1. Утверждаю
2. Директор института СПИНТех
3. НИУ МИЭТ
4. Проф. \_\_\_\_\_\_\_\_\_\_/Гагарина Л.Г./
5. «\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.

Пояснительная записка

Направление подготовки – 09.03.04

1. Квалификация – бакалавр

Руководитель выпускной работы:

К.т.н., профессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Касимов Р.А./

* 1. Исполнитель:
  2. Студент гр. ПИН-41 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Федотов А.А./

Москва 2021

Содержание

[Перечень используемых сокращений 3](#_Toc72852231)

[Введение 4](#_Toc72852232)

[1. Исследовательский раздел 5](#_Toc72852233)

[1.1 Протоколы синхронизации времени 5](#_Toc72852234)

[Выводы по разделу 21](#_Toc72852235)

[2. Конструкторский раздел 22](#_Toc72852236)

[2.1 Выбор языка 22](#_Toc72852237)

[2.3 Схема алгоритма 37](#_Toc72852238)

[2.4 Организация связи с другими модулями. 38](#_Toc72852239)

[Выводы по разделу 39](#_Toc72852240)

[3. Технологический раздел 40](#_Toc72852241)

[3.1 Методы тестирования 43](#_Toc72852242)

[3.2 Выбор средств для тестирования ПМ НКСС 45](#_Toc72852243)

[3.3 Прототипирование 46](#_Toc72852244)

[3.4 Отладка по точкам останова 47](#_Toc72852245)

[3.5 Отладка с помощью трассировки 48](#_Toc72852246)

[3.6 Отладка ПМ НКСС 50](#_Toc72852247)

[3.7 Модульное тестирование 51](#_Toc72852248)

[Выводы по разделу 52](#_Toc72852249)

[Заключение 53](#_Toc72852250)

[Список литературы: 54](#_Toc72852251)

[Приложение 1. Фрагмент кода программы 57](#_Toc72852252)

[Приложение 2. Код написанных тестов 92](#_Toc72852253)

# Перечень используемых сокращений

ПМ – Программный модуль

НКСС – настройка конфигурации сетевых сервисов

ПАК – программно-аппаратный комплекс

NTP – Network Time Protocol

ЯП – язык программирования

IDE (англ. Integrated Development Environment) - интегрированная среда разработки

JVM – виртуальная машина Java

ПО – программное обеспечение

SNTP (англ. Simple Network Time Protocol) – упрощенный протокол NTP

OSI (англ. Open Systems Interconnection model) — сетевая модель стека сетвых протоколов OSI/ISO.

RFC – документ из который содержит технические спецификации и стандарты, широко применяемые во всемирной сети.

VS (Visual Studio) – среда разработки ПО

UDP – протокол передачи данных

UTC – это всемирное координированное время, стандарт времени, принятый на Земле.

CLR — исполняющая среда для байт-кода CIL (MSIL), в который компилируются программы, написанные на .NET-совместимых языках программирования

BCL – стандартная библиотека классов платформы «.NET Framework».

# Введение

Информационные технологии все больше и больше проникают в жизнь людей. С их развитием появилась необходимость во взаимодействии компьютеров между собой. Для того, чтобы это взаимодействие обеспечить, создаются и постоянно улучшаются различные сетевые сервисы, которые позволяют решать те или иные задачи. Одной из таких задач является синхронизация времени сетевых устройств. Синхронизация времени необходима для согласования работы устройств и приложений, осуществляющих обработку данных в режиме реального времени. Кроме того, она необходима в системах управления для протоколирования каких-либо событий и своевременного реагирования на них. Для контроля синхронизации используется протокол NTP (Network Time Protocol – протокол сетевого времени), однако, чтобы правила протокола соблюдались, нужно правильно настроить конфигурацию устройства.

Целью данной работы является создание программного модуля, реализующего настройку конфигурации NTP на сетевых устройствах.

Выполнение выпускной квалификационной работы проходило в компании ООО «С-Терра СиЭсПи», специализирующейся на разработке и производстве средств для обеспечения сетевой информационной безопасности, а также ПО для реализации этих средств.

Разрабатываемый модуль ПМ НКСС имеет высокую практическую значимость при решении задач по настройке синхронизации сетевого времени на устройствах, поддерживающих протокол NTP.

Пояснительная записка состоит из введения, исследовательского, конструкторского, технического разделов, заключения и списка литературы.

В исследовательском разделе производится анализ предметной области настройки конфигурации сетевых устройств по протоколу NTP, рассматривается актуальность выбранной темы, производится сравнение существующих решений и выявляются их недостатки.

В конструкторском разделе выбирается язык и среда программирования, определяются необходимые библиотеки, производится разработка алгоритма программного модуля.

# Исследовательский раздел

Все сетевые устройства имеют внутренние часы. Они инициализируются при загрузке системы, затем время уже поддерживается с помощью регулярных прерываний от таймера, так что они работают даже при выключенном устройстве. Эти внутренние часы отслеживают как время, так и дату. Важно следить за точностью часов, иначе могут возникнуть различные проблемы с корреляцией логов, ПО и т. Д.

Рассмотрим некоторые протоколы, которые используются для синхронизации времени на разных устройствах.

# Протоколы синхронизации времени

Daytime и Time

Самые первые попытки синхронизировать время начали предприниматься в 1983 году. Именно тогда появился документ RFC 867, в котором описывался протокол DAYTIME. Немногим позже, в 1985 году появился документ RFC 868, описывающий протокол TIME. Первый из них предназначался для людей, второй же для компьютера.

По протоколу DAYTIME нет строгого стандарта вывода данных, считается что человек, увидевший выведенную строку сразу поймет какое число и который час. Поэтому он не предназначен для обработки компьютером.

В то же время, протокол TIME нужен для обмена времени между компьютерами. Разберем принцип его работы. К серверу подключается какой-либо компьютер. Этот сервер отсылает ему пакет по протоколу UDP. В данном пакете содержится одно 32-битное число. Это число показывает количество секунд, прошедших с первого января 1900 года. Это число переполнится через 136 лет, а значит, в 2036 году этот протокол перестанет работать.

NTP – протокол

Из всего вышеперечисленного легко понять, что протоколы DAYTIME и TIME не способны обеспечивать необходимую для нас точность синхронизации, т. к. не учитывают время задержки, за время которой данные доходят от сервера клиенту. Чтобы решить эту проблему, в 1985 году американский компьютерный инженер Дэвид Л. Миллс из университета Делавэра, разработал новый протокол. Этот протокол назвали NTP(Network Time Protocol), или в переводе на русский язык – протокол синхронизации времени. Этот протокол был описан в документе RFC 958.

Новый протокол использует так называемый алгоритм Марзулло, названный в честь Кейта Марзулло из университета Калифорнии в Сан-Диего. В четвертой версии протокола (NTPv4) есть возможность достигать точности вплоть до 10 мс, включая такой фактор, как учет времени передачи данных.

Как уже сказано выше, данный протокол применяется для синхронизации времени между устройствами. Для своей работы он применяет еще один протокол, предназначенный для передачи данных – UDP.

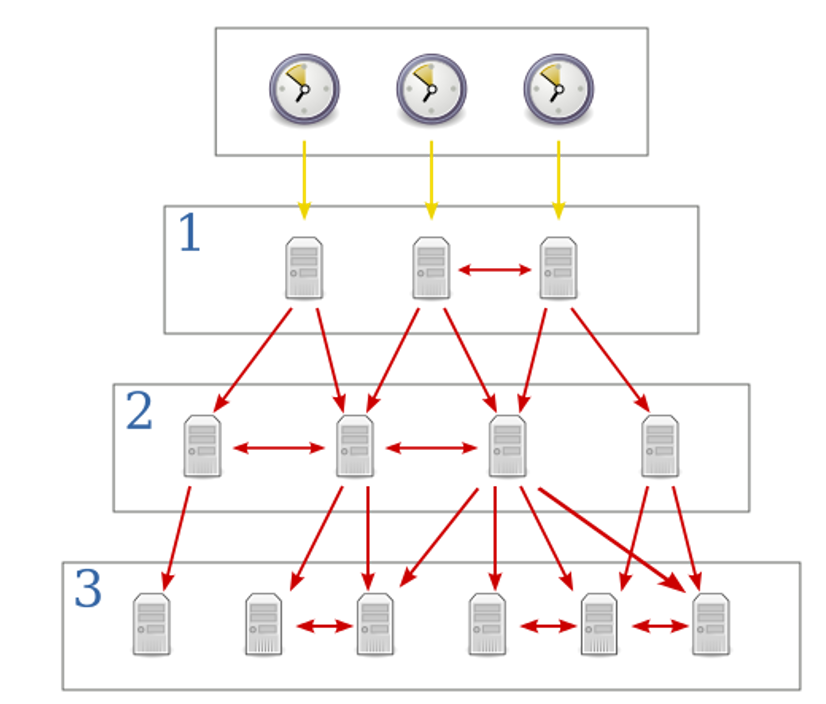


Рисунок 1 Схема работы NTP протокола.

Все сервера, работающие по протоколу NTP находятся в некой иерархической сети (см. рис. 1). Каждый новый уровень такой сети называется stratum, или же ярусом. На самом верхнем уровне (stratum0) находятся высокоточные часы, их еще называют эталонными. В качестве таких часов используются сигнал GPS или службы ACTS. Серверы первого яруса могут получать данные от этих часов. Сервера второго, в свою очередь. Могут получать данные от серверов первого яруса. Сервера третьего от второго и далее по аналогии. Всего в такой иерархии может быть до 15 ярусов.

Как NTP-серверы, так и NTP-клиенты получают данные от серверов первого яруса, хотя по факту лучше клиентам так не делать. Все дело в том, что огромное количество клиентских запросов могут оказаться слишком большой нагрузкой для серверов из stratum1. Будет гораздо удобнее, если настроить локальный сервер, к которому и будут подключаться клиенты.

Важно отметить, что этот протокол не настраивает текущее время на устройстве. Принцип действия заключается в том, что протокол корректирует время устройства с использованием разницы между временем на сервере и на внутренних часах.

Протокол NTP постоянно совершенствовался: NTPv1 (1988 г, RFC 1059), NTPv2 (1989 г., RFC1119), NTPv3 (1992 г., RFC1305), NTPv4 (1996 г., RFC2030).

SNTP протокол

Существует также и упрощенная версия протокола NTP. Она называется SNTP (Simple Network Time Protocol) и описана в документе RFC 4330. Этот протокол был создан для того, чтобы синхронизировать время в тех системах, где не трубется высокая точность.

Все основные плюсы NTP-протокола проявляются только в описанной выше иерархической сети. Для того, чтобы отправлять показания конечным пользователям, этот NTP оказался слишком сложным. Чтобы решить такую проблему был создан SNTP.

Отметим, SNTP не является каким-то новым протоколом. Он скорее является способом использования пакетов и серверов протокола NTP в приложениях, где не требуется высокая точность. Клиент будет использовать только какую-то часть информационного пакета, который был полученн от сервера.

Стоит сразу сказать, что SNTP – это не новый протокол. Это скорее способ использования NTP-пакетов и NTP-серверов в таких приложениях, где не особо требуется очень высокая точность, либо если такая точность просто недостижима. В таком случае клиент использует только часть информации пакета, полученного от сервера. SNTP может работать с любыми NTP-серверами, а еще с особенными SNTP-серверами. Их особенностью является то, что в своих откликах они будут заполнять только самые необходимые данные пакета.

Упрощенный протокол не образует сеть серверов. Вместо этого он образует пары клиент-сервер. Любой NTP-сервер одновременно является еще и SNTP-сервером. Клиент, который не передает полученное с сервера время далее, может работать и как NTP-клиент, и как SNTP-клиент. Все зависит от условий. Именно для протоколов SNTP и NTP зарезервирован 123 UDP порт.

В большинстве сетевых устройств используется протокол NTP. Разберем некоторые примеры сетевых устройств, которые при своей работе используют данный протокол.

* 1. Виды сетевых устройств

Прежде чем разбирать виды сетевых устройств, нужно разобраться с сетевой моделью OSI. Эта модель определяет несколько уровней взаимодействия между узлами сети. Каждый из этих уровней предоставляет по несколько специфических функций. Нижние уровни, такие как: физический, канальный и сетевой — определяют процесс передачи данных как таковой. Сетевые интерфейсы оконечных узлов представляют эти функции, но этого недостаточно, чтобы обеспечить связь между произвольными узлами в локальной сети и, тем более, в Интернет. Все дело в том, что невозможно установить физические связи между всеми узлами. Для того, чтобы решить эту проблему используют дополнительное сетевое оборудование.

Основными типами такого оборудования такие устройства как:

* Концентраторы
* Коммутаторы
* Маршрутизаторы

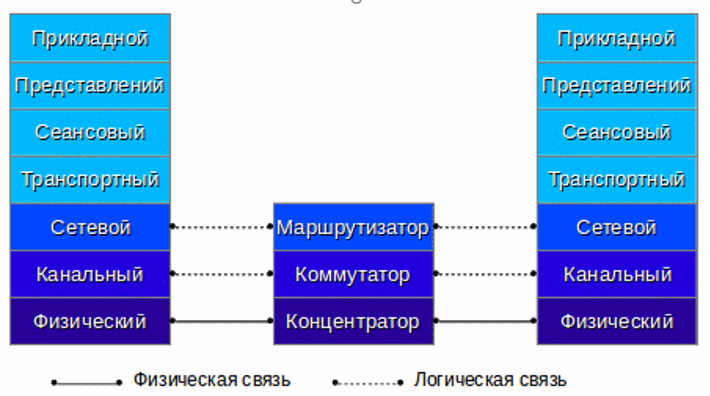


Рисунок 2 модель OSI

Концентратор

Концентратор (также хаб от англ. hub — центр) – сетевое устройство, работающее на первом уровне.

Все основные задачи хаба сводятся к пересылке полученных данных на все остальные подключенные к нему порты. Во время его работы никакой обработки данных не производится, из-за чего сеть, построенная на концентраторах имеет все недостатки такой топологии как «общая шина», кроме одного: если из строя выйдет какой-либо узел, то вся остальная сеть продолжит нормальное функционирование.

Сегодня концентраторы практически нигде не используются. Вместо них пришли другие, более совершенные устройства – коммутаторы. Единственное преимущество хаба в том, что он очень дешевый. Оно влияло лишь в первые годы развития сетей. С улучшением и удешевлением микропроцессорных компонентов данное преимущество концентратора полностью свелось на нет. Дело в том, что цена вычислительной части коммутаторов и маршрутизаторов составляет малую часть по сравнению с ценой разъемов, разделительных трансформаторов, корпуса и блока питания, общих для концентратора и коммутатора.

Поговорим подробнее про недостатки концентратора, которые следуют из недостатков топологии «общая шина». Один из самых важных недостатков – это снижение пропускной способности сети при увеличении количества узлов. Кроме того, из-за того, что узлы не изолированы друг от друга на физическом уровне, все они будут работать со скоростью самого медленного узла. Рассмотрим пример: в сети есть узлы со скоростью 100 Мбит/с и всего один узел со скоростью 10 Мбит/с. Все узлы будут работать со скоростью 10 Мбит/с, даже если на этом узле вообще ничего не происходит. Ещё одним важным недостатком является то, сетевой траффик отправляется во все порты. Из-за этого снижается уровень сетевой безопасности, ведь это даёт возможность подключения и успешного использования снифферов (считыватели траффика).



