УДК *004.514:004:451*

Э. В. Тищенко,

ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация

В. Ю. Корякин *ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация*

Тищенко Эльдар Валерьевич – студент ФКБ ТЭК, ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва

Корякин Владислав Юрьевич – студент ФКБ ТЭК, ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва

ИССЛЕДОВАНИЕ ФУНКЦИОНАЛА ПО УПРАВЛЕНИЮ СЕТЬЮ ПОСРЕДСТВОМ ПОДСИСТЕМЫ D-BUS В ОС АЛЬТ»

Аннотация: *Данная статья посвящена исследованию возможностей управления сетевыми подключениями через универсальную системную шину D-bus с использованием ключевого сервиса NetworkManager. D-bus на данный момент является стандартом для межпроцессного взаимодействия, который позволяет обеспечивать простое и асинхронное взаимодействие между компонентами системы.*

В работе изучены теоретические основы D-Bus, архитектуры NetworkManager и их взаимодействия. Практическая часть представляет собой серию экспериментов, направленных на изучение архитектуры D-Bus, мониторинга сетевых событий, прямого управления соединениями и разработку автоматизированного python-агента.

Результаты исследования демонстрируют, что D-Bus представляет собой мощный, событийно-ориентированный механизм для управления и контроля над сетевыми настройками, что предполагает наличие широких возможностей для автоматизации системных процессов.

Ключевые слова: *D-Bus, сетевые настройки ОС Альт, интерфейсы, сигналы, NetworkManager, API.*

Abstract: *This article is devoted to the study of the possibilities of managing network connections via the universal D-bus system bus using the key service NetworkManager. D-bus is currently the standard for interprocess communication, which allows for simple and asynchronous interaction between system components.*

The paper examines the theoretical foundations of D-Bus, the architecture of NetworkManager and their interaction. The practical part is a series of experiments aimed at studying the D-Bus architecture, monitoring network events, direct connection management, and developing an automated python agent.

The results of the study demonstrate that D-Bus provides a powerful, event-oriented mechanism for managing and controlling network settings, which implies extensive opportunities for automating system processes.

Keywords: *D-Bus, Alt OS network settings, interfaces, signals, NetworkManager, API.*

Введение и теоретические основы

Сетевая подсистема является фундаментальным компонентом любой современной операционной системы. Хотя методы управления по средствам вызова утилит командной строки надежны, они не позволяют достичь нужной гибкости и оперативности для построения сложных систем автоматизации. В операционных системах семейства Linux, включая ОС Альт, де-факто стандартом для межпроцессного взаимодействия (IPC) является шина сообщений D-Bus. Этот механизм обеспечивает универсальную и масштабируемую связь между системными службами и приложениями [1, с. 325].

В современных дистрибутивах Linux, включая basealt, управление сетью осуществляется преимущественно с помощью демона NetworkManager. Он был выбран для исследования, поскольку стал де-факто стандартом и полностью основан на взаимодействии с шиной D-Bus. NetworkManager обеспечивает автоматическое обнаружение сетевых устройств, унифицированное управление проводными и беспроводными подключениями, а его интеграция с D-Bus делает его идеальным инструментом для изучения программного управления сетью [2]. Выбирать etcnet, к примеру, нельзя, т.к. etcnet является способом управления сетью через статические файлы, а D-Bus используется для динамического управления интерфейсами и события через API, а не через прямое редактирование конфигурационных файлов [4]. Далее в эксперименте №1 будет проведен сравнительный анализ производительности двух автоматизированных методов для подтверждения тезиса. Изучение принципов работы D-Bus API NetworkManager даст понимание того, как реализовывать программное управление сетью, способное реагировать на изменения в системе, а не просто выполнять предопределенные команды [6].

Цель данной статьи – исследование и демонстрация на практике возможности управления сетью в ОС Альт через D-Bus API сервиса NetworkManager.

Подсистема межпроцессного взаимодействия D-Bus

D-Bus (Desktop Bus) — это система межпроцессного взаимодействия (IPC), являющаяся системной шиной для обмена сообщениями между приложениями. Работа шины зависит от следующих принципов [7]:

- шина: центральный демон, который отвечает за пересылку сообщений между процессами. Существует 2 шины – системная, которая отвечает за коммуникацию системных серверов, и шина сеанса, с помощью которой связываются между собой пользовательские приложения;

- сервисы: сервисами называют приложения, которые регистрируют свое имя при подключении к шине для обеспечения доступа других приложений к ним [5, с. 203];
- объекты и пути: сервисы предоставляют свой функционал и доступ к своим данных по средствам объектов, адресация к которым осуществляется благодаря иерархическим путям;
- интерфейсы: интерфейсы логически группируют методы и сигналы, формируются объектами;
- методы: методами являются функции, которые вызываются клиентами для выполнения какого-либо действия;
- сигналы: сообщения, которые сервис рассылает всем подписчикам для уведомления о событиях [3].

