

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  
Подвальников А.С.

Проверил:  
Гриценко Н.Ю.

Минск 2023

## СОДЕРЖАНИЕ

1 Цели работы.....	3
2 Краткие теоретические сведения.....	4
3 Полученные результаты .....	5
Выводы .....	6
Список использованных источников .....	7
Приложение А .....	9

## **1 ЦЕЛИ РАБОТЫ**

Изучить механизмы управления процессами в операционной системе Windows. Изучить операции по приостановке, возобновлению и завершению процессов, а также оценку их эффективности. Реализовать приложение для отслеживания и управления процессами в операционной системе Windows, что позволит осуществлять операции по приостановке, возобновлению и завершению процессов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс Windows API (Application Programming Interface) представляет собой набор функций, предоставляемых операционной системой Windows для разработки приложений. Эти функции позволяют взаимодействовать с ОС на низком уровне, осуществляя такие действия, как управление окнами, обработка сообщений, работа с файлами и устройствами.

**Порождение процессов и потоков:** Для порождения нового процесса в Win32 API используется функция `CreateProcess`. Она создает новый процесс и возвращает его идентификатор (PID). Созданный процесс может быть независимым или связанным с родительским процессом. Для порождения нового потока внутри процесса используется функция `CreateThread`. Она создает новый поток и возвращает его дескриптор. Потоки внутри процесса выполняются параллельно и могут иметь различные задачи и приоритеты.

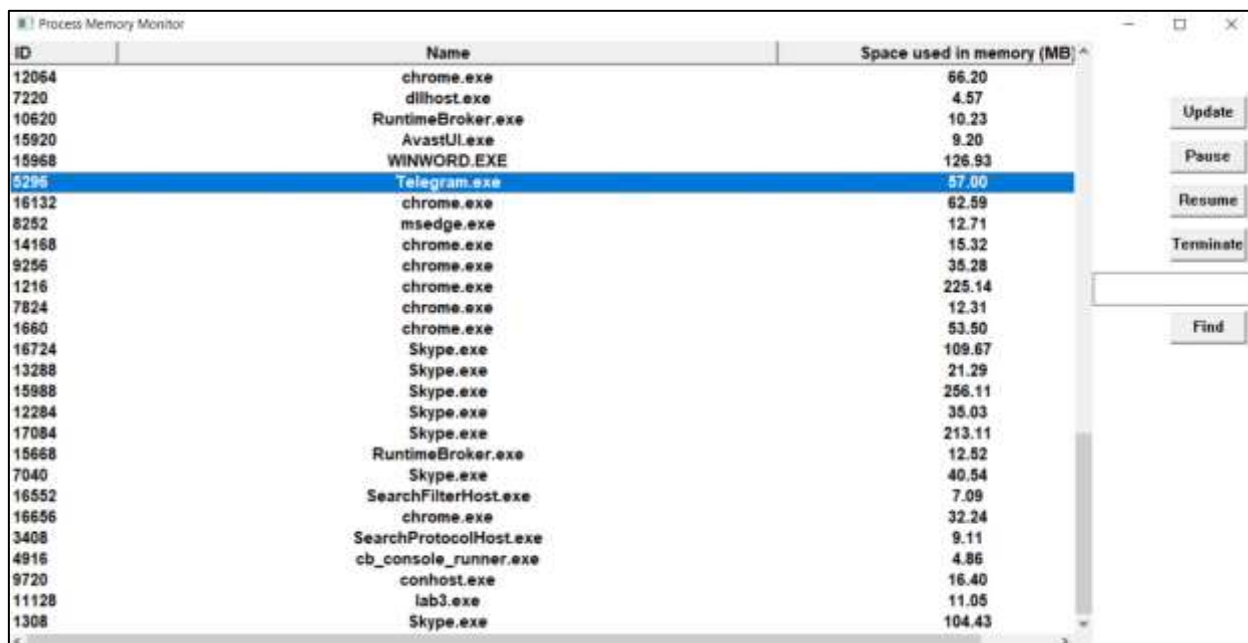
**Завершение процессов и потоков:** Для завершения процесса в Win32 API используется функция `TerminateProcess`. Она принимает дескриптор процесса и код завершения и принудительно завершает процесс. Завершение процесса может быть вызвано различными причинами, такими как успешное выполнение, ошибка или запрос пользователя. При завершении процесса освобождаются все его ресурсы. Для завершения потока внутри процесса используется функция `ExitThread`. Она завершает текущий поток и освобождает его ресурсы. При завершении потока можно указать код завершения, который может быть использован другими потоками или процессами для обработки.

