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 Таргонский Д.А.

Проверил:

ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc179320137)

[2 Краткие теоретические сведения 4](#_Toc179320138)

[3 Инструментальная языковая среда 5](#_Toc179320139)

[4 Описание программного продукта 6](#_Toc179320140)

[4.1 Описание используемых функций 6](#_Toc179320141)

[4.2 Алгоритм работы программы 7](#_Toc179320142)

[5 Результат выполнения программы 8](#_Toc179320143)

[Заключение 9](#_Toc179320144)

[Список использованных источников 10](#_Toc179320145)

[Приложения А (обязательное) исходный код продукта 11](#_Toc179320146)

# **1 ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы заключается в изучении и закреплении навыков работы с механизмами управления и мониторинга системы, а также в освоении технологий многомодульных приложений. В условиях современного программирования, где управление системой и взаимодействие между модулями становятся важными аспектами разработки, необходимо понимать, как организовать эффективное взаимодействие между компонентами программного обеспечения.

В ходе выполнения лабораторной работы будут исследованы различные подходы к организации многомодульных приложений, включая использование динамических библиотек (DLL), а также изучение функций API для работы с библиотеками. Это включает в себя изучение механизмов, таких как статическое и динамическое подключение библиотек, явный и неявный импорт функций, а также управление ресурсами и взаимодействие между модулями.

Задача состоит в реализации программы, которая:

* Организует взаимодействие между модулями программы. Это позволит разделить функциональность на отдельные компоненты, что упростит разработку, тестирование и поддержку программного обеспечения.
* Обеспечивает корректное взаимодействие между модулями, включая обработку ошибок и управление ресурсами. Важно реализовать механизмы, которые будут контролировать доступ к общим ресурсам, гарантируя, что данные передаются корректно и без потерь.
* Сравнивает эффективность различных подходов к организации многомодульных приложений. В рамках работы будет проведен анализ производительности подходов, использующих статическое и динамическое подключение библиотек, что поможет понять преимущества и недостатки каждого из подходов в зависимости от условий задачи.
* Измеряет время выполнения операций взаимодействия между модулями. Программа будет фиксировать время выполнения операций в обоих подходах, что позволит провести количественный анализ и сделать выводы о том, какой подход более эффективен в различных сценариях.

Таким образом, выполнение данной лабораторной работы не только углубит теоретические знания, но и даст практические навыки, необходимые для разработки многомодульных приложений и управления системными ресурсами.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Подсистема управления и мониторинга системы в операционных системах отвечает за организацию взаимодействия между различными компонентами программного обеспечения, а также за контроль и управление системными ресурсами. Основные методы управления и мониторинга включают использование реестра, журналов событий, WMI (Windows Management Instrumentation), а также механизмов многомодульных приложений, таких как динамические библиотеки (DLL).

Реестр Windows — это иерархическая база данных, используемая для хранения настроек и параметров конфигурации системы, а также приложений. Реестр состоит из ключей и значений, которые могут содержать различные типы данных, такие как строки, числа, двоичные данные и т.д. Реестр используется для хранения информации о конфигурации оборудования, программного обеспечения, пользовательских настройках и других параметрах. Для работы с реестром в Windows предоставляется API, включающий функции для создания, чтения, записи и удаления ключей и значений.

WMI (Windows Management Instrumentation) — это инфраструктура управления и мониторинга системы, предоставляемая операционной системой Windows. WMI позволяет получать информацию о состоянии системы, такую как конфигурация оборудования, параметры операционной системы, состояние служб и другие данные. WMI использует язык запросов WQL (WMI Query Language) для получения данных из системы.

Динамические библиотеки (DLL) — это модули, которые могут быть загружены и использованы программой во время выполнения. DLL позволяют разделить функциональность программы на отдельные компоненты, что упрощает разработку, тестирование и поддержку программного обеспечения. DLL могут быть подключены статически (во время компиляции) или динамически (во время выполнения). Для работы с DLL в Windows предоставляется API, включающий функции для загрузки библиотек, получения адресов функций и вызова функций из DLL.

Многомодульные приложения — это программы, которые состоят из нескольких компонентов, взаимодействующих друг с другом. В таких приложениях каждый модуль выполняет определенную задачу, а взаимодействие между модулями организуется через интерфейсы или общие ресурсы. Многомодульные приложения позволяют упростить разработку, тестирование и поддержку программного обеспечения, а также повысить его гибкость и масштабируемость.

