Министерство образования и науки Российской Федерации

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| «Утверждаю» | | | | |
| Зав. кафедрой «МО и ПЭВМ» | | | | |
|  | | | Макарычев П.П. | |
| « » |  |  | | 2018 |

**Пояснительная записка**

к курсовой работе по дисциплине

«Операционные системы»

на тему: «Моделирование системы управления хранилищем нефти»

|  |  |
| --- | --- |
| Автор работы: | Дунаев В.Ю. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсового проекта | ПГУ 09.03.04 - 06КР161.11 ПЗ |
| Группа | 16ВП1 |
| Руководитель работы | Шашков Б.Д., к.т.н., профессор |
| Работа защищена «\_\_» \_\_\_\_ 2019 г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2019

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Реферат**  Пояснительная записка содержит 47 листов, 6 рисунков, 1 таблицу, 5 использованных источников, 2 приложения.  ОПЕРАЦИОННЫЙ СИСТЕМЫ, UNIX, ПРИЛОЖЕНИЯ С ПАРАЛЛЕЛЬНЫМИ ПРОЦЕССАМИ, СИГНАЛЫ  Объектом исследования являются средства управления параллельными процессами в UNIX.  Цель работы – разработка программы, моделирующей систему управления железнодорожным переездом.  Разработка проводилась на языке программирования C++ в среде программирования Code::blocks.  Осуществлено функциональное тестирование разработанного программного обеспечения, которое показало корректность его работы. | | | | | | | | | | |
|  |  |  |  |  | ПГУ 09.03.04 - 06КР161.24 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № докум. | Подп. | Дата |
| Разраб. | | Угроватов Д.В. |  |  | «Моделирование системы управления железнодорожным переездом»  Пояснительная записка. | Лит. | | | Лист | Листов |
| Пров. | | Шашков Б.Д. |  |  |  |  |  | 3 | 47 |
|  | |  |  |  | Группа 16ВП1 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

# Оглавление

[Введение 5](#_Toc8676648)

[1. Разработка системы управления хранилищем нефти 6](#_Toc8676649)

[1.1. Постановка задачи 6](#_Toc8676650)

[1.2. Анализ требований 7](#_Toc8676651)

[1.2.1. Функциональные требования 7](#_Toc8676652)

[1.2.2. Нефункциональные требования 9](#_Toc8676653)

[1.3. Проектирование 11](#_Toc8676654)

[1.3.1. Проектирование структур и функций 11](#_Toc8676655)

[1.3.2. Проектирование параллельных вычислений 13](#_Toc8676656)

[1.3.3. Проектирование интерфейса 15](#_Toc8676657)

[1.4. Реализация программы 17](#_Toc8676658)

[1.5. Тестирование 18](#_Toc8676659)

[Заключение 19](#_Toc8676660)

[Список литературы 20](#_Toc8676661)

[Приложение А – Исходный код программы 21](#_Toc8676662)

[Приложение B – Результаты тестирования 46](#_Toc8676663)

# Введение

Одной из самых актуальных задач в IT-сфере является создание приложений с параллельными вычислениями. В таких приложения вычисления, производимые компьютером, происходят одновременно. Это позволяется значительно повысить эффективность работы программ.

Основная сложность при проектировании параллельных программ – обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами.

Современные операционные системы предоставляют разнообразные средства для создания приложения с параллельными вычисления. Одной из таких операционных систем является Linux.

Linux – семейство Unix-подобных операционных систем, включающих тот или иной набор утилит и программ проекта GNU

Unix – семейство переносимых, многозадачных и многопользовательских операционных систем.

Одна из важных особенностей операционной системы Linux – поддержка стандартов POSIX.

POSIX (Portable Operating System Interface) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой, библиотеку языка C и набор приложений и их интерфейсов.

Целью данной работы является создание программы, моделирующей систему управления хранилищем нефти.

В рамках данной работы будут пройдены следующие этапы:

* Составлены требования к программе
* Спроектирована программа
* Реализована программа
* Протестирована программа
* Сделаны выводы

# Разработка системы управления хранилищем нефти

# Постановка задачи

В рамках курсовой работы необходимо разработать программу, моделирующую работу хранилища нефти.

Хранилище нефти состоит из нескольких резервуаров и предназначено для накопления, хранения и отгрузки нефтепродуктов. Каждый из резервуаров предназначен для работы с определенным видом нефтепродуктов. Резервуар может находиться в рабочем или нерабочем состоянии. Каждый из резервуаров оборудован двумя насосами, один из которых осуществляем закачку нефти в резервуар, а другой – откачку нефти из резервуара. Каждый из насосов может находиться во включенном или выключенном состоянии. Оба насоса каждого из резервуаров могут работать одновременно. Откачка нефти из резервуара возможно только тогда, когда уровень нефтепродуктов в резервуаре не менее определенного нижнего допустимого уровня. Закачка нефти в резервуар возможно только тогда, когда уровень нефтепродуктов в резервуаре не превышает определенного верхнего уровня. Насосы каждого резервуара управляются пользователем в ручном режиме. При этом автоматически контролируется уровень нефтепродуктов в резервуаре, и, в случае необходимости, любой из насосов может быть автоматически отключен. Пользователь может изменять состояние и режимы работы резервуаров нефтехранилища, включать и выключать насосы. Кроме того, пользователь может изменять параметры системы: нижний и верхний допустимый уровень нефтепродуктов для каждого резервуара, и скорость перекачки нефтепродуктов каждым из насосов.

Способ обмена информацией между параллельными процессами – каналы.

Разработка программа должна быть выполнена на языке C или С++.

Программное обеспечение должно быть полностью отлажено и протестировано, функционировать под управлением ОС семейства Linux.

Требования к интерфейсу пользователя отсутствуют.

# Анализ требований

# Функциональные требования

Анализ задания на разработку позволяет выделить следующие варианты использования (Рисунок 1).

