

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 №6
на тему:
«НЕКОТОРЫЕ СЛУЖЕБНЫЕ И ТЕХНОЛОГИЧЕСКИЕ ЗАДАЧИ»
БГУИР 6-05-0612-02 67

Выполнил студент группы 353503
КОХАН Артём Игоревич

(дата, подпись студента)

Проверил ассистент каф. информатики
Гриценко Никита Юрьевич

(дата, подпись преподавателя)

Минск 2025

СОДЕРЖАНИЕ

1 Индивидуальное задание.....	3
2 Краткие теоритические сведения	4
3 Описание функций программы.....	5
4 Пример выполнения программы	7
4.1 Запуск программы и процесс выполнения	7
4.2 Описание работы и результатов	7
Вывод.....	9
Список использованных источников	10
Приложение А (справочное) Исходный код	11

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Приложение, демонстрирующее функциональность в соответствии с выбранным заданием. При необходимости – взаимодействие с другим ПО (прикладным и/или системным). Для библиотек и сборок постановка задачи может включать написание подключаемого модуля, головной программы или всего программного комплекса.

Утилита, обеспечивающая 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 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Системное программирование включает конфигурирование, мониторинг и управление операционной системой, что подразумевает глубокое понимание взаимодействия с её внутренними компонентами, такими как реестр, журналы и библиотеки. Реестр *Windows* представляет собой иерархическую базу данных, содержащую важные сведения о настройках и параметрах операционной системы, программного обеспечения и подключенного оборудования. Он структурирован на основе ключей и значений, что позволяет эффективно хранить и организовывать данные [1].

Журналирование в *Windows* позволяет фиксировать события, происходящие в системе, что помогает отслеживать изменения и выявлять проблемы. Существуют различные виды журналов, такие как системные журналы, журналы безопасности и приложений, которые хранят сведения о важных действиях, ошибках и предупреждениях [2]. Журналы предоставляют интерфейсы *API* для чтения, записи и анализа данных, что полезно для мониторинга и диагностики системы.

Технологические аспекты системного программирования включают использование библиотек, таких как динамические библиотеки (*DLL*), которые позволяют разделять и повторно использовать код между разными приложениями. Динамические библиотеки могут подключаться к программам как статически, так и динамически, предоставляя возможность загружать и использовать функции только при необходимости [3]. Это упрощает разработку, уменьшает размер исполняемых файлов и улучшает использование ресурсов системы.

Для обеспечения мониторинга и управления системой также важны механизмы перехвата и обработки сообщений, такие как *WinHook*. *WinHook* позволяет обрабатывать оконные сообщения в операционной системе *Windows*, что дает возможность контролировать взаимодействие между приложениями и системой на низком уровне [4].

В лабораторной работе также будет рассмотрен механизм просмотра информации о системе, что включает сбор данных о аппаратном обеспечении и программной среде системы. Для этой цели могут быть использованы специализированные функции *Windows Management Instrumentation* (*WMI*), реестр, а также другие доступные источники информации. Это позволит понять структуру операционной системы и особенности её функционирования, а также даст практический опыт взаимодействия с низкоуровневыми системными интерфейсами.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа представляет собой утилиту для сбора и отображения системной информации о компьютере под управлением операционной системы Windows. Основным классом программы является SystemInfo, который инкапсулирует логику сбора данных о различных компонентах системы. Класс содержит несколько вложенных структур для хранения информации: OSInfo для данных об операционной системе, CPUInfo для сведений о процессоре, MemoryInfo для информации о памяти, DiskInfo для данных о дисках, GraphicsInfo для сведений о графической подсистеме и NetworkInfo для сетевой информации.

Метод CollectOSInfo собирает информацию об операционной системе. Для получения данных используется комбинация вызовов WinAPI и чтения системного реестра. Из реестра считывается название продукта, версия сборки и дата установки системы. Для определения архитектуры ОС используется функция GetSystemInfo. Метод FormatBytes предоставляет удобное форматирование байтов в килобайты, мегабайты или гигабайты для улучшения читаемости вывода.

