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»
БГУИР 6-05-0612-02 23

Выполнил студент группы 353503
СЕБЕЛЕВ Дмитрий Юрьевич

(дата, подпись студента)

Проверил ассистент каф. информатики
ГРИЦЕНКО Никита Юрьевич

(дата, подпись преподавателя)

Минск 2025

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Целью данной работы является разработка программы, предоставляющей комплексный обзор ключевых системных и аппаратных характеристик. Программа должна собирать, структурировать и отображать для пользователя сведения о версии ОС, параметрах центрального процессора, объеме оперативной памяти, конфигурации системы (имя компьютера, системный каталог, текущий пользователь), а также дополнительные метрики, такие как разрешение экрана, время непрерывной работы и информация о дисковом пространстве. Это обеспечивает удобный инструмент для быстрой диагностики и инвентаризации локальной системы.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

2.1 Получение информации о программной конфигурации системы

В программе реализован сбор ключевых данных об операционной системе и пользовательском окружении. Для этого используются несколько специализированных функций, обращающихся к Windows API. Функция GetOSVersion использует вызов GetVersionEx со структурой OSVERSIONINFOEX для точного определения версии и номера сборки ОС. Функции GetSystemDirectoryPath и GetCurrentUser получают путь к системному каталогу и имя текущего пользователя через вызовы GetSystemDirectoryA и GetUserNameA соответственно. Для получения имени компьютера функция GetComputerNameFromRegistry напрямую обращается к системному реестру Windows, выполняя чтение из ключа SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName с помощью функций RegOpenKeyExA и RegQueryValueExA [1].

2.2 Сбор данных об аппаратном обеспечении

Также в программе реализованны методы для сбора информации об основных аппаратных компонентах системы, таких как центральный процессор и оперативная память. Функция GetCPUInfo вызывает GetSystemInfo для заполнения структуры SYSTEM_INFO, из которой извлекаются данные об архитектуре процессора (x64, x86 и др.) и количестве логических ядер. Для получения полного наименования модели процессора, функция GetProcessorModel выполняет чтение строкового значения ProcessorNameString из соответствующего раздела системного реестра. Функция GetMemoryInfo, в свою очередь, использует GlobalMemoryStatusEx для получения сведений об общем и доступном объеме физической оперативной памяти, которые затем форматируются в мегабайтах [2].

2.3 Сбор дополнительных системных метрик

Программа включает в себя сбора прочих важных системных показателей. Функция GetScreenResolution вызывает GetSystemMetrics с флагами SM_CXSCREEN и SM_CYSCREEN для определения разрешения основного дисплея в пикселях. Для вычисления времени непрерывной работы системы функция GetSystemUptime использует GetTickCount64 [3], которая возвращает количество миллисекунд с момента запуска ОС, после чего это значение преобразуется в формат «дни, часы, минуты, секунды». Функция GetDiskInfo сначала определяет системный диск с помощью GetWindowsDirectoryA, а затем вызывает GetDiskFreeSpaceExA для получения данных об общем и свободном дисковом пространстве, представляя результат в гигабайтах [4].

2.4 Агрегация и отображение результатов

Координацию всех операций выполняет главная функция main. Вначале работы устанавливается кодовая страница UTF-8 вызовом SetConsoleOutputCP для корректной работы с кириллицей в консоли. Затем происходит последовательный вызов всех вышеописанных функций сбора данных. Возвращаемые ими строки сохраняются в векторе std::vector. После сбора всей информации программа формирует итоговый отчет с помощью std::stringstream, объединяя все метрики в единый текстовый блок с разделителями. Финальный отчет выводится пользователю в стандартный поток вывода консоли, что обеспечивает наглядное и удобное представление собранной информации. Программный код представлен в приложении А.

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Пример выполнения задания

На рисунке 3.1 представлен консольный вывод программы, содержащий комплексную информацию о программной и аппаратной конфигурации системы.