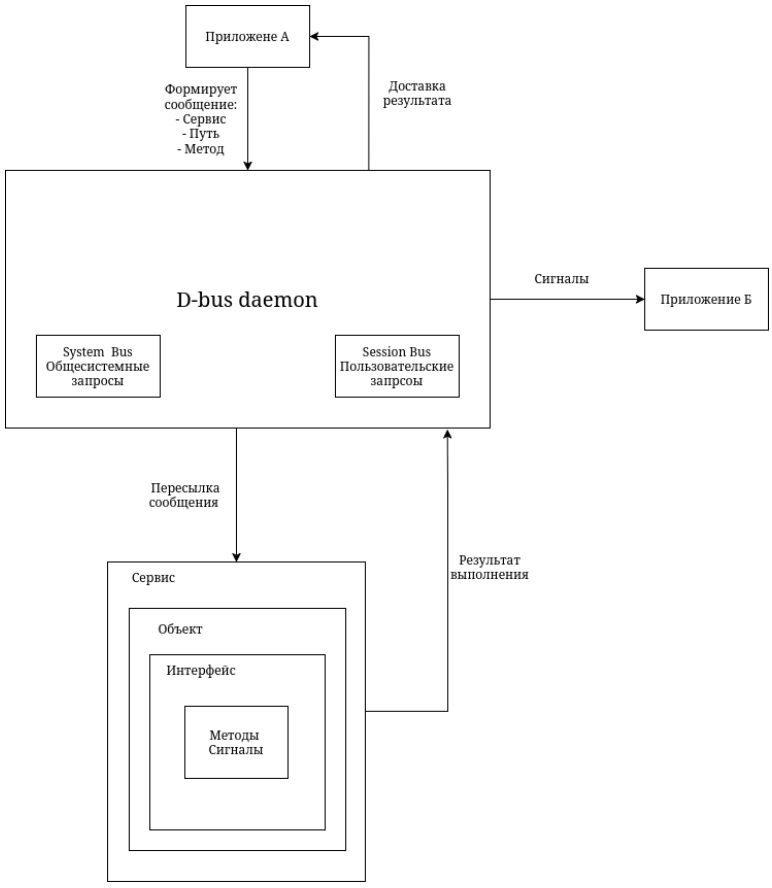

Рис. 1. Структурная схема IPC

Сервис NetworkManager и его интеграция с D-Bus

NetworkManager – это системный демон, его основной задачей является стандартизация и упрощение процессов настройки и поддержания сетевых соединений. Он автоматически обнаруживает сетевые устройства, управляет профилями подключений и выбирает оптимальное соединение [2].

Полностью раскрыть свой потенциал NetworkManager может через D-Bus. Он регистрируется на системной шине под именем org.freedesktop.NetworkManager и предоставляет набор объектов для управления. Интерфейс D-Bus предоставляет возможность:

- получать список сетевых интерфейсов и их состояние;
- создавать, изменять и удалять настройки/конфигурации подключения к сети;
- активировать и деактивировать соединения;
- подписываться на сигналы об изменении состояния сети, устройствах или точках доступа Wi-Fi.

Иными словами, D-Bus – это не способ обмена информацией, а фундаментальный слой, на котором построена вся современная архитектура управления сетью в ОС Альт.

Методология исследования

Объект исследования: подсистема управления сетью операционной системы Альт.

Предмет исследования: функциональные возможности по управлению сетью, предоставляемые через D-Bus API сервиса NetworkManager.

Среда проведения: исследование проводилось на операционной системе basealt. В качестве основных инструментов для взаимодействия с D-Bus использовались утилиты командной строки gdbus и busctl, а также язык программирования Python с библиотекой python-dbus для разработки программного агента.

Экспериментальная часть была спланирована таким образом, чтобы последовательно исследовать все аспекты взаимодействия: от изучения структуры API до создания автоматизированного решения. Общий план экспериментов представлен в таблице 1.