**Изменение приоритетов процессов и потоков:** Для изменения приоритета процесса в Win32 API используется функция `SetPriorityClass`. Она принимает дескриптор процесса и новый приоритет и изменяет приоритет процесса. Приоритет процесса определяет, как много процессорного времени будет выделено данному процессу в сравнении с другими процессами. Высокий приоритет может ускорить выполнение процесса, но может также привести к ухудшению производительности других процессов. Для изменения приоритета потока внутри процесса используется функция `SetThreadPriority`. Она принимает дескриптор потока и новый приоритет и изменяет приоритет потока. Приоритет потока определяет, как много процессорного времени будет выделено данному потоку в сравнении с другими потоками внутри процесса.

**Исследование эффективности:** Для исследования эффективности процессов и потоков в Win32 API можно использовать различные функции для измерения времени выполнения, использования процессора и других ресурсов. Например, функция `GetProcessTimes` позволяет получить информацию о времени выполнения процесса, а функция `GetThreadTimes` - о времени выполнения потока. Эти функции могут быть полезны при профилировании и оптимизации процессов и потоков для достижения лучшей производительности и эффективности системы.

### 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

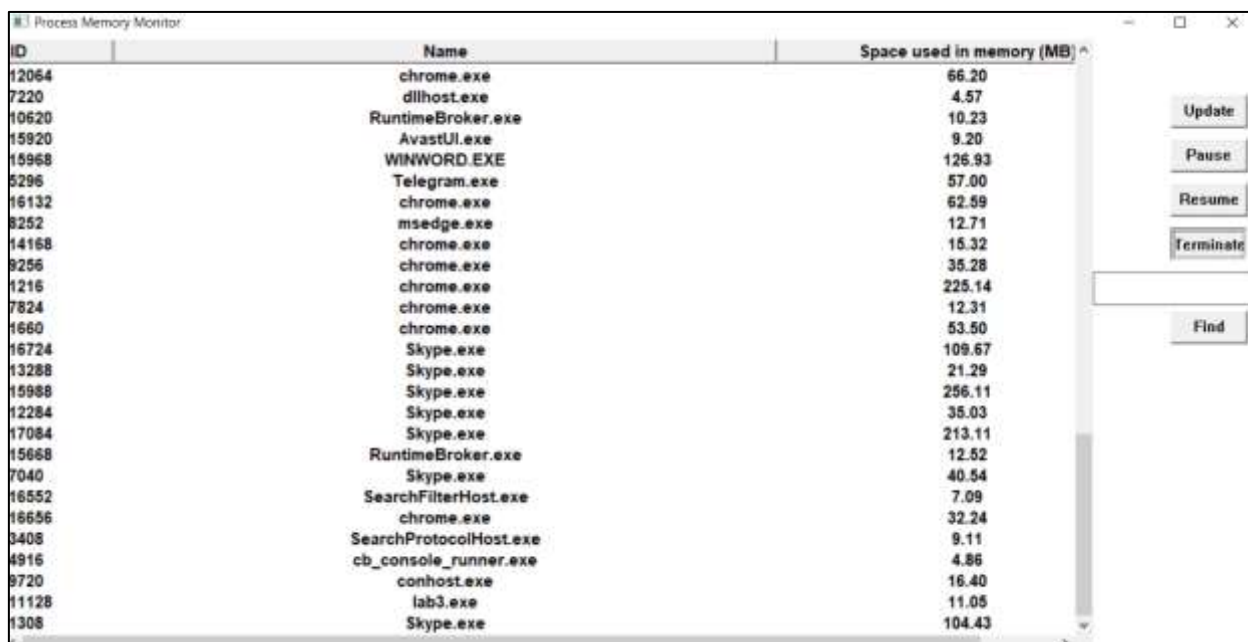
В ходе выполнения лабораторной работы было реализовано приложение для отслеживания и управления процессами в операционной системе Windows, которое позволяет осуществлять операции по приостановке, возобновлению и завершению процессов. Результат работы программы показан на рисунках 3.1, 3.2 и 3.3



