ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИМЕНИ И.М. ГУБКИНА»

ФАКУЛЬТЕТ КОМПЛЕКСНОЙ БЕЗОПАСНОСТИ ТЭК
КАФЕДРА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №14

по дисциплине «Специализированные языки и технологии программирования»

на тему «Использование D-Bus в ОС Аврора.»

Выполнил студент:
группы КА-22-06
Воронин Алексей Дмитриевич

Преподаватель:
Греков Владимир Сергеевич

Москва, 2026

Оглавление

Цель работы

- Изучить архитектуру D-Bus и его применение в ОС Аврора
- Освоить принципы межпроцессного взаимодействия через D-Bus
- Разработать расширение функциональности существующего примера

Задание 1: Анализ примера проекта

- Изучить структуру проекта projects/dbus_services_example_aurora · main · omprussia / Education / Разработка приложений на Qt · GitLab.
- Определить основные компоненты системы.
- Проанализировать существующие интерфейсы и методы.

Проект представляет собой простое приложение для Aurora OS, демонстрирующее взаимодействие с системными D-Bus сервисами. Основной экран приложения описан в файле MainPage.qml.

В проекте представлено три интерфейса:

1. com.jolla.settings.ui. Через данный интерфейс вызывается метод showSettings, который открывает главное окно настроек системы.

2. com.nokia.profiled. Метод set_profile, которому передаётся строковый аргумент "silent" – это переключает устройство в беззвучный режим.

3. org.PulseAudio.ServerLookup1. В отличие от предыдущих, здесь не вызывается метод, а считывается свойство Address, которое содержит адрес D-Bus сокета основного сервера PulseAudio

Задание 2: Реализация дополнительного функционала (на выбор)

- Получение информации о системе
- Реализация сложного типа данных для передачи
- Создание механизма подписки на системные события
- Разработка сервиса для работы с аппаратными функциями

Выбран вариант реализации получения информации о системе. Для этого задействован системный сервис ru.omp.deviceinfo, предоставляемый платформой ОС Аврора

Задание для самостоятельной работы

- Добавить аутентификацию.
- Реализовать сервис для отправки системных уведомлений.

В приложении реализована функция аутентификации пользователя по фиксированному паролю. Интерфейс входа содержит поле PasswordField для ввода пароля и кнопку «Войти». При совпадении введённого значения с заданным устанавливается флаг authenticated = true, что открывает доступ к основному функционалу.

Функция отправки уведомлений реализована с помощью компонента Notification из модуля Nemo.Notifications. Пользователь может ввести произвольный текст в поле TextField, после чего при нажатии кнопки «Отправить уведомление» создаётся уведомление с заголовком «Пользовательское уведомление» и введённым текстом.

Итоговый код в MainPage.qml:

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import Nemo.DBus 2.0
import Nemo.Notifications 1.0

Page {
    id: page
    objectName: "mainPage"
    allowedOrientations: Orientation.All

    property bool authenticated: false
    property string systemInfo: ""
    property string notificationText: ""

    Notification { id: notification }
```

```
DBusInterface {
    id: deviceInfoInterface
    bus: DBus.SystemBus
    service: 'ru.omp.deviceinfo'
    path: '/ru/omp/deviceinfo/Features'
    iface: 'ru.omp.deviceinfo.Features'
}


function authenticate() {
    if (passwordField.text === "0000") {
        authenticated = true
        notification.summary = "Успех"
        notification.body = "Вы успешно авторизовались"
        notification.publish()
    } else {
        notification.summary = "Ошибка"
        notification.body = "Неверный пароль"
        notification.publish()
    }
}


function updateSystemInfo() {
    systemInfo = "Запрос данных..."
    var info = ""
```

```
try {
   var certified = deviceInfoInterface.getProperty("isOsCertified")
   info += "Сертифицированная ОС: " + (certified ? "Да" : "Нет") +
"\n"
} catch(e) {
   info += "Сертифицированная ОС: ошибка\n"
}

try {
   var emulated = deviceInfoInterface.getProperty("isOsEmulated")
   info += "Эмулируемая ОС: " + (emulated ? "Да" : "Нет") + "\n"
} catch(e) {
   info += "Эмулируемая ОС: ошибка\n"
}

try {
   var osType = deviceInfoInterface.getProperty("osType")
   info += "Битность ОС: " + osType + "\n"
} catch(e) {
   info += "Битность ОС: ошибка\n"
}


var pendingCalls = 0
var methods = [
   { name: "getOsVersion", desc: "Версия ОС" },
   { name: "getDeviceModel", desc: "Модель устройства" },
```

