МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.03 –	Информационная безопасность автоматизированных систем
Специализация –	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ТРЕНАЖЕР ПО ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ НА БАЗЕ КОНТЕЙНЕРОВ

Студент _____ Нагаев Михаил Михайлович

Руководитель _____ Лупанов М.Ю.

Нормоконтролёр _____ Фатеев А.Г.

Рецензент _____ Старкин А.В.

Работа допущена к защите (протокол заседания кафедры от 26.06.21г. № 16)

Заведующий кафедрой _____ С.Л. Зефиров

Работа защищена с отметкой отлично (протокол заседания ГЭК от 28.06.21г № 8)

Секретарь ГЭК _____ А.П. Иванов

Пенза 2021

Оглавление

Введение 3
1 Анализ используемых средств моделирования информационных систем . 5
2 Выбор средств моделирования информационных систем для обучения информационной безопасности 10
3 Разработка тренажера 12
3.1 Разработка контейнеров для тренажера по информационной безопасности 12
3.2 Разработка подсистемы хранения, доставки и развертывания контейнеров 22
3.3 Разработка подсистемы запуска контейнеров и формирования задания 24
3.4 Разработка подсистемы автоматического контроля выполнения заданий 30
3.5 Разработка пользовательского интерфейса тренажера 32
3.6 Разработка пользовательского интерфейса программы тьютора 41
3.7 Установка и настройка тренажера 45
4 Тестирование тренажера 49
4.1 Тестовый план для программы «Тренажер ИБОС» 49
4.2 Тестовые сценарии 51
4.3 Результаты тестирования 57
4.4 Заключение по результатам тестирования 59
5 Вредные психофизические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие 60
Заключение 64
Список используемых источников 65

Введение

В современном мире информационные технологии интенсивно развиваются, что ведет к компьютеризации любой сферы жизнедеятельности людей, организаций и государства. Но внедрение электронно-вычислительной техники во все аспекты жизни несет не только улучшения, но и открывает множество возможностей для нарушителей, которые с помощью специализированного оборудования и программного обеспечения могут прервать информационные процессы, которые поддерживают и обеспечивают деятельность организации, или получить конфиденциальные сведения, циркулирующие в информационных системах, что также негативно скажется на работе организации. В связи с этим проблема информационной безопасности становится все более актуальной.

Для решения данной проблемы необходимы специалисты по информационной безопасности, которые могут предотвратить несанкционированный доступ к информационной системе, а также использование, раскрытие, изменение или уничтожение информации. Но при этом появляется новая проблема – проблема обучения информационной безопасности.

Для полноценного обучения информационной безопасности нужна не только теоретическая база, но и практическая. Но при этом возникает множество трудностей, связанных с получением практических навыков и умений. Именно поэтому необходимо средство, позволяющее эффективно получать соответствующие знания и опыт. Таким средством может стать программный тренажер, который, предоставляя все необходимые условия для практики, обеспечит возможности получения и закрепления практических навыков по обеспечению информационной безопасности.

Исходя из вышеизложенного нельзя не отметить актуальность темы выпускной квалификационной работы «Тренажер по информационной

безопасности на базе контейнеров», так как использование контейнерной виртуализации позволяет ускорить процесс развертывания информационных систем, сохранить высокую производительность учебных стендов, автоматизировать управление и оценивание работ при закреплении практических навыков будущего специалиста по информационной безопасности. Всё это позволяет многократно увеличить эффективность учебного процесса.

Таким образом, целью работы является разработка тренажера по информационной безопасности на базе контейнеров.

1 Анализ используемых средств моделирования информационных систем

В широком смысле, моделью системы называют представление системы в некоторой форме, отличной от реального существования[1]. Модель информационной системы (ИС) должна повторять исходную, реальную ИС по своим характеристикам и качествам. Исходя из специфики обучения информационной безопасности (ИБ), модели ИС должны быть:

— целенаправленными;

— полными, с точки зрения решаемой задачи (получаемых навыков);

— удобными в использовании;

— адаптивными, то есть должны присутствовать возможности модификации;

— адекватными, то есть модели должны полностью, в соответствии с поставленной задачей, повторять реальную ИС;

— управляемыми, то есть должна присутствовать возможность централизованного управления ИС.

Первый способ моделирования основан на построении полнофункционального макета ИС. Данный подход предполагает создание ИС «в натуральную величину»: развертывание аппаратных средств, объединение всех аппаратных средств линиями связи, развертывание телекоммуникационных комплектующих, развертывание системного и прикладного программного обеспечения, установку аппаратных и программных средств защиты информации.

Такая модель обеспечивает получение практических навыков специалистом в полном объеме, так как обучение производиться на точной копии реальной ИС.

Но такая модель имеет ряд недостатков. Прежде всего построение полнофункциональных макетов ИС требует больших финансовых затрат, в виду стоимости на аппаратные и программные средства, а также площади, которая

будет занята макетом ИС. Также на развертывание и обслуживание всех необходимых средств нужны временные и людские ресурсы, что также осложняет использование такой модели. Кроме того, для данной модели требования адаптивности и управляемости будут выполнены не в полной мере. Модификация полнофункциональных макетов возможна, но при этом для таких модификаций нужно большое количество времени, что недопустимо при проведении учебного процесса. А управлять несколькими ИС, которые построены на базе полнофункциональных макетов, также затруднительно, так как такие макеты должны быть изолированными.

Дальнейшие способы моделирования основаны на предоставлении вычислительных ресурсов и их логического объединения изолированным процессам, которые выполняются на одном физическом устройстве, то есть дальнейшие способы моделирования предполагают использование технологии виртуализации[2].

Способы моделирования, связанные с виртуализацией, имеют важное преимущество: различные ИС могут быть развернуты всего лишь на одном физическом ресурсе (на одном компьютере). Данная особенность значительно упрощает процесс подготовки ИС, а также позволяет сэкономить как финансовые, так и временные ресурсы.

Второй способ моделирования основан на использовании виртуальных машин с эмуляцией оборудования[3]. Виртуальной машиной называют программную или аппаратную систему, которая эмулирует некоторую платформу, создавая на ней изолированные операционные системы[4]. Важной особенностью такого способа виртуализации является то, что кроме операционной системы с программным обеспечением, также эмулируются все необходимые аппаратные средства и соединения (BIOS, оперативная память, жесткий диск и другие периферийные устройства). На рисунке 1 представлена схема работы виртуальных машин с эмуляцией оборудования.

Рисунок 1 – Схема работы виртуальных машин с эмуляцией оборудования

Данный подход позволяет построить ИС, приближенную к реальной, на одном физическом устройстве. Причем все подсистемы (операционные системы) будут изолированы друг от друга, а также от хост-системы на аппаратном уровне. Поэтому использование виртуальных машин с эмуляцией оборудования обеспечивает получение практических навыков специалистом в полном объеме, так как развертывается виртуальная ИС, приближенная к реальной.

Но данный способ не лишен недостатков, так как данный подход к моделированию является достаточно ресурсозатратным. Для реализации нескольких виртуальных машин такого вида необходимо большое количество оперативной и долговременной памяти, а также вычислительных ресурсов, так как на эмуляцию всего лишь аппаратных частей может уйти большая часть выделенных вычислительных средств. Такой способ моделирования не удовлетворяет лишь требованию управляемости, так как отсутствуют возможности централизованной настройки нескольких систем.

Третий способ моделирования основан на использование виртуальных машин с полной виртуализацией[3], использующих аппаратную виртуализацию, если аппаратные комплектующие рабочих станций поддерживают такой вид виртуализации (наиболее распространены технологии Intel-VT или AMD-VT), или программную виртуализацию в ином случае[5]. Данный способ имеет сходства с предыдущим, но главным отличием является наличие «посредника» между гостевыми операционными системами и хост системой. Данным «посредником» является гипервизор. Именно данная программа или аппаратная

схема (в случае аппаратной виртуализации) позволяет одновременно и параллельно выполнять несколько операционных систем. Гипервизор обеспечивает изоляцию и разделение ресурсов между системами, а также средства связи и взаимодействия между ними. Схема работы виртуальных машин с полной виртуализацией представлена на рисунке 2.

Рисунок 2 – Схема работы виртуальных машин с полной виртуализацией

Как и предыдущий способ моделирования, данный подход позволяет изолировать друг от друга и от хост-системы гостевые операционные системы, а также настроить между ними связи с помощью гипервизора, что позволяет строить ИС, приближенные к реальным. Поэтому данный способ моделирования обеспечивает получение практических навыков специалистом в полном объеме.

Также важной особенностью данного способа моделирования является ограничение, состоящее в том, что гостевые операционные системы должны поддерживать аппаратные средства хост-системы. При этом производительность таких виртуальных машин выше виртуальных машин с эмуляцией оборудования, но она не равняется производительности хост-системы, так как вычислительные ресурсы используются гипервизором. Данный способ моделирования удовлетворяет всем требованиям, описанным выше.

Четвертый способ моделирования основан на контейнерной виртуализации[3]. Контейнеризация – это способ виртуализации на уровне операционной системы, при котором создается и поддерживается несколько изолированных контейнеров (экземпляров) пространства пользователя[6]. Такие контейнеры полностью повторяют отдельный экземпляр операционной системы.

В отличии от предыдущих способов виртуализации, в которых могли быть запущены различные операционные системы, контейнеры могут быть запущены только с ядром хост-системы. Все контейнеры используют одно общее ядро, что повышает производительность такой системы, так как нет расходов вычислительных ресурсов на виртуализацию аппаратных средств и системного программного обеспечения гостевых операционных систем. Схема работы контейнерной виртуализации представлена на рисунке 3.

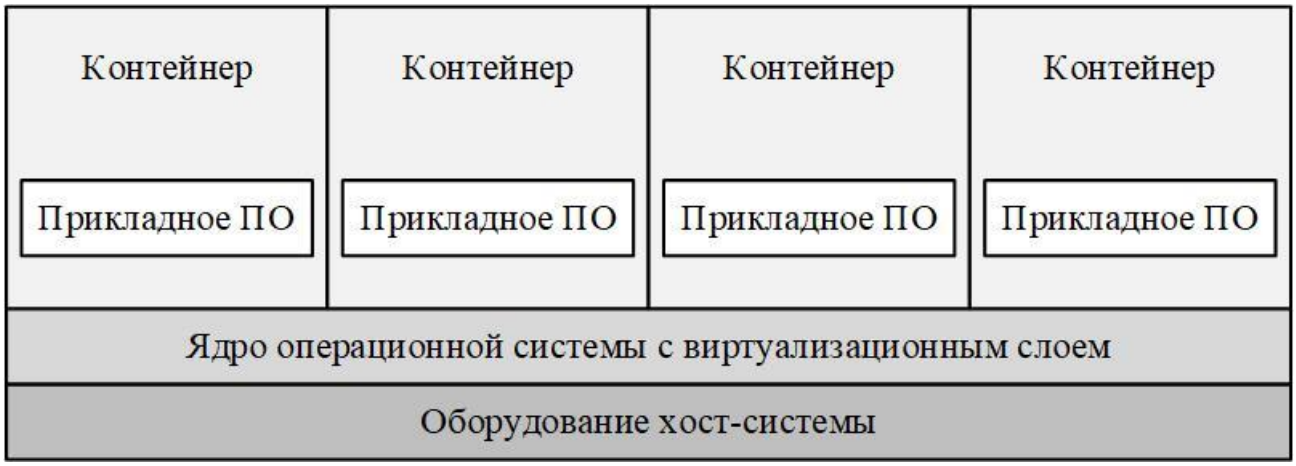

Рисунок 3 – Схема работы контейнерной виртуализации

Данный подход также позволяет построить ИС, приближенную к реальной, на одном физическом устройстве. Причем все контейнеры будут изолированы друг от друга, а также от хост-системы на уровне операционной системы. Поэтому использование контейнерной виртуализации обеспечивает получение практических навыков специалистом в полном объеме, так как развертывается ИС, приближенная к реальной.

Но у данного способа также есть свои недостатки. Данный вид виртуализации имеет сложный механизм настройки, состоящий из множества инструментов. Но при этом данный способ моделирования удовлетворяет всем требованиям, описанным выше, кроме того такой подход обеспечивает наиболее быстрое развертывание системы и автоматическую конфигурацию.

2 Выбор средств моделирования информационных систем для обучения информационной безопасности

Исходя из анализа используемых средств моделирования ИС, имеются следующие подходы при моделировании ИС:

— построение полнофункционального макета информационной системы (ПМ ИС);

— построение информационной системы на базе виртуальных машин с эмуляцией оборудования (ВМ с ЭО);

— построение информационной системы на базе виртуальных машин с полной виртуализацией (ВМ с ПВ);

— построение информационной системы на базе контейнерной виртуализации (КВ).

Исходя из общих особенностей обучения в современном мире и обучения ИБ, средства моделирования должны удовлетворять следующему перечню требований:

— приемлемая стоимость необходимого оборудования;

— минимальные временные затраты на развертывание ИС;

— высокая производительность моделируемой ИС;

— возможность транспортировки заданной среды;

— автоматизация процесса управления ИС:

— автоматизация процесса оценивания результатов работы в ИС;

— защита от ошибок.

На основе заданных требований и будет осуществлен выбор средств моделирования ИС. Соответственно в таблице 1 представлено сопоставление возможных подходов к моделированию ИС и выполнение ими необходимых требований.

Таблица 1 – Сопоставление подходов к моделированию и требований

Требования	ПМ ИС	ВМ с ЭО	ВМ с ПВ	КВ
Стоимость	Высокая	Низкая	Низкая	Низкая
Временные затраты на развертывание и последующее использование	Высокие	Умеренные	Умеренные	Низкие
Снижение производительности	Нет	Значительное	Незначительное	Нет
Транспортировка заданной среды	Нет	Нет	Есть	Есть
Автоматизация управления	Нет	Нет	Частично	Есть
Автоматизация оценивания	Нет	Нет	Нет	Есть
Защита от ошибок	Нет	Есть	Есть	Есть

Из таблицы 1 следует, что наилучшим решением для моделирования ИС является контейнерная виртуализация, так как она покрывает все необходимые требования к средствам моделирования.

