

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 №3

на тему

**УПРАВЛЕНИЕ ПАМЯТЬЮ И ВВОДОМ-ВЫВОДОМ,  
РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ВВОДА-ВЫВОДА WINDOWS.  
ФУНКЦИИ API ПОДСИСТЕМЫ ПАМЯТИ WIN 32. ОРГАНИЗАЦИЯ И  
КОНТРОЛЬ АСИНХРОННЫХ ОПЕРАЦИЙ ВВОДА-ВЫВОДА.  
ОТОБРАЖЕНИЕ ФАЙЛОВ В ПАМЯТЬ**

Студент

Преподаватель

М. А. Шкарубский

Н. Ю. Гриценко

Минск 2023

## СОДЕРЖАНИЕ

1 Постановка задачи .....	3
2 Теоретические сведения.....	4
2.1 Работа с памятью.....	4
2.2 Виртуальная память .....	4
2.3 Ввод-вывод.....	5
2.4 Асинхронность .....	5
3 Результат выполнения .....	7
Заключение.....	8
Список использованных источников .....	9
Приложение А (обязательное) Листинг кода.....	10

## **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является создание приложения для мониторинга и управления системной памятью, отображающее текущее потребление памяти и других ресурсов вычислительного устройства различными процессами.

## **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

### **2.1 Работа с памятью**

Работа с памятью является важной частью работы с Win 32. Управление памятью в современных компьютерных системах — это процесс распределения и освобождения ресурсов памяти для эффективного выполнения программ. Он включает в себя следующие аспекты:

1 Выделение памяти: процесс выделения памяти предоставляет программам необходимое пространство для хранения данных во время выполнения. Это может включать в себя выделение стековой памяти для локальных переменных и динамическое выделение памяти для переменных переменной длины.

2 Освобождение памяти представляет собой процесс возврата выделенной ранее памяти обратно в систему. Это важно для предотвращения утечек памяти и эффективного использования ресурсов [1].

3 Управление фрагментацией: фрагментация памяти может привести к неэффективному использованию ресурсов, когда свободное пространство разделено на небольшие фрагменты. Управление фрагментацией включает в себя стратегии выделения и освобождения памяти для минимизации фрагментации.

В Win 32 операционная система предоставляет механизм управления памятью через "кучу" (Heap). Куча предоставляет динамическую память, которая может быть выделена и освобождена во время выполнения программы.

Функции API для работы с кучей:

- HeapCreate (создаёт новый объект кучи);
- HeapAlloc (выделяет блок памяти из кучи);
- HeapFree (освобождает ранее выделенный блок памяти).

### **2.2 Виртуальная память**

Win 32 использует концепцию виртуальной памяти, которая позволяет каждому процессу иметь своё собственное виртуальное адресное пространство. Это адресное пространство может быть больше, чем физическая память на компьютере.

Функции API для работы с виртуальной памятью:

- VirtualAlloc (выделяет виртуальную память в адресном пространстве процесса);

– VirtualFree (освобождает ранее выделенную виртуальную память).

## **2.3 Ввод-вывод**

Ввод-вывод (I/O) в компьютерных системах представляет собой процесс передачи данных между компьютером и внешними устройствами. Включает в себя следующие аспекты:

1 Синхронный ввод-вывод блокирует выполнение программы до завершения операции ввода-вывода, в то время как асинхронный позволяет программе продолжать выполнение других задач во время операции I/O.

2 Буферизация данных ввода-вывода включает в себя временное хранение данных в памяти перед передачей или после получения. Это повышает производительность, так как необходимо меньше обращений к физическим устройствам.

3 Отображение файлов в память представляет собой создание виртуального отображения файла в адресном пространстве процесса, что облегчает чтение и запись данных в файл. CreateFileMapping и MapViewOfFile используются для отображения файла в виртуальное адресное пространство процесса.

4 Асинхронные операции I/O позволяют программе продолжать выполнение других задач во время ожидания завершения операций ввода-вывода. ReadFileEx и WriteFileEx совместно с использованием структур OVERLAPPED позволяют асинхронные операции. [2]

5 Функции API для работы с файлами и устройствами [3]: Win32 API предоставляет функции для работы с файлами и устройствами, включая создание, чтение, запись и управление файлами и драйверами устройств. CreateFile создаёт или открывает файл или ввод-выводное устройство. ReadFile и WriteFile чтение и запись данных в файл.