The screenshot shows a window titled "Process Memory Monitor". It contains a table with three columns: "ID", "Name", and "Space used in memory (MB)". The table lists various running processes. The process "Telegram.exe" with ID 5296 is highlighted in blue. To the right of the table, there are buttons for "Update", "Pause", "Resume", "Terminate", and "Find".

ID	Name	Space used in memory (MB)
12064	chrome.exe	66.20
7220	dllhost.exe	4.57
10620	RuntimeBroker.exe	10.23
15920	AvastUI.exe	9.20
15968	WINWORD.EXE	126.93
5296	Telegram.exe	57.00
16132	chrome.exe	62.59
8252	msedge.exe	12.71
14168	chrome.exe	15.32
9256	chrome.exe	35.28
1216	chrome.exe	225.14
7824	chrome.exe	12.31
1660	chrome.exe	53.50
16724	Skype.exe	109.67
13288	Skype.exe	21.29
15988	Skype.exe	256.11
12284	Skype.exe	35.03
17084	Skype.exe	213.11
15668	RuntimeBroker.exe	12.52
7040	Skype.exe	40.54
16552	SearchFilterHost.exe	7.09
16656	chrome.exe	32.24
3408	SearchProtocolHost.exe	9.11
4916	cb_console_runner.exe	4.86
9720	conhost.exe	16.40
11128	lab3.exe	11.05
1308	Skype.exe	104.43

Рисунок 3.1 – Результат работы программ (1)



This screenshot is identical to the one in Figure 3.1, showing the same list of processes in the "Process Memory Monitor" window. The "Telegram.exe" process (ID 5296) is still highlighted.

ID	Name	Space used in memory (MB)
12064	chrome.exe	66.20
7220	dllhost.exe	4.57
10620	RuntimeBroker.exe	10.23
15920	AvastUI.exe	9.20
15968	WINWORD.EXE	126.93
5296	Telegram.exe	57.00
16132	chrome.exe	62.59
8252	msedge.exe	12.71
14168	chrome.exe	15.32
9256	chrome.exe	35.28
1216	chrome.exe	225.14
7824	chrome.exe	12.31
1660	chrome.exe	53.50
16724	Skype.exe	109.67
13288	Skype.exe	21.29
15988	Skype.exe	256.11
12284	Skype.exe	35.03
17084	Skype.exe	213.11
15668	RuntimeBroker.exe	12.52
7040	Skype.exe	40.54
16552	SearchFilterHost.exe	7.09
16656	chrome.exe	32.24
3408	SearchProtocolHost.exe	9.11
4916	cb_console_runner.exe	4.86
9720	conhost.exe	16.40
11128	lab3.exe	11.05
1308	Skype.exe	104.43

Рисунок 3.2 – Результат работы программ (2)