Контейнерная виртуализация, действительно, является наилучшим решением при выборе средств моделирования ИС, так как она обеспечивают изоляцию контейнеров друг от друга, время загрузки (развертывания) контейнеров с необходимым программным обеспечением составляет секунды, готовые контейнеры имеют небольшой информационной объем (до сотен Мб), имеются механизмы централизованного управления. Кроме того, именно данный способ виртуализации из представленных является наиболее производительным.

Использование полнофункциональных макетов ИС не является целесообразным из-за большой стоимости всех комплектующих ИС. Использование виртуальных машин с эмуляцией оборудования или с полной виртуализацией имеет место быть, но они перед виртуальными контейнерами имеют ряд недостатков, таких как долгое время загрузки и создания виртуальных машин, большой занимаемый информационный объем (десятки Гб), а также им необходимо большее количество вычислительных ресурсов.

3 Разработка тренажера

3.1 Разработка контейнеров для тренажера по информационной безопасности

В связи с тем, что темы «Управление сетевыми интерфейсами в операционной системе» и «Управление сетевым экраном в операционной системе» предполагают получение практических навыков по работе с сетевой подсистемой операционной системы, было принято решение реализовать обе темы в одном контейнере. Данная работа получала название «Управление сетевыми интерфейсами и сетевым экраном в операционной системе».

Таким образом, необходимо разработать 6 контейнеров для тренажера по следующим темам:

— управление пользователями и группами в операционной системе;

— разграничение прав доступа к локальным ресурсам средствами операционной системы;

— управление системными сервисами операционной системы;

— управление сетевыми интерфейсами и сетевым экраном в операционной системе;

— работа с системными журналами в операционной системе;

— настройка удаленного управления на базе сетевого протокола ssh.

Для каждой темы было продумано содержание.

Содержание работы по теме «Управление пользователями и группами в операционной системе»:

— создание пользователей с их описанием;

— создание групп;

— добавление пользователей в групп.

Содержание работы по теме «Разграничение прав доступа к локальным ресурсам средствами операционной системы»:

— добавление пользователей в группы;

— назначение прав доступа в соответствии с матрицей доступа;

— назначение владельцев и групп;

— проверка назначенных прав;

— назначение дополнительных прав в соответствии с матрицей доступа (POSIX ACL);

— проверка назначенных прав (POSIX ACL).

Содержание работы по теме «Управление системными сервисами операционной системы»:

— проверка статуса служб systemd;

— обзор состояния всей системы systemd;

— работа со службами systemd;

— создание своей службы (юнит .service);

— создание своего юнита .timer;

— создание своего юнита .path.

В данной работе следует учесть, что создаваемые службы должны запускать уже заранее заготовленные программы. Поэтому для данной работы необходимо составить три программы для каждого из трех юнитов: .service, .timer и .path.

Содержание работы по теме «Управление сетевыми интерфейсами и межсетевым экраном в операционной системе»:

— управление IP-адресами, интерфейсами и таблицей маршрутизации.

— добавление таблицы и цепочек межсетевого экрана;

— добавление правил фильтрации и логирования трафика в файл;

— проверка настроенного межсетевого экрана;

— удаление элементов межсетевого экрана;

— настройка MAC-адресов.

Содержание работы по теме «Работа с системными журналами в операционной системе»:

— работа с системными журналами;

— анализ журнала auth.log;

— анализ журнала apt/history.log;

— управление журналами systemd – journald;

— проверка размера и удаление журнала journald;

— исправление ошибок сервисов systemd по журналам journald.

В данной работе для последнего пункта содержания следует учесть, что заранее должны быть подготовлены программы и службы, которые некорректно выполняются в системе, и ошибки которых следует исправить.

Содержание работы по теме «Настройка удаленного управления на базе сетевого протокола ssh»:

— подключение по ssh к удаленной системе;

— настройка аутентификации по ключам для пользователей;

— копирование файлов на удаленную систему по scp;

— закрытие доступа по паролю и запрет доступа root.

При разработке были учтены возможности оптимизации образов контейнеров. Она заключается в уменьшении суммарного информационного размера всех Docker-образов контейнеров. Данная оптимизация заключается в том, что любой Docker-образ состоит из серии слоев. Слой – это «промежуточный» образ. Любая команда, указываемая в Dockerfile вызывает изменение предыдущего слоя, тем самым создавая новый слой. Таким образом, если есть несколько Docker-образов, состоящих из нескольких одинаковых слоев, то такие слои не хранятся в хост-системе отдельно для каждого Docker-образа. Данные слои хранятся в одном экземпляре, а при необходимости используются любым Docker-образом, который состоит из этих слоев.[8]

Исходя из этого, было принято решение о разработке новых контейнеров на основе предыдущих. То есть каждый последующий Docker-образ базируется на основе предыдущего Docker-образа, но при этом добавляются новые слои, которые конфигурируют в системе необходимые настройки, а также устанавливают необходимые программы. Такая концепция позволяет

сэкономить на размере всех Docker-образов, так как размер всех 6 Docker-образов будет определяться размером только последнего Docker-образа.

В качестве первого базового Docker-образа был выбран образ «debian:latest» , который предоставляется официальным реестром Docker Hub. Именно на его основе были разработаны Docker-образы контейнеров

Далее был проведен анализ необходимых дополнительных программ и настроек в контейнерах по каждой из тем.

Для контейнера, который обеспечивает получение навыков по теме «Управление пользователями и группами в операционной системе», достаточно базового образа «debian:latest». Но для удобства редактирования и просмотра текстовых файлов было решено добавить программу «nano». В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian1».

Содержание Dockerfile для Docker-образа «bos/debian1»:

```
FROM debian:latest
RUN apt-get update && apt-get install -y nano
```

Для контейнера, который обеспечивает получение навыков по теме «Разграничение прав доступа к локальным ресурсам средствами операционной системы», следует добавить программу «acl», которая позволит задавать acl-права доступа. В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian2».

Содержание Dockerfile для Docker-образа «bos/debian2»:

```
FROM bos/debian1
RUN apt-get update && apt-get install -y acl
```

Для контейнера, который обеспечивает получение навыков по теме «Управление системными сервисами операционной системы», была задана такая конфигурация Docker-образа, которая запускает систему инициализации systemd.

Для создания возможности мониторинга введенных команд пользователем внутри контейнера, в Dockerfile были введены команды создания файла

.bash_history, а также установки прав к нему «777», что позволяет просматривать данный файл в монтированного из контейнера в хост-систему каталога.

Кроме того по ранее сформированному содержанию данной работы, предполагается наличие трех программ, которые будут запускаться создаваемыми службами systemd.

Первая программа programm1 должна запускаться службой .service. Данная программа, работая в фоновом режиме, записывает в файл /mnt/logs_programm1 сколько секунд она работает. Программа была написана на языке C++. Код программы:

```
#include <unistd.h>
#include <fstream>
using namespace std;
int main(int argc, char **argv)
{
    long int x = 0;
    while (true)
    {
        ofstream fil("/mnt/logs_programm1");
        fil << "Programm working " << x * 5 << " seconds\n";
        fil.close();
        x++;
        sleep(5);
    }
    return 0;
}
```

Для данной программы был создан текстовый файл readme1 с её описанием. Содержание файла readme1:

```
The program writes to the file /mnt/logs_programm1 how many seconds
it is running every 5 seconds.
The program does not end
```

Вторая программа programm2 должна запускаться при срабатывании юнита .timer. Данная программа, дописывает в конец файла /mnt/logs_programm2 в какое время она была запущена. Программа была написана на языке C++. Код программы:

```
#include <unistd.h>
#include <fstream>
#include <ctime>
using namespace std;
int main(int argc, char **argv)
{
```

```
    time_t seconds = time(NULL);
    tm* timeinfo = localtime(&seconds);
    ofstream fil("/mnt/logs_programm2", ios::app);
    fil << "Programm2 start: " << asctime(timeinfo) << '\n';
    fil.close();
    return 0;
}
```

Для данной программы был создан текстовый файл readme2 с её описанием. Содержание файла readme2:

```
The program prints the following line to the end of the file
'/mnt/logs_programm2':
Programm2 start: <current time>
```

Третья программа programm3 должна запускаться при срабатывании юнита .path. Данная программа, дописывает в конец файла /mnt/logs_programm3, информацию о том, что файл /mnt/text.txt был изменен. Программа была написана на языке C++. Код программы:

```
#include <unistd.h>
#include <fstream>
#include <ctime>
using namespace std;
int main(int argc, char **argv)
{
    time_t seconds = time(NULL);
    tm* timeinfo = localtime(&seconds);
    ofstream fil("/mnt/logs_programm3", ios::app);
    fil << "File /mnt/text.txt has changed! Time: " <<
asctime(timeinfo) << '\n';
    fil.close();
    return 0;
}
```

Для данной программы был создан текстовый файл readme3 с её описанием. Содержание файла readme3:

```
The program prints the following line to the end of the file
'/mnt/logs_programm3':
File /mnt/text.txt has changed! Time: <current time>
```

Файлы программ programm1, programm2 и programm3 с файлами инструкциями (readme1, readme2, readme3) были перемещены в каталог с Dockerfile. После чего Dockerfile был дополнен инструкцией «COPY . /mnt», которая копирует все файлы из текущего каталога хост-системы в каталог /mnt контейнера.

В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian3».

Содержание Dockerfile для Docker-образа «bos/debian3»:

```
FROM bos/debian2

ENV container docker
ENV LC_ALL C
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update \
    && apt-get install -y systemd systemd-sysv \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

RUN touch /root/.bash_history && chmod 777 /root/.bash_history

COPY . /mnt

RUN rm -f /lib/systemd/system/multi-user.target.wants/* \
    /etc/systemd/system/*.wants/* \
    /lib/systemd/system/local-fs.target.wants/* \
    /lib/systemd/system/sockets.target.wants/*udev* \
    /lib/systemd/system/sockets.target.wants/*initctl* \
    /lib/systemd/system/sysinit.target.wants/systemd-tmpfiles-
 setup* \
    /lib/systemd/system/systemd-update-utmp* \
    /mnt/Dockerfile

VOLUME [ "/sys/fs/cgroup" ]

CMD ["/lib/systemd/systemd"]
```

Для контейнера, который обеспечивает получение навыков по теме «Управление сетевыми интерфейсами и сетевым экраном в операционной системе», следует добавить межсетевой экран nftables и систему ulogd2. Также было решено добавить программу tcpdump для анализа входящего трафика. Кроме того необходимо удалить файлы программ контейнера предыдущей темы. Данные файлы находились в каталоге /mnt. В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian4».

Содержание Dockerfile для Docker-образа «bos/debian4»:

```
FROM bos/debian3
RUN rm /mnt/*
RUN apt-get update && apt-get install nftables -y && apt-get install
tcpdump -y && apt-get install -y ulogd2
```

Для контейнера, который обеспечивает получение навыков по теме «Работа с системными журналами в операционной системе», следует установить пакет «rsyslog», который обеспечит ведение системных журналов.

Последний пункт содержания предусматривает наличие служб systemd, которые запускаются с ошибками. Для этого были разработаны две программы ExecProgrammOne и ExecProgrammTwo с юнитами, которые их запускают ProgrammOne.service и ProgrammTwo.service. Программы были написаны на языке C++.

Первая программа ExecProgrammOne проверяет наличие в файле /mnt/FileProgrammOne строки «option yes». Если данная строка отсутствует, то программа в стандартный поток ошибок пишет соответствующее сообщение. В случае наличия нужной строки программа завершается с кодом 0. Код программы:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    string line;
    string x = "option yes";
    ifstream file("/mnt/FileProgrammOne");
    if (!file)
    {
        cerr << "File existence error. The file /mnt/FileProgrammOne
does not exist. You shouldn't have deleted it :). Restart the lab;)";
        return 1;
    }
    if (file.is_open())
    {
        getline(file, line);
        if (line != x)
        {
            cerr << "File settings error /mnt/FileProgrammOne. The
first line must be \"option yes\". Change this and restart the
service.";
            return 1;
        }
```

```
    }
    file.close();
    return 0;
}
```

Текст юнита ProgrammOne.service, который запускает программу ExecProgrammOne:

```
[Unit]
Description=ProgrammOne for Journald Lab. Good Luck you)

[Service]
ExecStart=/mnt/ExecProgrammOne

[Install]
WantedBy=multi-user.target
```

Вторая программа ExecProgrammTwo проверяет наличие файла /mnt/FileProgrammTwo. Если данный файл отсутствует, то программа в стандартный поток ошибок пишет соответствующее сообщение. В случае наличия нужного файла программа завершается с кодом 0. Код программы:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    ifstream file("/mnt/FileProgrammTwo");
    if (!file)
    {
        cerr << "File existence error. The file /mnt/FileProgrammTwo
does not exist. Please create him!";
        return 1;
    }
    file.close();
    return 0;
}
```

Текст юнита ProgrammTwo.service, который запускает программу ExecProgrammTwo:

```
[Unit]
Description=ProgrammOne for Journald Lab. Nice work!

[Service]
```

```
ExecStart=/mnt/ExecProgrammTwo

[Install]
WantedBy=multi-user.target
```

Файлы программ ExecProgrammOne и ExecProgrammTwo, файлы юнитов ProgrammOne.service и ProgrammTwo.service и текстовый файл FileProgrammOne были перемещены в каталог с Dockerfile. После чего Dockerfile был дополнен инструкциями «COPY», которые копируют программы в каталог /mnt, а юниты в каталог /etc/system/system.

В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian5».