Метод CollectCPUInfo извлекает информацию о центральном процессоре. Из реестра получается название процессора, информация о производителе и тактовая частота. Количество логических процессоров определяется через GetSystemInfo, а количество физических ядер считывается из реестра или вычисляется на основе доступных данных.

Метод CollectMemoryInfo использует функцию GlobalMemoryStatusEx для получения подробной информации об использовании оперативной и виртуальной памяти. Собираются данные об общем объеме физической памяти, доступной памяти, общей виртуальной памяти и проценте использования памяти.

Метод CollectDiskInfo перебирает все логические диски системы с помощью GetLogicalDrives и собирает информацию только о фиксированных дисках. Для каждого диска определяется файловая система, общий размер, свободное пространство и метка тома с использованием функций GetVolumeInformationA и GetDiskFreeSpaceExA.

Метод CollectGraphicsInfo находит основное графическое устройство через EnumDisplayDevicesA и получает его настройки разрешения и глубины цвета с помощью EnumDisplaySettingsA. Метод CollectNetworkInfo собирает сетевую информацию, включая имя компьютера через GetComputerNameA, имя пользователя через GetUserNomeA и MAC-адрес основного сетевого адаптера через GetAdaptersInfo.

Метод CollectProcessesInfo использует EnumProcesses для получения списка запущенных процессов и GetModuleBaseNameA для определения их имен. Отображаются только первые 15 процессов для ограничения объема вывода. Конструктор класса SystemInfo автоматически запускает сбор всей информации при создании объекта. Метод DisplayInfo форматирует и выводит собранные данные в структурированном виде с разделением на категории.

В функции main создается экземпляр SystemInfo и вызывается метод DisplayInfo для отображения результатов. Программа использует различные библиотеки WinAPI, включая iphlpapi для сетевой информации, psapi для данных о процессах и advapi32 для работы с реестром. Утилита демонстрирует практическое использование WinAPI для получения низкоуровневой системной информации и может быть полезной для диагностики и мониторинга системы.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

Программа была запущена на компьютере под управлением операционной системы Windows 10 Enterprise версии 10.0.19045 сборки 19045 64-битной архитектуры. При запуске утилита начала автоматический сбор comprehensive информации о всех основных компонентах системы, отобразив начальное сообщение "Collecting system information..." в процессе инициализации.

```
== OPERATING SYSTEM ==
OS Name: Windows 10 Enterprise
Version: 10.0.19045
Build: 19045
Architecture: 64-bit
Install Date: N/A

== PROCESSOR ==
CPU: AMD Ryzen 7 5800H with Radeon Graphics
Manufacturer: AMD
Cores: 8 physical, 16 logical
Clock Speed: 3194 MHz
Architecture: x64

== MEMORY ==
Total Physical: 15.35 GB
Available Physical: 5.82 GB
Total Virtual: 128.00 TB
Available Virtual: 128.00 TB
Memory Usage: 62%

== STORAGE ==
C:\ ()
File System: NTFS
Total Size: 571.37 GB
Free Space: 271.66 GB
Used: 52.5%

D:\ ()
File System: FAT32
Total Size: 510.98 MB
Free Space: 469.27 MB
Used: 8.2%

K:\ (Новый том)
File System: NTFS
Total Size: 208.14 GB
Free Space: 178.60 GB
Used: 14.2%

== GRAPHICS ==
Graphics Card: AMD Radeon(TM) Graphics
Resolution: 1920 x 1080
Color Depth: 32 bits

== NETWORK ==
Computer Name: DESKTOP-1QCKF8F
User Name: KOXAN
MAC Address: 4c-d5-77-e0-c0-70
```

Рисунок 4.1 – Результат выполнения программы

4.2 Описание работы и результатов

Информация о процессоре показала, что система использует процессор AMD Ryzen 7 5800H с Radeon Graphics от производителя AMD. Программа

корректно определила архитектуру процессора как x64, тактовую частоту 3194 МГц, а также правильно идентифицировала 8 физических ядер и 16 логических процессоров.