## **2.4 Асинхронность**

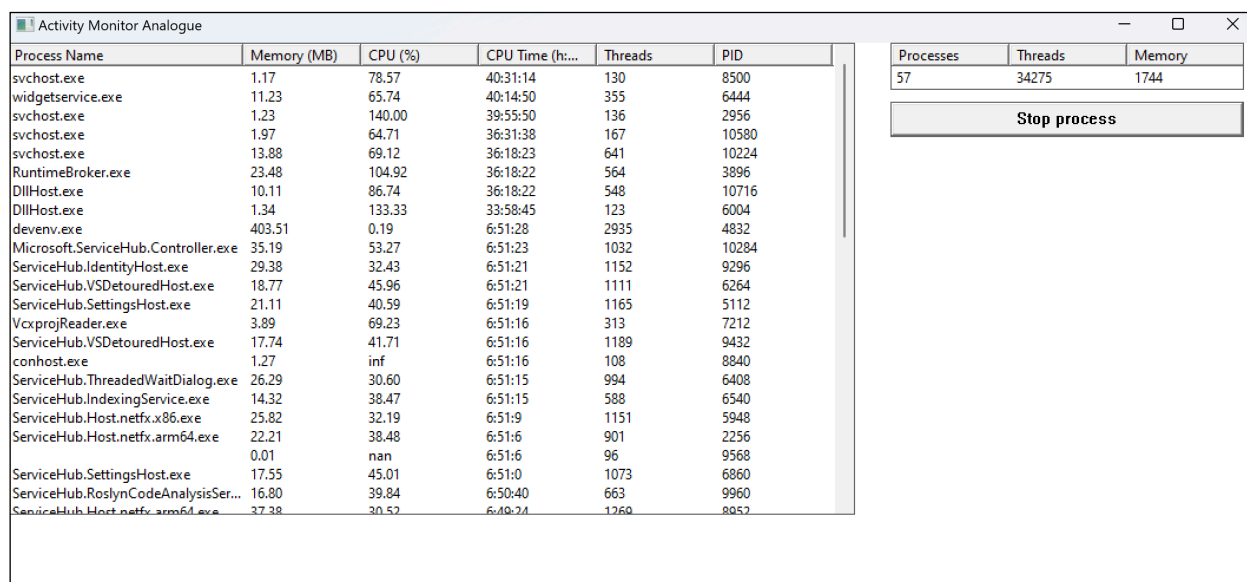
Асинхронность – это концепция, когда операции выполняются независимо от основного потока выполнения. Вместо блокировки программы в ожидании завершения операции, программа продолжает свою работу, а результат асинхронной операции обрабатывается позднее. Это особенно полезно в веб-разработке, обработке ввода-вывода и других сценариях, где задержка может быть нежелательной.

Для реализации асинхронных операций часто используются очереди и события. Очереди позволяют планировать задачи на выполнение в будущем, а события уведомляют о завершении операции или других изменениях состояния. Это снижает блокировку и позволяет эффективно использовать ресурсы.

В лабораторной работе асинхронность используется для выполнения фоновой задачи по обновлению списка процессов. Это улучшает отзывчивость интерфейса, так как пользователь может продолжать взаимодействие с приложением, даже если долгая операция все ещё выполняется в фоновом режиме.

### 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В рамках выполнения лабораторной работы было разработано приложение мониторинга и управления системной памятью, отображающее текущее потребление памяти и других ресурсов вычислительного устройства различными процессами. На рисунке 1 отображено окно такого приложения.