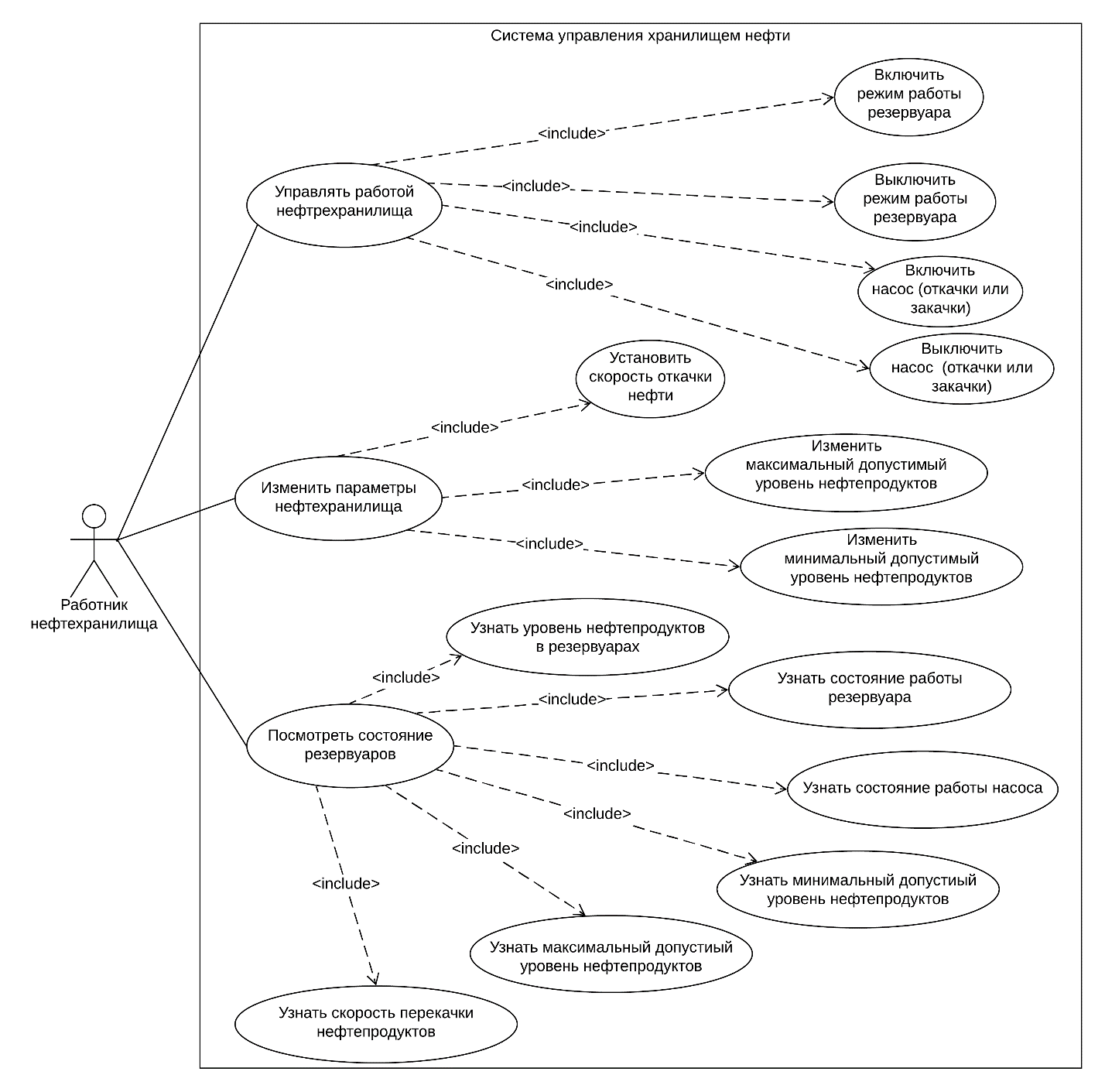


Рисунок 1 – Диаграмма вариантов использования

Была составлена спецификация для некоторых прецедентов из диаграммы вариантов использования (Рисунок 2 - 4).

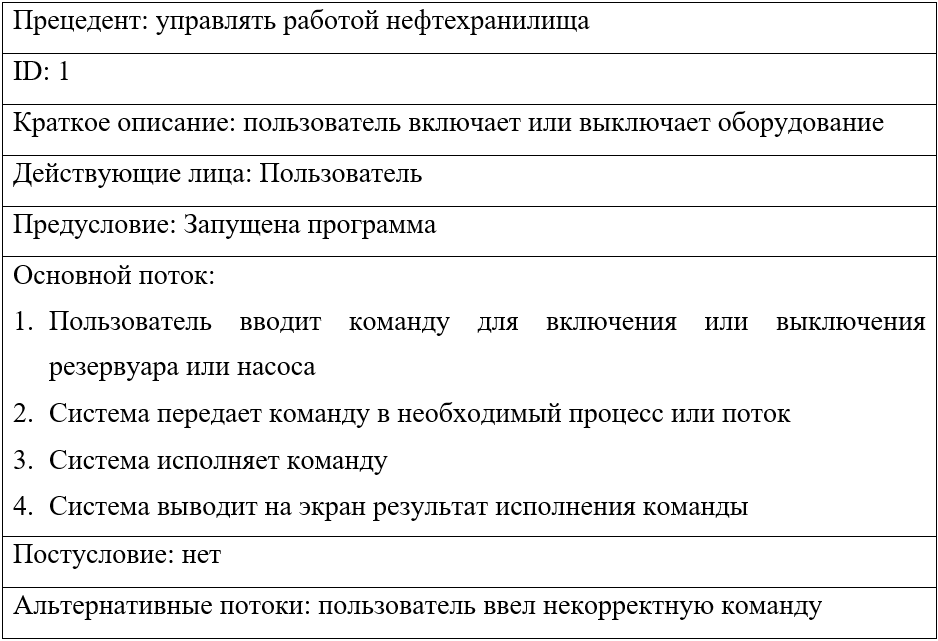


Рисунок 2 – Спецификация прецедента «Управлять работой нефтехранилища»