Таблица 1 – Информация о проведенных экспериментах

№	Цель эксперимента	Используемые инструменты	Ожидаемый результат
1	Автоматическое переключение на резервный канал	NetworkManager, nmcli, D-Bus API, python	Сравнение двух методов переключения, которые реализованы посредством кода
2	Динамическое изменение контекста безопасности	Nmcli, python, D-Bus API, iptables	Python-агент, меняющий политики безопасности firewall в зависимости от подключения

Эксперимент 1. Автоматическое переключение на резервный канал

Цель данного эксперимента заключается в том, чтобы исследовать и сравнить эффективность двух разных методов реализации: Polling-метода (опросный) и событийно-ориентированного метода (D-Bus) на базе NetworkManager.

Polling-метод.

В основе данного метода лежит периодический опрос состояния сетевого интерфейса посредством утилиты nmcli. Для реализации был написан Bash-скрипт, который через заданный интервал проверяет состояние основного

интерфейса и при обнаружении сбоя/отказа инициирует переключение на резервное подключение. Ниже приведен листинг скрипта:

```
#!/bin/bash

CHECK_INTERVAL=1

MAIN_DEV="enp0s3"
MAIN_CON="main-nat"
BACKUP_CON="second-nat"

echo "Запуск мониторинга (Polling). Интервал: ${CHECK_INTERVAL}c"
echo "Нажмите Ctrl+C для остановки"

while true; do
   STATE=$(nmcli -g GENERAL.STATE device show $MAIN_DEV | cut -c 1)
   #echo $STATE

   if [ "$STATE" == '3' ]; then
      echo "[FAIL] Обнаружен отказ $MAIN_DEV в $(date +%H:%M:%S.%N)"
      echo "Переключение на $BACKUP_CON..."

      # Замер времени начала переключения
      start_time=$(date +%s%N)

      nmcli connection up "$BACKUP_CON"

      end_time=$(date +%s%N)
      echo "Переключение завершено за $(( ($end_time - $start_time) / 1000000 )) мс"
```

```
        # Выходим, чтобы не зациклиться, если второй тоже упадет
        break
    fi

    sleep $CHECK_INTERVAL
done
```

Для тестирования необходимо выключить резервный интерфейс. После запускаем скрипт ./polling_test.sh и отключаем основное устройство при выключенном втором nmcli dev disconnect enpos3. Ниже представлен листинг работы скрипта при обнаружении неисправности работы основного интерфейса:

```
[root@rtr test_area]# ./polling_test.sh
Запуск мониторинга (Polling). Интервал: 1c
Нажмите Ctrl+C для остановки
[FAIL] Обнаружен отказ enp0s3 в 16:42:30.635068158
Переключение на second-nat...
Подключение успешно активировано (активный путь D-Bus: /org/freedesktop/NetworkManager/ActiveConnection/86)
Переключение завершено за 716 мс
```

Из листинга выше видно, что процесс переключения на резервное подключение занял 716 мс.

Событийно-ориентированный метод.

