РОССИЙСКАЯ ФЕДЕРАЦИЯ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и компьютерных наук

Кафедра информационной безопасности

КУРСОВАЯ РАБОТА

по курсу: «Администрирование операционных систем»

на тему: «Мониторинг сетевых устройств при помощи SNMP протокола»

Выполнили:

Студенты группы ИБАС-168-1

Петров Роман Владимирович

Урбан Назар Юрьевич

Проверил:

Зав. Кафедрой информационной безапосности

Оленников Евгений Алекснадрович

Оглавление

[ВВЕДЕНИЕ 3](#_Toc533598268)

[ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 3](#_Toc533598269)

[1.1. АРХИТЕКТУРА SNMP 3](#_Toc533598270)

[1.2 АРХИТЕКТУРА PYSNMP 5](#_Toc533598271)

[ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 6](#_Toc533598272)

[2.1 АРХИТЕКТУРА ПРИЛОЖЕНИЯ 6](#_Toc533598273)

[2.2. АЛГОРИТМЫ МОНИТОРИНГА 7](#_Toc533598274)

[2.3. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС 10](#_Toc533598275)

[Глава 3. Пользовательский интерфейс 11](#_Toc533598276)

[ЗАКЛЮЧЕНИЕ 12](#_Toc533598277)

[ПРИЛОЖЕНИЕ 13](#_Toc533598278)

[СПИСОК ЛИТЕРАТУРЫ 14](#_Toc533598279)

# ВВЕДЕНИЕ

Существует множество готовых приложений для мониторинга хостов в сети, которые используют различные протоколы и методы получения информации от удаленного хоста. Цель данной курсовой работы — разработка приложения для мониторинга устройств в сети с помощью протокола SNMP в языковой среде Python 3 на базе фреймворка Qt.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. изучить архитектуру SNMP, документации Pysnmp;
2. реализовать алгоритмы сетевого мониторинга на языке Python;
3. разработать графическую оболочку на основе фреймворка Qt;
4. связать графическую оболочку и алгоритмы мониторинга.

Актуальность методов, используемых в программе, обусловлена тем, что они могут быть применимы для решения практических задач в области сетевого администрирования, разработки новых и совершенствования существующих SNMP агентов.

В главе 1 представлено теоретическое описание реализации приложения.

В главе 2  описана архитектура приложения.

В главе 3 содержится перечисление и описание доступных для пользователя действий.

# ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Существует множество методов, с помощью которых можно реализовать мониторинг удаленных устройств.

Для реализации алгоритмов мониторинга был выбран протокол SNMP. Данный протокол повсеместно распространен: его поддерживают все сетевые устройства: маршрутизаторы, рабочие станции, принтеры, сервера и так далее. Также SNMP может использоваться не только для мониторинга, но и для настройки сетевых устройств.

Для работы с протоколом в SNMP в языковой среде Python 3 реализовано несколько библиотек, наиболее популярные: netsnmp, pysnmp, libsnmp. Pysnmp был выбран для разработки приложения, так как в netsnmp реализован интерфейс всего для нескольких команд SNMP, а libsnmp заброшен разработчиками и не поддерживается. К преимуществам pysnmp относится поддержка всех трех версий протокола SNMP, возможность реализации синхронных и асинхронных интерфейсов и тот факт, что библиотека написана на чистом Python, что упрощает мультиплатформенность. К тому же pysnmp имеет достаточно удобный в обращении инструментарий разработчика.

## 1.1 АРХИТЕКТУРА SNMP

SNMP (Simple Network Management Protocol) — стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP. Уникальность протокола обусловлена тем, что он является повсеместно распространенным: его поддерживают все устройства, обладающие сетевыми картами: маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и т.д. Протокол позволяет получать и изменять сетевую информацию устройства, поэтому обычно используется в различном программном и аппаратном обеспечении для настройки и мониторинга сети. Он состоит из набора стандартов для сетевого управления, включая протокол прикладного уровня, схему баз данных и набор объектов данных.