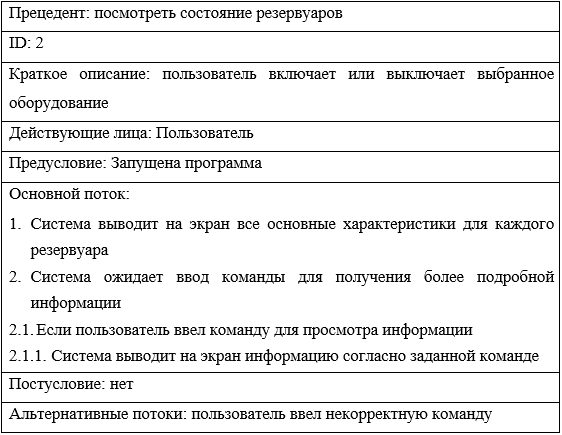


Рисунок 2 – Спецификация прецедента «Посмотреть состояние резервуаров»

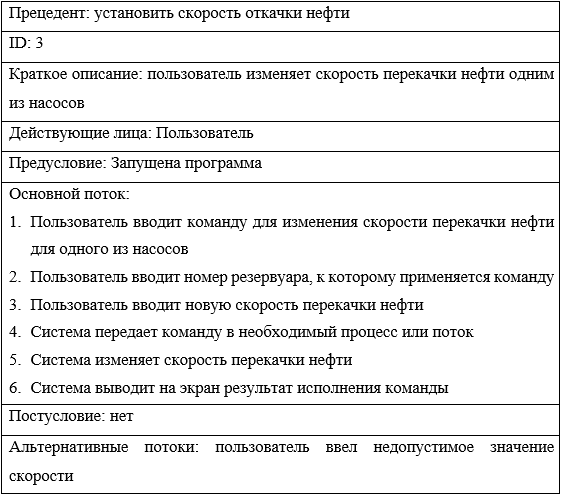


Рисунок 4 - Спецификация прецедента «Установить скорость откачки нефти»

# Нефункциональные требования

Одним из основных требований к программе является то, что все резервуары в хранилище нефти должны работать одновременно. Кроме того, каждый из резервуаров оборудован двумя насосами, один из которых осуществляем закачку нефти в резервуар, а другой – откачку нефти из резервуара. Оба насоса каждого из резервуаров могут работать одновременно.

Исходя из вышеописанных требований можно сделать вывод, что система должна осуществлять параллельные вычисления: хранилище нефти должно параллельно моделировать работу каждого из резервуаров, а в каждом резервуаре должны параллельно работать насос закачки и насос откачки.

Также к системе предъявляется требование автоматического контроля уровня нефтепродуктов в резервуарах (уровень нефти всегда должен быть между заданным минимально допустимым и максимально допустимым уровнем). Кроме того, для системы автоматического контроля желательно обеспечить независимость ее работоспособности от работоспособности насосов.

Исходя из этих требований, можно сделать вывод, что в системе должны существовать параллельные вычисления для контроля уровня нефти в резервуаре.

Таким образом в будущей программе необходимо реализовать достаточно много параллельных вычислений. Одной из важных задач при проектировании такой программы является необходимость обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами. Для решения данной задачи сформулировано еще одно нефункциональное требование – «способ обмена информацией между параллельными процессами – каналы».

# Проектирование

# Проектирование структур и функций

Для проектирования структур и функций были выделены следующие сущности:

* Насос
* Резервуар
* Хранилище нефти

Проектирование структур и функций для сущности «Насос»:

Структура:

/\*\*

\* насос для перекачки нефтепродуктов

\*/

struct pump {

/\*\*

\* текущее значение нефти (величина, которая будет изменяться)

\*/

int\* value;

/\*\*

\* скорость перекачки (величина, на которую будет изменять уровень нефти)

\*/

int delta;

/\*\*

\* состояние работы насоса(PUMP\_ON - включен, PUMP\_OFF выключен)

\*/

int state;

};

Функции:

* Создать насос
* Включить насос
* Выключить насос
* Получить состояние работы насоса (включен или выключен)
* Установить скорость перекачки нефти
* Удалить насос

Проектирование структур и функций для сущности «Резервуар»:

Структура:

/\*\*

\* резервуар для хранения нефтепродуктов

\*/

struct storage\_tank {

/\*\*

\* минимальный уровень нефтепродуктов в резервуаре

\*/

unsigned int minimum\_level;

/\*\*

\* максимальный уровень нефтепродуктов в резервуаре

\*/

unsigned int maximum\_level;

/\*\*

\* уровень нефтепродуктов в резервуаре

\*/

int current\_level;

/\*\*

\* состояния работы резервуара (STORAGE\_TANK\_ON - включен, STORAGE\_TANK\_OFF - выключен)

\*/

int state;

/\*\*

\* насос для закачивания нефтепродуктов

\*/

pump\* injection\_pump;

/\*\*

\* насос для откачивания нефтепродуктов

\*/

pump\* pumping\_pump;

};

Функции:

* Создать резервуар
* Переключить резервуар в рабочее состояние
* Переключить резервуар в нерабочее состояние
* Получить состояние работы резервуара (включен или выключен)
* Установить минимально допустимый уровень нефтепродуктов
* Получить минимально допустимый уровень нефтепродуктов
* Установить максимально допустимый уровень нефтепродуктов
* Получить максимально допустимый уровень нефтепродуктов
* Получить текущий уровень нефтепродуктов
* Включить насос закачки
* Выключить насос закачки
* Установить скорость закачки
* Получить скорость закачки
* Получить состояние работы насоса закачки (включен или выключен)
* Включить насос откачки
* Выключить насос откачки
* Установить скорость откачки
* Получить скорость откачки
* Получить состояние работы насоса откачки (включен или выключен)
* Удалить резервуар

Проектирование структур и функций для сущности «хранилище нефти»:

Структура нефтехранилища включает в себя массив резервуаров и количество резервуаров.

