МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Отчет о выполнении летней учебной практики

Работа в виртуальной среде Linux.

Студент,
группы 5130201/40003 _____ Четвергов И.С

Доцент

_____ Глазунов Н.В

«____»_____ 20__ г.

Санкт-Петербург, 2025

Содержание

1 Введение

В данной работе рассматриваются процессы установки и настройки виртуальной среды для разработки программного обеспечения на Unix-подобной системе **Linux**. Виртуальная машина на базе **Debian 12** развёрнута без графической оболочки, что позволяет сосредоточиться на работе с командным интерпретатором и изучении системных механизмов.

Основная цель – изучение принципов работы операционной системы, взаимодействие с файловой системой и исследование динамических и статических библиотек. В рамках исследования рассматриваются методы оптимизации программного кода и анализ влияния различных компиляторных флагов на производительность.

Практическая часть включает в себя настройку доступа к виртуальной машине, работу с основными командами интерпретатора, создание статических и динамических библиотек, а также разработку программ для работы с бинарными файлами и псевдографикой. Особое внимание уделяется исследованию зависимостей библиотек, принципам динамической загрузки и взаимодействию с пользователем через командную строку.

Кроме того, проводится сравнительный анализ существующих методов работы с бинарными данными, позволяющий выявить недостатки базовых решений и предложить альтернативные подходы, учитывающие эффективность и удобство взаимодействия с файловыми структурами.

2 Постановка задачи

Цель данной работы — изучение принципов работы операционной системы **Linux**, настройка виртуальной среды, исследование динамических и статических библиотек, а также работа с двоичными файлами и псевдографическим интерфейсом.

Для достижения этой цели необходимо выполнить следующие задачи:

- Установить и настроить виртуальную машину на базе **Debian 12** в среде **VirtualBox**.
- Настроить доступ к системе через SSH и проброс портов для взаимодействия с виртуальной машиной.
- Освоить работу с командным интерпретатором, включая редактирование команд, автоматическое дополнение и работу с историей команд.
- Исследовать принципы работы динамических и статических библиотек, разработать собственные библиотеки и провести тестирование их производительности.
- Разработать программу для работы с бинарными файлами, создать структуру заголовка базы данных и реализовать механизм обработки данных.
- Исследовать алгоритмы работы с псевдографикой, разработать интерфейс с использованием библиотеки **ncurses** и протестировать корректность работы программы.

Результаты работы должны включать анализ влияния оптимизационных флагов на производительность, исследование зависимостей библиотек и тестирование взаимодействия с пользователем. Полученные данные позволят оценить эффективность примененных подходов и выявить возможные пути их оптимизации.

3 Особенности реализации

Проект организован в соответствии с принципами модульного подхода и разделения функциональности. Основные компоненты структурированы следующим образом:

- **analysis** – скрипты для обработки данных и визуализации результатов.
- **assets** – вспомогательные файлы, включая зависимости, результаты тестов и базу данных.
- **include** – заголовочные файлы, определяющие API для работы с ключевыми модулями проекта.
- **libs** – статические и динамические библиотеки, используемые для тестирования производительности.
- **src** – исходные файлы, реализующие алгоритмы обработки данных и пользовательский интерфейс.
- **tests** – набор тестов для проверки корректности работы программ и анализа производительности.

Сборка проекта выполняется с помощью **CMake**, что упрощает управление зависимостями и настройку компиляции. Включены механизмы статической и динамической линковки, позволяющие провести сравнительный анализ эффективности различных методов использования библиотек.

3.1 Сравнительное тестирование

Одним из ключевых аспектов реализации является исследование производительности различных подходов к поиску минимального элемента массива. В рамках проекта организованы тесты для:

- статической линковки (`libsearch_min_static.a`);
- динамической линковки (`libsearch_min_shared.so`);
- загрузки библиотек во время выполнения (`dlopen`).

Для проведения тестов используется цель `run_all_tests`, запускающая программу с различными оптимизационными флагами компиляции (`O0, O1, O2, Os`) и записывающая результаты в файл `results.txt`. Сравнение времени выполнения осуществляется с помощью утилиты `time`, а визуализация данных выполняется скриптом `graph.py`.

Дополнительно протестированы различные методы загрузки библиотек, что позволило оценить влияние динамической линковки и механизма `dlopen` на скорость работы программы. Такой подход обеспечивает гибкость тестирования и выбор оптимального решения для дальнейшего использования.