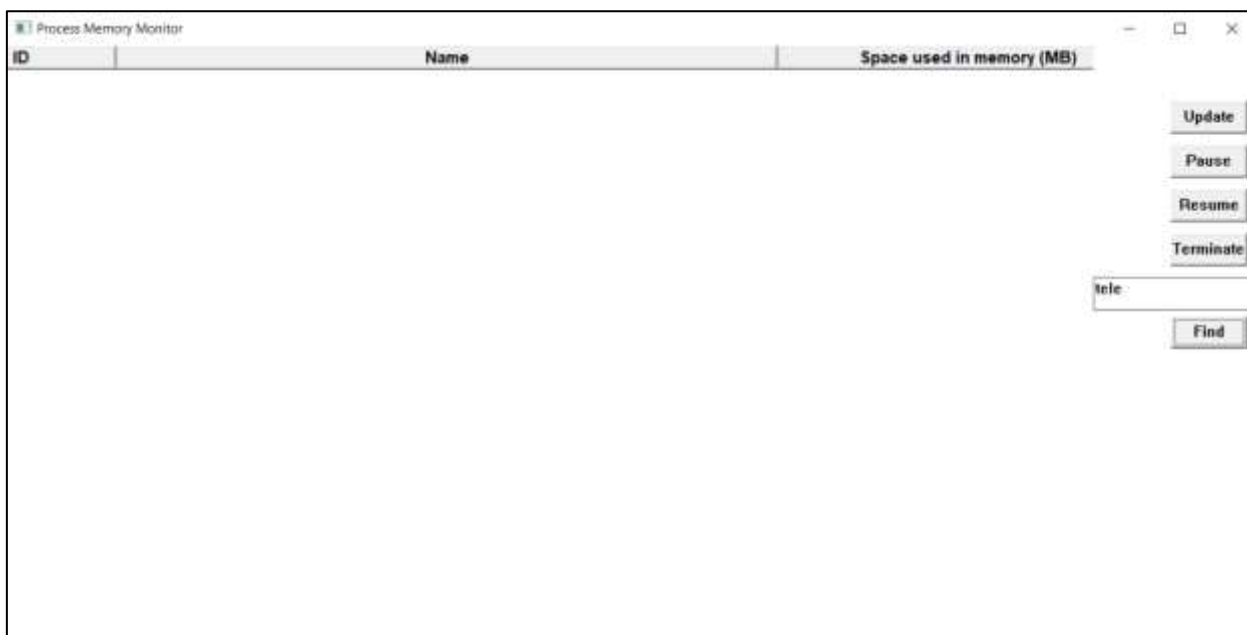


Рисунок 3.3 – Результат работы программ (3)

## **ВЫВОДЫ**

В ходе выполнения данной лабораторной работы были изучены механизмы управления процессами в операционной системе Windows. Изучены операции по приостановке, возобновлению и завершению процессов, а также оценка их эффективности. Реализовано приложение для отслеживания и управления процессами в операционной системе Windows, которое позволяет осуществлять операции по приостановке, возобновлению и завершению процессов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ил.
- [2] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess> – Дата доступа 10.10.2023
- [3] [Электронный ресурс]. – Режим доступа: <http://winapi.freetechsecrets.com/win32/WIN32TerminateProcess.htm> – Дата доступа 10.10.2023
- [4] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-resumethread> – Дата доступа 23.09.2023
- [5] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-suspendthread> – Дата доступа 23.09.2023
- [6] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/procthread/suspending-thread-execution> – Дата доступа 11.10.2023