Анализ памяти выявил, что общий объем физической памяти составляет 15.35 ГБ, из которых доступно 5.99 ГБ, что соответствует 61% использованию памяти. Программа успешно обработала данные о виртуальной памяти системы, показав общий объем 128.00 ТБ с полной доступностью.

В разделе хранилищ программа обнаружила и проанализировала три диска с различными характеристиками. Для диска C:\ файловой системы NTFS был определен общий размер 571.37 ГБ с свободным пространством 271.66 ГБ (использовано 52.5%). Диск D:\ файловой системы FAT32 имел размер 510.98 МБ с свободными 469.27 МБ (использовано 8.2%). Диск K:\ с меткой "НОВЫЙ ТОМ" файловой системы NTFS показал размер 208.14 ГБ со свободным пространством 178.60 ГБ (использовано 14.2%).

Графическая подсистема была идентифицирована как видеокарта AMD Radeon (TM) Graphics с разрешением экрана 1920 x 1080 пикселей и глубиной цвета 32 бита. Сетевая информация включала имя компьютера DESKTOP-1QCKF8F, имя пользователя KOXAN и MAC-адрес 4c-d5-77-e0-c0-70.

ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана и протестирована утилита для сбора и анализа системной информации компьютера под управлением операционной системы Windows.

Результаты тестирования, показали, что разработанная система успешно собирает и корректно отображает детальную информацию о конфигурации оборудования и состоянии системных ресурсов. Утилита точно идентифицировала характеристики процессора AMD Ryzen 7 5800H, объем оперативной памяти 15.35 ГБ, параметры трех логических дисков с различными файловыми системами, а также сетевые настройки компьютера. Особенностью реализации стало автоматическое форматирование данных с преобразованием единиц измерения и расчетом процентных соотношений использования ресурсов.

Полученные результаты наглядно демонстрируют принципы взаимодействия с системными компонентами Windows через API и могут быть использованы для создания инструментов мониторинга, диагностики и анализа производительности компьютерных систем. Программа успешно решает поставленную задачу и представляет собой законченный пример реализации системной утилиты с использованием интерфейсов WinAPI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft Docs: Introduction to the Windows Registry [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry>. – Дата доступа: 19.11.2025.

[2] Microsoft Docs: Event Logging [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging>. – Дата доступа: 19.11.2025.

[3] Microsoft Docs: Dynamic-Link Libraries (DLLs) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries>. – Дата доступа: 19.11.2025.

[4] Microsoft Docs: Using Hooks [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/winmsg/using-hooks>. – Дата доступа: 19.11.2025.

ПРИЛОЖЕНИЕ А

(справочное)

Исходный код

Листинг А.1 – Реализация Lab6.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <sstream>
#include <iphlpapi.h>
#include <psapi.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "psapi.lib")
#pragma comment(lib, "advapi32.lib")
#pragma comment(lib, "user32.lib")

class SystemInfo {
private:
    struct OSInfo {
        std::string name;
        std::string version;
        std::string build;
        std::string architecture;
        std::string installDate;
    };

    struct CPUInfo {
        std::string name;
        std::string manufacturer;
        DWORD numberOfCores;
        DWORD numberOfLogicalProcessors;
        DWORD clockSpeed;
        std::string architecture;
    };

    struct MemoryInfo {
        UONGLONG totalPhysical;
        UONGLONG availablePhysical;
        UONGLONG totalVirtual;
        UONGLONG availableVirtual;
        DWORD memoryLoad;
    };

    struct DiskInfo {
        std::string drive;
        std::string fileSystem;
        UONGLONG totalSize;
        UONGLONG freeSpace;
        std::string volumeName;
    };

    struct GraphicsInfo {
        std::string cardName;
        DWORD horizontalResolution;
        DWORD verticalResolution;
        DWORD colorDepth;
    };
};
```