Содержание Dockerfile для Docker-образа «bos/debian5»:

```
FROM bos/debian4

COPY ./ProgrammOne.service /etc/systemd/system
COPY ./ProgrammTwo.service /etc/systemd/system
COPY ./ExecProgrammOne /mnt
COPY ./ExecProgrammTwo /mnt
COPY ./FileProgrammOne /mnt

RUN apt-get update && apt-get install rsyslog –y
```

Для контейнера, который обеспечивает получение навыков по теме «Настройка удаленного управления на базе сетевого протокола ssh», следует добавить набор программ OpenSSH-server. Необходимо удалить файлы программ контейнера предыдущей темы. Данные файлы находились в каталоге /mnt. Также необходимо установить пароль root, так как при подключении по SSH он будет запрашиваться. Также в конфигурационном файле sshd_config необходимо разрешить root-доступ и разрешить парольную аутентификацию. В соответствии с этим был создан Dockerfile, на основе которого был создан Docker-образ «bos/debian6».

Содержание Dockerfile для Docker-образа «bos/debian6»:

```
FROM bos/debian5
RUN echo "root:123ibst789DEB" | chpasswd
RUN rm /mnt/*
RUN apt-get update && apt-get install openssh-server –y
```

```
RUN sed -E 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config -i && sed -E 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config -i &&  ssh-keygen -A
```

Все Docker-образы были собраны. В результате было получено 6 Docker-образов общим размером 215 Мбайт.

Для каждого Docker-образа был получен хэш-код от хэш-функции SHA256. В таблице 2 представлены данные хэш-коды:

Таблица 2 – Хэши Docker-образов

Docker-образ	Хэш-код SHA256
bos/debian1	218ceef2378360b8cc20eb19122b3c67607dea59383184f5789b69f60091f85d
bos/debian2	729d66c56f36300f76b70ecf1578195af0bbd407448d7b516ad388cd9494ac6c
bos/debian3	b2a43c9e970a5242cea48558eecd22a7a4c09d50bcedb084e66d5c997caaf737
bos/debian4	db8af1314c8b3a5082f6415a5ff46d44b923f97b8da9dc4de6ba41de0796b1b6
bos/debian5	9076dc755529611919c83246ec15379bc0464c102c24fb60579301eb072cc3a4
bos/debian6	da105b9960d3869d6518a60676d5da16d1fed786c5699d11e735936ec944c185

3.2 Разработка подсистемы хранения, доставки и развертывания контейнеров

Исходя из малого размера созданных Docker-образов, а также достоинств локального хранения файлов была выбрана следующая концепция:

— Docker-образы хранятся по стандартному пути /var/lib/docker на каждом компьютере локально;

— в сетевом хранилище хранится архив с экспортированными Docker-образами в формате .tar, а также файлами Dockerfile для каждой работы;

— на каждом компьютере локально хранится файл check_image.conf, который содержит полный путь до архива с Docker-образами в сетевом хранилище, а также эталонные хэш-коды для Docker-образов;

— при запуске тренажера проводится проверка наличия необходимых Docker-образов, а также проверка соответствия их хэш-кодов эталонным значениям из файла check_image.conf;

— в случае отсутствия необходимых Docker-образов или несоответствия хэш-кодов эталонным значениям должна проводиться процедура загрузки Docker-образов из заготовленного архива .tar из сетевого хранилища;

— после процедуры загрузки Docker-образов проводится дополнительная проверка соответствия хэш-кодов эталонным значениям;

— в случае неудовлетворительного результата дополнительной проверки должно выводиться сообщение об данной ошибке.

Данная концепция является наиболее рациональной, так как она позволяет путем централизованной настройки файла check_image.conf на компьютерах в классе или аудитории изменить или обновить используемые Docker-образы. Также такая система будет являться автономной, так как в случае корректной загрузки Docker-образов из сетевого хранилища, данные Docker-образы хранятся локально, что позволяет пользоваться тренажером даже в отсутствии соединения с локальной сетью.

Для реализации данного функционала были написаны две функции на языке программирования Python с использованием Docker SDK: CheckImages() и LoadImages().

Функция CheckImages() обращается к файлу check_images.conf, из которого считывает эталонные значения хэш-кодов Docker-образов. После чего она проверяет наличие необходимых Docker-образов, а также соответствие хэш-кодов данных Docker-образов эталонным значениям. Функция возвращает список названий Docker-образов, которые отсутствуют в системе или не прошли проверку. Текст данной функции:

```
def CheckImages():
    clientDocker = docker.from_env()
    list_hash = dict()
    with open('/etc/trainerIBOS/check_images.conf', 'r') as FILE:
        FileImagesTar = FILE.readline().split('=')[1].rstrip()
        for line in FILE:
            lst = line.split('=')
            list_hash[lst[0]] = [lst[1].rstrip(), False]

    list_images = clientDocker.images.list()
```

```
    for image in list_images:
        if len(image.tags) == 1:
            name_image = image.tags[0].split(':')[0]
            if name_image in list_hash:
                if list_hash[name_image][0] == image.id:
                    list_hash[name_image][1] = True
    lst_error = []
    for key in list_hash:
        if list_hash[key][1] == False:
            lst_error.append(key)

    return lst_error
```

Функция LoadImages () обращается к файлу check_images.conf, из которого считывает место хранения архива с экспортируемыми эталонными Docker-образами. После чего осуществляется загрузка Docker-образов в систему из архива. Функция возвращает код завершения: 0 – корректное выполнение, 1 – некорректное выполнение. Текст данной функции:

```
def LoadImages():
    clientDocker = docker.from_env()
    with open('/etc/trainerIBOS/check_images.conf', 'r') as FILE:
        FileImagesTar = FILE.readline().split('=')[1].rstrip()
    if os.path.exists(FileImagesTar):
        ImagesTar = open(FileImagesTar, 'rb')
        clientDocker.images.load(ImagesTar)
        return 0
    else:
        return 1
```

Применение данных функций перед запуском тренажера позволит загрузить необходимые Docker-образы в случае их отсутствия, а также проверить корректность ранее загруженных Docker-образов.

3.3 Разработка подсистемы запуска контейнеров и формирования задания

Режим тренировки должен обеспечивать информационные сообщения о промежуточных результатах выполнения работы. Для создания такого функционала было решено использовать пользовательские юниты Systemd. Для каждой работы было создана по два юнита: .path и .service.

Юнит .path следит за изменениями, происходящими в файлах контейнера, а юнит .service, срабатывая по команде юнита .path, запускает скрипт промежуточной проверки, который будет информировать о выполненных пунктах задания.

Для проверки изменений файлов, которые происходят в контейнере, из хост-системы использовались Docker Volumes (Docker-томы).

Docker-том – это файловая система, которая расположена на хост-машине за пределами контейнеров. Созданием, изменением и удалением занимается демон Docker. Docker-том представляет собой средство для постоянного хранения информации, то есть в случае подключения Docker-тома к контейнеру, вывода информации в Docker-том, данная информация сохраняется даже после завершения работы контейнера. Docker-томы самостоятельны и обособлены от запущенных и завершенных контейнеров. Один Docker-том может использоваться несколькими контейнерами при необходимости. Именно Docker-томы позволяют организовать эффективное чтение из хост системы.[9]

Схема работы Docker-томов представлена на рисунке 4.

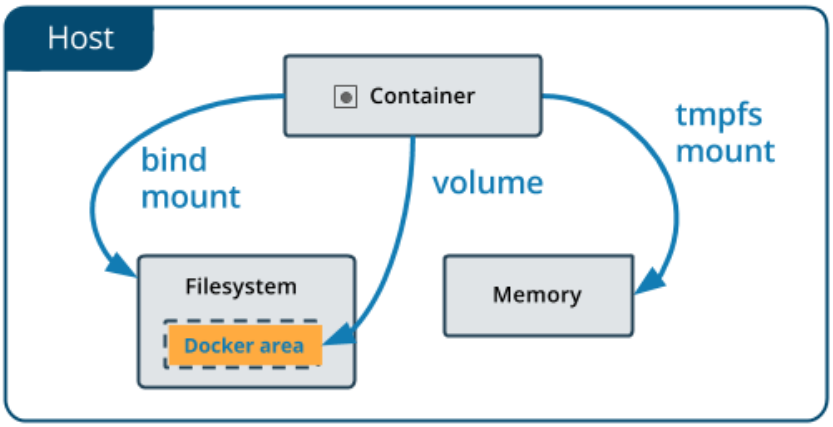

Рисунок 4 – Схема работы Docker-томов

Далее были проанализировано, какие каталоги контейнера необходимо монтировать Docker-томом для чтения файлов, в которых находится информация о выполненных или невыполненных действиях. Необходимые Docker-томы для каждого из контейнеров представлены в таблице 3.

Таблица 3 – Docker-томы для контейнеров

Docker-образ	Название Docker-тома	Монтируемые каталоги контейнера	Режим монтирования
bos/debian1	trainerIBOS_VOLUME	/etc	rw
bos/debian2	trainerIBOS_VOLUME	/etc	rw
	trainerIBOS_VOLUME_mnt	/mnt	rw
bos/debian3	trainerIBOS_VOLUME	/root	rw
bos/debian4	trainerIBOS_VOLUME	/root	rw
	trainerIBOS_VOLUME2	/mnt	rw
bos/debian5	trainerIBOS_VOLUME	/root	rw
bos/debian6	trainerIBOS_VOLUME	/home	rw
	trainerIBOS_VOLUME2	/home	rw

В таблице 2 было учтено, что на основе Docker-образа bos/debian6 будут запускаться два контейнера для подключения к друг другу по протоколу SSH, поэтому будут создаваться два Docker-тома для каталогов /home обоих контейнеров.

Также для контейнеров, построенных на основе bos/debian3 и более поздних, будет осуществляться bind монтирование каталога хост-системы /sys/fs/cgroup в каталога /sys/fs/cgroup контейнера в режиме чтения. Это необходимо для загрузки системы инициализации Systemd.

Были разработаны шаблоны юинтов .path и .service для каждого контейнера. Шаблоны юнитов .service идентичны друг другу, отличия заключаются лишь в описании самого юнита. Шаблоны юнитов .path различны, так как в разных работах необходимо следить за разными файлами и папками. Ниже приведен пример шаблона юнита Lab1.path первой работы, который следит за файлами /etc/passwd и /etc/group контейнера:

```
[Path]
PathChanged=/var/lib/docker/volumes/{0}/_data/passwd
PathChanged=/var/lib/docker/volumes/{0}/_data/group
[Install]
WantedBy=multi-user.target
```

В данном шаблоне были используется конструкция {0}, которая необходима для программной подстановки имени используемого Docker-тома.

Шаблон юнита Lab1.service первой работы, который запускает скрипты промежуточной проверки:

```
[Unit]
Description= CheckLab1

[Service]
ExecStart=python3 {0}

[Install]
WantedBy=multi-user.target
```

В данном шаблоне также используется конструкция {0} для программной подстановки нужного имени скрипта промежуточной проверки.

Для практической работы «Настройка удаленного управления на базе сетевого протокола ssh» и соответствующего контейнера bos/debian6 не были созданы шаблоны юнитов, так как данная работа не предусматривает информационные сообщения в режиме тренировки.

Шаблоны юнитов .path представлены в таблице 4.

Таблица 4 – Текст шаблонов юнитов .path

Контейнер	Текст шаблона юнита .path
bos/debian1	[Path] PathChanged=/var/lib/docker/volumes/{0}/_data/passwd PathChanged=/var/lib/docker/volumes/{0}/_data/group [Install] WantedBy=multi-user.target
bos/debian2	[Path] PathChanged=/var/lib/docker/volumes/{0}/_data/group PathChanged=/var/lib/docker/volumes/{1}/_data/{2} PathChanged=/var/lib/docker/volumes/{1}/_data/{3} PathChanged=/var/lib/docker/volumes/{1}/_data/{4} PathChanged=/var/lib/docker/volumes/{1}/_data/{5} PathChanged=/var/lib/docker/volumes/{1}/_data/{2}/text.txt PathChanged=/var/lib/docker/volumes/{1}/_data/{3}/text.txt PathChanged=/var/lib/docker/volumes/{1}/_data/{4}/text.txt PathChanged=/var/lib/docker/volumes/{1}/_data/{5}/text.txt [Install] WantedBy=multi-user.target
bos/debian3	[Path] PathChanged=/var/lib/docker/volumes/{0}/_data/.bash_history [Install] WantedBy=multi-user.target
bos/debian4	[Path] PathChanged=/var/lib/docker/volumes/{0}/_data/.bash_history [Install] WantedBy=multi-user.target

Продолжение таблицы 4

Контейнер	Текст шаблона юнита .path
bos/debian5	[Path] PathChanged=/var/lib/docker/volumes/{0}/_data/.bash_history [Install] WantedBy=multi-user.target
bos/debian6	Пользовательские юниты Systemd не используются

Для запуска контейнеров был написан модуль PyDocker.py на языке Python с использованием Docker SDK. В данном модуле для каждого контейнера содержится по две функции: функция запуска контейнера и формирования его задания, а также функция завершения работы контейнера. Также в данный модуль были добавлены функции проверки и загрузки Docker-образов CheckImages() и LoadImages().

Для формирования задания необходимы редактируемые наборы данных с, например, именами пользователей, которых нужно создать в системе, именами системных служб Systemd, которые нужно перезапустить и т.д. Данные наборы данных были разработаны в виде обычных текстовых файлов в формате .conf. Такие текстовые файлы были названы Lab<X>-names.conf (где <X> - номер работы). В данных файлах в некоторой строке стоит число, показывающее сколько строк после него будет содержать однотипную информацию для формирования задания. Например, для первой работы по управлению пользователями и группами в файле Lab1-names.conf в самом начале стоит число 71, после чего идет 71 строка с логинами и именами пользователей. После последней пары логина и имени пользователя идет новое число 28, показывающее, что после него будет идти 28 возможных имен групп. Подобные файлы были созданы для каждой темы. Из данных файлов будет производиться случайная выборка некоторого количество имен и значений для формирования заданий по каждой теме.