```
    { name: "getScreenResolution", desc: "Разрешение экрана" },
    { name: "hasWlan", desc: "Наличие WLAN" },
    { name: "getRamTotalSize", desc: "Общий объём ОЗУ (байт)" }
]

function handleResult(methodDesc, result) {
    var value = result
    if (typeof value === "boolean") {
        value = value ? "Да" : "Нет"
    }
    info += methodDesc + ": " + value + "\n"
    pendingCalls--
    if (pendingCalls === 0) {
        systemInfo = info
    }
}

function handleError(methodDesc, error) {
    info += methodDesc + ": ошибка (" + error + ")\n"
    pendingCalls--
    if (pendingCalls === 0) {
        systemInfo = info
    }
}

pendingCalls = methods.length
methods.forEach(function(m) {
```

```
            deviceInfoInterface.typedCall(m.name, [],
                function(result) { handleResult(m.desc, result); },
                function(error) { handleError(m.desc, error); }
            )
        })
    }

    function showCustomNotification() {
        var text = notificationText.trim()
        if (text === "") {
            text = "Это тестовое уведомление"
        }
        notification.summary = "Пользовательское уведомление"
        notification.body = text
        notification.publish()
        notificationText = ""
    }

    Column {
        anchors.fill: parent
        spacing: Theme.paddingLarge

        PageHeader { title: "Аутентификация и информация об устройстве"
}


        Column {
```

```
    width: parent.width
    spacing: Theme.paddingMedium
    visible: !authenticated

    PasswordField {
        id: passwordField
        width: parent.width
        placeholderText: "Введите пароль (0000)"
        inputMethodHints: Qt.ImhDigitsOnly
        maximumLength: 4
    }

    Button {
        text: "Войти"
        width: parent.width
        onClicked: authenticate()
    }
}

Column {
    width: parent.width
    spacing: Theme.paddingMedium
    visible: authenticated


    SectionHeader { text: "Информация об устройстве" }
```

```
Button {
    text: "Получить информацию об устройстве"
    width: parent.width
    onClicked: updateSystemInfo()
}

Label {
    width: parent.width
    wrapMode: Text.Wrap
    text: systemInfo
    visible: systemInfo !== ""
    color: Theme.highlightColor
    font.pixelSize: Theme.fontSizeSmall
}


SectionHeader { text: "Отправить своё уведомление" }

TextField {
    width: parent.width
    placeholderText: "Текст уведомления"
    text: notificationText
    onTextChanged: notificationText = text
}