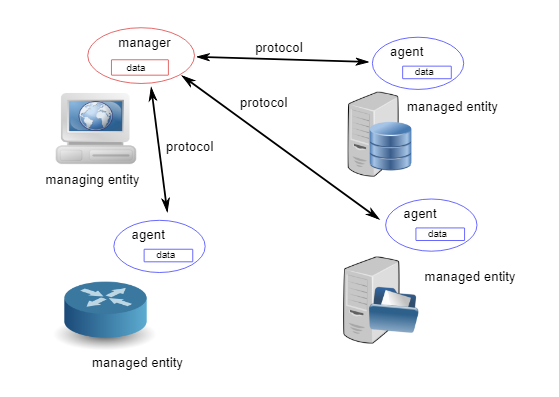


Рисунок 1. Графическая схема работы SNMP протокола

При использовании SNMP один или более административных компьютеров (где функционируют программные средства, называемые менеджерами) выполняют отслеживание и/или управление группой хостов или устройств в компьютерной сети. (см рис. 1) На каждой управляемой системе есть постоянно запущенная программа, называемая агент, которая через SNMP передаёт информацию менеджеру.

SNMP использует информацию о сетевом устройстве, хранящуюся в цифровом формате. Для упрощения применяются базы управляющей информации (MIB). Базы MIB описывают структуру управляемых данных на подсистеме устройства; они используют иерархическое пространство имён, содержащее идентификаторы объектов (OID-ы). Каждый OID состоит из двух частей: текстового имени и SNMP адреса в цифровом виде. Базы MIB являются необязательными и выполняют вспомогательную роль по переводу имени объекта, что значительно упрощает практическое применение протокола SNMP. При этом MIB таблицы различных производителей не совпадают, что приводит к необходимости определять вендора для корректной работы SNMP протокола.

**1.2 АРХИТЕКТУРА PYSNMP**

Pysnmp - библиотека Python, представляющая собой ядро SNMP для всех трех версий и набор стандартных приложений, написанных на чистом Python. Поддерживает роли менеджера/агента/прокси сети, работу с MIB на стороне менеджера/агента, асинхронную работу и несколько сетевых каналов.

Внутренние компоненты pysnmp:

* Ядро SNMP — центральный компонент, реализующий управление другими компонентами системы SNMP.
* Транспортная подсистема используется для отправления и принятия SNMP пакетов в сети.
* Диспетчер сообщений и PDU — определяет алгоритм действий по обработке сообщений SNMP. В его основные обязанности входит отправка Protocol Data Unit (PDU) из приложений SNMP через различные подсистемы вплоть до транспортного диспетчера, а также передача сообщений SNMP из сети в приложения SNMP.
* Модули обработки сообщений — обрабатывают операции уровня сообщений для настоящих и, возможно, будущих версий протокола SNMP. Наиболее важно, что они включают в себя разбор/сборку сообщений и, возможно, вызов служб безопасности при необходимости.
* Модули безопасности сообщений — выполняют аутентификацию и/или шифрование сообщений. Все модули безопасности имеют общий API-интерфейс, используемый подсистемой обработки сообщений.
* Подсистема контроля доступа — использует информацию «Local Configuration Datastore» (LCD) для авторизации удаленного доступа к управляемым объектам. Это используется при запуске в роли агента.
* Коллекция модулей и объектов MIB, которые используются ядром SNMP для хранения его конфигурации и рабочей статистики. Все вместе они называются LCD.

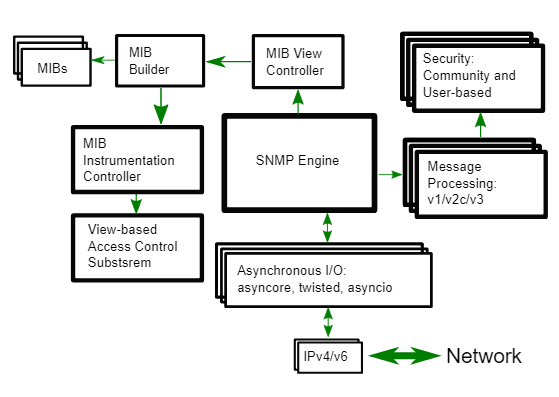


Рисунок 2. Схема внутреннего устройства pysnmp

Все вместе описанные выше компоненты (см. рис. 1) составляют библиотеку pysnmp, предоставляющую удобный для разработчика инструментарий, необходимый для работы с протоколом SNMP и позволяющий реализовать любые функции актуальной версии протокола.

**ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Разработку данного приложения можно разделить на две части: алгоритмы мониторинга и графический интерфейс. Соединение с устройством и запросы реализованы в модуле «main» в классе «Connection». Графический интерфейс состоит из нескольких окон, каждое из которых реализовано в своем классе.

