УДК 004.005.21

Р. В. Тихоліз
О. М. Рейда

ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ТА КЕРУВАННЯ СЛУЖБАМИ ОС WINDOWS

Вінницький національний технічний університет;

Анотація

Розроблено структуру програмної системи; розроблено алгоритм аналізу стану системних служб; розроблено модуль моніторингу, запуску, зупинки та конфігурації параметрів служб у реальному часі.

Ключові слова: служби ОС Windows, оптимізація системи, C++, Windows API, Windows Forms, моніторинг служб, системне адміністрування.

Abstract

The structure of the software system has been developed; an algorithm for analyzing the state of system services; a module for monitoring, starting, stopping, and configuring service parameters in real time has been developed.

Keywords: Windows OS services, system optimization, C++, Windows API, Windows Forms, service monitoring, system administration.

Вступ

Забезпечення стабільного та надійного функціонування операційної системи Windows [1] є критично важливим завданням для будь-якої сучасної інформаційної системи. В умовах зростання вимог до продуктивності та безпеки обчислювальних ресурсів, процес управління фоновими процесами, зокрема системними службами, набуває особливого значення. Від коректності конфігурації та стану служб залежить не лише швидкість завантаження системи, а й загальна безпека, стабільність роботи додатків та раціональне використання системних ресурсів.

Метою роботи є розробка програмного засобу WinService Manager для оптимізації робочого процесу та підвищення продуктивності використання робочих станцій [2].

Результати дослідження

Програмний застосунок реалізує графічний інтерфейс для управління службами Windows. Основна логіка базується на взаємодії користувача з кнопками (btnStart, btnStop, btnRefresh) та таблицею dataGridView1, де відображаються доступні служби. Клас ServiceManager забезпечує функціональність через методи, які викликаються при натисканні відповідних кнопок. Алгоритм включає:

1. Ініціалізацію форми та завантаження списку служб через LoadServices.
2. Обробку введених даних (наприклад, текст у txtServiceName).
3. Виклик методів StartService, StopService, RestartService або UpdateServiceInfo залежно від дії користувача.
4. Оновлення інформації про службу через UpdateServiceInfo.

Детальний алгоритм функціональних методів класу

Нижче наведено детальні алгоритми для всіх основних методів.

Алгоритм методу StartService (рисунок 1)

```
void ServiceManager::StartService(System::String^ serviceName) {
    try {
        ServiceController^ service = gcnew ServiceController(serviceName);
        if (service->Status != ServiceControllerStatus::Running) {
            service->Start();
            service->WaitForStatus(ServiceControllerStatus::Running, TimeSpan::FromSeconds(10));
        }
    }
    catch (Exception^ ex) {
        throw gcnew Exception("Помилка при запуску служби: " + ex->Message);
    }
}
```

Рисунок 1 – Механізм запуску служби

Опис роботи алгоритму:

Алгоритм методу StartService:

1. Отримує ім'я служби (serviceName).
2. Створює об'єкт ServiceController для керування службою.
3. Перевіряє, чи служба не запущена (Status != Running).
4. Якщо служба не запущена:
   - Запускає службу (Start()).
   - Чекає до 10 секунд, поки служба перейде в стан Running (WaitForStatus).
5. Якщо виникає помилка, генерує виняток із повідомленням про помилку.

Алгоритм методу StopService (рисунок 2)

```
void ServiceManager::StopService(System::String^ serviceName) {
    try {
        ServiceController^ service = gcnew ServiceController(serviceName);
        if (service->Status == ServiceControllerStatus::Running) {
            service->Stop();
            service->WaitForStatus(ServiceControllerStatus::Stopped, TimeSpan::FromSeconds(10));
        }
    }
    catch (Exception^ ex) {
        throw gcnew Exception("Програма запущена без прав адмінітрстора для операції: зупинка " + ex->Message);
    }
}
```

Рисунок 2 – Механізм зупинки служби

Опис роботи алгоритму:

1. Отримує ім'я служби (serviceName).
2. Створює об'єкт ServiceController для керування службою.
3. Перевіряє, чи служба запущена (Status == Running).
4. Якщо служба запущена:
   - Зупиняє службу (Stop()).
   - Чекає до 10 секунд, поки служба перейде в стан Stopped (WaitForStatus).
5. Якщо виникає помилка, генерує виняток із повідомленням про помилку.