Так как тренажер предполагает различные задания в зависимости от уровня сложности, функции запуска контейнеров имеют два параметра train и

diff. Параметр train – это логический флаг, определяющий был ли выбран режим тренировки. Параметр diff – целое число, определяющее уровень сложности.

Кроме того функции запуска контейнеров возвращают единственное значение deskLab – текст сформированного задания.

Каждая функция запуска контейнеров выполняет:

— формирование задания;

— сохранение основной информации задания в файл в виде потока байт с помощью сериализации, выполняемое модулем pickle;

— создание необходимых Docker-элементов для выполнения работы (Docker-томы, Docker-сети);

— запуск контейнера с необходимыми параметрами;

— если флаг train = True, создание и запуск пользовательских юнитов systemd, позволяющих следить за файлами контейнера;

— создание вопросов, которые будут заданы по завершению работы контейнеров, а также сохранение данных вопросов в файл.

При запуске одного контейнера ему всегда дается одно и то же имя: «trainerIBOS». В случае необходимости создания двух контейнеров, второму контейнеру дается имя «IBOStrainer». Для файла с основной информацией задания было дано название «trainerIBOS.task», а для файла с контрольными вопросами «trainerIBOS.question».

Функции завершения работы контейнеров принимают лишь один аргумент – train. Данный аргумент – флаг, определяющий был ли выключен режим тренировки. Данный аргумент необходим, так как в случае тренировки в хост-системе создаются и запускаются юниты Systemd, которые после выполнения работы необходимо завершить и удалить.

Функции завершений работы контейнеров, если работа предполагает в конце контрольные вопросы, возвращает список данных вопросов, которые считываются из заготовленного файла, а также стоимость данных вопросов в процентах.

Каждая функция завершения работы контейнера выполняет:

— удаление Docker-элементов (Docker-томов, Docker-сетей);

— остановку и удаление Docker-контейнера;

— если флаг train = True, остановку и удаление пользовательских юнитов Systemd;

— удаление файлов с основной информацией сформированного задания;

— чтение контрольных вопросов из файла и последующее удаление данного файла.

В таблице 5 представлены названия функций.

Таблица 5 – Названия функций для запуска и завершения работы контейнеров

Контейнер	Функция запуска	Функций завершения работы
bos/debian1	startTaskControlUserAndGroup(diff, train)	killTaskControlUserAndGroup(train)
bos/debian2	startChmodLab(diff, train)	killChmodLab(train)
bos/debian3	startSystemdLab(diff, train)	killSystemdLab(train)
bos/debian4	startNetworkLab(diff, train)	killNetworkLab(train)
bos/debian5	startJournalLab(diff, train)	killJournalLab(train)
bos/debian6	startSSHLab(diff, train)	killSSHLab(train)

Кроме того модуль PyDocker.py включает разработанную функцию copy_to(src, dst), которая позволяет копировать файлы из хост-системы в контейнер, используя Docker SDK. Полный код модуля PyDocker.py представлен в каталоге trainer репозитория MikhailNagaev/trainerIBOS.[7]

3.4 Разработка подсистемы автоматического контроля выполнения заданий

Для автоматического контроля выполнения заданий в различных контейнерах были разработаны скрипты проверки, написанные на языке программирования Python. Данные скрипты осуществляют:

— считывает основную информацию о сформированном задании из соотствующего файла;

— проверку правильности выполнения заданий;

— подсчёт процента выполнения;

— формирование текстового сообщения о том, что было сделано в работе неверно.

Скрипты не принимают никаких аргументов, вся необходимая информация читается непосредственно из файла задания trainerIBOS.task и файлов самого контейнера, которые находятся в Docker-томе.

Каждый скрипт проверки возвращает три значения: код завершения проверки (0 – всё выполнено верно в соответствии с заданием, 1 – есть ошибки), процент правильности выполнения задания, текстовое сообщение об ошибках.

Алгоритмы проверки основывались:

— анализе файлов контейнера с помощью заранее созданных Docker-томов;

— выполнении команд внутри контейнера и последующего анализа их вывода.

Также были разработаны скрипты промежуточной проверки в режиме тренировки. Данные скрипты работают аналогично основным скриптам проверки, но для них было учтено, что они будут запускаться обособленно от основной программы с помощью юнитов .service. Данный факт означает, что данные скрипты нельзя будет использовать в основной программе в виде функций, которые возвращали бы результат проверки. Поэтому была использована следующая концепция:

— передача информации между скриптом промежуточной проверки и основной программой осуществляется с помощью сокетов;

— в основной программе тренажера будет запущен tcp-сервер ожидающий сообщений на порту 8997 (данный порт используется по умолчанию, но его можно изменить в конфигурационном файле port_time.conf);

— в конце промежуточной проверки скрипт запускает tcp-клиент, которые отправляет некоторую байтовую комбинацию на сервер на порт 8997;

— сервер принимает байтовую комбинацию и в соответствии с таблицей сообщений будет выводить нужное сообщение на экран.

Таблицы соответствия байтовых комбинаций конкретным сообщениями (на русском языке) будут располагаться в файлах Lab<X>-code.conf (где <X> - номер работы). Пример таблицы соответствия для первой практической работы представлен ниже:

```
codeLab1-1=Задание 1 выполнено. Пользователи созданы верно.
codeLab1-2=Задание 2 выполнено. Группы созданы верно.
codeLab1-3=Задание 3 выполнено. Пользователи внесены в группы верно.
```

Аналогичные таблицы соответствия были созданы для каждой практической работы.

Скрипты проверки и промежуточной проверки режима тренировки представлены в каталоге /etc-trainerIBOS/script репозитория MikhailNagaev/trainerIBOS. Также важно, что для четвертой практической работы «Управление сетевыми интерфейсами и сетевым экраном в операционной системе» были созданы два скрипта проверки. Один из скриптов с помощью юнитов Systemd запускается каждый раз, когда в контейнере выполняется команда. Данный скрипт отслеживает все изменения происходящие с файлами контейнера в течении всей работы, а не только конечное состояние и содержание файлов. Это позволяет осуществить проверку в случае, когда файлы системы в течении работы изменяются многократно.[7]

3.5 Разработка пользовательского интерфейса тренажера

Для создания графического приложения было решено использовать PyQt5 – набор расширений графического фреймворка Qt. Для создания формы приложения использовался конструктор Qt Designer. Разработанная форма представленна на рисунке 5.

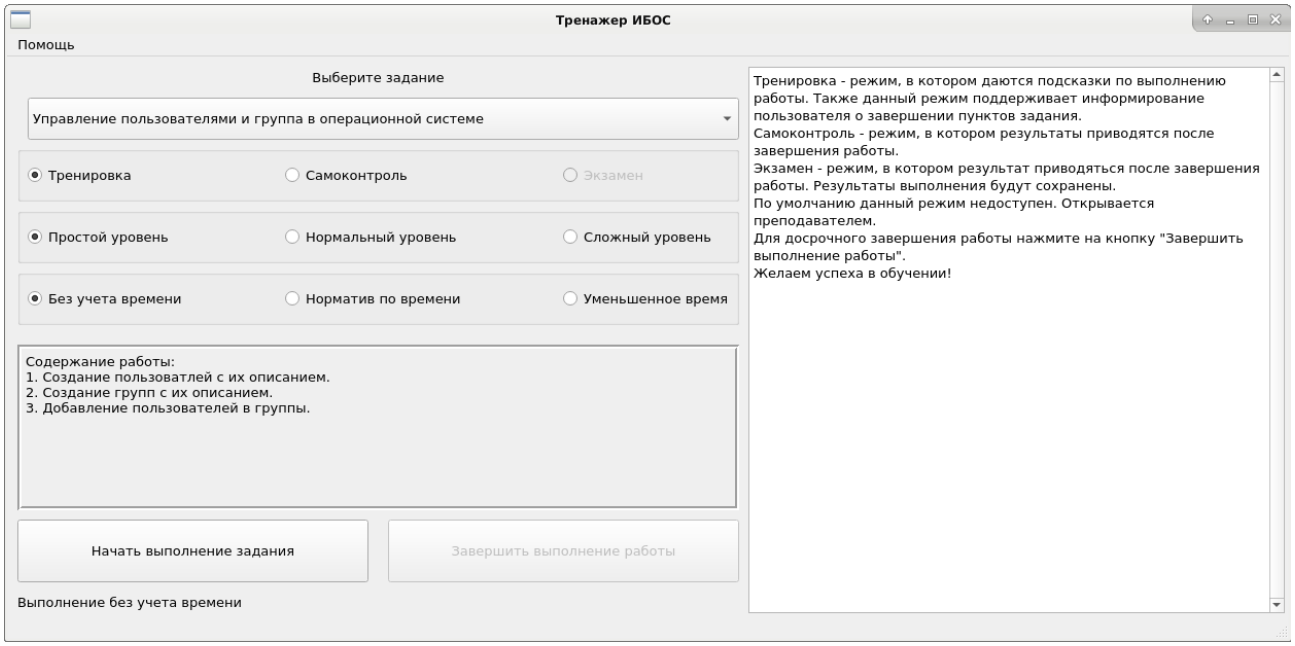

Рисунок 5 – Разработанная форма тренажера

На рисунке 6 представлена форма с пронумерованными элементами.

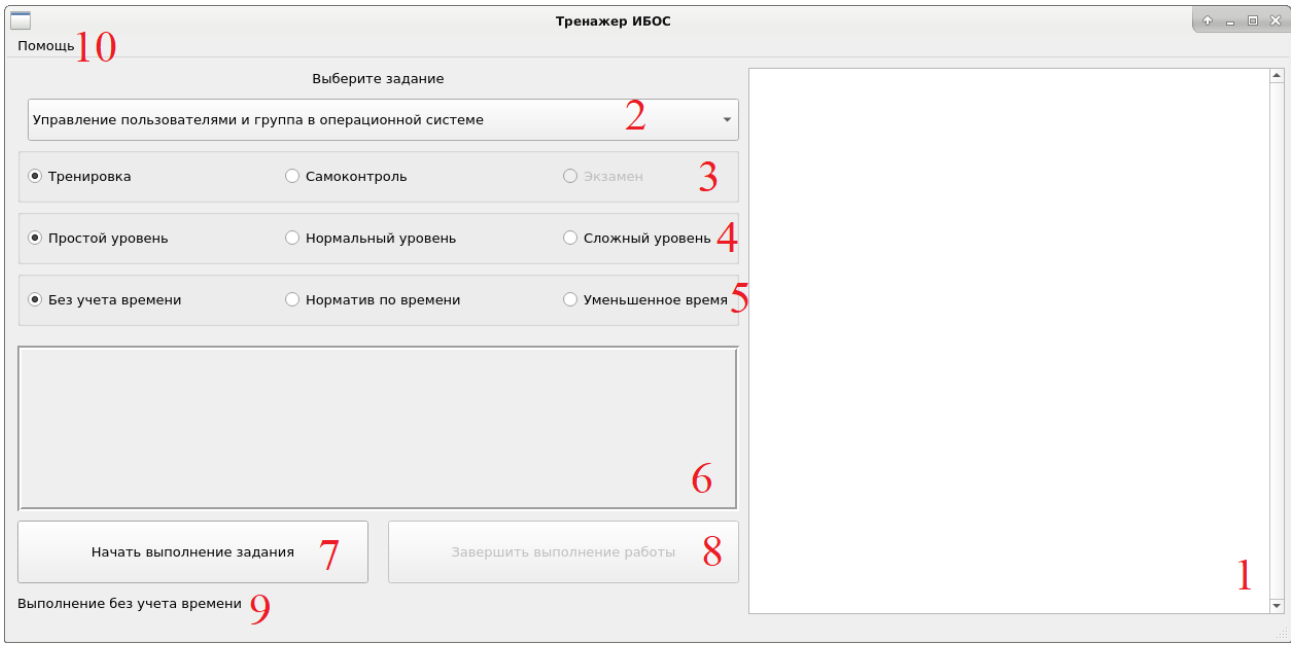

Рисунок 6 – Форма тренажера с пронумерованными элементами

В таблице 6 представлено описание элементов тренажера.

Таблица 6 – Описание элементов формы тренажера

№ элемента	Тип элемента	Описание
1	Текстовое поле (режим чтения)	Основной элемент формы. В данном текстовом поле будут выводиться все сообщения, относящиеся к выполнению практических работ. К таким сообщениям относятся: задание на практическую

Продолжение таблицы 6

№ элемента	Тип элемента	Описание
1	Текстовое поле (режим чтения)	работу, подсказки по выполнению в режиме тренировки, результаты выполнения работы, ошибки выполнения работы и т.д.
2	КомбоБокс (выпадающий список)	Данный выпадающий список необходим для выбора темы практической работы.
3	Группа переключателей	Данная группа определяет режим работы тренажера. Режим «Экзамен» по умолчанию закрыт. Его должен открывать преподаватель.
4		Данная группа определяет уровеень сложности практической работы.
5		Данная группа определяет доступное время выполнения работы. Для каждой работы время для пункта «норматив по времени» и «уменьшенное время» индивидуален.
6	Лейбл (невыделяемое текстовое поле)	Данный лейбл используется для отображения краткого содержания работы. При выборе темы практической работы из выпадающего списка (элемент №2), краткое содержание будет выводиться автоматически.
7	Кнопка	Кнопка начала работы по заданным параметрам.
8		Кнопка завершения работы. Изначально неактивна.
9	Лейбл (невыделяемое текстовое поле)	Данный лейбл выводит информацию о времени выполнения работы. В нем отображается инфомация о том, сколько отведено на выбранную в списке тему, в соответствии с выбором из группы переключателей (элемент №5). При запуске работы в лейбле отображается продолжительность выполнения.
10	Строка меню	Данная строка меню содержит одну кнопку «Помощь». Данная кнопка при нажатии её нажатии в основном текстовом поле (элемент №1) выводит краткую инструкицю по использованию тренажера.