**2.1 АРХИТЕКТУРА ПРИЛОЖЕНИЯ**

Графический интерфейс с помощью второго потока регулярно посылает запросы через SNMP агент. В результате, главный поток, в котором находится отрисовщик, остается свободен при запросе информации с удаленного хоста.

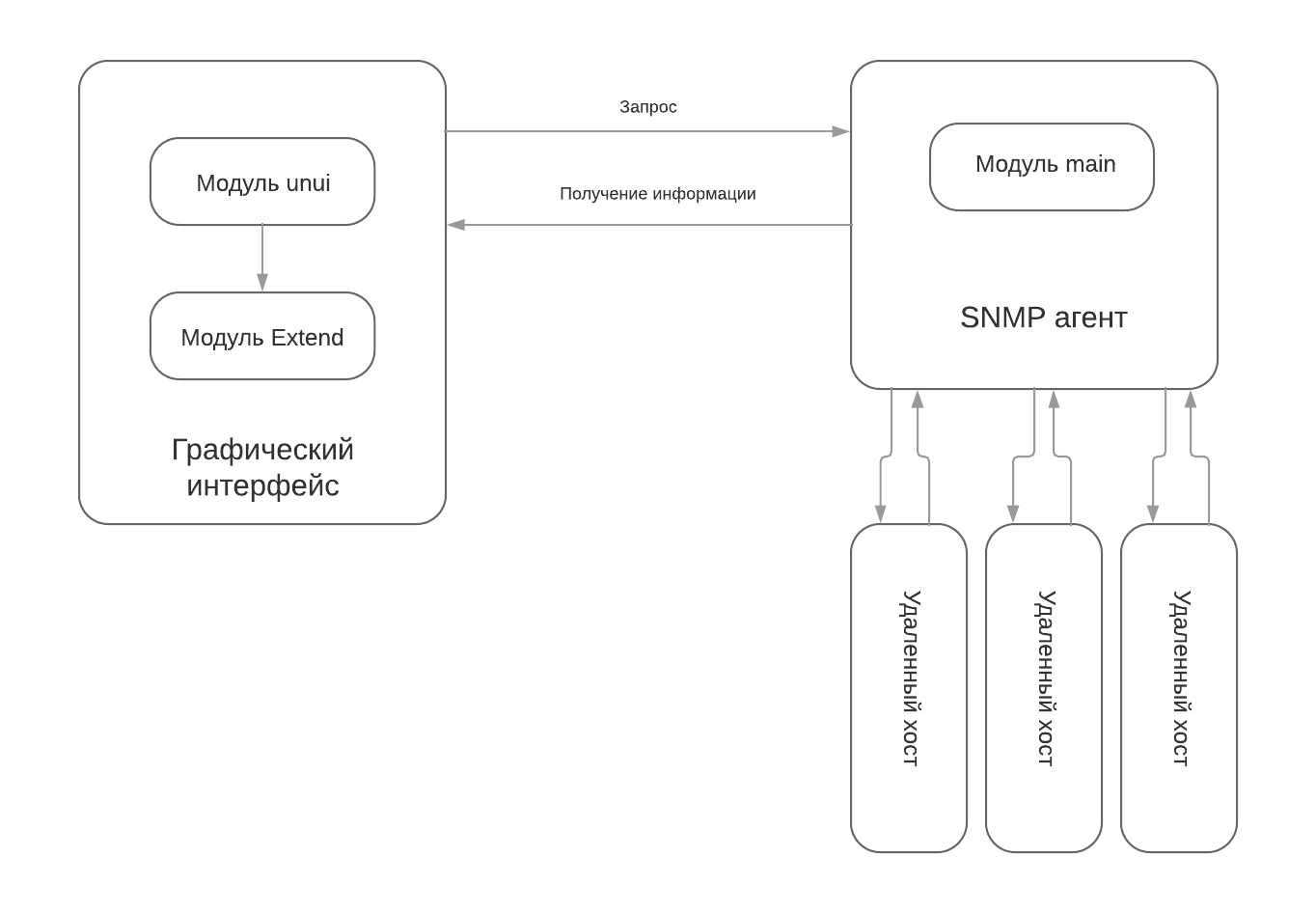


Рисунок 3. Архитектура приложения

2.2. АЛГОРИТМЫ МОНИТОРИНГА

Основным компонентом, выполняющим все необходимые для реализации мониторинга удаленных устройств SNMP запросы, выступает класс «Connection».