Рисунок 3 сетевой концентратор

Коммутатор

Сетевой коммутатор (свитч) – устройство, которое предназначено для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети (рисунок 4).

Разберем принцип его работы. Данное устройство хранит в своей памяти таблицу, в которой находятся соответствующие узлам коммутатора адреса. При включении коммутатора эта таблица будет пустой и будет запущен режим обучения. В таком режиме поступающие на какой-либо порт данные передаются на все остальные порты, а МАС-адрес порта-отправителя заносится в таблицу. Если же МАС-адрес хоста-получателя имеется в таблице, данные передаются только получателю. Таким образом, со временем проходящий трафик локализируется.

Существует три способа коммутации. Каждый из них — это комбинация таких параметров, как время ожидания и надёжность передачи.

Способы коммутации

* С промежуточным хранением (Store and Forward). Коммутатор читает всю информацию в кадре, проверяет его на отсутствие ошибок, выбирает порт коммутации и после этого посылает в него кадр.
* Сквозной (cut-through). Коммутатор считывает в кадре только адрес назначения и после выполняет коммутацию. Этот режим уменьшает задержки при передаче, но в нём нет метода обнаружения ошибок.
* Бесфрагментный (fragment-free) или гибридный . Этот режим является модификацией сквозного режима, который частично решает проблему коллизий. В теории поврежденные кадры (обычно из-за столкновений) часто короче минимального допустимого размера кадра Ethernet, равного 64 байтам. Поэтому в этом режиме коммутатор отбрасывает кадры длиной меньше 64 байт, а все остальные после прочтения первых 64 байт в сквозном режиме передаёт дальше



Рисунок 4 сетевой коммутатор

Маршрутизатор

Маршрутизатор (роутер) – устройство, работающее на третьем уровне и выполняющее функции перенаправления трафика между сетями.

Он может связывать разнородные сети с различными архитектурами. Чтобы принять решение о том, нужно ли пересылать пакеты данных, используется информация о топологии сети и определенные правила, которые задает системный администратор. Обычно маршрутизатор берет адрес получателя и определяет путь, по которому нужно передавать данные с помощью таблицы маршрутизации. Если в таблице такого маршрута нет, то пакет отбрасывается

Есть и другие способы для определения маршрута, по которому будут пересылаться пакеты. Например, иногда может использоваться адрес отправителя или используемые протоколы верхних уровней и другая информация, которая может содержаться в заголовках пакетов. Часто маршрутизаторы могут проводить трансляцию адресов отправителя и получателя, шифрование/расшифровывание передаваемых данных и т. д.



Рисунок 5 маршрутизатор

* 1. Обзор аналогичных программных решений

В ходе предварительных исследований был проведен анализ существующих решений с функционалом, требуемым от разрабатываемого программного модуля. При анализе учитывался не только необходимый функционал: настройка параметров для синхронизации времени на устройствах по протоколу NTP, но и смежные возможности, облегчающие использование решения; возможности компании разработчика по актуализации функционала, и его расширению. Характеристики рассмотренных решений в сравнении с ПМ НКСС представлены в табл. 1.

Таблица 1

Сравнение аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Аналоги  Критерии | MikroTik RouterOS | Quagga | Junos OS | Cisco IOS | C-Терра Шлюз |
| Возможность настройки NTP | Есть | Нет | Есть | Есть | Есть |
| Способ настройки NTP конфигурации | С помощью конфигурационных файлов и команд | Нет | С помощью команд | С помощью команд | С помощью команд |
| Возможность сохранения логов NTP в отдельный файл | Нет | Нет | Нет | Нет | Да |
| Необходимость загрузки дополнительных файлов | Есть | Нет | Нет | Нет | Нет |
| Наличие пользовательского интерфейса | Графический интерфейс | Консоль | Графический интерфейс | Графический интерфейс | Консоль |

MikroTik RouterOS

MikroTik — это производитель сетевых устройств из Латвии. Компания занимается разработкой и продажей как проводного, так и беспроводного сетевого оборудования, в том числе маршрутизаторы, сетевые коммутаторы (коммутаторы), точки доступа, а также программное обеспечение, куда входят операционные системы и вспомогательное ПО. Эта компания основана в 1996 году с для продажи оборудования на развивающихся рынках.

Одним из продуктов данной компании является RouterOS. Это сетевая операционная система, созданная на базе Linux. Эта система предназначена для того, чтобы устанавливать на маршрутизаторы MikroTik RouterBoard. Кроме того, есть возможность установить эту систему на ПК, превратив его в маршрутизатор с функциями брандмауэра, VPN-сервера / клиента, QoS, точки доступа и других. Система также может служить в качестве адаптивного портала на основе системы беспроводного доступа. Также существует специальная версия RouterOS, называемая Cloud Hosted Router, для облачных виртуальных машин

Описываемая операционная система имеет несколько уровней лицензий. С каждым уровнем возрастает число функций. Кроме того, существует ПО под названием Winbox. Это ПО предоставляет пользователю графический интерфейс (GUI) для настройки ОС. Доступ к устройствам, находящимся под управлением RouterOS возможен также через веб-интерфейс, FTP, Telnet, и SSH. Существует интерфейс программирования приложений, который позволяет создавать специализированные приложения для управления и мониторинга.

Данная система может поддерживать множество сервисов и протоколов, которые могут использовать средние или крупные провайдеры, такие как OSPF, BGP, VPLS/MPLS. ОС является достаточно гибкой системой, и хорошо поддерживается компанией, как в рамках форума и предоставления различных Wiki-материалов, так и специализированных примеров конфигураций.

Данная система обеспечивает поддержку почти всех сетевых интерфейсов на ядре Linux. Из беспроводных чипсетов поддерживаются решения, основанные на Atheros и Prism. Компания также активно работает над модернизацией ПО, благодаря которой будет возможна полная совместимость устройств и ПО компании с сетевыми технологиями, вроде IPv6.

RouterOS предоставляет администратору графический интерфейс для более наглядной и простой настройки файервола, маршрутизации и управления QoS. Также, внутри интерфейса WinBox почти полностью реализована функциональность Linux-утилит, таких как iptables, iproute2, управление трафиком и QoS на основе алгоритма HTB.

Quagga

Quagga — пакет свободного программного обеспечения, поддерживающий протоколы динамической маршрутизации IP. Компьютер с установленным и сконфигурированным пакетом Quagga становится способен использовать любые из перечисленных ниже протоколов динамической маршрутизации:

* Routing Information Protocol (RIP): v1-3;
* Open Shortest Path First (OSPF): v2-3;
* Border Gateway Protocol (BGP): v4;
* Intermediate System to Intermediate System (IS-IS);
* Protocol Independent Multicast (PIM, только PIM-SSM).

Пакет Quagga может быть установлен на UNIX-подобные операционные системы. Quagga — это усовершенствованная версия GNU Zebra, компьютерной программы, развитие которой остановилось в 2005 году.

Такое название данный продукт получил от одного животного. Квагга – это один из подвидов зебры, который обитал в южной Африке. К сожалению, на данный момент этот вид был истреблен в конце XIX‐ого века. В отличие от животного, программный продукт пережил ныне заброшенный проект GNU Zebra. Последняя стабильная версия Zebra (0.95a) была датирована 2005-09-08. Большинство BGP маршрутизаторов, которые до этого использовали GNU Zebra, перешли на Quagga.

Quagga состоит из базового ядра (core daemon) zebra, выполняющего роль промежуточного уровня абстракции (abstraction layer) ядра ОС, и предоставляющего Zserv API клиентам по протоколу TCP. Клиентами Zserv выступают службы (демоны):

* ospfd (протокол OSPFv2);
* ripd (протокол RIP v1, V2);
* ospf6d (протокол OSPFv3 IPv6);
* ripngd (протокол RIPng IPv6);
* bgpd (протокол BGPv4+, включая поддержку multicast и IPv6));
* isisd (протокол IS-IS);
* pimd (протокол PIM, пока только PIM-SSM).

Благодаря библиотеке Quagga довольно сильно упрощается разработка дополнительных модулей. Из-за чего все ее службы могут использовать унифицированный способ конфигурации и управления.

JUNOS

JUNOS — это операционная система, которая используется на оборудовании компании Juniper Networks.

Эта система создавалась на основе FreeBSD – свободно распространяемой ОС. Сейчас одним из главных конкурентов этой системы считается IOS-XR откомпании Cisco Systems.

В системе junos присутствует возможность для установки дополнительного ПО. Это ПО обычно распространяется в виде пакетов, которые подписаны соответствующими сертификатами компании.

Окружение пользователя кроме всего прочего включает в себя набор классических для системы FreeBSD утилит. Чтобы внести какие-либо изменения, нужно использовать утилиту cli. С ее помощью управляются все интегральные схемы специального назначения

Серии оборудования Juniper Networks, работающие под управлением JUNOS:

* сервисные шлюзы — SRX
* сервисные маршрутизаторы — J
* маршрутизаторы — M, MX, T
* коммутаторы — EX
* коммутаторы для ЦОД – QFX

Благодаря командному интерфейсу, кроме выполнения команд также можно вводить новую конфигурацию. Стоит сразу сказать, что новая конфигурация применяется только после выполнения команды commit. В данной системе существует возможность откатываться на одну из предыдущих (поддерживается возможность сохранения 50) версий конфигурации, либо, если она в течение какого-то промежутка времени не была применена, то откат происходит автоматически.

Как и большинство других интерфейсов командной строки маршрутизаторов, junos поддерживает автодополнение по уникальной комбинации первых символов команды (например, sh int fe-1/1/1 ex будет расшифрованно как show interface fe-1/1/1 extensive). Автодополнение происходит не при интерпретации команды, а при нажатии пробела. Поддерживается контекстная справка (вызов — знак вопроса).

Конфигурация данной системы представляет из себя некоторые директивы для конфигурирования различных подсистем. Эти директивы могут содержать вложенные элементы, которые описывают настройки отдельных компонентов. Как пример, конфигурация для интерфейса ethernet может содержать вложенные элементы, предназначенные для настройки отдельных подсетей, а те в свою очередь, иметь настройки для разных протоколов (ip4, ip6).

Те узлы, которые не содержат вложенных элементов (их еще называют оконченными) заканчиваются точкой с запятой. В то же время, те узлы, в которых есть вложенные элементы задают их при помощи фигурных скобок, а точка запятой в этом случае не ставится. Конфигурирование системы осуществляться либо вводом готового текстового блока, либо использованием сокращённых директив при помощи команды set.

Cisco IOS

Cisco IOS (от англ. Internetwork Operating System — Межсетевая Операционная Система) — программное обеспечение, которое используется в маршрутизаторах и сетевых коммутаторах Cisco. Cisco IOS является операционной системой, которая выполняет функции сетевой организации, маршрутизации, коммутации и передачи данных.

В данной системе присутствует довольно специфический интерейс командной строки, которые копируются многими сетевыми продуктами для удобства работысистемных администраторов. В интерфейсе содержится набор многословных команд, который может отличаться в зависимости от выбранного режима и уровня привелегий. Global configuration mode дает возможность изменять настройки системы и сетевых интерфейсов.

Всем командам интерфейса приписывается определённый уровень привилегий от 0 до 15, к которым могут обратиться только те пользователи, у которых соответствующий уровень привилегий. С помощью командного интерфейса можно определить доступные команды для каждого уровня привилегий.

Существуют разные компоновки IOS отличающиеся функционалом:

* IP Base — начальный уровень функциональности, включается во все другие.
* IP Services (для L3 свитчей) — протоколы динамической маршрутизации, NAT, IP SLA.
* Advanced IP Services — добавляется поддержка IPv6.
* IP Voice — добавляет функциональность VoIP и VoFR.
* Advanced Security — добавляется IOS/Firewall, IDS, SCTP, SSH и IPSec (DES, 3DES и AES).
* Service Provider Services — добавляется Netflow, SSH, BGP, ATM и VoATM.
* Enterprise Base — добавляется поддержка L3-протоколов (IPX и AppleTalk), а также DLSw+, STUN/BSTUN и RSRB.

С-Терра Шлюз

ПАК «С-Терра Шлюз» является программно-аппаратным средством защиты сетей, подсетей, офисов и самого шлюза от несанкционированного доступа.

ПАК «С-Терра Шлюз» используется для того, чтобы защищать от несанкционированного доступа к информации с ограниченным доступом, которая не содержит каких-либо сведений, составляющих государственную тайну. ПАК «С-Терра Шлюз» может применяться:

* в значимых объектах критической информационной инфраструктуры (КИИ) до 1 категории включительно;
* в государственных информационных системах (ГИС) до 1 класса защищенности включительно;
* в автоматизированных системах управления производственными и технологическими процессами (АСУТП) до 1 класса защищенности включительно;
* в информационных системах персональных данных (ИСПДн), обеспечивающих 1, 2, 3 и 4 уровни защищенности персональных данных;
* в информационных системах общего пользования (ИСОП) II класса.

ПАК «С-Терра Шлюз» предназначен для работы на аппаратных платформах в архитектуре Intel (x86-64 совместимых) универсального назначения, отвечающих следующим минимальным требованиям: имеющих от 1 процессора, от 2 Гб ОЗУ, от 3 сетевых интерфейсов, подключаемых к внешним сетям:

* Аквариус T30S100DC, T30S001DC, T40S102DF-B;
* LN-S, LN-Si, LN-M, LN-L, LN-XL;
* АТБ-АТОМ-1.

ПАК «С-Терра Шлюз» работает под управлением операционной системы Debian Linux 9 с установленными последними обновлениями безопасности.

* 1. Требования к ПМ НКСС.

ПМ НКСС должен обеспечивать выполнение следующих функций:

* Правильное распознавание введенных команд
* При вводе неверной команды должно выводиться сообщение об ошибке с указанием места ошибки
* Должна быть возможность настройки конфигурации в качестве NTP-клиента
* Должна быть возможность настройки конфигурации в качестве NTP-сервера
* Должна быть возможность указания NTP-серверов, с которыми проводится синхронизация
* Должна быть возможность с просмотра статуса текущих настроек с помощью определенных команд
* Должна быть предусмотрена возможность сохранения логов в отдельный лог-файл
* Все возвращаемые данные должны выводиться обратно на командную строку
  1. Постановка цели и задач

Цель разработки:

Создать программный модуль, позволяющий упростить настройку конфигурации NTP сервисов на сетевых устройствах.

Задачи:

* Исследование предметной области
* Сравнительный анализ существующих решений
* Выбор платформы для реализации модуля
* Выбор языка и среды разработки
* Разработка схемы данных ПМ НКСС
* Разработка схемы алгоритма ПМ НКСС
* Программная реализация ПМ НКСС
* Тестирование и отладка модуля
* Разработка руководства оператора

Предполагаемый алгоритм решения

ПМ получает команду и необходимые данные из консоли. На основе введенной команды выполняются действия по изменению настроек конфигурации или формировании сообщения о текущих настройках NTP.

Если в настройках указан файл для логирования, то в этот файл сохраняются логи работы NTP-сервиса.

# Выводы по разделу

В исследовательском разделе была обоснована актуальность разработки ПМ НКСС. Исследована предметная область и проведен сравнительный анализ решений для настройки конфигурации NTP на сетевых устройствах.