При разработке кода для тренажера были учтены следующие моменты:

— работа с графической формой тренажера должна быть максимально простой, она сводится к выбору темы практической работы, выбору режима, сложности и времени выполнения, после чего необходимо нажать кнопку начала выполнения, чтобы запустить необходимый контейнер и сопутствующее Docker-элементы, а также сформировать и вывести в текстовое поле задание;

— после начала выполнения работы все элементы графической формы должны стать недоступными за исключением кнопки заверешения работы, она, наоборот, становится доступной;

— после заверешения выполнения работы из-за истечения времени или нажатия на соответствующую кнопку, в текстовое поле должна быть выведена вся необходимая информация: результат выполнения, процент правильности выполнения задания, а также данные о совершенных ошибках;

— контрольные вопросы, генерируемые подсистемой запуска контейнеров и формирования задания, должны задваться в конце выполняемой работы;

— в случае неправильных ответов на контрольные вопросы, в текстовое поле должно выводиться соответствующее сообщение об ошибке;

— режим «Экзамен» по умолчанию недоступен;

— режим «Экзамен» может быть открыт только преподавателем.

Так как для каждой работы необходимо выводить её краткое содержание, были созданы текстовые файлы Lab<X>-Desc.conf (где <X> - номер работы), из которых считывается краткое содержание и выводиться на экран.

Также для режима тренировки были созданы файлы Lab<X>-Hint.conf (где <X> - номер работы), в которых содержаться методические указания к выполнению работы.

Для централизованного открытия режима «Экзамен» тренажеров, работающих на разных компьютерах в классе или аудитории был разработан следующий алгоритм:

— при запуске тренажера создается файл runningtrainer.conf, в котором содержится информация о времени запуска тренажера, а также о пользователе запустившем тренажер;

— преподаватель с помощью протокола ssh программно подключается к различным компьютерам в локальной сети, и если на компьютерах имеется файл runningtrainer.conf, то считается, что тренажер на компьютере запущен;

— для известных компьютеров с запущенным тренажером преподаватель по протоколу ssh с помощью программы scp может отправить на удаленный компьютер специальный конфигурационный файл, открывающий экзамен;

— тренажер в случае появления данного файла прочитывает его содержимое и на основе этого открывает режим «Экзамена» с необходимыми настройками;

— после чего данный конфигурационный файл удаляется из системы.

Весь вышеописанный функционал реализуется модулем trainer.py, который подключает заранее созданный интерфейс в виде модуля interfacePyQt.py, который был создан автоматическиз с помощью программы Qt Designer. Модуль trainer.py состоит из двух классов, один из которых наследует программный итерфейс от класса, подключаемого модуля interfacePyQt.py.

Весь модуль trainer.py состоит из методов класса ExampleApp и обособленных функций. Также имеется класс BrowserHandler, который реализует tcp-сервер, работающий в отдельном потоке.

Ниже приведено описание используемых методов и функций:

Функция TimeToAsk(question, procent) выводит вопросы на экран в виде MsgBox, подсчитывает количество правильных ответов на вопросы, считает на сколько процентов было отвечено верно. Данная функция выполняется после

выполнения практической работы. В качестве аргументов принимает список вопросов, а также суммарную стоимость всех вопросов в процентах. Возвращает три значения: код завершения (0 – если на все вопросы были даны верные ответы, 1 – если был дан хотя бы один неверный ответ), текст ошибок, процент правильности выполнения тестов.

В конструкторе класса ExampleApp производится базовая настройка приложения:

— проверка и загрузка Docker-образов с помщью методов CheckImages() и LoadImages();

— подключение сигналов различных элементов графической формы к слотам Qt (методам класса ExampleApp);

— создание файла runningtrainer.conf;

— создание отдельного потока для экземпляра класса BrowserHandler, который будет реализовывать tcp-сервер;

— чтение файла, содержащего краткое описание для первой практической работы, а также отображение данного описания в соответствующее поле.

Метод GoToExam класса ExampleApp настраивает активность переключателей в случае выбора режима «Экзамен». Данный метод является слотом для сигнала от таймера, который детектирует появление специального файла, в котором находится информация о настройке запускаемого экзамена.

Метод GoToKnow класса ExampleApp выполняет соответствующие функции запуска контейнеров в соответствии с выбранными настройками: темой, режимом, уровнем сложности и временем выполнения. Вместе с контейнером происходит запуск таймера выполнения задания. Также данный метод выводит методические указания на текстовое поле, если выбран режим тренировки. Кроме того, данный метод с помощью другого метода appEnabled делает неактивными все элементы графической формы, кроме текстового поля и

кнопки завершения выполнения работы. Данный метод является слотом для сигнала нажатия на кнопку начала выполнения задания.

Метод appEnabled класса ExampleApp изменяет активность элементов графической формы. Принимает в качетсве аргумента логическое значение. Если передаваемое значение True, то все элементы формы, кроме кнопки завершения работы, становятся активными. Если передаваемое значение False, то все элементы формы, кроме кнопки заверешения работы, становятся, наоборот, неактивными.

Метод onChanged класса ExamleApp поддерживает актуальность краткого содержания, выводимого на экран. При выборе новой темы практического задания автоматически считывается краткое содержание из соответствующего файла и выводится на экран. Данный метод является слотом для сигнала изменения выбранного элемента в выпадающем списке.

Метод enterCode класса ExampleApp занимается обработкой текста из второго потока, в котором выполняется tcp-сервер. Данный метод принимает коды, которые приходят от вспомогательных скриптов проверки, после чего в соответствии с пришедшим кодом и файлом Lab<X>-code.conf (где <X> - это номер практической работы) выводит сообщение на экран.

Метод timerShow класса ExampleApp выводит в соответствующий лейбл время с начала выполнения работы. В случае завершения времени, данный метод вызывает метод EndLab(), который завершает выполнение практических работ. Данный метод является слотом для сигналов таймера, который создается в конструкторе класса.

Метод EditTime() класса ExampleApp выводит информацию о максимальном допустимом времени, которое отведено на выполнение работы. Данный метод является слотом для сигнала изменения выбранного переключателя, определяющего время выполнения работы.

Метод EndLab() класса ExampleApp выполняет соответствующие функции завершения выполнения работы. Данный метод также выводит на экран соответствующие результат выполнения работы. Кроме того, в случае выбора

режима «Экзамен» будет сохранена информация о результатах работы в файл. Данный метод является слотом для сигнала нажатия на кнопку завершения выполнения работы, а также слотом для сигнала срабатывания таймера времени выполнения работы.

Метод closeEvent() класса ExampleApp обеспечивает невозможность закрытия приложения стандартными средствами в случае, если запущены контейнеры. Данный метод работает по следующему принципу: если была нажата кнопка завершения работы приложения, но при этом запущен контейнер для выполнения какой-либо работы, то выводится сообщение о том, чтобы пользователь завершил выполнение работы с помощью соответствующей кнопки. Это необходимо для корректной работы всех элементов Docker, так как аварийное завершение работы приложения может повлечь нестабильную работу приложения в будущих запусках.

Метод timerEventCheckExam() класса ExampleApp необходим для открытия режима «Экзамен». Данный метод является слотом для сигнала от одного из таймеров.

В классе BrowserHandler имеется лишь один метод run, который обеспечивает работу tcp-сервера. В случае приема сообщений от скриптов промежуточной проверки, данный метод с помощью заранее созданного слота отправляет данные в класс ExampleApp. При этом соотнесение кодов и сообщений производится непосредственно в методах класса ExampleApp.

Полный код модуля trainer.py представлен в каталоге trainer репозитория MikhailNagaev/trainerIBOS.[7]

Для лучшего понимания работы основной программы тренажера была разработана архитектура тренажера, на которой отображены основные функциональные элементы тренажера, а также их взаимосвязи между собой.

На рисунке 7 представлена архитектура тренажера.

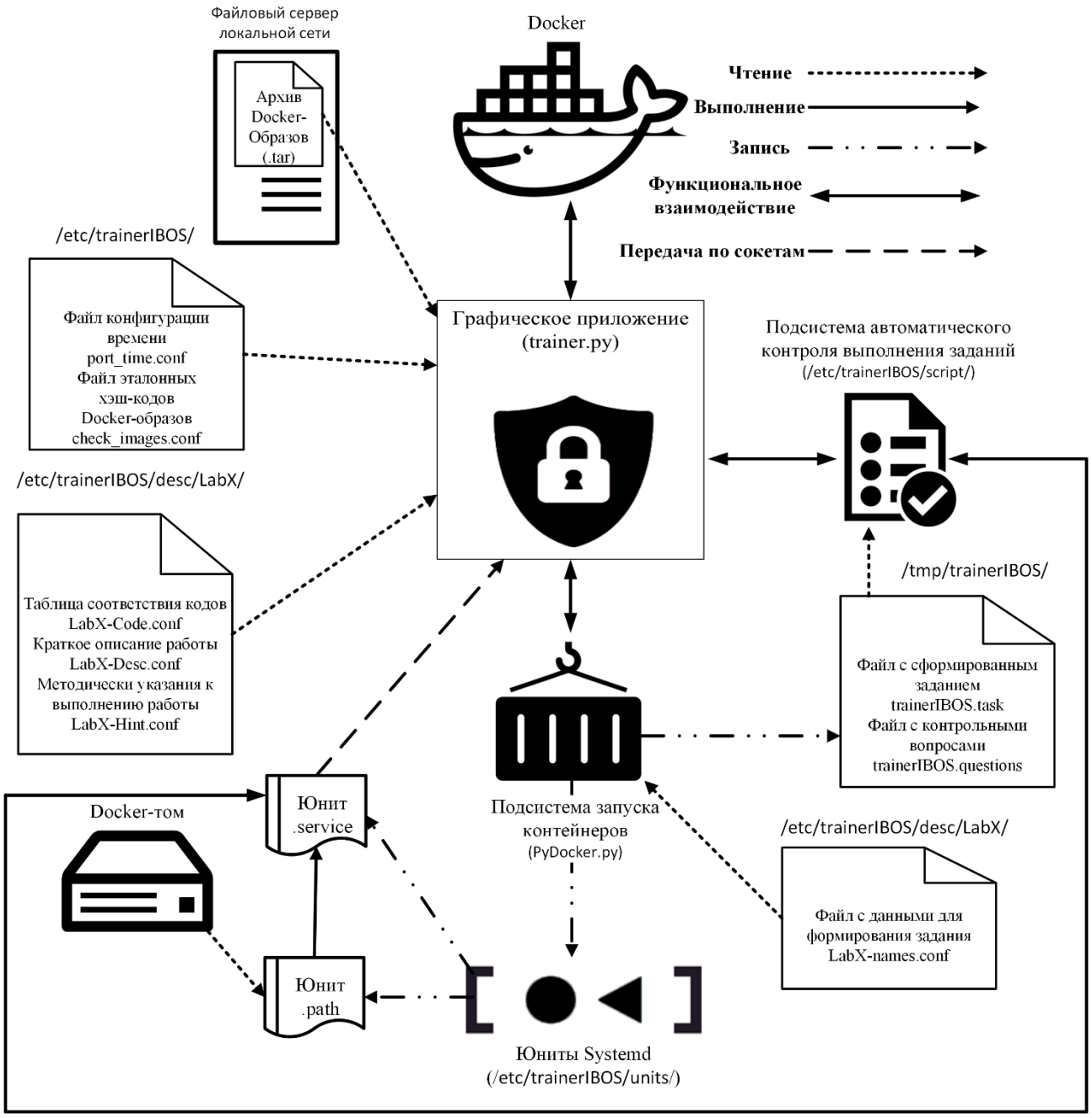

Рисунок 7 – Архитектура тренажера по информационной безопасности

Для лучшего понимания, каким образом необходимо работать с тренажером была разработана UML-диаграмма вариантов использования, которая представлена на рисунке 8. Данная диаграмма описывает типичные взаимодействия между пользователем и тренажером.

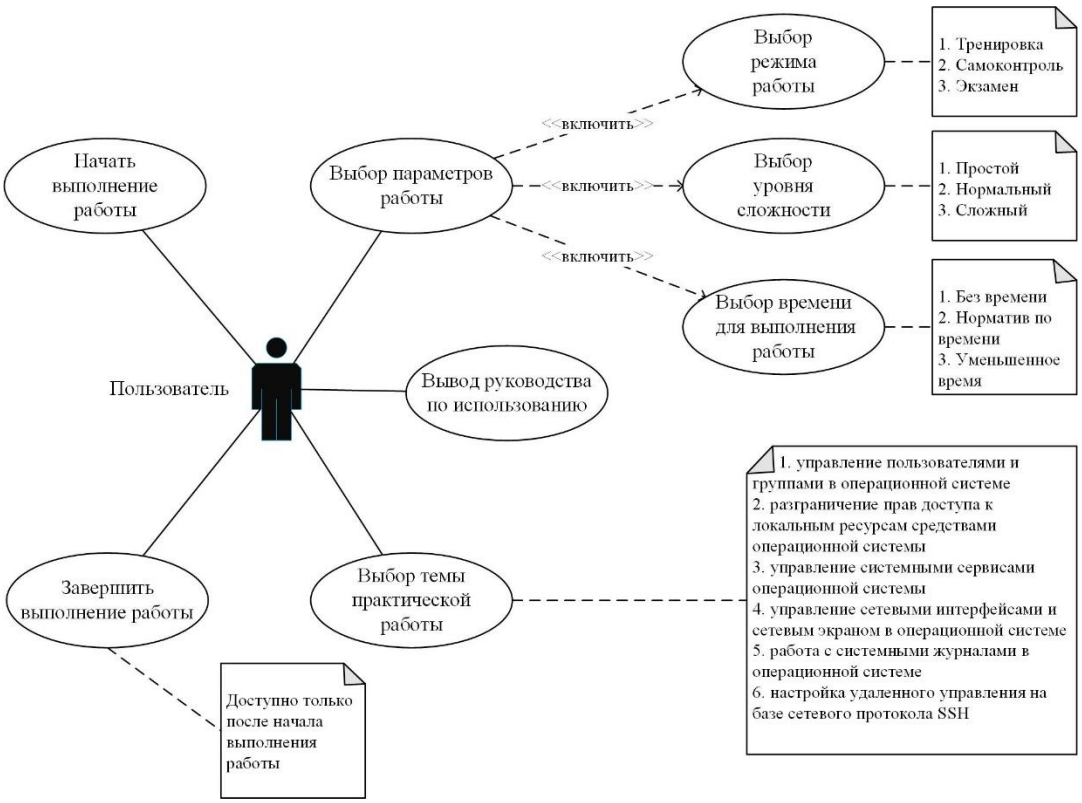

Рисунок 8 – UML-диаграмма вариантов использования

3.6 Разработка пользовательского интерфейса программы тьютора

Для создания графического приложения тьютора также был использован PyQt5. Форма приложения была создана в Qt Designer. Разработанная форма представленна на рисунке 9.