**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Листинг кода**

**Main.cpp**

```
#include <windows.h>
#include <WinBase.h>
#include <psapi.h>
#include <vector>
#include <string>
#include <commctrl.h>
#include <shlwapi.h>
#include <algorithm>

#define CLASS_NAME "MyWindowClass"
#define IDC_PROCESS_LISTVIEW 101
#define IDC_UPDATE_BUTTON 102
#define IDC_PAUSE_BUTTON 200
#define IDC_RESUME_BUTTON 201
#define IDC_TERMINATE_BUTTON 202
#define IDC_PROCESS_SEARCH_EDIT 300
#define IDC_PROCESS_SEARCH_BUTTON 301

typedef NTSTATUS (NTAPI* NtSuspendProcess) (IN HANDLE ProcessHandle);
typedef NTSTATUS (NTAPI* NtResumeProcess) (IN HANDLE ProcessHandle);

NtSuspendProcess pfnNtSuspendProcess = NULL;
NtResumeProcess pfnNtResumeProcess = NULL;
HWND hwndEdit;
std::string searchQuery;

int selectedPID = -1;

std::string GetProcessName(DWORD processId) {
    std::string name;
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
        PROCESS_VM_READ, FALSE, processId);
    if (hProcess != nullptr) {
        char buffer[MAX_PATH];
        if (GetModuleFileNameExA(hProcess, NULL, buffer, MAX_PATH)) {
            name = PathFindFileNameA(buffer);
        }
        CloseHandle(hProcess);
    }
    return name;
}

std::string toLowerCase(const std::string& str) {
```

```

        std::string lowerStr(str);
        std::transform(lowerStr.begin(), lowerStr.end(), lowerStr.begin(),
            [](unsigned char c){ return std::tolower(c); });
        return lowerStr;
    }

void GetProcessMemoryUsage(DWORD processId, SIZE_T& workingSetSize,
    SIZE_T& privateUsage) {
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
        PROCESS_VM_READ, FALSE, processId);
    if (hProcess) {
        PROCESS_MEMORY_COUNTERS pmc;
        if (GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc))) {
            workingSetSize = pmc.WorkingSetSize;
            privateUsage = pmc.PagefileUsage;
        }
        CloseHandle(hProcess);
    }
}

void AddProcessesToListView(HWND listView, std::string searchQuery) {
    DWORD aProcesses[1024], cbNeeded, cProcesses;
    EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded);
    cProcesses = cbNeeded / sizeof(DWORD);

    LVITEM lvI;
    lvI.mask = LVIF_TEXT;
    lvI.state = 0;
    lvI.stateMask = 0;

    char buffer[1024];
    int itemCount = 0;

    std::string searchQueryLower = toLowerCase(searchQuery);

    for (DWORD i = 0; i < cProcesses; i++) {
        SIZE_T workingSetSize;
        SIZE_T privateUsage;

        GetProcessMemoryUsage(aProcesses[i], workingSetSize,
            privateUsage);

        std::string processName = GetProcessName(aProcesses[i]);

        // Filter by searchQuery
        if(processName.empty() || (searchQueryLower[0] != '\\0' &&
            toLowerCase(processName).find(searchQueryLower) == std::string::npos ))
            continue;

        lvI.iItem = itemCount;

        lvI.iSubItem = 0;
    }
}

```

```

        wsprintf(buffer, "%d", aProcesses[i]);
        lvI.pszText = buffer;
        ListView_InsertItem(listView, &lvI);

        lvI.iSubItem = 1;
        strncpy(buffer, processName.c_str(), sizeof(buffer) /
sizeof(buffer[0]));
        ListView_SetItemText(listView, itemCount, 1, buffer);

        lvI.iSubItem = 2;
        sprintf_s(buffer, sizeof(buffer), "%.2lf", (double)workingSetSize
/ (1024 * 1024));
        ListView_SetItemText(listView, itemCount, 2, buffer);

        ++itemCount;
    }
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam) {
    static HWND listView = NULL;
    static HWND button = NULL;

    switch (uMsg) {
    case WM_CREATE:
        {
            INITCOMMONCONTROLSEX icex;
            icex.dwICC = ICC_LISTVIEW_CLASSES | ICC_STANDARD_CLASSES;
            InitCommonControlsEx(&icex);

            DWORD listViewStyles = WS_CHILD | WS_VISIBLE | LVS_REPORT |
LVS_EDITLABELS;

            listView = CreateWindowEx(0, WC_LISTVIEW, "", listViewStyles,
0, 0, 500, 500, hwnd, (HMENU)IDC_PROCESS_LISTVIEW,
GetModuleHandle(NULL), NULL);

            // Устанавливаем стиль "выделение полной строки"
            ListView_SetExtendedListViewStyle(listView,
LVS_EX_FULLROWSELECT);

            HFONT hFont = CreateFont(18, 0, 0, 0, FW_BOLD, FALSE, FALSE,
FALSE, ANSI_CHARSET,
                                OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN, "Arial");

            SendMessage(listView, WM_SETFONT,
reinterpret_cast<WPARAM>(hFont), TRUE);

            LVCOLUMN lvc;
            lvc.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;

