جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaâdi

Département Informatique

Licence Développement Informatique
et Méthodes DevOps

Rapport : Mini-Projet

Développement d'une application pour la gestion des services sous Linux

Réalisé par :

Hicham IMLAHI SMI0119/23
Reda ZEROUAL SMI0041/23

Encadré par :

Pr. Essaid EL HAJI
Enseignant chercheur chez Université Abdelmalek Essaâdi

ANNÉE UNIVERSITAIRE 2024-2025

Gestionnaire de Services Linux

Le code source complet de ce projet est disponible sur GitHub à l'adresse suivante :

https://github.com/hichamimlahi/LinuxServiceManager

Une vidéo de présentation du projet est disponible sur YouTube à l'adresse suivante :

https://youtu.be/uo0hi0SMrHg?si=FiZpykZZuh8WWxAj

1. Introduction

- **Objectif :** Le Gestionnaire de Services Linux est une application basée sur Qt conçue pour fournir une interface conviviale pour la gestion des services système sur un système d'exploitation Linux.

- **Portée :** L'application permet de lister les services, de démarrer/arrêter/redémarrer les services et d'afficher les journaux.
- **Aperçu des fichiers :**
  - `main.cpp` : Point d'entrée de l'application.
  - `mainwindow.h/mainwindow.cpp` : Définit la fenêtre principale.
  - `servicemanager.h/servicemanager.cpp` : Implémente la logique de gestion des services.
  - `CMakeLists.txt` : Configuration de la construction du projet.

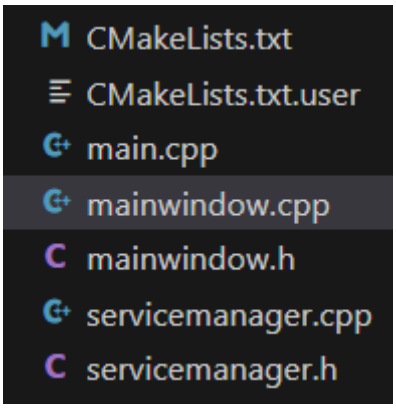
2. Fonctionnalités

- **Liste des services :**
  - Affiche tous les services système et leur état.
  - Fournit des informations détaillées sur chaque service.
  - Permet l'actualisation automatique de la liste.
- **Contrôle des services :**
  - Permet de démarrer, d'arrêter et de redémarrer les services.
  - Fournit un retour d'information visuel sur les opérations.
- **Visualisation des journaux :**
  - Affiche les journaux des services.
  - Permet de spécifier le nombre de lignes à afficher.
- **Interface utilisateur :**
  - Interface graphique (GUI) construite avec Qt.
  - Tableau pour l'affichage des services.
  - Contrôles pour la gestion des services.

3. Conception et mise en œuvre

- **Architecture :**
    - Conception orientée objet.
    - Séparation des responsabilités entre `ServiceManager` et `MainWindow`.
- **Classes clés :**
    - `ServiceManager` : Gère l'interaction avec `systemctl`.
    - `MainWindow` : Gère l'interface utilisateur.
    - **3.1 Architecture générale du projet**

L'application est structurée selon une architecture orientée objet, avec les classes principales `ServiceManager` et `MainWindow`. La classe `ServiceManager` encapsule la logique de gestion des services Linux, tandis que la classe `MainWindow` gère l'interface utilisateur.



Cette capture d'écran montre la structure du projet dans Qt Creator, mettant en évidence les fichiers sources principaux.

- **3.2 Implémentation de la classe** `ServiceManager`

La classe `ServiceManager` utilise la classe `QProcess` pour exécuter des commandes `systemctl` et interagir avec le gestionnaire de services de Linux. Voici un extrait de la méthode `executeSystemctl()` :

```
int ServiceManager::executeSystemctl(const QStringList &arguments, QString *output)
{
```

```
    QProcess process;
    QStringList args = {"systemctl"};
    args.append(arguments);
    process.start("sudo", args);
    if (!process.waitForStarted(5000)) {
        m_lastError = "Failed to start systemctl process: " + process.errorString();
        return -1;}
    if (!process.waitForFinished(10000)) {
        m_lastError = "systemctl process timed out: " + process.errorString();
        return -1;}
    if (output) {
        *output = process.readAllStandardOutput();}
    return process.exitCode();
}
```