```

    struct NetworkInfo {
        std::string hostName;
        std::string userName;
        std::string macAddress;
    };

    OSInfo osInfo;
    CPUInfo cpuInfo;
    MemoryInfo memoryInfo;
    std::vector<DiskInfo> disksInfo;
    GraphicsInfo graphicsInfo;
    NetworkInfo networkInfo;
    std::vector<std::string> runningProcesses;

    std::string ReadRegistryString(HKEY hKey, const std::string& subKey, const
std::string& valueName) {
        HKEY hRegistryKey;
        CHAR buffer[1024];
        DWORD bufferSize = sizeof(buffer);
        std::string result = "N/A";

        if (RegOpenKeyExA(hKey, subKey.c_str(), 0, KEY_READ, &hRegistryKey) == ERROR_SUCCESS) {
            if (RegQueryValueExA(hRegistryKey, valueName.c_str(), NULL, NULL,
(LPBYTE)buffer, &bufferSize) == ERROR_SUCCESS) {
                result = buffer;
            }
            RegCloseKey(hRegistryKey);
        }
        return result;
    }

    std::string FormatBytes(ULLONG bytes) {
        const char* suffixes[] = { "B", "KB", "MB", "GB", "TB" };
        int suffixIndex = 0;
        double size = static_cast<double>(bytes);

        while (size >= 1024 && suffixIndex < 4) {
            size /= 1024;
            suffixIndex++;
        }

        std::stringstream ss;
        ss << std::fixed << std::setprecision(2) << size << " " <<
suffixes[suffixIndex];
        return ss.str();
    }

    std::string GetMACAddress() {
        IP_ADAPTER_INFO adapterInfo[16];
        DWORD bufferSize = sizeof(adapterInfo);

        if (GetAdaptersInfo(adapterInfo, &bufferSize) == ERROR_SUCCESS) {
            PIP_ADAPTER_INFO pAdapterInfo = adapterInfo;
            if (pAdapterInfo) {
                std::stringstream ss;
                for (int i = 0; i < pAdapterInfo->AddressLength; i++) {
                    if (i == (pAdapterInfo->AddressLength - 1))
                        ss << std::hex << std::setw(2) << std::setfill('0') <<
(int)pAdapterInfo->Address[i];
                    else
                        ss << std::hex << std::setw(2) << std::setfill('0') <<
(int)pAdapterInfo->Address[i] << "-";
                }
            }
        }
    }
}

```

```

        }
        return ss.str();
    }
    return "N/A";
}

void CollectOSInfo() {
    // Получаем версию ОС
    HMODULE hNtdll = GetModuleHandleA("ntdll.dll");
    if (hNtdll) {
        typedef LONG(WINAPI* RtlGetVersionPtr)(PRTL_OSVERSIONINFO);
        RtlGetVersionPtr RtlGetVersion = (RtlGetVersionPtr)GetProcAddress(hNtdll, "RtlGetVersion");

        if (RtlGetVersion) {
            RTL_OSVERSIONINFO osVersion = { 0 };
            osVersion.dwOSVersionInfoSize = sizeof(osVersion);
            if (RtlGetVersion(&osVersion) == 0) {
                osInfo.version = std::to_string(osVersion.dwMajorVersion)
+ "." +
                    std::to_string(osVersion.dwMinorVersion) + "." +
                    std::to_string(osVersion.dwBuildNumber);
            }
        }
    }

    osInfo.name = ReadRegistryString(HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "ProductName");

    osInfo.build = ReadRegistryString(HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
        "CurrentBuild");

