

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 Win
32. Организация и контроль асинхронных операций ввода-вывода.
Отображение файлов в память»

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 | 8 |

1 ЦЕЛИ РАБОТЫ

Изучить расширенные возможности управления памятью в Windows с использованием набора API-функций подсистемы памяти Win32. Также изучить расширенные возможности управления вводом-выводом в Windows с использованием набора API-функций подсистемы ввода-вывода Win32. Реализовать приложение, которое эффективно отслеживает и контролирует использование памяти отдельных процессов, и обрабатывает операции ввода-вывода для обновления этих данных.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Application Programming Interface) - это набор функций, предоставленных операционной системой Microsoft Windows для управления и манипуляции элементами операционной системы, такими как окна, файлы, процессы, память и оборудование.

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.

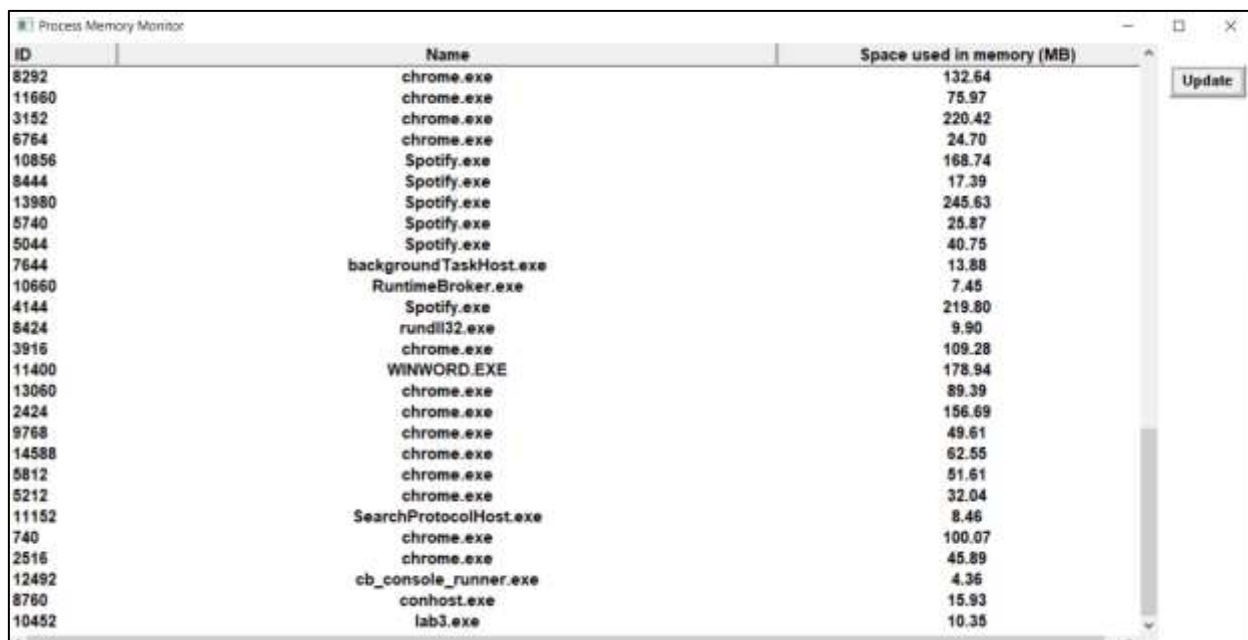
Для работы с памятью процесса в Win32 API используются различные функции. Например, функция `OpenProcess` позволяет получить дескриптор процесса на основе его идентификатора (PID). Получив дескриптор процесса, разработчики могут использовать другие функции, такие как `GetProcessMemoryInfo`, для получения информации о потреблении памяти процессом. Это может быть полезно для мониторинга и оптимизации процессов приложений, управления выделением и освобождением памяти и других задач, связанных с управлением ресурсами.

Кроме работы с памятью, Win32 API также предоставляет различные функции для работы с файлами и оборудованием. Функции чтения и записи файлов позволяют осуществлять операции ввода и вывода данных. Управление файловыми дескрипторами позволяет открывать, закрывать и манипулировать файлами. Мониторинг и манипуляции событиями ввода-вывода позволяют приложениям реагировать на различные события, связанные с оборудованием или файловой системой.