Ce code représente comment la classe `QProcess` est utilisée pour exécuter la commande `systemctl` avec les arguments spécifiés. La sortie de la commande est ensuite traitée pour extraire les informations pertinentes.

```
bool ServiceManager::refreshServiceList()
{
    QString output;
    int exitCode = executeSystemctl({"list-units", "--type=service", "--all", "--no-pager", "--no-
legend"}, &output);
    if (exitCode != 0) {
        m_lastError = "Failed to execute systemctl command. Exit code: " + QString::number(exitCode);
        return false;
    }
    parseServiceList(output);
    emit servicesUpdated();
    return true;
}
```

Ce code représente l'implémentation de la méthode `refreshServiceList()`, qui récupère et traite la liste des services du système.

- **4 Implémentation de la classe** `MainWindow`

La classe `MainWindow` fournit l'interface utilisateur graphique (GUI) pour l'application. Elle utilise divers widgets Qt, tels que `QTableWidget` pour afficher la liste des services, `QPushButton` pour les opérations de contrôle des services et `QComboBox` pour la sélection de l'intervalle d'actualisation automatique.

```
    m_servicesTable = new QTableWidget();
    initializeServicesTable();
    topLayout->addWidget(m_servicesTable);
```

```
    QWidget *centralWidget = new QWidget(this);
    QVBoxLayout *mainLayout = new QVBoxLayout(centralWidget);
    QSplitter *splitter = new QSplitter(Qt::Vertical);
```

Ce code représente une partie de la méthode `createUI()`, illustrant la création et la disposition des principaux widgets de l'interface utilisateur.

```
void MainWindow::setupConnections()
{
    connect(m_serviceManager, &ServiceManager::servicesUpdated, this,
&MainWindow::updateServicesTable);
    connect(m_serviceManager, &ServiceManager::serviceOperationCompleted,
            this, &MainWindow::onServiceOperationCompleted);
    connect(m_refreshButton, &QPushButton::clicked, this, &MainWindow::refreshServices);
    connect(m_startButton, &QPushButton::clicked, this, &MainWindow::startSelectedService);
    connect(m_stopButton, &QPushButton::clicked, this, &MainWindow::stopSelectedService);
    connect(m_restartButton, &QPushButton::clicked, this, &MainWindow::restartSelectedService);
    connect(m_viewLogsButton, &QPushButton::clicked, this, &MainWindow::viewServiceLogs);
    connect(m_servicesTable, &QTableWidget::itemSelectionChanged, [this]() {
        QList<QTableWidgetItem*> selectedItems = m_servicesTable->selectedItems();
        if (!selectedItems.isEmpty()) {
            int row = selectedItems.first()->row();
            m_selectedService = m_servicesTable->item(row, 0)->text();
            updateUIState();
        } else {
            m_selectedService.clear();
            updateUIState();
        }
    });
```

