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 №6

на тему

**НЕКОТОРЫЕ СЛУЖЕБНЫЕ И ТЕХНОЛОГИЧЕСКИЕ ЗАДАЧИ**

Выполнил: студент гр.253505 Павлович В.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc184502714)

[2 Краткие теоретические сведения 4](#_Toc184502715)

[3 Описание работы программы 5](#_Toc184502716)

[3.1 Сбор информации о процессоре и времени работы системы 5](#_Toc184502717)

[3.2 Сбор информации об оперативной памяти 5](#_Toc184502718)

[3.3 Сбор информации о запущенных процессах и кеш-памяти 6](#_Toc184502719)

[3.4 Сбор информации об операционной системе и сети 7](#_Toc184502720)

[Заключение 8](#_Toc184502721)

[Список использованных источников 9](#_Toc184502722)

[ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы 10](#_Toc184502723)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью лабораторной работы является изучение отдельных задач, связанных с конфигурированием, мониторингом, управлением системой, а также технологические аспекты: библиотеки и сборки.

В качестве задачи необходимо разработать утилиту, обеспечивающую сбор информации о системе и отображение ее в удобном виде: аппаратное обеспечение, операционная система, количественные характеристики и т.д.

Состав информации – на усмотрение разработчика, в качестве ориентира – стандартные «системные» сведения, можно расширить или специализировать на конкретном разделе. Источники информации: в основном реестр и/или специализированные системные функции, содержимое файловой системы, WMI, собственные измерения характеристик и т.д.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Реестр – центральная иерархическая база данных, используемая в Windows 98, Windows CE, Windows NT и Windows 2000, используется для хранения сведений, необходимых для настройки системы для одного или нескольких пользователей, приложений и аппаратных устройств.

Реестр содержит сведения, на которые Windows постоянно ссылается во время операции, такие как профили для каждого пользователя, приложения, установленные на компьютере, типы документов, которые могут создаваться, параметры таблицы свойств для папок и значков приложений, оборудование, которое установлено в системе, и используемые порты. [1]

Инструментарий управления Windows (WMI) – это инфраструктура для управления данными и операциями в операционных системах windows. Хотя вы можете создавать скрипты WMI или приложения для автоматизации административных задач на удаленных компьютерах, WMI также предоставляет данные управления другим частям операционной системы и продуктам. Например, System Center Operations Manager или Удаленное управление Windows. [2]

WinHook – это механизм, с помощью которого приложение может перехватывать события, такие как сообщения, действия мыши и нажатия клавиш. Функция, перехватывающая событие определенного типа, называется процедурой перехватчика. Процедура перехватчика может действовать для каждого получаемого события, а затем изменять или отменять событие. [3]

3 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В этом разделе будут рассмотрены основные функции программы, разработанной программы.

## **3.1 Сбор информации о процессоре и времени работы системы**

Функция PrintSystemInfo предназначена для получения основных сведений о системе, таких как архитектура процессора, количество ядер и размер страницы памяти. Для этого используется системный вызов GetSystemInfo [4], который предоставляет прямой доступ к информации об аппаратных характеристиках системы. Эти данные помогают пользователю определить ключевые параметры оборудования.

Функция PrintProcessorFrequency извлекает частоту процессора в мегагерцах из реестра Windows с помощью API-функции RegQueryValueEx [5]. Это позволяет уточнить тактовую частоту процессора, что является важным для оценки производительности.

Функция PrintUptime использует API-функцию GetTickCount [6], которая возвращает время работы системы в миллисекундах. Результат преобразуется в дни, часы, минуты и секунды для удобного представления пользователю.

Пример вывода данной информации представлен на рисунке 3.1.

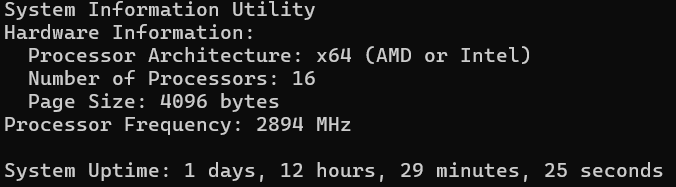


Рисунок 3.1 – Информация о процессоре и времени работы системы

## **3.2 Сбор информации об оперативной памяти**

Функция PrintMemoryInfo использует вызов GlobalMemoryStatusEx для получения информации о состоянии памяти, включая общий объем физической и виртуальной памяти, а также доступные ресурсы. Это позволяет пользователю оценить текущее использование памяти системой.

Пример вывода данной информации представлен на рисунке 3.2.