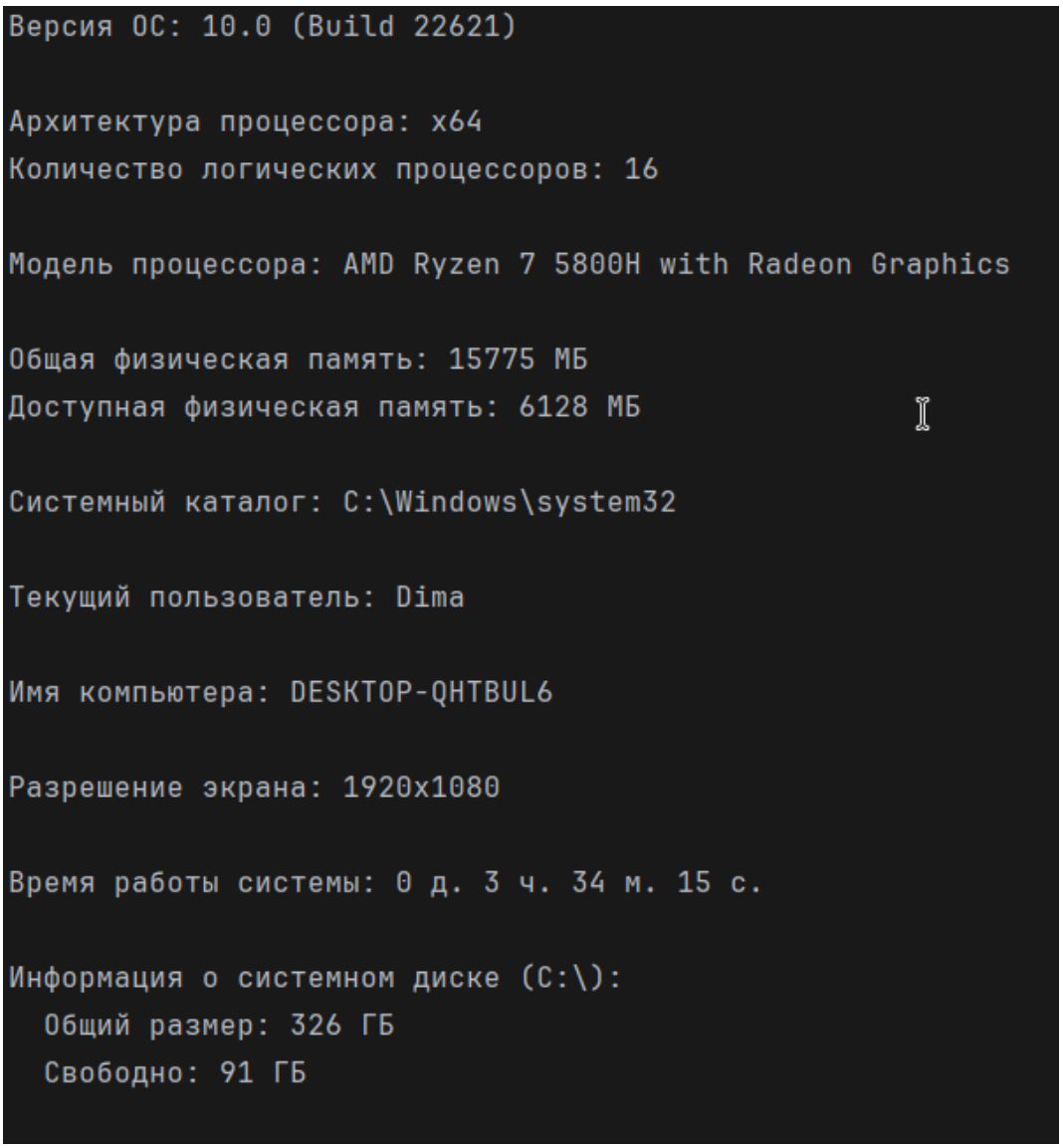

Рисунок 3.1 – Вывод информации о системе

ВЫВОД

В ходе выполнения лабораторной работы были освоены практические навыки использования WinAPI для получения детальной информации об аппаратной и программной конфигурации операционной системы Windows. Разработанная утилита предназначена для сбора и отображения ключевых системных параметров, обеспечивая пользователю наглядный и структурированный обзор состояния локальной машины.

Программа осуществляет сбор широкого спектра данных, включая версию операционной системы, архитектуру и модель процессора, количество логических ядер, общий и доступный объём оперативной памяти, системные каталоги, имя текущего пользователя и компьютера. Дополнительно выводятся сведения о разрешении экрана, времени непрерывной работы системы и доступном дисковом пространстве системного накопителя. Представленная информация может быть использована для оперативной диагностики, инвентаризации оборудования и получения общей сводки о конфигурации системы.

Для извлечения данных применяются различные функции Windows API. Сведения об операционной системе и процессоре получены с использованием GetVersionEx и GetSystemInfo. Доступ к параметрам, хранящимся в системном реестре, таким как модель процессора и имя компьютера, осуществляется через функции RegOpenKeyExA и RegQueryValueExA. Информация об оперативной памяти извлекается с помощью GlobalMemoryStatusEx, а данные о разрешении экрана и времени работы системы — посредством GetSystemMetrics и GetTickCount64 соответственно.