# Конструкторский раздел

# Выбор языка

В ходе исследовательской работы был проведен сравнительный анализ языков программирования, результаты которого приведены ниже в таблице.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Язык  Критерий | С++ | Java | Python | C# | Objective C |
| Скорость работы | Высокая | Высокая | Низкая | Высокая | Высокая |
| Знание языка, опыт работы с ним | Есть | Есть | Нет | Нет | Нет |
| Синтаксис (удобство разработки) | Не очень удобный | Удобный | Не удобный | Удобный | Не удобный |
| Использование языка в других проектах кампании | Используется | Используется | Не используется | Не используется | Не используется |
| Простота соединения со смежными ПМ | Просто, т. к. смежные модули написаны на этом языке | Сложно | Сложно | Сложно | Сложно |

C++

C++ (читается как си-плюс-плюс) — это язык программирования общего назначения. Был разработан Бьярном Страуструпом в 1985 как расширения языка Си. В ранние годы его называли «Си с классами».

Может поддерживать различные парадигмы программирования, такие как:

* процедурное программирование
* объектно-ориентированное программирование,
* обобщённое программирование.

Данный язык имеет довольно богатую стандартную библиотеку. Эта библиотека включает в себя самые распространённые контейнеры и алгоритмы, такие как ввод-вывод данных, регулярные выражения, поддержка многопоточности и многие другие возможности.

В этом языке сочетаются свойства не только высокоуровневых, но и низкоуровневых языков. Если сравнить его с языком Си, то становится заметно, что большое внимание в нем уделено теме ООП и обобщенному программированию.

Этот язык очень широко используется при разработке ПО. Он заслуженно является одним из самых популярных языков программирования. Он может использоваться в создании операционных систем, при разработке разных прикладных программ, высокопроизводительных серверов и даже при разработке игр.

На протяжении всего существования языка, он развивался по нескольким принципам:

* его функции должны быть полезны в реальных программах
* каждая функция должна быть реализована достаточно очевидным способом
* программисты имеют право иметь свой собственный стиль написания кода, а язык должен иметь возможность поддерживать его
* типы, разработанные пользователем, должны иметь такую же поддержку, как и встроенные изначально типы
* ниже С++ не должно быть языка (за исключением ассемблера)
* неиспользуемые функции не должны отрицательно влиять на работу программы и т. д.

Даже из названия понятно, что он разрабатывался на основе языка Си. Его синтаксис был унаследован оттуда же. Чуть ли не самым основным принципом было сохранения его совместимости с языком Си. Существует довольно много программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, тем не менее, не все программы, написанные на Си поддерживаются компиляторами С++.

C#

C# (произносится как си шарп) — объектно-ориентированный язык программирования.

Его создали примерно в 2000 году в компании Microsoft для разработки на из платформе .NЕТ. Разработкой занимались Андерс Хейлсберг и его команда. Свое название язык получил благодаря нотной записи, в которой знак «#» обозначает повышение ноты на полтона. По еще одной версии, название получилось так же, как и у языка С++, только решетка означает 4 плюсика, что показывает, что язык является инкрементом языка C++.

В стандарте Ecma перечислены следующие цели для разработки C#:

* Этот язык задуман как простой, современный, объектно-ориентированный язык программирования общего назначения.
* Язык и его реализации должны обеспечивать поддержку принципов разработки ПО, таких как строгая типизация, проверка границ массива, обнаружение попыток использования неинициализированных переменных и автоматическая сборка мусора
* Важны надежность, надежность и продуктивность программного обеспечения.
* Язык предназначен для использования при разработке программных компонентов, подходящих для развертывания в распределенных средах.
* Поддержка интернационализации очень важна.
* Хотя приложения C # должны быть экономичными в отношении требований к памяти и вычислительной мощности , язык не предназначен для прямой конкуренции по производительности и размеру с C или языком ассемблера.

Его синтаксис схож с языком С и подобными ему. Эта схожесть заключается в нескольких вещах, таких как: точка с запятой для обозначения конца строки, фигурные скобки для группирования операторов, присваивание знаком равенства, квадратные скобки в массивах и т. д. С# может поддерживать полиморфизм, перегрузку операторов, (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, исключения, комментарии в формате XML и статическую типизацию.

Язык создавался в качестве языка прикладного уровня для CLR. Поэтому его возможности сильно зависят от возможностей CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, cможет ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR.

Objective-C

Objective-C — это объектно-ориентированный язык программирования, который добавляет обмен сообщениями в стиле SmallTalk. Используется корпорацией Apple в их продуктах macOS, IOS.

Обратимся к истории. В начале 1980 годов очень популярным было структурное программирование. В его основе лежит представление программы в виде структуры, состоящей из отдельных блоков. Благодаря такому программист мог разделить алгоритм на небольшие блоки. Тем не менее, по мере развития компьютерных технологий, возрастала сложность задач, которые необходимо было решать. С таким увеличением сложности структурное программирование приводило к снижению качества и удобства разработки. Разработчикам приходилось часто писать узкоспециализированные функции, которые редко могли использоваться в других программах.

Примерно в это же время Брэдом Коксом и Томом Лавом в их компании Productivity Products International (PPI) задались целью решить эту проблему. В результате их работы был создан язык Objective-C.

Новый язык должен был быть простым и основываться на языке Си, чтобы разработчикам было проще на него перейти. Одной из целей было также создание модели, в которой сами классы являются полноценными объектами, поддерживалась бы интроспекция и динамическая обработка сообщений.

Objective-C является расширением языка Си: любая программа, написанная на Си, является программой на Objective-C. Одной из отличительных черт Objective-C является динамичность: решения, обычно принимаемые на этапе компиляции, здесь откладываются до этапа выполнения.

Язык Objective-C может работать с метаинформацией. Другими словами, при выполнении всегда есть возможность узнать класс объекта, список его методов, проверить является ли класс потомком другого класса и т. д. В языке также есть поддержка протоколов (ᴨонятия интерфейса объекта и протокола четко разделены). Поддерживается не множественное наследование, но для протоколов поддерживается и множественное наследование. Объект может быть унаследован от другого объекта и сразу нескольких протоколов (хотя это скорее не наследование протокола, а его поддержка). На данный момент язык Objective-C поддерживается компиляторами Clang и GCC (под управлением Windows используется в составе MinGW или cygwin).

Java

Java (читается как джава) — объектно-ориентированный язык программирования общего назначения, со строгой типизацией.

Его разработала компания Sun Microsystems, которая позже была приобретена компанией Oracle. Разработка ведётся сообществом, организованным через Java Community Process. Язык и основные реализующие его технологии распространяются по лицензии. Права на торговую марку принадлежат корпорации Oracle.

Приложения, которые разрабатываются на этом языке переводятся в специальный байт-код, благодаря чему могут запускаться на любой архитектуре, в которой есть JVM. JVM (Java Virtual Machine) – виртуальная машина, которая запускает байт-код. Официально язык был запущен в 1995 году. В данное время стабильно занимает самые высокие места в рейтинге лучших языков программирования.

Изначально этот язык назывался Oak («Дуб»). Его разрабатывал Джеймс Гослинг для того, чтобы создавать ПО для бытовых электронных устройств. Из-за того, что язык с таким названием уже существовал, Oak был переименован в Java. По одной из версий, язык получил такое название из-за острова Ява, по другой название является аллюзией на кофе-машину, которая является примером бытового устройства, для которого этот язык и создавался. В соответствии с этимологией в русскоязычной литературе с конца двадцатого и до первых лет двадцать первого века название языка нередко переводилось как Ява, а не транскрибировалось.

Начиная с середины 1990-х годов язык стал очень широко использоваться для разработки клиентских приложений и серверного ПО. Тогда же определённое распространение получила технология Java-апплетов — графических Java-приложений, которые встраивались в веб-страницы. С развитием возможностей динамических веб-страниц в 2000-е годы технология стала применяться редко.

В веб-разработке применяется Spring Framework, для документирования используется утилита Javadoc.

Python

Python (в русском языке встречаются названия пито́н или па́йтон) — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.

Язык разрабатывался с целью повысить производительность разработчика и улучшить читаемость кода. Он полностью объектно-ориентированный. Необычной особенностью этого языка является отделение блоков кода пробельными отступами вместо точки с запятой. Синтаксис довольно простой и минималистичный, из-за чего довольно редко приходится обращаться к документации. Часто этот язык используется для написания скриптов. Одним из основных недостатков является низкая скорость работы и более высокое потребление памяти в сравнении с аналогичным кодом, написанным на Си или С++

Python является языком, поддерживающим различные парадигмы:

* программирования: императивное, метапрограммирование
* процедурное
* структурное
* объектно-ориентированное программирование
* функциональное программирование.

Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается с помощью декораторов, более полноценная поддержка обеспечивается благодаря дополнительным фреймворкам. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений.

Самыми основными чертами этого языка являются: динамическая типизация, обработка исключений, автоматическое управление памятью, многопоточные вычисления, высокоуровневые структуры данных и т. д. Кроме того, есть возможность разбивать программы на модули. Далее эти модули можно разбивать на пакеты

В стандартную библиотеку питона входит большое количество полезных функций от утилит для работы с текстом, до средств для написания сетевых приложений. Также всегда можно подключить дополнительные возможности вроде матмоделирования, написания веб-приложений, разработки игр и т. д. Существует и специализированный репозиторий программного обеспечения, написанного на Python, — PyPI. Данный репозиторий предоставляет средства для простой установки пакетов в операционную систему и стал стандартом для Python.

Python уж давно считается одним из самых популярных языков программирования он может использоваться в анализе данных, машинном обучении, DevOps и веб-разработке, а также в других сферах. Благодаря хорошей читабельности кода, простому синтаксису и отсутствию необходимости в компиляции этот язык очень хорошо подходит для обучения программированию.

В итоге сравнительного анализа был выбран язык С++, как наиболее удовлетворяющий поставленным задачам для разработки ПМ НКСС. Выбор этого языка программирования продиктован требованиями и дальнейшем использованием модуля работниками ООО «С-Терра».

* 1. Выбор среды программирования.

Современная IDE для разработки на C++ должна поддерживать последнюю версию стандарта. В среде должна быть реализована поддержка отладчика и фреймворков для тестирования.

От IDE требуется настройка сборки под разные платформы, возможность работы с системой контроля версий Git. Разработка ПМ происходит на удаленной виртуальной машине, следовательно необходима возможность подключения по SSH.

Для выбранного языка был проведен анализ и выбор интегрированной среды разработки. Результаты сравнения представлены в таблице 2. Оценка проводилась по ключевым для разработки характеристикам, определяющим удобство написания ПО и скорость разработки.

Таблица 2

Сравнение сред разработки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Eclipse | VS Code | Visual Studio | Code::Blocks | WinSCP |
| Возможность работы с Git | Есть | Есть | Есть | Есть | Нет |
| Удобство/опыт использования | Нет | Есть | Есть |  | Нет |
| Возможность подключения по SSH | Нет | Есть | Нет | Нет | Есть |
| Поддержка фреймворков для тестирования | Есть | Есть | Есть | Есть | Есть |
| Способ распространения | Бесплатная | Бесплатная | Бесплатная с ограничениями | Бесплатная | Бесплатная |

Ниже приведена подробная информация по каждой из представленных сред разработки

Eclipse

Eclipse — бесплатная интегрированная среда разработки для модульных кроссплатформенных приложений. Была разработана и все еще поддерживается Eclipse Foundation.

Самые известные приложения, основанные на этой платформе – это разные среды разработки ПО на разных языках. Самым популярным считается «Java IDE». Она не зависит от каких-либо проприетарных расширений и использует открытый API для доступа к своей платформе.

В самую первую очередь, Eclipse – это платформа, на которой разрабатываются расширения. Поэтому он стал очень популярным. При необходимости, любой программист может добавить свои модули. Уже существуют такие модули как: Java Development Tools (JDT), C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и средства для языков Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP, X10 (X10DT) и пр. от различных разработчиков. С помщью различных расширений можно добавить различные диспетчеры для работы с БД, серверами и т. п.

Наиболее известный модуль платформы – это Eclipse JDT. Он ориентируется на групповую разработку. В среде разработки есть система управления версиями, такими как CVS или GIT. Также при необходимости можно добавить отдельные плагины для других систем вроде Subversion, MS SourceSafe и т. д. Также есть поддержка трекера ошибок вроде Bugzilla, Jira и т. д. В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.

Eclipse написана на Java, поэтому является продуктом, не зависящим от платформы, за исключением библиотеки SWT, которая разрабатывается для всех распространённых платформ. Библиотека SWT используется вместо стандартной для Java библиотеки Swing. Она целиком и полностью опирается на нижележащую платформу, что обеспечивает быстроту и натуральный внешний вид пользовательского интерфейса, но иногда она вызывает на разных платформах проблемы совместимости и устойчивости приложений.

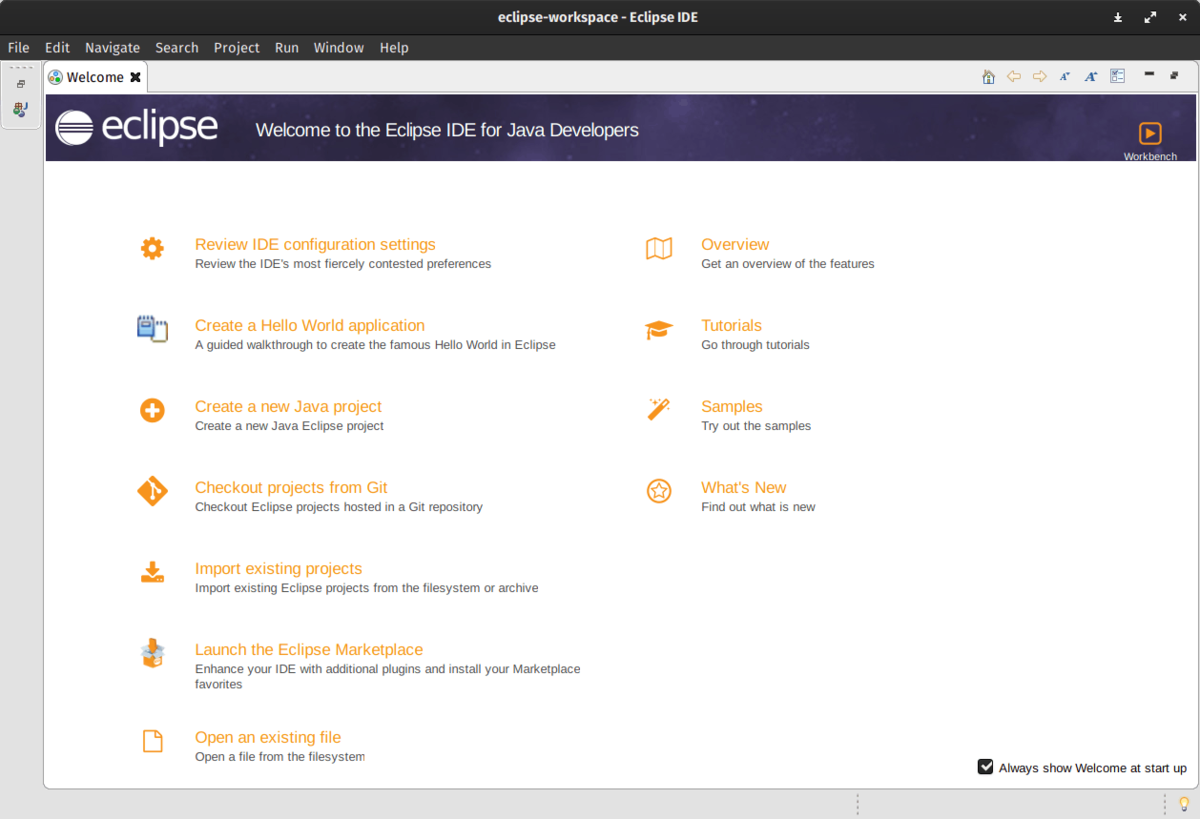


Рисунок 6 внешний вид Eclipse

Visual Studio Code

Visual Studio Code — это редактор исходного кода, разработанный компанией Microsoft для Windows, Linux и macOS.

