**Win32 API (далі WinAPI) - це набір функцій (API - Application Programming Interface), що працюють під управлінням ОС Windows. Вони містяться в бібліотеці windows.h.**

**Типи даних:**

Дескриптор - це ідентифікатор будь-якого об'єкта. Для різних типів об'єктів існують різні дескриптори. Дескриптор об'єкта можна описати так:

HANDLE h;

Є також дескриптори кисті, курсору миші, шрифту і т.д. З їх допомогою ми можемо при ініціалізації або в процесі роботи додатка поміняти якісь настройки, чого, наприклад, ми не могли зробити в командному додатку. Використовуються вони в описових функціях, які керують типу: CreateProcess (), ShowWindow () і т.д. або як значення, що повертаються деякими функціями.

// Отримує дескриптор для пристрою введення або виведення:

HANDLE h = GetStdHandle (DWORD nStdHandle);

Список типів дескрипторів:

HANDLE - дескриптор об'єкта.

HBITMAP - дескриптор растрового зображення. Від фрази handle bitmap.

HBRUSH - дескриптор кисті. Від фрази handle brush.

HCURSOR - дескриптор курсора. Від фрази handle cursor.

HDC - дескриптор контексту пристрою. Від фрази handle device context.

HFONT - дескриптор шрифту. Від фрази handle font.

HICONS - дескриптор криптограми. Від фрази handle icons.

HINSTANCE - дескриптор екземпляра додатка. Від фрази handle instance.

HMENU - дескриптор меню. Від фрази handle menu.

HPEN - дескриптор пера. Від фрази handle pen.

HWND - дескриптор вікна. Від фрази handle window.

**Додаткові рядкові типи даних**

LPCSTR - покажчик на константну рядок, що закінчується нуль-термінатором. Від фрази long pointer constant string.

LPCTSTR - покажчик на константну рядок, без UNICODE. Від фрази long pointer constant TCHAR string. Це надбудова функції LPCSTR.

LPCWSTR - покажчик на константну UNICODE рядок. Від фрази фрази long pointer constant wide character string. Це надбудова функції LPCSTR.

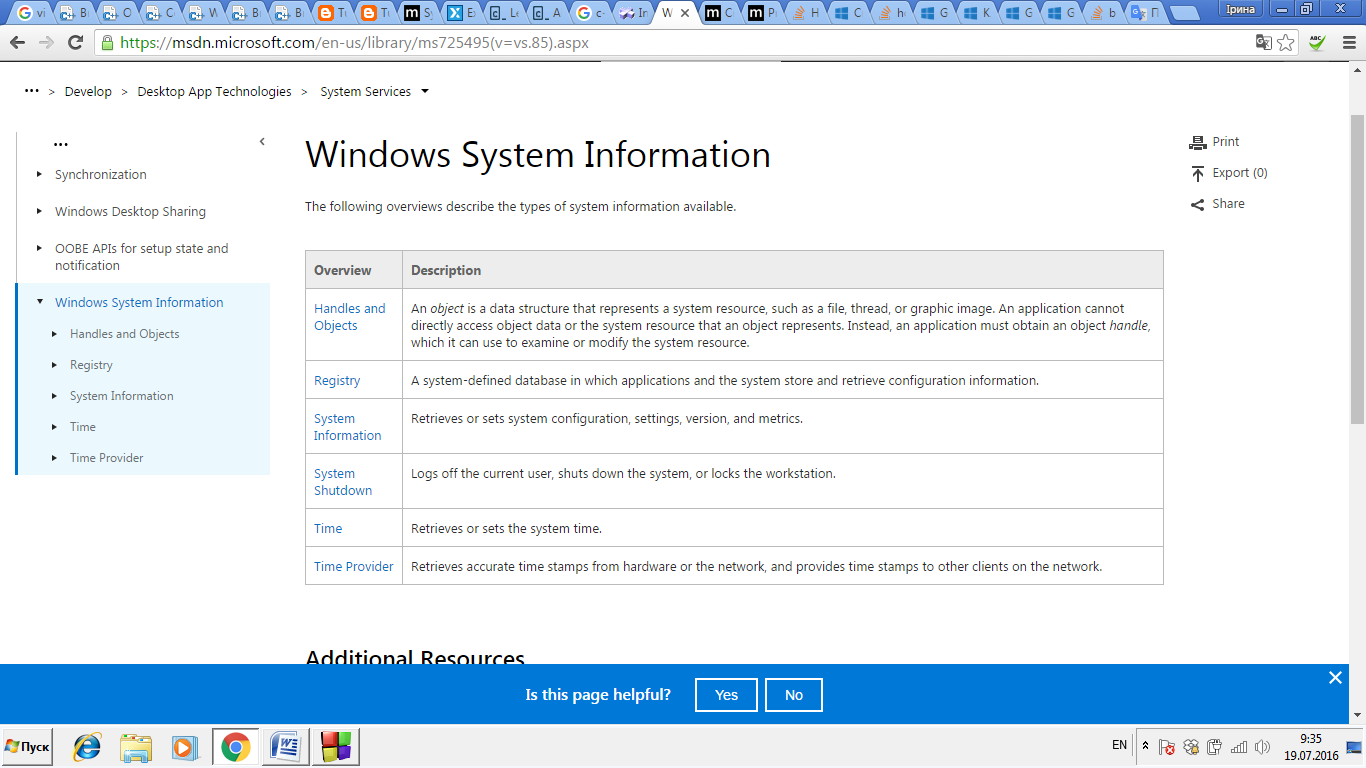
LPSTR - покажчик на рядок, що закінчується нуль-термінатором. Від фрази long pointer string.