Класс «Connection» реализует подключение к удаленному SNMP агенту и с использованием библиотеки pysnmp отправляет и получает все необходимые для вывода графическому интерфейсу данные. Для мониторинга конкретных устройств приложению необходимо создать соответствующие подключения. Для этого при создании нового подключения из графического интерфейса определяется соответствующий объект класса «Connection», в переменные которого записываются необходимые ip-адрес и snmp-группа (далее «community»). На основе этих данных выполняются все дальнейшие методы.

Для получения любой информации посредством SNMP используются методы pysnmp «getcmd» и «nextcmd». Обе команды при вызове выполняют SNMP-запрос по указанным значениям ip, community и порту. Также методы getcmd и nextcmd возвращают генераторы python, содержащие четыре переменных: errorIndication, errorStatus, errorIndex, varBinds. Переменные ErrorIndication, errorStatus, errorIndex содержат информацию о ошибке, если таковая произошла при выполнении SNMP-запроса, в противном случае они равны none. varBinds же содержат объект SNMP MIB, в котором содержится вся информация о соответствующем MIB. Отличие стандартного pysnmp метода nextcmd заключается в том, что он берет не указанный MIB, а следующий за ним в дереве MIB. Также python генераторы, полученные после выполнения этих методов можно использовать для получения следующих в цепочке OID значений при помощи передачи генератора стандартному python методу «next» (см листинг 2).

|  |
| --- |
| def get\_oid(self, command):    return (getCmd(SnmpEngine(),                   CommunityData(self.community),                   UdpTransportTarget((self.ipaddr, 161)),                   ContextData(),                   ObjectType(command))) |

Листинг 1. Метод «get\_oid» класса «Connection»

В приложении стандартные методы pysnmp интегрированы в методы класса «Connection» для удобства использования. Метод «get\_oid» класса «Connection» (см. листинг 1) реализует «getcmd» с использованием переменных класса. Для вызова стандартных pysnmp методов «getcmd» и «nextcmd» необходимо указать целевой ip-адрес, community, OID/MIB, и порт. Первые два параметра находятся в переменных класса, определенных при создании объект «Connection». UDP порт для выполнения get и set SNMP запросов всегда один и равен 161. Для использования SNMP-ловушек применяется порт UPD 162.

Вместо стандартного pysnmp метода «getnext» в классе «Connection» реализован метод «get\_next\_oid» (см. листинг 2), который берет следующую итерацию полученного из «get\_oid» генератора. Эти действия аналогичны «nextcmd», но позволяют получать информацию по табличному MIB объекта вместо его цифрового представления (OID).

|  |
| --- |
| def get\_next\_oid(self, OID): #main cmd, берет next oid из MIB    errorIndication, errorStatus, errorIndex, varBinds = next(self.get\_oid(OID))    for name, val in varBinds:        return (val.prettyPrint()) |

Листинг 2. Метод «get\_next\_oid» класса «Connection»

Все остальные методы, непосредственно возвращающие необходимую информации оформлены отдельно в том же классе и используют описанные ранее методы «get\_oid» и «get\_next\_oid», передавая им необходимое значение OID или MIB-описание.

Метод «get\_systeminfo» позволяет получить краткую системную информацию устройства, в которую обычно входит название операционной системы, hostname, производитель, разрядность и прочее (см. Листинг 3). В его структуру входит описанный ранее метод «get\_next\_oid», на вход которого подается MIB «SNMPv2-MIB::sysDescr.0». Для применения MIB-таблицы используется встроенная в pysnmp библиотеку таблица, применимая для большинства сетевых устройств.

|  |
| --- |
| def get\_systeminfo(self):    return self.get\_next\_oid(ObjectIdentity('SNMPv2-MIB','sysDescr',0).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')) |

Листинг 3. Метод «get\_systeminfo»

Основной задачей приложения является получение всей доступной информации о сетевых интерфейсах устройства. Для решения этой задачи используется метод «get\_interfaces» (см. Приложение: листинг 1). В нем при помощи next циклом перебираются все интерфейсы сетевого устройства и записываются в лист. Далее при помощи стандартного метода pysnmp getcmd (прим — использование описанного выше метода get\_oid невозможно, так как необходимо послать множество MIB в одном SNMP запросе) осуществляется запрос всей необходимой информации о интерфейсах по их индексам. Полученная информация оформляется в виде кортежа листов, каждый из которых хранит информацию о конкретном сетевом интерфейсе. Информацию об одном сетевом интерфейсе можно оформить в виде Таблицы 1.