Считается, что это легкий редактор кода для разработки приложений на разных платформах. В него входит отладчик, инструменты для работы с GIT, подсветка синтаксиса, а также различные средства для рефакторинга. Данный продукт имеет довольно большие возможности по кастомизации. Сюда можно отнести пользовательские темы, настраиваемые сочетания клавиш и файлы конфигурации. VS Code распространяется бесплатно и разрабатывается как ПО с открытым исходным кодом, однако готовые сборки распространяются под лицензией.

Многие возможности Visual Studio Code недоступны через графический интерфейс, зачастую они используются через палитру команд или JSON-файлы (например, пользовательские настройки). Палитра команд представляет собой подобие командной строки, которая вызывается сочетанием клавиш.

Данная среда разработки также позволяет заменять кодовую страницу при сохранении документа, символы перевода строки и язык программирования текущего документа.

Стоит добавить, что в 2018 году появилось расширение, позволяющее вести разработку на Python. Данное расширение с открытым кодом дает разработчикам довольно большие возможности для при редактировании, отладке и тестировании написанного кода.

С помощью встроенного в данный продукт пользовательского интерфейса есть возможность установить несколько тысяч расширений, находящихся в категории «языки программирования»

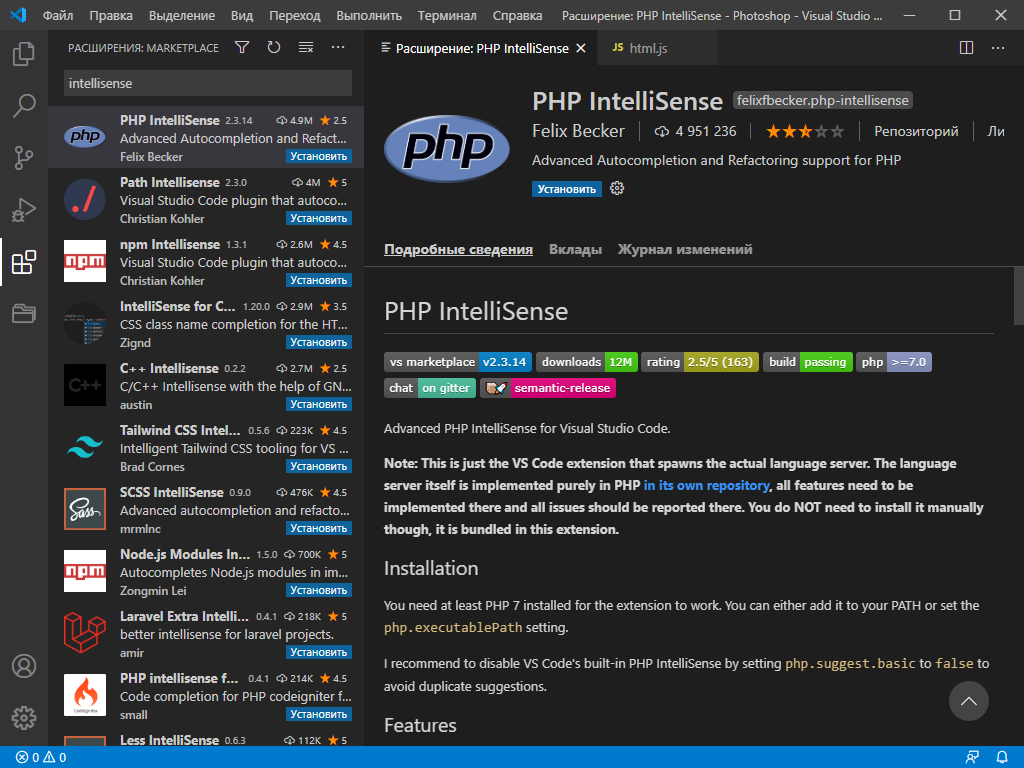


Рисунок 7 Внешний вид Visual Studio Code

Visual Studio

Visual Studio — линейка продуктов от компании Microsoft. Они включают интегрированную среду разработки ПО и ряд других инструментальных средств.

С помощью данных продуктов можно разрабатывать консольные приложения, игры, приложения с GUI, в том числе с помощью технологии Windows Forms веб-сайты, веб-приложения и т. д.

Внутри данного продукта содержится редактор исходного кода, который может поддерживать технологию IntelliSense и проводить простейший рефакторинг кода. Также встроен отладчик, который может работать в виде отладчика уровня исходного кода, либо в виде отладчика машинного уровня. Остальные встраиваемые инструменты могут включать в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. В данной среде разработке можно как создавать, так и подключать новые плагины (такие как Subversion и Visual SourceSafe), которые расширяют функциональные возможности IDE. Также можно добавить новые наборы инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Некоторые из наиболее популярных дополнений:

* dbForge Fusion for Oracle
* dbForge Fusion for MySQL
* dbForge Fusion for SQL Server
* Review Assistant — плагин просмотра и редактирования кода
* AnkhSVN — свободная реализация клиента Subversion в Visual Studio (в настоящее время поддерживаются версии с 2005 по 2013).

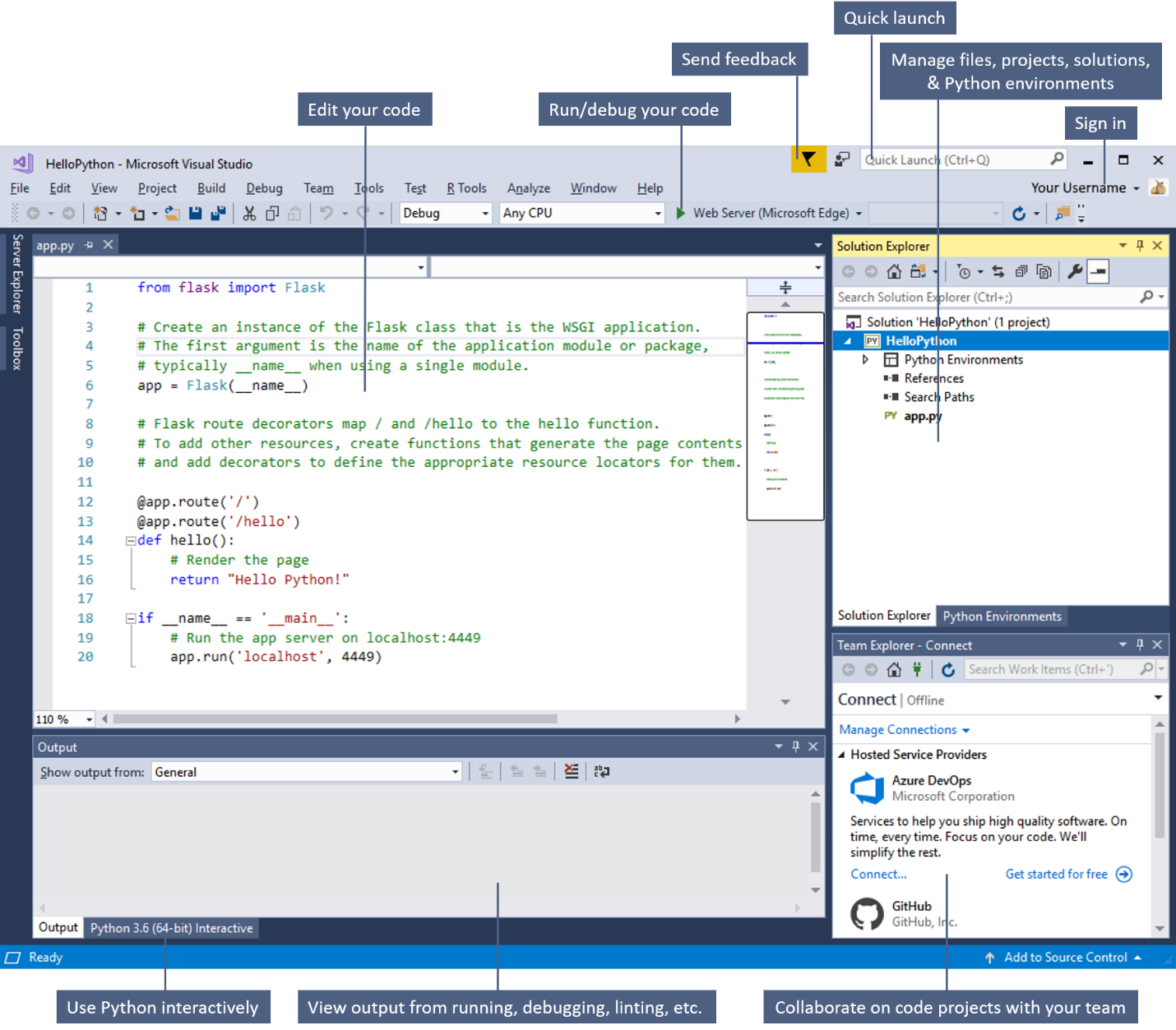


Рисунок 8 внешний вид MS Visual Studio

Code :: Blocks

Code :: Blocks - кросс-платформенная IDE c открытым исходным кодом, которая поддерживает несколько компиляторов , включая GCC, Clang и Visual C++ . Он разработан на C++ с использованием wxWidgets в качестве инструментария графического интерфейса. При использовании архитектуры плагина его возможности и функции определяются предоставленными плагинами. В настоящее время Code :: Blocks ориентирован на C , C ++ и Fortran . Он имеет настраиваемую систему сборки и дополнительную поддержку Make .

Code :: Blocks разрабатывается для Windows и Linux и был перенесен на FreeBSD , OpenBSD и Solaris. Последний двоичный файл для версии macOS - 13.12, выпущенный 12 декабря 2013 г. (совместим с Mac OS X 10.6 и более поздними версиями), но могут быть скомпилированы более свежие версии, и MacPorts поставляет версию 17.12.

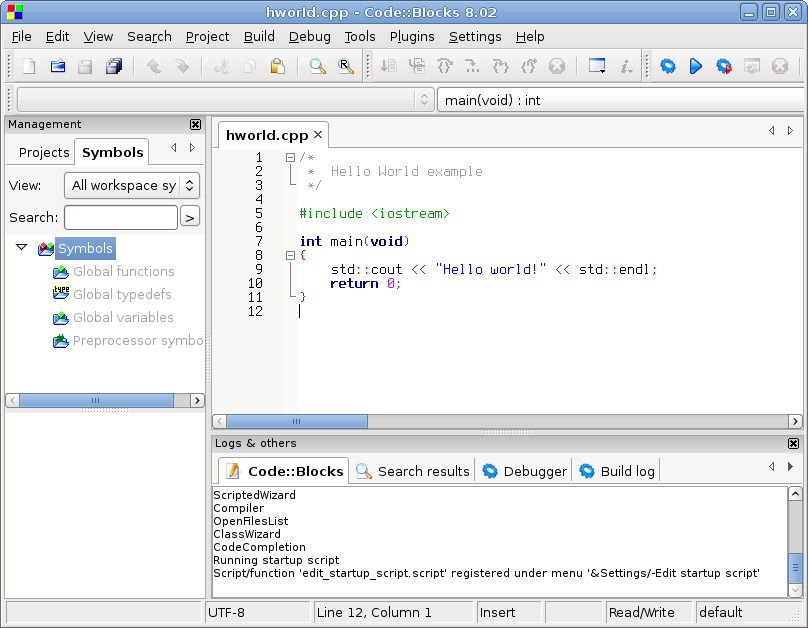


Рисунок 9внешний вид Code::Blocks

WinSCP

WinSCP — это свободно распространяемый графический клиент протоколов SFTP и SCP, работающий на Windows. Распространяется он по лицензии GNU GPL. Обеспечивает защищённое копирование различных файлов между компьютером и серверами, которые поддерживают эти протоколы.

Основные возможности:

* Графический интерфейс (GUI) в стиле Norton Commander и как в проводнике Windows Explorer (на выбор).
* Все самые основные файловые операции, такие как копирование, удаление и т. д.
* Автоматизация с использованием скриптов или интерфейса командной строки.
* Интеграция с Pageant (PuTTY Agent) с возможностью авторизации по открытым ключам.
* Интеграция с Windows (поддержка Drag&Drop, ярлыков, поддержка схем URL).
* Работа с ключами и версиями протокола SSH.
* Встроенный текстовый редактор и возможность выбора другого текстового редактора
* Поддержка различных типов авторизации, например: по паролю, аутентификации с открытым ключом, Kerberos.
* Возможность сохранить настройки соединений.
* Синхронизация папок по нескольким автоматическим и полуавтоматическим алгоритмам.
* Локализации интерфейса для нескольких десятков языков, в том числе русского.
* Возможность работы с использованием файла конфигурации вместо хранения настроек в реестре, что бывает очень удобно при запуске с переносных носителей.
* Поддержка протоколов SFTP и SCP поверх SSH-1 и SSH-2, а также FTP.
* Плагин для поддержки протокола SFTP в программе FAR Manager.

Другими словами, WinSCP не совсем среда разработки в чистом виде, скорее это просто клиент для более удобного подключения к серверу. Он позволяет менять файлы на сервере без использования командной строки, в том числе утилиты Vim. Поскольку для разработки требуется подключение по протоколу SCP данный вариант имеет один из высших приоритетов при выборе среды разработки.