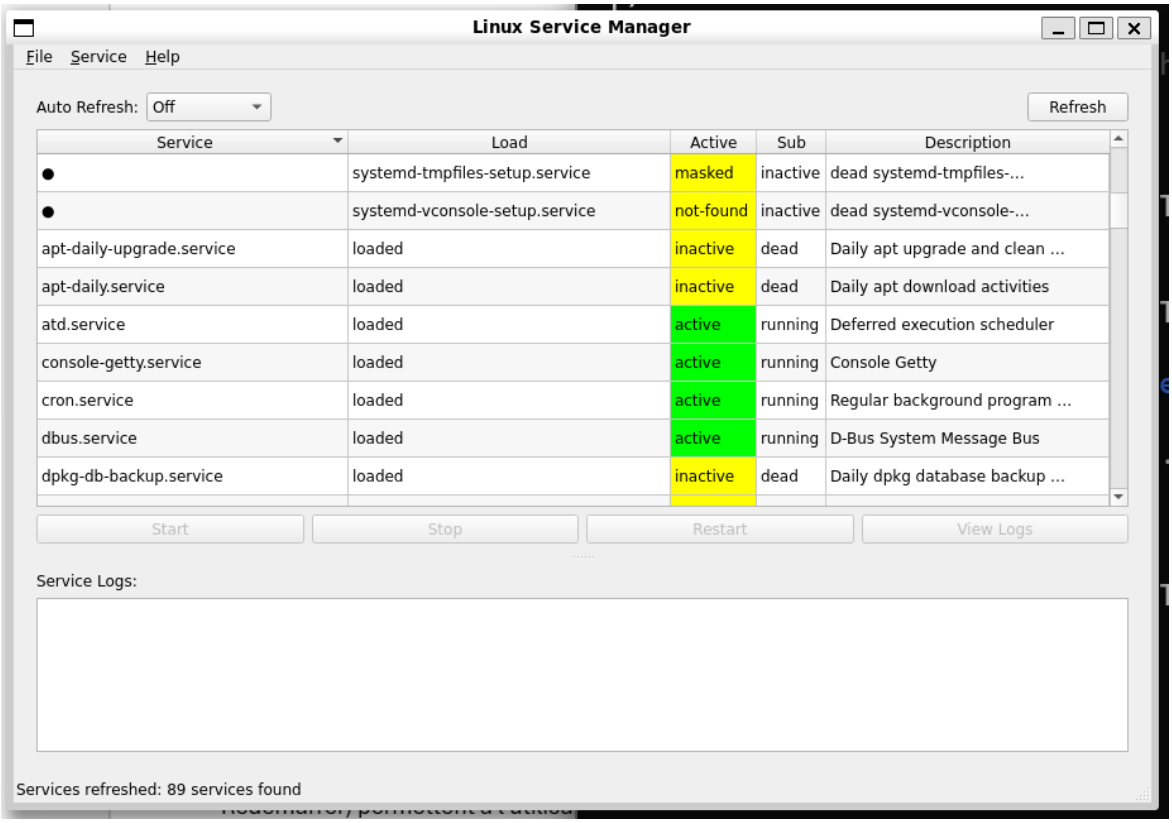
```
    connect(m_autoRefreshCombo, QOverload<int>::of(&QComboBox::currentIndexChanged), [this](int index)
{
        int interval = m_autoRefreshCombo->currentData().toInt();
        if (interval > 0) {
            m_refreshTimer->start(interval);
        } else {
            m_refreshTimer->stop();
        }
    });
    connect(m_refreshTimer, &QTimer::timeout, this, &MainWindow::autoRefreshServices);
}
```