Таблица 1. Информация о сетевых интерфейсах

|  |  |
| --- | --- |
| **Информация** | **Используемый MIB** |
| ip-адресс | IF-MIB::IPAdEntAddr |
| маска | IF-MIB::IPadentnetmask |
| описание | IF-MIB::ifDescr |
| состояние | IF-MIB::ifOperStatus |
| тип интерфейса | IF-MIB::ifType |
| MTU | IF-MIB::ifMtu |
| скорость | IF-MIB::ifSpeed |
| физический адрес | IF-MIB::ifPhysAddress |
| sysuptime интерфейса | IF-MIB::ifLastChange |
| полное число полученных байтов | IF-MIB::ifInOctets |
| число полученных unicast пакетов | IF-MIB::ifInUcastPkts |
| число полученных broadcast пакетов | IF-MIB::ifInNUcastPkts |
| число полученных, но отвергнутых пакетов | IF-MIB::ifInDiscards |
| число пакетов, полученных с ошибкой | IF-MIB::ifInErrors |
| число пакетов, полученных с ошибочным кодом протокола | IF-MIB::ifInUnknownProtos |
| полное число отправленных байтов | IF-MIB::ifOutOctets |
| число отправленных unicast пакетов | IF-MIB::ifOutUcastPkts |
| число отправленных broadcast пакетов | IF-MIB::ifOutNUcastPkts |
| число отправленных, но отвергнутых пакетов | IF-MIB::ifOutDiscards |
| число пакетов, отправленных с ошибкой | IF-MIB::ifOutErrors |

Помимо вывода системной информации и получения таблицы сетевых интерфейсов  класс «Connection» позволяет получить аптайм сетевого устройства при помощи метода «get\_uptime», представленного в листинге 4. Также метод «get\_ifrouter», представленный в листинге 5 позволяет выяснить, является ли подключенное сетевое устройство маршрутизатором.

|  |
| --- |
| def get\_ifrouter(self): #маршрутизатор ли устройство    errorIndication, errorStatus, errorIndex, varBinds = next(self.get\_oid((ObjectIdentity('IP-MIB', 'ipForwarding',0).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs'))))    for varBind in varBinds:        if(varBind=='notForwarding'):           return(0)        else: return(1) |

Листинг 5. Метод определения маршрутизатора.

**2.3 ГРАФИЧЕСКИЙ ИНТЕРФЕЙС**

Графический интерфейс написан с помощью библиотеки PyQt5, которая является API для фреймворка Qt в языковой среде Python 3.

Модуль «unui» содержит классы «Connection» и «MyWin», которые создают объекты класса «dd» и «Ui\_MainWindow» соответственно, а также соотносят несколько обработчиков событий, массив соединений «Connection» и булевые переменные mrglobal, lock, lock2, которые позволяют избежать конкурентного доступа двух потоков: первый поток используется для отрисовки графического интерфейса, второй — для регулярного получения информации с удаленных хостов. В основном коде модуля инициализируются глобальные переменные и запускается главное окно программы.

В модуле «Extend» реализован класс «Ui\_MainWindow», внутри него реализован «dd». Класс «dd» содержит класс «Interface». Классовая организация модуля необходима для динамического создания объектов и обращения к ним. «Ui\_MainWindow» реализует главную форму приложения и содержит массив объектов «dd». «dd» — класс, который создает объект соединения на главной форме. «Interface» создает объект интерфейса внутри контейнера соединения.

**ГЛАВА 3. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС**

При запуске пользователя встречает главная форма программы. Здесь находится контейнер, который содержит в себе соединения.

