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 по дисциплине: «Научно-исследовательская работа»

«Работа в виртуальной среде»

Студент,
группы 5130201/40003 _____ Адиатуллин Т. Р.

Работу
принял _____ Глазунов В. В.

«____»_____ 2025 г.

Санкт-Петербург, 2025

Содержание

Введение

В рамках практики были изучены основные технологии разработки в операционной системе GNU/Linux. Освоены навыки работы с виртуальными машинами, командной строкой, системами сборки, а также создание и использование статических и динамических библиотек. В качестве среды использовалась минимальная установка Debian 12 в VirtualBox без графического интерфейса, что обеспечило полноценную работу в терминале.

Был проведён анализ компиляции программ с различными уровнями оптимизации, а также сравнительное исследование производительности рекурсивных и итеративных реализаций алгоритма возведения в степень целых чисел.

Целью работы стало понимание принципов функционирования операционной системы, взаимодействия с файловой системой и освоение методов работы с библиотеками. Практическая часть включала настройку доступа к виртуальной машине, выполнение команд в терминале, создание библиотек и разработку программ, способных обрабатывать бинарные файлы и использовать элементы псевдографики.

В рамках проекта была реализована программа с использованием библиотеки **ncurses**, предназначенная для работы с базой данных. Основное внимание уделялось обработке бинарных файлов и работе со структурами данных. Полученные знания и навыки являются актуальными для системного программирования и разработки кросс-платформенных приложений на базе Linux.

1. Постановка задач

Цель исследования — освоение принципов функционирования ОС, взаимодействия с файловой системой, а также изучение принципов работы со статическими и динамическими библиотеками. Практическая часть включает в себя настройку доступа к виртуальной машине, выполнение команд через интерпретатор, создание библиотек и разработку программ, обрабатывающих бинарные файлы и использующих псевдографику.

1.1. Установка и найстройка среды

В рамках практики необходимо:

- установить VirtualBox и загрузить в нём Debian 12 без графической оболочки;
- создать пользователя с логином своей фамилии и персональным именем хоста;
- настроить проброс порта 22 из гостевой ОС в хостовую для подключения по SSH.

1.2. Основы работы в командной строке

Следует освоить базовые приёмы работы в терминале **Linux**:

- изучить ключевые команды;
- научиться пользоваться историей команд;
- настроить вход по SSH с использованием ключей.

1.3. Подготовка среды разработки

Для выполнения заданий нужно установить компилятор и инструменты разработки, включая **CMake**.

1.3.1. Задание 1. Статические и динамические библиотеки

Требуется:

1. Реализовать функцию возведения в степень целых чисел:
    - с использованием рекурсии;

- с применением итеративного подхода (без рекурсии).

2. Создать две версии библиотеки:
   - статическую;
   - динамическую (разделяемую).
3. Провести тестирование производительности с использованием различных флагов оптимизации компилятора:
   - `-O0`, `-O1`, `-O2`, `-Os`.
4. Реализовать загрузку динамической библиотеки во время выполнения с использованием функций `dlopen`.

1.3.2. Задание 2. Двоичные файлы и визуализация структур в псевдографике

Требуется:

1. В рамках индивидуального задания необходимо реализовать структуру с именем **ZNAK**, включающую следующие поля:
   - фамилия (строка Си);
   - имя (строка Си);
   - знак Зодиака (перечисляемый тип).
   - дата рождения (битовая структура).
2. Требуется разработать дополнительную структуру, представляющую заголовок базы данных. Поля этой структуры:
   - сигнатура формата файла (4 байта) — первые четыре буквы фамилии разработчика, записанные латиницей;
   - номер транзакции (4 байта) — счётчик, увеличивающийся при каждом обращении к базе (чтение или запись);
   - число записей (4 байта) — количество структур типа **ZNAK**, следующих за заголовком;
   - контрольная сумма CRC-32 (4 байта) — вычисляется на основе всех байтов структур **ZNAK** с помощью функции из библиотеки **zlib.h**.
3. Программа должна иметь псевдографический интерфейс, реализованный с использованием библиотеки **ncurses**. Реализуются следующие режимы работы:

- создание базы данных при её отсутствии:
  - ввод массива структур **ZNAK** с клавиатуры;
  - формирование бинарного файла и запись корректного заголовка.
- Индивидуальное задание (поиск информации по фамилии людей):
  - найти и вывести информацию о людях, чья фамилия введена с клавиатуры;
  - при отсутствии соответствующего пользователя — вывести сообщение об этом;
  - произвести обновление заголовка базы.
- Добавление новой записи:
  - ввод одной структуры **ZNAK**;
  - дозапись в файл и обновление заголовка (счётчик, количество, CRC).

4. После компиляции и запуска программы необходимо проанализировать подключённые разделяемые библиотеки с помощью утилиты **ldd**, а также построить рекурсивное дерево зависимостей.

2. Особенности реализации

2.1. Установка и найстройка среды виртуализации

2.1.1. Выбор и конфигурация виртуальной среды

Для разработки и выполнения лабораторных заданий была выбрана Oracle VirtualBox — стабильная и кроссплатформенная система виртуализации. Установка производилась на macOs (см. рисунок 1).

Создана новая виртуальная машина с параметрами:

- **Оперативная память:** 4 ГБ;
- **Жёсткий диск:** 10 ГБ (динамически расширяемый);
- **Образ системы:** 4 ГБ минимальный ISO-файл **Debian 12**.

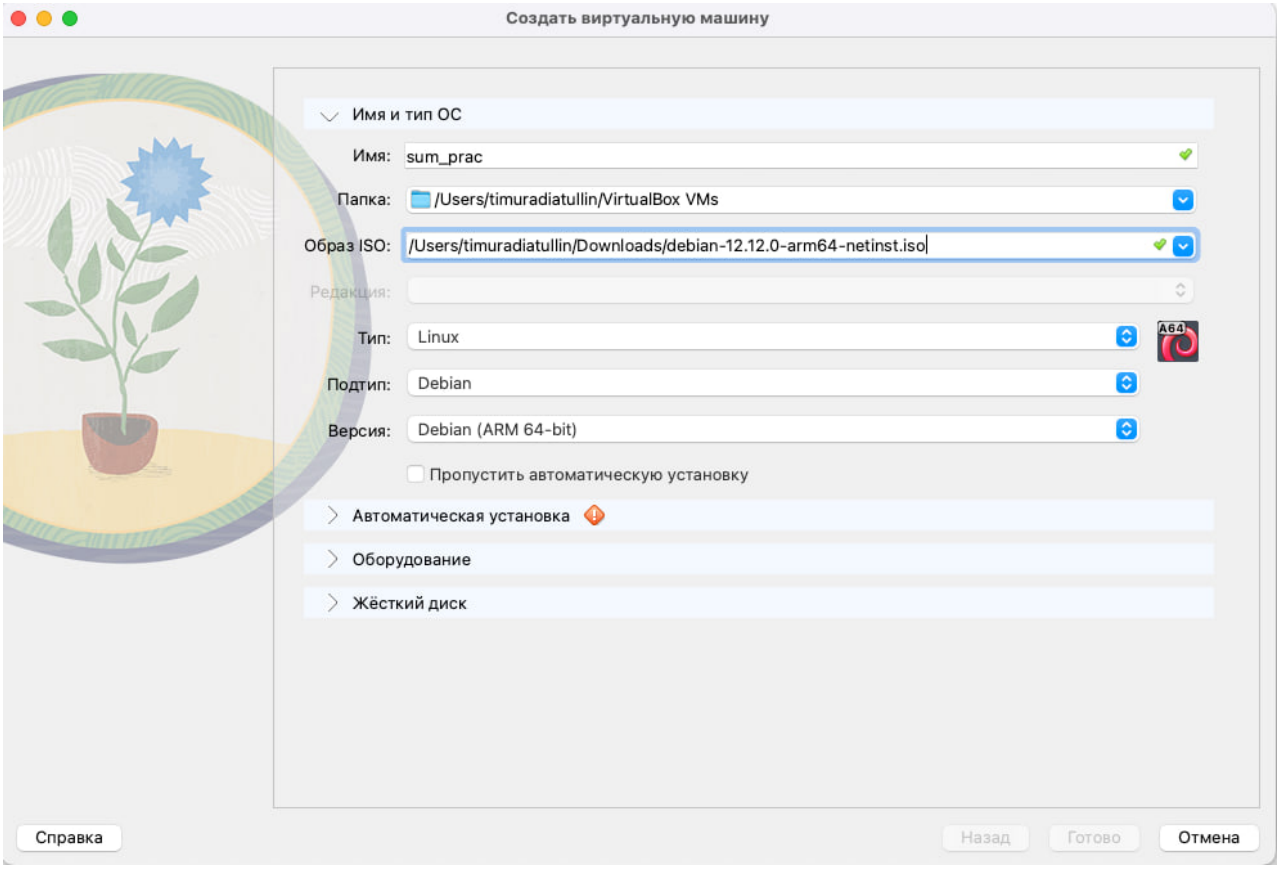

Рис. 1: Создание и настройка виртуальной машины в VirtualBox

2.1.2. Замена имени хоста

На этапе установки системе было присвоено имя хоста 333 (см. рисунок 2), что позволяет однозначно идентифицировать её в сети и командной строке.

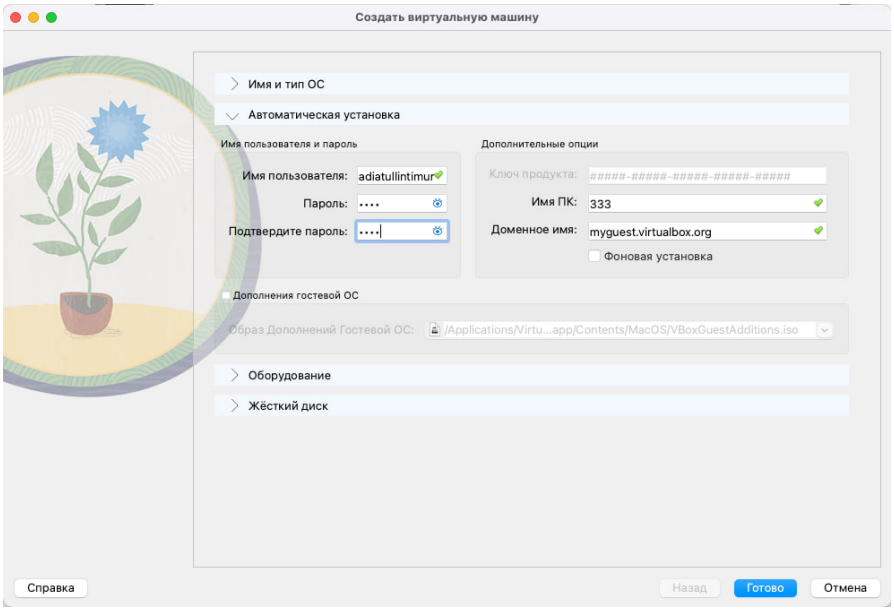

Рис. 2: Задание имени хоста для виртуальной машины

2.1.3. Установка операционной системы

На рисунке (см. рисунок 3). изображен процесс установки Debian 12 проходил в стандартном. Для экономии ресурсов выбрана минимальная установка без GUI, подходящая для терминальной работы, компиляции и тестирования программ на языке C.

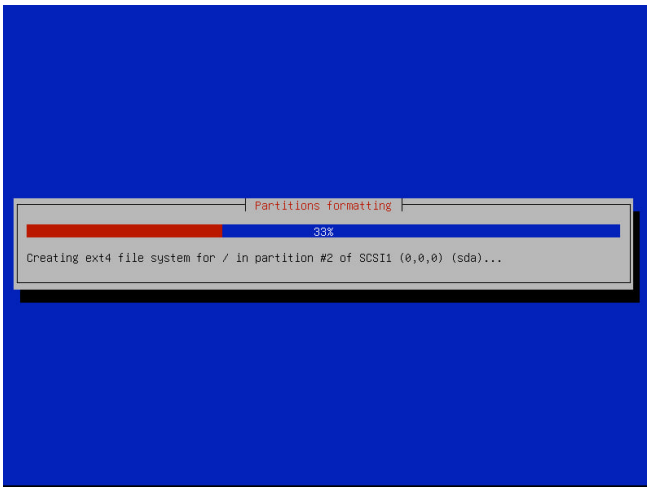

Рис. 3: Создание и настройка виртуальной машины в VirtualBox

2.1.4. Финальная конфигурация машины

Итоговые параметры системы представлены (см. рисунок 4). После установки операционной системы дополнительно проверены параметры конфигурации виртуальной машины в VirtualBox.

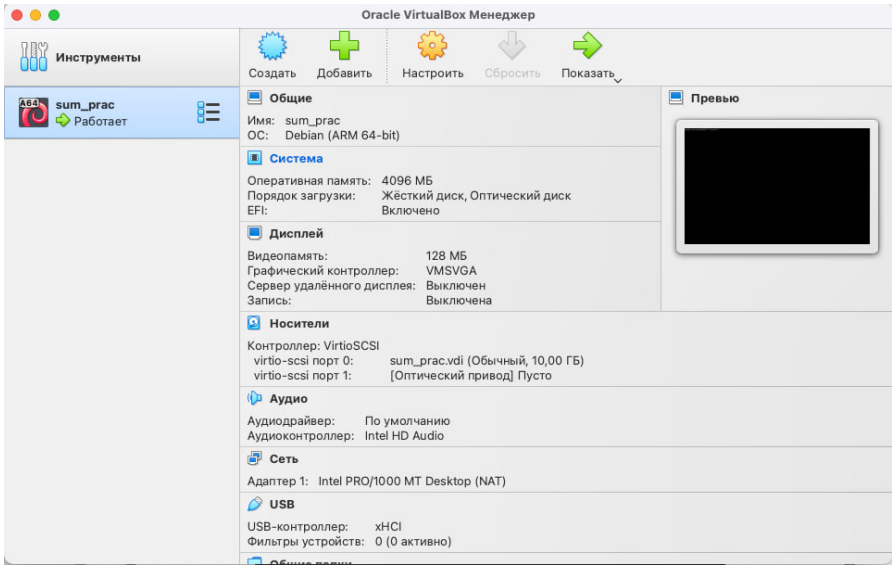

Рис. 4: Виртуальная машина

2.1.5. Создание учетной записи

Во время установки гостевой операционной системы Debian 12 (64-битная) была создана учётная запись с логином adiatullintimur, соответствующим фамилии пользователя в латинской транслитерации, как это предусматривалось заданием. Изначально имя хоста (hostname) было задано как случайная строка символов, а затем было изменено индивидуально пользователем - 333. Аутентификация осуществлялась через стандартный механизм ввода логина и пароля в текстовой консоли tty1, поскольку графическая среда в системе отсутствует — это также соответствовало условиям задания (см. рисунок 5).