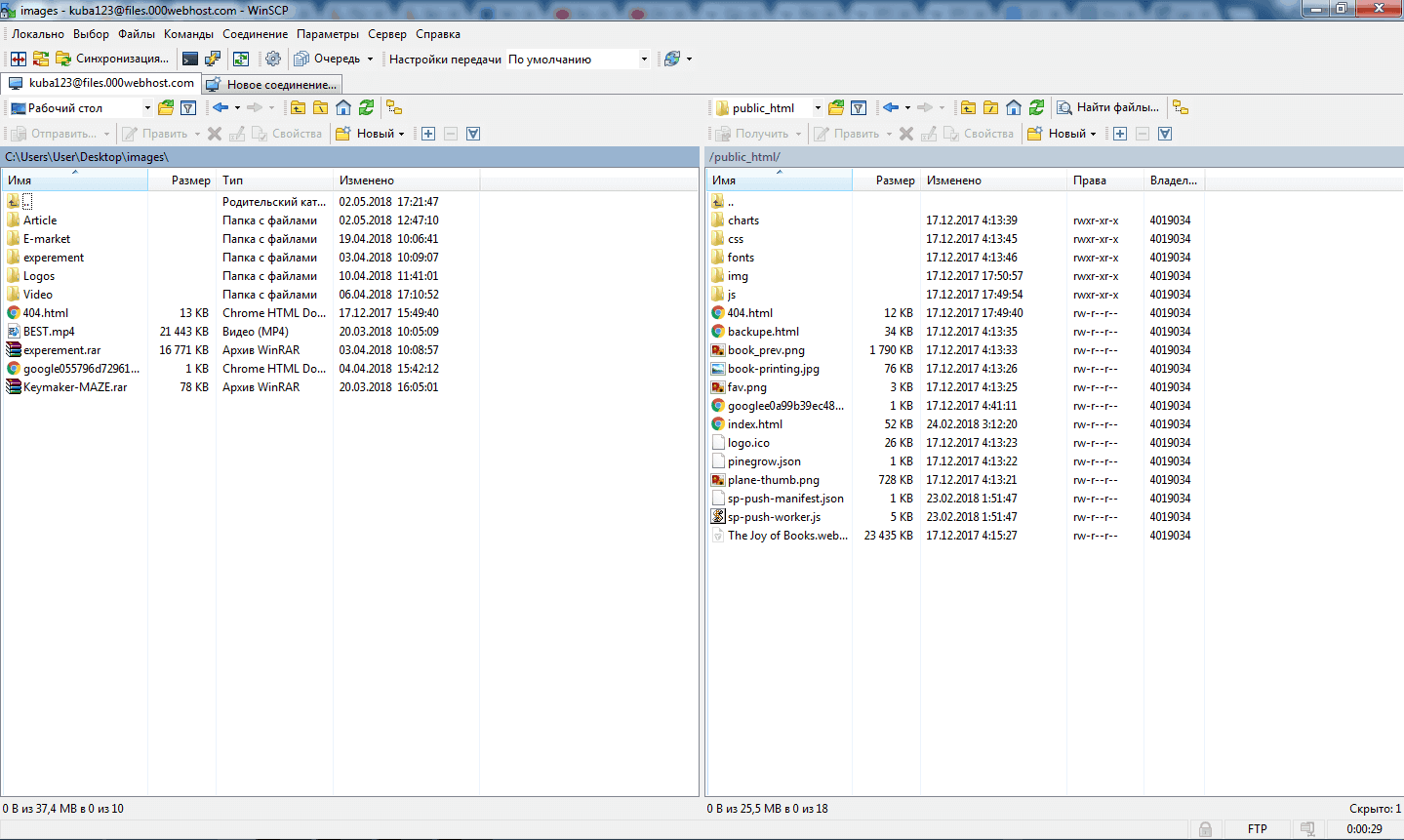


Рисунок 10 внешний вид WinSCP

# 2.3 Схема алгоритма

В начале работы ПМ НКСС получает на вход Cisco-like команду и ищет ее среди имеющихся в базе команд. В том случае, если введенная команда отсутствует, на консоль выводится сообщение об ошибке. Если же, такая команда есть проверяется, правильно ли введены данные для выполнения этой команды. В случае ошибки ПМ НКСС завершает свою работу и выводит сообщение об ошибке. Если же ошибок нет, то выполняются действия в зависимости от введенной команды. Команды могут быть на добавление нового NTP-сервера, на удаление одного из указанных серверов. В этом случае выполняются соответствующие настройки конфигурации. Также сетевой администратор может ввести команды на отображение текущего статуса. В этом случае в консоль выводятся запрошенные данные. Для наглядности, ниже представлена блок-схема алгоритма.

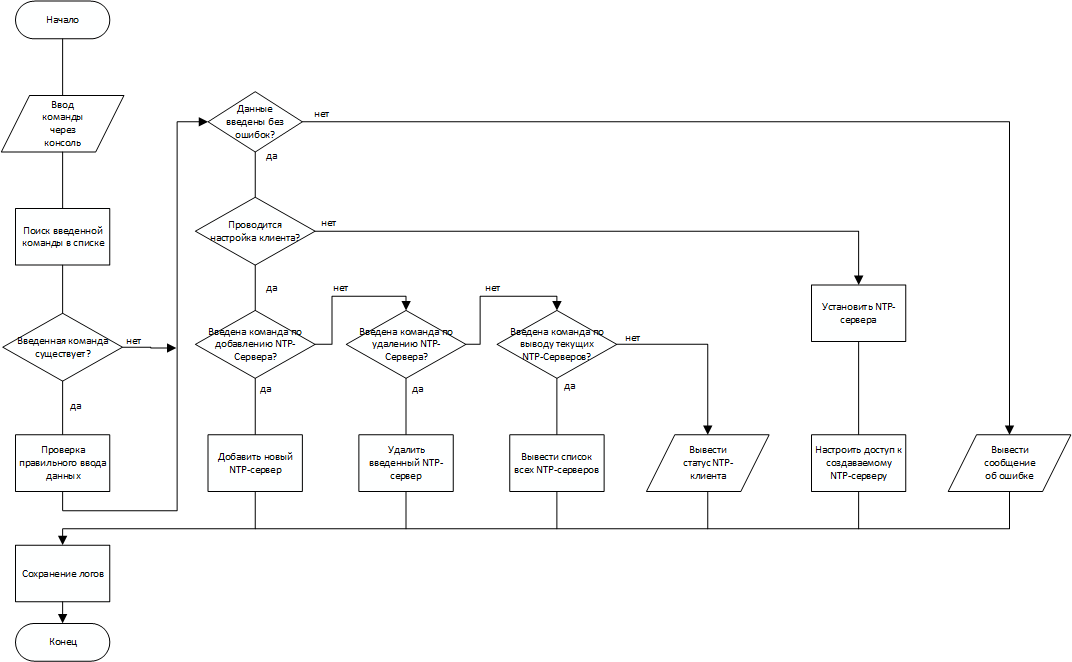


Рисунок 11 схема алгоритма

# 2.4 Организация связи с другими модулями.

Разрабатываемый ПМ НКСС является частью ПАК «С-Терра Шлюз» предназначен для упрощения настройки конфигурации NTP сервисов. Для своей работы он использует список команд, которые поддерживает ПАК, список модулей, реализующих эти команды, получает данные от модуля ввода данных, выводит данные с помощью модуля вывода и в зависимости от введенной команды и введенных данных, ПМ НКСС выставляет нужные настройки конфигурации.

Список команд, которые выполняет ПМ:

* Добавить новый NTP-сервер
* Удалить ранее введенный NTP-сервер
* Вывести в консоль список NTP-серверов
* Вывести на консоль статус NTP настроек



Рисунок 12схема данных

# Выводы по разделу

В конструкторском разделе был определен язык программирования для разработки, выбрана среда программирования.

Была спроектирована архитектура программного модуля, сформирован алгоритм работы.

# Технологический раздел

Тестирование ПО – это такой процесс, в котором проводятся испытания программного продукта. Оно проводится с целью проверить, насколько реальное поведение программы будет соответствовать ожидаемым прогнозам. Для этого проводится некоторое количество тестов, которые были выбраны по определенном принципам.

Самые первые программные системы обычно разрабатывались либо в качестве различных научных исследований, либо в качестве систем для военных. Тестирование в то время было строго формализованным. Записывалось абсолютно все: все тестовые процедуры, все тестовые данные, все полученные результаты. Тестирование тогда было отдельным процессом, который начинался после завершения написания кода. При этом, оно проводилось теми же людьми, что и разрабатывали код.

В 1960-е годы большое внимание уделяли исчерпывающему тестированию. Его проводили с использованием абсолютно всех путей в коде, вводились все возможные входные данные. Вскоре такой подход стал считаться слишком сложным. На то было несколько причин. Во-первых, входных данных было очень много, что уже сильно усложняло тестирование. Во-вторых, в программах существует просто огромное множество путей выполнения. В-третьих, было очень сложно искать проблемы в архитектуре и спецификациях. Из-за всех этих причин, данный способ тестирования был отклонен и признан теоретически невозможным.

Перенесемся в начало 1970-х годов. В это время тестирование ПО обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения». В зарождавшейся программной инженерии верификация ПО значилась как «доказательство правильности». Такая концепция была теоретически перспективной, но по факту она требовала довольно много времени и не была достаточно всеобъемлющей. Позже было решено, что доказывать правильность – не самый эффективный метод для тестирования ПО. Стоит заметить, что демонстрация правильной работы используется и в наши дни. В качестве примера можно представить премо-сдаточные исрытания.

Во второй же половине 1970-х годов тестирование стали представлять как выполнение программы с целью найти в ней какие-либо ошибки, а не доказать ее работоспособность. Успешный тест – это такой тест, которые обнаруживает ранее неизвестные ошибки или проблемы. Такой подход противоположен указанному ранее.

Указанные выше два определения – это парадокс тестирования. В его основе лежат два утверждения. С одной стороны тестирование позволяет нам убедиться, что продукт работает хорошо, а с другой – тестирование находит ошибки в программах, тем самым показывая, что программа не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки программного обеспечения.

В 1980-е годы тестирование расширилось таким понятием, как предупреждение дефектов. Проектирование тестов – это один из наиболее эффективных методов, предназначенных для предупреждения ошибок. В это же время начали часто говорить о том, что нужна методология тестирования. Конкретно, что в тестирование нужно включать проверки на всем протяжении цикла разработки, причем этот процесс должен быть управляемым.

Во время тестирования необходимо проверять не только саму собранную программу, но и требования, архитектуру и даже сами тесты. Традиционное тестирование, которое существовало до этого времени, относилось только к готовой системе, которая скомпилирована (сейчас это в основном называется системное тестирование), но позже все тестировщики стали вовлекаться во все аспекты жизненного цикла разработки. Благодаря такому подходу, стало возможно гораздо раньше находить проблемы в требованиях и архитектуре, из-за чего значительно сокращались сроки и бюджет, необходимые для разработки программ.

Ближе к середине 1980-х годов начали появляться первые инструменты для автоматического тестирования. Считалось, что компьютер может выполнить гораздо больше тестов, чем человек, при этом сделает это более точно и надежно. Поначалу эти инструменты были крайне простыми и не имели возможности написания сценариев на скриптовых языках.

В начале 1990-х годов в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений. Это означало, что идет переход от тестирования к обеспечению качества, которое охватывает весь цикл разработки ПО. В это время начинают появляться разные инструменты, предназначенные для поддержки процесса тестирования. Среди них: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования.

В середине 1990-х годов с развитием Интернета и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование» (по аналогии с гибкими методологиями программирования).

Существуют различные классификации тестирования:

* По объекту тестирования (функциональное, производительности, безопасности, локализации и т. д.)
* По знанию внутреннего строения системы (черного/белого/серого ящика)
* По степени автоматизации (ручное, автоматическое, полуавтоматическое)
* По времени проведения тестирования (альфа, бета)

И множество других классификаций, для наглядности представленных ниже.

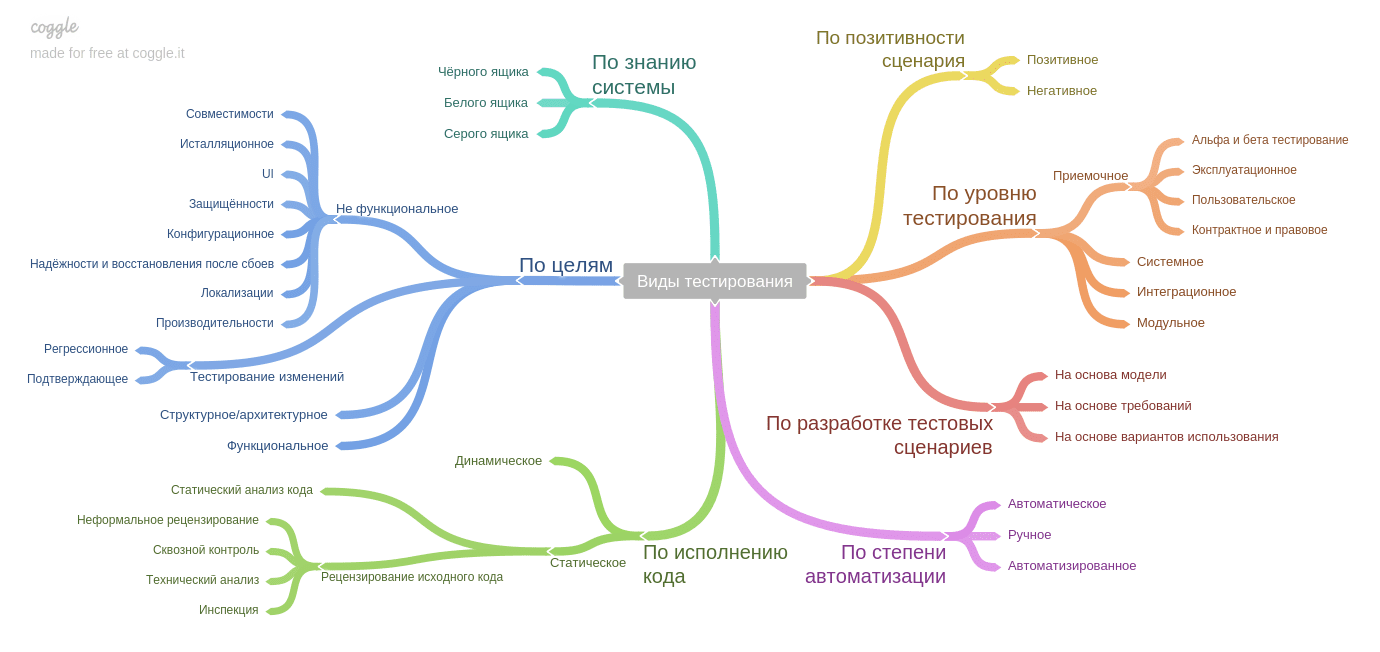


Рисунок 13виды тестирования

Разберем некоторые из методов тестирования.

# 3.1 Методы тестирования

Классификация по объекту тестирования.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, необходимые пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Тестирование производительности (англ. Performance Testing) в инженерии программного обеспечения — тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов. Данный вид тестирования включает в себя нагрузочное тестирование, стресс-тестирование, тестирование стабильности, конфигурационное тестирование.

Юзабилити тестирование - исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, пользовательский интерфейс или устройство) для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования взаимодействия человек-компьютер в целом — формулируют универсальные принципы.

Тестирование безопасности — оценка уязвимости ПО к различным атакам. В ходе тестирования испытатель играет роль взломщика. Он может использовать любые средства от попыток узнать пароль с помощью внешних средств до просмотра несекретных данных в надежде найти ключ для входа в систему.

Также существует много тестов по объекту тестирования, но они узконаправлены, поэтому рассматриваться не будут.

Среди перечисленных выше методов тестирования самым важным является функциональное тестирование. Поэтому оно обязательно будет проводится для разрабатываемого ПМ НКСС. У модуля пользовательским интерфейсом является Cisco-подобная консоль, в которую пользователь вводит команды. Поэтому юзабилити-тестирования проводиться не будет. В рамках данной работы не поднимались вопросы безопасности приложения, так что этот параметр также не тестируется.

Классификация по уровню детализации

Модульное (unit) тестирование проверяет отдельные небольшие части приложения, которые могут функционировать отдельно от других частей. Как правило, тестируются отдельные функции или классы. Этот вид тестирования позволяет на начальном этапе разработки обнаружить многие ошибки.

Интеграционное тестирование уже направлено на проверку взаимодействия отдельных модулей друг с другом.

Системное тестирование проверяет приложение как единое целое, собранное из отдельных модулей.

Приемочное тестирование оценивает соответствие требований к программному продукту

Для ПМ НКСС будут использоваться все методы из данной классификации.

Классификация по знанию внутреннего строения системы

Тестирование чёрного ящика или поведенческое тестирование — стратегия (метод) тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве (коде) тестируемого объекта. Иначе говоря, тестированием чёрного ящика занимаются тестировщики, не имеющие доступ к исходному коду приложения. Под стратегией понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций

Тестирование белого ящика (англ. white-box testing), также тестирование стеклянного ящика (англ. glass-box testing), структурное тестирование (англ. structural testing) — тестирование, которое учитывает внутренние механизмы системы или компонента (ISO/IEC/IEEE 24765). Обычно включает тестирование ветвей, маршрутов, операторов. При тестировании выбирают входы для выполнения разных частей кода и определяют ожидаемые результаты.

