Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики

и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе №4

на тему

«Управление процессами и потоками (Windows). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности»

Выполнил:

студент гр. 153504

Сивый А.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты 5](#_1fob9te)

[Вывод 6](#_4q3p755gzys6)

[Список использованных источников 7](#_4q3p755gzys6)

[Приложение А (обязательное) Листинг кода 8](#_4q3p755gzys6)

**1 ЦЕЛИ РАБОТЫ**

1 Изучить способы управления процессами и потоками в Windows.

2 Изучить, как происходит порождение, завершение, изменение приоритетов процессов и потоков.

3 Разработать приложение для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе Windows является важной задачей для обеспечения эффективной работы системы. Процесс представляет собой экземпляр выполняющейся программы, которая может содержать один или несколько потоков. Потоки, в свою очередь, представляют собой независимые последовательности инструкций, которые могут выполняться параллельно в рамках процесса.

При порождении процесса операционная система выделяет ему ресурсы, такие как память, файловые дескрипторы и другие системные ресурсы. Процесс может быть порожден другим процессом с помощью системного вызова, либо может быть запущен из командной строки или графического интерфейса операционной системы.

Завершение процесса может происходить по различным причинам, например, когда программа завершила свое выполнение или была принудительно остановлена операционной системой. При завершении процесса освобождаются все выделенные ему ресурсы.

В данной лабораторной работе использовались такие функции Win32 Api, как:

1 CreateProcess – функция, которая используется для создания нового процесса и его первичного потока. Она принимает несколько параметров: lpApplicationName, lpCommandLine, lpProcessAttribytes.

2 SuspendThread – используется для приостановки выполнения потока, она принимает один параметр: hThread – дескриптор потока, который нужно приостановить. Функция возвращает значение типа DWORD. Если она выполнена успешно, она возвращает ненулевое значение, которое представляет количество приостановленных потоков. В случае ошибки, функция возвращает 0.

3 ResumeThread – используется для возобновления выполнения приостановленного потока. Она принимает один параметр: hThread – дескриптор потока, который нужно возобновить. Функция возвращает значение типа DWORD. Если она выполнена успешно, она возвращает ненулевое значение, которое представляет количество приостановленных потоков после возобновления. В случае ошибки, возвращает 0.

4 TerminateProcess – используется для завершения процесса. Она принимает два параметра: hProcess – дескриптор процесса, который нужно приостановить и uExitCode – код завершения процесса. Функция возвращает значение типа BOOL. Если она выполнена успешно, она возвращает TRUE, иначе FALSE.

Это неполный список всех функций, необходимых для создания приложения, которое позволяет управлять процессами.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Была создана программа, которая позволяет наблюдать за всеми запущенными процессами в системе Windows и управлять ими. (рисунок 3.1).

