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

ОТЧЁТ
к лабораторной работе №4
на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (WINDOWS). ПОРОЖДЕНИЕ, ЗАВЕРШЕНИЕ, ИЗМЕНЕНИЕ ПРИОРИТЕТОВ ПРОЦЕССОВ И ПОТОКОВ, ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ.

Выполнил студент гр.153502 Миненков Я.А.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является разработка приложение для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы.

В качестве задачи необходимо реализовать возможность выбор процесса из списка всех запущенных и его дальнейшее завершение, приостановка, возобновление.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API предоставляет множество функций и возможностей для создания, управления и мониторинга процессов и потоков в операционной системе *Windows*.

Для создания нового процесса используется функция *CreateProcess*. Она позволяет указать исполняемый файл, командную строку, права доступа и множество других параметров.

Для взаимодействия с уже существующим процессом используется функция *OpenProcess*. Она позволяет получить дескриптор процесса и работать с ним.

Для мониторинга запущенных процессов можно использовать функцию *CreateToolhelp32Snapshot*. Функция *CreateToolhelp32Snapshot* - это одна из функций *Win32 API*, предназначенных для создания снимка (*snapshot*) определенных компонентов операционной системы *Windows*, таких как процессы, потоки и модули. Она является частью библиотеки "*Tool Help Library*" и предоставляет информацию о системных ресурсах для мониторинга и управления процессами.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

При запуске приложения в элемент *ListBox* загружаются все системные процессы. Для работы с процессами снизу представлены кнопки *Terminate Process*, *Suspend Threads*, *Resume Threads* (см. рисунок 1).