Функции для работы с хранилищем нефти дублируют функции для работы с резервуаром, за исключением функций создания и удаления хранилища нефти.

# Проектирование параллельных вычислений

Для создания программы с параллельными можно использовать следующие средства:

* Создание дочерних процессов
* Создание потоков выполнения

Для моделирования работы резервуаров было решено использовать дочерние процессы. Данный выбор связан со следующими факторами:

1. Резервуар достаточно крупная единица обработки
2. В синхронизации работы между дочерними процессами нет необходимости
3. Синхронизация с родительским процессом происходит достаточно редко (только по запросу пользователя)

Для моделирования насосов и системы контроля будут использованы потоки исполнения. Данный выбор связан со следующими факторами:

1. Насос и система контроля достаточно небольшие единицы обработки информации
2. При работе используется небольшое количество информации (уровень нефтепродуктов, максимально и минимально допустимый уровень).
3. Простота обработки данных. Потоки позволяют избежать использования каналов. Этот фактор очень важен, так как насосы с высокой частотой изменяют показатель уровня нефтепродуктов, и каждый раз для этого использовать каналы нецелесообразно.
4. Нет необходимости синхронизировать данные. Насос закачки, насос откачки и система контроля работают независимо друг от друга, т.е. порядок их работы не важен

Таким образом приложения будет состоять из одного основного процесса и нескольких дочерних (по одному на каждый резервуар). Каждый дочерний процесс состоит из 4 потоков: основной поток, поток для работы насоса закачки, поток для работы насоса откачки и поток для контроля уровня нефти.

Для обмена информацией между дочерними и родительским процессами будут созданы два массива каналов (по два на каждый резервуар): в первом массиве – каналы для передачи информации в дочерний процесс, второй – для получения информации из дочернего процесса.

Кроме того, при таком подходе может возникнуть deadlock: и родительский, и дочерний процесс пытаются читать информацию из канала, и тем самым находятся в бесконечном ожидании информации друг от друга. Чтобы избежать данной проблемы для чтения и записи в каналы будут созданы отдельные потоки.

# Проектирование интерфейса

Требования к интерфейсу не были предъявлены, соответственно можно спроектировать любой интерфейс, позволяющий использовать все функции приложения.

Основной задачей интерфейса является передача команд, которые вводит пользователь, в систему для их исполнения. Для данной задачи наиболее простым и удобным решением является реализация консольного командного интерфейса.

Также стоит учесть следующий фактор: программа моделирует работу сразу нескольких резервуаров. Человеку будет достаточно сложно при помощи команд отслеживать состояние всех резервуаров. Хорошим решение будет добавить автоматический вывод и обновление основных характеристик системы на экран. Также для визуализации резервуаров можно использовать псевдографику.

На рисунке 5 представлен спроектированный интерфейс:

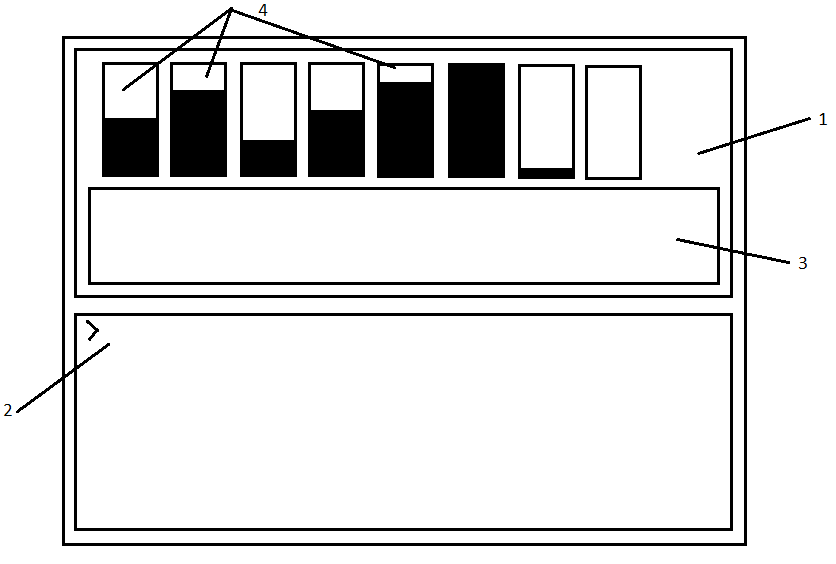


Рисунок 5 – интерфейс пользователя

Цифрами на рисунке обозначены следующие компоненты:

1. Область для вывода информации о системе
2. Область для ввода команд и вывода результаты их работы
3. Область для отображения основных характеристик резервуаров
4. Псевдографическое отображение состояния каждого из резервуаров

# Реализация программы

В соответствии с поставленными требованиями, на основе результатов проектирования была реализована программа (Рисунок 6).

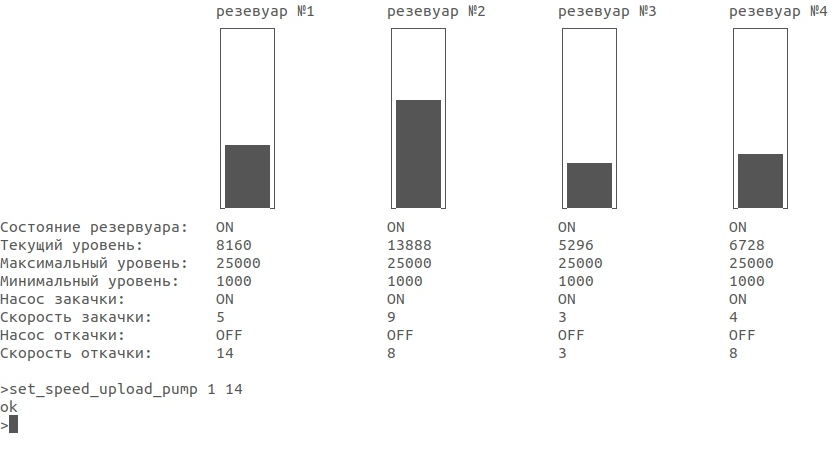


Рисунок 6 – Результат запуска программы

Код программы представлен в приложении А. Более подробное описание работы функций приложения можно посмотреть в коде. Весь код, кроме функций для работы с интерфейсом задокументирован, согласно стандарту Javadoc.

