Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

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. GDI

Студент: гр.153502

Логвинович М.В.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc147333809)

[2 Теоретические сведения 4](#_Toc147333810)

[3 Результат выполнения программы 5](#_Toc147333811)

[Список использованных источников 6](#_Toc147333812)

[Приложение А 7](#_Toc147333813)

# **1 ЦЕЛЬ РАБОТЫ**

Создать приложение для мониторинга и управления системной памятью, отображающее текущее потребление памяти различными процессами.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление памятью и вводом-выводом являются фундаментальными аспектами в разработке программного обеспечения. В операционной системе Windows, как и в других современных ОС, существует множество механизмов и API для управления памятью и вводом-выводом данных. Управление памятью включает в себя выделение и освобождение памяти, а также управление ресурсами, чтобы избежать утечек памяти и повысить производительность.

PSAPI (Process Status API) предоставляет функции и структуры для получения информации о процессах и их модулях. Он позволяет мониторить процессы и получать информацию о загруженных модулях в адресном пространстве процесса. Чтобы определить эффективность приложения, может потребоваться изучить использование памяти. PSAPI предоставляет функцию GetProcessMemoryInfo для получения сведений об использовании памяти процесса [1, 2].

ToolHelp32 предоставляет функции для получения информации о процессах, потоках и модулях в системе. Он был популярным средством для мониторинга процессов до появления PSAPI. Однако стоит отметить, что многие функции ToolHelp32 считаются устаревшими в более современных версиях Windows и могут не поддерживаться [3].

Для получения списка процессов используется функция Process32First, которая извлекает сведения о первом процессе, обнаруженном в системном снимке процессов, который создается при помощи функции CreateToolhelp32Snapshot. Структура LPPROCESSENTRY32 описывает запись из списка процессов, находящихся в системном адресном пространстве при получении снимка.

# **3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Результатом выполнения лабораторной работы является графическое приложение для мониторинга и управления системной памятью, отображающее текущее потребление памяти различными процессами. Также реализован механизм сортировки элементов по идентификатору процесса при нажатии на название любого из столбцов. Результат работы программы показан на рисунке 1.

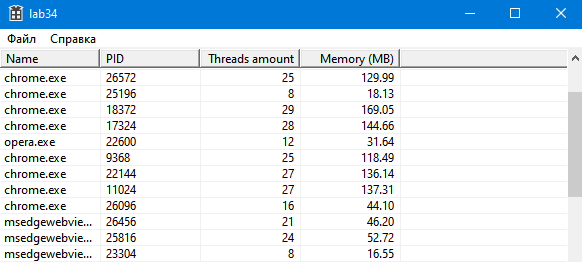


Рисунок 1 – Результат работы программы

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/psapi/psapi-reference

[2] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/psapi/collecting-memory-usage-information-for-a-process

[3] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/toolhelp/taking-a-snapshot-and-viewing-processes

# **ПРИЛОЖЕНИЕ А**

**Исходный код программы**

**Файл lab.cpp**

#include <string>

#include "framework.h"

#include "lab34.h"

#include "tlhelp32.h"

#include <commctrl.h>

#include "psapi.h"

#define MAX\_LOADSTRING 100

#pragma comment(lib, "comctl32.lib")

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

#define ID\_LISTVIEW 100

HWND hWndListView;

bool isAscending = true;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

HWND addButton(HWND hWnd, int x, int y, int width, int height, int id, LPCWSTR buttonText);

int CALLBACK CompareFunc(LPARAM lParam1, LPARAM lParam2, LPARAM sortParam)

{

int pid1 = static\_cast<int>(lParam1);

int pid2 = static\_cast<int>(lParam2);

if (isAscending)

return pid1 - pid2; // Compare PIDs in ascending order

else

return pid2 - pid1; // Compare PIDs in descending order

}

HWND CreateListView(HWND hwndParent)

{

RECT rcClient; // The parent window's client area.

GetClientRect(hwndParent, &rcClient);

InitCommonControls();

// Create the list-view window in report view with label editing enabled.

hWndListView = CreateWindow(WC\_LISTVIEW,

L"", // window name

// child window, first column of list view is left-aligned

WS\_CHILD | LVS\_REPORT | WS\_VISIBLE,

0, 0,

rcClient.right - rcClient.left,

rcClient.bottom - rcClient.top,

hwndParent,

(HMENU)ID\_LISTVIEW,

hInst,

NULL);

ListView\_SetExtendedListViewStyle(hWndListView, LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES | LVS\_EX\_HEADERDRAGDROP);

return (hWndListView);

}

// InitListViewColumns: Adds columns to a list-view control.

// hWndListView: Handle to the list-view control.

// Returns TRUE if successful, and FALSE otherwise.

BOOL InitListViewColumns(HWND hWndListView)

{

WCHAR szText[256]; // Temporary buffer.

LVCOLUMN lvc;

int iCol;

// Initialize the LVCOLUMN structure.