```

```

        lvc.fmt = LVCFMT_CENTER;

        lvc.iSubItem = 0;
        lvc.pszText = (LPCSTR)"ID"; // type cast to LPCSTR
        lvc.cx = 100;
        ListView_InsertColumn(listView, 0, &lvc);

        lvc.iSubItem = 1;
        lvc.pszText = (LPCSTR)"Name"; // type cast to LPCSTR
        lvc.cx = 600;
        ListView_InsertColumn(listView, 1, &lvc);

        lvc.iSubItem = 2;
        lvc.pszText = (LPCSTR)"Space used in memory (MB)"; // type
cast to LPCSTR
        lvc.cx = 345;
        ListView_InsertColumn(listView, 2, &lvc);

        button = CreateWindow("BUTTON", "Update", WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON,
                                1055, 0, 70, 30, hwnd,
(HMENU) IDC_UPDATE_BUTTON, GetModuleHandle(NULL), NULL);

        // Create Pause button
        CreateWindow("BUTTON", "Pause", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                                1055, 60, 70, 30, hwnd,
(HMENU) IDC_PAUSE_BUTTON, GetModuleHandle(NULL), NULL);

        // Create Resume button
        CreateWindow("BUTTON", "Resume", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                                1055, 100, 70, 30, hwnd,
(HMENU) IDC_RESUME_BUTTON, GetModuleHandle(NULL), NULL);

        // Create Terminate button
        CreateWindow("BUTTON", "Terminate", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                                1055, 140, 70, 30, hwnd,
(HMENU) IDC_TERMINATE_BUTTON, GetModuleHandle(NULL), NULL);

        hwndEdit = CreateWindow("EDIT", "", WS_CHILD | WS_VISIBLE |
WS_BORDER | ES_MULTILINE | ES_AUTOVSCROLL | ES_AUTOHSCROLL,
                                1055, 100, 500, 70, hwnd,
(HMENU) IDC_PROCESS_SEARCH_EDIT,
                                GetModuleHandle(NULL), NULL);

        CreateWindow("BUTTON", "Find", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
                                1155, 275, 60, 30, hwnd, (HMENU) IDC_PROCESS_SEARCH_BUTTON,
                                GetModuleHandle(NULL), NULL);
    }
    break;

```

```

case WM_SIZE:
{
    int buttonXPos;
    int buttonYPos = 50;

    buttonXPos = LOWORD(lParam)-80;
    SetWindowPos(button, NULL, buttonXPos, buttonYPos, 70, 30,
SWP_NOZORDER);

    buttonYPos += 40;
    SetWindowPos(GetDlgItem(hwnd, IDC_PAUSE_BUTTON), NULL, buttonXPos,
buttonYPos, 70, 30, SWP_NOZORDER);

    buttonYPos += 40;
    SetWindowPos(GetDlgItem(hwnd, IDC_RESUME_BUTTON), NULL, buttonXPos,
buttonYPos, 70, 30, SWP_NOZORDER);

    buttonYPos += 40;
    SetWindowPos(GetDlgItem(hwnd, IDC_TERMINATE_BUTTON), NULL, buttonXPos,
buttonYPos, 70, 30, SWP_NOZORDER);

    int searchFieldXPos = buttonXPos - 70;
    int searchButtonXPos = buttonXPos;
    SetWindowPos(hwndEdit, NULL, searchFieldXPos, buttonYPos + 40, 150,
30, SWP_NOZORDER);
    SetWindowPos(GetDlgItem(hwnd, IDC_PROCESS_SEARCH_BUTTON), NULL,
searchButtonXPos, buttonYPos + 60 + 15, 70, 30, SWP_NOZORDER);

    // Update the size of the ListView to fill the window
    SetWindowPos(listView, NULL, 0, 0, LOWORD(lParam) - 150,
HIWORD(lParam), SWP_NOZORDER);

    return 0;
}

case WM_NOTIFY:
{
    switch (((LPMHDR)lParam)->code)
    {
        case LVN_ITEMCHANGED: // This notification code is sent to a ListView
when an item changes
            NMLISTVIEW* pnmv = (NMLISTVIEW*)lParam;
            // Ensure this is the notification message we are interested in
            if (pnmv && ((pnmv->uOldState & LVIS_SELECTED) != (pnmv->uNewState
& LVIS_SELECTED)))
            {
                int i = pnmv->iItem;
                BOOL selected = pnmv->uNewState & LVIS_SELECTED;
                if (selected)
                {
                    // The item has just been selected, do something with it
                    CHAR buffer[256];