# Тестирование

Для проверки работоспособности системы было проведено функциональное тестирование. В таблице 1 представлены функциональные тесты.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Параметры теста** | **Результат** |
| Запустить программу | Количество резервуаров в хранилище нефти: 6 | Тест пройден  (Рисунок B.1). |
| Просмотр основных параметров системы | – | Тест пройден  (Рисунок B.1). Все выведенные параметры отображаются и изменяются корректно. |
| Включить резервуар | Номер резервуара: 1 | Тест пройден  (Рисунок B.2) |
| Включить насос откачки | Номер резервуара: 3 | Тест пройден  (Рисунок B.2) |
| Включить насос закачки | Номер резервуара: 6 | Тест пройден  (Рисунок B.2) |
| Выключить резервуар | Номер резервуара: 2 | Тест пройден  (Рисунок B.3) |
| Выключить насос откачки | Номер резервуара: 4 | Тест пройден  (Рисунок B.3) |
| Выключить насос закачки | Номер резервуара: 3 | Тест пройден  (Рисунок B.3) |
| Установить скорость закачки нефти | Номер резервуара: 1  Устанавливаемая скорость: 100 | Тест пройден  (Рисунок B.4). |
| Изменить максимально допустимый уровень нефтепродуктов | Номер резервуара: 5  Устанавливаемый максимальный уровень: 33300 | Тест пройден  (Рисунок B.4). |
| Изменить минимально допустимый уровень нефтепродуктов | Номер резервуара: 6  Устанавливаемый минимальный уровень: 689 | Тест пройден  (Рисунок B.4). |

# Заключение

В рамках данной курсовой работы были выполнены все поставленные задачи:

* Составлены требования к программе
* Спроектирована программа
* Реализована программа
* Протестирована программа

Во время выполнения курсового проекта были изучены способы создания и синхронизации параллельных вычислений, функции для работы с каналами в Unix-подобных системах, принципы работы в ОС семейства Linux.

Была создана программа на языке C для моделирования работы хранилища нефти.

Созданная программ позволяет выполнять следующие функции:

* Запускать моделирование нефтехранилища с различным количеством резервуаров
* Изменять параметры системы (состояние работы насосов и резервуаров, скорость закачки и откачки, максимально и минимально допустимый уровень нефтепродуктов)
* Просматривать состояние системы, как при помощи псевдографического изображения, так и при помощи ввода команд.

При создании программы были использованы методы для выполнения параллельных вычислений: работа с дочерними процессами и с потоками исполнения.

Созданная программа была протестирована, согласно поставленным требованиям. Тестирование показало корректность работы программы.

Полученное приложение удовлетворяет всем изложенным требованиям.

# Список литературы

1. Шашков Б.Д. Методические указания к лабораторным работам по дисциплине «Операционные системы и сети» / Пенза, издательство ПГУ, 2013.
2. Э. Таненбаум, А. Вудхалл. Операционные системы. Разработка и реализация / Питер, издательство «Питер-пресс», 2007 – 704 с.
3. Руководство по программированию на  
    C [Электронный ресурс]. – Режим доступа:   
   [http://www.cplusplus.com](http://www.cplusplus.com/) (Дата обращения: 13.05.2019)
4. Изучаем процессы в Linux   
   [Электронный ресурс]. – Режим доступа:   
   <https://habr.com/ru/post/423049/> (Дата обращения: 13.05.2019)
5. Знакомство с межпроцессорным взаимодействием [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/122108/> (Дата обращения: 13.05.2019)

# Приложение А – Исходный код программы

Файл main.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "oil\_storage.h"

#include "oil\_storage\_interface.h"

#define MIN\_LEVEL\_STORAGE\_DEFAULT 1000

#define MAX\_LEVEL\_STORAGE\_DEFAULT 25000

#define MAX\_SPEED 10

int main(int argc, char\* argv[]) {

srand((unsigned int)time(0));

size\_t cnt\_tanks = 5;

if (argc > 1) {

cnt\_tanks = (size\_t)strtol(argv[1], NULL, 10);

}

oil\_storage\* os = create\_oil\_storage(cnt\_tanks, MIN\_LEVEL\_STORAGE\_DEFAULT, MAX\_LEVEL\_STORAGE\_DEFAULT, 0, 0);

for (unsigned int i = 0; i < cnt\_tanks; ++i) {

turn\_on\_download\_pump(os, i);

set\_speed\_download\_pump(os, i, (unsigned int)(rand() % MAX\_SPEED + 1));

set\_speed\_upload\_pump(os, i, (unsigned int)(rand() % MAX\_SPEED + 1));

}

start\_oil\_storage\_interface(os);

finalize\_oil\_storage(os);

return 0;

}

Файл oil\_storage.c

#include "oil\_storage.h"

#include "storage\_tank.h"

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>

#ifndef \_\_OPERATION\_NUMBER

#define \_\_OPERATION\_NUMBER

#define CREATE\_STORAGE\_TANK 0

#define TURN\_ON\_STORAGE\_TANK 1

#define TURN\_OFF\_STORAGE\_TANK 2

#define GET\_STATE\_TANK 3

#define SET\_MINIMUM\_LEVEL\_TANK 4

#define GET\_MINIMUM\_LEVEL\_TANK 5

#define SET\_MAXIMUM\_LEVEL\_TANK 6

#define GET\_MAXIMUM\_LEVEL\_TANK 7

#define GET\_CURRENT\_LEVEL\_TANK 8

#define TURN\_ON\_DOWNLOAD\_PUMP 9

#define TURN\_OFF\_DOWNLOAD\_PUMP 10

#define GET\_STATE\_DOWNLOAD\_PUMP 11

#define SET\_SPEED\_DOWNLOAD\_PUMP 12

#define GET\_SPEED\_DOWNLOAD\_PUMP 13

#define TURN\_ON\_UPLOAD\_PUMP 14

#define TURN\_OFF\_UPLOAD\_PUMP 15

#define GET\_STATE\_UPLOAD\_PUMP 16

#define SET\_SPEED\_UPLOAD\_PUMP 17

#define GET\_SPEED\_UPLOAD\_PUMP 18

#define FINALIZE\_STORAGE\_TANK -1

#endif

/\*\*

\* создать процессы для каждого резевуара

\* @param os указатель на нефтрехранилище

\*/

static void \_create\_process\_for\_tanks(oil\_storage \* os);