Рис. 5: Логин

2.1.6. Настройка проброса портов

Дабы получить удаленный доступ к гостевой операционной системе была выполнена найстройка проброса TCP-порта 22 с виртуальной машины на хост-систему (см. рисунок 6).

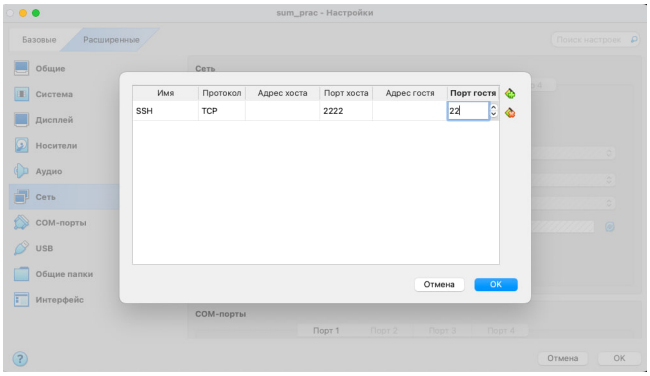

Рис. 6: Проброс порта в гостевой операционной системе

В большинстве случаев виртуальная машина работает в изолированной среде, а её сеть настроена в режиме **NAT (Network Address Translation)**. Это означает следующее:

- виртуальная машина имеет **локальный (внутренний) IP-адрес**, который недоступен извне;
- внешние компьютеры, включая хост-систему, не могут напрямую подключиться к этой машине, так как трафик не маршрутизируется внутрь.

Для обхода этих ограничений используется проброс портов (port forwarding):

Мы настраиваем хост-систему так, чтобы весь трафик, поступающий на порт 2222, автоматически перенаправлялся на порт 22 (SSH) внутри виртуальной машины.

Хост-система принимает соединение и направляет его внутрь виртуальной машины, обеспечивая доступ к SSH, несмотря на изоляцию.

После настройки проброса порта в гостевой ОС, подключаемся через хост-систему (см. рисунок 7).

```
❯ ssh -p 2222 adiatullintimur@localhost
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
ED25519 key fingerprint is SHA256:5ZWcE60U2UIFetyJmRRZSwUKqWyh0PF5KkEKisUcUYM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (ED25519) to the list of known hosts.
adiatullintimur@localhost's password:
Linux 333 6.1.0-39-arm64 #1 SMP Debian 6.1.148-1 (2025-08-26) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Sep 10 18:11:49 2025
adiatullintimur@333:~$ _
```

Рис. 7: Проброс порта в хост-системе

2.1.7. Настройка беспарольной аутентификации по SSH

Аутентификация по ключу - это не только удобство, но и значительное повышение безопасности. Ниже приведены её ключевые преимущества:

- закрытый ключ невозможно подобрать, в отличие от слабых или утёкших паролей;
- в отличие от паролей, которые могут быть перехвачены или сохранены, приватный ключ остаётся на клиенте.

Для подключения к виртуальной машине без пароля была настроена SSH-аутентификация по ключу (см. рисунок 8). Процесс состоял из следующих шагов:

1. Генерация пары ключей на хост-системе

В терминале хост-системы выполнить команду:

```
ssh-keygen -t ed25519 -C "timur@local"
```

По умолчанию ключи сохраняются в каталоге `/.ssh`. Публичный ключ будет содержаться в файле `id_ed25519.pub`.

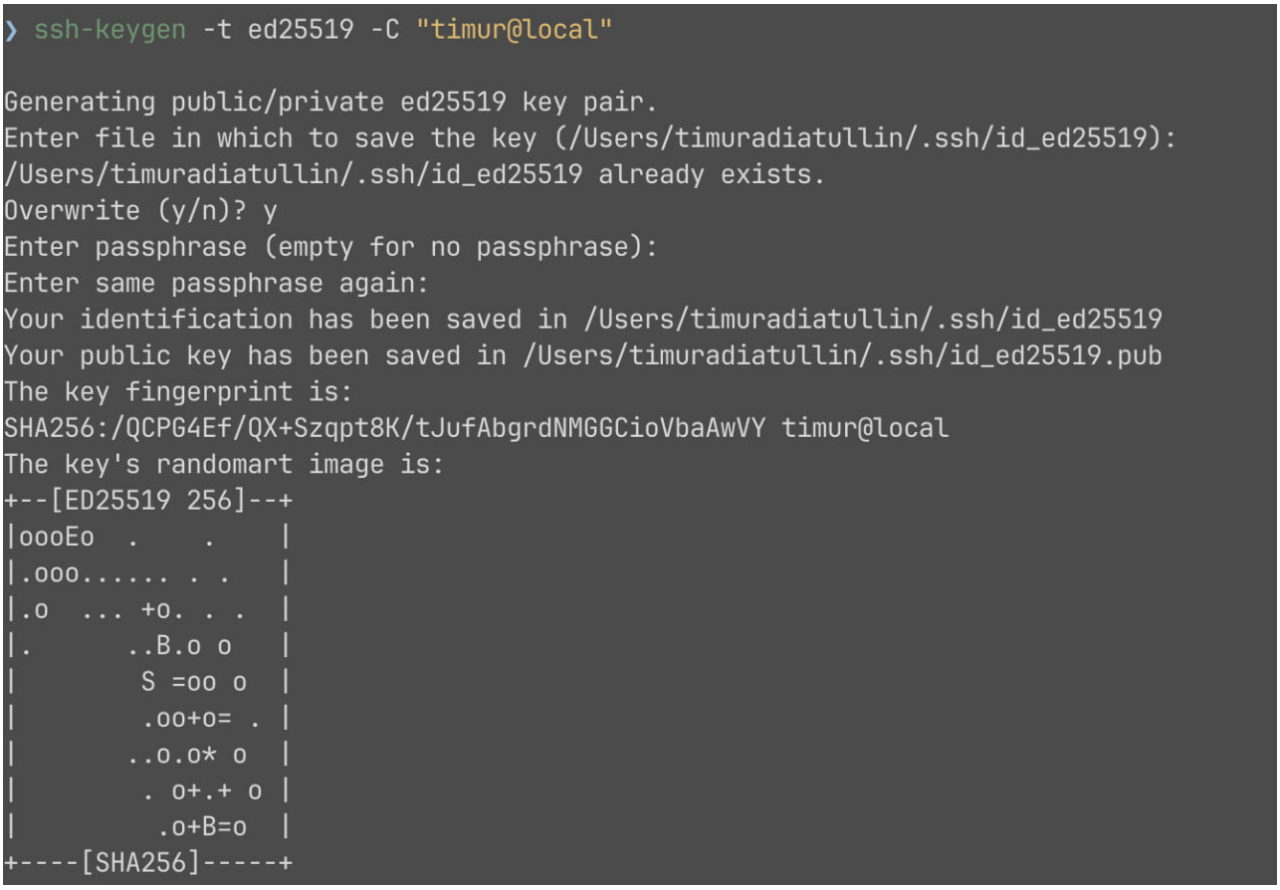

Рис. 8: Генерация ключа в терминале bash

2. Добавление публичного ключа на виртуальную машину

Для настройки беспарольной аутентификации по SSH необходимо передать публичный ключ на гостевую систему. Это можно сделать с помощью утилиты `ssh-copy-id`, которая автоматически создаёт каталог `~/.ssh` на виртуальной машине (если он отсутствует), добавляет ключ в файл `authorized_keys`, и устанавливает необходимые права доступа.

```
ssh-copy-id -p 2222 -i ~/.ssh/id_ed25519.pub adiatullintimur@localhost
```

Описание параметров:

- `-p 2222` — указывает нестандартный порт подключения к SSH-серверу виртуальной машины.
- `-i /.ssh/id_ed25519.pub` — путь к публичному ключу, который будет добавлен.
- `adiatullintimur@localhost` — имя пользователя и адрес виртуальной машины.

После выполнения команды, ключ будет добавлен в файл `authorized_keys` на гостевой системе, что позволит подключаться по SSH без запроса пароля.

3. Назначение прав доступа к ключам

Для корректной работы SSH необходимо установить следующие права:

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

4. Подключение к виртуальной машине

Для подключения с хоста к виртуальной машине выполнить:

```
ssh adiatullin@localhost -p 2222
```

Таким образом теперь мы можем подключиться по ключу. Результат представлен на рисунке 9.

```
1 ❯ ssh -p 2222 'adiatullintimur@localhost'
Enter passphrase for key '/Users/timuradiatullin/.ssh/id_ed25519':
Linux 333 6.1.0-39-arm64 #1 SMP Debian 6.1.148-1 (2025-08-26) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Sep 10 18:26:56 2025 from 10.0.2.2
```

Рис. 9: Вход в хост-системе по ключу

2.2. Основы работы в командной строке

2.2.1. Автодополнение команд с помощью Tab

В терминале Debian доступна функция автодополнения: при вводе первых символов команды достаточно нажать клавишу Tab, чтобы завершить команду автоматически. Например, при наборе wh система при двойном нажатии предлагает выбор таких команд, как **whoami**, **whereis** и т.д., а da — до **date**, **dash**, если не возникает неоднозначности. Если же существует несколько возможных продолжений, двойное нажатие Tab отобразит список всех доступных вариантов.

Рисунок 10 демонстрирует примеры автодополнения команд в терминале с помощью клавиши Tab.

```
adiatullintimur@333:~$ wh
whatis          whereis         which           which.debianutils  while           whiptail        who             whoami
adiatullintimur@333:~$ t
tabs            tbl             tic             toe             traceproto.db   trap            tsort
tac             tee             time            top             traceroute      troff           tty
tail            telnet          timedatectl     touch           traceroute6     true            type
tar             tempfile        timeout         tput            traceroute6.db  truncate        typeset
tasksel         test            times           tr              traceroute.db   tryaffix        tzselect
taskset         then            tload           traceproto      traceroute-nanog tset
adiatullintimur@333:~$ t_
```

Рис. 10: Функция автодополнения команд Tab

2.2.2. Использование встроенной справки через man

Команда **man** в Linux предоставляет доступ к подробной справочной информации по большинству утилит. С её помощью можно узнать назначение команды, синтаксис, параметры и возможные способы использования.

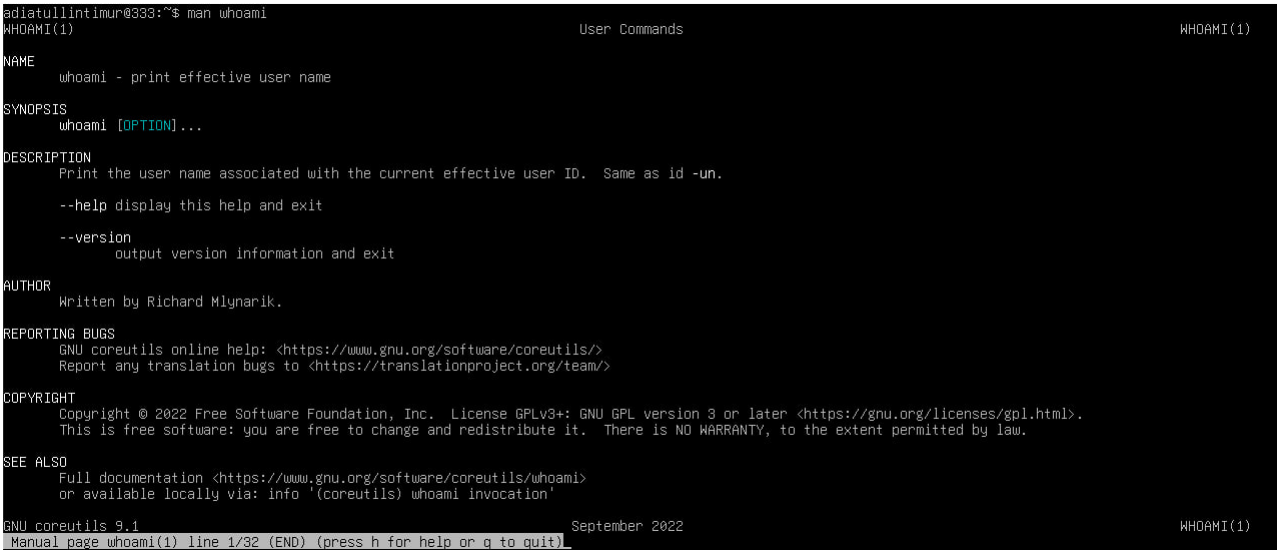

Рис. 11: Встроенная справка whoami

Например, при вызове **man whoami** (см. рисунок 11) отображается руководство по команде **whoami**, показывающее, что она используется для вывода имени текущего пользователя. Эта утилита не имеет дополнительных опций и подходит для быстрой идентификации пользователя.

