

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.**

Студент
Преподаватель

М. А. Шкарубский
Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи	3
2 Теоретические сведения.....	4
2.1 Процессы в Win32 API.....	4
2.2 Понятие потока.....	4
3 Результат выполнения	6
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. Приложение также позволяет останавливать процессы, отображая изменения в списке текущих процессов.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Процессы в Win32 API

Процесс в операционной системе Windows представляет собой выполняющуюся программу [1]. Каждый процесс имеет своё собственное виртуальное адресное пространство, наборы ресурсов и инструкций. Операционная система управляет множеством процессов, обеспечивая их изоляцию и параллельное выполнение.

Для создания нового процесса в Win32 используется функция `CreateProcess`. Она принимает параметры, такие как путь к исполняемому файлу, командная строка, атрибуты безопасности и др.

Процесс может быть завершён функцией `ExitProcess`, которая принимает код завершения. Также другой процесс может завершить процесс с использованием функции `TerminateProcess`.

Процесс также может быть приостановлен и возобновлён. Для этого используются функции `SuspendProcess` и `ResumeProcess` соответственно.

Каждый процесс имеет свой приоритет выполнения, который определяет, насколько процесс предпочтителен для выполнения относительно других процессов. Этот приоритет можно изменить с использованием функции `SetPriorityClass`.

2.2 Понятие потока

Поток представляет собой наименьшую единицу выполнения внутри процесса. Процесс может иметь несколько потоков, каждый из которых выполняется параллельно [2].

Функция `CreateThread` используется для создания нового потока. Она принимает указатель на функцию, которая будет выполнена в созданном потоке.

Поток завершается, когда функция, переданная при его создании, завершает своё выполнение. Также можно использовать функцию `ExitThread` для явного завершения потока.

Каждый поток имеет свой приоритет выполнения внутри процесса. Приоритет потока можно изменить, используя функции `SetThreadPriority`.

В многозадачных сценариях потоки могут взаимодействовать между собой. Для предотвращения конфликтов и обеспечения синхронизации используются объекты синхронизации, такие как мьютексы и семафоры.

Мьютекс (Mutex) представляет собой механизм синхронизации, который используется для управления доступом к ресурсам, разделяемым несколькими потоками. Он гарантирует, что только один поток имеет доступ к ресурсу в определенный момент времени. Мьютексы широко используются для синхронизации доступа к общим ресурсам внутри процесса. Например, при доступе к общим структурам данных, файлам или другим критическим ресурсам.

Семафор – это абстрактный объект, который используется для контроля доступа к общему ресурсу в многозадачной среде. В отличие от мьютекса, семафор может позволить нескольким потокам одновременно получать доступ к ресурсу. Он эффективен при управлении ресурсами, которые могут обслуживаться несколькими потоками параллельно. Например, при ограничении числа одновременно работающих потоков в системе.

Windows поддерживает как предпосылочную (preemptive), так и кооперативную многозадачность. Предпосылочная многозадачность означает, что операционная система может вытеснить текущий поток в любой момент и передать управление другому [3].

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	10380
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	27.38	20.52	6:40:24	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.

Помимо вышеупомянутой функциональности, пользователю доступна функция остановки процессов (см. рисунок 2).

Process Name	Memory (MB)	CPU (%)	CPU Time (h:mm:ss)	Threads	PID
msedgeview2.exe	18.59	77.94	8:10:32	245	12220
msedgeview2.exe	15.33	45.00	8:10:32	158	4784
MoNotificationUsr.exe	12.17	27.27	8:9:34	185	8256
msedgeview2.exe	25.99	36.00	8:9:0	281	3704
msedgeview2.exe	12.20	88.89	8:9:0	235	6392
RuntimeBroker.exe	39.34	147.17	8:8:22	528	300
DllHost.exe	18.26	70.91	8:8:22	268	8464
sihost.exe	32.17	117.02	8:6:6	536	11332
explorer.exe	93.86	107.35	8:6:5	2537	1812
SearchHost.exe	86.28	63.08	8:6:4	1598	9980
StartMenuExperienceHost.exe	31.52	58.28	8:6:4	727	9560
Widgets.exe	33.73	106.25	8:6:3	788	6256
ShellExperienceHost.exe	33.59	56.36	8:6:3	501	6320
msedgeview2.exe	8.74	166.67	7:29:6	1187	7936
msedgeview2.exe	9.18	400.00	7:29:6	144	10556
msedgeview2.exe	3.19	152.52	7:29:5	347	7016
msedgeview2.exe	4.02	89.72	7:29:5	288	7084
msedgeview2.exe	0.22	320.00	7:29:5	160	11692
msedgeview2.exe	6.36	71.07	7:29:5	319	11896
DllHost.exe	9.80	36.36	6:55:18	126	11764
VCpkgSvc.exe	78.76	114.89	6:42:51	151	11216
Lab3.exe	29.36	437.76	0:0:36	184	10724
mspaint.exe	113.80	135.42	0:0:20	619	7728

Processes	Threads	Memory
59	35436	1555

Stop process

Рисунок 2 – Подлежащий удалению процесс

Нажав на название процесса, пользователь выделяет соответствующий процесс для удаления. По срабатыванию кнопки Stop process происходит остановка процесса и удаление его из списка активных процессов. Результат работы можно увидеть на рисунке 3.

Activity Monitor Analogue					
Process Name	Memory (MB)	CPU (%)	CPU Time (h:...	Threads	PID
msedgeview.exe	13.97	49.70	8:17:42	376	7032
msedgeview2.exe	18.59	78.57	8:17:42	245	12220
msedgeview2.exe	15.33	55.00	8:17:42	158	4784
MoNotificationUs.exe	12.64	36.36	8:16:44	185	8256
msedgeview2.exe	25.99	36.00	8:16:10	281	3704
msedgeview2.exe	12.20	88.89	8:16:10	235	6392
RuntimeBroker.exe	38.90	141.79	8:15:32	518	300
DllHost.exe	18.30	66.67	8:15:32	266	8464
sihost.exe	32.76	122.86	8:13:17	544	11332
Widgets.exe	33.89	105.45	8:13:14	784	6256
msedgeview2.exe	8.88	166.09	7:36:16	1185	7936
msedgeview2.exe	9.27	600.00	7:36:16	144	10556
msedgeview2.exe	3.22	151.43	7:36:15	347	7016
msedgeview2.exe	4.02	89.72	7:36:15	288	7084
msedgeview2.exe	0.22	320.00	7:36:15	160	11692
msedgeview2.exe	6.36	71.07	7:36:15	319	11896
DllHost.exe	9.85	45.45	7:2:28	125	11764
VCpkgSrv.exe	78.76	119.15	6:50:2	151	11216
explorer.exe	181.16	141.43	0:5:58	2285	5340
SearchHost.exe	117.73	53.51	0:5:55	1642	5700
StartMenuExperienceHost.exe	103.61	70.11	0:5:55	714	11232
ShellExperienceHost.exe	55.88	200.00	0:5:54	504	11640
Lab3.exe	29.08	423.21	0:0:41	185	9184

Processes	Threads	Memory
57	34764	1690

Stop process

Рисунок 3 – Результат остановки процесса

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 в Win32 API [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/memory/memory-management>.

[2] Процессы и потоки в Win32 API [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads>

[3] Синхронизация процессов в Win32 API [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/interprocess-synchronization>.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Файл Lab4.cpp

```
#pragma once

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

#ifdef 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"%0.2f", 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 / 100000000;

        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);

        cbNeeded = 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;
}

```