/\*\*

\* функция управления резервуаром

\* @param fd\_in файловый дескриптор канала для чтения команд

\* @param fd\_out файловый дескриптор канала для ответа на команды

\*/

static void \_manage\_storage\_tank(int fd\_in, int fd\_out);

/\*\*

\* записывает номер команды в указанный файл

\* @param fd дескриптор файла для записи

\* @param operation\_number номер команды

\*/

static void \_send\_operation\_number(int fd, int operation\_number);

/\*\*

\* Хранилище нефти

\*/

struct \_oil\_storage {

/\*\*

\* количество резевуаров

\*/

size\_t tanks\_count;

/\*\*

\* идентификаторы процессов, в которых осуществляется управление резервуаром

\*/

pid\_t\* pids;

/\*\*

\* каналы для передачи информации резервуарам

\*/

int\*\* pipe\_fds\_in;

/\*\*

\* каналы для получения информации от резервуаров

\*/

int\*\* pipe\_fds\_out;

};

oil\_storage\* create\_oil\_storage(size\_t storage\_tanks\_count, unsigned int min\_level, unsigned int max\_level, unsigned int speed\_download\_pump, unsigned int speed\_upload\_pump) {

oil\_storage\* os = malloc(sizeof(oil\_storage));

os->tanks\_count = storage\_tanks\_count;

os->pids = malloc(sizeof(pid\_t) \* os->tanks\_count);

os->pipe\_fds\_in = malloc(sizeof(int\*) \* os->tanks\_count);

os->pipe\_fds\_out = malloc(sizeof(int\*) \* os->tanks\_count);

for (int i = 0; i < os->tanks\_count; ++i) {

os->pipe\_fds\_in[i] = malloc(sizeof(int) \* 2);

os->pipe\_fds\_out[i] = malloc(sizeof(int) \* 2);

pipe(os->pipe\_fds\_in[i]);

pipe(os->pipe\_fds\_out[i]);

}

\_create\_process\_for\_tanks(os);

for (int i = 0; i < os->tanks\_count; ++i) {

\_send\_operation\_number(os->pipe\_fds\_in[i][1], CREATE\_STORAGE\_TANK);

unsigned int params[] = {

min\_level,

max\_level,

speed\_download\_pump,

speed\_upload\_pump,

};

write(os->pipe\_fds\_in[i][1], params, sizeof(params));

}

return os;

}

void turn\_on\_tank(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_ON\_STORAGE\_TANK);

}

void turn\_off\_tank(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_OFF\_STORAGE\_TANK);

}

int get\_state\_tank(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_STATE\_TANK);

int state\_st;

read(os->pipe\_fds\_out[number][0], &state\_st, sizeof(state\_st));

return state\_st;

}

void set\_minimum\_level\_tank(oil\_storage\* os, unsigned int number, unsigned int min\_level) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], SET\_MINIMUM\_LEVEL\_TANK);

write(os->pipe\_fds\_in[number][1], &min\_level, sizeof(min\_level));

}

unsigned int get\_minimum\_level\_tank(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_MINIMUM\_LEVEL\_TANK);

unsigned int min\_level;

read(os->pipe\_fds\_out[number][0], &min\_level, sizeof(min\_level));

return min\_level;

}

void set\_maximum\_level\_tank(oil\_storage\* os, unsigned int number, unsigned int max\_level) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], SET\_MAXIMUM\_LEVEL\_TANK);

write(os->pipe\_fds\_in[number][1], &max\_level, sizeof(max\_level));

}

unsigned int get\_maximum\_level\_tank(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_MAXIMUM\_LEVEL\_TANK);

unsigned int max\_level;

read(os->pipe\_fds\_out[number][0], &max\_level, sizeof(max\_level));

return max\_level;

}

unsigned int get\_current\_level\_tank(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_CURRENT\_LEVEL\_TANK);

unsigned int cur\_level;

read(os->pipe\_fds\_out[number][0], &cur\_level, sizeof(cur\_level));

return cur\_level;

}

void finalize\_oil\_storage(oil\_storage\* os) {

for (int i = 0; i < os->tanks\_count; ++i) {

\_send\_operation\_number(os->pipe\_fds\_in[i][1], FINALIZE\_STORAGE\_TANK);

}

for (int i = 0; i < os->tanks\_count; ++i) {

waitpid(os->pids[i], NULL, 0);

close(os->pipe\_fds\_in[i][1]);

close(os->pipe\_fds\_out[i][0]);

free(os->pipe\_fds\_out[i]);

free(os->pipe\_fds\_in[i]);

}

free(os->pipe\_fds\_out);

free(os->pipe\_fds\_in);

free(os->pids);

free(os);

}

void turn\_on\_download\_pump(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_ON\_DOWNLOAD\_PUMP);

}

void turn\_off\_download\_pump(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_OFF\_DOWNLOAD\_PUMP);

}

int get\_state\_download\_pump(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_STATE\_DOWNLOAD\_PUMP);

int state\_dp;

read(os->pipe\_fds\_out[number][0], &state\_dp, sizeof(state\_dp));

return state\_dp;

}

void set\_speed\_download\_pump(oil\_storage\* os, unsigned int number, unsigned int download\_speed) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], SET\_SPEED\_DOWNLOAD\_PUMP);

write(os->pipe\_fds\_in[number][1], &download\_speed, sizeof(download\_speed));