Алгоритм методу RestartService (рисунок 3)

```
void ServiceManager::RestartService(System::String^ serviceName) {
    try {
        ServiceController^ service = gcnew ServiceController(serviceName);
        if (service->Status == ServiceControllerStatus::Running) {
            service->Stop();
            service->WaitForStatus(ServiceControllerStatus::Stopped, TimeSpan::FromSeconds(10));
        }
        service->Start();
        service->WaitForStatus(ServiceControllerStatus::Running, TimeSpan::FromSeconds(10));
    }
    catch (Exception^ ex) {
        throw gcnew Exception("Помилка при перезапуску служби: " + ex->Message);
    }
}
```

Рисунок 3 – Механізм перезапуску служби

Опис роботи алгоритму:
Алгоритм методу RestartService:

1. Отримує ім'я служби (serviceName).
2. Створює об'єкт ServiceController для керування службою.
3. Перевіряє, чи служба запущена (Status == Running).
4. Якщо служба запущена:
   - o Зупиняє службу (Stop()).
   - o Чекає до 10 секунд, поки служба перейде в стан Stopped (WaitForStatus).
5. Запускає службу (Start()).
6. Чекає до 10 секунд, поки служба перейде в стан Running (WaitForStatus).
7. Якщо виникає помилка, генерує виняток із повідомленням про помилку.

Алгоритм методу UpdateServiceInfo (рисунок 4)

```
void ServiceManager::UpdateServiceInfo(System::String^ technicalServiceName, System::Windows::Forms::Label^
    lblServiceInfo, System::Windows::Forms::StatusStrip^ statusStrip) {
    try {
        ServiceController^ service = gcnew ServiceController(technicalServiceName);
        ManagementObjectSearcher^ searcher = gcnew ManagementObjectSearcher(
            "SELECT * FROM Win32_Service WHERE Name = '" + technicalServiceName + "'");

        String^ serviceDescription = "Опис не знайдено";
        bool descriptionFound = false;

        for each (ManagementObject ^ serviceObj in searcher->Get()) {
            if (serviceObj->GetPropertyValue("Description") != nullptr) {
                serviceDescription = serviceObj->GetPropertyValue("Description")->ToString();
                descriptionFound = true;
            }
        }

        lblServiceInfo->Text = "Служба: " + technicalServiceName + "\nТехнічне ім'я служби: " + service->ServiceName +
            "\nСтатус: " + service->Status.ToString() +
            "\nТип запуску: " + service->StartType.ToString() +
            "\nОпис: " + serviceDescription;

        if (statusStrip->Items->Count > 0) {
            statusStrip->Items[0]->Text = technicalServiceName;
        }
        else {
            ToolStripStatusLabel^ statusLabel = gcnew ToolStripStatusLabel();
            statusLabel->Text = technicalServiceName;
            statusStrip->Items->Add(statusLabel);
        }
    }
    catch (Exception^ ex) {
        lblServiceInfo->Text = "Помилка при отриманні інформації: " + ex->Message;
    }
}
```

Рисунок 4 – Механізм оновлення списку служб

Опис роботи алгоритму:

1. Отримує технічне ім'я служби (technicalServiceName), мітку (lblServiceInfo) та статусну панель (statusStrip).
2. Створює об'єкт ServiceController для керування службою.
3. Створює ManagementObjectSearcher для пошуку інформації про службу в Win32_Service за ім'ям.
4. Задає початкове значення опису служби ("Опис не знайдено") та прапорець descriptionFound = false.
5. Для кожного об'єкта, отриманого з пошуку:

o Якщо властивість Description не порожня, оновлює serviceDescription і встановлює descriptionFound = true.

6. Оновлює текст мітки lblServiceInfo інформацією:

o Ім'я служби, технічне ім'я, статус, тип запуску, опис.

7. Оновлює statusStrip:

o Якщо є елементи, встановлює текст першого елемента в technicalServiceName.

o Якщо немає елементів, створює новий ToolStripStatusLabel із текстом technicalServiceName і додає до statusStrip.

8. У разі помилки встановлює текст lblServiceInfo із повідомленням про помилку.

Головним алгоритмом, який отримує масив служб, фільтрує їх, ініціалізує форму та завантажує список служб через та дозволяє оперувати та керувати різними службами, є LoadServices (рис.5).