Таким образом, методы управления и мониторинга системы, а также технологии многомодульных приложений, обеспечивают эффективное и безопасное взаимодействие между компонентами программного обеспечения, что является ключевым аспектом разработки современных приложений.

# **3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для разработки программы был выбран язык программирования C++. Это компилируемый, высокопроизводительный язык программирования общего назначения, поддерживающий как низкоуровневые, так и высокоуровневые парадигмы программирования. C++ используется для разработки системного программного обеспечения, приложений, драйверов устройств, а также других программ, требующих высокой эффективности и прямого доступа к системным ресурсам.

В качестве интегрированной среды разработки (IDE) был выбран Microsoft Visual Studio, что является важным решением для разработчиков. Интегрированная среда разработки — это программа, которая объединяет все необходимые инструменты для написания кода, отладки и тестирования в одном интерфейсе, что значительно упрощает процесс разработки. Microsoft Visual Studio является одной из самых популярных IDE для разработки на C++. Она предлагает широкий набор функций, включая редактор кода с подсветкой синтаксиса и автозавершением, инструменты для отладки, встроенные средства для юнит-тестирования и возможности для профилирования и анализа производительности кода.

Разработка осуществляется на Microsoft Windows 10. Использование этой операционной системы позволяет эффективно работать с API и системными вызовами Windows, что важно для разработки программ, которые будут функционировать в этой среде. Windows 10 предоставляет разработчикам мощные инструменты для управления процессами и ресурсами, что критично для создания высокоэффективных приложений.

Вся работа ведется на ноутбуке, что обеспечивает мобильность и возможность работать в разных условиях — будь то офис, дом или в пути. Современные ноутбуки могут обеспечить достаточную производительность для разработки и тестирования программ, особенно если они оснащены хорошими процессорами и достаточным объемом оперативной памяти.

Таким образом, выбор Microsoft Visual Studio в качестве IDE, использование операционной системы Windows 10 и работа на ноутбуке создают оптимальные условия для разработки программ на C++. Это позволяет эффективно использовать все доступные инструменты и ресурсы, что в свою очередь способствует созданию качественного и производительного программного продукта.

4 ОПИСАНИЕ ПРОГРАММНОГО ПРОДУКТА

Программа демонстрирует взаимодействие с WMI (Windows Management Instrumentation) для получения информации о системе, такой как имя компьютера, производитель, модель, количество процессоров и объем физической памяти. Для этого используются следующие ключевые функции:

1. CoInitializeEx(): Инициализирует библиотеку COM (Component Object Model), которая необходима для работы с WMI. Эта функция задает режим многопоточности для работы с COM-объектами.
2. CoInitializeSecurity(): Устанавливает параметры безопасности для работы с COM-объектами. Это необходимо для корректного взаимодействия с WMI.
3. CoCreateInstance(): Создает экземпляр объекта IWbemLocator, который используется для подключения к пространству имен WMI.
4. IWbemLocator::ConnectServer(): Подключается к указанному пространству имен WMI (в данном случае ROOT\\CIMV2), чтобы получить доступ к системным данным.
5. CoSetProxyBlanket(): Устанавливает параметры безопасности для прокси-сервера, чтобы обеспечить корректное взаимодействие с WMI.
6. IWbemServices::ExecQuery(): Выполняет запрос к WMI с использованием языка запросов WQL (WMI Query Language). В данном случае запрос выбирает все данные из класса Win32\_ComputerSystem.
7. IEnumWbemClassObject::Next(): Извлекает объекты, возвращенные запросом, по одному за раз.
8. IWbemClassObject::Get(): Получает значения свойств объекта, таких как UserName, Name, Manufacturer, Model, NumberOfProcessors и TotalPhysicalMemory.
9. VariantClear(): Очищает ресурсы, связанные с объектом VARIANT, который используется для хранения значений свойств.
10. Release(): Освобождает ресурсы, связанные с COM-объектами, такими как IWbemLocator, IWbemServices и IEnumWbemClassObject.
11. CoUninitialize(): Завершает работу с COM-библиотекой, освобождая все ресурсы, выделенные при инициализации.