2.2.3. Остальные команды терминала

Аналогичным образом можно изучать и другие основные команды Linux. Например: **date** — показывает текущую дату и время, поддерживает форматирование; **uptime** — выводит, сколько времени работает система и её среднюю нагрузку, **df** — отображает использование дискового пространства; **cp, mv, touch, rm** — команды для копирования, перемещения, создания и удаления файлов; **stat** — показывает подробную информацию о файле; **file** — определяет тип содержимого файла; **find** — ищет файлы по критериям; **whereis** — ищет местоположение исполняемых файлов; **cat, less** — вывод содержимого файла, **less** удобнее для прокрутки; **grep** — ищет строки по шаблону в файлах; **hexdump** — показывает файл в шестнадцатеричном виде; **time** — измеряет время выполнения команды (см. рисунки 12, 13, 14).

```
adiatullintimur@333:~$ date
Wed Sep 10 06:36:49 PM MSK 2025
adiatullintimur@333:~$ uptime
 18:36:51 up 25 min,  2 users,  load average: 0.00, 0.00, 0.00
adiatullintimur@333:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             1956076       0   1956076   0% /dev
tmpfs             401392     484    400908   1% /run
/dev/sda2        8716732 1362756   6889592  17% /
tmpfs            2006940       0   2006940   0% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
/dev/sda1         524008    6232    517776   2% /boot/efi
tmpfs             401388       0    401388   0% /run/user/1000
adiatullintimur@333:~$ cd prac/
adiatullintimur@333:~/prac$ file exp.c
exp.c: empty
```

Рис. 12: Примеры команд

```
adiatullintimur@333:~$ touch some.c
adiatullintimur@333:~$ ls
prac  some.c
adiatullintimur@333:~$ mv some.c prac/
adiatullintimur@333:~$ cd prac/
adiatullintimur@333:~/prac$ ls
exp.c  newfile.c  some.c
adiatullintimur@333:~/prac$ _
```

Рис. 13: Примеры команд

```
adiatullintimur@333:~/prac/build$ hexdump -C znak.db
00000000  61 64 69 61 04 00 00 00  01 00 00 00 1b 07 cf 7d  |adia...........}|
00000010  41 64 69 61 74 75 6c 6c  69 6e 00 00 7e 00 00 00  |Adiatullin..~...|
00000020  2d 00 00 00 2d 00 00 00  00 00 00 00 00 00 00 00  |-...-...........|
00000030  54 69 6d 75 72 00 00 00  01 00 00 00 00 00 00 00  |Timur...........|
00000040  00 00 00 00 00 00 00 00  90 b1 15 fc aa aa 00 00  |................|
00000050  0b 00 00 00 0d 0c d6 07                           |........|
00000058
adiatullintimur@333:~/prac/build$ time ./main_static 10 10
Тест на 100000 повторов:
Результат: 10000000000. Итеративно: 0.002059 сек (в среднем 0.000000021 сек)
Результат: 10000000000. Рекурсивно: 0.003080 сек (в среднем 0.000000031 сек)

real    0m0.012s
user    0m0.008s
sys     0m0.002s
adiatullintimur@333:~/prac/build$ _
```

Рис. 14: Примеры команд

2.2.4. История команд

В процессе работы в терминале Linux все введенные команды автоматически сохраняются в истории, что позволяет легко возвращаться к ранее выполненным операциям. **history** — выводит список ранее выполненных команд с номерами. Указав число, можно ограничить вывод последних команд: history 42 — последние 42 команды (см. рисунок 15).

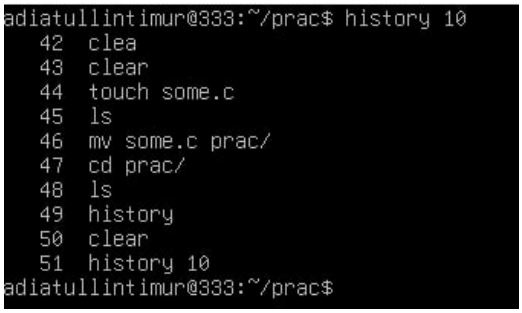

Рис. 15: Применение history

2.3. Установка среды разработки

2.3.1. Установка инструментов

Для подготовки виртуальной машины к разработке необходимо установить базовые инструменты с помощью пакетного менеджера `apt` и утилиты `sudo`.

- `apt` (Advanced Package Tool) — утилита для управления пакетами в Debian-подобных системах.
- `sudo` (superuser do) — позволяет выполнять команды от имени суперпользователя.

Для обновления списка доступных пакетов выполните команду:

```
sudo apt update
```

Для установки необходимых инструментов используйте:

```
sudo apt install build-essential cmake
```

- `build-essential` — включает компилятор языка C, утилиту `make` и другие инструменты, необходимые для сборки программ.
- `cmake` — генератор сборочных скриптов, который использовался в проекте для конфигурации, управления зависимостями и генерации Makefile.

2.3.2. Исследование разделяемых библиотек

После установки необходимых компонентов была выполнена проверка системных зависимостей с помощью утилиты `ldd`. Эта команда позволяет определить:

- имена динамически загружаемых библиотек, от которых зависит программа;

- физические пути к данным библиотекам;
- адреса загрузки в памяти.

Анализ выполнялся на примере интерпретатора командной строки `/bin/bash`. Результат показан на рисунке 16.

```
adiatullintimur@333:~$ ldd /bin/bash
        linux-vdso.so.1 (0x0000ffff936b4000)
        libtinfo.so.6 => /lib/aarch64-linux-gnu/libtinfo.so.6 (0x0000ffff934b0000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff93300000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffff93670000)
adiatullintimur@333:~$ _
```

Рис. 16: Вывод команды `ldd /bin/bash`

По результатам вывода можно выделить несколько критически важных библиотек:

- `linux-vdso.so.1` — виртуальная динамическая библиотека, которая генерируется ядром для ускорения системных вызовов;
- `libtinfo.so.6` — библиотека, отвечающая за взаимодействие с терминалом (цвета, курсор и т.д.). Используется для управления терминалом;
- `libc.so.6` — основная библиотека C, обеспечивающая взаимодействие с ядром;
- `ld-linux-aarch64.so.2` — динамический компоновщик/загрузчик для архитектуры AArch64 (ARM 64-bit). Его основная задача — подготовить и запустить программу.

Рекурсивный анализ зависимостей: Для более глубокого понимания структуры зависимостей была реализована последовательная проверка с помощью `ldd` не только основного бинарного файла, но и каждой найденной библиотеки. Такой подход позволяет отследить цепочки вложенных (косвенных) зависимостей, что особенно полезно при отладке или переносе программного окружения. Результат зависимости представлен на рисунках 17.

Рис. 17: Последовательная проверка с помощью ldd /lib/aarch64-linux-gnu/libtinfo.so.6 и ldd /lib/aarch64-linux-gnu/libc.so.6

2.3.3. Автоматизация сборки с помощью CMake

В качестве системы сборки выбран **CMake** — мощный и гибкий инструмент, подходящий как для небольших проектов, так и для крупных кросс-платформенных систем.

Преимущества использования CMake:

- независимость от платформы и компилятора;
- автоматическая генерация `Makefile` или файлов для других систем;
- возможность управления зависимостями и флагами компиляции;
- интеграция с системой тестирования `CTest`;
- поддержка кастомных целей и скриптов (например, shell-обёрток).

Генерация Makefile:

```
mkdir build && cd build
cmake ..
make
```

Такой подход сочетает в себе простоту Make с гибкостью и абстракцией CMake.

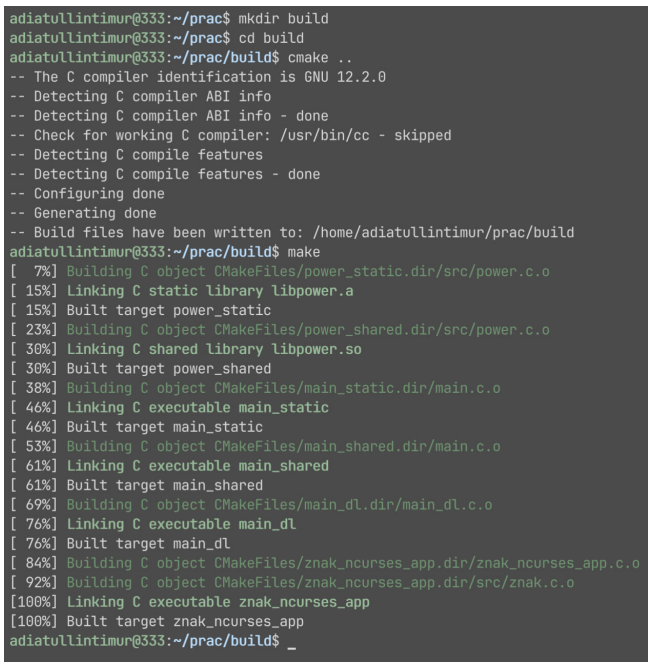

Рис. 18: Процесс сборки

Структура CMakeLists.txt и назначение директив

Описание используемых команд на рисунке 19

- `cmake_minimum_required(VERSION 3.10)` — задаёт минимально допустимую версию CMake, необходимую для сборки проекта.
- `project(power_prac C)` — определяет имя проекта и указывает, что используется язык C.
- `set(CMAKE_C_STANDARD 11)` — устанавливает стандарт языка C.
- `include_directories(headers)` — добавляет папку `headers` в список директорий с заголовочными файлами.
- `add_library(... STATIC ...)` — создаёт статическую библиотеку. В примере:
  - `power_static` — библиотека из `src/power.c`;
  - `set_target_properties(... OUTPUT_NAME "power")` — задаёт имя выходного файла без приставки _static.
- `add_library(... SHARED ...)` — аналогично, создаёт динамическую (разделяемую) библиотеку.
- `add_executable(...)` — создаёт исполняемый файл.
  - `main_static` и `main_shared` — основной файл `main.c`, связанный с соответствующей библиотекой через:

– `target_link_libraries(...)` — подключает библиотеки к целевому исполняемому файлу.

- `main_dl` — исполняемый файл, использующий `dlopen()` для динамической загрузки библиотек во время выполнения. Линкуется с `libdl` через `target_link_libraries(main_dl dl)`.

- `add_executabLe(znak_ncurses_app ...)` — создаёт приложение с псевдографическим интерфейсом. Связывается с библиотеками `znak`, `ncurses` и `zlib`.

```
cmake_minimum_required(VERSION 3.10)
project(power_prac C)

set(CMAKE_C_STANDARD 11)
include_directories(headers)

# Статическая библиотека
add_library(power_static STATIC src/power.c)
set_target_properties(power_static PROPERTIES OUTPUT_NAME "power")

# Динамическая библиотека
add_library(power_shared SHARED src/power.c)
set_target_properties(power_shared PROPERTIES OUTPUT_NAME "power")

# Исполняемый файл со статической линковкой
add_executable(main_static main.c)
target_link_libraries(main_static power_static)

# Исполняемый файл с динамической линковкой
add_executable(main_shared main.c)
target_link_libraries(main_shared power_shared)

# Исполняемый файл с использование библиотеки через dlopen
add_executable(main_dl main_dl.c)
target_link_libraries(main_dl dl)

# Добавляем исполняемый файл с псевдографикой
add_executable(znak_ncurses_app
    znak_ncurses_app.c   # главный исходник в корне
    src/znak.c           # исходник с реализацией
)

# Подключаем заголовочные файлы
target_include_directories(znak_ncurses_app PRIVATE src/headers)

# Линкуем ncurses и zlib
target_link_libraries(znak_ncurses_app ncurses z)


add_custom_target(db
  COMMAND $(PROJECT_SOURCE_DIR)/build/znak_ncurses_app
  WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
  COMMENT "running db"
)
```

Рис. 19: Используемый `CMakeLists.txt` в работе

2.4. Задание 1. Статические и динамические библиотеки

2.4.1. Реализация функций

Индивидуальное задание заключалось в разработке функции возведения в степень целых чисел.

Функция была реализована на языке программирования `C` в двух вариантах: рекурсивном и итеративном. Это позволяет сравнить производительность и особенности реализации двух подходов.

Объявления функций были вынесены в отдельный заголовочный файл `power.h` (см. рисунок 20), обеспечивающий модульность и повторное использование кода. Для предотвращения многократного подключения заголовочного файла использовалась стандартная защита через препроцессорные директивы:

```
#pragma once
```

```
#pragma once
#include <stdio.h>

int power_iterative(int base, int exp);
int power_recursive(int base, int exp);
```

Рис. 20: Заголовочный файл

power_iterative — итеративное возведение в степень

Функция `power_iterative(base, exp)` выполняет возведение целого числа `base` в степень `exp` с использованием итеративного алгоритма **быстрого возведения в степень** (или двоичного возведения в степень).

Временная сложность этого алгоритма — $\Theta(\log n)$, где n — показатель степени. Пространственная сложность — $\Theta(1)$, поскольку он не использует дополнительную память, пропорциональную показателю степени.

Algorithm 1 power_iterative(base, exp)

```
result ← 1
b ← base
while exp > 0 do
    if exp % 2 == 1 then
        result ← result × b
    end if
    b ← b × b
    exp ← exp / 2
end while
return result
```

power_recursive — рекурсивное возведение в степень

Функция `power_recursive(base, exp)` выполняет возведение целого числа `base` в степень `exp` с использованием рекурсивного подхода, основанного на методе «возведение в степень через разложение». Это позволяет сократить глубину рекурсии и повысить эффективность. Временная сложность — $\Theta(\log n)$, пространственная — $\Theta(\log n)$ из-за глубины рекурсивного стека.

Algorithm 2 power_recursive(base, exp)

```
if exp = 0 then
    return 1
end if
if exp = 1 then
    return base
end if
half ← power_recursive(base, exp ÷ 2)
if exp mod 2 = 0 then
    return half × half
else
    return half × half × base
end if
```

2.4.2. Создание библиотек

Статическая линковка и её особенности

При использовании статической компоновке весь необходимый код внешних библиотек включается непосредственно в итоговый исполняемый файл. Такой подход делает программу полностью автономной, не зависящей от наличия соответствующих библиотек в системе пользователя. В результате существенно увеличивается размер скомпилированного файла, поскольку он содержит не только основной код, но и все подключённые зависимости.

Такой способ сборки особенно полезен при распространении программного обеспечения, поскольку позволяет доставлять один самодостаточный исполняемый файл. Однако при обновлении исходных библиотек необходимо пересобирать проект, чтобы воспользоваться новыми версиями или исправлениями — автоматическое обновление в этом случае невозможно.

Создание статической библиотеки

Для создания статической библиотеки использовался генератор сборки CMake. Библиотека была определена с использованием директивы `STATIC`, что указывает компилятору на необходимость создания именно статической версии:

```
add_library(power_static STATIC src/power.c)
set_target_properties(power_static PROPERTIES OUTPUT_NAME "power")

add_executable(main_static main.c)
target_link_libraries(main_static power_static)
```

При подключении такой библиотеки весь её код будет встроен в исполняемый файл во время сборки, обеспечивая автономную работу без внешних зависимостей.

Динамическая линковка и подключение библиотеки во время выполнения

При динамической компоновке внешний код не встраивается в исполняемый файл на этапе сборки. Вместо этого, программа получает доступ к библиотекам во время выполнения. Операционная система автоматически загружает необходимые `.so`-файлы из известных путей или тех, что определены переменными окружения.