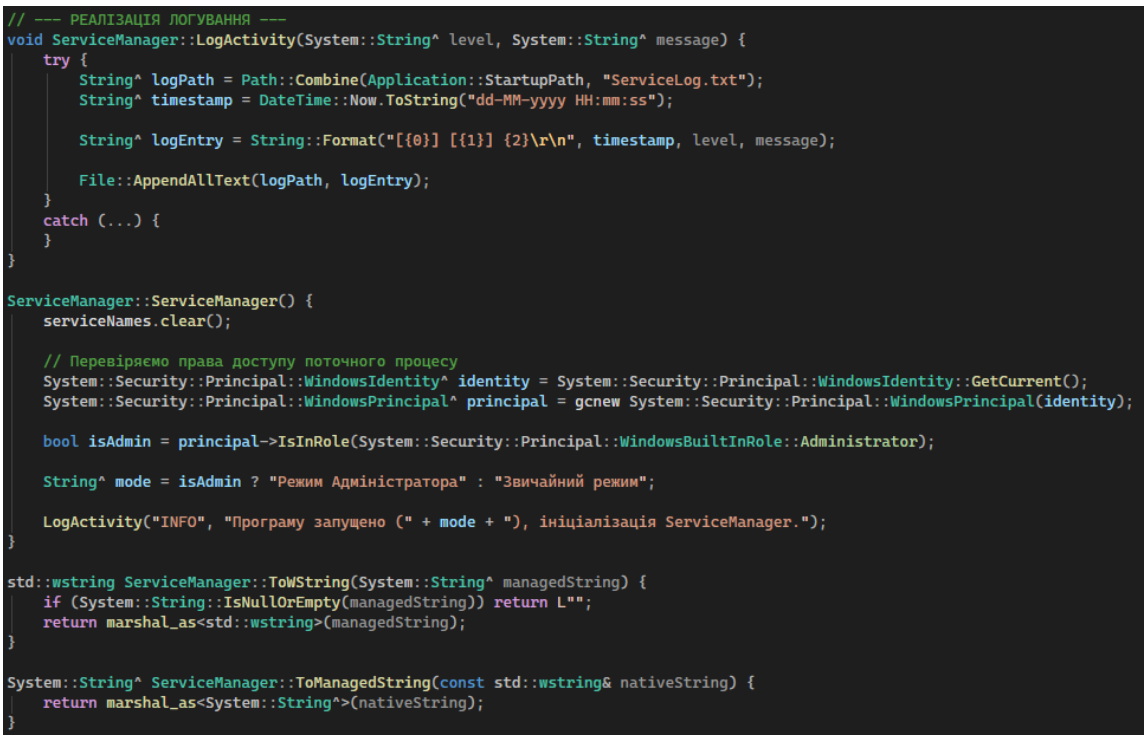
```
void ServiceManager::LoadServices(System::String^ searchText, System::Windows::Forms::DataGridView^ dataGridView) {
    dataGridView->Columns->Clear();
    dataGridView->Rows->Clear();

    dataGridView->Columns->Add("ServiceName", "Назва служби");
    dataGridView->Columns->Add("TechnicalServiceName", "Технічна назва");
    dataGridView->Columns->Add("Status", "Статус");

    dataGridView->AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode::Fill;
    dataGridView->RowHeadersVisible = false;

    array<ServiceController^>^ services = ServiceController::GetServices();
    std::wstring searchTextW = ToWString(searchText->ToLower());

    for each (ServiceController ^ service in services) {
        std::wstring serviceNameW = ToWString(service->ServiceName->ToLower());
        std::wstring displayNameW = ToWString(service->DisplayName->ToLower());

        if (serviceNameW.find(searchTextW) != std::wstring::npos ||
            displayNameW.find(searchTextW) != std::wstring::npos) {
            dataGridView->Rows->Add(service->DisplayName, service->ServiceName, service->Status.ToString());
        }
    }
}
```

Рисунок 5 – Механізм реалізації ініціалізації списку служб

Алгоритм методу LogActivity (рисунок 6)

Опис роботи алгоритму ініціалізації та первинного логування (Конструктор ServiceManager):