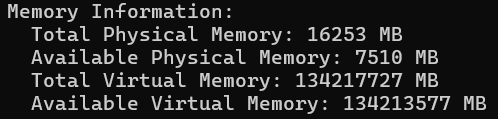


Рисунок 3.2 – Информация об оперативной памяти

## **3.3 Сбор информации о запущенных процессах и кеш-памяти**

Функция PrintProcessesInfo использует CreateToolhelp32Snapshot для создания снимка всех запущенных процессов в системе. С помощью функций Process32First и Process32Next программа формирует список имен активных приложений, что полезно для мониторинга текущей активности системы.

Функция PrintCacheInfo предоставляет информацию о кэшах процессора (L1, L3), включая их размер, тип (инструкционный, данных, унифицированный) и уровень. Для этого используется вызов GetLogicalProcessorInformation [7], который извлекает подробные данные о логических процессорах и связанных с ними кэшах. Дополнительно функция cache\_sum суммирует объемы кэшей каждого уровня, чтобы предоставить общую информацию о кэш-памяти процессора.

Пример вывода данной информации представлен на рисунке 3.3.

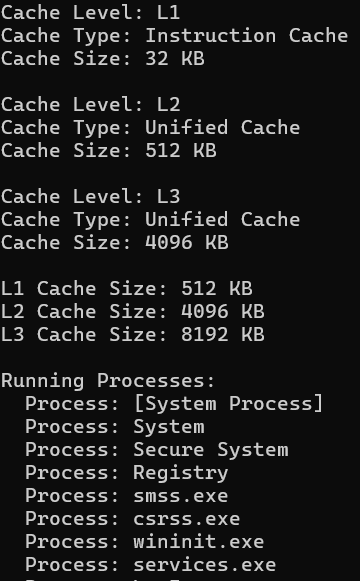


Рисунок 3.3 – Информация о запущенных процессах и кеш-памяти

## **3.4 Сбор информации об операционной системе и сети**

Функция PrintOSVersion предназначена для получения информации о версии операционной системы. Она создает структуру OSVERSIONINFOEX и заполняет ее с помощью вызова GetVersionEx [8]. Если вызов успешен, программа выводит на экран версию ОС, номер сборки и тип системы (NT или другой). Это позволяет пользователю быстро узнать основные характеристики установленной операционной системы.

Функция PrintNetworkInfo собирает данные о сетевых адаптерах с помощью GetAdaptersInfo [9]. Сначала она запрашивает размер буфера, необходимого для хранения информации об адаптерах, а затем выделяет память для этой информации. Если вызов проходит успешно, программа выводит данные о каждом адаптере, включая имя адаптера, MAC-адрес, IP-адрес и описание. Это помогает пользователю получить представление о сетевых подключениях и конфигурации системы.

Пример вывода данной информации представлен на рисунке 3.4.

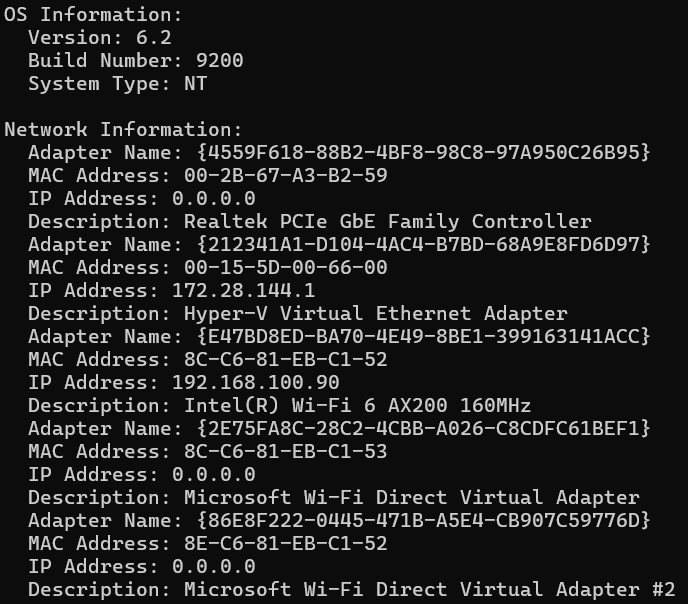


Рисунок 3.4 – Информация об операционной системе и сети

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана утилита для сбора и отображения информации о системе. Программа соответствует требованиям задания и позволяет получить сведения о системных и аппаратных характеристиках компьютера.

Основные данные, которые предоставляет программа, включают архитектуру и количество процессоров, частоту процессора, информацию о кэш-памяти (L1, L2, L3), общий объем и доступное количество оперативной и виртуальной памяти, а также информацию об операционной системе, включая версию, номер сборки и тип системы. Кроме того, программа отображает время непрерывной работы системы (uptime), сведения о сетевых адаптерах (их названия, MAC-адреса и IP-адреса) и список запущенных процессов.