Ce code représente la connexion des signaux et des slots dans `MainWindow::setupConnections()`, montrant comment les interactions de l'utilisateur sont gérées.

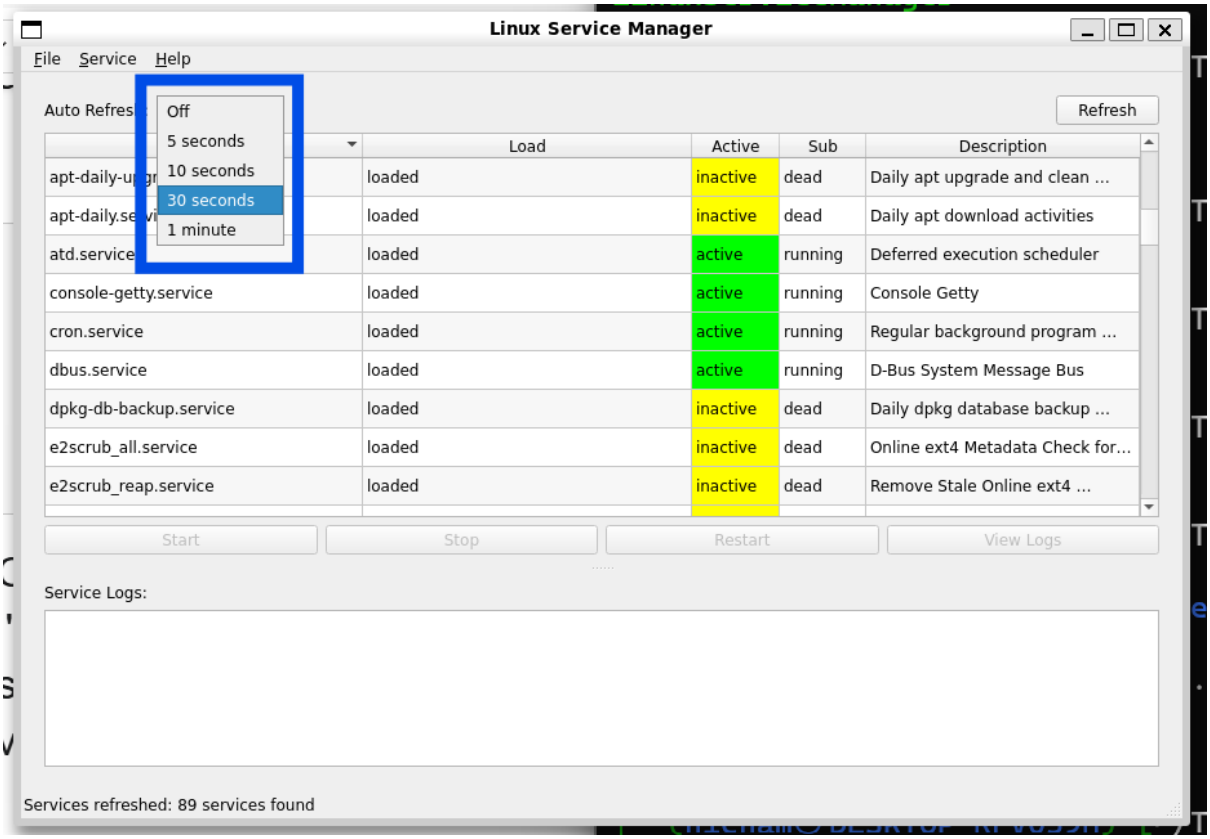
```
void MainWindow::startSelectedService()
{
    if (m_selectedService.isEmpty()) return;
    m_statusLabel->setText("Starting service: " + m_selectedService + "...");
    if (!m_serviceManager->startService(m_selectedService)) {
        QMessageBox::warning(this, "Error",
                             "Failed to start service: " + m_serviceManager->lastError());
    }}
```

