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. 253503
Тимошевич К.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Задачей данной лабораторной работы является создание программы для сбора и отображения системной информации. Цель работы – реализовать программу, которая будет собирать информацию о системе, включая данные об аппаратном обеспечении, операционной системе и других характеристиках. Основное внимание уделяется использованию *API Windows* для извлечения данных, таких как информация из реестра, файловой системы и *WMI (Windows Management Instrumentation)*.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В данном разделе описаны основные функции программы, использованные для реализации сетевого чата, а также методы организации взаимодействия между сервером и клиентами.

2.1 Сбор информации о процессоре

Функция *PrintSystemInfo* отвечает за получение базовых сведений о системе, таких как архитектура процессора, число ядер и размер страницы памяти. Используется системный вызов *GetSystemInfo* [1], который обеспечивает прямой доступ к информации об аппаратных характеристиках системы. Полученные данные помогают пользователю определить основные параметры оборудования.

Функция *PrintProcessorFrequency* читает частоту процессора в мегагерцах из реестра Windows с использованием *API*-функции *RegQueryValueEx* [2]. Это позволяет уточнить тактовую частоту процессора, что важно для оценки производительности.

2.2 Анализ оперативной памяти и времени работы системы

Функция *PrintMemoryInfo* использует вызов *GlobalMemoryStatusEx* для получения данных о состоянии памяти, таких как общий объем физической и виртуальной памяти, а также доступные ресурсы. Это позволяет пользователю оценить текущее использование памяти системой.Функция *PrintUptime* использует *API*-функцию *GetTickCount* [3], которая возвращает время работы системы в миллисекундах. Результат преобразуется в дни, часы, минуты и секунды для удобного представления пользователю.

2.3 Обзор запущенных процессов и анализ кеш-памяти

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

Функция PrintCacheInfo предоставляет данные о кэшах процессора *(L1, L2, L3*), включая их размер, тип (инструкционный, данных, унифицированный) и уровень. Для вычислений используется вызов *GetLogicalProcessorInformation* [4], который извлекает подробную информацию о логических процессорах и связанных с ними кэшах. Дополнительно функция *cache_sum* суммирует объемы кэшей каждого уровня, чтобы предоставить общие данные о кэш-памяти процессора.

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Примеры выполнения задания

На рисунке 3.1 показан вывод информации о процессоре, памяти, а также об операционной системе, частоте процессора и времени работы системы.

```
System Information Utility
Hardware Information:
  Processor Architecture: x64 (AMD or Intel)
  Number of Processors: 12
  Page Size: 4096 bytes

Memory Information:
  Total Physical Memory: 15722 MB
  Available Physical Memory: 5879 MB
  Total Virtual Memory: 134217727 MB
  Available Virtual Memory: 134213552 MB

OS Information:
  Version: 10.0
  Build Number: 19045
  System Type: NT

System Uptime: 0 days, 22 hours, 15 minutes, 18 seconds
Processor Frequency: 2096 MHz
```

Рисунок 3.1 - Вывод информации

На рисунке 3.2 продемонстрирована часть вывода информации о кеш-памяти.

```
Cache Level: L1
Cache Type: Data Cache
Cache Size: 32 KB

Cache Level: L1
Cache Type: Instruction Cache
Cache Size: 32 KB

Cache Level: L2
Cache Type: Unified Cache
Cache Size: 512 KB

Cache Level: L3
Cache Type: Unified Cache
Cache Size: 4096 KB

L1 Cache Size: 384 KB
L2 Cache Size: 3072 KB
L3 Cache Size: 8192 KB
```

Рисунок 3.2 - Вывод информации о кеш-памяти

На рисунке 3.3 показана часть списка запущенных программ на устройстве.