В данном методе используется прямое взаимодействие с NetworkManager через D-Bus. Для проверки работоспособности был написан Python-скрипт, который подписывается на сигнал PropertiesChanged сетевого устройства и реагирует на изменение его состояния. При получении сигнала о сбое/отключении основного устройства, python-агент немедленно вызывает метод ActivateConnection посредством D-Bus, тем самым активируя резервное подклчение. Листинг программы представлен ниже:

```
import dbus
```

```
import dbus.mainloop.glib
from gi.repository import GLib
import time

MAIN_IFACE = "enp0s3"       # Интерфейс, который мониторим
BACKUP_CON_NAME = "second-nat" # Имя подключения для переключения
BACKUP_IFACE = "enp0s8"     # Физический интерфейс для бэкапа

NM_DEVICE_STATE_ACTIVATED = 100
NM_DEVICE_STATE_DISCONNECTED = 30
NM_DEVICE_STATE_FAILED = 120

def get_device_path(interface_name):
    """Находит объектный путь устройства по имени (например, enp0s3)"""
    bus = dbus.SystemBus()
    nm = bus.get_object('org.freedesktop.NetworkManager',
'/org/freedesktop/NetworkManager')
    devices = nm.GetDevices(dbus_interface='org.freedesktop.NetworkManager')

    for dev_path in devices:
        dev_obj = bus.get_object('org.freedesktop.NetworkManager', dev_path)
        iface = dev_obj.Get('org.freedesktop.NetworkManager.Device', 'Interface',
                            dbus_interface='org.freedesktop.DBus.Properties')
        if iface == interface_name:
            return dev_path
    return None

def get_connection_path(connection_name):
    """
    Находит объектный путь подключения (connection path) по его имени.
```

```
    Это обязательно, так как D-Bus работает с путями, а не с именами.
    """
    bus = dbus.SystemBus()
    settings_proxy = bus.get_object('org.freedesktop.NetworkManager',
                                    '/org/freedesktop/NetworkManager/Settings')
    settings_iface = dbus.Interface(settings_proxy,
'org.freedesktop.NetworkManager.Settings')

    connections = settings_iface.ListConnections()

    for conn_path in connections:
        conn_proxy = bus.get_object('org.freedesktop.NetworkManager', conn_path)
        conn_settings = dbus.Interface(conn_proxy,
'org.freedesktop.NetworkManager.Settings.Connection')



        config = conn_settings.GetSettings()

        # config['connection']['id'] - это то самое имя, которое мы видим в nmcli
        if config['connection']['id'] == connection_name:
            return conn_path

    print(f"[ОШИБКА] Подключение '{connection_name}' не найдено в
настройках!")
    return None

def properties_changed(interface_name, changed_properties,
invalidated_properties):
    """Обработчик сигнала об изменении свойств основного устройства"""
    if 'State' in changed_properties:
```

```
        new_state = changed_properties['State']

        if new_state == NM_DEVICE_STATE_DISCONNECTED or new_state ==
NM_DEVICE_STATE_FAILED:
            print(f"[SIGNAL] Фиксация отказа основного канала (State:
{new_state}) в {time.time()}")

            start_time = time.time()

            try:

                nm_proxy = bus.get_object('org.freedesktop.NetworkManager',
'/org/freedesktop/NetworkManager')
                nm_iface = dbus.Interface(nm_proxy,
'org.freedesktop.NetworkManager')

                print(f"-> Активация {BACKUP_CON_NAME} на
{BACKUP_IFACE}...")

                nm_iface.ActivateConnection(
                    backup_conn_path,
                    backup_dev_path,
                    "/"
                )

                end_time = time.time()
                elapsed_ms = int((end_time - start_time) * 1000)

                print(f"[УСПЕХ] Переключение выполнено за {elapsed_ms} мс (Pure
D-Bus)")
```

```
            except dbus.exceptions.DBusException as e:
                print(f"[ОШИБКА] Не удалось переключиться: {e}")



            loop.quit()

if __name__ == "__main__":
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    bus = dbus.SystemBus()

    print("--- Инициализация Pure D-Bus Failover ---")

    backup_dev_path = get_device_path(BACKUP_IFACE)
    if not backup_dev_path:
        exit(1)
    print(f"Устройство бэкапа найдено: {backup_dev_path}")

    backup_conn_path = get_connection_path(BACKUP_CON_NAME)
    if not backup_conn_path:
        exit(1)
    print(f"Подключение найдено: {backup_conn_path}")

    main_dev_path = get_device_path(MAIN_IFACE)
    if not main_dev_path:
        exit(1)
    print(f"Мониторинг устройства: {main_dev_path}")
    print("Ожидание событий...\n")

    bus.add_signal_receiver(
```

```
    properties_changed,
    bus_name='org.freedesktop.NetworkManager',
    path=main_dev_path,
    dbus_interface='org.freedesktop.DBus.Properties',
    signal_name='PropertiesChanged'
)

loop = GLib.MainLoop()
loop.run()
```

Далее необходимо предварительно выключить резервный интерфейс, запустить выполнение скрипта и эмулировать сбой основного интерфейса. В результате получаем такой вывод скрипта:

```
[SIGNAL] Фиксация отказа основного канала (State: 30) в 1770039928.9765048
-> Активация second-nat на enp0s8...
[УСПЕХ] Переключение выполнено за 29 мс (Pure D-Bus)
```

Из вывода программы можно заметить скорость выполнения переключения на резервное подключение – 29 мс, что значительно превосходит по скорости реакции традиционный Polling подход. Использование D-Bus не создаёт дополнительной нагрузки на CPU, поскольку мониторинг состояния сетевого интерфейса осуществляется событийно – за счёт обработки сигналов NetworkManager, а не посредством постоянного опроса состояния системы. Сравнительный анализ представлен в виде таблицы ниже:

Таблица 2 – сравнительный анализ подходов.

