Утверждаю

Руководитель практики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, печать предприятия)

«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии (издательско-полиграфический комплекс)»

ОТЧЁТ

по производственной технологической практике

в БГТУ 26.07.2023 – 30.06.2023 и 07.08.2023 – 25.08.2023

(наименование предприятия, сроки практики)

Исполнитель

Студент 3 курса 1 группы \_\_\_\_\_\_\_\_\_\_\_ Шкабров Д.С.

(подпись, дата) (Ф.И.О.)

Руководитель практики

от предприятия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, печать предприятия) (подпись, дата) (Ф.И.О.)

Руководитель практики

от университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, уч. звание) (подпись, дата) (Ф.И.О.)

Отчёт защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc140825208)

[1. Знакомство с программным обеспечением, используемым на предприятии 6](#_Toc140825209)

[2. Задания на производственную практику 7](#_Toc140825210)

[2.1 Разработать приложения, выбирающие арбитра. 7](#_Toc140825211)

[2.2 Разработать приложения устанавливающие кольцевое соединение. 7](#_Toc140825212)

[2.3 Внесите изменение в приложение, описанное в п. 2.2. 7](#_Toc140825213)

[2.4 Разработайте приложение, создающее контрольную точку системы, описанной в п. 2.3. 7](#_Toc140825214)

[3. Выполнение приложений 8](#_Toc140825215)

[3.1 Разработка приложения, выбирающего арбитра 8](#_Toc140825216)

[3.2 Разработка приложений, устанавливающих кольцевое соединение 8](#_Toc140825217)

3.3 Внесение изменений в приложение, добавление подсчета прошедших сообщений в каждом узле

[3.4 Разработка приложения, создающего контрольную точку системы, устанавливающей кольцевое соединение 9](#_Toc140825218)

[ЗАКЛЮЧЕНИЕ 11](#_Toc140825219)

[Список используемых источников 12](#_Toc140825220)

# **ВВЕДЕНИЕ**

Цель производственной практики – закрепить и углубить знания, полученные студентами в процессе теоретического обучения, привить необходимые умения и навыки для работы по избранной специальности, приобрести первоначальный профессиональный опыт. В процессе прохождения практики студенты учатся самостоятельно отбирать и систематизировать информацию в рамках поставленных перед ними задач; применять полученные знания на практике; изучать технологию и оборудование, используемые в рамках конкретного производства; развивать навыки работы в коллективе; осуществлять самоконтроль. Это позволяет улучшить умение находить решения проблем, развить критическое мышление, работать в команде и следовать инструкциям. Производственная практика также позволяет студентам определить свои сильные стороны и интересы в конкретной области деятельности.

При прохождении практики были поставлены следующие задачи:

- ознакомление с организационной структурой предприятия;

- ознакомление с ПО, с которым работаем на предприятии;

- выполнение индивидуального задания.

# **Знакомство с программным обеспечением, используемым на предприятии**

Для выполнения практики, одним из требований, является изучение программного обеспечения X2Go и работа с ним для выполнения индивидуальных заданий на платформе Linux.

X2Go - это бесплатное программное обеспечение для удаленного рабочего стола с открытым исходным кодом, которое позволяет пользователям подключаться к рабочему столу Linux с компьютера Windows по протоколу NX.

Технология NX, широко известная как NX, представляет собой программное приложение для удаленного доступа, совместного использования рабочего стола, виртуального рабочего стола (только в Linux) и передачи файлов между компьютерами. NX сжимает данные, чтобы минимизировать объем передаваемых данных, а кэширует данные, чтобы поддерживать максимально возможный отклик сеанса. Например, при первом открытии меню это может занять несколько секунд, но впоследствии открывается почти мгновенно. NX быстрее, чем его предшественники, поскольку он устраняет большую часть X круговых обходов,

X2Go использует сжатие данных и оптимизированную передачу графических данных для обеспечения быстрого и отзывчивого опыта удаленного доступа. Он поддерживает множество операционных систем, включая Linux, Windows и MacOS.

X2Go обычно используется системными администраторами, разработчиками и другими пользователями, которые работают с удаленными серверами и им требуются высокая производительность и надежность удаленного доступа к графическим приложениям. Он предоставляет безопасный и удобный способ работы с удаленными рабочими столами, уменьшая задержки и сохраняя качество изображения.

