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 WIN32. ОРГАНИЗАЦИЯ И КОНТРОЛЬ АСИНХРОННЫХ ОПЕРАЦИЙ ВВОДА-ВЫВОДА. ОТОБРАЖЕНИЕ ФАЙЛОВ В ПАМЯТЬ**

Студент В. А. Сидорко

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 3](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучить принципы управления памятью и вводом-выводом, расширенные возможности ввода-вывода Windows, функции API подсистемы памяти Win32, организацию и контроль асинхронных операций ввода-вывода, отображение файлов в память. Для достижения цели будет создано оконное приложение для мониторинга и управления системной памятью, отображающее текущее потребление памяти различными процессами.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Возможность управлять памятью процесса является одной из ключевых функций *Win32 API*. Память в контексте операционной системы *Windows* представляет собой виртуальное адресное пространство, которое выделяется для каждого процесса. Приватная область памяти отображает физическую память на виртуальное адресное пространство процесса и служит для хранения кода, данных и стека процесса. Доля рабочего набора процесса (*Working Set*) представляет собой подмножество приватной области памяти, которое активно используется процессом в данный момент. Для получения информации о процессах в данной работе была использована библиотека *Psapi.h*, предоставляющая следующие функции:

1 *EnumProcesses* – позволяет получить список идентификаторов всех запущенных процессов, заполняя ими массив значений *DWORD*.

2 *OpenProcess* – в качестве параметра принимает идентификатор процесса и возвращает дескриптор соответствующего процесса. По окончании работы с дескриптором процесса вызывается функция *CloseHandle*.

3 *EnumProcessModules* – используется для получения дескрипторов всех модулей, указанного процесса. Каждый процесс состоит из одного или нескольких модулей. Модуль – это исполняемый файл или библиотека *DLL*.

4 *GetProcessMemoryInfo* – в качестве входных данных принимает дескриптор процесса и заполняет структуру *PROCESS\_MEMORY\_COUNTERS* сведениями о статистике памяти для процесса. используется для получения информации о памяти, используемой указанным процессом. Она принимает три параметра: дескриптор процесса, указатель на структуру PROCESS\_MEMORY\_COUNTERS, которая будет содержать информацию о памяти, используемой процессом, и размер этой структуры в байтах.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение для мониторинга и управления системной памятью, отображающее текущее потребление памяти различными процессами (рисунок 1).

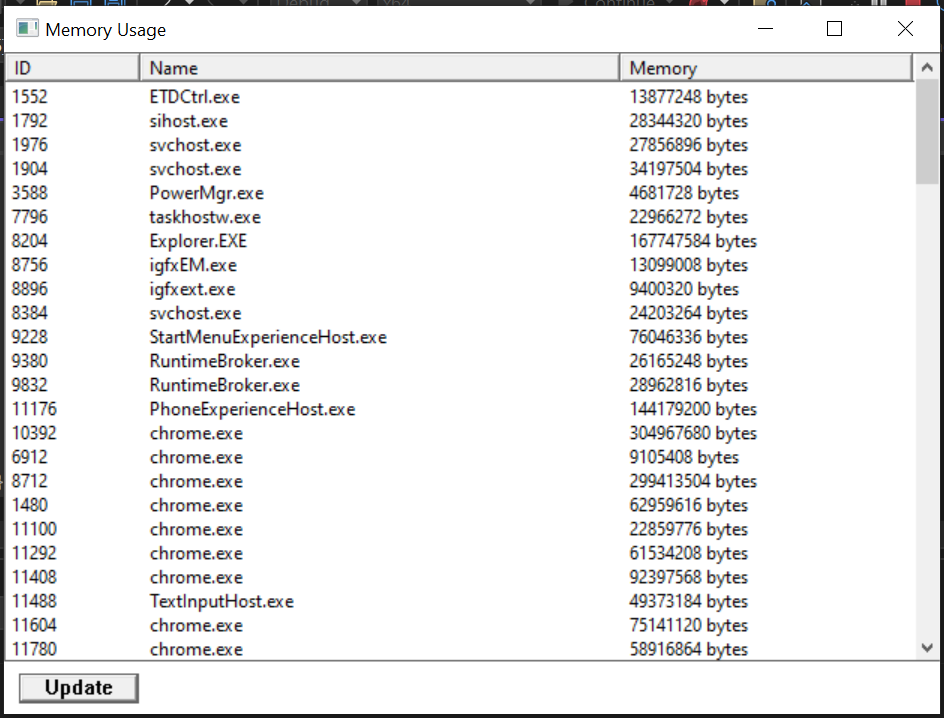


Рисунок 1 – Окно приложения

В окне приложения отражены идентификатор процесса (*ID*), его название (*Name*), и объем используемой памяти в байтах (*Memory*). Кнопка «*Update*» обновляет данные о запущенных процессах.

# ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены и применены на практике знания о функции API подсистемы памяти Win32. В результате проделанной работы было создано оконное приложение для мониторинга и управления системной памятью, отображающее текущее потребление памяти различными процессами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Управления памятью Windows [Электронный ресурс]. – Режим доступа : learn.microsoft.com/ru-ru/windowshardware/drivers/kernel/managing-memory-for-drivers.
2. Щупак, Ю.А. Win32 API. Разработка приложений для Windows /Ю.А. Щупак. – СПб : Питер, 2008. – 592 с.
3. Psapi, документация Microsoft [Электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/en-us/windows/win32/api/psapi/>.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл main.cpp

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

#include <Windows.h>

#include <psapi.h>

#include "commctrl.h"

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure);

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

void UpdateProcessesInfo(HWND hWnd);

LPWSTR GetProcessInfo(DWORD processId);

void AddWidgets(HWND hWnd);

HINSTANCE hInstance;

HWND hWndListView;

HWND hWndUpdateButton;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd) {

hInstance = hInst;

WNDCLASS MainClass = NewWindowClass(hInst, L"Main Window Class", WndProc);

if (!RegisterClass(&MainClass)) { return -1; }

MSG MainMessage = { 0 };

CreateWindow(L"Main Window Class", L"Memory Usage",

WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 50, 640, 480, NULL, NULL,

hInst, NULL);

while (GetMessage(&MainMessage, NULL, NULL, NULL)) {

TranslateMessage(&MainMessage);

DispatchMessage(&MainMessage);

}

}

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure) {

WNDCLASS windowClass = {};

windowClass.hInstance = hInst;

windowClass.lpszClassName = name;

windowClass.lpfnWndProc = procedure;

return windowClass;

}

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

{

switch (uMsg)

{

case WM\_COMMAND:

if (lParam == (LPARAM)hWndUpdateButton) {

UpdateProcessesInfo(hWnd);

}

break;

case WM\_CREATE:

AddWidgets(hWnd);

UpdateProcessesInfo(hWnd);

break;

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

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

}

void AddWidgets(HWND hWnd)

{

InitCommonControls();

hWndListView = CreateWindow(WC\_LISTVIEW, L"",

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

0, 0, 625, 406, hWnd, NULL, hInstance, 0);

ListView\_SetExtendedListViewStyle(hWndListView, LVS\_EX\_FULLROWSELECT);

LVCOLUMN lvcolId;

lvcolId.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT;

lvcolId.pszText = LPWCH(L"ID");

lvcolId.cx = 90;

ListView\_InsertColumn(hWndListView, 0, &lvcolId);

LVCOLUMN lvcolName;

lvcolId.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT;

lvcolName.pszText = LPWCH(L"Name");

ListView\_InsertColumn(hWndListView, 1, &lvcolName);

ListView\_SetColumnWidth(hWndListView, 1, 320);

LVCOLUMN lvcolMemory;

lvcolId.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT;

lvcolMemory.pszText = LPWCH(L"Memory");

ListView\_InsertColumn(hWndListView, 2, &lvcolMemory);

ListView\_SetColumnWidth(hWndListView, 2, 195);

hWndUpdateButton = CreateWindow(L"BUTTON", L"Update",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 414, 80, 20, hWnd, NULL, hInstance, NULL);

}

void UpdateProcessesInfo(HWND hWnd) {

// array of process ids

DWORD aProcesses[1024];

// number of bytes returned in process ids array

DWORD cbNeeded;

// number of process ids returned

DWORD cProcesses;

if (!EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded)) {

MessageBox(hWnd, L"Failed to get process ids!", L"Error", MB\_OK);

return;

}

cProcesses = cbNeeded / sizeof(DWORD);

ListView\_DeleteAllItems(hWndListView);

UINT count = 0;

for (UINT i = 0; i < cProcesses; i++) {

if (aProcesses[i] != 0) {

TCHAR szProcessName[MAX\_PATH] = L"<unknown>";

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, aProcesses[i]);

PROCESS\_MEMORY\_COUNTERS pmc;

WCHAR szInfo[MAX\_PATH];

WCHAR szWorkingSet[100];

if (hProcess != NULL) {

HMODULE hModule;

DWORD cbNeeded;

if (EnumProcessModules(hProcess, &hModule, sizeof(hModule), &cbNeeded)) {

GetModuleBaseName(hProcess, hModule, szProcessName,

sizeof(szProcessName) / sizeof(TCHAR));

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

wsprintf(szWorkingSet, TEXT("%lu bytes"), pmc.WorkingSetSize);

}

CloseHandle(hProcess);

LVITEM lvItem;

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = count;

lvItem.iSubItem = 0;

lvItem.pszText = new wchar\_t[0];

ListView\_InsertItem(hWndListView, &lvItem);

wchar\_t buffer[MAX\_PATH];

wsprintf(szInfo, TEXT("%lu"), aProcesses[i]);

ListView\_SetItemText(hWndListView, count, 0, szInfo)

wsprintf(szInfo, TEXT("%s"), szProcessName);

ListView\_SetItemText(hWndListView, count, 1, szInfo);

ListView\_SetItemText(hWndListView, count, 2, szWorkingSet);

count++;

}

}

}

}