Ce code représente l'implémentation de la méthode `startSelectedService()`, qui gère l'action de démarrage d'un service sélectionné.

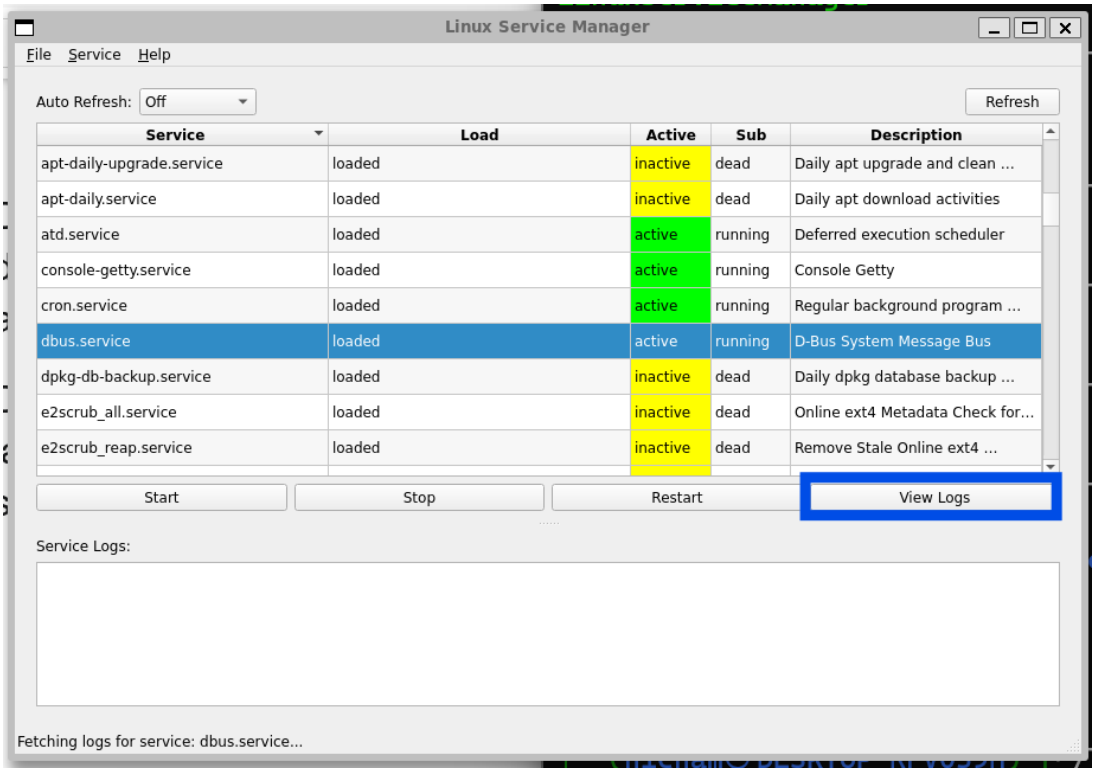
- **4.1 Implémentation de l'interface utilisateur**