Тестирование серого ящика – это метод тестирования программного продукта или приложения с частичным знанием его внутреннего устройства. Для выполнения тестирования «серого ящика» нет необходимости в доступе тестировщика к исходному коду. Тесты пишутся на основе знания алгоритма, архитектуры, внутренних состояний или других высокоуровневых описаний поведения программы.

В данном случае разработчик также является тестировщиком, поэтому будет использоваться метод белого ящика.

Классификация по степени автоматизации

Ручное тестирование (manual testing) — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно производится тестировщиком без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя. В роли тестировщиков могут выступать и обычные пользователи, сообщая разработчикам о найденных ошибках.

Автоматизированное тестирование программного обеспечения — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Для данного ПМ НКСС будет использоваться как ручное, так и автоматическое тестирование.

# 3.2 Выбор средств для тестирования ПМ НКСС

Для выбора рассмотрим средства тестирования для НКСС

* функциональное тестирование;
* метод белого и черного ящика;
* модульное, интеграционное и системное тестирование;
* автоматизированное и ручное.

Средства и методы отладки программы

Существует несколько методов отладки, подразделяемые на статические и динамические, которые применяются в зависимости от требуемых условий и конкретной реализации программы.

Статические методы – это методы, для выполнения которых не требуется запуск и исполнение программы. Эти методы требуют значительных временных затрат от программиста при незначительных затратах вычислительных мощностей. К статическим методам отладки программ относят прокрутку программы вручную или с помощью программных анализаторов (например, компилятор), а также коллективную проверку кода программы и проверку кода программистом-технологом.

Экспериментально установлено, что в программах ручными методами удается обнаруживать от 30 до 70 % программных и алгоритмических ошибок из общего числа ошибок, выявленных при отладке. При этом одновременно осуществляется доработка программ с целью улучшения их структуры, логики обработки данных и для снижения сложности последующего автомати­зированного тестирования на РС.

В динамических методах используются в большей степени вычислительные мощности, и отладка происходит совместно с исполнением программы. Данный подвид методов отладки, как правило, привязан к конкретному транслятору (компилятору), либо к среде разработки. К динамическим методам отладки относят такие методы, как поиск ошибок с использованием сторонних системных средств, а также использование специальных отладчиков.

# 3.3 Прототипирование

Прототипирование – проверка функций (модулей, библиотек, и т.п.) в изоляции с помощью небольших примеров кода (прототипов). Прототип легче отлаживать, чем целевую систему. Если проблема воспроизводиться с помощью прототипа, отладка упрощается. Unit тестирование в этом смысле более эффективный метод отладки, поскольку unit test-ы выполняются автоматически и «накапливаются» для будущего реюза, а прототипы редко становятся частью системы.

Качества, которыми должен обладать эффективный прототип:

* Создание прототипа не должно быть затяжным
* Хорошие прототипы обычно одноразовые. Они нужны для того, чтобы донести идею продукта до заинтересованного лица. После этого прототип может больше не использоваться.
* Хороший прототип является сфокусированным, т. е. нужно обращать внимание на сложные части при его создании
* Необходимо обращать внимание на элементы взаимодействия, которые принесут пользу вашему продукту.

# 3.4 Отладка по точкам останова

В программировании точка останова (англ. breakpoint) — это запланированное прерывание программы. При этом вызывается отладчик (сразу с этим программа самостоятельно может использовать эти точки останова для своих нужд). После перехода к отладчику разработчик может отслеживать текущее состояние программы, используя логи, состояние памяти, стека и т. д. Это нужно для того, чтобы определить, правильно ли ведет себя программа, либо обнаружить ошибку. В отличие от остановки программы, в данном случае выполнение можно продолжить с того же места, где была точка.

На практике точка останова – это одно, либо несколько каких-либо условий для прерывания программы. Наиболее часто используется условие останова при переходе управления к указанной инструкции программы (instruction breakpoint). Другое условие останова — операция чтения, записи или изменения указанной ячейки или диапазона ячеек памяти (data breakpoint или watchpoint).

В процессе разработки и отладки ПМ НКСС довольно активно использовался такой динамический метод отладки, как отладка с использованием точек останова. Для выполнения данного метода отладки был использован стандартный отладчик, встроенный в среду разработки Visual Studio Code.

Суть такого метода отладки заключается в следующем. Специальные точки размещаются в тех местах кода, где необходимо проконтролировать состояние программы. После запуска в режиме отладки с точками останова, программа будет выполняться до того момента, как дойдет до такой точки. При достижении точки штатное выполнение кода программы будет приостановлено и управление передается на отладчик среды программирования. После этого становится возможным отследить состояния различных параметров программы на текущий момент, включая значения переменных и стек вызовов. На рисунке ниже изображено как выглядит отладка по точкам останова.

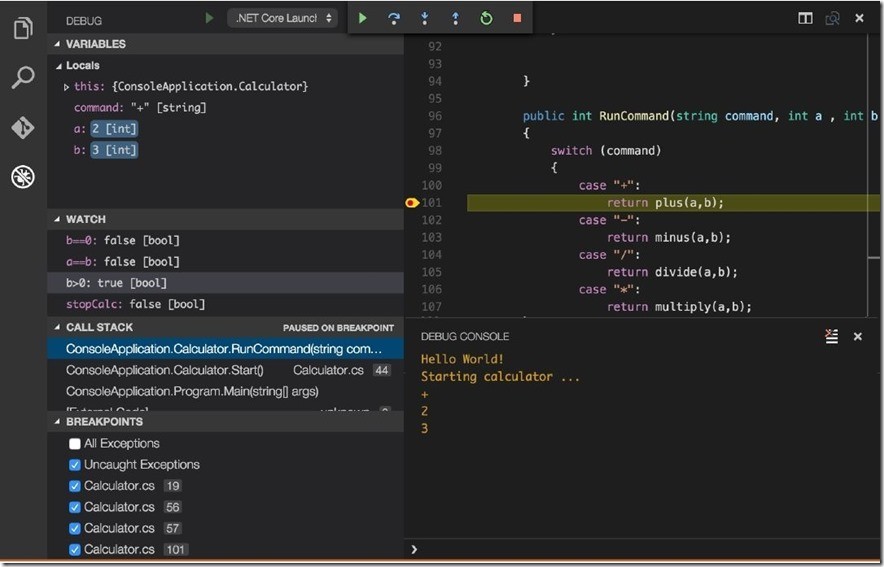


Рисунок 14 отладка в VS Code

# 3.5 Отладка с помощью трассировки

Трассировка – это метод отладки, при котором работа программы выполняется пошагово с шагом по строкам. Разработчик контролирует каждый шаг выполнения программы, стремясь полностью отследить её поведение.

При трассировке, изображенной на рисунке ниже, разработчик располагает следующими командами контроля исполнения программы:

шаг внутрь (англ. Step Into) – программа выполнит текущую строчку кода и курсор перейдет на следующую;

принудительный шаг внутрь (англ. Force Step Into) – программа выполнит текущую строчку кода и курсор перейдет на следующую игнорируя все возможные ограничения;

шаг с обходом (англ. Step Over) – практически аналогичен предыдущей команде, но если текущая строка подразумевает вход в отдельно описанную процедуру, то код, содержащийся в ней, выполнится без остановок;

выполнить до курсора(англ. Run to Cursor) – программа будет выполнять без остановки все строки, пока не дойдет до строки, на которой находится курсор.

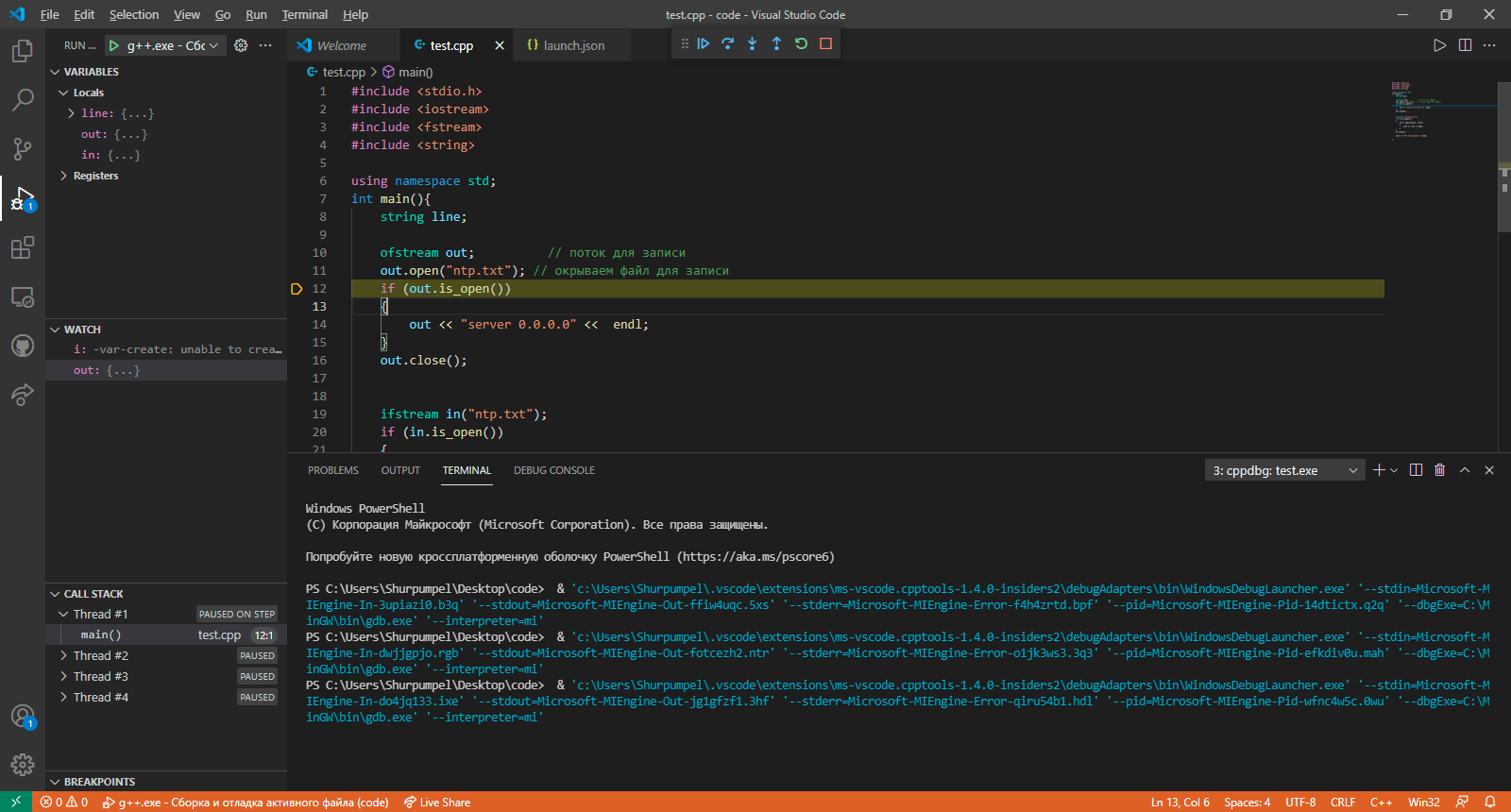


Рисунок 15трассировка в VS Code

# 3.6 Отладка ПМ НКСС

Ошибки, возникающие при разработке программного обеспечения, можно разделить на синтаксические и логические.

Синтаксические – это такие ошибки, которые не позволяют транслятору или интерпретатору однозначно интерпретировать написанный исходный код программы. Иначе говоря, это ошибки в записи конструкций языка программирования. Зачастую такие ошибки вызваны обычными опечатками.

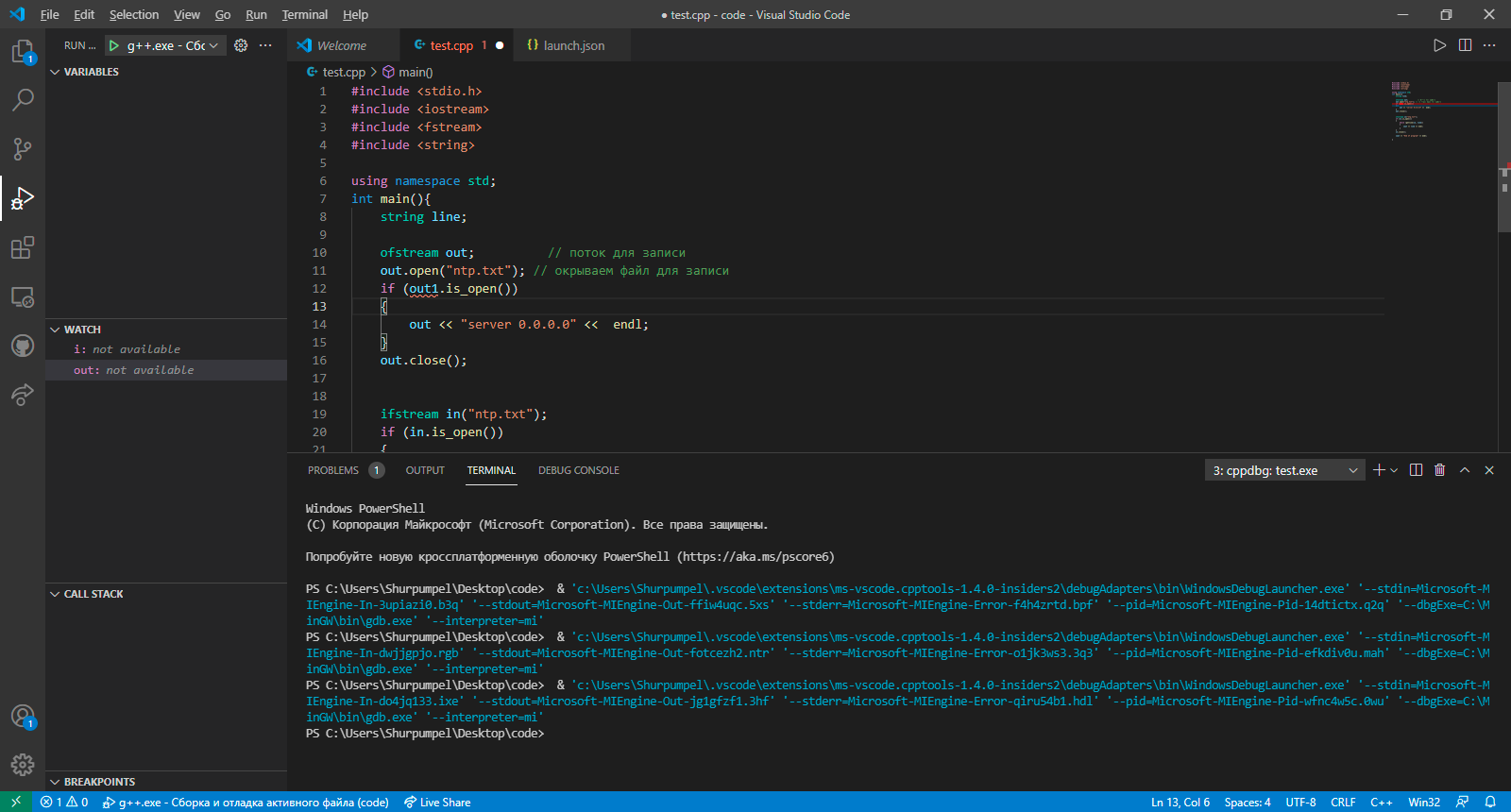
В случае с разработкой ПМ НКССС подобные проблемы выявляет встроенный анализатор среды разработки VS Code еще до трансляции кода и сборки проекта. Среда разработки указывает программисту файл, в котором обнаружена ошибка, ее расположение в коде этого файла, выделяет проблемное место и показывает подробную информацию во всплывающей подсказке при наведении курсора на это место. В качестве проблемных мест, говоря о средах разработки, могут быть следующие ошибки:

* ошибки подключения библиотек;
* ссылки на ненайденные файлы;
* использование необъявленных или неинициализированных переменных;
* ошибки построения циклов, условий и прочих стандартных структур;
* несоответствие типов данных при работе со строгой типизацией.