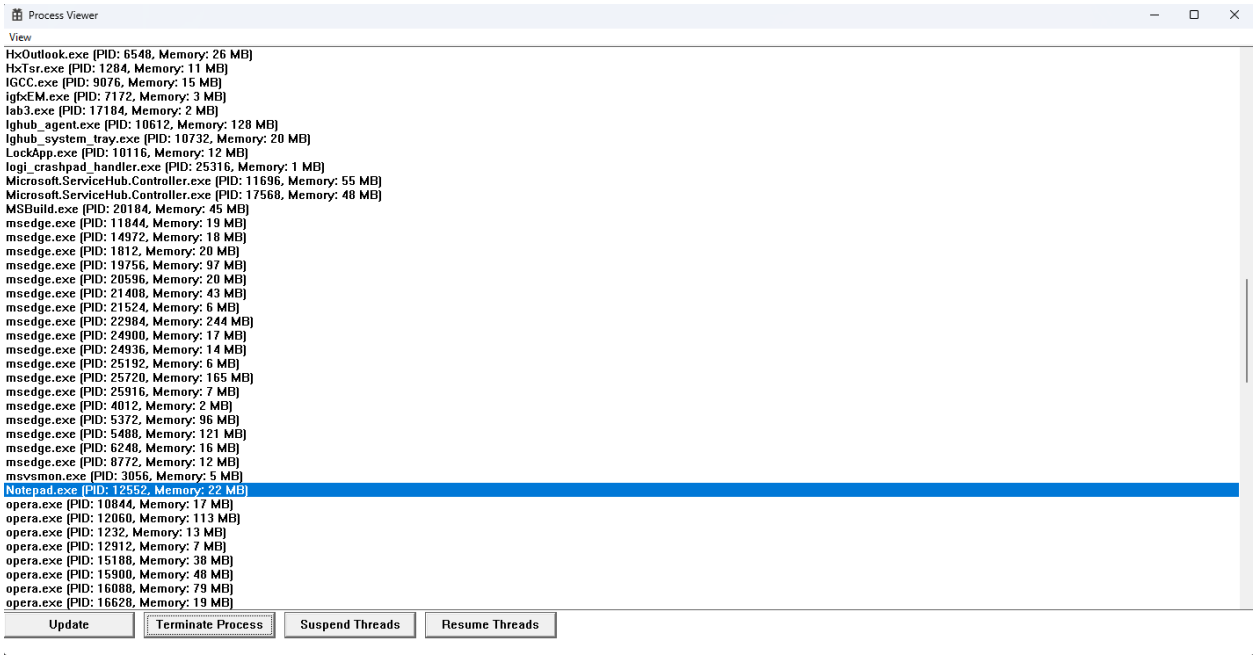

Рисунок 1 – Главное окно приложения

Для примера, завершим процесс *Notepad.exe* с помощью кнопки Terminate и нажмем кнопку *Update*, чтобы обновить список запущенных процессов. Как можно заметить, поле *Notepad.exe* исчезло из элемента *ListBox* (см. рисунок 2).

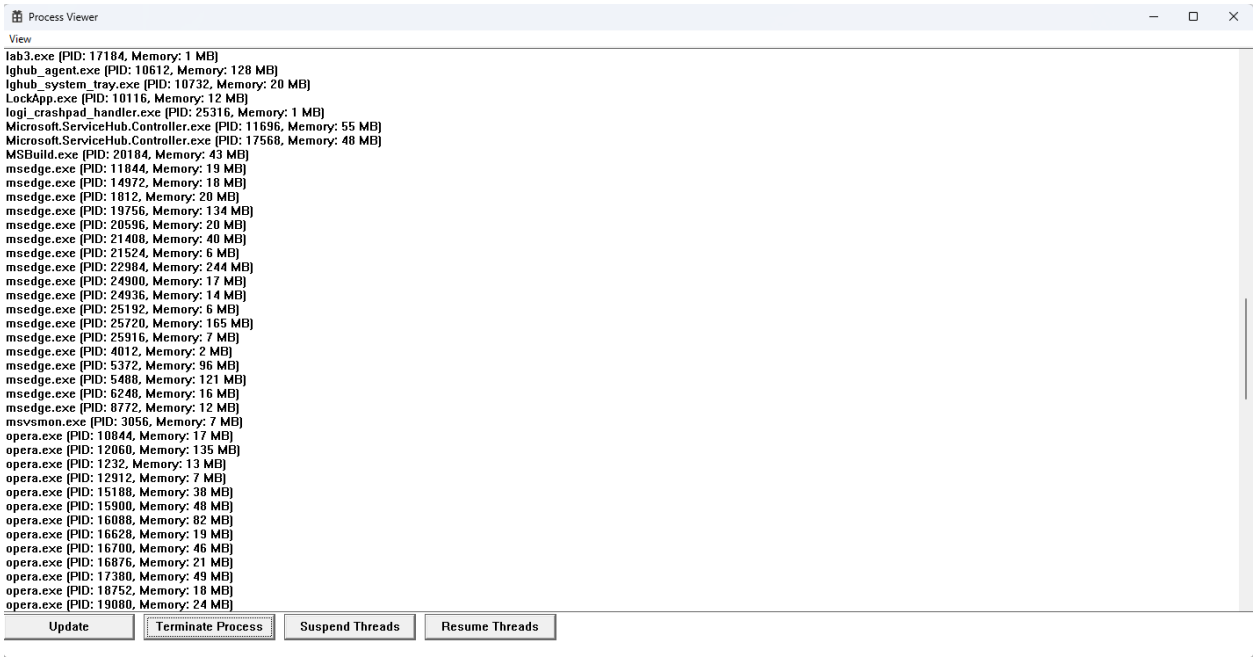

Рисунок 2 – Результат выполнения Terminate

Нажатие кнопки *Suspend Threads* для выбранного потока, приостанавливает все его процессы.