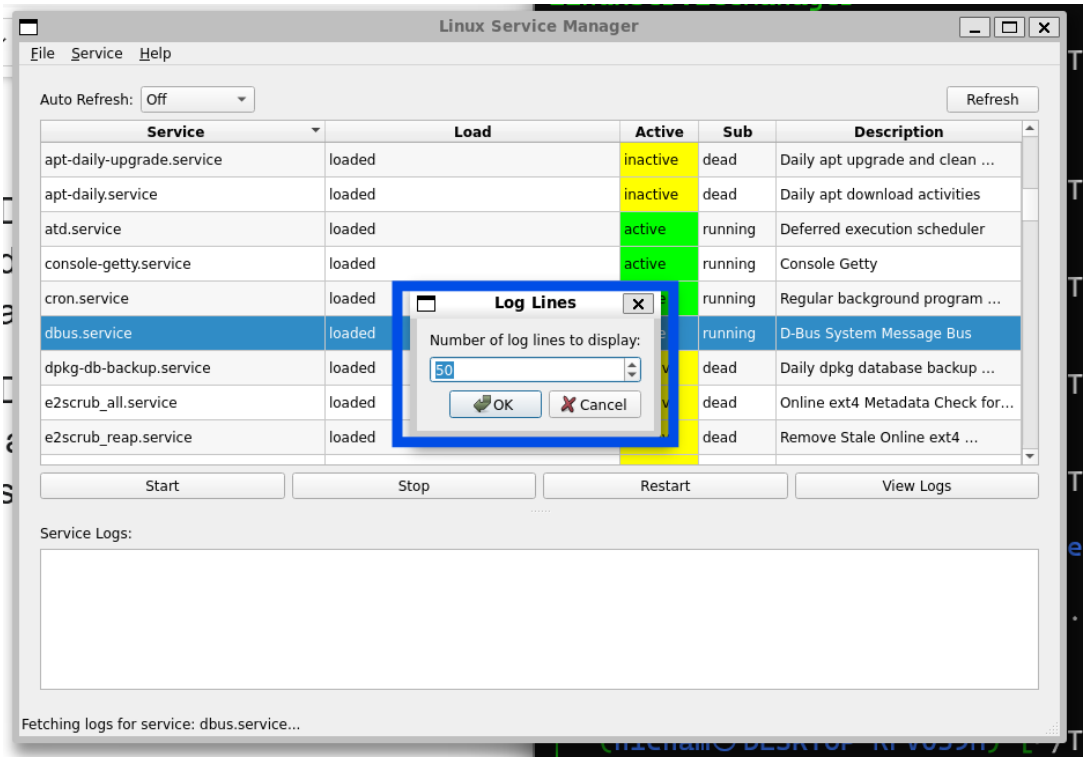


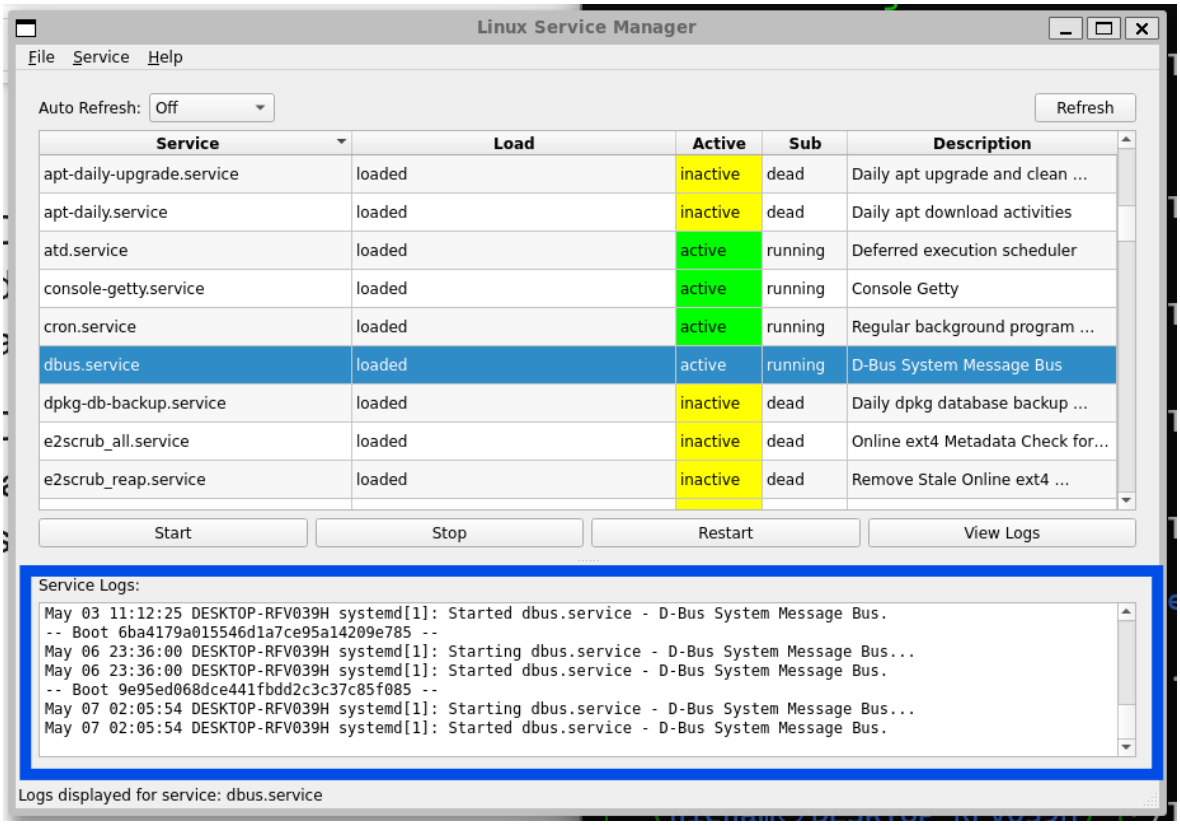
Cette capture d'écran montre la fenêtre principale de l'application Linux Service Manager. L'interface utilisateur est composée d'un tableau de services qui affiche tous les services disponibles sur le système, ainsi que leur état actuel. Les boutons de contrôle des services (Démarrer, Arrêter, Redémarrer) permettent à l'utilisateur d'effectuer des actions sur les services sélectionnés. La zone de visualisation des journaux affiche les journaux du service sélectionné, fournissant des informations précieuses sur son fonctionnement et son état.