LPTSTR - покажчик на рядок, без UNICODE. Від фрази long pointer TCHAR string. Це надбудова функції LPSTR.

LPWSTR - покажчик на UNICODE рядок. Від фрази long pointer wide character string. Це надбудова функції LPSTR.

TCHAR - символьний тип - аналог char і wchar\_t.

На сайті документації від Майкрософт на вкладці systemServices можна побачити функції для роботи з інформацієї про систему.



# Для читання інформації про систему можна скористатись функцією GetSystemInfo.

# Опис функції:

void WINAPI GetSystemInfo(

\_Out\_ LPSYSTEM\_INFO lpSystemInfo

);

# *Приклад з описом деяких функцій:*

#include <windows.h>

#include <stdio.h>

//#pragma comment(lib, "user32.lib")

int main()

{

SYSTEM\_INFO siSysInfo;

// копіюємо інформацію про «залізо» в SYSTEM\_INFO структуру.

GetSystemInfo(&siSysInfo);

// показуємо вміст структури SYSTEM\_INFO.

printf("Hardware information: \n");

printf(" OEM ID: %u\n", siSysInfo.dwOemId);

printf(" Number of processors: %u\n",

siSysInfo.dwNumberOfProcessors);

printf(" Page size: %u\n", siSysInfo.dwPageSize);

printf(" Processor type: %u\n", siSysInfo.dwProcessorType);

printf(" Minimum application address: %lx\n",

siSysInfo.lpMinimumApplicationAddress);

printf(" Maximum application address: %lx\n",

siSysInfo.lpMaximumApplicationAddress);

printf(" Active processor mask: %u\n",

siSysInfo.dwActiveProcessorMask);

}

**Запуск іншої exe програми**

Для цього можна скористатись наступними функціями:

ShellExecute

ShellExecuteEx

CreateProcess

CreateProcessAsUser

CreateProcessWithLogonW

system

\_exec

\_spawn, \_wspawn

**ShellExecute**

Ця універсальна функція Windows API, ймовірно, кращий вибір, якщо ви хочете запустити програму. Вона дозволяє контролювати важливі аспекти роботи програми, наприклад, вікно і його нормальний розмір або зведений до мінімуму, або навіть приховане при запуску. Ви можете передати будь-яку кількість параметрів - імен файлів, перемикачів і опцій - до програми, і ви можете вибрати (робочий) каталог за замовчуванням для програми.