Для получения данных использовались системные функции Windows API, такие как GetSystemInfo, GlobalMemoryStatusEx, GetAdaptersInfo, а также функции работы с реестром для извлечения частоты процессора. Разработанная программа продемонстрировала корректную работу всех компонентов, обеспечивая получение и отображение данных в удобной текстовой форме. Тестирование подтвердило надежность решения и точность предоставляемой информации.

Эта работа позволила изучить применение Windows API для решения служебных и технологических задач, а также использование реестра и системных библиотек для работы с данными операционной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сведения о реестре Windows для опытных пользователей [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/troubleshoot/windows-server/performance/windows-registry-advanced-users.

[2] Инструментарий управления Windows (WMI) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/wmi-start-page.

[3] Обзор перехватчиков – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/winmsg/about-hooks.

[4] Function GetSystemInfo (sysinfoapi.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/

sysinfoapi/nf-sysinfoapi-gets ysteminfo

[5] Функция RegQueryValueEx (winreg.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regqueryva lueexa

[6] функция GetTickCount (sysinfoapi.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/

sysinfoapi/nf-sysinfoapi-getti ckcount.

[7] Функция GetLogicalProcessorInformation (sysinfoapi.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-getl ogicalprocessorinformation.

[8] Функция GetVersionExA (sysinfoapi.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/

sysinfoapi/nf-sysinfoapi-getversionexa.

[9] Функция GetAdaptersInfo (iphlpapi.h) – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/

iphlpapi/nf-iphlpapi-getadaptersinfo.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <tchar.h>

#include <string>

#include <intrin.h>

#include <vector>

#include <iphlpapi.h>

#include <tlhelp32.h>

#include <cstdint>

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

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

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

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

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable: 4996)

void PrintSystemInfo() {

SYSTEM\_INFO si;

GetSystemInfo(&si);

std::cout << "Hardware Information:" << std::endl;

std::cout << " Processor Architecture: ";

switch (si.wProcessorArchitecture) {

case PROCESSOR\_ARCHITECTURE\_AMD64:

std::cout << "x64 (AMD or Intel)" << std::endl;

break;

case PROCESSOR\_ARCHITECTURE\_ARM:

std::cout << "ARM" << std::endl;

break;

case PROCESSOR\_ARCHITECTURE\_ARM64:

std::cout << "ARM64" << std::endl;

break;

case PROCESSOR\_ARCHITECTURE\_IA64:

std::cout << "Intel Itanium-based" << std::endl;

break;

case PROCESSOR\_ARCHITECTURE\_INTEL:

std::cout << "x86" << std::endl;

break;

case PROCESSOR\_ARCHITECTURE\_UNKNOWN:

std::cout << "Unknown architecture" << std::endl;

break;

default:

std::cout << "Undefined architecture" << std::endl;

break;

}

std::cout << " Number of Processors: " << si.dwNumberOfProcessors << std::endl;

std::cout << " Page Size: " << si.dwPageSize << " bytes" << std::endl;

}

void PrintMemoryInfo() {

MEMORYSTATUSEX memInfo;

memInfo.dwLength = sizeof(MEMORYSTATUSEX);

if (GlobalMemoryStatusEx(&memInfo)) {

std::cout << "\nMemory Information:" << std::endl;

std::cout << " Total Physical Memory: " << memInfo.ullTotalPhys / (1024 \* 1024) << " MB" << std::endl;

std::cout << " Available Physical Memory: " << memInfo.ullAvailPhys / (1024 \* 1024) << " MB" << std::endl;

std::cout << " Total Virtual Memory: " << memInfo.ullTotalVirtual / (1024 \* 1024) << " MB" << std::endl;

std::cout << " Available Virtual Memory: " << memInfo.ullAvailVirtual / (1024 \* 1024) << " MB" << std::endl;

}

else {

std::cerr << "Failed to retrieve memory information." << std::endl;

}

}

void PrintUptime() {

DWORD uptime = GetTickCount() / 1000;

DWORD seconds = uptime % 60;

DWORD minutes = (uptime / 60) % 60;

DWORD hours = (uptime / 3600) % 24;

DWORD days = uptime / 86400;

std::cout << "\nSystem Uptime: ";

std::cout << days << " days, " << hours << " hours, " << minutes << " minutes, " << seconds << " seconds" << std::endl;

}

void PrintProcessorFrequency() {

SYSTEM\_INFO si;

GetSystemInfo(&si);

HKEY hKey;

DWORD data, dataSize = sizeof(data);

if (RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, TEXT("HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0"), 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

if (RegQueryValueEx(hKey, TEXT("~MHz"), NULL, NULL, (LPBYTE)&data, &dataSize) == ERROR\_SUCCESS) {

std::cout << "Processor Frequency: " << data << " MHz" << std::endl;

}

RegCloseKey(hKey);