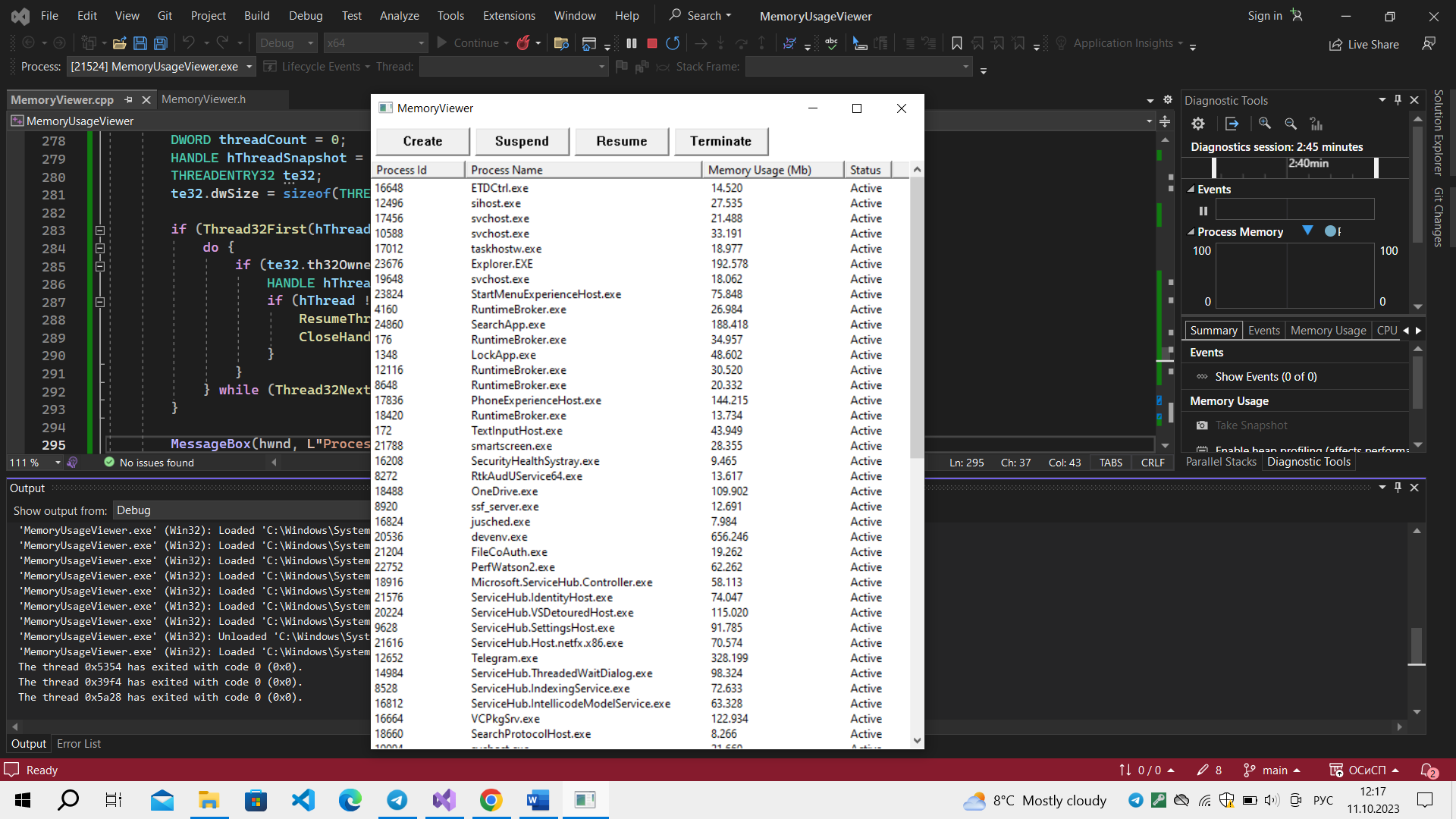


Рисунок 3.1 – Результат работы программы

В окне созданной программы можно наблюдать список всех запущенных процессов. Данные таблицы ежесекундно обновляются, что позволяет наблюдать изменение потребляемой памяти или появление новых процессов. Также можно наблюдать кнопки, при нажатии на которые происходит создание, приостановка, возобновление и остановка процессов.

**ВЫВОД**

В результате выполнения работы было разработано приложение, которое позволяет управлять и наблюдать за памятью, используемой разными процессами Windows. Для этой цели был изучен теоретический материал по различным библиотекам. Также программный продукт имеет возможность порождать, приостанавливать, восстанавливать и останавливать процессы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Psapi, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/psapi/>.

[2] Управления памятью Windows [Электронный ресурс]. – Режим доступа: learn.microsoft.com/ru-ru/windowshardware/drivers/kernel/managing-memory-for-drivers.