Таким образом, программа использует эти функции для взаимодействия с WMI и получения информации о системе.

**4.2 Алгоритм работы программы**

Алгоритм работы программы включает несколько ключевых этапов, которые обеспечивают корректное выполнение запросов к WMI и обработку полученных данных.

1. Инициализация COM-библиотеки: Программа начинает с вызова функции CoInitializeEx(), которая инициализирует COM-библиотеку. Это необходимо для работы с WMI.
2. Инициализация безопасности: Функция CoInitializeSecurity() устанавливает параметры безопасности для работы с COM-объектами, чтобы обеспечить корректное взаимодействие с WMI.
3. Создание объекта IWbemLocator: Функция CoCreateInstance() создает экземпляр объекта IWbemLocator, который используется для подключения к пространству имен WMI.
4. Подключение к пространству имен WMI: Метод IWbemLocator::ConnectServer() подключается к пространству имен ROOT\\CIMV2, чтобы получить доступ к системным данным.
5. Установка параметров безопасности прокси-сервера: Функция CoSetProxyBlanket() устанавливает параметры безопасности для прокси-сервера, чтобы обеспечить корректное взаимодействие с WMI.
6. Выполнение запроса к WMI: Метод IWbemServices::ExecQuery() выполняет запрос к WMI с использованием WQL. В данном случае запрос выбирает все данные из класса Win32\_ComputerSystem.
7. Обработка результатов запроса: В цикле программа извлекает объекты, возвращенные запросом, с помощью метода IEnumWbemClassObject::Next(). Для каждого объекта программа получает значения свойств, таких как UserName, Name, Manufacturer, Model, NumberOfProcessors и TotalPhysicalMemory, с помощью метода IWbemClassObject::Get().
8. Вывод информации: Программа выводит полученные значения свойств в консоль.
9. Освобождение ресурсов: После завершения работы программа освобождает ресурсы, связанные с COM-объектами, с помощью методов Release(), и завершает работу с COM-библиотекой с помощью функции CoUninitialize().

Таким образом, программа демонстрирует эффективное использование WMI для получения информации о системе. Она позволяет выполнять запросы к WMI, обрабатывать полученные данные и выводить их в консоль, что делает её полезной для диагностики и мониторинга системных параметров.

**5 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Программа, демонстрирующая взаимодействие с WMI для получения информации о системе, успешно выполняет запросы к WMI и выводит данные о компьютере, такие как имя пользователя, имя компьютера, производитель, модель, количество процессоров и объем физической памяти. В результате работы программы пользователь получает подробную информацию о системе, которая может быть полезна для диагностики и мониторинга.

На рисунке 5.1 представлен фрагмент выполнения программы, где выводятся данные о системе.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 5.1 – Фрагмент выполнения программы

Таким образом, программа демонстрирует эффективное использование WMI для получения информации о системе. Она позволяет выполнять запросы к WMI, обрабатывать полученные данные и выводить их в консоль, что делает её полезной для диагностики и мониторинга системных параметров. Результаты выполнения программы подтверждают корректность работы механизма взаимодействия с WMI и обработки системных данных.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана программа, демонстрирующая взаимодействие с WMI (Windows Management Instrumentation) для получения информации о системе. Программа позволяет выполнять запросы к WMI, обрабатывать полученные данные и выводить их в консоль. Этот проект позволил глубоко изучить основы работы с системными данными, включая использование WMI, COM-интерфейсов и обработку системных параметров.

В процессе работы были изучены ключевые функции для работы с WMI, такие как CoInitializeEx(), CoInitializeSecurity(), CoCreateInstance(), IWbemLocator::ConnectServer(),IWbemServices::ExecQuery(),IWbemClassObject::Get(), VariantClear(), Release() и CoUninitialize(). Эти функции позволили организовать взаимодействие с WMI, выполнять запросы и обрабатывать полученные данные, что является важным навыком для разработки системных приложений.

Программа демонстрирует работу с WMI, который используется для получения информации о системе, такой как имя пользователя, имя компьютера, производитель, модель, количество процессоров и объем физической памяти. Это позволило понять, как работает WMI и как можно реализовать его функциональность с использованием системных API.