Button {
    text: "Отправить уведомление"
```

```
            width: parent.width
            onClicked: showCustomNotification()
        }
    }
}
}
```

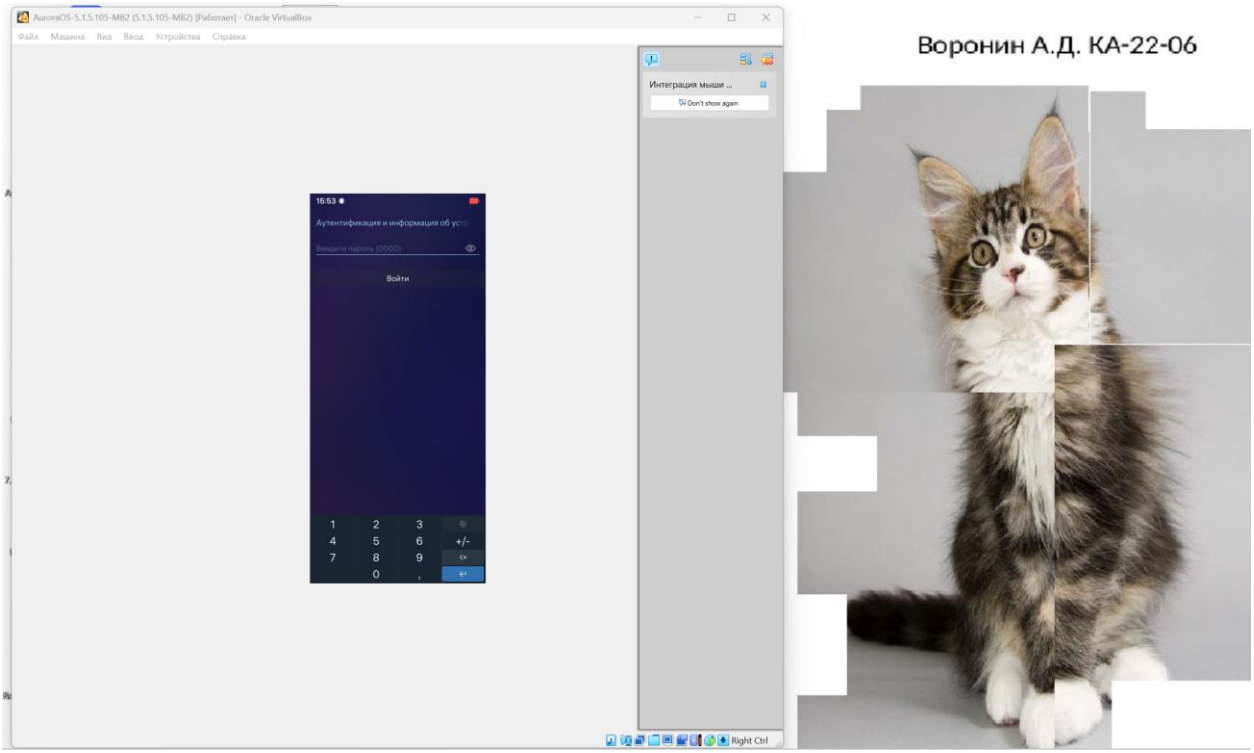

Рисунок 1 – Вход с паролем

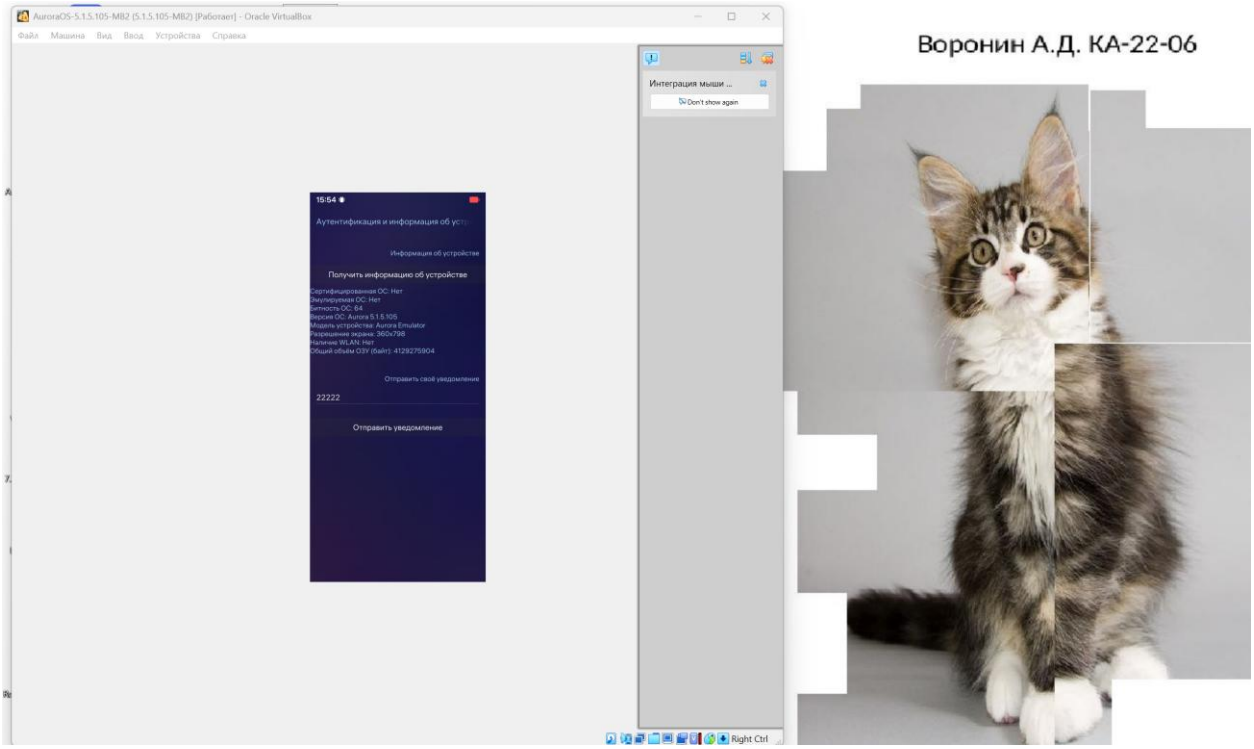

Рисунок 2 – Получение информации об устройстве

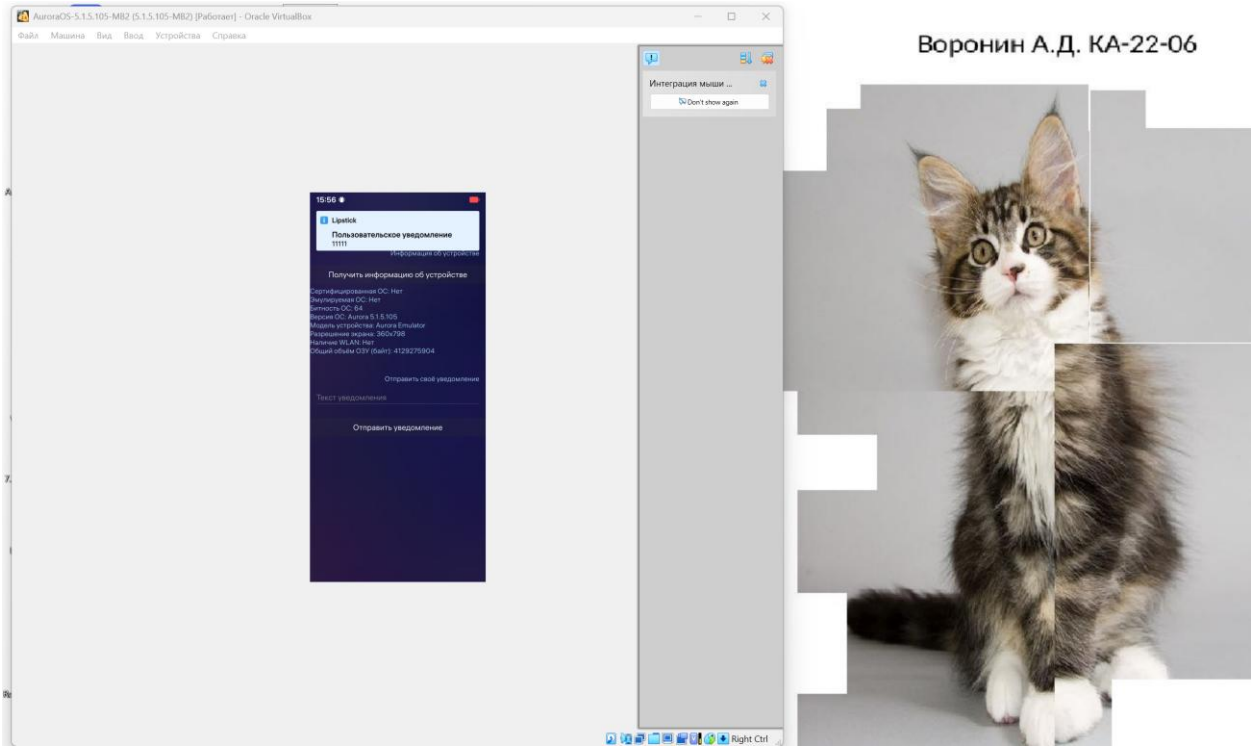

Рисунок 3 – Получение уведомления

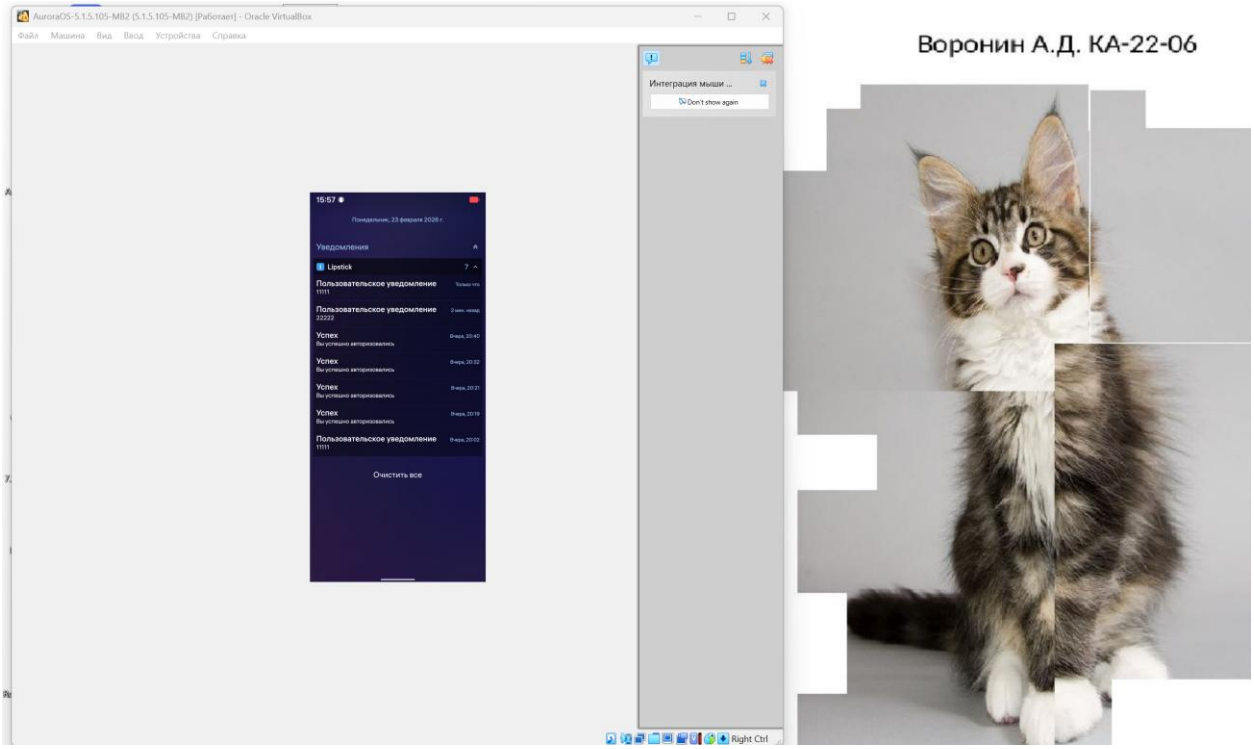

Рисунок 4 – Уведомления системы

ЗАКЛЮЧЕНИЕ

В заключение, подводя итог проделанной работе, все поставленные в начале цели и задания данной лабораторной работы были достигнуты в полном объеме.

Контрольные вопросы

1. Какие виды шин D-Bus существуют и чем они отличаются?

Существуют два вида шин:

- Системная
- Сессионная

Системная шина служит для общесистемных коммуникаций, сессионная - для связи между программами одного пользователя. Системная шина создаётся при старте D-Bus, сессионная - для каждого входа пользователя в систему.

2. Как организована безопасность в D-Bus?

- Изоляция системной и сессионной шин
- Политики доступа
- Аутентификация и авторизация

3. Какие типы данных поддерживаются в D-Bus?

Базовые

- String
- Boolean
- Int

И т.д.

Сложные

- ARRAY — массив из элементов одного типа.
- STRUCT — набор элементов фиксированного количества, каждый из любого типа.
- DICT — словарь

4. Как обрабатываются асинхронные вызовы?

В QML (плагин Nemo.DBus) асинхронность реализована через метод typedCall, который принимает две функции обратного вызова: callback для успешного завершения и errorCallback для обработки

ошибок. Эти функции вызываются автоматически, когда сервис возвращает результат, что позволяет обновлять интерфейс или обрабатывать данные по мере поступления ответов.