Для осуществления операций ввода и вывода данных в Win32 API используются различные функции. Одна из функций, `GetProcessName`, получает идентификатор процесса и возвращает его имя. Для этого она использует функции Win32 API такие как `OpenProcess`, `GetModuleFileNameExA` и `CloseHandle`. Еще одна функция, такая как `GetProcessMemoryUsage`, используется для извлечения информации о памяти, используемой процессом. Она также использует функции Win32 API, включая `PROCESS_QUERY_INFORMATION` и `PROCESS_VM_READ`, для получения требуемой информации о памяти. Процедура `AddProcessesToListView` используется для представления информации в элементе управления `ListView`. Она получает указатель на `ListView` и добавляет в него список процессов и информацию о памяти.

3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано приложение, которое эффективно отслеживает и контролирует использование памяти отдельных процессов, и обрабатывает операции ввода-вывода для обновления этих данных. Результат работы программы показан на рисунке 3.1



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 processes, including multiple instances of chrome.exe, Spotify.exe, and other system and application processes. An "Update" button is located on the right side of the table.

| ID | Name | Space used in memory (MB) |
|-------|------------------------|---------------------------|
| 8292 | chrome.exe | 132.64 |
| 11660 | chrome.exe | 75.97 |
| 3152 | chrome.exe | 220.42 |
| 6764 | chrome.exe | 24.70 |
| 10856 | Spotify.exe | 168.74 |
| 8444 | Spotify.exe | 17.39 |
| 13980 | Spotify.exe | 245.63 |
| 5740 | Spotify.exe | 25.87 |
| 5044 | Spotify.exe | 40.75 |
| 7644 | backgroundTaskHost.exe | 13.88 |
| 10660 | RuntimeBroker.exe | 7.45 |
| 4144 | Spotify.exe | 219.80 |
| 8424 | rundll32.exe | 9.90 |
| 3916 | chrome.exe | 109.28 |
| 11400 | WINWORD.EXE | 178.94 |
| 13060 | chrome.exe | 89.39 |
| 2424 | chrome.exe | 156.69 |
| 9768 | chrome.exe | 49.61 |
| 14588 | chrome.exe | 62.55 |
| 5812 | chrome.exe | 51.61 |
| 5212 | chrome.exe | 32.04 |
| 11152 | SearchProtocolHost.exe | 8.46 |
| 740 | chrome.exe | 100.07 |
| 2516 | chrome.exe | 45.89 |
| 12492 | cb_console_runner.exe | 4.36 |
| 8760 | conhost.exe | 15.93 |
| 10452 | lab3.exe | 10.35 |

Рисунок 3.1 – Результат работы программ

ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены расширенные возможности управления памятью в Windows с использованием набора API-функций подсистемы памяти Win32. Были изучены расширенные возможности управления вводом-выводом в Windows с использованием набора API-функций подсистемы ввода-вывода Win32. Реализовано приложение, которое эффективно отслеживает и контролирует использование памяти отдельных процессов, и обрабатывает операции ввода-вывода для обновления этих данных.

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/psapi/psapi-functions> – Дата доступа 08.10.2023

[3] [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/controls/common-controls-intro> – Дата доступа 08.10.2023

[4] [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/> – Дата доступа 23.09.2023

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>

#define CLASS_NAME "MyWindowClass"
#define IDC_PROCESS_LISTVIEW 101
#define IDC_UPDATE_BUTTON 102

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)) {
            // Use PathFindFileNameA to get the file name
            name = PathFindFileNameA(buffer);
        }
        CloseHandle(hProcess);
    }
    return name;
}

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) {
    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;

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

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

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

    if(processName.empty())
        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);

        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 = "ID";
        lvc.cx = 100;
        ListView_InsertColumn(listView, 0, &lvc);

        lvc.iSubItem = 1;
        lvc.pszText = "Name";
        lvc.cx = 600;
        ListView_InsertColumn(listView, 1, &lvc);

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

        button = CreateWindow("BUTTON", "Update", WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON,
                                510, 0, 70, 30, hwnd,
(HMENU) IDC_UPDATE_BUTTON, GetModuleHandle(NULL), NULL);

        return 0;
    }

    case WM_SIZE:
        SetWindowPos(listView, NULL, 0, 0, LOWORD(lParam)-90,
HIWORD(lParam),
                        SWP_NOZORDER);
        SetWindowPos(button, NULL, LOWORD(lParam)-80, 20, 70, 30,
                        SWP_NOZORDER);

```

```

        return 0;

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

        // parse the menu selections:
        switch (wmId)
        {
            case IDC_UPDATE_BUTTON:
                ListView_DeleteAllItems(listView);

                AddProcessesToListView(listView);

                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);

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

return 0;
}
```