Выполненная работа улучшила понимание работы с системными данными, а также подготовила к дальнейшему изучению более сложных тем, таких как управление системными ресурсами, мониторинг и диагностика. Эти навыки будут полезны в будущих проектах и помогут в разработке приложений, требующих высокой производительности и надежности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API Microsoft Software Incorp. USA. [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows Microsoft Software Incorp. Лекция 1. [Электронный ресурс]. – Режим доступа : https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка с помощью WinAPI. [Электронный ресурс]. – Режим доступа : https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <comdef.h>

#include <Wbemidl.h>

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

using namespace std;

int main() {

HRESULT hr;

hr = CoInitializeEx(0, COINIT\_MULTITHREADED);

if (FAILED(hr)) {

cerr << "Failed to initialize COM library. Error code: " << hr << endl;

return 1;

}

hr = CoInitializeSecurity(

nullptr,

-1,

nullptr,

nullptr,

RPC\_C\_AUTHN\_LEVEL\_PKT,

RPC\_C\_IMP\_LEVEL\_IMPERSONATE,

nullptr,

EOAC\_NONE,

nullptr

);

if (FAILED(hr)) {

cerr << "Failed to initialize security. Error code: " << hr << endl;

CoUninitialize();

return 1;

}

IWbemLocator\* pLocator = nullptr;

hr = CoCreateInstance(

CLSID\_WbemLocator,

0,

CLSCTX\_INPROC\_SERVER,

IID\_IWbemLocator,

(LPVOID\*)&pLocator

);

if (FAILED(hr)) {

cerr << "Failed to create IWbemLocator object. Error code: " << hr << endl;

CoUninitialize();

return 1;

}

IWbemServices\* pService = nullptr;

hr = pLocator->ConnectServer(

\_bstr\_t(L"ROOT\\CIMV2"),

nullptr,

nullptr,

0,

0,

0,

0,

&pService

);

if (FAILED(hr)) {

cerr << "Could not connect to WMI namespace. Error code: " << hr << endl;

pLocator->Release();

CoUninitialize();

return 1;

}

hr = CoSetProxyBlanket(

pService,

RPC\_C\_AUTHN\_WINNT,

RPC\_C\_AUTHZ\_NONE,

nullptr,

RPC\_C\_AUTHN\_LEVEL\_CALL,

RPC\_C\_IMP\_LEVEL\_IMPERSONATE,

nullptr,

EOAC\_NONE

);

if (FAILED(hr)) {

cerr << "Could not set proxy blanket. Error code: " << hr << endl;

pService->Release();

pLocator->Release();

CoUninitialize();

return 1;

}

IEnumWbemClassObject\* pEnumerator = nullptr;

hr = pService->ExecQuery(

bstr\_t("WQL"),

bstr\_t("SELECT \* FROM Win32\_ComputerSystem"),

WBEM\_FLAG\_FORWARD\_ONLY | WBEM\_FLAG\_RETURN\_IMMEDIATELY,

nullptr,

&pEnumerator

);

if (FAILED(hr)) {

cerr << "Query for system data failed. Error code: " << hr << endl;

pService->Release();

pLocator->Release();

CoUninitialize();

return 1;

}

IWbemClassObject\* pClassObject = nullptr;

ULONG uCount = 0;

while (pEnumerator) {

hr = pEnumerator->Next(WBEM\_INFINITE, 1, &pClassObject, &uCount);

if (0 == uCount) break;

VARIANT varProperty;

hr = pClassObject->Get(L"UserName", 0, &varProperty, 0, 0);

wprintf(L"Username: %s\n", varProperty.bstrVal);

VariantClear(&varProperty);

hr = pClassObject->Get(L"Name", 0, &varProperty, 0, 0);

wprintf(L"Computer name: %s\n", varProperty.bstrVal);

VariantClear(&varProperty);

hr = pClassObject->Get(L"Manufacturer", 0, &varProperty, 0, 0);

wprintf(L"Manufacturer: %s\n", varProperty.bstrVal);

VariantClear(&varProperty);

hr = pClassObject->Get(L"Model", 0, &varProperty, 0, 0);

wprintf(L"Model: %s\n", varProperty.bstrVal);

VariantClear(&varProperty);

pClassObject->Release();

}

pEnumerator->Release();

pService->Release();

pLocator->Release();

CoUninitialize();

return 0;

}