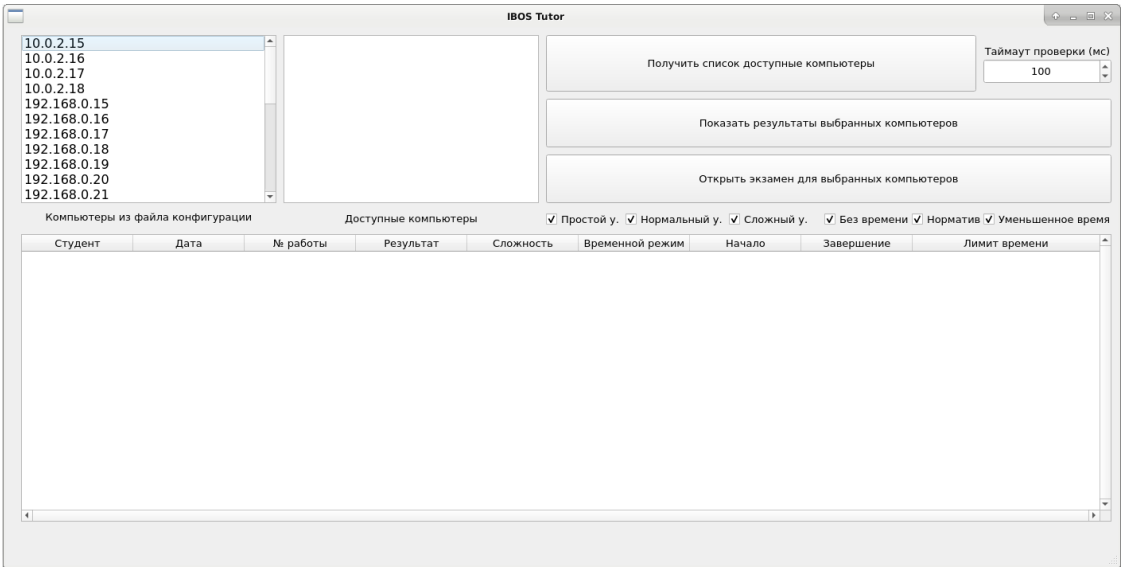

Рисунок 9 – Разработанная форма программы тьютора

На рисунке 10 представлена форма с пронумерованными элементами.

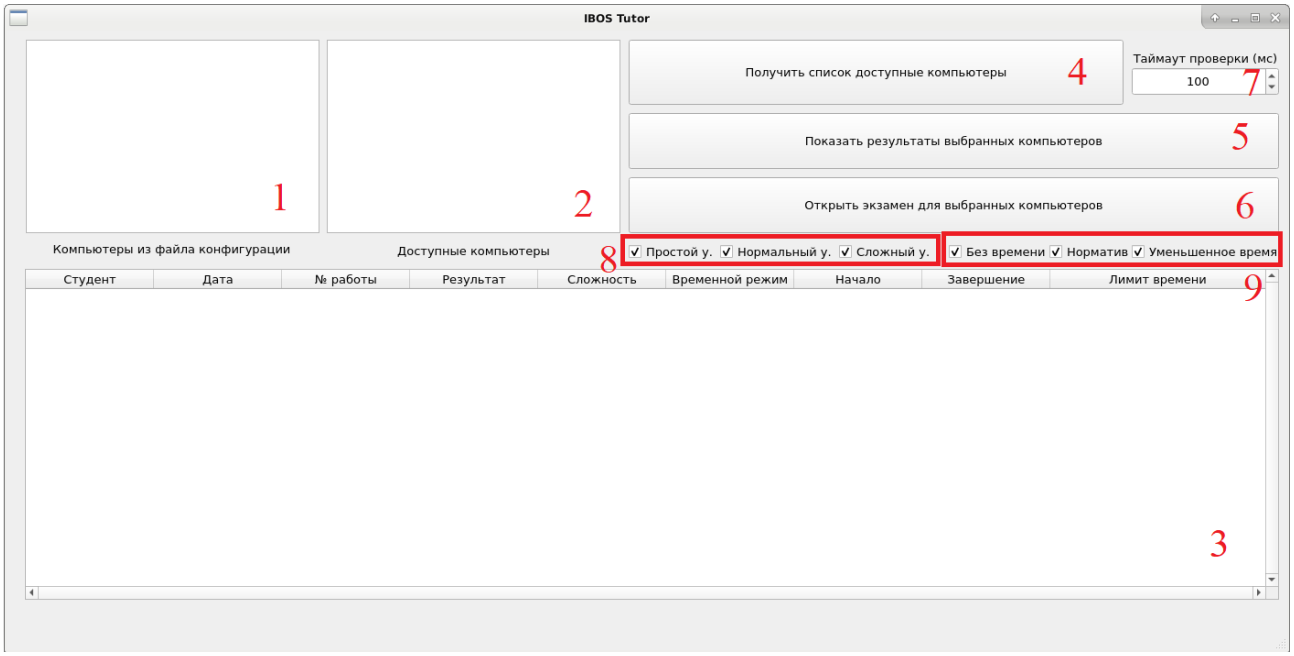

Рисунок 10 – Форма программы тьютора с пронумерованными элементами

В таблице 7 представлено описание элементов программы тьютора.

Таблица 7 – Описание элементов формы программы тьютора

№ элемента	Тип элемента	Описание
1	ЛистБокс (список)	В данный список загружаются IP-адреса компьютеров локальной сети из заготовленного файла.
2		В данный список загружаются IP-адреса компьютеров, на которых запущен тренажер.
3	Таблица	В данную таблицу будут записываться результаты выполнения практических работ с тренажеров.
4	Кнопка	Данная кнопка проверяет запуск тренажеров на компьютерах, которым соответствует выбранные в первом списке (элемент №1) IP-адреса. Если на компьютере был запущен тренажер, то IP-адрес компьютера будет добавлен в второй список (элемент №2).
5		Данная кнопка выводит результаты с тренажеров компьютеров, которым соотвествуют IP-адреса, выбранные в втором списке (элемент №2).

Продолжение таблицы 7

6	Кнопка	Данная кнопка выводит открывает режим экзамен для тренажеров компьютеров, которым соответствуют IP-адреса, выбранные в втором списке (элемент №2).
7	СпинБокс (счётчик)	Данный счётчик указывает, сколько времени программа тьютора будет пытаться подключиться к удаленному тренажеру.
8	Группа чекбоксов	Данная группа чекбоксов определяет, какие уровни сложности будут доступны для тренажера, в случае выбора режима «Экзамен».
9		Данная группа чекбоксов определяет, какие временные режимы будут доступны для тренажера, в случае выбора режима «Экзамен».

Была продумана концепция работы программы тьютора:

— существуют специальный файл tutorset.conf, в котором содержаться IP-адреса компьютеров в локальной сети;

— при запуске программы тьютора из вышеописанного файла считываются IP-адреса и заносятся в первый список (элемент №1);

— для дальнейшей работы необходимо выбрать IP-адреса компьютеров, на которых в данный момент запущены тренажеры, после чего нажать на кнопку «Получить список доступных компьютеров» (элемент №4);

— программа тьютора подключается по SSH к удаленным компьютерам и проверяет работу тренажра, в случае его корректной работы, текущий IP-адрес, имя пользователя, который запустил тренажер, и время запуска тренажера заносится в второй список в виде одной строки (элемент №2);

— подключение по SSH осуществляется по ключевой аутентификации, то есть ключи должны быть распределены по компьютерам заранее;

— для просмотра результатов с тренажеров удаленных компьютеров необходимо выбрать соответствующее записи в втором списке (элемент №2), а

после нажать на кнопку «Показать результаты выбранных компьютеров» (элемент №5);

— для открытия режима «Экзамен» на тренажерах удаленных компьютеров необходимо выбрать соответствующее записи в втором списке (элемент №2), а после нажать на кнопку «Открыть экзамен для выбранных компьютеров» (элемент №6);

— открытие режима «Экзамен» осуществляется путем копирования по scp на удаленный компьютер специального файла, который необходим тренажеру для открытия режима «Экзамен»;

— группы чекбоксов (элемент №8 и элемент №9) позволяют более гибко настроить режим «Экзамен».

Весь вышеописанный функционал реализуется модулем tutor.py, который подключает заранее созданный интерфейс в виде модуля FormsTutor.py, который был создан автоматически с помощью программы Qt Designer. Модуль tutor.py состоит из одного класса, который наследует программный итерфейс от класса, подключаемого модуля FormsTutor.py.

Основной класс, используемый в tutor.py, называется TutorApp. Весь функционал программы тьютора заключен в методах данного класса.

Ниже приведено описание используемых методов.

В конструкторе класса TutorApp производится базовая настройка приложения:

— подключение сигналов различных элементов графической формы к слотам Qt (методам класса TutorApp);

— настройка таблицы (элемента №3);

— чтение файла tutorset.conf и загрузка IP-адресов из него в первый список (элемент №1).

Метод ShowTrainer() проверяет работу тренажеров на компьютерах, IP-адреса которых были выбраны в первом списке (элемент №1), после чего

помещяет проверенные IP-адреса с именем пользователя, запустившего тренажер, и временем начала работы тренажера в второй спиоск (элемент №2).

Метод ShowResult() выводит результаты выполнения работы в режиме «Экзамен» для компьютеров, IP-адреса которых были выбраны в втором списке (элемент №2).

Метод ToGoExam() открывает режим «Экзамен» для компьютеров, IP-адреса которых были выбраны во втором списке.

Вспомогательные методы checkBoxMethod1() и checkBoxMethod2() проверяют, чтобы хотя бы один чекбокс в группах чекбоксов (элемент №8 и элемент №9) был выбран.

Полный код модуля tutor.py представлен в каталоге tutor репозитория MikhailNagaev/trainerIBOS.[7]

3.7 Установка и настройка тренажера

Для корректной работы тренажера требуется произвести его правильную установку. Для этого необходимо создать корректную среду функционирования с необходимыми каталогами и файлами, а также настроить соответствующие права для пользователей.

Тренажер без учета программы тьютора состоит из следующих каталогов и файлов:

— файл trainer.py, реализующий графический интерфейс тренажера;

— файл PyDocker.py, реализующий функционал по запуску и остановке контейнеров, а также формированию заданий;

— файл interfacePyQt.py, содержит графическую форму тренажера.

Данные файлы должны находиться в одном каталоге. Путь и наименование каталога не имеют значения.

Далее необходимо создать каталог /etc/trainerIBOS, в котором будут располагаться все конфигурационные файлы тренажера. В самой папке находятся два конфигурационных файла:

— check_images.conf, содержащий путь архива с эталонными Docker-образами и эталонные хэш-коды Docker-образов;

— port_time.conf, содержащий порт для работы промежуточных скриптов проверки и набор значений времени, отводимого для выполнения практических работ.

Кроме того, в каталоге /etc/trainerIBOS должны находиться три каталога:

— desc, содержащий в себе 6 подкаталогов с данными по каждой теме практической работы;

— script, содержащий в себе скрипты основных и промежуточных проверок;

— units, содержащий в себе шаблоны юнитов systemd.

Для каждой темы практической работы в подкаталогах каталога desc должно находиться четыре файла (<X> - номер практической работы по порядку):

— Lab<X>-code.conf, содержащий в себе таблицу соответствия кодов промежуточных проверок и выводимых сообщений;

— Lab<X>-Desc.conf, содержащий в себе краткое содержание работы;

— Lab<X>-Hint.conf, содержащий методические указания по выполнению работы;

— Lab<X>-names.conf, содержащий в себе наборы данных для формирования заданий.

На рисунке 11 представлена структура каталога /etc/trainerIBOS в виде дерева, которое в наглядной форме отображает содержимое каталога и его подкаталогов.