}

unsigned int get\_speed\_download\_pump(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_SPEED\_DOWNLOAD\_PUMP);

unsigned int download\_speed;

read(os->pipe\_fds\_out[number][0], &download\_speed, sizeof(download\_speed));

return download\_speed;

}

void turn\_on\_upload\_pump(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_ON\_UPLOAD\_PUMP);

}

void turn\_off\_upload\_pump(oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], TURN\_OFF\_UPLOAD\_PUMP);

}

int get\_state\_upload\_pump(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_STATE\_UPLOAD\_PUMP);

int state\_up;

read(os->pipe\_fds\_out[number][0], &state\_up, sizeof(state\_up));

return state\_up;

}

void set\_speed\_upload\_pump(oil\_storage\* os, unsigned int number, unsigned int upload\_speed) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], SET\_SPEED\_UPLOAD\_PUMP);

write(os->pipe\_fds\_in[number][1], &upload\_speed, sizeof(upload\_speed));

}

unsigned int get\_speed\_upload\_pump(const oil\_storage\* os, unsigned int number) {

\_send\_operation\_number(os->pipe\_fds\_in[number][1], GET\_SPEED\_UPLOAD\_PUMP);

unsigned int upload\_speed;

read(os->pipe\_fds\_out[number][0], &upload\_speed, sizeof(upload\_speed));

return upload\_speed;

}

size\_t get\_count\_tanks(const oil\_storage\* os) {

return os->tanks\_count;

}

static void \_create\_process\_for\_tanks(oil\_storage\* os) {

for (int i = 0; i < os->tanks\_count; ++i) {

os->pids[i] = fork();

if (os->pids[i] == 0) {

close(os->pipe\_fds\_in[i][1]);

close(os->pipe\_fds\_out[i][0]);

\_manage\_storage\_tank(os->pipe\_fds\_in[i][0], os->pipe\_fds\_out[i][1]);

close(os->pipe\_fds\_in[i][0]);

close(os->pipe\_fds\_out[i][1]);

\_exit(0);

}

close(os->pipe\_fds\_in[i][0]);

close(os->pipe\_fds\_out[i][1]);

}

}

static void \_manage\_storage\_tank(int fd\_in, int fd\_out) {

storage\_tank\* st = NULL;

for (;;) {

int operation\_number;

read(fd\_in, &operation\_number, sizeof(operation\_number));

switch (operation\_number) {

case CREATE\_STORAGE\_TANK: {

unsigned int params[4];

read(fd\_in, params, sizeof(params));

st = create\_storage\_tank(params[0], params[1], params[2], params[3]);

break;

}

case TURN\_ON\_STORAGE\_TANK: {

turn\_on\_storage\_tank(st);

break;

}

case TURN\_OFF\_STORAGE\_TANK: {

turn\_off\_storage\_tank(st);

break;

}

case GET\_STATE\_TANK: {

int state\_st = get\_state\_storage\_tank(st);

write(fd\_out, &state\_st, sizeof(state\_st));

break;

}

case SET\_MINIMUM\_LEVEL\_TANK: {

unsigned int min\_level;

read(fd\_in, &min\_level, sizeof(min\_level));

set\_minimum\_level\_storage\_tank(st, min\_level);

break;

}

case GET\_MINIMUM\_LEVEL\_TANK: {

unsigned int min\_level = get\_minimum\_level\_storage\_tank(st);

write(fd\_out, &min\_level, sizeof(min\_level));

break;

}

case SET\_MAXIMUM\_LEVEL\_TANK: {

unsigned int max\_level;

read(fd\_in, &max\_level, sizeof(max\_level));

set\_maximum\_level\_storage\_tank(st, max\_level);

break;

}

case GET\_MAXIMUM\_LEVEL\_TANK: {

unsigned int max\_level = get\_maximum\_level\_storage\_tank(st);

write(fd\_out, &max\_level, sizeof(max\_level));

break;

}

case GET\_CURRENT\_LEVEL\_TANK: {

unsigned int cur\_level = get\_current\_level\_storage\_tank(st);

write(fd\_out, &cur\_level, sizeof(cur\_level));

break;

}

case TURN\_ON\_DOWNLOAD\_PUMP: {

turn\_on\_injection\_pump(st);

break;

}

case TURN\_OFF\_DOWNLOAD\_PUMP: {

turn\_off\_injection\_pump(st);

break;

}

case GET\_STATE\_DOWNLOAD\_PUMP: {

int state\_dp = get\_state\_injection\_pump(st);

write(fd\_out, &state\_dp, sizeof(state\_dp));

break;

}

case SET\_SPEED\_DOWNLOAD\_PUMP: {

unsigned int speed\_dp;

read(fd\_in, &speed\_dp, sizeof(speed\_dp));

set\_speed\_injection\_pump(st, speed\_dp);

break;

}

case GET\_SPEED\_DOWNLOAD\_PUMP: {

unsigned int speed\_dp = get\_speed\_injection\_pump(st);

write(fd\_out, &speed\_dp, sizeof(speed\_dp));

break;

}

case TURN\_ON\_UPLOAD\_PUMP: {

turn\_on\_pumping\_pump(st);

break;

}

case TURN\_OFF\_UPLOAD\_PUMP: {

turn\_off\_pumping\_pump(st);

break;

}

case GET\_STATE\_UPLOAD\_PUMP: {

int state\_up = get\_state\_pumping\_pump(st);

write(fd\_out, &state\_up, sizeof(state\_up));

break;

}

case SET\_SPEED\_UPLOAD\_PUMP: {

unsigned int speed\_pp;

read(fd\_in, &speed\_pp, sizeof(speed\_pp));

set\_speed\_pumping\_pump(st, speed\_pp);

break;

}

case GET\_SPEED\_UPLOAD\_PUMP: {

unsigned int speed\_pp = get\_speed\_pumping\_pump(st);

write(fd\_out, &speed\_pp, sizeof(speed\_pp));

break;