Такой подход позволяет существенно сократить размер исполняемого файла, поскольку нет необходимости включать весь внешний код внутрь бинарника. Это также упрощает обслуживание программы: при обновлении библиотеки, её использование актуализируется автоматически без необходимости пересборки проекта.

Однако, существует и зависимость от окружения. Если необходимая библиотека отсутствует или недоступна, программа не сможет запуститься. Поэтому важно убедиться, что все зависимости установлены в системе.

Создание динамической библиотеки

Для генерации динамической библиотеки в проекте использовался генератор сборки `CMake`. Соответствующий фрагмент конфигурационного файла имеет следующий вид:

```
add_library(power_shared SHARED src/power.c)
set_target_properties(power_shared PROPERTIES OUTPUT_NAME "power")

add_executable(main_shared main.c)
target_link_libraries(main_shared power_shared)
```

Ключевое отличие от статической версии — использование ключевого слова `SHARED`, обозначающего создание разделяемой библиотеки. После сборки формируется файл с расширением `.so`, который затем связывается с исполняемым файлом во время выполнения.

Тестирование статической и динамической линковки

Тестируются две версии алгоритма:

1. power_iterative - итеративная версия
2. power_recurvice - рекурсивная версия

Результаты тестирования представлены в разделе 3.1, поэтому все подробности с точностью до микросекунд можно посмотреть там.

main.c — тестирование функций возведения в степень

Программа `main.c` предназначена для сравнения производительности двух реализаций возведения числа в степень: рекурсивной и итеративной. Измерение времени осуществляется с помощью стандартной библиотеки `time.h`. Количество повторов задаётся константой `REPEATS`.

Algorithm 3 main(base, exp)

if количество аргументов < 3 **then**
    вывести сообщение об ошибке
    завершить выполнение
end if
base ← `atoi(argv[1])`
exp ← `atoi(argv[2])`
объявить переменные: `start`, `end`, `result_iter`, `result_rec`
`iter_total` ← 0
`rec_total` ← 0
{Рекурсивный тест}
start ← текущее время
for $i = 0$ **to** REPEATS -1 **do**
    result_rec ← power_recursive(base, exp)
end for
end ← текущее время
rec_total ← end − start
{Итеративный тест}
start ← текущее время
for $i = 0$ **to** REPEATS -1 **do**
    result_iter ← power_iterative(base, exp)
end for
end ← текущее время
iter_total ← end − start
вывести количество повторов
вывести result_iter и среднее время итеративной реализации
вывести result_rec и среднее время рекурсивной реализации
return 0

Реализация динамической загрузки с использованием `dlopen()`

Для демонстрации возможностей динамической загрузки во время выполнения была разработана программа, позволяющая загрузить библиотеку `libpower.so` и вызвать реализованные в ней функции возведения в степень. Такой подход позволяет гибко управлять подключением зависимостей и обеспечивает модульность (см. рисунок 21).

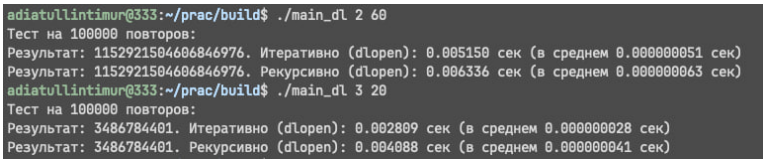

Рис. 21: Динамическая загрузка

Выполняются следующие шаги:

С помощью POSIX API (Portable Operating System Interface — это набор стандартов) производится динамическая загрузка разделяемой библиотеки:

- Используется вызов `dlopen()` с флагом `RTLD_LAZY` для отложенного связывания.
- С помощью `dlsym()` получаются указатели на функции `power_iterative` и `power_recursive`, реализующие итеративный и рекурсивный алгоритмы возведения в степень.
- После выполнения поиск вызывается через полученные указатели, а длительность каждого из вызовов измеряется с помощью `clock()` и `CLOCKS_PER_SEC`.

Algorithm 4 main(base, expr)

if количество аргументов < 3 **then**
  вывести сообщение "Usage: <имя $><$ *base* $><$ *exponent* $>$ "
  завершить выполнение с кодом 1
end if
base ← `atoi(argv[1])`
exp ← `atoi(argv[2])`
handle ← `dlopen("./libpower.so", RTLD_LAZY)`
if handle равно NULL **then**
  вывести сообщение об ошибке `dlerror()` в stderr
  завершить выполнение с кодом 1
end if
power_iterative ← `(power_iter_t)dlsym(handle, "power_iterative")`
power_recursive ← `(power_rec_t)dlsym(handle, "power_recursive")`
if произошла ошибка `dlsym` **then**
  вывести сообщение об ошибке `dlerror()` в stderr
  `dlclose(handle)`
  завершить выполнение с кодом 1
end if
объявить переменные: `start`, `end`, `result_iter`, `result_rec`
`iter_total` ← 0.0
`rec_total` ← 0.0
{Рекурсивный тест}
start ← `clock()`
for $i = 0$ **to** REPEATS -1 **do**
  result_rec ← `power_recursive(base, exp)`
end for
end ← `clock()`
rec_total ← (double)(end − start) / CLOCKS_PER_SEC
{Итеративный тест}
start ← `clock()`
for $i = 0$ **to** REPEATS -1 **do**
  result_iter ← `power_iterative(base, exp)`
end for
end ← `clock()`
iter_total ← (double)(end − start) / CLOCKS_PER_SEC
вывести "Тест на REPEATS повторов:"
вывести `result_iter`, `iter_total`, и `iter_total / REPEATS`
вывести `result_rec`, `rec_total`, и `rec_total / REPEATS`
`dlclose(handle)`
return 0

2.5. Задание 2. Реализация основной структуры

2.5.1. Заголовочный файл znak.h

Индивидуальное задание: реализовать структуру с именем **ZNAK**, содержащую следующие поля:

- Фамилия (строка Си);
- Имя (строка Си);
- Знак Зодиака (перечисляемый тип);
- Дата рождения (битовая структура).

```c
#pragma once

#include <stdint.h>
#include <stddef.h>

typedef enum {
    ARIES, TAURUS, GEMINI, CANCER,
    LEO, VIRGO, LIBRA, SCORPIO,
    SAGITTARIUS, CAPRICORN, AQUARIUS, PISCES
} ZODIAC;

typedef struct {
    uint8_t day   : 5;  // 0–31
    uint8_t month : 4;  // 0–12
    uint16_t year : 12; // 0–4095
} BIRTHDAYDATE;


typedef struct {
    char surname[32];
    char name[32];
    ZODIAC zodiac;
    BIRTHDAYDATE birth;
} ZNAK;


typedef struct {
    char signature[4];
    uint32_t transaction_id;
    uint32_t record_count;
    uint32_t crc32;
} DB_HEADER;


uint32_t calculate_crc32(ZNAK *records, size_t count);
```

Рис. 22: Заголовочный файл znak.h

На приведенном скриншоте 22 представлена реализация заголовочного файла znak.h, в котором описывается структура данных для представления информации о человеке. Во избежание повторного включения используется стандартная защита с помощью препроцессорных директив:

```
#pragma once
```

2.5.2. Контроль целостности данных с помощью CRC-32

Для проверки целостности бинарной базы данных используется контрольная сумма CRC-32 (см. рисунок 23), реализованная через стандартную библиотеку zlib. Это позволяет обнаруживать нарушения структуры при загрузке данных, обеспечивая надёжность хранения информации. Контрольная сумма вычисляется только по содержимому массива структур, исключая заголовок. Алгоритм основан на вызове crc32(). Обновление суммы и её проверка происходят на этапе сохранения файла. При несоответствии значений загрузка данных отменяется, сигнализируя о повреждении файла.

```
uint32_t calculate_crc32(ZNAK *records, size_t count) {
    return crc32(0L, (const unsigned char *)records, count * sizeof(ZNAK));
}
```

Рис. 23: Функция расчета crc32

2.5.3. Реализация в файле znak_ncurses_app.c

Файл `znak_ncurses_app.c` содержит реализацию функций для работы с базой данных. Основные функции включают:

- загрузку и сохранение данных;
- поиск человека по фамилии;
- создание нового файла, если его нет.

2.6. Задание 2. Реализация программы с использованием псевдографической библиотеки ncurses

2.6.1. Использование библиотеки `ncurses`

В ходе реализации проекта был разработан текстовый пользовательский интерфейс с применением библиотеки `ncurses`, которая обеспечивает контроль над отображением содержимого терминала, обработкой пользовательского ввода, а также предоставляет механизмы для построения псевдографических интерфейсов.

Основные функции, использованные в проекте

- `initscr()` — инициализация библиотеки `ncurses`, создаёт основной экран и подготавливает терминал к работе
- `cbreak()` — отключает буферизацию строкового ввода, позволяя обрабатывать символы по мере их ввода
- `refresh()` — обновляет физический экран, синхронизируя его с внутренним буфером
- `clear()` — очищает содержимое экрана перед отрисовкой новых элементов
- `mvprintw()` — функция для вывода отформатированного текста на экран (аналог `printf`)

Обработка пользовательского ввода

Для организации безопасного и интерактивного ввода данных в терминале использовались следующие функции библиотеки `ncurses`:

- `getnstr()` — выполняет ввод строки с ограничением по длине, предотвращая переполнение буфера. Применяется для безопасного считывания текстовых данных.
- `getch()` — ожидает нажатия клавиши пользователем и возвращает символ или код, соответствующий этой клавише. Используется для обработки выбора пользователя или подтверждения действия.
- `echo()` и `noecho()` — включают или отключают отображение вводимых символов в терминале. Используются для управления видимостью данных при вводе.

Завершение работы программы

- `endwin()` — корректно завершает работу с библиотекой `ncurses`, восстанавливая стандартное поведение терминала. Вызывается перед выходом из программы для освобождения ресурсов.

Где были задействованы данные функции

- Функция **run_ui()** использует комбинацию `initscr()`, `cbreak()`, `noecho()`, `echo()` и `clear()`.
- Функция **input_record()** использует комбинацию `clear()`, `getch()`, `echo()` и `noecho()`

В ходе выполнения операций с базой данных вывод результата осуществляется с помощью функции `mvprintw()`, которая позволяет позиционировать текст в окне терминала с точностью до строки и столбца. Это обеспечивает более гибкое и структурированное отображение информации на экране.

Использование `mvprintw()` позволяет отображать статус выполнения операций, уведомления о загрузке или сохранении данных, а также подтверждения ввода, делая интерфейс более информативным и удобным для пользователя.

Таким образом, библиотека `ncurses` предоставила все необходимые средства для построения интерактивного текстового интерфейса: реализация меню, форматированный вывод результатов и пошаговый ввод данных. Всё это поддерживает кроссплатформенность, стабильную работу в терминале и полностью отвечает требованиям задания.

2.6.2. Операции с базой данных

Для реализации работы с записями были использованы следующие функции, отвечающие за загрузку, сохранение и контроль целостности данных в бинарном файле:

- `calculate_crc32()` — вычисляет контрольную сумму CRC-32 по массиву записей, обеспечивая проверку целостности данных при сохранении и загрузке из файла.
- `save_to_file()` — осуществляет сохранение заголовка и массива записей в файл. Формирует структуру `DB_HEADER`, записывает в неё количество записей, идентификатор транзакции и CRC. Затем последовательно сохраняет заголовок и все людей в бинарном формате.
- `load_from_file()` — загружает данные из существующего бинарного файла. Считывает заголовок и массив данных. После успешной загрузки актуализирует идентификатор транзакции `current_tx_id` для обеспечения уникальности новых операций.

2.6.3. Функции пользовательского интерфейса

Пользовательский интерфейс реализован с помощью библиотеки `ncurses` и включает следующие функции:

- `input_record()` — обеспечивает ввод новой записи, считывая у пользователя фамилию, имя, дату рождения и знак зодиака. Добавляет данные в локальный массив.
- `show_records()` — отображает таблицу текущих данных с выравниванием по столбцам. Позволяет пользователю просмотреть внесённые данные.
- `find_by_surname()` — позволяет найти и вывести все записи, соответствующие введённой пользователем фамилии. Если совпадений нет, выдаёт соответствующее сообщение.
- `run_ui()` — основная функция интерфейса, управляющая главным меню программы и навигацией между всеми остальными функциями: вводом, просмотром, поиском, а также сохранением и загрузкой данных из файла.
- Функция `main()` является точкой входа в приложение. Она вызывает `run_ui()`, которая инициализирует библиотеку `ncurses` и управляет основным циклом взаимодействия с пользователем через текстовое меню.

Надёжность работы обеспечивается тем, что каждая стадия — от чтения и записи файла до обработки пользовательского выбора — включает встроенные механизмы обработки ошибок. Контроль целостности данных при этом выполняется с использованием контрольных сумм по алгоритму CRC-32.