Нажатие кнопки *Resume Threads* для выбранного потока, восстанавливает все его процессы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функция TerminateProcess (processthreadsapi.h) - Win32 apps [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess

[2] Функция SuspendThread (processthreadsapi.h) - Win32 apps [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-suspendthread

[3] Функция ResumeThread (processthreadsapi.h) - Win32 apps [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-resumethread

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл lab4.cpp

```
#include "framework.h"
#include "lab4.h"
#include <windows.h>
#include <CommCtrl.h>
#include <psapi.h>
#include <cstdio>
#include <thread>
#include <tlhelp32.h>


#define SIZE 1024
#define ID_UPDATE_BUTTON 1
#define ID_TERMINATE_BUTTON 2
#define ID_SUSPEND_BUTTON 3
#define ID_RESUME_BUTTON 4
#define IDM_SYSTEM_MEMORY 1100
#define IDM_SHOW_MEMORY_INFO 1101

DWORD processes[SIZE];

HMENU hMenu;
HWND hWnd;
HWND listBoxControl = NULL;
HWND terminateButton = NULL;
HWND suspendButton = NULL;
HWND updateButton = NULL;
HWND resumeButton = NULL;


MEMORYSTATUSEX memInfo;
bool showMemoryInfo = false;

#define MAX_LOADSTRING 100

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

#define IDM_CODE_SAMPLES 2001

ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);

void updateProcessList();
void StartUpdateThread();
bool SuspendProcess(DWORD processId);
bool ResumeProcess(DWORD processId);
DWORD getProcessId();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR    lpCmdLine,
    _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
```

```
    UNREFERENCED_PARAMETER(lpCmdLine);

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB3, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL          hAccelTable          =          LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB3));

    MSG msg;

    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB3));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB3);
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,  0,  CW_USEDEFAULT,  0,  nullptr,  nullptr,  hInstance,
nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

    SetWindowText(hWnd, L"Process Viewer");
```

```
    hMenu = CreateMenu();
    HMENU hSubMenu = CreatePopupMenu();
    AppendMenu(hSubMenu, MF_STRING, IDM_SYSTEM_MEMORY, L"System Memory");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hSubMenu, L"View");
    SetMenu(hWnd, hMenu);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
    {
        ZeroMemory(processes, sizeof(processes));

        RECT clientRect;
        GetClientRect(hWnd, &clientRect);

        listBoxControl = CreateWindowW(
            L"LISTBOX",
            L"",
            WS_CHILD | WS_VISIBLE | LBS_STANDARD,
            0, 0, clientRect.right - clientRect.left, clientRect.bottom -
clientRect.top - 50,
            hWnd,
            NULL,
            hInst,
            NULL);

        terminateButton = CreateWindowW(
            L"BUTTON",
            L"Terminate Process",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
            0, clientRect.bottom - clientRect.top - 50, 150, 30,
            hWnd,
            (HMENU)ID_TERMINATE_BUTTON,
            hInst,
            NULL);

        updateButton = CreateWindowW(
            L"BUTTON",
            L"Update",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
            160,
            clientRect.bottom - clientRect.top - 50,
            150,
            30,
            hWnd,
            (HMENU)ID_UPDATE_BUTTON,
            hInst,
            NULL);

        suspendButton = CreateWindowW(
            L"BUTTON",
            L"Suspend Threads",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
```

```
            480, clientRect.bottom - clientRect.top - 70, 150, 30,
            hWnd,
            (HMENU)ID_SUSPEND_BUTTON,
            hInst,
            NULL);

        resumeButton = CreateWindowW(
            L"BUTTON",
            L"Resume Threads",
            WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
            480, clientRect.bottom - clientRect.top - 70, 150, 30,
            hWnd,