Для подключения к удаленному Linux-компьютеру с помощью X2Go нужно было выполнить следующие шаги:

- установить X2Go на клиентском компьютере. Для этого можно скачать соответствующий пакет для своей операционной системы с официального сайта X2Go.

- запустить X2Go на клиентском компьютере и создать новую сессию. При создании новой сессии, нужно указать IP-адрес или доменное имя удаленного Linux-компьютера, выбрать соответствующий тип сессии, а также указать имя пользователя и пароль для входа.

- подключиться к удаленному Linux-компьютеру, запустив созданную сессию.

Важно отметить, чтобы было успешное подключение необходимо, чтобы на обоих компьютерах были настроены сетевые параметры и открыты нужные порты для передачи данных.

# **2. Задания на производственную практику**

## **2.1 Разработать приложения, выбирающие арбитра.**

Все работающие (и вновь подключающиеся) приложения должны выбрать одно из приложений, который называется арбитром. Такое приложение остается арбитром пока оно не завершится. Если арбитр завершил свою работу, процедура выбора запускается снова.

Система приложений не может существовать без арбитра более 60 секунд. Все приложения выдают каждые 60 сек:

TIME: время, PID: идентификатор процесса, ARBITER: PID арбитра.

## **Разработать приложения, устанавливающие кольцевое соединение.**

Все работающие (и вновь подключающиеся) приложения должны установить между собой кольцевое соединение. Кольцевое соединение - попарное соединение (все равно каким способом) приложений (узлов), которое позволяет передавать сообщение по замкнутой цепочке (любым способом). Если одно из приложений завершило свою работу соединение восстанавливается не менее чем за 60 сек. По кольцу сообщение передается с задержкой по 3 сек в каждом узле кольца. B качестве сообщения передается счетчик, который увеличивается только в одном узле (инициаторе соединения). Если инициатор соединения отключился, счетчик увеличивает любой из узлов (берет на себя роль инициатора). При отправке сообщения узел выдает на консоль:

PID: идентификатор процесса-INIT (только для инициатора), MARC: сообщение (счетчик).

## **Внесите изменение в приложение, описанное в п. 2.2.**

Счетчик увеличивается на 1 в каждом узле и отдельно подсчитывается количество прошедших через узел сообщений (у постоянно работающего процесса значение счетчиков совпадают).

При отправке сообщения узел выдает на консоль. PID: идентификатор процесса, MARC: значение счетчика, MESS: значение счетчика сообщений.

## **Разработайте приложение, создающее контрольную точку системы, описанной в п. 2.3.**

Приложение должно работать в двух режимах (указывается параметром): создать контрольную точку, запустить систему с контрольной точки.

# **Выполнение приложений**

## **Разработка приложения, выбирающего арбитра**

Реализованная программа является примером использования механизма обработки сигналов с использованием функции sigaction и самих сигналов SIGUSR1 и SIGUSR2 для реализации определения арбитра между несколькими процессами.

Сигналы SIGUSR1 и SIGUSR2 – это пользовательские сигналы в UNIX-подобных операционных системах. Они нужны для использования программистами в передаче пользовательских сигналов между процессами. Эти сигналы не имеют определенного назначения внутри операционной системы и могут быть использованы для реализации собственных функций и протоколов взаимодействия между процессами.

В функции sighandler сигналы SIGUSR1 и SIGUSR2 обрабатываются таким образом:

- если получен сигнал SIGUSR1, то устанавливается флаг arbitter\_answered в значение false, что значит определение нового арбитра, если текущий процесс не является арбитром.

- при получении сигнала SIGUSR2 устанавливается значение переменной arbitter в значение, переданное в info -> si\_pid, а также tot устанавливается флаг arbitter\_answered в значение true.

В функции FindArbitter сигнал SIGUSR1 отправляется всем процессам (кроме текущего) для определения арбитра. Если получен ответ сигналом SIGUSR2, то текущий процесс выбирается арбитром путем установки значения arbitter равным pid. Этот процесс повторяется в течение заданного времени.

В цикле while в функции main проверяется, не потерялся ли арбитр. Если текущий процесс не является арбитром и прошло определенное количество времени (заданное константой MICRO), то арбитр считается потерянным и происходит поиск нового арбитра с помощью вызова функции FindArbitter. Время работы программы также выводится на экран.

Таким образом, программа обеспечивает выбор арбитра между несколькими процессами с помощью механизма обработки сигналов.