Замеры	Время, мс	Процессор, %	Память, %
Polling	741.6	0.0-0.7	0.2
D-Bus	22.8	0.0	1.0

Таким образом, можно сделать вывод о том, что использование D-Bus в подобных вопросах является более выгодным решением с точки зрения нагрузки на систему.

Эксперимент 2. Динамическое изменение контекста безопасности.

Целью данного эксперимента является исследование возможности динамического изменения сетевой безопасности на основе получаемых сигналов от NetworkManager.

Суть эксперимента лежит в разработке микросервиса, который ловит сигнал переключения подключения и на его основе меняет настройки firewall.

Предварительно было создано новое подключение:

nmcli con add type ethernet con-name local-private-net ifname enp0s9 ip4 77.77.0.121/24

Python-агент подписывается на сигнал StateChanged от NetworkManager. При активации подключения, оно идентифицируется, после чего применяется соответствующая политика безопасности. Для приватно сети SSH разрешен, а для обычной – SSH блокируется. Далее представлен листинг агента:

```
import dbus
import dbus.mainloop.glib
from gi.repository import GLib
import subprocess

TRUSTED_UUID = "96e6f92e-8dc7-4b5f-b89e-c12748e84c8f"
UNTRUSTED_UUID = "6761581e-4c8e-49ec-a7aa-a9d98d115ca7"

BLOCK_RULE = "INPUT -p tcp --dport 22 -j DROP"

def get_uuid_by_device_path(dev_path):
    """Получает UUID активного подключения по пути устройства"""
    try:
        bus = dbus.SystemBus()
        dev_proxy = bus.get_object('org.freedesktop.NetworkManager', dev_path)
```

```
        active_conn_path                                                    =
dev_proxy.Get('org.freedesktop.NetworkManager.Device', 'ActiveConnection',
                                dbus_interface='org.freedesktop.DBus.Properties')

        if not active_conn_path or active_conn_path == "/":
            return None

        ac_proxy          =          bus.get_object('org.freedesktop.NetworkManager',
active_conn_path)
        uuid  =  ac_proxy.Get('org.freedesktop.NetworkManager.Connection.Active',
'Uuid',
                          dbus_interface='org.freedesktop.DBus.Properties')
        return uuid
    except Exception as e:
        print(f"Ошибка при получении UUID: {e}")
        return None

def apply_security_policy(uuid):
    """Логика безопасности"""
    if not uuid:
        return

    print(f"[*] Анализ безопасности для UUID: {uuid}")

    if uuid == UNTRUSTED_UUID:
        print(f"[ALERT] Недоверенная сеть. Блокируем SSH...")
        try:
            check  =  subprocess.run(["iptables",  "-C",  *BLOCK_RULE.split()],
capture_output=True)
```

```
            if check.returncode != 0:
                subprocess.run(["iptables", "-A", *BLOCK_RULE.split(), "-m",
"comment", "--comment", "DBUS_SEC_AGENT"], check=True)
                print("   -> Правило добавлено (SSH закрыт).")
            else:
                print("   -> Правило уже активно.")
        except subprocess.CalledProcessError:
            pass # Ошибки игнорируем (правило уже есть)

    elif uuid == TRUSTED_UUID:
        print(f"[INFO] Доверенная сеть. Открываем SSH...")
        try:
            while True:
                res = subprocess.run(["iptables", "-D", *BLOCK_RULE.split(), "-m",
"comment", "--comment", "DBUS_SEC_AGENT"],
                                     capture_output=True)
                if res.returncode != 0:
                    break
            print("   -> Правило удалено (SSH открыт).")
        except Exception:
            pass

def device_state_changed(new_state, old_state, reason, sender_path):
    """
    Аргументы сигнала StateChanged NM:
    new_state (uint32)
    old_state (uint32)
    reason (uint32)
    sender_path (str) - это мы просим добавить через path_keyword
    """
```

```
    NM_DEVICE_STATE_ACTIVATED = 100

    if new_state == NM_DEVICE_STATE_ACTIVATED:
        print(f"\n[EVENT] Интерфейс {sender_path} стал АКТИВНЫМ")
        uuid = get_uuid_by_device_path(sender_path)
        if uuid:
            apply_security_policy(uuid)
        else:
            print("Не удалось определить UUID (возможно, интерфейс без IP).")

if __name__ == "__main__":
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    bus = dbus.SystemBus()

    print("--- Агент безопасности запущен ---")
    print(f"Trusted: {TRUSTED_UUID}")
    print(f"Untrusted: {UNTRUSTED_UUID}")



    bus.add_signal_receiver(
        device_state_changed,
        bus_name='org.freedesktop.NetworkManager',
        dbus_interface='org.freedesktop.NetworkManager.Device',
        signal_name='StateChanged',
        path_keyword='sender_path'
    )

    loop = GLib.MainLoop()
    loop.run()
```

Для проверки работы, в одном терминале запускается скрипт, а в другом меняются подключения через nmcli con up.