}

else {

std::cerr << "Failed to retrieve processor frequency." << std::endl;

}

}

void PrintOSVersion() {

OSVERSIONINFOEX osInfo;

osInfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);

if (GetVersionEx((OSVERSIONINFO\*)&osInfo)) {

std::cout << "\nOS Information:" << std::endl;

std::cout << " Version: " << osInfo.dwMajorVersion << "." << osInfo.dwMinorVersion << std::endl;

std::cout << " Build Number: " << osInfo.dwBuildNumber << std::endl;

std::cout << " System Type: " << (osInfo.dwPlatformId == VER\_PLATFORM\_WIN32\_NT ? "NT" : "Other") << std::endl;

}

}

void PrintNetworkInfo() {

ULONG bufferSize = 0;

GetAdaptersInfo(NULL, &bufferSize);

PIP\_ADAPTER\_INFO pAdapterInfo = (IP\_ADAPTER\_INFO\*)malloc(bufferSize);

if (GetAdaptersInfo(pAdapterInfo, &bufferSize) == NO\_ERROR) {

PIP\_ADAPTER\_INFO pAdapter = pAdapterInfo;

std::cout << "\nNetwork Information:" << std::endl;

while (pAdapter) {

std::cout << " Adapter Name: " << pAdapter->AdapterName << std::endl;

std::cout << " MAC Address: ";

for (UINT i = 0; i < pAdapter->AddressLength; ++i) {

if (i != 0) std::cout << "-";

printf("%02X", pAdapter->Address[i]);

}

std::cout << std::endl;

std::cout << " IP Address: " << pAdapter->IpAddressList.IpAddress.String << std::endl;

std::cout << " Description: " << pAdapter->Description << std::endl;

pAdapter = pAdapter->Next;

}

}

free(pAdapterInfo);

}

void PrintProcessesInfo() {

PROCESSENTRY32 pe32;

HANDLE hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

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

std::cerr << "Failed to take snapshot of processes." << std::endl;

return;

}

pe32.dwSize = sizeof(PROCESSENTRY32);

if (!Process32First(hProcessSnap, &pe32)) {

std::cerr << "Failed to get first process." << std::endl;

CloseHandle(hProcessSnap);

return;

}

std::cout << "\nRunning Processes:" << std::endl;

do {

std::wcout << " Process: " << pe32.szExeFile << std::endl;

} while (Process32Next(hProcessSnap, &pe32));

CloseHandle(hProcessSnap);

}

void PrintCacheInfo() {

DWORD bufferSize = 0;

GetLogicalProcessorInformation(nullptr, &bufferSize);

std::vector<SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION> buffer(bufferSize / sizeof(SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION));

if (!GetLogicalProcessorInformation(buffer.data(), &bufferSize)) {

std::cerr << "Failed to get processor information.\n";

return;

}

for (const auto& info : buffer) {

if (info.Relationship == RelationCache) {

auto& cache = info.Cache;

std::string cacheType;

switch (cache.Type) {

case CacheData: cacheType = "Data Cache"; break;

case CacheInstruction: cacheType = "Instruction Cache"; break;

case CacheUnified: cacheType = "Unified Cache"; break;

default: cacheType = "Unknown Cache"; break;

}

std::cout << "Cache Level: L" << static\_cast<int>(cache.Level)

<< "\nCache Type: " << cacheType

<< "\nCache Size: " << cache.Size / 1024 << " KB\n\n";

}

}

}

void cache\_sum() {

DWORD bufferSize = 0;

GetLogicalProcessorInformation(nullptr, &bufferSize);

std::vector<SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION> buffer(bufferSize / sizeof(SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION));

if (!GetLogicalProcessorInformation(buffer.data(), &bufferSize)) {

std::cerr << "Failed to retrieve processor information.\n";

return;

}

size\_t l1Cache = 0, l2Cache = 0, l3Cache = 0;

for (const auto& info : buffer) {

if (info.Relationship == RelationCache) {

switch (info.Cache.Level) {

case 1: l1Cache += info.Cache.Size; break;

case 2: l2Cache += info.Cache.Size; break;

case 3: l3Cache += info.Cache.Size; break;

default: break;

}

}

}

std::cout << "L1 Cache Size: " << l1Cache / 1024 << " KB\n"

<< "L2 Cache Size: " << l2Cache / 1024 << " KB\n"

<< "L3 Cache Size: " << l3Cache / 1024 << " KB\n";

}

int main() {

std::cout << "System Information Utility" << std::endl;

PrintSystemInfo();

PrintProcessorFrequency();

PrintUptime();

PrintMemoryInfo();

PrintOSVersion();

PrintNetworkInfo();

PrintCacheInfo();

cache\_sum();

PrintProcessesInfo();

return 0;

}