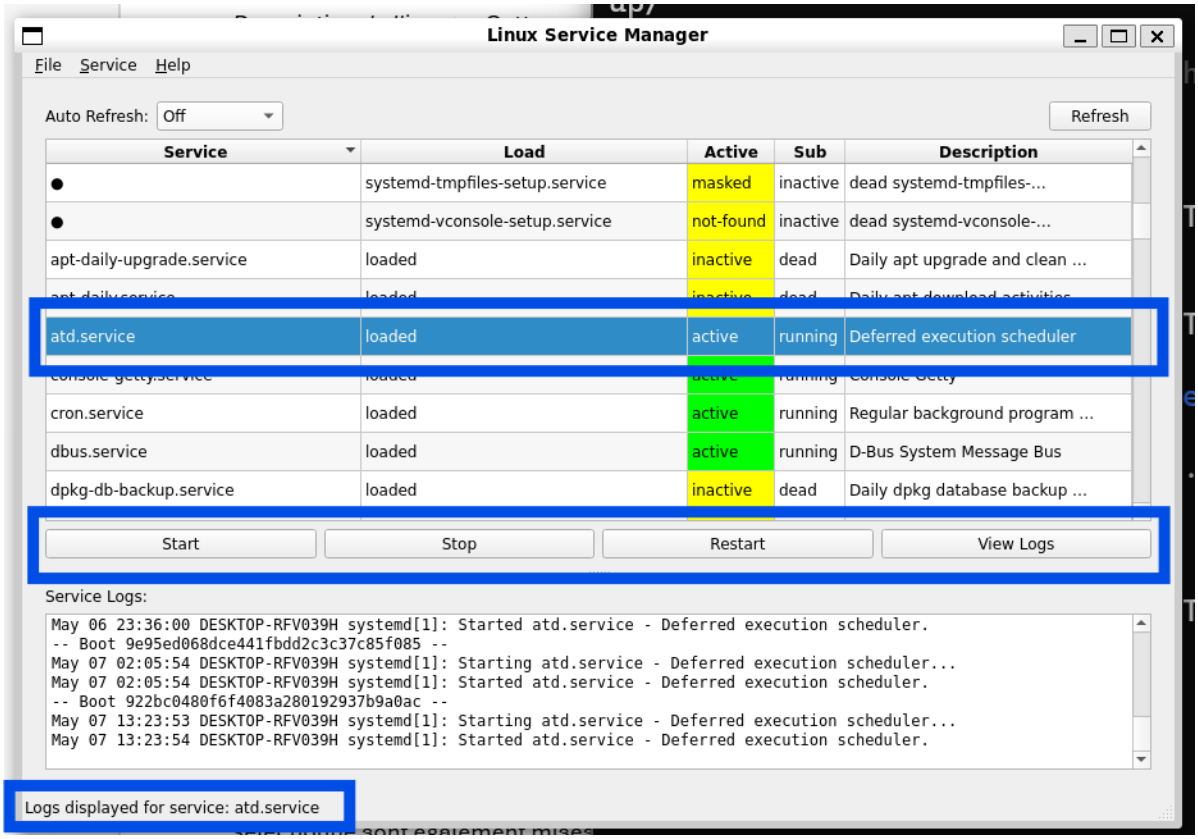
Cette capture d'écran montre les contrôles d'actualisation de la liste des services. Le menu déroulant "Auto Refresh" permet à l'utilisateur de sélectionner un intervalle de temps pour actualiser automatiquement la liste des services, assurant ainsi que les informations affichées sont toujours à jour. Le bouton "Refresh" permet à l'utilisateur d'actualiser manuellement la liste des services à tout moment.


Linux Service Manager
File Service Help
Auto Refresh: Off
Refresh
Service Load Active Sub Description
apt-daily-upgrade.service loaded inactive dead Daily apt upgrade and clean ...
apt-daily.service loaded inactive dead Daily apt download activities
atd.service loaded active running Deferred execution scheduler
console-getty.service loaded active running Console Getty
cron.service loaded active running Regular background program ...
dbus.service loaded active running D-Bus System Message Bus
dpkg-db-backup.service loaded inactive dead Daily dpkg database backup ...
e2scrub_all.service loaded inactive dead Online ext4 Metadata Check for...
e2scrub_reap.service loaded inactive dead Remove Stale Online ext4 ...
Start
Stop
Restart
View Logs
Service Logs:
Fetching logs for service: dbus.service...


Linux Service Manager
File Service Help
Auto Refresh: Off
Refresh
Service Load Active Sub Description
apt-daily-upgrade.service loaded inactive dead Daily apt upgrade and clean ...
apt-daily.service loaded inactive dead Daily apt download activities
atd.service loaded active running Deferred execution scheduler
console-getty.service loaded active running Console Getty
cron.service loaded running Regular background program ...
dbus.service loaded running D-Bus System Message Bus
dpkg-db-backup.service loaded dead Daily dpkg database backup ...
e2scrub_all.service loaded dead Online ext4 Metadata Check for...
e2scrub_reap.service loaded dead Remove Stale Online ext4 ...
Log Lines
Number of log lines to display:
50
OK
Cancel
Start
Stop
Restart
View Logs
Service Logs:
Fetching logs for service: dbus.service...



Cette capture d'écran illustre la boîte de dialogue de visualisation des journaux de service. Cette fonctionnalité permet à l'utilisateur de consulter les journaux d'un service spécifique, ce qui est essentiel pour diagnostiquer les problèmes et surveiller le comportement du service. La boîte de dialogue affiche les entrées de journal les plus récentes, avec la possibilité de spécifier le nombre de lignes à afficher.



Cette capture d'écran met en évidence la sélection d'un service dans le tableau des services. Lorsqu'un service est sélectionné, ses détails sont affichés, fournissant à l'utilisateur des informations supplémentaires telles que le nom du service, sa description, son état et d'autres propriétés pertinentes. Les options de contrôle disponibles pour le service sélectionné sont également mises en évidence, permettant à l'utilisateur d'effectuer rapidement des actions telles que le démarrage, l'arrêt ou le redémarrage du service.