Опис:

 HINSTANCE **ShellExecute**(   
           HWND     *hwnd*,    
           LPCTSTR *lpOperation*,     // Usually "open" or "print" etc.  
           LPCTSTR *lpFile*,              // program or document filename    
           LPCTSTR *lpParameters*,  // if program, command-line parameters   
           LPCTSTR *lpDirectory*,      // default directory   
           INT *nShowCmd* );          // usually SW\_SHOWNORMAL

Приклад:

ShellExecute( NULL, "open", "calc.exe", NULL, NULL, SW\_SHOWNORMAL );

Або з друком:

ShellExecute( NULL, "print", // verb: print

"c:\\temp\\UserGuide.PDF", // document to launch

NULL, // parms -- not used when launching a document"

NULL, // default dir (don't care here)

SW\_SHOWNORMAL

);

**CreateProcess**

Для більшості сценаріїв, запускається за допомогою параметрів, які в основному схожі на те, що ми бачили з ShellExecute (за винятком того, що немає ніякої операції дієслова - CreateProcess може тільки запускати програми):

PROCESS\_INFORMATION ePI={0};

STARTUPINFO rSI={0};

rSI.cb= sizeof( rSI );

rSI.dwFlags= STARTF\_USESHOWWINDOW;

rSI.wShowWindow= SW\_SHOWNORMAL; // or SW\_HIDE or SW\_MINIMIZED

BOOL fRet= CreateProcess(

"c:\\windows\\notepad.exe", // program name

" c:\\temp\\report.txt", // ...and parameters

NULL, NULL, // security stuff (use defaults)

TRUE, // inherit handles (not important here)

0, // don't need to set priority or other flags

NULL, // use default Environment vars

NULL, // don't set current directory

&rSI, // where we set up the ShowWIndow setting

&ePI // gets populated with handle info

);

***Приклад***:

#include <Windows.h>

VOID startup(LPCTSTR lpApplicationName)

{

// additional information

STARTUPINFO si;

PROCESS\_INFORMATION pi;

// set the size of the structures

ZeroMemory( &si, sizeof(si) );

si.cb = sizeof(si);

ZeroMemory( &pi, sizeof(pi) );

// start the program up

CreateProcess( lpApplicationName, // the path

argv[1], // Command line

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi ) // Pointer to PROCESS\_INFORMATION structure

);

// Close process and thread handles.

CloseHandle( pi.hProcess );

CloseHandle( pi.hThread );

}

**Зупинити процес:**

Можна скористатись функцією TerminateProcess

Приклад, як зупинити процес за іменем ехе файлу (у прикладі написана функція користувача(

#include <windows.h>

#include <process.h>

#include <Tlhelp32.h>

#include <winbase.h>

#include <string.h>

void killProcessByName(const char \*filename)

{

HANDLE hSnapShot = CreateToolhelp32Snapshot(TH32CS\_SNAPALL, NULL);

PROCESSENTRY32 pEntry;

pEntry.dwSize = sizeof (pEntry);

BOOL hRes = Process32First(hSnapShot, &pEntry);

while (hRes)

{

if (strcmp(pEntry.szExeFile, filename) == 0)

{

HANDLE hProcess = OpenProcess(PROCESS\_TERMINATE, 0,

(DWORD) pEntry.th32ProcessID);

if (hProcess != NULL)

{

TerminateProcess(hProcess, 9);

CloseHandle(hProcess);

}

}

hRes = Process32Next(hSnapShot, &pEntry);

}

CloseHandle(hSnapShot);

}

int main()

{

killProcessByName("notepad++.exe");

return 0;

}

# Функція CreateToolhelp32Snapshot

Моментальний знімок зазначених процесів, модулі і потоків, які використовуються цими процесами.