В случае если данный анализатор что-либо пропустит или если программист проигнорирует его предупреждения, интерпретатор завершит свою работу с ошибкой.

Пример работы синтаксического анализатора в VS Code показан на рисунке ниже.

Таким образом, выявление синтаксических ошибок является тривиальной задачей и не требует подробного рассмотрения.



Логические ошибки – это любые ошибки, которые приводят к неверным результатам работы программы, при этом код может выполняться без сбоев и не приводить к аварийной остановке. Наличие именно таких ошибок в коде выявляется на этапе тестирования программного обеспечения. Для локализации ошибок и нахождения причин их возникновения применяется анализ путей выполнения программы и текущих значений переменных. Данный анализ производится при помощи отладочного вывода и специальных инструментов – отладчиков, которые позволяют пошагово выполнять программу и отслеживать состояние переменных в памяти.

# 3.7 Модульное тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Специализированных утилит для создания Unit-тестов для языка С++ нет, поэтому было принято решение написать небольшой простенький ПМ, который будет проводить данное тестирование. Тесты отслеживают, как ведет себя ПМ НКСС при различных введенных данных. Проверяется поведение ПМ при вводе неправильных данных команды, при вводе правильных данных, при различных данных для команды.

Список проводимых тестов

* Проверка добавления нового сервера
* Проверка вывода статуса настроек NTP
* Проверка вывода ранее введенных настроек NTP
* Проверка вывода введенных NTP-серверов

В результате выполнения ПМ предназначенного для тестирования разрабатываемого ПМ НКСС, все тесты были пройдены

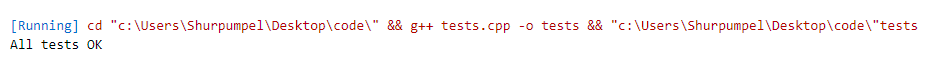


Рисунок 16 результаты Unit-тестирования

# Выводы по разделу

В данном разделе были приведены краткие описания методов тестирования. Из них выбраны наиболее подходящие для обеих частей ПМ НКСС. Описаны используемые для тестирования и отладки инструменты, процесс и результаты.

# Заключение

Данная работа была посвящена разработке программного модуля настройки конфигурации NTP сервиса. В ходе работы были выполнены следующие задачи:

* проведено исследование предметной области;
* проведен обзор существующих решений;
* выбор языка и среды программирования;
* разработана структура данных ПМ НКСС;
* разработан алгоритм работы ПМ НКСС;
* осуществлена программная реализация ПМ НКСС;
* проведены тестирование и отладка ПМ НКСС;
* разработано руководство оператора ПМ НКСС.

Цель разработки достигнута.

# Список литературы:

1. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series).
2. Alessandro Del Sole. Visual Studio Code Succinctly
3. А. Филимонов. Построение мультисервисных сетей Ethernet. — М.: BHV, 2007
4. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. Язык программирования C#. Классика Computers Science. 4-е издание
5. Алекс Макки. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов
6. Барри Берд. Java 9 для чайников
7. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем.
8. Бенджамин Дж. Эванс, Джеймс Гоф, Крис Ньюленд. Java: оптимизация программ. Практические методы повышения производительности приложений в JVM
9. Брюс Эккель. Философия Java
10. Бьёрн Страуструп. Дизайн и эволюция C++ = The Design and Evolution of C++. — СПб.: Питер, 2007. — 445 с.
11. Бьёрн Страуструп. Программирование: принципы и практика использования C++, исправленное издание
12. Бьёрн Страуструп. Программирование: принципы и практика с использованием С++. — 2016.
13. Бьёрн Страуструп. Язык программирования C++
14. Бьёрн Страуструп. Язык программирования C++. Специальное издание
15. Герберт Шилдт. C# 4.0: полное руководство
16. Герберт Шилдт. Java. Полное руководство, 10-е издание
17. Герберт Шилдт. Полный справочник по C++
18. Герберт Шилдт. Теория и практика C++
19. Гленфорд Майерс, Том Баджетт, Кори Сандлер. Искусство тестирования программ, 3-е издание
20. ГОСТ Р ИСО 7498-2-99. ВОС. Базовая эталонная модель. Часть 2. Архитектура защиты информации.
21. ГОСТ Р ИСО 7498-3-97. ВОС. Базовая эталонная модель. Часть 3. Присвоение имён и адресация.
22. ГОСТ Р ИСО/МЭК 7498-1-99. ВОС. Базовая эталонная модель. Часть 1. Базовая модель.
23. ГОСТ Р ИСО/МЭК 7498-4-99. ВОС. Базовая эталонная модель. Часть 4. Основы административного управления.
24. Джеймс Гослинг, Билл Джой, Гай Стил, Гилад Брача, Алекс Бакли. Язык программирования Java SE 8. Подробное описание, 5-е издание
25. Джозеф Албахари, Бен Албахари. C# 6.0. Справочник. Полное описание языка
26. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод = C# in Depth, 3rd ed
27. Джошуа Блох. Java. Эффективное программирование
28. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование.
29. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений
30. Карли Уотсон, Кристиан Нейгел, Якоб Хаммер Педерсен, и др. Visual C# 2008: базовый курс. Visual Studio® 2008
31. Кей С. Хорстманн. Java SE 8. Вводный курс
32. Кей С. Хорстманн. Java SE 9. Базовый курс
33. Кей С. Хорстманн. Java. Библиотека профессионала, том 1. Основы. 10-е издание
34. Кей С. Хорстманн. Java. Библиотека профессионала, том 2. Расширенные средства программирования. 10-е издани
35. Кишори Шаран. Java 9. Полный обзор нововведений
36. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов
37. Кристиан Нейгел, Карли Уотсон и др. Visual C# 2010: полный курс
38. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд
39. Майкл Приват, Роберт Уорнер. Разработка приложений для Mac OS X Lion. Программирование на Objective-C в Xcode
40. Майо Д. Самоучитель Microsoft Visual Studio 2010
41. Миллс, Дэвид Л. Сличение времени в компьютерных сетях. Протокол сетевого времени на Земле и в космосе. — Киев : Wircom, 2011. — С. 464.
42. Монахов Вадим. Язык программирования Java и среда NetBeans. — 3-е изд
43. Мэтт Нойбург. Программирование для iOS 7. Основы Objective-C, Xcode и Cocoa
44. Ник Рендольф, Дэвид Гарднер, Майкл Минутилло, Крис Андерсон. Visual Studio 2010 для профессионалов
45. Пауэрс Л., Снелл М. Microsoft Visual Studio 2008
46. С. Давыдов, А. Ефимов. IntelliJ IDEA. Профессиональное программирование на Java.
47. Семенов Ю.А. Сетевой протокол времени NTP // Telecommunication technologies - Телекоммуникационные технологии. — 2014.
48. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения.
49. Скотт Кнастер, Вакар Малик, Марк Далримпл. Objective-C и программирование для Mac OS X и iOS, 2-е издание
50. Скотт Мейерс. Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14
51. Фрэд Лонг, Дхрув Мохиндра, Роберт С. Сикорд, Дин Ф. Сазерленд, Дэвид Свобода. Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищённых программ
52. Э. Стиллмен, Дж. Грин. Изучаем C#. 2-е издание
53. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание

# Приложение 1. Фрагмент кода программы

CVS\_ID( "$Header: /cvs/cscons/cs\_config/showrun.cpp,v 1.213 2019/12/24 13:59:47 ira Exp $" )

#include <set>

#include <string>

#include <stdio.h>

#include <string.h>

#include <assert.h>

#include "os\_defs.h"

#include "cs\_config.h"

#include "cs\_defs.h"

#include "showrun.h"

#include "cs\_conf\_internal.h"

#include "cs\_defaults.h"

#include "cs\_acl\_vals.h"

#include "cs\_ini\_reader.h"

#include "cs\_logging.h"

#include "cs\_conf\_s.h"

#include "cs\_time\_range.h"

…

static void showPolicyMaps( CsConfig\* c, CsString& str, ShowRunMode mode )

{

    CsPolicyMap m = c->PolicyMapGetFirst();

    CsString   name((CsString\_I\*)NULL);

    CsPolicyMapEntry me(NULL);

    CsPolicyMapEntry default\_me(NULL);

    while(!m.is\_nil())

    {

        str += "policy-map ";

        str += m.GetName();

        str += EOL\_CHARS;

        ShowNonEmptyWithEol(

            " description ",

            m.GetDescription(),

            str

        );

        me = m.GetFirstEntry();

        while( !me.is\_nil() || !default\_me.is\_nil())

        {

            if( me.is\_nil() )

            {

                me = default\_me;

                default\_me = NULL;

            }

            else if( me.GetClassMap().IsDefault() )

            {

                // Default class must be the last in policy map

                default\_me = me;

                me = m.GetNextEntry();

                continue;

            }

            str+=" class ";

            str+=me.GetClassMapName();

            switch(me.GetType())

            {

            case CsPolicyMapEntry\_I::DSCP:

                str+=EOL\_CHARS;

                str+="  set dscp ";

                str+=me.GetPrintableValue();

                break;

            case CsPolicyMapEntry\_I::PRECEDENCE:

                str+=EOL\_CHARS;

                str+="  set precedence ";

                str.AppendUint( me.GetValue() );

                break;

            case CsPolicyMapEntry\_I::NONE:

                assert( 0 );

                break;

            }

            str+=EOL\_CHARS;

            me = m.GetNextEntry();

        }//while( !me.is\_nil() )

        m = c->PolicyMapGetNext();

    }// while(!m.is\_nil())

    ShowLineSeparator( str, mode );

}

static void ShowTimeRanges( CsConfig& cfg, CsString& cfg\_text )

{

    CsTimeRange time\_range = cfg.TimeRangeGetFirst();

    while( ! time\_range.is\_nil() )

    {

        CsAppendTimeRange( time\_range, cfg\_text );

        time\_range = cfg.TimeRangeGetNext();

    }

}

static void ShowObjectGroups( CsConfig& cfg, CsString& cfg\_text )

{

    CsObjectGroup obj\_group = cfg.ObjGroupGetFirst();

    while( ! obj\_group.is\_nil() )

    {

        CsAppendObjectGroup( obj\_group, CS\_SHOW\_RUN, cfg\_text );

        obj\_group = cfg.ObjGroupGetNext();

    }

}

static void showExtConf( CsConfig& cfg, CsString& cfg\_text )

{

    cfg.ShowExtConf(cfg\_text);// TODO check result

}

static void showNtp( CsConfig& cfg, CsString& cfg\_text )

{

    FILE\* file;

    if((file= fopen("/etc/ntp.conf", "r"))==NULL)

    {

        return;

    }

    while (!feof(file))

    {

        char str[500];

        fgets(str, 500, file);

        if(strstr(str, "server ") != NULL && strstr(str, "#") == NULL)

        {

            cfg\_text += "ntp ";

            cfg\_text += str;

            cfg\_text += EOL\_CHARS;

        }

    }

    fclose(file);

}

static CsString ShowRunCommon(

    CsConfig & cfg,

    ShowRunMode mode,

    bool hide\_confidential

)

{

    CsString r;

    if ( SHOW\_RUN\_IMMEDIATE\_CMDS != mode ) { showConfig( &cfg, r, mode ); }

    if ( SHOW\_RUN\_DEFERRED\_CMDS != mode )

        { ShowUsers( cfg, r, hide\_confidential ); }

    if ( SHOW\_RUN\_IMMEDIATE\_CMDS != mode )

    {

        showAAA( cfg, r, mode, hide\_confidential );

        showLocalSettings( cfg, mode, r, hide\_confidential );

        showNS( cfg, mode, r );

        showClassMaps( &cfg, r, mode );

        showPolicyMaps( &cfg, r, mode );

        showAclLogParams( &cfg, r );

        showNtp(cfg, r);

        if ( SHOW\_RUN\_FULL == mode ) { CsAppendLogParams( r ); }

        showIdentity( &cfg, r, mode );

        showIsakmpPolicy( &cfg, r, mode );

        showIsakmpKeys( & cfg, r, mode, hide\_confidential );

        showGlobLocalIpPool( &cfg, r, mode );

        showActivePool( &cfg, r );

        ShowClientIsakmpGroups( &cfg, r, mode );

        showGlobIpsecTransform( &cfg, r, mode );

        ShowObjectGroups( cfg, r );

        showGlobAccessList( &cfg, r, mode );

        ShowPortMaps( &cfg, r, mode );

        ShowInspectSettings( &cfg, r );

        ShowInspects( &cfg, r, mode );

        showCryptoMapList( cfg, true, r, mode, hide\_confidential );

        showCryptoMapList( cfg, false, r, mode, hide\_confidential );

        showInterface( &cfg, r, mode );

        showSNMPParams( &cfg, r, mode );

        showRouteEntries( cfg, r, mode );

        showGlobCATrustPoint( &cfg, r, mode );

        showIsakmpPeers( & cfg, r, mode, hide\_confidential );

        ShowTimeRanges( cfg, r );

        showExtConf( cfg, r );

    }

    r += "end" EOL\_CHARS;

    return r;

}

CsString ShowRun( CsConfig& cfg, ShowRunMode mode )

{

    return ShowRunCommon( cfg, mode, false );

}

CsString ShowRunHideConfidential( CsConfig& cfg, ShowRunMode mode )

{

    return ShowRunCommon( cfg, mode, true );

}

CVS\_ID( "$Header: /cvs/cscons/cs\_config/cs\_cmdexec.cpp,v 1.129 2021/03/15 10:01:40 tima Exp $" )

#include <fstream>

#include <stdio.h>

#include "os\_defs.h"

#include "os\_defs\_ex.h"

#include "cs\_conf\_s.h"

#include "cs\_route\_s.h"

#include "cs\_config.h"

#include "cs\_converter.h"

#include "cs\_cmdlist.h"

#include "cs\_utils.h"

#include "showrun.h"

#include "cmn\_licm.h"

#include "vers.h"

#include "file\_name\_consts.h"

#include "cmn\_ini.h"

#include "sysspec.h"

#include "cs\_call\_exec.h"

#include "filefunc.h"

#include "cs\_cmds.h"

#include "cs\_defs.h"

#include "cs\_sa\_control.h"

#include "cs\_cfg\_process.h"

#include "cs\_file\_prefix.h"

#include "cs\_text\_helpers.h"

#define STERRA\_CRYPTO\_PROVIDER\_SUFFIX     "st"

#define STERRA\_XP\_CRYPTO\_PROVIDER\_SUFFIX  "xp"

#define CRYPTOPRO\_CRYPTO\_PROVIDER\_SUFFIX  "cp"