Архитектура приложения построена по модульному принципу: каждая группа параметров обрабатывается в отдельной функции (например, GetOSVersion, GetCPUInfo, GetMemoryInfo), что повышает читаемость кода и упрощает его сопровождение. Главная функция выполняет координирующую роль, последовательно вызывая модули сбора данных, формируя итоговый отчёт и выводя его пользователю в консольном виде.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] WinAPI: Function RegOpenKeyExA [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regopenkeyexa. – Дата доступа: 05.12.2025.

[2] WinAPI: Function GlobalMemoryStatus [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winbase/nf-winbase-globalmemorystatus. – Дата доступа: 05.12.2025.

[3] WinAPI: Function GetTickCount [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-gettickcount. – Дата доступа: 05.12.2025.

[4] WinAPI: Function GetDiskFreeSpaceA [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-getdiskfreespacea. – Дата доступа: 05.12.2025.

ПРИЛОЖЕНИЕ А
(справочное)
Исходный код программы

```
#include <windows.h>
#include <Lmcons.h>
#include <bits/stdc++.h>

using namespace std;

string GetOSVersion()
{
    OSVERSIONINFOEX osvi;
    ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
    if (!GetVersionEx((OSVERSIONINFO *)&osvi))
    {
        return "Не удалось определить версию ОС.";
    }

    string version = "Версия ОС: " + to_string(osvi.dwMajorVersion) + "."
+
                     to_string(osvi.dwMinorVersion) + " (Build " +
                     to_string(osvi.dwBuildNumber) + ")";
    return version;
}

string GetCPUInfo()
{
    SYSTEM_INFO sysInfo;
    GetSystemInfo(&sysInfo);

    string cpuInfo = "Архитектура процессора: ";
    switch (sysInfo.wProcessorArchitecture)
    {
    case PROCESSOR_ARCHITECTURE_AMD64:
        cpuInfo += "x64";
        break;
    case PROCESSOR_ARCHITECTURE_INTEL:
        cpuInfo += "x86";
        break;
    case PROCESSOR_ARCHITECTURE_ARM:
        cpuInfo += "ARM";
        break;
    case PROCESSOR_ARCHITECTURE_ARM64:
        cpuInfo += "ARM64";
        break;
    default:
        cpuInfo += "Неизвестная архитектура";
        break;
    }

    cpuInfo += "\nКоличество логических процессоров: " +
to_string(sysInfo.dwNumberOfProcessors);
    return cpuInfo;
}

string GetProcessorModel()
{
    HKEY hKey;
    const char *subKey = "HARDWARE\\DESCRIPTION\\System\
\CentralProcessor\\0";
    const char *valueName = "ProcessorNameString";

    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, subKey, 0, KEY_READ, &hKey) !=
ERROR_SUCCESS)
    {
        return "Не удалось открыть реестр для получения модели
```

```
процессора.";
    }

    char processorName[256];
    DWORD bufferSize = sizeof(processorName);
    DWORD type = 0;

    if (RegQueryValueExA(hKey, valueName, NULL, &type,
(LPBYTE)processorName, &bufferSize) != ERROR_SUCCESS or type != REG_SZ)
    {
        RegCloseKey(hKey);
        return "Не удалось получить модель процессора из реестра.";
    }

    RegCloseKey(hKey);
    return string("Модель процессора: ") + processorName;
}

string GetMemoryInfo()
{
    MEMORYSTATUSEX memInfo;
    memInfo.dwLength = sizeof(MEMORYSTATUSEX);
    if (!GlobalMemoryStatusEx(&memInfo))
    {
        return "Не удалось получить информацию об оперативной памяти.";
    }

    DWORDLONG totalPhys = memInfo.ullTotalPhys / (1024 * 1024);
    DWORDLONG availPhys = memInfo.ullAvailPhys / (1024 * 1024);
    string memoryInfo = "Общая физическая память: " +
to_string(totalPhys) + " МБ\n" +
                        "Доступная физическая память: " +
to_string(availPhys) + " МБ";
    return memoryInfo;
}

string GetSystemDirectoryPath()
{
    char systemDir[MAX_PATH];
    UINT len = GetSystemDirectoryA(systemDir, MAX_PATH);
    if (len == 0 or len > MAX_PATH)