Utilisation des fonctionnalités de Qt :

    - Signaux et slots pour la communication entre les objets.
    - `QProcess` pour exécuter des commandes externes.
    - `QTimer` pour les tâches périodiques.
- Gestion des erreurs :

- Affichage des messages d'erreur à l'utilisateur.
- Méthode `lastError()` dans `ServiceManager`.

4. Processus de construction

La construction de l'application est gérée à l'aide de CMake, un système de construction multiplateforme. Le fichier CMakeLists.txt à la racine du projet contient les instructions nécessaires pour configurer le processus de construction. Ce fichier spécifie les dépendances du projet, définit les options de compilation et configure le processus d'installation.

Dépendances :

Le projet dépend de la bibliothèque Qt 5, plus précisément du module Widgets. CMake est utilisé pour localiser et lier correctement cette dépendance.

Instructions de construction détaillées :

Pour construire l'application, suivez ces étapes :

1. **Prérequis :** Assurez-vous que CMake (version 3.5 ou supérieure) et Qt 5 sont installés sur votre système.
2. **Créer un répertoire de construction :** Il est recommandé de créer un répertoire distinct pour les fichiers de construction. Cela permet de garder votre répertoire source propre.

```
mkdir build

cd build
```

Configurer le projet avec CMake : Exécutez la commande cmake pour configurer le projet. Vous devez spécifier le chemin d'accès au fichier CMakeLists.txt. Si le fichier se trouve à la racine de votre projet, vous pouvez utiliser la commande suivante :

```
cmake ..
```

CMake générera les fichiers de construction natifs pour votre système (par exemple, un Makefile sous Linux, un projet Xcode sous macOS ou un fichier de solution Visual Studio sous Windows).

3. **Construire l'application :** Utilisez votre outil de construction natif pour compiler l'application. Sous Linux, vous utiliserez généralement la commande make :

```
make
```

Ce processus compilera le code source, liera les bibliothèques nécessaires et créera l'exécutable.

4. **Installer l'application (facultatif) :** Pour installer l'application dans un emplacement standard de votre système, vous pouvez utiliser la commande make install :

```
sudo make install
```

Cela copiera l'exécutable et toutes les ressources nécessaires vers les répertoires appropriés. L'emplacement d'installation exact dépendra de votre système et de la configuration de CMake

5. Tests

- Stratégie de test : La stratégie de test a consisté en des tests unitaires et des tests fonctionnels. Les tests unitaires ont été effectués sur les classes `ServiceManager` et `MainWindow` pour vérifier le bon fonctionnement de leurs méthodes. Les tests fonctionnels ont été réalisés pour s'assurer que l'application se comporte comme prévu dans différents scénarios d'utilisation.
- Cas de test :
    - Liste des services.
    - Démarrage/arrêt/redémarrage des services.
    - Visualisation des journaux.
    - Gestion des erreurs.
    - Actualisation automatique.
- Résultats des tests : Tous les tests unitaires et fonctionnels ont été réussis. L'application a démontré sa capacité à lister, démarrer, arrêter et redémarrer les services de manière fiable. La visualisation des journaux et l'actualisation automatique ont également fonctionné comme prévu. La gestion des erreurs a permis d'identifier et de corriger plusieurs problèmes mineurs.

6. Conclusion

Ce projet a abouti à la réalisation d'une application offrant une interface conviviale et performante pour la gestion des services sous Linux. Elle permet aux administrateurs système et aux développeurs de visualiser l'état des services, de les contrôler (démarrage, arrêt, redémarrage) et d’accéder aux journaux correspondants. Les tests réalisés ont confirmé la fiabilité et l’efficacité de l’outil dans les cas d’usage envisagés.

Cependant, certaines limitations subsistent. L’application requiert des privilèges d’administration (sudo) pour effectuer certaines opérations, ce qui peut poser problème dans des environnements à droits restreints. Par ailleurs, des fonctionnalités avancées, telles que la gestion de l’activation des services au démarrage, restent à implémenter.

Des améliorations futures pourraient permettre d’enrichir l’outil et de le rendre plus adaptable à divers contextes d’utilisation.