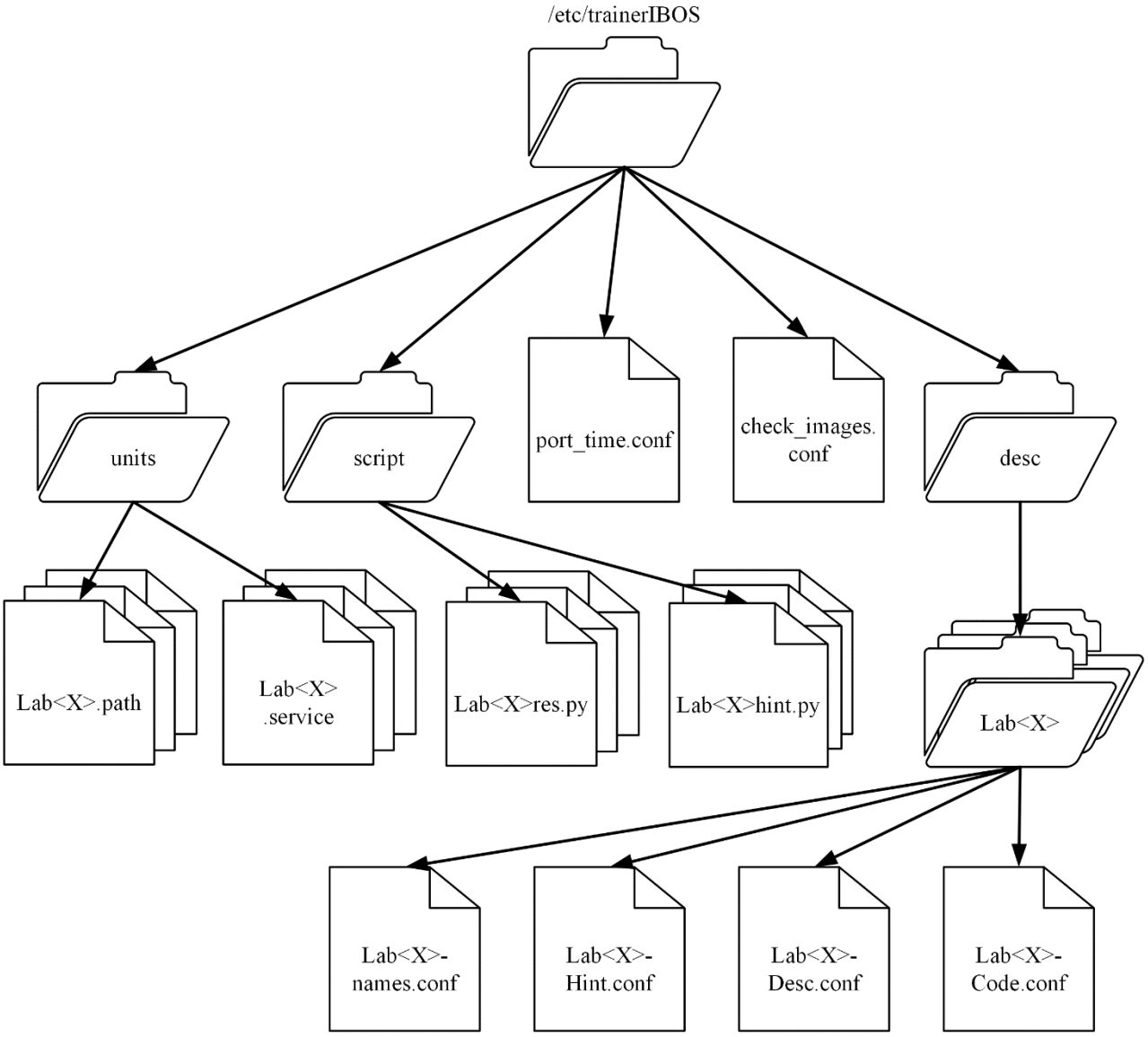

Рисунок 11 – Дерево каталога /etc/trainerIBOS

Необходимо в систему добавить расширения для языка программирования Python. А именно необходимо добавить библиотеку PyQt5, а также Docker SDK.

Далее необходимо настроить права доступа в операционной системе. Прежде всего необходимо создать группу trainerIBOS. После необходимо рекурсивно для всех подкаталогов и файлов назначить данную группу для каталога /etc/trainerIBOS.

Далее необходимо поместить юниты trainerright.path и trainerright.service в каталог /etc/systemd/system, после чего настроить включение данных юнитов. Они необходимы для корректной работы Docker-томов, так как при их создании доступ на их чтение имеет только root, поэтому данные юниты при создании

нового Docker-тома автоматически назначают ему необходимые права на чтения для других пользователей.

Далее необходимо добавить пользователей, которые будут работать с тренажером, в группы: trainerIBOS и docker. Группа docker будет создана автоматически при установке одноименного программного обеспечения.

Далее необходимо собрать Docker-образы из соответствующих Dockerfile. Но если в системе или в локальной сети присутствуют архив с уже готовыми Docker-образами, достаточно указать соответствующий путь к архиву в файл check_images.conf и запустить тренажер, после чего следовать инструкциям. Docker-образы будут восстановлены из архива.

Для запуска тренажера необходимо войти в систему пользователем, который был добавлен в группы trainerIBOS и docker. После чего необходимо перейти в каталог с файлом trainer.py, который необходимо выполнить командой:

```
python3 trainer.py
```

Все выше описываемые файлы находятся в репозитории MikhailNagaev/trainerIBOS.[7]

4 Тестирование тренажера

4.1 Тестовый план для программы «Тренажер ИБОС»

Данный тестовый план предполагает проведение тестирования программы «Тренажер ИБОС». Необходимость тестирования обусловлена тем, что приложение будет использоваться в образовательных целях, а так для контрольных мероприятий по проверке знаний обучаемых. Именно поэтому значительные ошибки, нарушающие полноценную работу тренажера должны отсутствовать, а незначительные ошибки должны быть сведены к минимуму.

Виды тестирования.

Исходя из архитектуры приложения, а также специфики решаемых задач, будут использованы следующие методы тестирования:

— функциональное тестирование (тестирование «белого ящика»), которое необходимо для проверки соответствия программного обеспечения, заявленным требованиям;

— модульное тестирование, необходимое для проверки отдельно взятых модулей;

— системное тестирование, необходимое для комплексной проверки функционала всего приложения;

— юзабилити тестирование, которое направлено на определение степени удобства использования и понятности использования приложения.

Операционные системы и сопутствующее программное обеспечение.

Для корректной работы программного обеспечения необходима операционная система Linux с ядром версии 4.19 или выше. Выбор операционной системы строится на том, что тренажер рассматривает обеспечение безопасности в UNIX-подобных системах. При этом используются виртуальные контейнеры, которые строятся на базе операционной системы хост-

системы, поэтому использование других операционных систем не позволит запустить необходимые контейнеры.

Приложение написано на языке Python (версия 3.0 или выше). В качестве графического фреймворка использовалось Qt5 (набор расширений для языка Python PyQt5).

Для работы с виртуальными контейнерами должна быть установлена система Docker.

Также для работы тренажера необходимо Docker SDK для управления системой Docker.

Функционал приложения.

Программа «Тренажер ИБОС» предназначена для выработки и проверки у обучаемых практических навыков по управлению параметрами подсистем обеспечения информационной безопасности в операционных системах.

Критерии начала тестирования:

— готовность тестовой платформы;

— законченность разработки функциональных модулей программы;

— готовность графической формы приложения;

— объединение всех модулей программы.

Критерии выхода из тестирования:

— весь набор тестов успешно пройден;

— отсутствие критических ошибок, нарушающих полноценное функционирование приложения;

— выдержано определенное время без появления новых ошибок.

Характеристики оборудования.

Программа должна тестироваться на следующей аппаратной среде:

— CPU Intel/Amd с 2 ядрами и выше;

— оперативная память 4 Гб и выше;

— свободное дисковое пространство не менее 20 Гб.

4.2 Тестовые сценарии

Тестовые сценарии были сгруппированы в наборы тестовых сценариев, по виду тестирования. В перечне наборов отсутствует набор тестов по юзабилити тестированию, так как степень удобства и понятности использования программой будет определена в процесса выполнения системного тестирования.

Набор тестовых сценариев №1 модульного тестирования.

Перед началом тестирования необходимо проверить корректность созданного набора Docker-образов, на основе которых будут построены контейнеры для выполнения практических работ. Необходимо проверить наличие всех необходимых конфигураций и файлов в системе контейнера, а также его безаварийную работу.

Для этого необходимо запустить контейнеры с сопутствующими Docker-элементами, после чего открыть терминал с командной оболочкой bash контейнера. Далее необходимо проверить корректность созданного контейнера на предмет возможности выполнения работы по соответствующей теме. В процессе проверки контейнер должен оставаться активным. Также все пункты, которые предусмотрены заданием на заданную практическую тему, должны быть реализуемы в данном контейнере.

В случае корректной настройки Docker-образов можно переходить к тестовым сценариям.

Тестовый сценарий №1.1 – Модуль PyDocker.py.

Описание: Данный модуль реализует подсистему запуска контейнеров и формирования задания. Модуль состоит из 6 функций запуска контейнеров и 6 функций завершения их работы. Проводится тестирование перечисленных функций.

Входные данные: уровень сложности, выбор режима тренировки.

Действия пользователя: выполнение функций запуска, выполнение функций завершения их работы.

Ожидаемый результат: запущенный контейнер с необходимой конфигурацией, а также сформированное задание для заданного уровня сложности. В случае выбора режима тренировки: создание и запуск модулей systemd, необходимых для промежуточных проверок. Созданные файлы основной информации задания и контрольных вопросов.

Тестовый сценарий №1.2 – Модуль trainer.py.

Описание: Данный модуль реализует пользовательский интерфейс тренажера.

Входные данные: отсутствуют.

Действия пользователя: выбор уровней сложности, выбор режима работы, выбор времени, отводимого на выполнение работы, выбор темы практической работы.

Ожидаемый результат: безаварийная работа, вывод необходимых информационных сведений, в соответствии с действиями пользователя.

Тестовый сценарий №1.3 – Набор скриптов основных и промежуточных проверок выполнения задания.

Описание: Набор скриптов, написанных на языке программирования Python, реализует систему автоматического контроля выполнения заданий и предназначен для проверки правильности выполнения заданий, сформированных с помощью подсистемы запуска контейнеров. Для тестирования данных скриптов необходим запуск соответствующих контейнеров с последующим выполнением необходимых заданий.

Входные данные: файл основной информации задания.

Действия пользователя: запуск необходимых контейнеров с помощью подсистемы запуска контейнеров и формирования задания, после чего производится ручное выполнение скриптов автоматического контроля правильности выполнения заданий.

Ожидаемый результат: от основных проверок: код завершения работы (0 – успешное выполнение, 1 – не успешное выполнение), текст, обозначающий все ошибки, совершенные при выполнении работы, процент правильности

выполнения работы. От промежуточных проверок: отправка кодов выполненного задания по сокетам на соответствующий порт.

Набор тестовых сценариев №2 функционального тестирования.

Тестовый сценарий №2.1 – Проверка наличия необходимых Docker-образов в системе.

Описание: Реализуется с помощью функции CheckImages(). Функция должна проверять наличие необходимых Docker-образов и сопоставлять их хэш-коды с эталонными.

Входные данные: файл с заготовленными эталонными хэш-кодами Docker-образов.

Действия пользователя: запуск тренажера (функция CheckImages() вызывается автоматически), проведение операций с Docker-образами: их удаление, а также модификация. Повторный запуск тренажера.

Ожидаемый результат: список имен Docker-образов, которые отсутствуют или хэш-коды которых не совпадают с эталонными.

Тестовый сценарий №2.2 – Проверка функции восстановления Docker-образов.

Описание: Реализуется с помощью функции LoadImages(). Функция загружает эталонные Docker-образы из заготовленного архива .tar.

Входные данные: файл-архив Docker-образов, файл с заготовленными эталонными хэш-кодами Docker-образов (в начале этого файла прописан путь в сети до архива Docker-образов).

Действия пользователя: запуск тренажера.

Ожидаемый результат: наличие в системе необходимых Docker-образов с правильным значением хэш-кодов.

Тестовый сценарий №2.3 – Проверка подсистемы формирования задания.

Описание: Проверяется подсистема запуска контейнеров и формирования задания. В соответствии с выбранным уровнем сложности должны генерироваться задания с случайными данными, которые читаются из специальных файлов.

Входные данные: файл с информацией необходимой для генерации задания.

Действия пользователя: первоначально производиться выбор темы практического задания и выбор уровня сложности, после чего необходимо нажать на кнопку «Начать выполнение задания».

Ожидаемый результат: сформированное задание в соответствии с уровнем сложности в информационном окне.

Тестовый сценарий №2.4 – Проверка подсистемы автоматического контроля выполнения заданий.

Описание: Проверяется подсистема автоматического контроля выполнений заданий, которая должна обеспечивать корректную проверку выполненных заданий.

Входные данные: файл с основной информацией задания (данный файл должен формироваться подсистемой запуска контейнеров).

Действия пользователя: с помощью подсистемы запуска контейнеров необходимо запустить контейнер, после чего в нем необходимо произвести действия в соответствии с сформированным заданием, после чего вручную выполняется скрипт проверки.

Ожидаемый результат: скрипты проверки должны возвращать результат работы, процент правильности выполнения, а также текст, иллюстрирующий совершенные ошибки.

Тестовый сценарий №2.5 – Проверка работоспособности режимов «Тренировка» и «Самоконтроль».

Описание: Проверяется функционал двух основных режимов работы тренажера. Завершение работы в данных режимах производится путем нажатия на кнопку «Завершить выполнение работы».

Входные данные: конфигурационные файлы, содержащие методические указания к выполнению, коды сообщений, а также информацию необходимую для формирования задания.

Действия пользователя: первоначально производиться выбор темы практического задания, выбор уровня сложности и выбор режима работы, после чего необходимо нажать на кнопку «Начать выполнение задания».

Ожидаемый результат: корректная работоспособность основной программы в различных режимах. В режиме «Тренировка» для каждой из практических тем должны выводиться соответствующие методические указания, а также в течение выполнения работы должны выводиться информационные сообщения, которые оповещают пользователя о завершении пунктов задания. После окончания работы должны выводиться результат работы, процент правильности выполнения, а также текст, иллюстрирующий совершенные ошибки. В режиме «Самоконтроль» реализуется только функционал основной проверки выполнения задания.

Тестовый сценарий №2.6 – Проверка работы таймеров.

Описание: Проверяется работоспособность функционала программы по завершению выполнения работы при завершении таймера, подсчитывающего время, отведенного на выполнение работы.