Process Name	Memory (MB)	CPU (%)	CPU Time (h:mm:ss)	Threads	PID
svchost.exe	1.17	78.57	40:31:14	130	8500
widgetservice.exe	11.23	65.74	40:14:50	355	6444
svchost.exe	1.23	140.00	39:55:50	136	2956
svchost.exe	1.97	64.71	36:31:38	167	10580
svchost.exe	13.88	69.12	36:18:23	641	10224
RuntimeBroker.exe	23.48	104.92	36:18:22	564	3896
DllHost.exe	10.11	86.74	36:18:22	548	10716
DllHost.exe	1.34	133.33	33:58:45	123	6004
devenv.exe	403.51	0.19	6:51:28	2935	4832
Microsoft.ServiceHub.Controller.exe	35.19	53.27	6:51:23	1032	10284
ServiceHub.IdentityHost.exe	29.38	32.43	6:51:21	1152	9296
ServiceHub.VSDetouredHost.exe	18.77	45.96	6:51:21	1111	6264
ServiceHub.SettingsHost.exe	21.11	40.59	6:51:19	1165	5112
VcxprojReader.exe	3.89	69.23	6:51:16	313	7212
ServiceHub.VSDetouredHost.exe	17.74	41.71	6:51:16	1189	9432
conhost.exe	1.27	inf	6:51:16	108	8840
ServiceHub.ThreadedWaitDialog.exe	26.29	30.60	6:51:15	994	6408
ServiceHub.IndexingService.exe	14.32	38.47	6:51:15	588	6540
ServiceHub.Host.netfx.x86.exe	25.82	32.19	6:51:9	1151	5948
ServiceHub.Host.netfx.arm64.exe	22.21	38.48	6:51:6	901	2256
	0.01	nan	6:51:6	96	9568
ServiceHub.SettingsHost.exe	17.55	45.01	6:51:0	1073	6860
ServiceHub.RoslynCodeAnalysisSer...	16.80	39.84	6:50:40	663	9960
ServiceHub.Host.netfx.arm64.exe	37.38	30.52	6:49:24	1269	8952

Processes	Threads	Memory
57	34275	1744

Stop process

Рисунок 1 – Окно приложения

В процессе использования приложения пользователь может просматривать запущенные процессы, потребляемые ими ресурсы, их идентификаторы и названия, а также продолжительность активного режима.

Более того, пользователь также может отслеживать суммарную потребляемую память, количество процессов и общее количество потоков на все процессы.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения лабораторной работы были изучены и освоены способы управления памятью и вводом-выводом, расширенные возможности ввода-вывода Windows, функции API подсистемы памяти Win 32, организация и контроль асинхронных операций ввода-вывода. Помимо этого, было разработано приложение, использующее вышеперечисленные технологии для обеспечения надежности и корректности.



## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- [1] Управление памятью [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/memory/memory-management>.
- [2] Асинхронная операция [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/wininet/asynchronous-operation>.
- [3] Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dms.karelia.ru/win32/>.

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Листинг кода

#### Листинг 1 – Файл Lab3.cpp

```
#pragma once

#include <windows.h>
#include <CommCtrl.h>
#include <psapi.h>
#include <string>
#include <sstream>
#include <fstream>
#include <iostream>

#ifndef UNICODE
#define UNICODE
#endif

#define STOP_BTN_ID 999

HWND hwndListView;
HWND hwndTotalsView;
HWND hwndStopButton;
HANDLE hUpdateThread, hRefreshThread;
DWORD processes[1024], cbNeeded, cProcesses;
int totalsIndex;

bool getMemoryInfo(HANDLE hProcess, wchar_t(&memoryUsageDisplay)[256]) {
    PROCESS_MEMORY_COUNTERS_EX pmc;

    if (GetProcessMemoryInfo(hProcess, (PROCESS_MEMORY_COUNTERS*)&pmc,
        sizeof(pmc)))
    {
        swprintf(memoryUsageDisplay, 256, L"%.2f", (double) ((double)
            pmc.WorkingSetSize / (1024 * 1024)));

        return true;
    }

    return false;
}

bool getCPUUsage(HANDLE hProcess, wchar_t (&cpuUsageDisplay) [256]) {
    FILETIME ftCreation, ftExit, ftKernel, ftUser;

    if (GetProcessTimes(hProcess, (FILETIME*)&ftCreation, (FILETIME*)&ftExit,
        (FILETIME*)&ftKernel, (FILETIME*)&ftUser)) {
        ULONGLONG processTime = ((ULONGLONG) (ftKernel.dwHighDateTime << 32) |
```

```

ftKernel.dwLowDateTime);
    ULONGLONG systemTime = ((ULONGLONG)(ftUser.dwHighDateTime << 32) |
ftUser.dwLowDateTime);

    //wchar_t cpuUsageDisplay[256];
    double cpuUsage = processTime * 100.0 / systemTime;

    swprintf(cpuUsageDisplay, 256, L"%.2f", cpuUsage);

    return true;
}

return false;
}

wchar_t* getActiveTime(HANDLE hProcess) {
    FILETIME ftCreation, ftExit, ftKernel, ftUser;

    if (GetProcessTimes(hProcess, (FILETIME*)&ftCreation, (FILETIME*)&ftExit,
(FILETIME*)&ftKernel, (FILETIME*)&ftUser)) {
        FILETIME ftCurrent;

        GetSystemTimeAsFileTime(&ftCurrent);

        ULONGLONG activeTime = *((ULONGLONG*)&ftCurrent) -
*((ULONGLONG*)&ftCreation);
        ULONGLONG activeSeconds = activeTime / 10000000;

        int hours = activeSeconds / 3600;
        int minutes = (activeSeconds % 3600) / 60;
        int seconds = activeSeconds % 60;

        wchar_t* activeTimeDisplay = _wcsdup((std::to_wstring(hours) + L":" +
std::to_wstring(minutes) + L":" + std::to_wstring(seconds)).c_str());

        return activeTimeDisplay;
    }

    return (wchar_t*) L"Err";
}

wchar_t* getThreadCount(HANDLE hProcess) {
    DWORD threadCount;

    GetProcessHandleCount(hProcess, &threadCount);

    return _wcsdup(std::to_wstring(threadCount).c_str());
}

DWORD WINAPI RefreshThreadProc(LPVOID lpParam) {
    while (true) {
        HWND hwnd = (HWND)lpParam;

        EnumProcesses(processes, sizeof(processes), &cbNeeded);
    }
}