//////////////////////////////////////////////////////////////////////////

//  function declaration

static CMN\_STATUS ObtainSerialNumber( CsString\* p\_sn /\* OUT \*/ )

{

    char sn[MAX\_SN\_BUF];

    size\_t buf\_len = MAX\_SN\_BUF;

    if ( SysSpec::RESULT\_OK != SysSpec::ObtainSerialNumber( & buf\_len, sn ) )

    {

        (\*p\_sn) = SERIAL\_NUM\_UNKNOWN;

        return STATUS\_FAIL;

    }

    (\*p\_sn) = sn;

    return STATUS\_OK;

}

static CMN\_STATUS ObtainHwRevision( CsString\* p\_hw\_rev /\* OUT \*/ )

{

    char hv\_rev[MAX\_HW\_REVISION\_BUF];

    size\_t buf\_len = MAX\_HW\_REVISION\_BUF;

    if ( SysSpec::RESULT\_OK != SysSpec::ObtainHwRevision( & buf\_len, hv\_rev ) )

    {

        (\*p\_hw\_rev) = HW\_REVISION\_UNKNOWN;

        return STATUS\_FAIL;

    }

    (\*p\_hw\_rev) = hv\_rev;

    return STATUS\_OK;

}

//////////////////////////////////////////////////////////////////////////

// class CsCmdExecuter

CsCmdExecuter\_skl::CsCmdExecuter\_skl()

{

    m\_LastResult = (const char\*)NULL;

}

CsCmdExecuter\_skl::~CsCmdExecuter\_skl()

{

    m\_LastResult = (const char\*)NULL;

}

CMN\_STATUS CsCmdExecuter\_skl::ParseCmd( const cs\_cmdlist\_t\* cmd )

{

    CMN\_STATUS status = STATUS\_FAIL;

    if ( NULL == cmd ) { return status; }

    bool is\_config\_mode = false;

    const cs\_cmdlist\_t\* cur = cmd;

    if( 0 == strcmp( cur->pCmdName, "do" ) )

    {

        is\_config\_mode = true;

        cur = cur->next;

    }

    if( csCmpCmd( &cur, csCmdShowRun ) ||

        csCmpCmd( &cur, csCmdWriteTerminal )

      )

    {

        status = ProcessShowRun( cur );

    }

    else if( csCmpCmd( &cur, csCmdShowLM ) )

    {

        status = ProcessShowLM( cur );

    }

    else if( csCmpCmd( &cur, csCmdSystem ) )

    {

        status = ProcessSystem( cur );

    }

    else if( csCmpCmd( &cur, csCmdRun ) )

    {

        status = ProcessRun( cur );

    }

    else if( csCmpCmd( &cur, csCmdShowVersionCSP ) )

    {

        ProcessShowVersionCSP();

        status = STATUS\_OK;

    }

    else if( csCmpCmd( &cur, csCmdShowVersion ) )

    {

        status = ProcessShowVersion( cur );

    }

    else if( csCmpCmd( &cur, csCmdShowCryptoIsakmpPolicy ) )

    {

        status = ProcessShowCryptoIsakmpPolicy();

    }

    else if ( csCmpCmd( &cur, csCmdShowIpRoute ) )

    {

        status = ProcessShowIpRoute();

    }

    else if ( csCmpCmd( &cur, csCmdShowPrivilege ) )

    {

        ProcessShowPrivilege();

        status = STATUS\_OK;

    }

    else if ( csCmpCmd( &cur, csCmdShowTerminal ) )

    {

        status = ProcessShowTerminal();

    }

    else if ( csCmpCmd( &cur, csCmdShowClock ) )

    {

        status = ProcessShowClock();

    }

    else if ( csCmpCmd( &cur, csCmdShowSubst ) )

    {

        status = ProcessShowSubst( cur );

    }

    else if ( csCmpCmd( &cur, csCmdShowSerialNumber ) )

    {

        status = ObtainSerialNumber( & m\_LastResult );

    }

    else if ( csCmpCmd( &cur, csCmdShowHwRevision ) )

    {

        status = ObtainHwRevision( & m\_LastResult );

    }

    else if ( csCmpCmd( &cur, csCmdClearCrypto ) )

    {

        status = ProcessClearCrypto( cur );

    }

    else if ( csCmpCmd( &cur, csCmdCopyFromRunningCfg ) )

    {

        status = ProcessCopyFromRunningCfg( cur );

    }

    else if ( csCmpCmd( &cur, csCmdCopyTermInput ) )

    {

        status = ProcessCopyTermInput( cur );

    }

    else if ( csCmpCmd( &cur, csCmdCopy ) )

    {

        status = ProcessCopy( cur );

    }

    else if ( csCmpCmd( &cur, csCmdClearRunningCfg ) )

    {

        status = CsClearRunningCfg( is\_config\_mode, m\_LastResult );

    }

    else if (   csCmpCmd( &cur, csCmdWriteErase ) ||

                csCmpCmd( &cur, csCmdEraseStartupCfg )

            )

    {

        m\_LastResult = MSG\_USE\_CLEAR\_RUNNING\_CFG;

        status = STATUS\_FAIL;

    }

    else if ( csCmpCmd( &cur, csCmdMore ) )

    {

        status = ProcessMore( cur );

    }

    else if ( csCmpCmd( &cur, csCmdDir ) )

    {

        status = ProcessDir( cur );

    }

    else if ( csCmpCmd( &cur, csCmdDelete ) )

    {

        status = ProcessDelete( cur );

    }

    else if ( csCmpCmd( &cur, csCmdSyncNetIfs ) )

    {

        status = CallSyncNetIfs( is\_config\_mode );

    }

    else if( csCmpCmd( &cur, csCmdShowCryptoIsakmpSa ) )

    {

        status = ProcessShowSa( "isakmp", cur );

    }

    else if( csCmpCmd( &cur, csCmdShowCryptoIpsecSa ) )

    {

        status = ProcessShowSa( "ipsec", cur );

    }

    else if (   csCmpCmd( & cur, csCmdShowACLs ) ||

                csCmpCmd( & cur, csCmdShowIpACLs )

            )

    {

        status = ProcessShowACLs( cur );

    }

    else if ( csCmpCmd( & cur, csCmdShowObjectGroup ) )

    {

        status = ProcessShowObjectGroup( cur );

    }

    else if ( csCmpCmd( & cur, csCmdShowInterfaces ) )

    {

        status = ProcessShowInterfaces( cur );

    }

    else if ( csCmpCmd( & cur, csCmdShowIpInterface ) )

    {

        status = ProcessShowIpInterface( cur );

    }

    else if ( csCmpCmd( &cur, csCmdNTPStatus) )

    {

        status = ProcessShowNTPStatus();

    }else if ( csCmpCmd( &cur, csCmdNTPAssociations) )

    {

        status = ProcessShowNTPAssociations();

    }

    return status;

}

CMN\_STATUS CsCmdExecuter\_skl::ProcessShowNTPStatus()

{

    system("ntpq -n -c sysinfo -c sysstats");

    CMN\_STATUS r = STATUS\_OK;

    return r;

}

CMN\_STATUS CsCmdExecuter\_skl::ProcessShowNTPAssociations()

{

    system("ntpq -p");

    CMN\_STATUS r = STATUS\_OK;

    return r;

}

}

CVS\_ID( "$Header: /cvs/cscons/cs\_config/cs\_config.cpp,v 1.323 2021/03/11 08:12:02 amuravyev Exp $" )

#include <stdio.h>

#include <ctype.h>

#include <assert.h>

#include <time.h>

#ifdef WIN32

    #include <io.h> // for isatty()

    #include <conio.h>

#endif

#ifdef UNIX

    #include <unistd.h> //for gethostname

    #include <stdlib.h>

    #include <netinet/in.h>

    #include <arpa/nameser.h>

    #include <resolv.h>

#endif

#include "os\_defs.h"

#include "cs\_config.h"

const CsConfig\_skl::CmdItem CsConfig\_skl::m\_cmdArrRoot[] =

{

    { "crypto", m\_cmdArrCrypto, 0 },

    { "ip", m\_cmdArrIp, 0 },

    { "exit", NULL, &CsConfig\_skl::ProcessEnd },

    { "end", NULL, &CsConfig\_skl::ProcessEnd },

    { "snmp-server", m\_cmdArrSnmpServer, &CsConfig\_skl::ProcessSNMP },

    { "access-list", NULL, &CsConfig\_skl::ProcessAccessList },

    { "logging", NULL, &CsConfig\_skl::ProcessLogging },

    { "enable", NULL, &CsConfig\_skl::ProcessEnablePassword },

    { "username", NULL, &CsConfig\_skl::ProcessUser },

    { "hostname", NULL, &CsConfig\_skl::ProcessHostname },

    { "interface", m\_cmdArrInterface, &CsConfig\_skl::ProcessInterface },

    { "class-map", NULL, &CsConfig\_skl::ProcessClassMap },

    { "policy-map", NULL, &CsConfig\_skl::ProcessPolicyMap },

    { "time-range", NULL, &CsConfig\_skl::ProcessTimeRange },

    { "object-group", NULL, &CsConfig\_skl::ProcessObjGroup },

    { "radius-server", m\_cmdArrRadiusServer, 0 },

    { "aaa", m\_cmdArrAaa, 0 },

    { "ntp", m\_cmdArrNtpConfiguration, 0 },

    { NULL, NULL, 0 }

};

…

//"ntp ..."

const CsConfig\_skl::CmdItem CsConfig\_skl::m\_cmdArrNtpConfiguration[] =

{

    { "server", NULL, &CsConfig\_skl::ProcessNtpServer },

    { NULL, NULL, 0 }

};

CsConfig\_skl::CsConfig\_skl()

:   m\_CurrentObject( NULL ),

    m\_CurrentState( CS\_STATE\_INITIAL ),

    m\_dfBit( DF\_BIT\_COPY ),

    m\_Identity( IDENT\_ADDRESS ),

    m\_KeepAliveTime( 0 ),

    m\_KeepAliveRetry( 0 ),

    m\_keepAliveRetryCount( 0 ),

    m\_ipsecSaLifetimeSec( IPSEC\_SA\_LIFETIME\_SEC\_DEFAULT ),

    m\_ipsecSaLifetimeKb( IPSEC\_SA\_LIFETIME\_KB\_DEFAULT ),

    m\_LoadMessage( (CsString\_I\*)NULL ),

    m\_LEditor( NULL ),

    m\_cfgActionRequired( 0 ),

    m\_isakmpFragmentation( false ),

    m\_radiusServerIp( NULL ),

    m\_radiusServerAuthPort( DEFAULT\_RADIUS\_SERVER\_AUTH\_PORT ),

    m\_radiusServerAcctPort( DEFAULT\_RADIUS\_SERVER\_ACCT\_PORT ),

    m\_radiusRetransmit( RADIUS\_RETRANSMIT\_DEFAULT ),

    m\_radiusTimeout( RADIUS\_TIMEOUT\_DEFAULT ),

    m\_accountingUpdatePeriod( 0 ),

    m\_ikeSessionTimeMax( DEFAULT\_IKE\_SESSION\_TIME\_MAX ),

    m\_ikeSaLifetimeDelta( 0 ),

    m\_ikeInitiatorSessionsMax( DEFAULT\_IKE\_INITIATOR\_SESSIONS\_MAX ),

    m\_ikeResponderSessionsMax( DEFAULT\_IKE\_RESPONDER\_SESSIONS\_MAX )

{

    m\_numericVals[NUM\_VAL\_ACL\_LOG\_UPDATE\_THRESHOLD].m\_defaultValue = 0;

    m\_numericVals[NUM\_VAL\_ACL\_LOGGING\_INTERVAL].m\_defaultValue = 0;

    for( uint32\_t i = 0; i < NUM\_VAL\_COUNT; ++i )

    {

        m\_numericVals[i].m\_value = m\_numericVals[i].m\_defaultValue;

        m\_numericVals[i].m\_isSet = false;

    }

}

CsConfig\_skl::~CsConfig\_skl()

{

}

…

CMN\_STATUS CsConfig\_skl::ProcessNtpServer( const cs\_cmdlist\_t\* cmd )

{

    assert( NULL != cmd );

    CMN\_STATUS status = STATUS\_FAIL;

    const cs\_cmdlist\_t\* cur = cmd;

    CsIpAddr addr;

    cur = cur->next;

    addr.SetStrIp( cur->pCmdName );

    CsString stringForNtpConf = "server ";

        stringForNtpConf += addr.GetStrIp();

        FILE\* file;

    if(NEGATIVE\_CMD\_PREFIX == cmd->cmdPrefix)

    {

        if((file= fopen("/etc/ntp.conf", "r"))==NULL){

            return STATUS\_FAIL;

        }

        CsString readString = "";

        while (!feof(file))

        {

            char str[500];

            fgets(str, 500, file);

            if(strstr(str, stringForNtpConf.GetStr())!=NULL && strstr(str, "#") == NULL){

                continue;

            }

            readString+=str;

        }

        fclose(file);

        if((file= fopen("/etc/ntp.conf", "w"))==NULL){

            return STATUS\_FAIL;

        }

        fputs(readString.GetStr(), file);

        fclose(file);

        status = STATUS\_OK;

    }else

    {

        if((file= fopen("/etc/ntp.conf", "ab"))==NULL){

            return status;

        }

        fputs(stringForNtpConf.GetStr(), file);

        fputs(EOL\_CHARS, file);

        fclose(file);

        system("systemctl start ntp.service");

        status = STATUS\_OK;

    }

    system("systemctl restart ntp.service");

    return status;

}

<Command

name="ntp"

><Subcommands

><Command

description="Show ntp status"

name="status"

></Command

><Command

description="Show ntp associations"

name="associations"

></Command

></Subcommands

></Command

><Command

description="IP information"

name="ip"

><Subcommands

><Command

description="IP routing table"

name="route"

></Command

>

# Приложение 2. Код написанных тестов

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <cassert>

#include "test.h"

using namespace std;

void TestShowRun();

void TestNTPServer();

void TestShowNTPStatus();

void TestShowNTPAssociations();

bool FindString(string ss);

int main()

{

    TestShowRun();

    TestNTPServer();

    TestShowNTPStatus();

    TestShowNTPAssociations();

    cout << "All tests OK" << endl;

    return 0;

}

void TestShowRun()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    system("show run | include ntp");

    bool ff = strcpy(m\_LastResult, "ntp server 0.0.0.0");

    assert(ff == true);

    system("ntp server 0.0.0.0.0");

    system("show run | include ntp");

    bool ff = strcpy(m\_LastResult, "");

    assert(ff == true);

}

void TestNTPServer()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    bool ff = FindString("server 0.0.0.0");

    assert(ff == true);

    system("no ntp server 0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

    system("ntp server 0.0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

    system("ntp server 0.0.0.0");

    system("no ntp server 0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

}

void TestShowNTPAssociations()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    bool ff = false;

    char\* a = strstr(m\_LastResult,"0.0.0.0");

    if(a != NULL)

    {

        ff = true;

    }

    assert(ff == true);

}

bool FindString(string ss)

{

    ifstream in("ntp.conf");

    string line;

    bool isThisStringInConfig = false;

    if (in.is\_open())

    {

        while (getline(in, line))

        {

            char str[line.length()+1];

            strcpy(str, line.c\_str());

            char str1[line.length()+1];

            strcpy(str1, line.c\_str());

            if (strcmp(str1, str)==0)

            {

                isThisStringInConfig = true;

            }

        }

    }

    in.close();

    return isThisStringInConfig;

}