    {
        return "Не удалось получить системный каталог.";
    }
    return string("Системный каталог: ") + systemDir;
}

string GetCurrentUser()
{
    char username[UNLEN + 1];
    DWORD username_len = UNLEN + 1;
    if (GetUserNameA(username, &username_len))
    {
        return string("Текущий пользователь: ") + username;
    }
    return "Не удалось получить имя текущего пользователя.";
}

string GetComputerNameFromRegistry()
{
    HKEY hKey;
    const char *subKey = "SYSTEM\\CurrentControlSet\\Control\
\ComputerName\\ComputerName";
    const char *valueName = "ComputerName";

    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, subKey, 0, KEY_READ, &hKey) !=
ERROR_SUCCESS)
    {
        return "Не удалось открыть реестр для получения имени
компьютера.";
    }
```

```
    char computerName[256];
    DWORD bufferSize = sizeof(computerName);
    DWORD type = 0;

    if (RegQueryValueExA(hKey, valueName, NULL, &type,
(LPBYTE)computerName, &bufferSize) != ERROR_SUCCESS or type != REG_SZ)
    {
        RegCloseKey(hKey);
        return "Не удалось получить имя компьютера из реестра.";
    }

    RegCloseKey(hKey);
    return string("Имя компьютера: ") + computerName;
}

string GetScreenResolution()
{
    int screenWidth = GetSystemMetrics(SM_CXSCREEN);
    int screenHeight = GetSystemMetrics(SM_CYSCREEN);
    return "Разрешение экрана: " + to_string(screenWidth) + "x" +
to_string(screenHeight);
}

string GetSystemUptime()
{
    ULONGLONG uptimeMillis = GetTickCount64();
    ULONGLONG uptimeSeconds = uptimeMillis / 1000;
    ULONGLONG days = uptimeSeconds / (24 * 3600);
    uptimeSeconds %= (24 * 3600);
    ULONGLONG hours = uptimeSeconds / 3600;
    uptimeSeconds %= 3600;
    ULONGLONG minutes = uptimeSeconds / 60;
    ULONGLONG seconds = uptimeSeconds % 60;

    string uptime = "Время работы системы: " + to_string(days) + " д. "
+
                    to_string(hours) + " ч. " +
                    to_string(minutes) + " м. " +
                    to_string(seconds) + " с.";
    return uptime;
}

string GetDiskInfo()
{
    char winDir[MAX_PATH];
    if (GetWindowsDirectoryA(winDir, MAX_PATH) == 0) {
        return "Не удалось определить системный диск.";
    }

    char drive[] = "C:\\";
    drive[0] = winDir[0];

    ULARGE_INTEGER freeBytesAvailableToCaller, totalNumberOfBytes,
totalNumberOfFreeBytes;

    if (GetDiskFreeSpaceExA(drive, &freeBytesAvailableToCaller,
&totalNumberOfBytes, &totalNumberOfFreeBytes))
    {
        DWORDLONG totalSizeGB = totalNumberOfBytes.QuadPart / (1024 *
1024 * 1024);
        DWORDLONG freeSizeGB = totalNumberOfFreeBytes.QuadPart / (1024 *
1024 * 1024);
        return "Информация о системном диске (" + string(1, drive[0]) +
":\\):\n" +
               "  Общий размер: " + to_string(totalSizeGB) + " ГБ\n" +
               "  Свободно: " + to_string(freeSizeGB) + " ГБ";
    }
    else
    {
        return "Не удалось получить информацию о диске.";
```

```
    }
}


int main()
{
    SetConsoleOutputCP(CP_UTF8);
    vector<string> systemInfo;
    systemInfo.push_back(GetOSVersion());
    systemInfo.push_back(GetCPUInfo());
    systemInfo.push_back(GetProcessorModel());
    systemInfo.push_back(GetMemoryInfo());
    systemInfo.push_back(GetSystemDirectoryPath());
    systemInfo.push_back(GetCurrentUser());
    systemInfo.push_back(GetComputerNameFromRegistry());
    systemInfo.push_back(GetScreenResolution());
    systemInfo.push_back(GetSystemUptime());
    systemInfo.push_back(GetDiskInfo());

    stringstream ss;
    for (const auto &info : systemInfo)
    {
        ss << info << "\n\n";
    }

    string finalInfo = ss.str();
    if (!finalInfo.empty()) {
        finalInfo.pop_back();
        finalInfo.pop_back();
    }

    cout << finalInfo << endl;

    return 0;
}
```