```

```

        cProcesses = cbNeeded / sizeof(DWORD);

        Sleep(1000);
    }
}

DWORD WINAPI UpdateThreadProc(LPVOID lpParam) {
    int itemCount = ListView_GetItemCount(hwndListView);
    HWND hwnd = (HWND)lpParam;

    while (true) {

        int scrollPos = GetScrollPos(hwndListView, SB_VERT);

        if (itemCount != cProcesses) {
            ListView_DeleteAllItems(hwndListView);

            int trueCounter = 0;
            for (DWORD i = 0; i < cProcesses; i++)
            {
                HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ, FALSE, processes[i]);

                if (hProcess == NULL)
                    continue;

                TCHAR szProcessName[MAX_PATH] = L"";
                HMODULE hModule;
                DWORD cbNeeded;

                if (EnumProcessModules(hProcess, &hModule, sizeof(hModule),
&cbNeeded))
                    GetModuleBaseName(hProcess, hModule, szProcessName,
sizeof(szProcessName));

                LVITEM lvItem = { 0 };
                lvItem.mask = LVIF_TEXT;
                lvItem.iItem = trueCounter++;

                int itemIndex = ListView_InsertItem(hwndListView, &lvItem);

                CloseHandle(hProcess);
            }

            itemCount = cProcesses;
        }

        int trueCounter = 0;
        for (DWORD i = 0; i < cProcesses; i++)
        {
            HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ, FALSE, processes[i]);

            if (hProcess == NULL)
                continue;

            TCHAR szProcessName[MAX_PATH] = L"";

```

```

        HMODULE hModule;
        DWORD cbNeeded;

        if (EnumProcessModules(hProcess, &hModule, sizeof(hModule),
&cbNeeded))
            GetModuleBaseName(hProcess, hModule, szProcessName,
sizeof(szProcessName));

        // MEMORY USAGE DISPLAY
        wchar_t memoryUsageDisplay[256];
        getMemoryInfo(hProcess, memoryUsageDisplay);

        // CPU USAGE DISPLAY
        wchar_t cpuUsageDisplay[256];
        getCPUUsage(hProcess, cpuUsageDisplay);

        // ACTIVE TIME DISPLAY
        wchar_t* activeTimeDisplay = getActiveTime(hProcess);

        // THREADS PER PROCESS DISPLAY
        wchar_t* threadCountDisplay = getThreadCount(hProcess);

        // PID
        wchar_t* pidDisplay =
_wcsdup(std::to_wstring(GetProcessId(hProcess)).c_str());

        ListView_SetItemText(hwndListView, trueCounter, 0,
szProcessName);
        ListView_SetItemText(hwndListView, trueCounter, 1,
memoryUsageDisplay);
        ListView_SetItemText(hwndListView, trueCounter, 2,
cpuUsageDisplay);
        ListView_SetItemText(hwndListView, trueCounter, 3,
activeTimeDisplay);
        ListView_SetItemText(hwndListView, trueCounter, 4,
threadCountDisplay);
        ListView_SetItemText(hwndListView, trueCounter, 5, pidDisplay);

        trueCounter++;

        CloseHandle(hProcess);
    }

    SetScrollPos(hwndListView, SB_VERT, scrollPos, TRUE);

    // TOTALS

    DWORD totalProcesses = ListView_GetItemCount(hwndListView);
    wchar_t totalProcessesDisplay[256];

    _itow_s(totalProcesses, totalProcessesDisplay, 10);

    DWORD totalThreads = 0;
    DWORD totalMemory = 0;

    for (DWORD i = 0; i < totalProcesses; i++) {
        wchar_t threadCountDisplay[256];