#include <windows.h>

#include <tlhelp32.h>

#include <tchar.h>

// Forward declarations:

BOOL GetProcessList( );

BOOL ListProcessModules( DWORD dwPID );

BOOL ListProcessThreads( DWORD dwOwnerPID );

void printError( TCHAR\* msg );

int main( void )

{

GetProcessList( );

return 0;

}

BOOL GetProcessList( )

{

HANDLE hProcessSnap;

HANDLE hProcess;

PROCESSENTRY32 pe32;

DWORD dwPriorityClass;

// Take a snapshot of all processes in the system.

hProcessSnap = CreateToolhelp32Snapshot( TH32CS\_SNAPPROCESS, 0 );

if( hProcessSnap == INVALID\_HANDLE\_VALUE )

{

printError( TEXT("CreateToolhelp32Snapshot (of processes)") );

return( FALSE );

}

// Set the size of the structure before using it.

pe32.dwSize = sizeof( PROCESSENTRY32 );

// Retrieve information about the first process,

// and exit if unsuccessful

if( !Process32First( hProcessSnap, &pe32 ) )

{

printError( TEXT("Process32First") ); // show cause of failure

CloseHandle( hProcessSnap ); // clean the snapshot object

return( FALSE );

}

// Now walk the snapshot of processes, and

// display information about each process in turn

do

{

\_tprintf( TEXT("\n\n=====================================================" ));

\_tprintf( TEXT("\nPROCESS NAME: %s"), pe32.szExeFile );

\_tprintf( TEXT("\n-------------------------------------------------------" ));

// Retrieve the priority class.

dwPriorityClass = 0;

hProcess = OpenProcess( PROCESS\_ALL\_ACCESS, FALSE, pe32.th32ProcessID );

if( hProcess == NULL )

printError( TEXT("OpenProcess") );

else

{

dwPriorityClass = GetPriorityClass( hProcess );

if( !dwPriorityClass )

printError( TEXT("GetPriorityClass") );

CloseHandle( hProcess );

}

\_tprintf( TEXT("\n Process ID = 0x%08X"), pe32.th32ProcessID );

\_tprintf( TEXT("\n Thread count = %d"), pe32.cntThreads );

\_tprintf( TEXT("\n Parent process ID = 0x%08X"), pe32.th32ParentProcessID );

\_tprintf( TEXT("\n Priority base = %d"), pe32.pcPriClassBase );

if( dwPriorityClass )

\_tprintf( TEXT("\n Priority class = %d"), dwPriorityClass );

// List the modules and threads associated with this process

ListProcessModules( pe32.th32ProcessID );

ListProcessThreads( pe32.th32ProcessID );

} while( Process32Next( hProcessSnap, &pe32 ) );

CloseHandle( hProcessSnap );

return( TRUE );

}

BOOL ListProcessModules( DWORD dwPID )

{

HANDLE hModuleSnap = INVALID\_HANDLE\_VALUE;

MODULEENTRY32 me32;

// Take a snapshot of all modules in the specified process.

hModuleSnap = CreateToolhelp32Snapshot( TH32CS\_SNAPMODULE, dwPID );

if( hModuleSnap == INVALID\_HANDLE\_VALUE )

{

printError( TEXT("CreateToolhelp32Snapshot (of modules)") );

return( FALSE );

}

// Set the size of the structure before using it.

me32.dwSize = sizeof( MODULEENTRY32 );

// Retrieve information about the first module,

// and exit if unsuccessful

if( !Module32First( hModuleSnap, &me32 ) )

{

printError( TEXT("Module32First") ); // show cause of failure

CloseHandle( hModuleSnap ); // clean the snapshot object

return( FALSE );

}

// Now walk the module list of the process,

// and display information about each module

do

{

\_tprintf( TEXT("\n\n MODULE NAME: %s"), me32.szModule );

\_tprintf( TEXT("\n Executable = %s"), me32.szExePath );

\_tprintf( TEXT("\n Process ID = 0x%08X"), me32.th32ProcessID );

\_tprintf( TEXT("\n Ref count (g) = 0x%04X"), me32.GlblcntUsage );

\_tprintf( TEXT("\n Ref count (p) = 0x%04X"), me32.ProccntUsage );

\_tprintf( TEXT("\n Base address = 0x%08X"), (DWORD) me32.modBaseAddr );

\_tprintf( TEXT("\n Base size = %d"), me32.modBaseSize );

} while( Module32Next( hModuleSnap, &me32 ) );

CloseHandle( hModuleSnap );

return( TRUE );

}

BOOL ListProcessThreads( DWORD dwOwnerPID )

{

HANDLE hThreadSnap = INVALID\_HANDLE\_VALUE;