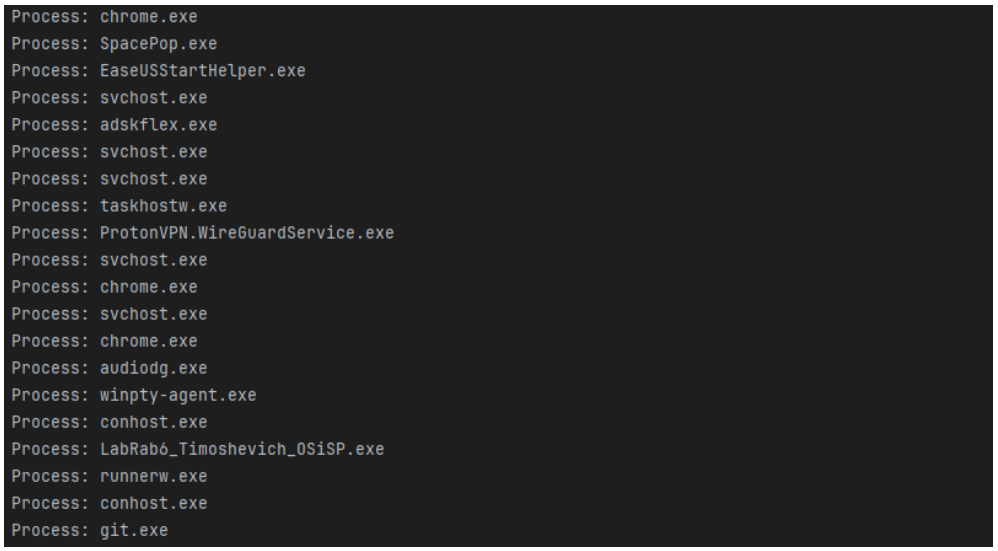

Рисунок 3.3 - Получение запущенных процессов

Вывод информации о сетевых адаптерах продемонстрирован на рисунке 3.4.

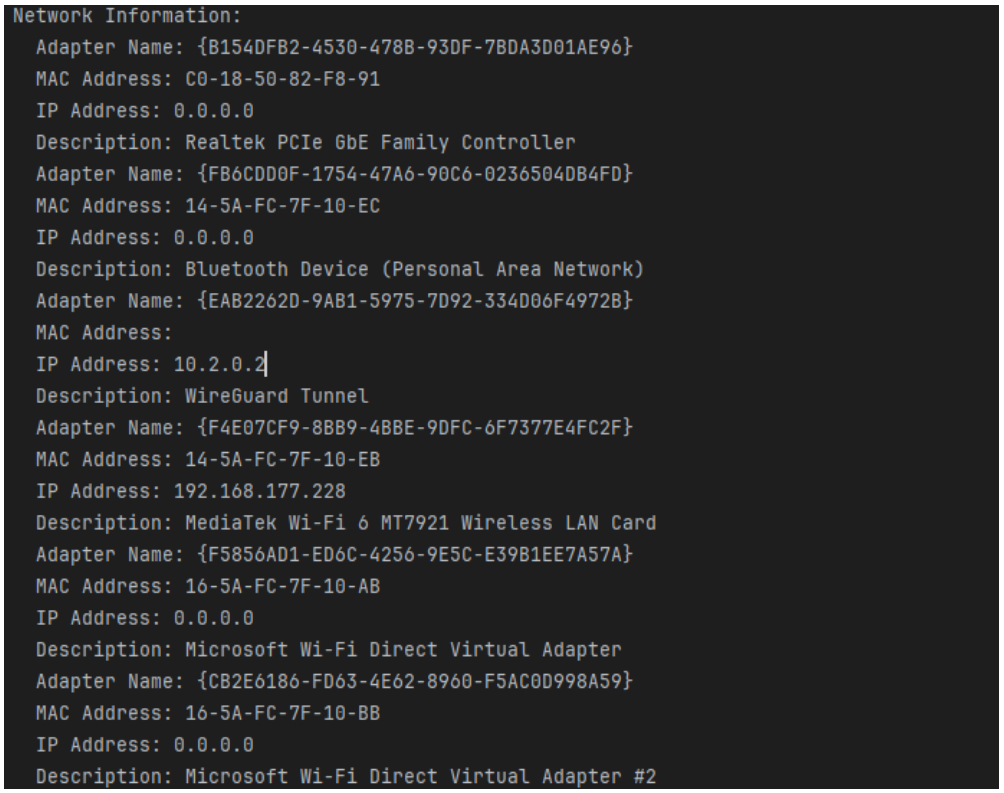

Рисунок 3.4 - Информация о сетевых адаптерах

ВЫВОД

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

Программа предоставляет следующие данные: архитектура и количество процессоров, частота процессора, информация о кеш-памяти *(L1, L2, L3*), общий объем и доступное количество оперативной и виртуальной памяти, информация об операционной системе, включая версию, номер сборки и тип системы, время непрерывной работы системы (uptime), сведения о сетевых адаптерах, включая 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, обеспечив получение и отображение данных в удобной текстовой форме. Тестирование подтвердило надежность решения и точность предоставляемой информации.

Данная 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] Function GetSystemInfo() [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo. – Дата доступа: 26.11.2024.

[2] Функция RegQueryValueEx [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regqueryvalueexa. – Дата доступа: 26.11.2024.

[3] функция GetTickCount() [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-gettickcount. – Дата доступа: 26.11.2024.

[4] Функция GetLogicalProcessorInformation () (sysinfoapi.h) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-getlogicalprocessorinformation. – Дата доступа: 27.11.2024.

ПРИЛОЖЕНИЕ А
(справочное)
Исходный код

```
#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")

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 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 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 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();
    PrintMemoryInfo();
    PrintOSVersion();
    PrintUptime();
    PrintProcessorFrequency();
    PrintCacheInfo();
    cache_sum();
    PrintNetworkInfo();
    PrintProcessesInfo();
    return 0;
}
```