[3] Управление памятью и вводом-выводом в ОС Windows. [Электронный ресурс] – Режим доступа: https://studopedia.ru/25\_3449\_tema-upravlenie-pamyatyu-i-vvodomvivodom-v-os-Windows.html

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <Psapi.h>

#include <stdio.h>

#include <wchar.h>

#include <string>

#include <TlHelp32.h>

#include "MemoryViewer.h"

#include <windows.h>

#include <tlhelp32.h>

int UpdateProcessList() {

DWORD processes[1024];

DWORD bytesReturned;

// Получаем идентификаторы запущенных процессов

if (!EnumProcesses(processes, sizeof(processes), &bytesReturned)) {

std::cout << "Error with call EnumProcesses" << std::endl;

return 1;

}

// Вычисление количества процессов

int numProcesses = bytesReturned / sizeof(DWORD);

// Выводим идентификаторы процессор

int row = 0;

int count = ListView\_GetItemCount(hList);

for (int i = 0; i < numProcesses; i++) {

// Открываем процесс для получения дополнительной ифнормации

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, processes[i]);

if (hProcess != NULL) {

HMODULE hModule;

DWORD cbNeeded;

PROCESS\_MEMORY\_COUNTERS pmc;

pmc.cb = sizeof(pmc);

char path[MAX\_PATH];

DWORD size = MAX\_PATH;

bool condition = EnumProcessModules(hProcess, &hModule, sizeof(hModule), &cbNeeded) &&

(GetModuleBaseNameA(hProcess, hModule, path, sizeof(path)) != 0) &&

GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc));

//// Получаем базовый модуль процесса

if (condition) {

if (count <= row)

{

LVITEM lvItem;

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = row;

lvItem.iSubItem = 0;

lvItem.pszText = new wchar\_t[0];

ListView\_InsertItem(hList, &lvItem);

}

wchar\_t buffer[256];

swprintf\_s(buffer, L"%d", processes[i]);

ListView\_SetItemText(hList, row, 0, buffer);

int length = strlen(path) + 1;

int size = MultiByteToWideChar(CP\_UTF8, 0, path, length, NULL, 0);

MultiByteToWideChar(CP\_UTF8, 0, path, length, buffer, size);

ListView\_SetItemText(hList, row, 1, buffer);

swprintf\_s(buffer, L"% .3lf", (double)pmc.WorkingSetSize / (1024 \* 1024));

ListView\_SetItemText(hList, row, 2, buffer);

DWORD exitCode;

if (GetExitCodeProcess(hProcess, &exitCode))

{

if (exitCode == STILL\_ACTIVE)

{

length = strlen(path) + 1;

size = MultiByteToWideChar(CP\_UTF8, 0, "Active", length, NULL, 0);

MultiByteToWideChar(CP\_UTF8, 0, "Active", length, buffer, size);

}

else {

// Процесс приостановлен

length = strlen(path) + 1;

size = MultiByteToWideChar(CP\_UTF8, 0, "Suspended", length, NULL, 0);

MultiByteToWideChar(CP\_UTF8, 0, "Suspended", length, buffer, size);

}

}

else {

length = strlen(path) + 1;

size = MultiByteToWideChar(CP\_UTF8, 0, "Undefined", length, NULL, 0);

MultiByteToWideChar(CP\_UTF8, 0, "Undefined", length, buffer, size);

}

ListView\_SetItemText(hList, row, 3, buffer);

row++;

std::cout << "Process Identificator: " << processes[i] << " Process Name: " << path << " Memory usage: " << std::fixed << std::setprecision(2) << (double)pmc.WorkingSetSize / (1024 \* 1024) << " Mb" << std::endl;

}

// Закрываем дескриптор процесса

CloseHandle(hProcess);

}

}

for (int i = row; i < count; i++) {

ListView\_DeleteItem(hList, i);

}

return 0;

}

VOID CALLBACK TimerCallback(HWND hWnd, UINT uMsg, UINT\_PTR idEvent, DWORD dwTime);

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLine, int nCmdShow) {

hInstance = hInst;

WNDCLASSEX wc = { sizeof(WNDCLASSEX) };

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpfnWndProc = WindowProc;

wc.lpszClassName = L"MyWindowClass";

wc.style = CS\_HREDRAW | CS\_VREDRAW;

if (!RegisterClassEx(&wc)) {

MessageBox(NULL, L"Register window class error!", L"Error!", MB\_ICONERROR | MB\_OK);

return 1;