3.2 Работа с базой данных

В проекте используется бинарный файл `students.db`, содержащий данные о студентах. Файл реализован как бинарная база данных с заголовком и записями, представленными в виде структуры `STUDENT`.

Структура базы данных

Заголовок файла представлен структурой `DB`:

- **Сигнатура** (4 байта) – содержит первые 4 буквы фамилии студента.
- **Номер транзакции** (4 байта) – увеличивается при каждом обращении к базе.
- **Количество записей** (4 байта) – число сохранённых структур в файле.
- **Контрольная сумма** (4 байта) – вычисляется по алгоритму `CRC-32` для проверки целостности данных.

Каждая запись представлена **структурой** `STUDENT`:

- **Фамилия и инициалы** – строка фиксированной длины (`MAX_NAME_LEN`).
- **Номер группы** – строка фиксированной длины (`MAX_GROUP_LEN`).
- **Успеваемость** – массив из `MAX_MARKS` целых чисел, содержащий оценки студента.
- **Количество оценок** – число записанных значений успеваемости.

Методы работы с базой

Для работы с файлом реализованы функции:

- `save_DB()` – сохраняет заголовок и данные студентов в бинарный файл.
- `load_DB()` – загружает данные из файла в оперативную память.

Запись и чтение данных реализованы в бинарном формате, что минимизирует размер файла и ускоряет доступ к информации. Контроль целостности обеспечивается обновлением номера транзакции и пересчётом `CRC-32`.

Такой подход обеспечивает эффективное хранение и обработку данных, а также удобный механизм поиска студентов по заданным критериям.

4 Результаты работы и эксперименты

В ходе работы проведено тестирование эффективности различных методов поиска минимального элемента массива. Анализ выполнен для двух подходов: **статической линковки** и **динамической загрузки библиотек**. Эксперименты проводились на массиве из 9999999 элементов.

4.1 Тестирование производительности

Для оценки производительности использовались тестовые программы, собранные с различными уровнями оптимизации (`O0, O1, O2, Os`). Измерение времени выполнения осуществлялось при помощи утилиты `time`, а результаты сохранялись в файл `results.txt`. Основные тесты включали:

- поиск минимального элемента с использованием статической линковки;
- поиск минимального элемента с динамической линковкой;
- анализ влияния флагов оптимизации компилятора на скорость работы;
- исследование влияния метода загрузки библиотеки (`dlopen`) на производительность.

4.2 Анализ полученных данных

На основе проведенных тестов были получены следующие ключевые результаты:

- **Итеративный метод** работает значительно быстрее рекурсивного. Например, при оптимизации `O1` статический итеративный поиск выполнялся за 0.07 сек, тогда как рекурсивный — за 0.36 сек. Это подтверждает влияние вложенных вызовов на производительность.
- **Оптимизационные флаги компиляции** оказывают значительное влияние. `O2` обеспечил наименьшее время выполнения, а `Os` (оптимизация по размеру) показал сопоставимые результаты, подтверждая высокую эффективность данных режимов.
- **Разница между статической и динамической линковкой** минимальна. При итеративном подходе разница составляет не более 0.01 сек, тогда как при рекурсивном методе динамическая линковка несколько ускоряет выполнение.
- **Динамическая загрузка библиотек** с использованием `dlopen` привела к увеличению времени выполнения, особенно для рекурсивного метода, где время выполнения оказалось выше, чем при стандартной динамической линковке.

4.3 Визуализация результатов

Для удобства анализа данные из `results.txt` были обработаны скриптом `parse_res.py`, превратив их в таблицу `result_pardes.csv`, что позволило построить с помощью скрипта `graph.py` графическое представление зависимости времени выполнения от флагов оптимизации. Итоговый график представлен на рисунке 1.