THREADENTRY32 te32;

// Take a snapshot of all running threads

hThreadSnap = CreateToolhelp32Snapshot( TH32CS\_SNAPTHREAD, 0 );

if( hThreadSnap == INVALID\_HANDLE\_VALUE )

return( FALSE );

// Fill in the size of the structure before using it.

te32.dwSize = sizeof(THREADENTRY32);

// Retrieve information about the first thread,

// and exit if unsuccessful

if( !Thread32First( hThreadSnap, &te32 ) )

{

printError( TEXT("Thread32First") ); // show cause of failure

CloseHandle( hThreadSnap ); // clean the snapshot object

return( FALSE );

}

// Now walk the thread list of the system,

// and display information about each thread

// associated with the specified process

do

{

if( te32.th32OwnerProcessID == dwOwnerPID )

{

\_tprintf( TEXT("\n\n THREAD ID = 0x%08X"), te32.th32ThreadID );

\_tprintf( TEXT("\n Base priority = %d"), te32.tpBasePri );

\_tprintf( TEXT("\n Delta priority = %d"), te32.tpDeltaPri );

\_tprintf( TEXT("\n"));

}

} while( Thread32Next(hThreadSnap, &te32 ) );

CloseHandle( hThreadSnap );

return( TRUE );

}

void printError( TCHAR\* msg )

{

DWORD eNum;

TCHAR sysMsg[256];

TCHAR\* p;

eNum = GetLastError( );

FormatMessage( FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, eNum,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), // Default language

sysMsg, 256, NULL );

// Trim the end of the line and terminate it with a null

p = sysMsg;

while( ( \*p > 31 ) || ( \*p == 9 ) )

++p;

do { \*p-- = 0; } while( ( p >= sysMsg ) &&

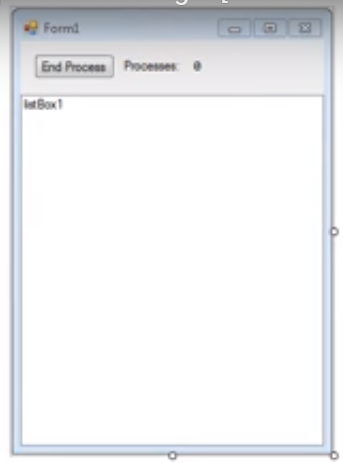
( ( \*p == '.' ) || ( \*p < 33 ) ) );

// Display the message

\_tprintf( TEXT("\n WARNING: %s failed with error %d (%s)"), msg, eNum, sysMsg );

}

**C# приклад простого диспетчеру задач:**



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Diagnostics;

namespace SimpleTaskManager

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

Process[] procs;

private void GetProcesses()

{

procs = Process.GetProcesses();

if (Convert.ToInt32(label2.Text) != procs.Length) // Check if new processes have been started or terminated

{

listBox1.Items.Clear();

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

{

listBox1.Items.Add(procs[i].ProcessName); // Add the process name to the listbox

}

label2.Text = procs.Length.ToString();

}

}

private void Form1\_Load(object sender, EventArgs e)

{

GetProcesses();

}

// Check every 1 second for changes in the processes list

private void timer1\_Tick(object sender, EventArgs e)

{

GetProcesses();

}

private void button1\_Click(object sender, EventArgs e)

{

procs[listBox1.SelectedIndex].Kill(); // Kill the process coresponding to the selected index of listbox1

}

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

{

procs[listBox1.SelectedIndex].Kill();

}

}

}

**Завдання**

**Створити проект який використовує функції запуску чи зупинення процесів, а також написати власні функції-обгортки для огляду системи. Працювати, використовуючи Git.**

**Або**

**В один загальний файл на С++ додати одну з наступних функцій:**

1. **Функцію сортування ряду**
2. **Функцію знаходження медіани заданого ряду**
3. **Функцію знаходження моди заданого ряду**
4. **Функцію обчислення середнього значення заданого ряду**
5. **Функцію обчислення суми заданого ряду**
6. **Функцію знаходження найбільших 5 елементів ряду**
7. **Функцію знаходження найменших 5 елементів ряду**