1. Очищує внутрішній масив імен служб (serviceNames) для підготовки до роботи.
2. Звертається до системи (через WindowsIdentity::GetCurrent()), щоб отримати токен безпеки поточного користувача.
3. Створює об'єкт WindowsPrincipal для перевірки прав доступу поточного процесу.



```
// --- РЕАЛІЗАЦІЯ ЛОГУВАННЯ ---
void ServiceManager::LogActivity(System::String^ level, System::String^ message) {
    try {
        String^ logPath = Path::Combine(Application::StartupPath, "ServiceLog.txt");
        String^ timestamp = DateTime::Now.ToString("dd-MM-yyyy HH:mm:ss");

        String^ logEntry = String::Format("[{0}] [{1}] {2}\r\n", timestamp, level, message);

        File::AppendAllText(logPath, logEntry);
    }
    catch (...) {
    }
}

ServiceManager::ServiceManager() {
    serviceNames.clear();

    // Перевіряємо права доступу поточного процесу
    System::Security::Principal::WindowsIdentity^ identity = System::Security::Principal::WindowsIdentity::GetCurrent();
    System::Security::Principal::WindowsPrincipal^ principal = gcnew System::Security::Principal::WindowsPrincipal(identity);

    bool isAdmin = principal->IsInRole(System::Security::Principal::WindowsBuiltInRole::Administrator);

    String^ mode = isAdmin ? "Режим Адміністратора" : "Звичайний режим";

    LogActivity("INFO", "Програму запущено (" + mode + "), ініціалізація ServiceManager.");
}

std::wstring ServiceManager::ToWString(System::String^ managedString) {
    if (System::String::IsNullOrEmpty(managedString)) return L"";
    return marshal_as<std::wstring>(managedString);
}

System::String^ ServiceManager::ToManagedString(const std::wstring& nativeString) {
    return marshal_as<System::String^>(nativeString);
}
```

Рисунок 6 – Механізм логування

4. Виконує перевірку наявності у процесу ролі Адміністратора (WindowsBuiltInRole::Administrator) та зберігає результат у логічну змінну isAdmin.
5. Формує текстовий рядок mode, який набуває значення "Режим Адміністратора" або "Звичайний режим" залежно від результату перевірки привілеїв.
6. Викликає метод LogActivity, передаючи рівень "INFO" та сформоване повідомлення про успішний запуск застосунку із зазначенням поточного режиму роботи.

Опис роботи алгоритму збереження логів (метод LogActivity):

1. Отримує вхідні параметри: рівень важливості події (level, наприклад, "INFO", "WARNING", "ERROR") та текст самого повідомлення (message).
2. Визначає абсолютний шлях до файлу журналу (logPath), об'єднуючи шлях до директорії запуску застосунку (Application::StartupPath) та назву файлу "ServiceLog.txt".
3. Отримує поточну системну дату та час (DateTime::Now) і форматує їх у текстову змінну (timestamp) за стандартом "dd-MM-yyyy HH:mm:ss".
4. Формує фінальний рядок запису (logEntry) за суворим шаблоном: [Час] [Рівень] Повідомлення, додаючи в кінці символи переходу на новий рядок (\r\n).
5. Викликає системний метод File::AppendAllText для запису рядка у файл. Якщо файл не існує — він створюється автоматично; якщо існує — новий текст дописується в кінець без видалення попередніх записів.
6. У разі виникнення помилки запису (наприклад, файл тимчасово заблоковано іншим системним процесом або відсутні права на запис у директорію), блок catch перехоплює виняток і дозволяє програмі продовжити роботу без аварійного завершення (запобігає крашу).

Висновки

Комплексне поєднання засобів ServiceController, розширених запитів WMI, контролю безпеки та механізму логування формує стійку та завершену програмну систему. Розроблений застосунок є повноцінним та безпечним інструментом для адміністрування служб Windows, який ефективно перетворює складні системні виклики на інтуїтивно зрозумілі дії для кінцевого користувача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Офіційний сайт Microsoft Windows [Електронний ресурс] / Microsoft. – 2026. – Режим доступу: https://www.microsoft.com/uk-ua/windows?r=1.

2. What is Microsoft Management Console? [Електронний ресурс] / Microsoft Learn. – 2026. – Режим доступу: https://learn.microsoft.com/en-us/troubleshoot/windows-server/system-management-components/what-is-microsoft-management-console.

Тихоліз Ростислав Віталійович — студент групи ПІ-22б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: 7rostyk@gmail.com

Науковий керівник: ***Рейда Олександр Миколайович*** — доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця.

Tykholiz Rostyslav V. — Department of Information Technology and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, email : 7rostyk@gmail.com

Supervisor: ***Reida Oleksandr M.*** — Dr. Sc. (Eng.), Associate Professor of the Department of Software, Vinnytsia National Technical University, Vinnytsia.