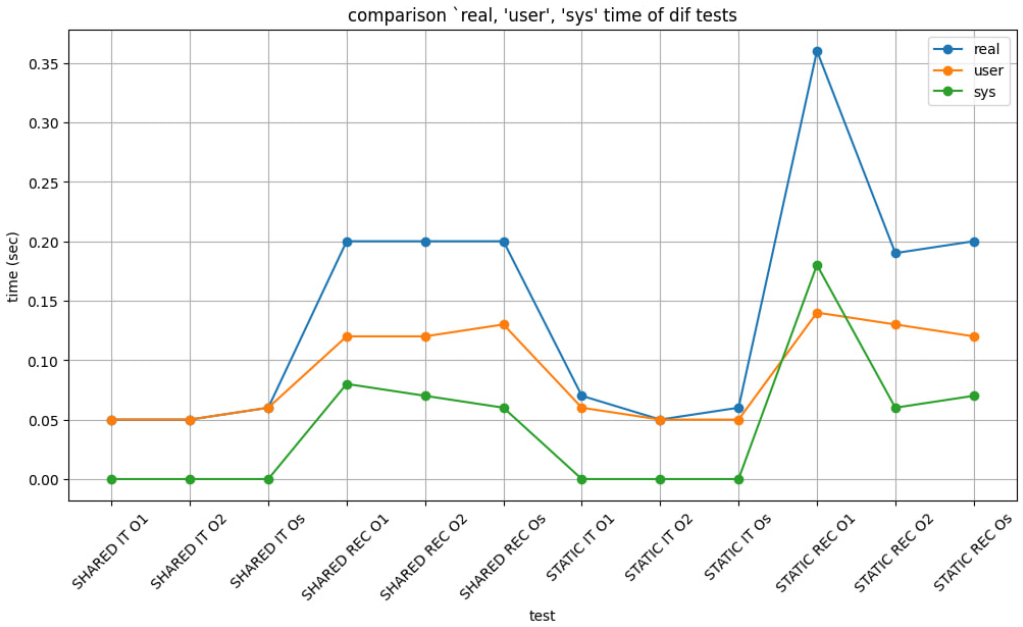

Рис. 1: Влияние флагов оптимизации на скорость выполнения

Таким образом, проведенные эксперименты позволили оценить влияние различных факторов на производительность алгоритма поиска и подтвердить эффективность статической линковки и оптимизационных флагов `O2` и `Os`.

5 Заключение

В ходе выполнения работы были исследованы различные аспекты программирования в среде Linux, включая настройку виртуального окружения, работу с командным интерпретатором, разработку и тестирование программного обеспечения. Основное внимание уделялось изучению механизмов статической и динамической линковки, работе с бинарными файлами и анализу производительности алгоритмов.

Проведенные эксперименты подтвердили, что:

- Оптимизационные флаги компиляции (`O2`, `Os`) оказывают значительное влияние на производительность программ, сокращая время выполнения алгоритмов.
- Итеративная реализация поиска минимального элемента массива показывает лучшую эффективность по сравнению с рекурсивной, особенно при больших объемах данных.
- Динамическая загрузка библиотек (`dlopen`) увеличивает накладные расходы, снижая скорость выполнения по сравнению со статической линковкой.
- Бинарное хранение данных обеспечивает компактность и быстродействие, а проверка целостности с использованием `CRC-32` гарантирует корректность хранимой информации.

Настроенный процесс сборки с помощью **CMake** позволил гибко управлять зависимостями, тестировать различные варианты линковки и проводить сравнительный анализ производительности. Разработанный набор тестов и механизм визуализации результатов обеспечили детальное изучение факторов, влияющих на скорость работы программ.

Полученные знания и проведенные эксперименты представляют ценность при дальнейшей разработке высокопроизводительных систем, позволяя выбирать оптимальные методики работы с памятью, библиотеками и алгоритмами обработки данных.

6 Список литературы

1. Фонд свободного программного обеспечения. *Документация GNU/Linux*. URL: https://www.gnu.org/doc (дата обращения: 16.06.2025).

2. Kitware. *Документация по CMake*. URL: https://cmake.org/documentation (дата обращения: 16.06.2025).

3. Williams, R. *A Painless Guide to CRC Error Detection Algorithms*. URL: https://www.ross.net/crc/download/crc_v3.txt (дата обращения: 16.06.2025).

4. GNU Core Utilities. *Описание утилиты time*. URL: https://www.gnu.org/software/coreutils/manual/html_node/time-invocation.html (дата обращения: 16.06.2025).

5. Zajac, D. *Programming with ncurses*. URL: https://invisible-island.net/ncurses/ncurses-intro.html (дата обращения: 16.06.2025).

6. Fog, A. *Optimization of Computer Programs*. URL: https://www.agner.org/optimize (дата обращения: 16.06.2025).