Входные данные: конфигурационный файл с значениями времени, отводимого для выполнения работы.

Действия пользователя: запуск контейнеров из тренажера, совершение в них некоторых действий, предусмотренных заданием, ожидание завершения таймера.

Ожидаемый результат: безаварийная работа программы, остановка и удаление виртуальных контейнеров, а также сопутствующих Docker-элементов, вывод информации о завершении работы на экран.

Тестовый сценарий №2.7 – Проверка функционала обеспечивающего работу с программой тьютора.

Описание: Проверяется работоспособность функционала, который необходим для корректного взаимодействия с программой тьютора. К данному функционалу относится создание файла, показывающего то, что в данный

момент тренажер запущен, а также детектирование файла, в котором передается информация для открытия режима «Экзамен».

Входные данные: отсутствуют.

Действия пользователя: запуск программы.

Ожидаемый результат: создание в каталоге /tmp/trainerIBOS файла runningtrainer.conf, детектирование файла exam.conf в том же каталоге. В случае корректного шаблона данного файла, должен открыться режим «Экзамен».

Тестовый сценарий №2.8 – Проверка работоспособности режима «Экзамен».

Описание: Проверяется функционал режима работы тренажера. Завершение работы в данном режиме производится путем нажатия на кнопку «Завершить выполнение работы».

Входные данные: конфигурационные файлы, содержащие методические указания к выполнению, коды сообщений, а также информацию необходимую для формирования задания.

Действия пользователя: первоначально производиться выбор темы практического задания, выбор уровня сложности и выбор режима работы, после чего необходимо нажать на кнопку «Начать выполнение задания».

Ожидаемый результат: корректная работоспособность основной программы. После окончания работы должны выводиться результат работы, процент правильности выполнения, а также текст, иллюстрирующий совершенные ошибки. Кроме того, в конец файла result.csv, находящейся в каталоге /etc/trainerIBOS, должна записаться информация о результатах экзамена.

Тестовый сценарий №2.9 – Проверка функционала по заданию контрольных вопросов.

Описание: Проверяется функционал программы по заданию контрольных вопросов и автоматической проверки ответов на них. Данный функционал реализуется функцией TimeToAsk().

Входные данные: файл с сформированными вопросами (данный файл должен быть сформирован подсистемой запуска контейнеров).

Действия пользователя: завершить выполнение практической работы с помощью нажатия на соответствующую кнопку.

Ожидаемый результат: отображение вопросов из файла с возможными вариантами ответов. Кроме того, функция должна вернуть результат тестирования, процент правильности выполнения тестов, а также текст, иллюстрирующий ошибки.

Набор тестовых сценариев №3 системного тестирования.

Тестовый сценарий №3.1 – Системное тестирование.

Описание: тестирование всего приложения, то есть симуляция работы за приложением.

Входные данные: отсутствуют.

Действия пользователя: работа с тренажером.

Ожидаемый результат: безаварийная и безошибочная работа программы.

Тестовый сценарий №3.2 – Тестирование с заведомо неверным набором Docker-элементов.

Описание: производится тестирование с заранее уже созданными Docker-элементами, из-за которых могут возникнуть ошибки при их использовании. К Docker-элементам относятся: Docker-томы и Docker-сети.

Входные данные: отсутствуют.

Действия пользователя: работа с тренажером.

Ожидаемый результат: безаварийная и безошибочная работа.

4.3 Результаты тестирования

Результаты тестирования представлены в таблице 8.

Таблица 8 – Результаты тестирования

Тестовый набор	№ тестового сценария	Результат (совпадение с ожидаемым результатом)	Примечание
Модульное тестирование	№1.1	Успех	-
	№1.2.1	Ошибка	Было выявлено, что при изменении темы не изменяется время, отводимое на выполнение. Данная ошибка была исправлена
	№1.2.2	Успех	Повторное тестирование.
	№1.3	Успех	-
Функциональное тестирование	№2.1	Успех	-
	№2.2	Успех	-
	№2.3.1	Ошибка	Выводилось некорректное описание в теме по управлению сетевой подсистемы. А именно название базовой и основной цепочки были перепутаны местами. Данная ошибка была исправлена.
	№2.3.2	Успех	Повторное тестирование.
	№2.4	Успех	-
	№2.5	Успех	-
	№2.6.1	Ошибка	Было выявлено, что при досрочном завершении работы (с помощью соответствующей кнопки),
	№2.6.1	Ошибка	таймер времени продолжает работать при прохождении контрольного тестирования. Данная ошибка была исправлена.
	№2.6.2	Успех	Повторное тестирование.
	№2.7	Успех	-
	№2.8	Успех	-
	№2.9.1	Ошибка	Было выявлено, что текст, иллюстрирующий ошибки тестирования, формировался неверно.
	№2.9.1	Успех	Повторное тестирование
Системное тестирование	№3.1	Успех	-
	№2.2	Успех	-

Совместно с системным тестированием проводилось юзабилити тестирование, из которого было выявлено, что программа является удобной в использовании, так как работа с графической формой интуитивно понятна.

4.4 Заключение по результатам тестирования

Критических ошибок, нарушающих безаварийную работу программы, обнаружено не было. Были обнаружены некритичные ошибки, которые были исправлены, после чего было проведено повторное тестирование. После чего все тесты были успешно пройдены. В результате тестирование завершено в соответствии с критериями выхода из тестирования.

5 Вредные психофизические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Работа специалиста по защите информации тесно связана с работой за компьютером, а также работой с бумагами. При этом специалисты постоянно находятся в напряженном состоянии, так как приходиться решать большое количество важнейших вопросов в области безопасности. Это современная необходимость, так как от работы такого специалиста зависит функционирование всех бизнес процессов организации. Но всё это оказывает вредные воздействия на организм человека. Особенно стоит отметить болезни и заболевания, связанные с сердечно-сосудистой системой, которые являются наиболее скрытыми, но от этого не менее опасными для человека.

К самым распространённым заболеваниям сердечно-сосудистой системы относятся:

— инсульт;
— ишемическая болезнь сердца (инфаркт);
— артериальная гипертензия;
— гипертония;
— сердечная недостаточность.

Стоит отметить, что подобные заболевания являются основной причиной смертности людей в мире, так как каждый год именно от болезней сердечно-сосудистой системы умирает большее количество людей, чем от любых других болезней. Именно поэтому необходимо проводить профилактические мероприятия, направленные на уменьшение вероятности заболеваниями сердечно-сосудистой системы.

Прежде всего необходимо определить вредные психофизические факторы, которые влияют на сердечно-сосудистую систему специалиста по информационной безопасности. Такие факторы можно разделить на две

категории: устранимые и неустранимые. Неустранимые факторы – это факторы, которые свойственны конкретному человеку и которые нельзя изменить. Устранимые факторы – это факторы, которые приобретаются человеком в течение жизнедеятельности, а так же которые можно изменить и устранить, приняв необходимые меры.

Неустранимые факторы относится к любому человеку, в том числе и к специалисту по защите информации, и их следует перечислить:

— возраст, так как после 65 лет риски развития заболевания сердечно-сосудистой системы достаточно увеличивается;

— пол, так как мужчины более подвержены подобному виду заболеваний;

— наследственность, так как наследственные качества могут увеличить риск подобных заболеваний.

К устранимым факторам, свойственным для специалистов по защите информации, относятся:

— избыточный вес, который не только увеличивает вероятность заболевания болезнями сердечно-сосудистой системы, но и негативно влияет на процесс выздоровления от уже имеющейся болезни;

— низкая физическая активность, которая ведет к уменьшению всех физических параметров организма, что в свою очередь увеличивает риск сердечно-сосудистых заболеваний в два раза;

— неправильное питание, так как переизбыток или, наоборот, недостаток в рационе насыщенных животных жиров, которые имеют высокое содержание холестерина, ведет к развитию сердечно-сосудистых заболеваний;

— стресс, так как в таком состоянии организм человека функционирует неполноценно, в частности это касается: сосудов и обмена веществ, что также приводит к увеличению вероятности сердечно-сосудистых заболеваний.

Перечисленные выше факторы могут быть свойственными для специалистов по защите информации, так как их работа предполагает

умственную деятельность, но малую физическую активность, что также ведет к избыточному весу. При этом специалисты по защите информации, как уже говорилось ранее, находятся в постоянном напряжении, что приводит к стрессовым ситуациям, которые могут перейти в хронический стресс. А недостаток времени часто провоцирует на неправильное питание.

Исходя из вредных психофизических факторов, которые сказываются на развитии сердечно-сосудистых заболеваний, можно сформировать перечень необходимых профилактических методов и средств физической культуры, снижающих их воздействие:

— ограничение потребления алкоголя, так как алкоголь при превышении нормы его потребления несет сугубо отрицательное воздействие на сердечно-сосудистую систему человека;

— контроль веса тела, так как это позволит поддерживать полноценную функциональность организма:

— правильное питание;

— физические нагрузки;

— контроль стрессовых ситуаций, который заключается в уменьшении переживаний по рабочим и бытовым вопросам жизни человека.

Также стоит отметить, что правильно питание должно заключаться не только в отказе от вредной пищи, но и в потреблении всего необходимого для полноценного функционирования организма человека. К таким продуктам относятся: мясо, рыба, овощи и фрукты. Такой подход к питанию также позволит снизить вес, что в свою очередь позитивно скажется на профилактике сердечно-сосудистых заболевания.

Кроме того, необходимо с особой ответственностью отнестись к физическим нагрузкам, так как их необходимый объем позволит не только снизить риски сердечно-сосудистых заболеваний, но и улучшить состояние организма в целом. Для лучшей организации физических нагрузок необходимо составить план занятий: например, посещать тренажерный зал или бассейн по

определённым дням недели, совершать утренние или вечерние пробежки в парке по выходным. Также, по возможности, стоит больше передвигаться самостоятельно, без помощи автотранспорта.

Особое внимание необходимо уделить контролю стрессовых ситуаций. Прежде всего необходимо понимать, что хроническое недосыпание, а также постоянное психологическое напряжение приводят к появлению и усугублению стресса. Недосыпания необходимо избегать путем реорганизация своего времени и графика дня. От психологического напряжения избавиться нельзя, что связано с спецификой работы специалиста по защите информации. Но стоит применять различные меры и средства по психологической релаксации, например, после работы. К таким мерам и средствам релаксации можно отнести: прогулку в парке, физические нагрузки, хобби.

Данные методы и средства в должной мере позволят уменьшить риски заболевания сердечно-сосудистыми заболеваниями. Но стоит понимать, что кроме самостоятельной профилактики сердечно-сосудистых заболеваний, также необходимо постоянно проходить медицинские осмотры, которые позволят не упустить начавшую развитие болезнь.

Заключение

В результате был разработан тренажер по информационной безопасности на базе контейнеров. Данный тренажер предназначен для выработки и проверки у обучаемых практических навыков по управлению параметрами подсистем обеспечения информационной безопасности в операционных системах.

Тренажер обеспечивает обучение по 6 темам:

— управление пользователями и группами в операционной системе;

— разграничение прав доступа к локальным ресурсам средствами операционной системы;

— управление системными сервисами операционной системы;

— управление сетевыми интерфейсами и сетевым экраном в операционной системе;

— работа с системными журналами в операционной системе;

— настройка удаленного управления на базе сетевого протокола ssh.

Укомплектованные по темам методические материалы, задания, сформированные с использованием генератора псевдослучайной последовательности чисел, аналитика неправильных действий, проверка теоретической базы, а также возможности гибкой настройки тренажера позволят начинающему специалисту быстро освоиться в необходимой теме по информационной безопасности. Кроме того, в процессе выполнения практических работ формируются необходимые навыки работы в терминалах операционных систем семейства Linux. Именно поэтому тренажер по информационной безопасности – это необходимое решение при обучении информационной безопасности, так как оно сочетает в себе простоту и эффективность получения и закрепления практических навыков.

Таким образом, дипломный проект был выполнен в полном объеме в соответствии с техническим заданием.

Список используемых источников

1 О.И. Шелухин, А.М. Тенякшев, А.В. Осин, Моделирование информационных систем, учебное пособие. Москва: изд. «САЙНС-ПРЕСС», 2005 г. с.327. ISBN 5-93108-072-4.

2 Виртуализация // Википедия. [2021]. Дата обновления: 03.01.2021. URL: https://ru.wikipedia.org/?curid=163014&oldid=111475382 (дата обращения: 12.02.2021).

3 М. Тим Джонс, Виртуальный Linux. Обзор методов виртуализации, архитектур и реализаций // IBM [2007]. Дата обновления: 28.04.2007. URL: https://ibm.com/developerworks/ru/library/l-linuxvirt/ (дата обращения 12.02.2021).

4 Виртуальная машина // Википедия. [2020]. Дата обновления: 13.07.2020. URL: https://ru.wikipedia.org/?curid=65682&oldid=108168203 (дата обращения: 12.02.2021).

5 Аппаратная виртуализация // Википедия. [2020]. Дата обновления: 13.12.2020. URL: https://ru.wikipedia.org/?curid=942289&oldid=111050875 (дата обращения: 12.02.2021).

6 Контейнеризация // Википедия. [2021]. Дата обновления: 01.02.2021. URL: https://ru.wikipedia.org/?curid=2504275&oldid=112114229 (дата обращения: 12.02.2021).

7 trainerIBOS. Репозиторий GitHub // дата обновления 26.05.2021 URL: https://github.com/MikhailNagaev/trainerIBOS (дата обращения 26.05.2021).

8 Оптимизация образов Docker // Хабр [2014]. Дата обновления 28.08.2014. URL: https://habr.com/ru/post/234829/ (дата обращения: 26.05.2021).

9 Use Volumes // Docker Docs. URL: https://docs.docker.com/storage/volumes (дата обращения: 26.05.2021).

.