// The mask specifies that the format, width, text,

// and subitem members of the structure are valid.

lvc.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

// set which members contain valid information

// Add the columns.

for (iCol = 0; iCol < 4; iCol++)

{

lvc.iSubItem = iCol;

lvc.pszText = szText;

lvc.cx = 100; // Width of column in pixels.

if (iCol < 2)

lvc.fmt = LVCFMT\_LEFT; // Left-aligned column.

else

lvc.fmt = LVCFMT\_RIGHT; // Right-aligned column.

// Load the names of the column headings from the string resources.

LoadString(hInst,

IDS\_STRING104 + iCol,

szText,

sizeof(szText) / sizeof(szText[0]));

// Insert the columns into the list view.

if (ListView\_InsertColumn(hWndListView, iCol, &lvc) == -1)

return FALSE;

}

return TRUE;

}

PROCESS\_MEMORY\_COUNTERS GetMemoryInfo(DWORD processID)

{

HANDLE hProcess;

PROCESS\_MEMORY\_COUNTERS pmc; // structure the contains memory info for a process

// flags: certain process info, read info in process

// which are required for getprocessmemoryinfo call

hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, processID);

if (hProcess != NULL) {

// Retrieving information about the memory usage of the specified process.

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

{

CloseHandle(hProcess);

return pmc;

}

else {

CloseHandle(hProcess);

}

}

return pmc;

}

// get running processes using Toolhelp32 library

void GetRunningProcesses(HWND hWndListView) {

HANDLE hProcessSnap;

PROCESSENTRY32 pe32; // structure describes entry from processes list

// Take a snapshot of all processes in the system.

// th32cs\_snapprocess flag includes all system processes in the snapshot, 0 is id of process (used with other flags)

hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

if (hProcessSnap == INVALID\_HANDLE\_VALUE) {

return;

}

// Set the size of the structure before using it.

pe32.dwSize = sizeof(PROCESSENTRY32);

// Delete all items in the list view.

ListView\_DeleteAllItems(hWndListView);

// Now walk the snapshot of processes and populate the list view.

if (Process32First(hProcessSnap, &pe32)) { // extract first process info fron snap to pe32 structure

PROCESS\_MEMORY\_COUNTERS pmc;

pmc.cb = sizeof(PROCESS\_MEMORY\_COUNTERS);

int i = 0;

do {

// Add information about the process to the list view.

LVITEM lvItem;

lvItem.mask = LVIF\_TEXT | LVIF\_PARAM;

lvItem.iItem = i; // Index of the new item

lvItem.iSubItem = 0; // Index of the sub-item (first column).

lvItem.pszText = pe32.szExeFile; // Process name.

lvItem.lParam = pe32.th32ProcessID;

int nIndex = ListView\_InsertItem(hWndListView, &lvItem);

// Display the PID in the second column.

WCHAR pidBuffer[16];

swprintf\_s(pidBuffer, L"%lu", pe32.th32ProcessID);

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = nIndex;

lvItem.iSubItem = 1; // Index of the sub-item (second column).

lvItem.pszText = pidBuffer;

ListView\_SetItem(hWndListView, &lvItem);

// Display the number of threads in the third column.

WCHAR threadsBuffer[16];

swprintf\_s(threadsBuffer, L"%lu", pe32.cntThreads);

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = nIndex;

lvItem.iSubItem = 2; // Index of the sub-item (third column).

lvItem.pszText = threadsBuffer;

ListView\_SetItem(hWndListView, &lvItem);

// Display process memory in fourth column

pmc = GetMemoryInfo(pe32.th32ProcessID);

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = nIndex;

lvItem.iSubItem = 3; // Index of the sub-item (fourth column).

double memoryUsageMB = static\_cast<double>(pmc.WorkingSetSize) / 1048576.0; // 1 megabyte = 1048576 bytes

WCHAR buffer[32];

swprintf\_s(buffer, L"%.2f", memoryUsageMB);

lvItem.pszText = buffer;

ListView\_SetItem(hWndListView, &lvItem);

} while (Process32Next(hProcessSnap, &pe32)); // retrieve info about next process in snapshot

}

// Clean up the snapshot object.

CloseHandle(hProcessSnap);

}

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_LAB34, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int) msg.wParam;

}

// ФУНКЦИЯ: MyRegisterClass()

// ЦЕЛЬ: Регистрирует класс окна.

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\_LAB34));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB34);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

// КОММЕНТАРИИ:

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, 600, 800, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

CreateListView(hWnd);

InitListViewColumns(hWndListView);

GetRunningProcesses(hWndListView);

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

EndPaint(hWnd, &ps);

}

break;

case WM\_NOTIFY:

{

LPNMHDR tc;

LPNMLISTVIEW pnmlv;

DWORD subitm;

tc = (LPNMHDR)lParam;

if ((tc->hwndFrom == hWndListView) && (tc->code == LVN\_COLUMNCLICK)) {

pnmlv = (LPNMLISTVIEW)lParam;

subitm = pnmlv->iSubItem;

ListView\_SortItems(hWndListView, CompareFunc, subitm);

isAscending = !isAscending;

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}