            (HMENU)ID_RESUME_BUTTON,
            hInst,
            NULL);

        updateProcessList();
    }
    break;
    case WM_SIZE:
    {
        RECT clientRect;
        GetClientRect(hWnd, &clientRect);

        SetWindowPos(listBoxControl,    0,    0,    0,    clientRect.right    -
clientRect.left, clientRect.bottom - clientRect.top - 50, SWP_NOMOVE);
        SetWindowPos(terminateButton,    0,    160,    clientRect.bottom    -
clientRect.top - 50, 150, 30, SWP_NOSIZE);
        SetWindowPos(updateButton, 0, 0, clientRect.bottom - clientRect.top -
50, 150, 30, SWP_NOSIZE);
        SetWindowPos(suspendButton, 0, 320, clientRect.bottom - clientRect.top
- 50, 150, 30, SWP_NOSIZE);
        SetWindowPos(resumeButton, 0, 480, clientRect.bottom - clientRect.top
- 50, 150, 30, SWP_NOSIZE);

    }
    break;

    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        switch (wmId)
        {
        case IDM_SYSTEM_MEMORY:
        {
            MEMORYSTATUSEX memoryStatus;
            memoryStatus.dwLength = sizeof(memoryStatus);
            GlobalMemoryStatusEx(&memoryStatus);

            wchar_t message[256];
            swprintf_s(message, L"Total physical memory: %lld MB\nAvailable
physical memory: %lld MB",
                memoryStatus.ullTotalPhys      /      (1024      *      1024),
memoryStatus.ullAvailPhys / (1024 * 1024));

            MessageBox(hWnd, message, L"Memory Info", MB_OK);
        }
        break;
        case ID_UPDATE_BUTTON:
            StartUpdateThread();
            /*updateProcessList();*/
            break;
        case IDM_EXIT:
```

```
            DestroyWindow(hWnd);
            break;
        case ID_TERMINATE_BUTTON:
        {
            int index = SendMessage(listBoxControl, LB_GETCURSEL, 0, 0);

            DWORD processId = getProcessId();

            if (index != LB_ERR)
            {
                HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
processId);
                if (hProcess != NULL)
                {
                    BOOL result = TerminateProcess(hProcess, 1);
                    CloseHandle(hProcess);
                    if (!result)
                    {
                        TCHAR buffer[256];
                        SendMessage(listBoxControl, LB_GETTEXT, index,
(LPARAM)buffer);

                        TCHAR msg[512];
                        _stprintf_s(msg, _countof(msg), TEXT("Не удалось
завершить процесс: %s"), buffer);
                        MessageBox(NULL, msg, TEXT("Ошибка"), MB_OK |
MB_ICONERROR);
                    }
                    else
                    {
                        TCHAR buffer[256];
                        SendMessage(listBoxControl, LB_GETTEXT, index,
(LPARAM)buffer);
                        updateProcessList();
                    }
                }
                else
                {
                    TCHAR buffer[256];
                    SendMessage(listBoxControl, LB_GETTEXT, index,
(LPARAM)buffer);

                    TCHAR msg[512];
                    _stprintf_s(msg, _countof(msg), TEXT("Не удалось получить
доступ к процессу: %s"), buffer);
                    MessageBox(NULL, msg, TEXT("Ошибка"), MB_OK |
MB_ICONERROR);
                }
            }
        }
        break;
        case ID_SUSPEND_BUTTON:
        {
            int index = SendMessage(listBoxControl, LB_GETCURSEL, 0, 0);

            DWORD processId = getProcessId();

            if (SuspendProcess(processId)) {
                MessageBox(NULL, L"Процесс успешно приостановлен", L"Успех",
MB_ICONINFORMATION);
            }
            else {
```

```
                MessageBox(NULL,    L"Не    удалось    приостановить    процесс",
L"Ошибка", MB_ICONERROR);
            }
        }
        break;
        case ID_RESUME_BUTTON:
        {
            int index = SendMessage(listBoxControl, LB_GETCURSEL, 0, 0);

            DWORD processId = getProcessId();

            if (ResumeProcess(processId)) {
                MessageBox(NULL,  L"Процесс  успешно  возобновлен",  L"Успех",
MB_ICONINFORMATION);
            }
            else {
                MessageBox(NULL, L"Не удалось возобновить процесс", L"Ошибка",
MB_ICONERROR);