При проверке политик для разных подключений можно заметить корректность работы агента:

```
# приватное подключение
INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh /*
DBUS_SEC_AGENT */

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

#обычное подключение
[root@rtr test_area]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Данный эксперимент показывает, что с помощью D-Bus можно создавать сложные системы для гибкого обеспечения безопасности сети.

Заключение

В ходе статьи было подробно изучено управление сетью в basealt с помощью NetworkManager и шины D-Bus. При проведении экспериментов были наглядно показаны возможности взаимодействия с сетью через API, цель была достигнута.

Было выяснено, что D-Bus – это не просто набор команд, а целая система управления сетевой инфраструктурой. Эксперименты подтвердили возможность построения систем, способных в реальном времени реагировать на изменения состояния сети. Благодаря D-Bus API появляется возможность создавать более надежные, гибкие и быстрые решения, которые буду интегрированы в ядро ОС.

С практической точки зрения ценность данного исследования в демонстрации силы D-Bus как инструмента автоматизации.

Список литературы

1. Шевченко Д.А., Боготов И.С. Применение D-Bus для управления KDE в ОС АЛЬТ // Вестник Науки. - 2025. - №6. - С. 324-333.
2. Part III. D-Bus API Reference // NetworkManager Developer Documentation URL: https://networkmanager.dev/docs/api/latest/spec.html (дата обращения: 06.12.2025).
3. Документация freedesktop.org // dbus URL: https://www.freedesktop.org/wiki/Software/dbus/ (дата обращения: 01.02.2026).
4. НОРМИРОВАНИЕ ПОКАЗАТЕЛЕЙ ПРИ ОРГАНИЗАЦИИ КИБЕРПОЛИГОНА ПО СЕТЕВЫМ ТЕХНОЛОГИЯМ Уймин А.Г. В книге: Нефть и газ - 2024. Тезисы докладов 78-ой Международной молодежной научной конференции. Москва, 2024. С. 1253-1254.
5. В.Г. Резник ОПЕРАЦИОННЫЕ СИСТЕМЫ. ЧАСТЬ 2: дис. доцент инф. наук: 09.03.01. - 2016, 216 с.
6. Дэвид Уиллер, Джон Палмиери, Колин Уолтерс Вводное руководство D-Bus // 2020 URL: https://dbus.freedesktop.org/doc/dbus-tutorial.html (дата обращения: 02.02.2026).

7. Потапов А.А.,Чулисов Е.В. Работа сетевой подсистемы в отечественных ОС // Мировая наука. - 2025. - №1(94).

References

1. Shevchenko, D. A., & Bogotov, I. S. (2025). Using D-Bus to control KDE in ALT OS. Vestnik Nauki, 324-333.
2. (2018). Part III. D-Bus API Reference. NetworkManager Developer Documentation. Retrieved June 12, 2025, from https://networkmanager.dev/docs/api/latest/spec.html
3. (2026). freedesktop.org documentation. D-Bus. Retrieved February 01, 2026, from https://www.freedesktop.org/wiki/Software/dbus/
4. Uimin, A. G. (2024). NORMALIZATION OF INDICATORS IN THE ORGANIZATION OF A CYBER POLYGON ON NETWORK TECHNOLOGIES. Oil and gas - 2024. Abstracts of the 78th International Youth Scientific Conference, 1253-1254.
5. Reznik, V.G. (2016). OPERATING SYSTEMS. PART 2. TOMSK STATE UNIVERSITY OF MANAGEMENT SYSTEMS AND RADIO ELECTRONICS.
6. Wheeler, D., Palmieri, J., & Walters, C. (2020). D-Bus Tutorial. Retrieved February 02, 2026, from https://dbus.freedesktop.org/doc/dbus-tutorial.html
7. Potapov, A. A., & Chulisov, E. V. (2025). The work of the network subsystem in domestic operating systems. Mirovaya nauka, 1(94). https://doi.org/10.5281/zenodo.14889548