```

```

        wchar_t processMemoryUsageDisplay[256];

        ListView_GetItemText(hwndListView, i, 4, threadCountDisplay,
sizeof(threadCountDisplay));
        ListView_GetItemText(hwndListView, i, 1,
processMemoryUsageDisplay, sizeof(processMemoryUsageDisplay));

        totalThreads += _wtoi(threadCountDisplay);
        totalMemory += _wtoi(processMemoryUsageDisplay);
    }

    wchar_t totalThreadsDisplay[256];
    _itow_s(totalThreads, totalThreadsDisplay, 10);

    wchar_t totalMemoryUsageDisplay[256];
    _itow_s(totalMemory, totalMemoryUsageDisplay, 10);

    ListView_SetItemText(hwndTotalsView, totalsIndex, 0,
totalProcessesDisplay);
    ListView_SetItemText(hwndTotalsView, totalsIndex, 1,
totalThreadsDisplay);
    ListView_SetItemText(hwndTotalsView, totalsIndex, 2,
totalMemoryUsageDisplay);

    Sleep(250);
}

return 0;
}

// Window procedure function
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    switch (uMsg)
    {
    case WM_CREATE:
    {
        // Initialize the task manager interface
        // You can create UI elements and set up data structures here

        // Create the list view control
        hwndListView = CreateWindowEx(
            0, // Optional window styles
            WC_LISTVIEW, // Window class name
            L"", // Window title
            WS_VISIBLE | WS_CHILD | WS_BORDER | LVS_REPORT, // Window style
            0, 0, 720, 400, // Size and position
            hwnd, // Parent window handle
            NULL, // Menu handle
            GetModuleHandle(NULL), // Instance handle
            NULL // Additional application data
        );
        ListView_SetExtendedListViewStyle(hwndListView,
LVS_EX_AUTOSIZECOLUMNS);

        hwndTotalsView = CreateWindowEx(

```

```

        0, // Optional window styles
        WC_LISTVIEW, // Window class name
        L"", // Window title
        WS_VISIBLE | WS_CHILD | WS_BORDER | LVS_REPORT, // Window style
        750, 0, 300, 40, // Size and position
        hwnd, // Parent window handle
        NULL, // Menu handle
        GetModuleHandle(NULL), // Instance handle
        NULL // Additional application data
    );
    ListView_SetExtendedListViewStyle(hwndTotalsView,
    LVS_EX_AUTOSIZECOLUMNS);

    hwndStopButton = CreateWindow(
        L"BUTTON",
        L"Stop process",
        WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
        750, 50, 300, 30,
        hwnd,
        (HMENU) STOP_BTN_ID,
        (HINSTANCE)GetWindowLongPtr(hwnd, GWLP_HINSTANCE),
        NULL
    );

    if (hwndListView == NULL or hwndTotalsView == NULL or hwndStopButton
    == NULL)
        return -1;

    // Set up the columns in the list view control
    LVCOLUMN lvColumn = { };
    lvColumn.mask = LVCF_TEXT | LVCF_WIDTH;
    lvColumn.cx = 200;
    lvColumn.pszText = (LPWSTR) L"Process Name";
    ListView_InsertColumn(hwndListView, 0, &lvColumn);

    lvColumn.cx = 100;
    lvColumn.pszText = (LPWSTR) L"Memory (MB)";
    ListView_InsertColumn(hwndListView, 1, &lvColumn);

    lvColumn.cx = 100;
    lvColumn.pszText = (LPWSTR) L"CPU (%)";
    ListView_InsertColumn(hwndListView, 2, &lvColumn);

    lvColumn.cx = 100;
    lvColumn.pszText = (LPWSTR)L"CPU Time (h:m:s)";
    ListView_InsertColumn(hwndListView, 3, &lvColumn);

    lvColumn.cx = 100;
    lvColumn.pszText = (LPWSTR)L"Threads";
    ListView_InsertColumn(hwndListView, 4, &lvColumn);

    lvColumn.cx = 100;
    lvColumn.pszText = (LPWSTR)L"PID";
    ListView_InsertColumn(hwndListView, 5, &lvColumn);

    DWORD processes[1024], cbNeeded, cProcesses;
    EnumProcesses(processes, sizeof(processes), &cbNeeded);
    cProcesses = cbNeeded / sizeof(DWORD);