2.6.4. Реализация функции интерфейса

Функция `input_record()` — ввод новой записи

Algorithm 5 input_record()

1: Создать объект `ZNAK z`
2: Создать строку `tmp` длиной 50 символов
3: Очистить экран
4: Вывести заголовок «Добавление новой записи»
5: Ввести фамилию → `z.surname`
6: Ввести имя → `z.name`
7: Ввести день рождения → `z.birth.day`
8: Ввести месяц рождения → `z.birth.month`
9: Ввести год рождения → `z.birth.year`
10: Ввести знак зодиака (0–11) → `z.zodiac`
11: **if** `z.zodiac` < 0 **or** `z.zodiac` > 11 **then**
12: `z.zodiac` $\leftarrow 0$
13: **end if**
14: Добавить `z` в массив `records`
15: Увеличить `record_count`
16: Увеличить `transaction_id`
17: Вывести сообщение «Запись добавлена»
18: Ожидать нажатие клавиши

Функция `show_records()` — отображение всех записей

Algorithm 6 show_records()

1: Очистить экран
2: Вывести заголовок «Список записей»
3: Вывести таблицу: «№ | Surname | Name | Birthday | Zodiac»
4: **for** $i = 0$ **to** `record_count` -1 **do**
5: Вывести номер $i + 1$, `records[i].surname`, `name`, `birth`, `zodiac`
6: **end for**
7: Вывести «Нажмите любую клавишу для возврата»
8: Ожидать нажатие клавиши

Функция `find_by_surname()` — поиск по фамилии

Algorithm 7 find_by_surname()

1: Создать строку `tmp` длиной 32 символа
2: Очистить экран
3: Вывести «Введите фамилию для поиска»
4: Считать `tmp`
5: `found` $\leftarrow 0$
6: **for** $i = 0$ **to** `record_count` -1 **do**
7: **if** `records[i].surname` = `tmp` **then**
8: Вывести имя, дату рождения, знак зодиака
9: `found` $\leftarrow$ `found` $+1$
10: **end if**
11: **end for**
12: **if** `found` = 0 **then**
13: Вывести «Людей с такой фамилией не найдено»
14: **end if**
15: Увеличить `transaction_id`

Функция `save_to_file()` — сохранение базы данных

Algorithm 8 save_to_file()

1: Открыть файл `DB_PATH` в режиме `"wb"`
2: **if** файл не открыт **then**
3: Вывести «Не удалось открыть файл для записи»
4: **return**
5: **end if**
6: `header.signature` $\leftarrow$ ”adia”
7: `header.transaction_id` $\leftarrow$ `transaction_id` $+1$
8: `header.record_count` $\leftarrow$ `record_count`
9: `header.crc32` $\leftarrow$ `calculate_crc32(records, record_count)`
10: Записать `header` и `records` в файл
11: Закрыть файл
12: Вывести «Файл успешно сохранён»

Функция `load_from_file()` — загрузка базы данных

Algorithm 9 load_from_file()

1: Открыть файл `DB_PATH` в режиме `"rb"`
2: **if** файл не открыт **then**
3: Вывести «Файл не найден. Введите новые данные»
4: **return**
5: **end if**
6: Считать `header`
7: Считать `records` (кол-во: `header.record_count`)
8: `record_count` $\leftarrow$ `header.record_count`
9: `transaction_id` $\leftarrow$ `header.transaction_id` $+1$
10: Закрыть файл
11: Вывести «Загружено записей: record_count»

Функция `run_ui()` — главное меню

Algorithm 10 run_ui()

```
Инициализировать ncurses: initscr(), cbreak(), noecho()
while true do
    Очистить экран
    Вывести пункты меню:
    1. Добавить новую запись
    2. Показать все записи
    3. Найти по фамилии
    4. Сохранить в файл
    5. Загрузить из файла
    6. Выйти
    Считать выбор пользователя → choice
    if choice = 1 then
        Вызвать input_record()
    end if
    if choice = 2 then
        Вызвать show_records()
    end if
    if choice = 3 then
        Вызвать find_by_surname()
    end if
    if choice = 4 then
        Вызвать save_to_file()
    end if
    if choice = 5 then
        Вызвать load_from_file()
    end if
    if choice = 6 then
        Завершить ncurses: endwin()
        return
    end if
end while
```

Основной цикл интерфейса: `run_ui()`

Функция `run_ui()` отвечает за запуск и работу пользовательского текстового интерфейса с использованием библиотеки `ncurses` (см. рисунок 24). На начальном этапе осуществляется инициализация экрана, переход в постсимвольный режим ввода и отключение отображения вводимых символов:

- `initscr()` — создаёт главное окно и подготавливает терминал к выводу;
- `cbreak()` — отключает буферизацию ввода;
- `noecho()` — предотвращает отображение вводимых пользователем символов.

Затем программа входит в бесконечный цикл, внутри которого отображается главное меню. В зависимости от выбора пользователя (переменная `choice`), вызываются соответствующие функции:

- `input_records()` — ввод новой записи
- `show_records()` — вывод таблицы записей;
- `find_by_surname()` — осуществляет поиск записей по введённой фамилии;
- `save_to_file()` — сохранение в файл;
- `load_from_file()` — загрузка данных из файла;
- `endwin()` — завершение работы с библиотекой `ncurses` и выход из интерфейса.

Конструкция `switch-case` используется для определения выбранного действия и управления логикой программы. Интерфейс формируется через `mvprintw()`, обеспечивая чёткое позиционирование элементов на экране терминала.

При вводе некорректного выбора, программа запрашивает повторный ввод у пользователя. Это также показывает, что программа рассматривает ошибки и не при этом не падает.

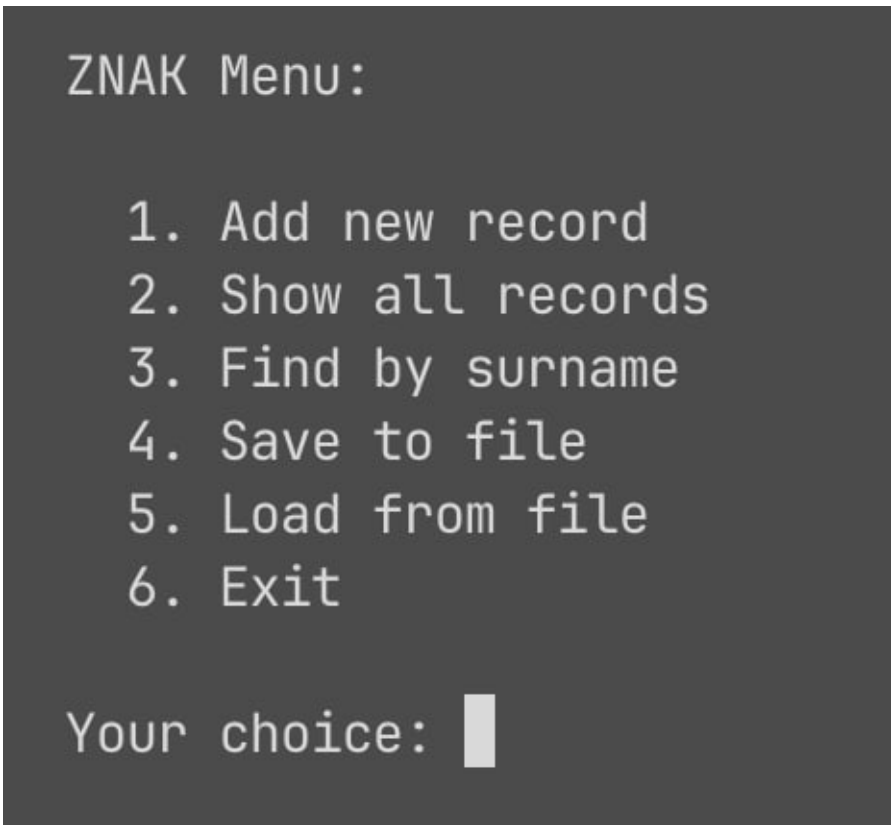

Рис. 24: Пользовательский Интерфейс

3. Результаты работы и эксперименты

3.1. Результаты тестирования и анализ зависимостей задания 1

3.1.1. Постановка и методика эксперимента реализаций алгоритма возведения в степень целого числа

Для объективной оценки производительности реализаций алгоритма возведения в степень — итеративной (power_iterative) и рекурсивной (power_recursive), был проведён эксперимент с использованием замеров времени исполнения.

Условия проведения:

- в качестве входных данных для функций использовались фиксированные значения: основание **base** и показатель степени **exp**, которые задаются пользователем.
- каждая реализация запускается **100000 раз** с одними и теми же входными параметрами для получения статистически значимых данных.
- замеры времени осуществляются с помощью функции `clock()` из стандартной библиотеки `<time.h>`;

- фиксируются общее и среднее время выполнения каждого алгоритма, что позволяет сравнить их эффективность;
- в ходе эксперимента не проводилась проверка на корректность выходного значения, только на скорость выполнения.

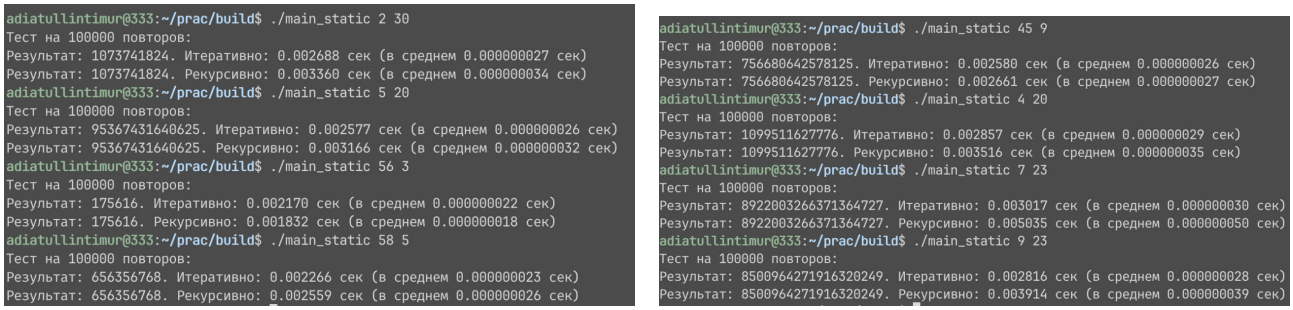

Рис. 25: Тестирование статической

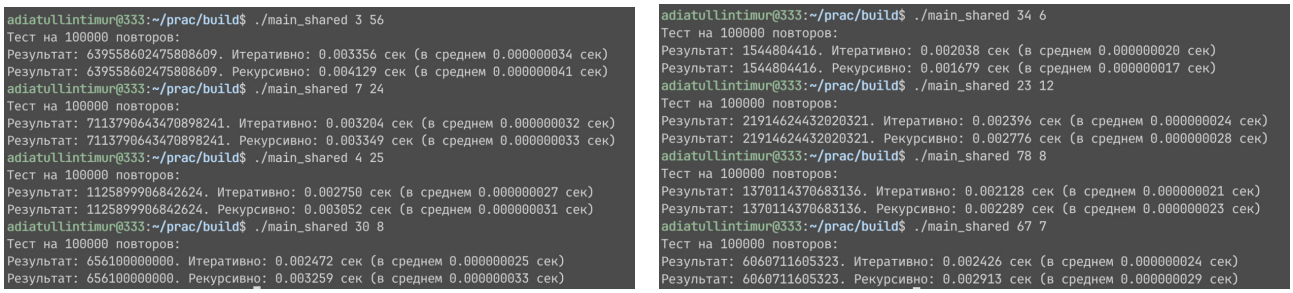

Рис. 26: Тестирование динамической

3.1.2. Влияние флагов оптимизации на скорость выполнения

Для анализа влияния флагов оптимизации компилятора были проведены тесты с ключами: `-O0`, `-O1`, `-O2`, `-Os`. Целью эксперимента было определить, как различные уровни оптимизации отражаются на производительности программы.

Полученные результаты позволили сравнить продолжительность выполнения одного и того же фрагмента кода при использовании разных флагов, сделать выводы об эффективности компиляции в зависимости от заданного уровня оптимизации.

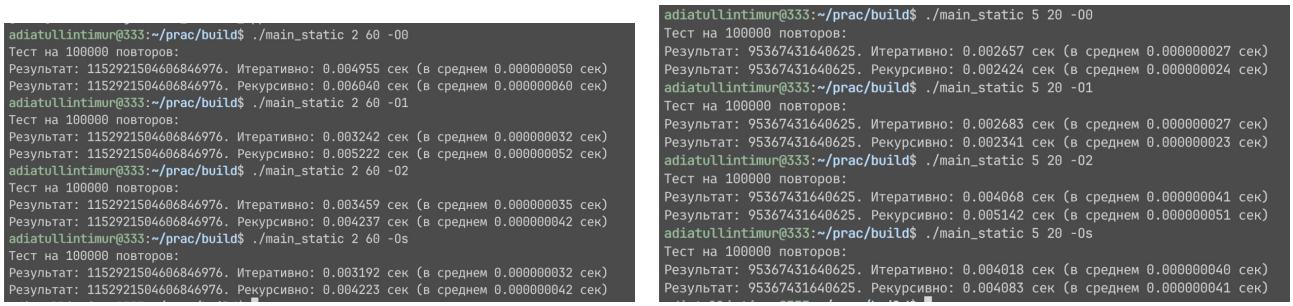

Рис. 27: Тестирование статической

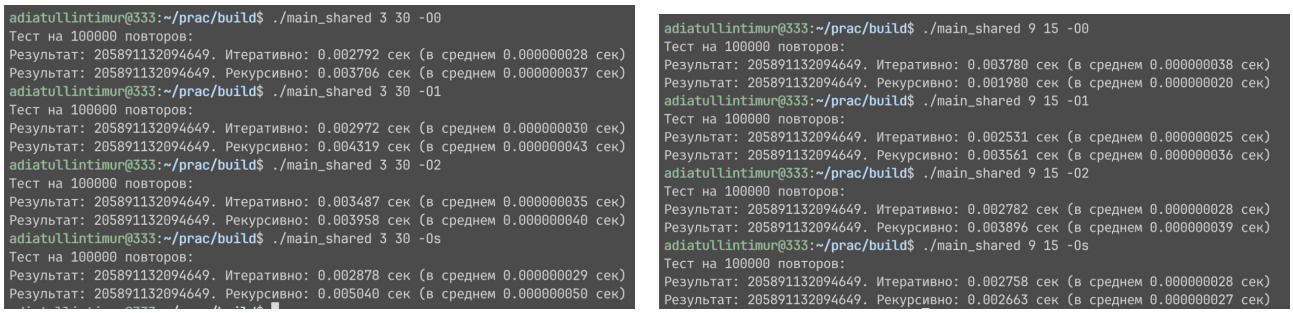

Рис. 28: Тестирование динамической

В ходе тестирования программы с различными уровнями оптимизации были получены следующие результаты:

- **-O0 (отключена оптимизация)**:
  - Итеративно: 0,00000002–0,00000005 с
  - Рекурсивно: 0,00000002–0,00000006 с
  - используется для отладки
- **-O1 (базовая оптимизация)**:
  - Итеративно: 0,00000002–0,00000003 с
  - Рекурсивно: 0,00000002–0,00000005 с

- **-O2 (расширенная оптимизация)**:
  - Итеративно: 0,00000002–0,00000004 с
  - Рекурсивно: 0,00000003–0,00000005 с
- **-Os (оптимизация по размеру)**:
  - Итеративно: 0,00000002–0,00000004 с
  - Рекурсивно: 0,00000002–0,00000005 с
  - Приемлемая скорость

Анализ результатов

- `-O0` — показал наихудшие временные характеристики для обеих реализаций. Этот уровень оптимизации не выполняет никаких преобразований кода, что делает его полезным для отладки, где важна точная связь между исходным кодом и исполняемой программой. Время выполнения при этом высокое и нестабильное.
- `-O1` - даёт некоторое ускорение по сравнению с `-O0`, особенно в рекурсивной реализации, но результаты нестабильные и по времени в ряде случаев хуже, чем у `-O2`. Этот уровень можно рассматривать как компромисс между скоростью компиляции и умеренным ускорением, но он не оптимален для производительности.
- `-O2` — продемонстрировал наилучшую производительность в обоих вариантах реализации (как итеративной, так и рекурсивной). Времена выполнения стабильные, минимальные среди всех уровней. Является предпочтительным флагом для задач, где критична скорость выполнения.
- `-Os` — результаты близки к `-O1` и в некоторых случаях немного лучше, но в целом уступают `-O2`. Основное назначение этого уровня — уменьшение размера исполняемого файла. Оптимален, если приоритет — компактность программы при приемлемом времени выполнения.