    SYSTEM_INFO sysInfo;
    GetSystemInfo(&sysInfo);
    osInfo.architecture = (sysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_AMD64) ? "64-bit" :
        (sysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_INTEL) ? "32-bit" : "Unknown";

    std::string installTimestamp = ReadRegistryString(HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "InstallDate");

    if (installTimestamp != "N/A") {
        try {
            time_t installTime = std::stol(installTimestamp);
            std::tm timeinfo;
            localtime_s(&timeinfo, &installTime);
            std::stringstream ss;
            ss << std::put_time(&timeinfo, "%Y-%m-%d %H:%M:%S");
            osInfo.installDate = ss.str();
        }
        catch (...) {
            osInfo.installDate = "N/A";
        }
    }
    else {
        osInfo.installDate = "N/A";
    }
}

void CollectCPUInfo() {
    cpuInfo.name = ReadRegistryString(HKEY_LOCAL_MACHINE,

```

```

        "HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
"ProcessorNameString");

    std::string vendor = ReadRegistryString(HKEY_LOCAL_MACHINE,
        "HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
"VendorIdentifier");

    if (vendor.find("GenuineIntel") != std::string::npos)
        cpuInfo.manufacturer = "Intel";
    else if (vendor.find("AuthenticAMD") != std::string::npos)
        cpuInfo.manufacturer = "AMD";
    else
        cpuInfo.manufacturer = vendor;

    SYSTEM_INFO sysInfo;
    GetSystemInfo(&sysInfo);
    cpuInfo.numberOfLogicalProcessors = sysInfo.dwNumberOfProcessors;

    HKEY hKey;
    DWORD mhz = 0;
    DWORD size = sizeof(DWORD);
    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE,
"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
0, KEY_READ, &hKey) == ERROR_SUCCESS) {
        RegQueryValueExA(hKey, "~MHz", NULL, NULL, (LPBYTE)&mhz, &size);
        RegCloseKey(hKey);
    }
    cpuInfo.clockSpeed = mhz;

    cpuInfo.architecture = (sysInfo.wProcessorArchitecture ==
PROCESSOR_ARCHITECTURE_AMD64) ? "x64" :
        (sysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_INTEL) ?
"x86" : "Unknown";

    DWORD cores = 0;
    size = sizeof(DWORD);
    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE,
"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
0, KEY_READ, &hKey) == ERROR_SUCCESS) {
        if (RegQueryValueExA(hKey, "NumberOfCores", NULL, NULL,
(LPBYTE)&cores, &size) == ERROR_SUCCESS) {
            cpuInfo.numberOfCores = cores;
        }
        else {
            cpuInfo.numberOfCores = cpuInfo.numberOfLogicalProcessors / 2;
            if (cpuInfo.numberOfCores == 0) cpuInfo.numberOfCores = 1;
        }
        RegCloseKey(hKey);
    }
}

void CollectMemoryInfo() {
    MEMORYSTATUSEX memoryStatus;
    memoryStatus.dwLength = sizeof(memoryStatus);

    if (GlobalMemoryStatusEx(&memoryStatus)) {
        memoryInfo.totalPhysical = memoryStatus.ullTotalPhys;
        memoryInfo.availablePhysical = memoryStatus.ullAvailPhys;
        memoryInfo.totalVirtual = memoryStatus.ullTotalVirtual;
        memoryInfo.availableVirtual = memoryStatus.ullAvailVirtual;
        memoryInfo.memoryLoad = memoryStatus.dwMemoryLoad;
    }
}

```

```

void CollectDiskInfo() {
    DWORD driveMask = GetLogicalDrives();

    for (char drive = 'A'; drive <= 'Z'; drive++) {
        if (driveMask & 1) {
            std::string rootPath = std::string(1, drive) + ":\\";

            UINT driveType = GetDriveTypeA(rootPath.c_str());

            if (driveType == DRIVE_FIXED) {
                DiskInfo disk;
                disk.drive = rootPath;