```

```

LVITEM lvItem = { 0 };
lvItem.mask = LVIF_TEXT;
lvItem.iItem = 0;

for (DWORD i = 0; i < cProcesses; i++)
    ListView_InsertItem(hwndListView, &lvItem);

LVCOLUMN tColumn = {};
tColumn.mask = LVCF_TEXT | LVCF_WIDTH;
tColumn.cx = 100;
tColumn.pszText = (LPWSTR)L"Processes";
ListView_InsertColumn(hwndTotalsView, 0, &tColumn);

tColumn.cx = 100;
tColumn.pszText = (LPWSTR)L"Threads";
ListView_InsertColumn(hwndTotalsView, 1, &tColumn);

tColumn.cx = 100;
tColumn.pszText = (LPWSTR)L"Memory";
ListView_InsertColumn(hwndTotalsView, 2, &tColumn);

LVITEM tItem = {};
tItem.mask = LVIF_TEXT;
tItem.iItem = 0;

totalsIndex = ListView_InsertItem(hwndTotalsView, &tItem);

hUpdateThread = CreateThread(NULL, 0, UpdateThreadProc, hwnd, 0,
NULL);
hRefreshThread = CreateThread(NULL, 0, RefreshThreadProc, hwnd, 0,
NULL);

break;
}

case WM_COMMAND:
{
    if (LOWORD(wParam) == STOP_BTN_ID && HIWORD(wParam) == BN_CLICKED)
    {
        for (int i = 0; i < ListView_GetItemCount(hwndListView); ++i) {
            UINT state = ListView_GetItemState(hwndListView, i,
LVIS_FOCUSED);

            bool isChecked = state == LVIS_FOCUSED;

            if (isChecked) {
                LVITEM lvItem = { 0 };
                lvItem.mask = LVIF_STATE | LVIF_TEXT;
                lvItem.iItem = i;
                lvItem.iSubItem = 5;
                lvItem.cchTextMax = 256;

                wchar_t buffer[256];
                lvItem.pszText = buffer;
                lvItem.stateMask = LVIS_STATEIMAGEMASK;

```



```

        ListView_GetItem(hwndListView, &lvItem);

        DWORD pid = _wtoi(buffer);

        HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, FALSE,
pid);

        if (hProcess != NULL) {
            TerminateProcess(hProcess, 0);
            CloseHandle(hProcess);
        }
        break;
    }
}
break;
}

case WM_CLOSE:
{
    DestroyWindow(hwnd);
    break;
}

case WM_DESTROY:
{
    PostQuitMessage(0);
    break;
}

default:
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
    // Register the window class
    const wchar_t CLASS_NAME[] = L"TaskManagerWindowClass";

    WNDCLASS wc = { };

    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = CLASS_NAME;

    RegisterClass(&wc);

    // Create the window
    HWND hwnd = CreateWindowEx(
        0,                                // Optional window styles
        CLASS_NAME,                        // Window class name
        L"Activity Monitor Analogue",     // Window title
        WS_OVERLAPPEDWINDOW,              // Window style

```

```

        // Size and position
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

        NULL, // Parent window
        NULL, // Menu
        hInstance, // Instance handle
        NULL // Additional application data
    );

    if (hwnd == NULL)
        return 0;

    // Display the window
    ShowWindow(hwnd, nCmdShow);

    // Run the message loop
    MSG msg = { };
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}

```