}

CreateMainWindow();

if (hwnd == NULL) {

MessageBox(NULL, L"Creating main window error!", L"Error!", MB\_ICONERROR | MB\_OK);

return 1;

}

UpdateProcessList();

UINT\_PTR timerId = SetTimer(NULL, 0, interval, TimerCallback);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

KillTimer(NULL, timerId);

return 0;

}

void CreateNewProcess() {

OPENFILENAME ofn;

wchar\_t filePath[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = L"Executable Files (\*.exe)\0\*.exe\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = filePath;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_EXPLORER | OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY;

if (GetOpenFileName(&ofn)) {

STARTUPINFO si = { 0 };

PROCESS\_INFORMATION pi = { 0 };

if (CreateProcess(NULL, filePath, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

MessageBox(NULL, L"Failed to start process!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

}

void SuspendProcess() {

// Получаем индекс выбранной строки

int iSelectedCount = ListView\_GetSelectedCount(hList);

if (iSelectedCount > 0)

{

int selectedIndex = ListView\_GetSelectionMark(hList);

// Выделям память под структуру для хранения информации о ячейке

LVITEM item;

item.mask = LVIF\_TEXT;

item.iItem = selectedIndex;

item.iSubItem = 0;

item.pszText = new wchar\_t[MAX\_PATH];

item.cchTextMax = MAX\_PATH;

// Получаем значение выбранной ячейки

ListView\_GetItemText(hList, selectedIndex, 0, item.pszText, MAX\_PATH);

DWORD processId = wcstoul(item.pszText, NULL, 10);

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, processId);

if (hProcess == NULL)

{

MessageBox(hwnd, L"Can't open process.", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

// Приостановка процесса

DWORD threadCount = 0;

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

THREADENTRY32 te32;

te32.dwSize = sizeof(THREADENTRY32);

if (Thread32First(hThreadSnapshot, &te32))

{

do

{

if (te32.th32OwnerProcessID == GetProcessId(hProcess))

{

HANDLE hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, te32.th32ThreadID);

if (hThread != NULL)

{

SuspendThread(hThread);

CloseHandle(hThread);

threadCount++;

}

}

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

}

MessageBox(hwnd, L"Process suspended!", L"Info", MB\_OK | MB\_ICONINFORMATION);

CloseHandle(hThreadSnapshot);

delete[] item.pszText;

}

else {

MessageBox(hwnd, L"Element not selected!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

void ResumeProcess() {

// Получаем индекс выбранной строки

int iSelectedCount = ListView\_GetSelectedCount(hList);

if (iSelectedCount > 0)

{

int selectedIndex = ListView\_GetSelectionMark(hList);

// Выделям память под структуру для хранения информации о ячейке

LVITEM item;

item.mask = LVIF\_TEXT;

item.iItem = selectedIndex;

item.iSubItem = 0;

item.pszText = new wchar\_t[MAX\_PATH];

item.cchTextMax = MAX\_PATH;

// Получаем значение выбранной ячейки

ListView\_GetItemText(hList, selectedIndex, 0, item.pszText, MAX\_PATH);

DWORD processId = wcstoul(item.pszText, NULL, 10);

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, processId);

if (hProcess == NULL)

{

MessageBox(hwnd, L"Can't open process.", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

// Возобновление процесса

DWORD threadCount = 0;

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

THREADENTRY32 te32;

te32.dwSize = sizeof(THREADENTRY32);

if (Thread32First(hThreadSnapshot, &te32)) {

do {

if (te32.th32OwnerProcessID == GetProcessId(hProcess)) {

HANDLE hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, te32.th32ThreadID);

if (hThread != NULL) {

ResumeThread(hThread);

CloseHandle(hThread);

}

}

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

}

MessageBox(hwnd, L"Process resumed!", L"Info", MB\_OK | MB\_ICONINFORMATION);

