số truyền là con trỏ Thread th1 và mã ExitCode = 1 để hủy Thread th1 xảy ra lỗi khi trả về false và hiển thị thông báo lỗi
  + 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 bằng hàm API.TerminateThread(th1, 1) và API.TerminateThread(th2, 1).
  + 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;

}

}

}

1. Bảng phân công công việc:

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và Tên** | **Công việc** |
| 1 | Đặng Minh Hiếu |  |
| 2 | Nguyễn Xuân Phi |  |
| 3 | Phạm Ngọc Quỳnh |  |
| 4 | Mai Xuân Thành |  |
| 5 | Nguyễn Thị Thanh Thư |  |

1. Tài liệu tham khảo

[1] <http://dhthuyloi.blogspot.com/2014/02/tai-lieu-he-ieu-hanh.html>

[2] <http://www.pinvoke.net/index.aspx>

[3] <https://www.codeproject.com>

[4] <https://stackoverflow.com>

[5] <https://docs.microsoft.com/en-us/windows/desktop/api/>