                CHAR fileSystem[32];
                DWORD serialNumber, maxComponentLength, fileSystemFlags;

                if (GetVolumeInformationA(rootPath.c_str(),
                    nullptr, 0, &serialNumber, &maxComponentLength,
                    &fileSystemFlags, fileSystem, sizeof(fileSystem))) {
                    disk.fileSystem = fileSystem;
                }

                ULARGE_INTEGER freeBytes, totalBytes, totalFreeBytes;
                if (GetDiskFreeSpaceExA(rootPath.c_str(), &freeBytes,
                    &totalBytes, &totalFreeBytes)) {
                    disk.totalSize = totalBytes.QuadPart;
                    disk.freeSpace = freeBytes.QuadPart;
                }
            }

            CHAR volumeName[256];
            if (GetVolumeInformationA(rootPath.c_str(), volumeName,
                sizeof(volumeName),
                &serialNumber, &maxComponentLength, &fileSystemFlags,
                fileSystem, sizeof(fileSystem))) {
                disk.volumeName = volumeName;
            }

            disksInfo.push_back(disk);
        }
        driveMask >>= 1;
    }
}

void CollectGraphicsInfo() {
    DISPLAY_DEVICEA displayDevice;
    ZeroMemory(&displayDevice, sizeof(displayDevice));
    displayDevice.cb = sizeof(displayDevice);

    for (int i = 0; EnumDisplayDevicesA(NULL, i, &displayDevice, 0); i++)
    {
        if (displayDevice.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE) {
            graphicsInfo.cardName = displayDevice.DeviceString;

            DEVMODEA devMode;
            devMode.dmSize = sizeof(devMode);
            if (EnumDisplaySettingsA(displayDevice.DeviceName,
                ENUM_CURRENT_SETTINGS, &devMode)) {
                graphicsInfo.horizontalResolution = devMode.dmPelsWidth;
                graphicsInfo.verticalResolution = devMode.dmPelsHeight;
                graphicsInfo.colorDepth = devMode.dmBitsPerPel;
            }
            break;
        }
    }
}

```

```

}

void CollectNetworkInfo() {
    CHAR computerName[256];
    DWORD size = sizeof(computerName);
    if (GetComputerNameA(computerName, &size)) {
        networkInfo.hostName = computerName;
    }

    CHAR userName[256];
    size = sizeof(userName);
    if (GetUserNameA(userName, &size)) {
        networkInfo.userName = userName;
    }

    networkInfo.macAddress = GetMACAddress();
}

void CollectProcessesInfo() {
    DWORD processes[1024];
    DWORD needed;

    if (EnumProcesses(processes, sizeof(processes), &needed)) {
        DWORD processCount = needed / sizeof(DWORD);

        for (DWORD i = 0; i < processCount && runningProcesses.size() < 15;
i++) {
            if (processes[i] != 0) {
                HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ, FALSE, processes[i]);
                if (hProcess) {
                    CHAR processName[MAX_PATH];
                    if (GetModuleBaseNameA(hProcess, NULL, processName,
sizeof(processName))) {
                        runningProcesses.push_back(processName);
                    }
                    CloseHandle(hProcess);
                }
            }
        }
    }
}

void CollectAllInfo() {
    CollectOSInfo();
    CollectCPUInfo();
    CollectMemoryInfo();
    CollectDiskInfo();
    CollectGraphicsInfo();
    CollectNetworkInfo();
    CollectProcessesInfo();
}

public:
    SystemInfo() {
        CollectAllInfo();
    }

    void DisplayInfo() {
        std::cout << "===== SYSTEM INFORMATION UTILITY =====" <<
std::endl;
        std::cout << "===== SYSTEM INFORMATION UTILITY =====" <<
std::endl;
        std::cout << std::endl;
    }
}

```