## **Разработка приложений, устанавливающих кольцевое соединение**

Приложение представляет собой сервер, который обменивается сообщениями между несколькими клиентами, как можно сказать, образует кольцо, где первый обменивается со вторым, второй с третьим и так до того, пока не дойдет до последнего с первым, замыкая и образуя кольцо. Приложение реализовано с использованием сокетов и применения протокола UDP.

При запуске программы каждый узел получает свой IP-адрес в формате "127.0.0.x", где x - последний октет IP-адреса. Затем каждый узел пытается привязать сокет к своему IP-адресу и порту 2222. Если привязка не удалась (возможно, этот адрес уже занят), узел инкрементирует последний октет IP-адреса на единицу и пытается привязаться снова. Когда привязка происходит успешно, узел определяет адрес следующего узла в кольце как IP-адрес текущего узла с инкрементированным последним октетом на единицу.

Дальше каждый узел создает сокет и начинает работу в бесконечном цикле. В цикле узел инкрементирует значение счетчика сообщений и отправляет его следующему узлу в кольце. Ожидая подтверждения о доставке сообщения, узел ждет и принимает сообщение от предыдущего узла и обновляет свое значение счетчика, если полученное сообщение больше его текущего значения. Если сообщение не является числом или меньше текущего значения счетчика, узел игнорирует его. Затем узел передает полученное сообщение следующему узлу в кольце и получает новое сообщение от него. Таким образом, процесс повторяется.

Вот так, каждый узел в кольце передает сообщение следующему узлу, обновляя значение своего счетчика, принимает сообщение от предыдущего узла и обновляет свое значение счетчика на основе полученного сообщения. По итогу каждый узел имеет доступ к актуальному значению счетчика, и сообщения в кольце передаются последовательно один за другим.

В процессе выполнения приложения выводятся отправленные сообщения, еще PID (идентификатор процесса) инициатора, поскольку счетчик увеличивает только экземпляр с начальным адресом, что позволяет определить PID инициатора.

Само приложение выполняет цикл повторения операций каждые 3 секунды, обеспечивая постоянный обмен сообщениями.

* 1. **Внесение изменений в приложение, добавление подсчета прошедших сообщений в каждом узле**

К предыдущему приложению вносятся изменения в виде счетчика для каждого узла, который увеличивается на 1 с отдельным подсчетом количества прошедших через узел сообщений. Также добавляется вывод в консоль.

## **Разработка приложения, создающего контрольную точку системы, устанавливающей кольцевое соединение**

Представленное приложение было доработано на основе исходного приложения, описанного в предыдущем пункте.

Контрольная точка — это метод, используемый в распределенных системах для фиксации состояния системы в определенный момент времени, позволяющий восстанавливаться после сбоев путем восстановления системы до предыдущего состояния.

Если пользователь выбирает установку контрольных точек (k == 1), программа входит в цикл, который обрабатывает процесс установки контрольных точек. Он инициализирует сокеты, привязывается к портам и устанавливает связь между процессами. Он использует разные стратегии в зависимости от того, является ли процесс инициатором или нет.

Если пользователь выбирает получение контрольной точки (k == 0), программа входит в другой цикл, который обрабатывает процесс получения информации о контрольной точке и, возможно, отправки подтверждений.

Программа использует файловый ввод-вывод для отслеживания процесса установления контрольных точек и сообщений, которыми обмениваются процессы.

В целом, этот код реализует распределенный механизм контрольных точек с использованием сокетов UDP, где узлы обмениваются данными для обмена данными контрольных точек и поддерживают синхронизированное состояние во всей распределенной системе.

# **ЗАКЛЮЧЕНИЕ**

Во время производственной практики, я получил опыт и знания, которые мне могут помочь в дальнейшей карьере. Работа в производственной среде предоставила мне поработать с реальными проектами и задачами, а также я научился работать в команде и эффективно распределять задачи. Я ознакомился с различными техническими процессами и оборудованием, что позволило мне применять теоретические знания на практике. По итогу работы были решены все задачи, поставленные в процессе производственной практики.

# **Список используемых источников**

1. Программа на Linux. Компилятор g++ [Электронный ресурс]. Режим доступа: https://metanit.com/cpp/tutorial/1.3.php.
2. Using C++ on Linux in VS Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs/cpp/config-linux.
3. Электронный ресурс StackOverflow. Режим доступа: https://ru.stackoverflow.com/.