            }
        }
        break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }
    break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

void updateProcessList()
{
    DWORD cbNeeded;
    DWORD cProcesses;
    unsigned int i;

    ZeroMemory(processes, sizeof(processes));

    if (!EnumProcesses(processes, sizeof(processes), &cbNeeded))
    {
        // Обработка ошибок
        return;
    }

    cProcesses = cbNeeded / sizeof(DWORD);

    SendMessage(listBoxControl, LB_RESETCONTENT, 0, 0);

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

            if (NULL != hProcess)
            {
```

```
                char szProcessName[MAX_PATH] = "<unknown>";

                if    (GetModuleBaseNameA(hProcess,    NULL,    szProcessName,
sizeof(szProcessName) / sizeof(char)))
                {
                    wchar_t buffer[256];

                    PROCESS_MEMORY_COUNTERS_EX pmc;
                    if                          (GetProcessMemoryInfo(hProcess,
(PROCESS_MEMORY_COUNTERS*)&pmc, sizeof(pmc)))
                    {
                        unsigned long long memoryMB = pmc.PrivateUsage / 1024
/ 1024;

                        swprintf_s(buffer, 256, L"%S (PID: %u, Memory: %llu
MB)", szProcessName, processes[i], memoryMB);
                    }
                    else
                    {
                        swprintf_s(buffer, 256, L"%S (PID: %u)", szProcessName,
processes[i]);
                    }

                    SendMessageW(listBoxControl,       LB_ADDSTRING,       0,
(LPARAM)buffer);
                }

                CloseHandle(hProcess);
            }
        }
    }
}

void StartUpdateThread()
{
    std::thread updateThread(updateProcessList);
    updateThread.detach();
}


bool SuspendProcess(DWORD processId) {
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processId);
    if (hProcess == NULL) {
        return false; // Не удалось открыть процесс
    }

    HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
    if (hThreadSnapshot == INVALID_HANDLE_VALUE) {
        CloseHandle(hProcess);
        return false; // Не удалось создать снимок потоков
    }

    THREADENTRY32 threadEntry;
    threadEntry.dwSize = sizeof(THREADENTRY32);
    if (Thread32First(hThreadSnapshot, &threadEntry)) {
        do {
            if (threadEntry.th32OwnerProcessID == processId) {
                HANDLE  hThread  =  OpenThread(THREAD_SUSPEND_RESUME,  FALSE,
threadEntry.th32ThreadID);
                if (hThread != NULL) {
                    SuspendThread(hThread);
                    CloseHandle(hThread);
                }
```

```
            }
        } while (Thread32Next(hThreadSnapshot, &threadEntry));
    }

    CloseHandle(hThreadSnapshot);
    CloseHandle(hProcess);
    return true;
}

bool ResumeProcess(DWORD processId) {
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processId);
    if (hProcess == NULL) {
        return false; // Не удалось открыть процесс
    }

    HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
    if (hThreadSnapshot == INVALID_HANDLE_VALUE) {
        CloseHandle(hProcess);
        return false; // Не удалось создать снимок потоков
    }

    THREADENTRY32 threadEntry;
    threadEntry.dwSize = sizeof(THREADENTRY32);
    if (Thread32First(hThreadSnapshot, &threadEntry)) {
        do {
            if (threadEntry.th32OwnerProcessID == processId) {
                HANDLE hThread = OpenThread(THREAD_SUSPEND_RESUME, FALSE,
threadEntry.th32ThreadID);
                if (hThread != NULL) {
                    ResumeThread(hThread);
                    CloseHandle(hThread);
                }
            }
        } while (Thread32Next(hThreadSnapshot, &threadEntry));
    }

    CloseHandle(hThreadSnapshot);
    CloseHandle(hProcess);
    return true;
}


DWORD getProcessId()
{
    int index = SendMessage(listBoxControl, LB_GETCURSEL, 0, 0);

    if (index != LB_ERR)
    {
        TCHAR buffer[256];
        SendMessage(listBoxControl,         LB_GETTEXT,         (WPARAM)index,
(LPARAM)buffer);

        DWORD processId;
        swscanf_s(buffer, L"%*[^:]: %u", &processId);

        return processId;
    }

    return -1;
}
```