3.1.3. Сравнение типов линковок

Статическая линковка (‘-static‘):

- итеративная реализация показала **наименьшее среднее время выполнения** — `0,00000023` с;
- также отличалась **наименьшей вариативностью между запусками**, обеспечивая стабильность результатов;

- размер исполняемого файла значительно увеличился;
- полная независимость от внешних библиотек повышает переносимость и автономность.

Динамическая линковка (‘-shared‘):

- итеративная версия показывала минимальную вариативность между запусками
- незначительно уступает статической по скорости
- обеспечивает меньший размер файла
- требует наличия соответствующих библиотек на целевой системе.

Вывод: выбор между статической и динамической линковкой зависит от приоритетов конкретной задачи.

- если важны стабильность работы и независимость от окружения — предпочтительнее статическая сборка.
- если критичен размер исполняемого файла и допустима зависимость от системных библиотек — рациональнее использовать динамическую линковку.

3.1.4. Анализ зависимостей исполняемого файла

```
adiatullintimur@333:~/prac/build$ make
[ 15%] Built target power_static
[ 30%] Built target power_shared
[ 46%] Built target main_static
[ 61%] Built target main_shared
[ 76%] Built target main_dl
[100%] Built target znak_ncurses_app
adiatullintimur@333:~/prac/build$ ldd main_static
        linux-vdso.so.1 (0x0000ffffa951a000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffffa92f0000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffffa94d0000)
adiatullintimur@333:~/prac/build$ ldd main_shared
        linux-vdso.so.1 (0x0000ffff9bfdc000)
        libpower.so => /home/adiatullintimur/prac/build/libpower.so (0x0000ffff9bf30000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff9bd80000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffff9bf90000)
adiatullintimur@333:~/prac/build$ ldd main_dl
        linux-vdso.so.1 (0x0000ffffa7494000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffffa7270000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffffa7450000)
```

Рис. 29: Результаты команды ldd

Выводы о зависимостях, представленных на рисунке 29

- **main_static**:
  - Не содержит явной зависимости от динамически подключаемой библиотеки `libpower.so`
  - В список зависимостей входят только стандартные системные библиотеки: `libc.so.6`, `ld-linux-aarch64.so.1`
  - Подтверждает факт статической линковки — все необходимые компоненты встроены в исполняемый файл
  - Повышает переносимость, но увеличивает размер
- **main_shared**:
  - Зависит от внешней библиотеки `libpower.so`, путь к которой явно указан
  - Необходимость наличия этой библиотеки в системе при запуске
  - Размер исполняемого файла меньше, чем у статически собранного
  - Облегчает обновление библиотеки без перекомпиляции основного кода
- **main_dl**:
  - Не содержит прямой ссылки на `libpower.so` при проверке ‘ldd‘
  - Подразумевает, что подключение библиотеки осуществляется динамически (через `dlopen()` во время выполнения)
  - Позволяет загружать библиотеки по требованию и реализовать плагиноподобную архитектуру
  - Требует дополнительной обработки ошибок при загрузке

Сходства при наблюдении
Общие черты линковки: статической, динамической и через dlopen

Несмотря на различия в реализации и способе подключения библиотек, все три подхода объединяет ряд характеристик:

- **Работают с одной и той же функциональностью:** независимо от способа подключения, во всех трёх случаях вызываются одни и те же функции из библиотеки поиска — поведение основного кода идентично
- Разница между статической и динамической сборкой подтверждается наличием или отсутствием зависимости от `libpower.so`
- Отсутствие libpower.so в зависимостях main_dl указывает на динамическую загрузку через dlopen

- **Совместимость с Unix-средой:** все подходы опираются на стандартные системные библиотеки (ld-linux-aarch64.so), что обеспечивает переносимость в рамках Linux-среды.

3.2. Результаты тестирования и анализ зависимостей задания 2

3.2.1. Постановка и методика эксперимента

Целью проводимых экспериментов является проверка корректности и устойчивости работы программного обеспечения, реализующего консольную систему управления платёжными поручениями с использованием библиотеки ncurses, а также верификация индивидуального функционала и анализ зависимостей исполняемого файла.

Объект тестирования: приложение `znak_ncurses_app`, реализующее следующие функции:

- добавление новой записи;
- отображение списка записей;
- поиск человека по фамилии;
- сохранение и загрузка базы данных в бинарном виде;
- управление внутренним счётчиком транзакций.

Средства тестирования:

- тестовый файл базы данных `znak.db`;
- интерфейсные действия пользователя в меню программы;
- утилита анализа зависимостей `ldd`;
- просмотр содержимого бинарного файла через hex-редактор.

3.2.2. Главное меню программы

После запуска пользователю отображается главное меню системы записи людей (см. рисунок 30). Интерфейс выполнен в текстовом режиме с использованием библиотеки `ncurses` и предлагает понятную навигацию по основным функциям программы. В меню представлены следующие пункты:

Рис. 30: Меню программы

3.2.3. Создание новой записи

При добавлении новой записи (см. рисунок 31) система запрашивает у пользователя: фамилию, знак Зодиака и дату рождения. После успешного ввода данных программа сообщает, что новая запись добавлена, и предлагает нажать любую клавишу для продолжения работы.

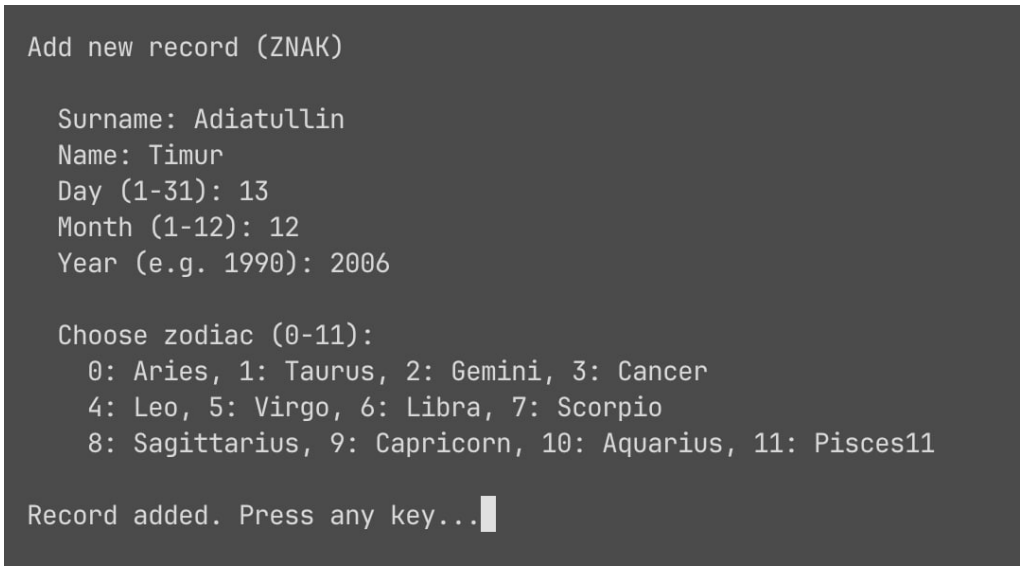

Рис. 31: Добавление записи

Также реализована проверка корректности ввода. Если пользователь, неверно укажет дату рождения, система уведомляет о неправильном вводе и подставляет значение по умолчанию (см. рисунок 32). Это позволяет избежать аварийного завершения программы и обеспечивает стабильность её работы.

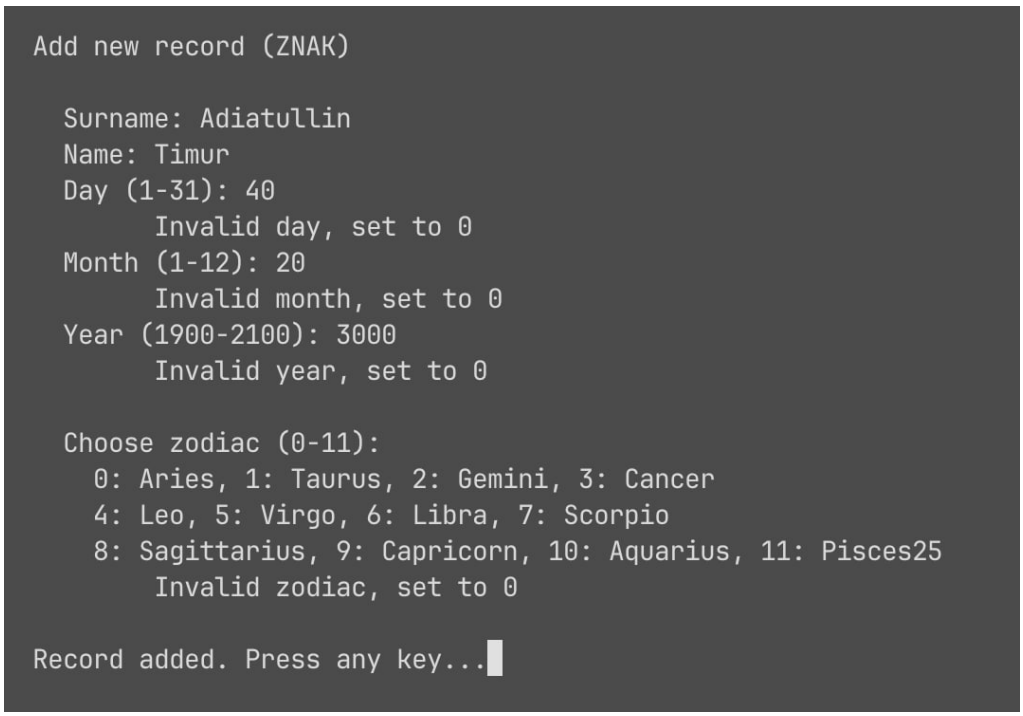

Рис. 32: Обработка пользовательского ввода

После добавления запись появляется в списке

Результат показан на рисунке 33.

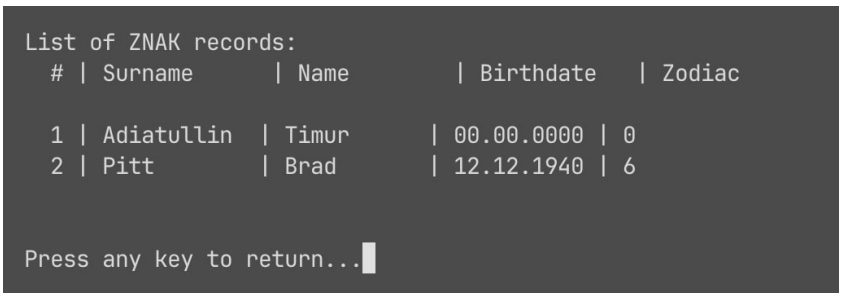

Рис. 33: Список записей

После ввода всех данных программа создает файл базы данных и сохраняет данные в бинарном формате (это можно сделать через команду под номером ’4’). Результат представлен на рисунках 34 и 35:

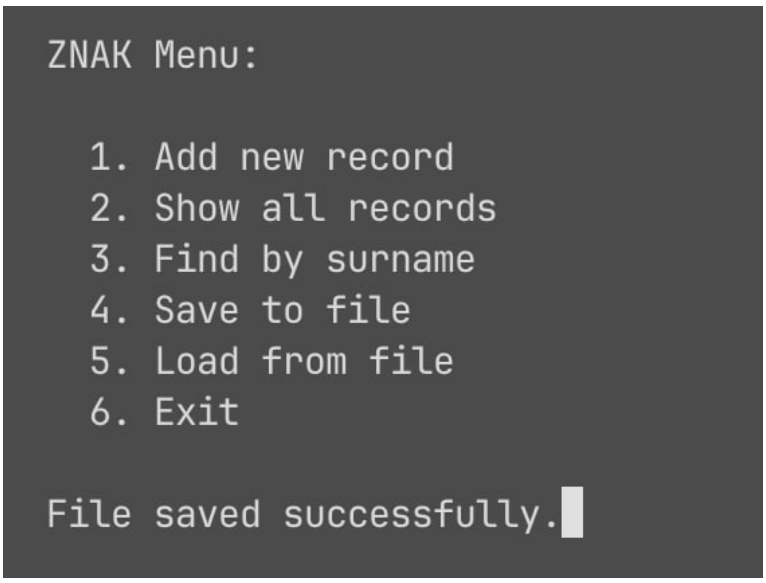

Рис. 34: Сохранение в файл

После ввода всех записей пользователь имеет возможность вывести их все (это можно сделать через команду под номером ’2’). Результат представлен на рисунке 36:

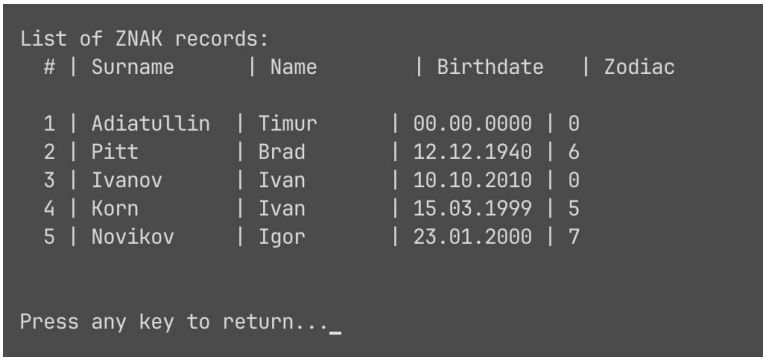

Рис. 36: Вывод базы данных

```
adiatullintimur@333:~/prac/build$ hexdump -C znak.db
00000000  61 64 69 61 08 00 00 00  05 00 00 00 7b 18 90 78  |adia........{..x|
00000010  41 64 69 61 74 75 6c 6c  69 6e 00 00 7e 00 00 00  |Adiatullin..~...|
00000020  2d 00 00 00 2d 00 00 00  00 00 00 00 00 00 00 00  |-...-...........|
00000030  54 69 6d 75 72 00 00 00  01 00 00 00 00 00 00 00  |Timur...........|
00000040  00 00 00 00 00 00 00 00  20 8b 13 e7 aa aa 00 00  |........ .......|
00000050  00 00 00 00 00 00 00 00  50 69 74 74 00 00 00 00  |........Pitt....|
00000060  20 00 00 00 20 00 00 00  20 00 00 00 20 00 00 00  | ... ... ... ...|
00000070  20 00 00 00 20 00 00 00  42 72 61 64 00 ff 00 00  | ... ...Brad....|
00000080  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  20 8b 13 e7 aa aa 00 00  06 00 00 00 0c 0c 94 07  | ...............|
000000a0  49 76 61 6e 6f 76 00 00  20 00 00 00 20 00 00 00  |Ivanov.. ... ...|
000000b0  20 00 00 00 20 00 00 00  20 00 00 00 20 00 00 00  | ... ... ... ...|
000000c0  49 76 61 6e 00 ff 00 00  01 00 00 00 00 00 00 00  |Ivan............|
000000d0  00 00 00 00 00 00 00 00  20 8b 13 e7 aa aa 00 00  |........ .......|
000000e0  00 00 00 00 0a 0a da 07  4b 6f 72 6e 00 00 00 00  |........Korn....|
000000f0  20 00 00 00 20 00 00 00  20 00 00 00 20 00 00 00  | ... ... ... ...|
00000100  20 00 00 00 20 00 00 00  49 76 61 6e 00 ff 00 00  | ... ...Ivan....|
00000110  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  20 8b 13 e7 aa aa 00 00  05 00 00 00 0f 03 cf 07  | ...............|
00000130  4e 6f 76 69 6b 6f 76 00  20 00 00 00 20 00 00 00  |Novikov. ... ...|
00000140  20 00 00 00 20 00 00 00  20 00 00 00 20 00 00 00  | ... ... ... ...|
00000150  49 67 6f 72 00 ff 00 00  01 00 00 00 00 00 00 00  |Igor............|
00000160  00 00 00 00 00 00 00 00  20 8b 13 e7 aa aa 00 00  |........ .......|
00000170  07 00 00 00 17 01 d0 07                           |........|
00000178
```

Рис. 35: Файл базы данных в шестнадцатиричном виде

3.2.4. Тестирование индивидуального задания

Для достоверной проверки работы программы были рассмотрены несколько случаев:

1. Поиск информации о людях по фамилии, введённой с клавиатуры, когда такая фамилия существует в базе (см. рисунок 37).

2. Поиск по фамилии, которой нет среди данных (см. рисунок 38).

```
Enter surname to search: Krasavin

Results for surname 'Krasavin':

  Krasavin Alex, 10.01.2006, Zodiac=2
```

Рис. 37: Пример поиска по существующей фамилии

```
Enter surname to search: Novikov

Results for surname 'Novikov':

No people found with surname 'Novikov'.
```

Рис. 38: Результат поиска по отсутствующей фамилии

3.2.5. Увеличение счетчика транзакций при доступе к базе данных

При каждом обращении к базе данных значение счётчика транзакций, хранящегося в заголовке файла, увеличивается. Это позволяет отслеживать все произведённые операции. Изменение счётчика можно наблюдать при просмотре файла в шестнадцатеричном формате (см. рисунки 39, 40).

```
adiatullintimur@333:~/prac/build$ hexdump -C znak.db
00000000  61 64 69 61 03 00 00 00  01 00 00 00 d1 ea 97 bf  |adia............|
00000010  41 64 69 61 74 75 6c 6c  69 6e 00 00 7e 00 00 00  |Adiatullin..~...|
00000020  2d 00 00 00 2d 00 00 00  00 00 00 00 00 00 00 00  |-...-...........|
00000030  54 69 6d 75 72 00 00 00  01 00 00 00 00 00 00 00  |Timur...........|
00000040  00 00 00 00 00 00 00 00  20 8b 08 e6 aa aa 00 00  |........ .......|
00000050  06 00 00 00 0d 0c d6 07                           |........|
00000058
adiatullintimur@333:~/prac/build$ _
```

```
ZNAK Menu:

  1. Add new record
  2. Show all records
  3. Find by surname
  4. Save to file
  5. Load from file
  6. Exit

File saved successfully.
```

```
adiatullintimur@333:~/prac/build$ hexdump -C znak.db
00000000  61 64 69 61 03 00 00 00  01 00 00 00 d1 ea 97 bf  |adia............|
00000010  41 64 69 61 74 75 6c 6c  69 6e 00 00 7e 00 00 00  |Adiatullin..~...|
00000020  2d 00 00 00 2d 00 00 00  00 00 00 00 00 00 00 00  |-...-...........|
00000030  54 69 6d 75 72 00 00 00  01 00 00 00 00 00 00 00  |Timur...........|
00000040  00 00 00 00 00 00 00 00  20 8b 08 e6 aa aa 00 00  |........ .......|
00000050  06 00 00 00 0d 0c d6 07                           |........|
00000058
adiatullintimur@333:~/prac/build$ hexdump -C znak.db
00000000  61 64 69 61 05 00 00 00  02 00 00 00 6d 1d 1c ad  |adia........m...|
00000010  41 64 69 61 74 75 6c 6c  69 6e 00 00 7e 00 00 00  |Adiatullin..~...|
00000020  2d 00 00 00 2d 00 00 00  00 00 00 00 00 00 00 00  |-...-...........|
00000030  54 69 6d 75 72 00 00 00  01 00 00 00 00 00 00 00  |Timur...........|
00000040  00 00 00 00 00 00 00 00  20 8b 08 e6 aa aa 00 00  |........ .......|
00000050  06 00 00 00 0d 0c d6 07  49 76 61 6e 6f 76 00 00  |........Ivanov..|
00000060  20 00 00 00 20 00 00 00  20 00 00 00 20 00 00 00  | ... ... ... ...|
00000070  20 00 00 00 20 00 00 00  49 76 61 6e 00 ff 00 00  | ... ...Ivan....|
00000080  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  20 8b 08 e6 aa aa 00 00  0b 00 00 00 0c 0c dc 07  | ...............|
000000a0
adiatullintimur@333:~/prac/build$ _
```

```
ZNAK Menu:

  1. Add new record
  2. Show all records
  3. Find by surname
  4. Save to file
  5. Load from file
  6. Exit

File saved successfully.
```

Рис. 39: Изменение в файле после добавления и сохранения записи

Счётчик транзакций, хранящийся в структуре `DB_HEADER`, увеличивается при каждом действии с базой данных — создании файла, записи, чтения, выполнении поиска по фамилии. Это позволяет фиксировать количество операций и проверять ожидаемое значение при чтении базы, тем самым обеспечивая контроль целостности. На рисунке 39 поле счётчика, расположенное сразу после сигнатуры файла, поначалу занимает 88 байт (`uint32_t`), но после загрузки из файла и сохранения в файл (это можно сделать через команду под номером ’5’ и ’4’) уже занимает 160 байта (`uint32_t`).

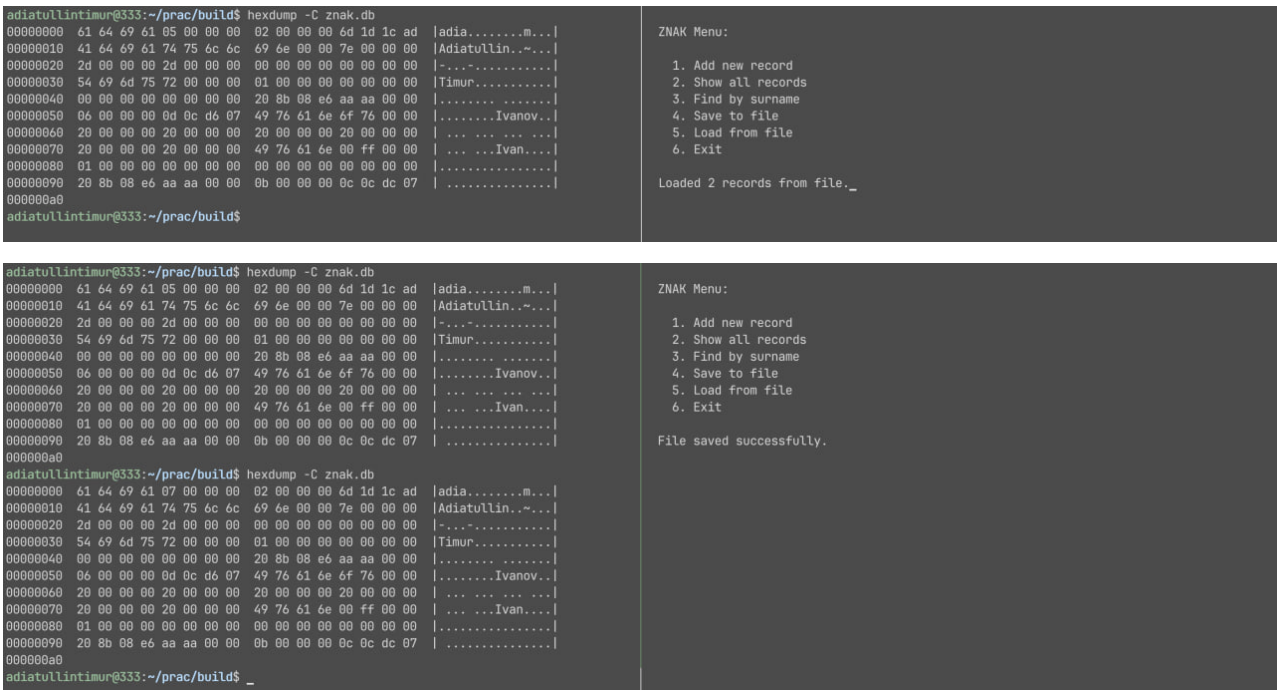

Рис. 40: Изменение в файле после загрузки и сохранения

3.2.6. Анализ зависимостей исполняемого файла

Для анализа зависимостей исполняемого файла использовалась утилита `ldd`, позволяющая определить динамические библиотеки, необходимые для запуска программы. Результат выполнения команды `ldd ./znak_ncurses_app` представлен ниже на рисунке 41:

```
adiatullintimur@333:~/prac/build$ ldd ./znak_ncurses_app
        linux-vdso.so.1 (0x0000ffff894fd000)
        libncurses.so.6 => /lib/aarch64-linux-gnu/libncurses.so.6 (0x0000ffff89440000)
        libtinfo.so.6 => /lib/aarch64-linux-gnu/libtinfo.so.6 (0x0000ffff893f0000)
        libz.so.1 => /lib/aarch64-linux-gnu/libz.so.1 (0x0000ffff893b0000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff89200000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffff894c0000)
```

Рис. 41: Результат выполнения команды `ldd ./znak_ncurses_app`

Интерпретация результатов:

- **linux-vdso.so.1** — специальная виртуальная библиотека, ускоряющая выполнение системных вызовов
- **libncurses.so.6** — отвечает за отображение текстового интерфейса и взаимодействие с терминалом
- **libtinfo.so.6** — низкоуровневая часть `ncurses`, обеспечивающая работу с терминальными возможностями
- **libz.so.1** — библиотека для работы с сжатием данных и проверки их целостности (например, с использованием CRC32)
- **libc.so.6** — основная системная библиотека C, включающая базовые функции управления памятью, строками и вводом-выводом
- **ld-linux-aarch64.so.1** — динамический загрузчик программ в 64-битных Linux-системах на архитектуре ARM (AArch64), отвечающий за поиск и подключение необходимых динамических библиотек при запуске приложений.

Дерево зависимостей `znak_ncurses_app`

Представленная конфигурация (см. рисунок 42) полностью соответствует заявленным техническим требованиям: библиотека `ncurses` используется для реализации пользовательского интерфейса, а `zlib` — для вычисления контрольных сумм. Обе библиотеки имеют стандартную зависимость от системной библиотеки языка C, что типично для Unix-подобных операционных систем.

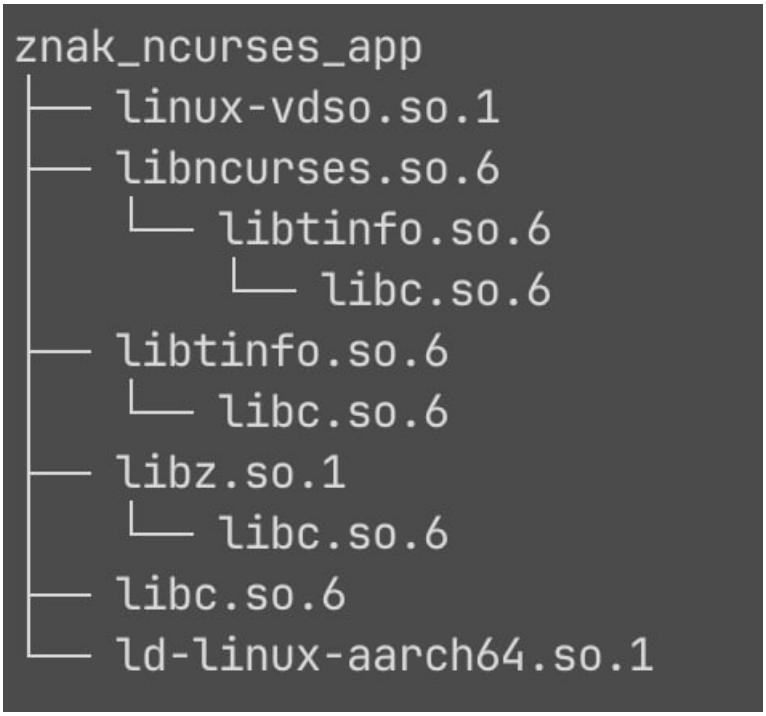

Рис. 42: Дерево зависимостей `znak_ncurses_app`

Заключение

В ходе выполнения лабораторной работы была последовательно проведена всесторонняя оценка различных аспектов сборки и функционирования программного обеспечения, связанного с записью данных о людях и алгоритмами возведения в степень целого числа. Исследование охватило несколько ключевых направлений, каждое из которых дополняло общее понимание влияния архитектурных решений на производительность, структуру зависимостей и удобство эксплуатации программного продукта.