CloseHandle(hThreadSnapshot);

CloseHandle(hProcess);

delete[] item.pszText;

}

else {

MessageBox(hwnd, L"Element not selected!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

void TerminateProcess() {

int iSelectedCount = ListView\_GetSelectedCount(hList);

if (iSelectedCount > 0)

{

int selectedIndex = ListView\_GetSelectionMark(hList);

// Выделям память под структуру для хранения информации о ячейке

LVITEM item;

item.mask = LVIF\_TEXT;

item.iItem = selectedIndex;

item.iSubItem = 0;

item.pszText = new wchar\_t[MAX\_PATH];

item.cchTextMax = MAX\_PATH;

// Получаем значение выбранной ячейки

ListView\_GetItemText(hList, selectedIndex, 0, item.pszText, MAX\_PATH);

DWORD processId = wcstoul(item.pszText, NULL, 10);

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, processId);

if (hProcess == NULL)

{

MessageBox(hwnd, L"Can't open process.", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

// Закрытие процесса

TerminateProcess(hProcess, 0);

CloseHandle(hProcess);

}

else {

MessageBox(hwnd, L"Element not selected!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_COMMAND:

if (lParam == (LPARAM)createButton)

CreateNewProcess();

else if (lParam == (LPARAM)suspendButton)

SuspendProcess();

else if (lParam == (LPARAM)resumeButton)

ResumeProcess();

else if (lParam == (LPARAM)terminateButton)

TerminateProcess();

break;

case WM\_SIZE:

{

break;

}

case WM\_CLOSE:

DestroyWindow(hwnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

}

VOID CALLBACK TimerCallback(HWND hWnd, UINT uMsg, UINT\_PTR idEvent, DWORD dwTime)

{

UpdateProcessList();

}

void CreateMainWindow() {

hwnd = CreateWindow(\_T("MyWindowClass"), L"MemoryViewer",WS\_OVERLAPPEDWINDOW, 100, 100, 600, 700, NULL, NULL, hInstance, NULL);

hList = CreateWindow(WC\_LISTVIEW, \_T(""), WS\_CHILD | WS\_VISIBLE | LVS\_REPORT, 0, 40, 585, 620, hwnd, NULL, NULL, NULL);

createButton = CreateWindowEx(0, L"BUTTON", L"Create", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 5, 5, 100, 30, hwnd, NULL, hInstance, NULL);

suspendButton = CreateWindowEx(0, L"BUTTON", L"Suspend", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 110, 5, 100, 30, hwnd, NULL, hInstance, NULL);

resumeButton = CreateWindowEx(0, L"BUTTON", L"Resume", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 215, 5, 100, 30, hwnd, NULL, hInstance, NULL);

terminateButton = CreateWindowEx(0, L"BUTTON", L"Terminate", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 320, 5, 100, 30, hwnd, NULL, hInstance, NULL);

ListView\_SetExtendedListViewStyle(hList, LVS\_EX\_FULLROWSELECT);

LVCOLUMN lvColumn;

lvColumn.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvColumn.cx = 100;

lvColumn.pszText = const\_cast<wchar\_t\*>(L"Process Id");

ListView\_InsertColumn(hList, 0, &lvColumn);

LVCOLUMN lvColumn2;

lvColumn2.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvColumn2.cx = 250;

lvColumn2.pszText = const\_cast<wchar\_t\*>(L"Process Name");

ListView\_InsertColumn(hList, 1, &lvColumn2);

LVCOLUMN lvColumn3;

lvColumn3.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvColumn3.cx = 150;

lvColumn3.pszText = const\_cast<wchar\_t\*>(L"Memory Usage (Mb)");

ListView\_InsertColumn(hList, 2, &lvColumn3);

LVCOLUMN lvColumn4;

lvColumn4.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvColumn4.cx = 50;

lvColumn4.pszText = const\_cast<wchar\_t\*>(L"Status");

ListView\_InsertColumn(hList, 3, &lvColumn4);

}