```

// Операционная система
std::cout << "==== OPERATING SYSTEM ===" << std::endl;
std::cout << "OS Name:      " << osInfo.name << std::endl;
std::cout << "Version:      " << osInfo.version << std::endl;
std::cout << "Build:        " << osInfo.build << std::endl;
std::cout << "Architecture: " << osInfo.architecture << std::endl;
std::cout << "Install Date: " << osInfo.installDate << std::endl <<
std::endl;

// Процессор
std::cout << "==== PROCESSOR ===" << std::endl;
std::cout << "CPU:           " << cpuInfo.name << std::endl;
std::cout << "Manufacturer: " << cpuInfo.manufacturer << std::endl;
std::cout << "Cores:         " << cpuInfo.numberOfCores << " physical,
"
                           << cpuInfo.numberOfLogicalProcessors << " logical" << std::endl;
std::cout << "Clock Speed:   " << cpuInfo.clockSpeed << " MHz" <<
std::endl;
std::cout << "Architecture: " << cpuInfo.architecture << std::endl <<
std::endl;

// Память
std::cout << "==== MEMORY ===" << std::endl;
std::cout << "Total          Physical:      " <<
FormatBytes(memoryInfo.totalPhysical) << std::endl;
std::cout << "Available       Physical:      " <<
FormatBytes(memoryInfo.availablePhysical) << std::endl;
std::cout << "Total          Virtual:      " <<
FormatBytes(memoryInfo.totalVirtual) << std::endl;
std::cout << "Available       Virtual:      " <<
FormatBytes(memoryInfo.availableVirtual) << std::endl;
std::cout << "Memory Usage:    " << memoryInfo.memoryLoad << "%" <<
std::endl << std::endl;

// Диски
std::cout << "==== STORAGE ===" << std::endl;
for (const auto& disk : disksInfo) {
    double usedPercent = 100.0 - ((double)disk.freeSpace / disk.totalSize * 100.0);
    std::cout << disk.drive << " (" << disk.volumeName << ")" <<
std::endl;
    std::cout << " File System: " << disk.fileSystem << std::endl;
    std::cout << " Total Size:  " << FormatBytes(disk.totalSize) <<
std::endl;
    std::cout << " Free Space:  " << FormatBytes(disk.freeSpace) <<
std::endl;
    std::cout << " Used:        " << std::fixed << std::setprecision(1)
<< usedPercent << "%" << std::endl << std::endl;
}

// Графика
std::cout << "==== GRAPHICS ===" << std::endl;
std::cout << "Graphics Card: " << graphicsInfo.cardName << std::endl;
std::cout << "Resolution:     " << graphicsInfo.horizontalResolution <<
" x "
                           << graphicsInfo.verticalResolution << std::endl;
std::cout << "Color Depth:    " << graphicsInfo.colorDepth << " bits"
<< std::endl << std::endl;

// Сеть
std::cout << "==== NETWORK ===" << std::endl;
std::cout << "Computer Name: " << networkInfo.hostName << std::endl;
std::cout << "User Name:      " << networkInfo.userName << std::endl;

```

```
    std::cout << "MAC Address:      " << networkInfo.macAddress << std::endl
<< std::endl;

    // Процессы
    std::cout << "==== RUNNING PROCESSES (first 15) ===" << std::endl;
    for (size_t i = 0; i < runningProcesses.size(); i++) {
        std::cout << std::setw(20) << std::left << runningProcesses[i];
        if ((i + 1) % 3 == 0) std::cout << std::endl;
    }
    if (!runningProcesses.empty() && runningProcesses.size() % 3 != 0) {
        std::cout << std::endl;
    }
}

};

int main() {
    std::setlocale(LC_ALL, "RU");

    std::cout << "Collecting system information..." << std::endl << std::endl;

    SystemInfo systemInfo;
    systemInfo.DisplayInfo();

    std::cout << std::endl << "Press Enter to exit...";
    std::cin.get();

    return 0;
}
```