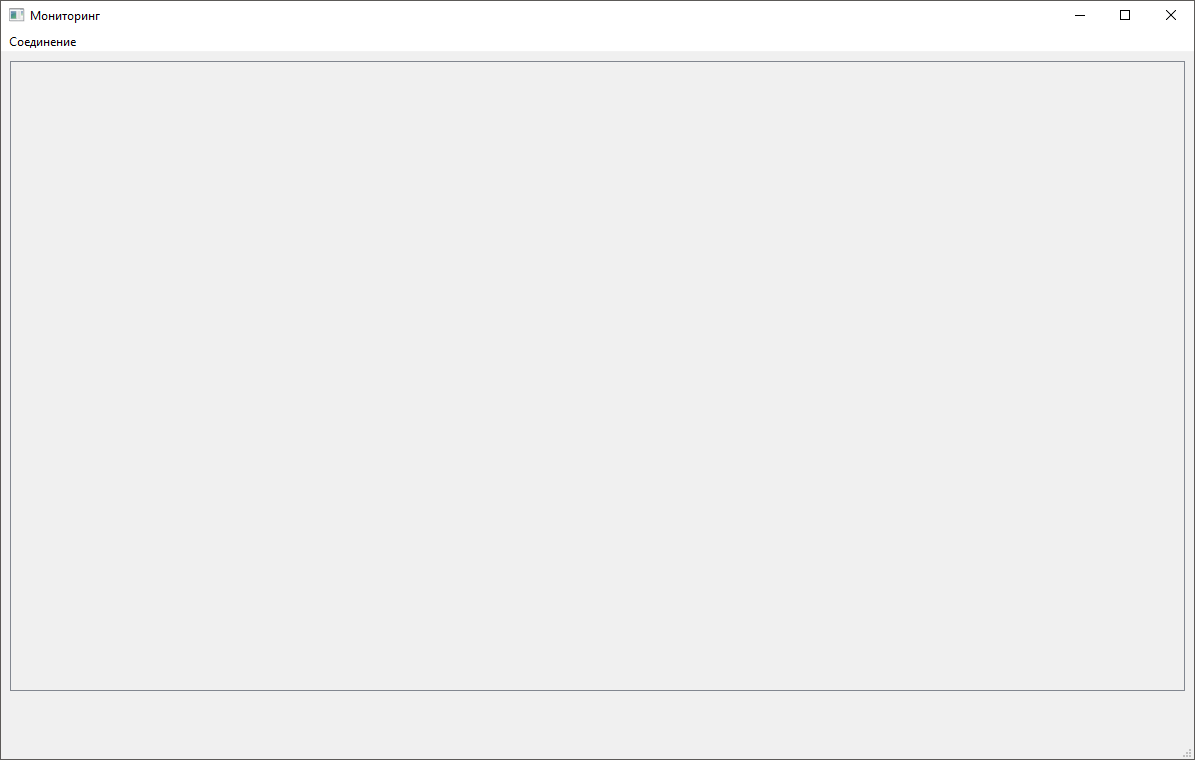


Рисунок 4. Главная форма программы без соединений

При нажатии выборе действия «Соединение» в верхнем меню у пользователя есть возможность создавать новое соединение.

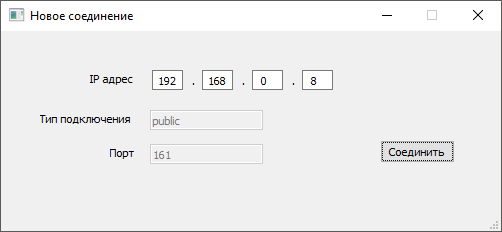


Рисунок 5. Окно создания нового интерфейса

При неправильном вводе у пользователя не будет возможности установить соединения. Если заданный хост недоступен, то пользователь будет об этом уведомлен и, соответственно, соединение не установится.



Рисунок 6. Главная форма с добавленным соединением

У пользователя есть возможность нажать на слайдер в левой части контейнера, в этом случае будет выводиться не только информация о системе, но еще аптайм  хоста и информация о его интерфейсах.

Обновление информации о соединении и интерфейсах происходит раз в 5 секунд.

Также присутствует кнопка удаления соединения.

**ЗАКЛЮЧЕНИЕ**

Таким образом, согласно целям курсовой работы, результатом является графическое приложение-менеджер SNMP для мониторинга сетевых устройств. Во время работы над проектом, были изучены возможности и пределы данного протокола в сфере мониторинга. Получены знания о работе с фреймворком Qt.

Трудности в реализации вызвали ( не было лол ) освоение инструментария pysnmp, помещение множества команд в один SNMP запрос и реализация перебора сетевых интерфейсов в таблице MIB.