1.1 Была успешно выполнена установка VirtualBox и установка Debian 12 без графической оболочки. Создан пользователь с логином на основе фамилии, настроен проброс порта 22 и выполнено подключение к системе по SSH.

1.2 Освоены ключевые команды терминала, работа с историей команд, а также настройка входа по SSH с использованием ключевой аутентификации. Были выполнены типовые команды и проверены man-страницы с пояснением специфики.

Произведена установка компилятора, утилит make и cmake. Среда разработки подготовлена для выполнения практических заданий.

1.3.1 Реализованы две версии алгоритма возведения в степень целого числа: итеративная и рекурсивная. Созданы статическая и динамическая версии библиотеки libpower. Для обеих реализаций проведено сравнительное тестирование производительности на множестве повторов с флагами компиляции -O0, -O1, -O2 и -Os. Зафиксировано, что итеративная версия демонстрировала наименьшее время выполнения при разных уровнях оптимизации.

Также реализована загрузка динамической библиотеки во время выполнения с помощью dlopen().

1.3.2 Создана структура ZNAK, включающая поля для фамилии, имени, даты рождения и знака зодиака. Разработана структура заголовка DB_HEADER, содержащая сигнатуру, номер транзакции, количество записей и CRC32. Реализована система взаимодействия с бинарным файлом znak.db: создание, дозапись и обновление заголовка.

Создан пользовательский интерфейс на базе ncurses с функциями: добавления записи, отображения списка записей, поиск записи по фамилии человека

Проанализированы зависимости каждого исполняемого файла с помощью утилиты ldd, составлено их рекурсивное дерево.

Таким образом, поставленные цели были достигнуты, а полученные результаты могут служить базой для построения более сложных систем с учётом требований к производительности, модульности и сопровождаемости.

Список литературы

[1] Debian Documentation. Официальный сайт Debian.
URL: https://www.debian.org/doc/ (дата обращения: 20.08.2025)

[2] CMake Reference Documentation. Официальный сайт CMake.
URL: https://cmake.org/documentation/ (дата обращения: 21.08.2025)

[3] Oracle VirtualBox.
URL: https://www.virtualbox.org/ (дата обращения: 20.09.2025)

[4] Новые проклятия: руководство по ncurses / Хабр.
URL: https://habr.com/ru/articles/778040/ (дата обращения: 22.08.2025)

Полный исходный код

power.h

```h
#pragma once
#include <stdio.h>

unsigned long long power_iterative(int base, int exp);
unsigned long long power_recursive(int base, int exp);
```

power.c

```c
#include "headers/power.h"
#include <stdio.h>

unsigned long long power_iterative(int base, int exp) {
    unsigned long long result = 1;
    unsigned long long b = base;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result *= b;
        }
        b *= b;
        exp /= 2;
    }
    return result;
}

unsigned long long power_recursive(int base, int exp) {
    if (exp == 0) return 1;
    if (exp == 1) return base;
    unsigned long long half = power_recursive(base, exp / 2);
    if (exp % 2 == 0)
        return half * half;
    else
        return half * half * base;
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define REPEATS 100000

#include "src/headers/power.h"
```

```c

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Usage: %s <base> <exponent>\n", argv[0]);
        return 1;
    }

    int base = atoi(argv[1]);
    int exp = atoi(argv[2]);

    clock_t start, end;
    unsigned long long result_iter = 0, result_rec = 0;
    double iter_total = 0.0, rec_total = 0.0;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_rec = power_recursive(base, exp);
    end = clock();
    rec_total = (double)(end - start) / CLOCKS_PER_SEC;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_iter = power_iterative(base, exp);
    end = clock();
    iter_total = (double)(end - start) / CLOCKS_PER_SEC;

    printf("     %d     :\n", REPEATS);
    printf("     : %llu.      : %.6f    (      %.9f   )\n",
           result_iter, iter_total, iter_total / REPEATS);
    printf("     : %llu.      : %.6f    (      %.9f   )\n",
           result_rec, rec_total, rec_total / REPEATS);

    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(power_prac C)

set(CMAKE_C_STANDARD 11)
include_directories(headers)

add_library(power_static STATIC src/power.c)
set_target_properties(power_static PROPERTIES OUTPUT_NAME "power")

add_library(power_shared SHARED src/power.c)
set_target_properties(power_shared PROPERTIES OUTPUT_NAME "power")

add_executable(main_static main.c)
```

```
target_link_libraries(main_static power_static)

add_executable(main_shared main.c)
target_link_libraries(main_shared power_shared)

add_executable(main_dl main_dl.c)
target_link_libraries(main_dl dl)

add_executable(znak_ncurses_app znak_ncurses_app.c src/znak.c)

target_include_directories(znak_ncurses_app PRIVATE src/headers)

target_link_libraries(znak_ncurses_app ncurses z)


add_custom_target(db
  COMMAND $(PROJECT_SOURCE_DIR)/build/znak_ncurses_app
  WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
  COMMENT "running db"
)
```

main_dl.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <time.h>

#define REPEATS 100000

typedef unsigned long long (*power_iter_t)(int, int);
typedef unsigned long long (*power_rec_t)(int, int);

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Usage: %s <base> <exponent>\n", argv[0]);
        return 1;
    }

    int base = atoi(argv[1]);
    int exp = atoi(argv[2]);

    void *handle = dlopen("./libpower.so", RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, "    dlopen: %s\n", dlerror());
        return 1;
    }

    power_iter_t power_iterative = (power_iter_t)dlsym(handle, "
   power_iterative");
```

```
    power_rec_t power_recursive = (power_rec_t)dlsym(handle, "
   power_recursive");

    char *error;
    if ((error = dlerror()) != NULL) {
        fprintf(stderr, "    dlsym: %s\n", error);
        dlclose(handle);
        return 1;
    }

    clock_t start, end;
    unsigned long long result_iter = 0, result_rec = 0;
    double iter_total = 0.0, rec_total = 0.0;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_rec = power_recursive(base, exp);
    end = clock();
    rec_total = (double)(end - start) / CLOCKS_PER_SEC;

    start = clock();
    for (int i = 0; i < REPEATS; ++i)
        result_iter = power_iterative(base, exp);
    end = clock();
    iter_total = (double)(end - start) / CLOCKS_PER_SEC;

    printf("     %d     :\n", REPEATS);
    printf("     : %llu.       (dlopen): %.6f    (      %.9f   )\n",
           result_iter, iter_total, iter_total / REPEATS);
    printf("     : %llu.       (dlopen): %.6f    (      %.9f   )\n",
           result_rec, rec_total, rec_total / REPEATS);

    dlclose(handle);
    return 0;
}
```

znak.h

```
#pragma once

#include <stdint.h>
#include <stddef.h>

typedef enum {
    ARIES, TAURUS, GEMINI, CANCER,
    LEO, VIRGO, LIBRA, SCORPIO,
    SAGITTARIUS, CAPRICORN, AQUARIUS, PISCES
} ZODIAC;

typedef struct {
```

```
    uint8_t day   : 5;  // -031
    uint8_t month : 4;  // -012
    uint16_t year : 12; // -04095
} BIRTHDAYDATE;


typedef struct {
    char surname[32];
    char name[32];
    ZODIAC zodiac;
    BIRTHDAYDATE birth;
} ZNAK;


typedef struct {
    char signature[4];
    uint32_t transaction_id;
    uint32_t record_count;
    uint32_t crc32;
} DB_HEADER;


uint32_t calculate_crc32(ZNAK *records, size_t count);
```

znak.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <zlib.h>
#include "headers/znak.h"

uint32_t calculate_crc32(ZNAK *records, size_t count) {
    return crc32(0L, (const unsigned char *)records, count * sizeof
   (ZNAK));
}
```

znak_ncurses_app.c

```
#include <ncurses.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <zlib.h>
#include "src/headers/znak.h"

#define MAX_RECORDS 100
#define DB_PATH "znak.db"
```

```
10
11  ZNAK records[MAX_RECORDS];
12  DB_HEADER header;
13  int record_count = 0;
14  uint32_t current_tx_id = 1; //
15
16  void input_record() {
17      ZNAK z;
18      char tmp[50];
19      int zodiac_choice;
20
21      clear();
22      mvprintw(1, 2, "Add new record (ZNAK)");
23
24      mvprintw(3, 4, "Surname: ");
25      echo();
26      getstr(z.surname);
27
28      mvprintw(4, 4, "Name: ");
29      getstr(z.name);
30
31      //
32      mvprintw(5, 4, "Day (1-31): ");
33      getstr(tmp);
34      int day = atoi(tmp);
35
36      if (day < 1 || day > 31) {
37          z.birth.day = 0;
38          mvprintw(6, 10, "Invalid day, set to 0");
39      } else {
40          z.birth.day = day;
41      }
42
43      //
44      mvprintw(7, 4, "Month (1-12): ");
45      getstr(tmp);
46      int month = atoi(tmp);
47      if (month < 1 || month > 12) {
48          z.birth.month = 0;
49          mvprintw(8, 10, "Invalid month, set to 0");
50      } else {
51          z.birth.month = month;
52      }
53
54      mvprintw(9, 4, "Year (1900-2100): ");
55      getstr(tmp);
56      int year = atoi(tmp);
57      if (year < 1900 || year > 2100) {
58          z.birth.year = 0;
59          mvprintw(10, 10, "Invalid year, set to 0");
60      } else {
61          z.birth.year = year;
```

```
    }

    mvprintw(12, 4, "Choose zodiac (0-11):");
    mvprintw(13, 6, "0: Aries, 1: Taurus, 2: Gemini, 3: Cancer");
    mvprintw(14, 6, "4: Leo, 5: Virgo, 6: Libra, 7: Scorpio");
    mvprintw(15, 6, "8: Sagittarius, 9: Capricorn, 10: Aquarius,
   11: Pisces");
    getstr(tmp);
    zodiac_choice = atoi(tmp);
    if (zodiac_choice < 0 || zodiac_choice > 11) {
        zodiac_choice = 0;
        mvprintw(16, 10, "Invalid zodiac, set to 0");
    }
    z.zodiac = (ZODIAC)zodiac_choice;

    noecho();
    records[record_count++] = z;

    mvprintw(18, 2, "Record added. Press any key...");
    getch();
    header.transaction_id = current_tx_id++;
}

void show_records() {
    clear();
    mvprintw(1, 2, "List of ZNAK records:");
    mvprintw(2, 2, "  # | Surname      | Name         | Birthdate
   | Zodiac");

    for (int i = 0; i < record_count; ++i) {
        mvprintw(4 + i, 2, " %2d | %-11s | %-10s | %02d.%02d.%04d |
    %d",
                 i + 1,
                 records[i].surname,
                 records[i].name,
                 records[i].birth.day,
                 records[i].birth.month,
                 records[i].birth.year,
                 records[i].zodiac);
    }

    mvprintw(6 + record_count, 2, "Press any key to return...");
    getch();
}

void find_by_surname() {
    char tmp[32];
    clear();
    mvprintw(1, 2, "Enter surname to search: ");
    echo();
    getstr(tmp);
    noecho();
```

```
111
112     int found = 0;
113     mvprintw(3, 2, "Results for surname '%s':", tmp);
114
115     for (int i = 0; i < record_count; ++i) {
116         if (strcmp(records[i].surname, tmp) == 0) {
117             mvprintw(5 + found, 4, "%s %s, %02d.%02d.%04d, Zodiac=%
    d",
118                     records[i].surname,
119                     records[i].name,
120                     records[i].birth.day,
121                     records[i].birth.month,
122                     records[i].birth.year,
123                     records[i].zodiac);
124             found++;
125         }
126     }
127
128     if (!found) {
129         mvprintw(5, 2, "No people found with surname '%s'.", tmp);
130     }
131
132     getch();
133     header.transaction_id = current_tx_id++;
134 }
135
136 void save_to_file() {
137     FILE *f = fopen(DB_PATH, "wb");
138     if (!f) {
139         mvprintw(10, 2, "Could not open file for writing.");
140         return;
141     }
142
143     memcpy(header.signature, "adia", 4);
144     header.transaction_id = ++current_tx_id;
145     header.record_count = record_count;
146     header.crc32 = calculate_crc32(records, record_count);
147
148     fwrite(&header, sizeof(DB_HEADER), 1, f);
149     fwrite(records, sizeof(ZNAK), record_count, f);
150     fclose(f);
151
152     mvprintw(10, 2, "File saved successfully.");
153 }
154
155 void load_from_file() {
156     FILE *f = fopen(DB_PATH, "rb");
157     if (!f) {
158         mvprintw(10, 2, "File not found. Please enter new data.");
159         return;
160     }
161
```

```
    fread(&header, sizeof(DB_HEADER), 1, f);
    fread(records, sizeof(ZNAK), header.record_count, f);
    record_count = header.record_count;
    current_tx_id = header.transaction_id + 1;
    fclose(f);

    mvprintw(10, 2, "Loaded %d records from file.", record_count);
}

void run_ui() {
    initscr();
    cbreak();
    noecho();

    int choice;
    while (true) {
        clear();
        mvprintw(1, 2, "ZNAK Menu:");
        mvprintw(3, 4, "1. Add new record");
        mvprintw(4, 4, "2. Show all records");
        mvprintw(5, 4, "3. Find by surname");
        mvprintw(6, 4, "4. Save to file");
        mvprintw(7, 4, "5. Load from file");
        mvprintw(8, 4, "6. Exit");

        mvprintw(10, 2, "Your choice: ");
        echo();
        scanw("%d", &choice);
        noecho();

        switch (choice) {
            case 1: input_record(); break;
            case 2: show_records(); break;
            case 3: find_by_surname(); break;
            case 4: save_to_file(); getch(); break;
            case 5: load_from_file(); getch(); break;
            case 6: endwin(); return;
            default: break;
        }
    }
}

int main() {
    run_ui();
    return 0;
}
```