```

```

        ListView_GetItemText(listView, i, 0, buffer, 256);
        selectedPID = atoi(buffer);
    }
    else
    {
        // The item has just been deselected, do something with it
        selectedPID = -1; // Reset to default value to indicate no
process is selected
    }
}
break;
}
break;
}

case WM_DRAWITEM:
{
    DRAWITEMSTRUCT* pDIS = (DRAWITEMSTRUCT*)lParam;
    if (pDIS->itemAction & ODA_SELECT) {
        COLORREF oldTextColor;
        if (pDIS->itemState & ODS_SELECTED) {
            oldTextColor = SetTextColor(pDIS->hDC, RGB(255, 255,
255));
            SetBkColor(pDIS->hDC, RGB(0, 0, 255));
        } else {
            oldTextColor = SetTextColor(pDIS->hDC, RGB(0, 0, 0));
            SetBkColor(pDIS->hDC, RGB(125, 30, 125));
        }
        // Override the default system behaviour to draw the text
background
        ExtTextOut(pDIS->hDC, pDIS->rcItem.left, pDIS->rcItem.top,
ETO_OPAQUE, &pDIS->rcItem, NULL, 0, NULL);
        // Draw the text
        DrawText(pDIS->hDC, (LPCSTR)pDIS->itemData, -1, &pDIS-
>rcItem, DT_CENTER);

        // Restore the old text color
        SetTextColor(pDIS->hDC, oldTextColor);
    }
    break;
}

case WM_COMMAND:
{
    int wmId = LOWORD(wParam);

    switch (wmId) {
    case IDC_UPDATE_BUTTON:
        ListView_DeleteAllItems(listView);
        AddProcessesToListView(listView, "");
        break;
    case IDC_PAUSE_BUTTON:

```

```

        if (selectedPID != -1) {
            HANDLE hProcess = OpenProcess(PROCESS_SUSPEND_RESUME, FALSE,
selectedPID);
            if (hProcess) {
                pfnNtSuspendProcess(hProcess);
                CloseHandle(hProcess);
            }
        }
        break;
    case IDC_RESUME_BUTTON:
        if (selectedPID != -1) {
            HANDLE hProcess = OpenProcess(PROCESS_SUSPEND_RESUME, FALSE,
selectedPID);
            if (hProcess) {
                pfnNtResumeProcess(hProcess);
                CloseHandle(hProcess);
            }
        }
        break;
    case IDC_TERMINATE_BUTTON:
        if (selectedPID != -1) {
            HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, FALSE,
selectedPID);
            if (hProcess) {
                TerminateProcess(hProcess, 0);
                CloseHandle(hProcess);
            }
        }
        break;
    case IDC_PROCESS_SEARCH_BUTTON:
    {
        char searchQuery[256];
        GetWindowText(hwndEdit, searchQuery, sizeof(searchQuery));
        // Search function
        ListView_DeleteAllItems(listView);
        // Applies searchQuery
        AddProcessesToListView(listView, searchQuery);
    }
    break;
    default:
        break;
    }
    return 0;
}

case WM_DESTROY:
    PostQuitMessage(0);
    return 0;
}

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

```

```

}

int WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, INT nShowCmd) {
    WNDCLASS wc = { 0 };
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = CLASS_NAME;

    if (!RegisterClass(&wc)) {
        MessageBox(NULL, "Could not register window class", "Error",
MB_OK);
        return 0;
    }

    HWND hwnd = CreateWindowEx(0, CLASS_NAME, "Process Memory Monitor",
                                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                                CW_USEDEFAULT, CW_USEDEFAULT,
                                CW_USEDEFAULT, NULL, NULL, hInstance,
NULL);

    if (!hwnd) {
        MessageBox(NULL, "Could not create window", "Error", MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nShowCmd);
    UpdateWindow(hwnd);

    HMODULE hNtdll = GetModuleHandle("ntdll.dll");
    if (hNtdll) {
        pfnNtSuspendProcess = (NtSuspendProcess)GetProcAddress(hNtdll,
"NtSuspendProcess");
        pfnNtResumeProcess = (NtResumeProcess)GetProcAddress(hNtdll,
"NtResumeProcess");
    }

    if (!pfnNtSuspendProcess || !pfnNtResumeProcess) {
        MessageBox(NULL, "Could not load function pointers", "Error",
MB_OK);
        return 0;
    }

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

```