Программа скомпилирована (прим. — скомпилирован код, написанный на языке C#, который запускает процесс интерпретатора языка Python 3 и передает ему путь до исполняемого скрипта — unui.py),  отлажена, аварийного завершения не замечено. Для тестирования SNMP соединений использовались виртуальные машины, в их числе Debian 9, СentOS 7, Windows 10, Windows server 2008 R2. Все описанные в первой главе цели успешно выполнены, алгоритмы из второй главы реализованы и итоговое приложение выполняет все поставленные задачи.

**ПРИЛОЖЕНИЕ**

Листинг 1.

|  |
| --- |
| def get\_interfaces(self):    resultlist=[] #финальный лист листов, каждый лист хранит [индекс, название, айпи, маска]    iplist=[]    masklist=[]    indlist = []  # хранит индексы интерфейсов    flag=1    oid\_generator = (self.snmp\_getnextoid('1.3.6.1.2.1.4.20.1.1'))    while(flag==1): #получение индексов, ipv4, масок        errorIndication, errorStatus, errorIndex, varBinds =next(oid\_generator)        if errorIndication:            return (errorIndication)        elif errorStatus:            return ('%s at %s' % (            errorStatus.prettyPrint(), errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))        else:            for varbind in varBinds:                if(str(varbind).find('mib-2.4.20.1.1')!=-1):                    iplist.append(str(varbind).split(' ')[-1])                if(str(varbind).find('mib-2.4.20.1.2')!=-1):                    indlist.append(str(varbind).split(' ')[-1])                elif (str(varbind).find('mib-2.4.20.1.3') != -1):                    masklist.append(str(varbind).split(' ')[-1])                elif(len(masklist)>0):                    flag = 0    for ind in range(1, len(indlist)+1): #получение названия интерфейсов и объединение всего в resultlist                                         #Результат:                                         # [                                         # индекс | айпи | маска | название | состояние | тип интерфейса | MTU | скорость | физический адрес | Sysuptime интерфейса |                                         #                                         #Полное число полученных байтов | число полученных unicast пакетов | число полученных broadcast пакетов | Число полученных но отвергнутых пакетов|                                         #Число пакетов, полученных с ошибкой | Число пакетов, полученных с ошибочным кодом протокола |                                         #                                         # Полное число отправленных байтов | число отправленных unicast пакетов | число отпарвленных broadcast пакетов | Число отправленных, но отвергнутых пакетов|                                         # Число пакетов, отправленных с ошибкой                                         # ]        templist=[]        templist.append(ind)        templist.append(iplist[ind-1])        templist.append(masklist[ind-1])        errorIndication, errorStatus, errorIndex,varBinds = next(getCmd(SnmpEngine(),                                CommunityData(self.community),                                UdpTransportTarget((self.ipaddr, 161)),                                ContextData(),                                ObjectType(ObjectIdentity('IF-MIB', 'ifDescr',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifOperStatus',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifType',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifMtu',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifSpeed',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifPhysAddress',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifLastChange',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                #полученные пакеты                                ObjectType(ObjectIdentity('IF-MIB', 'ifInOctets',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifInUcastPkts',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifInNUcastPkts',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifInDiscards',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifInErrors',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifInUnknownProtos',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                #отправленные пакеты                                ObjectType(ObjectIdentity('IF-MIB', 'ifOutOctets',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifOutUcastPkts',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifOutNUcastPkts',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifOutDiscards',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),                                ObjectType(ObjectIdentity('IF-MIB', 'ifOutErrors',ind).addMibSource('/opt/mibs/pysnmp').addMibSource('python\_packaged\_mibs')),        ))        if errorIndication:            return (errorIndication)        elif errorStatus:            return ('%s at %s' % (errorStatus.prettyPrint(),errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))        else:            for varBind in varBinds:                templist.append(' '.join([x.prettyPrint() for x in varBind]).split(' ')[-1])        resultlist.append(templist)    return resultlist |

**СПИСОК ЛИТЕРАТУРЫ**

1. Pysnmp official documentation. [Электронный ресурс] http://snmplabs.com/pysnmp/docs/snmp-design.html
2. PyQt5 Reference Guide. [Электронный ресурс]

http://pyqt.sourceforge.net/Docs/PyQt5/