}

case FINALIZE\_STORAGE\_TANK: {

finalize\_storage\_tank(st);

return;

}

default: {

continue;

}

}

}

}

static void \_send\_operation\_number(int fd, int operation\_number) {

write(fd, &operation\_number, sizeof(operation\_number));

}

Файл oil\_storage.h

#ifndef OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_H

#define OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_H

#include "oil\_storage\_def.h"

#include <stddef.h>

/\*\*

\* Хранилище нефти

\*/

struct \_oil\_storage;

typedef struct \_oil\_storage oil\_storage;

/\*\*

\* создать нефтехранилище

\* @param storage\_tanks\_count количество резервуаров

\* @param min\_level минимальный уровень нефтепродуктов в резервуаре

\* @param max\_level максимальный уровень нефтепродуктов в резервуаре

\* @param speed\_download\_pump скорость закачки нефтепродутов в резервуар

\* @param speed\_upload\_pump скорость откачки нефтпрепродуктов из резевуара

\* @return указатель на нефтрехранилище

\*/

oil\_storage\* create\_oil\_storage(size\_t storage\_tanks\_count, unsigned int min\_level, unsigned int max\_level, unsigned int speed\_download\_pump, unsigned int speed\_upload\_pump);

/\*\*

\* переключить резервуар в рабочее состояние

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_on\_tank(oil\_storage\* os, unsigned int number);

/\*\*

\* переключить резервуар в нерабочее состояние

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_off\_tank(oil\_storage\* os, unsigned int number);

/\*\*

\* Получить состояние работы резевуара

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\* @return STORAGE\_TANK\_OFF - нерабочее состояние, STORAGE\_TANK\_ON - рабочее состояние

\*/

int get\_state\_tank(const oil\_storage\* os, unsigned int number);

/\*\*

\* установить минимальный уровень нефти в резервуаре

\* @param os указатель на нефтехранилище

\* @param number номер резервуара

\* @param min\_level минимальный уровень нефти

\*/

void set\_minimum\_level\_tank(oil\_storage\* os, unsigned int number, unsigned int min\_level);

/\*\*

\* получить минимальный уровень нефти в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return минимальный уровень нефти

\*/

unsigned int get\_minimum\_level\_tank(const oil\_storage\* os, unsigned int number);

/\*\*

\* установить максимальный уровень нефти в резервуаре

\* @param os указатель на нефтехранилище

\* @param number номер резервуара

\* @param max\_level минимальный уровень нефти

\*/

void set\_maximum\_level\_tank(oil\_storage\* os, unsigned int number, unsigned int max\_level);

/\*\*

\* получить максимальный уровень нефти в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return минимальный уровень нефти

\*/

unsigned int get\_maximum\_level\_tank(const oil\_storage\* os, unsigned int number);

/\*\*

\* получить текущий уровень нефти в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return уровень нефти

\*/

unsigned int get\_current\_level\_tank(const oil\_storage\* os, unsigned int number);

/\*\*

\* уничтожить нефтрехранилище

\* @param os укзатель на нефтрехранилище

\*/

void finalize\_oil\_storage(oil\_storage\* os);

/\*\*

\* включить насос закачки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_on\_download\_pump(oil\_storage\* os, unsigned int number);

/\*\*

\* выключить насос закачки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_off\_download\_pump(oil\_storage\* os, unsigned int number);

/\*\*

\* получить состояние работы насоса загрузки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return PUMP\_OFF - нерабочее состояние, PUMP\_ON - рабочее состояние

\*/

int get\_state\_download\_pump(const oil\_storage\* os, unsigned int number);

/\*\*

\* установить скорость закачки насоса в резевуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @param download\_speed скорость закачки

\*/

void set\_speed\_download\_pump(oil\_storage\* os, unsigned int number, unsigned int download\_speed);

/\*\*

\* получить скорость закачки насоса в резевуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return скорость закачки

\*/

unsigned int get\_speed\_download\_pump(const oil\_storage\* os, unsigned int number);

/\*\*

\* включить насос откачки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_on\_upload\_pump(oil\_storage\* os, unsigned int number);

/\*\*

\* выключить насос откачки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резервуара

\*/

void turn\_off\_upload\_pump(oil\_storage\* os, unsigned int number);

/\*\*

\* получить состояние работы насоса отгрузки в резервуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return PUMP\_OFF - нерабочее состояние, PUMP\_ON - рабочее состояние

\*/

int get\_state\_upload\_pump(const oil\_storage\* os, unsigned int number);

/\*\*

\* установить скорость откачки насоса в резевуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @param upload\_speed скорость откачки

\*/

void set\_speed\_upload\_pump(oil\_storage\* os, unsigned int number, unsigned int upload\_speed);

/\*\*

\* получить скорость откачки насоса в резевуаре

\* @param os указатель на нефтрехранилище

\* @param number номер резевуара

\* @return скорость откачки

\*/

unsigned int get\_speed\_upload\_pump(const oil\_storage\* os, unsigned int number);

/\*\*

\* получить количество резервуаров в нефтехранилище

\* @param os указатель на нефтрехранилище

\* @return количество резервуаров

\*/

size\_t get\_count\_tanks(const oil\_storage\* os);

#endif //OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_H

Файл oil\_storage\_def.h

#ifndef OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_DEF\_H

#define OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_DEF\_H

#define STORAGE\_TANK\_ON 1 //резервуар в рабочем сосотянии

#define STORAGE\_TANK\_OFF 0 //резервуар в нерабочем состоянии

#define TIME\_UNIT 10 //время в мс, за которое происходит одно изменение

#define PUMP\_ON 1 //насос включен

#define PUMP\_OFF 0 //насос выключен

#endif //OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_DEF\_H

Файл oil\_storage\_interface.c

#include "oil\_storage\_interface.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <termios.h>

static size\_t width\_tank = 7;

static size\_t height\_tank = 11;

static size\_t distance\_between\_tanks = 12;

static char\* lower\_border\_empty = NULL;

static char\* lower\_border\_full = NULL;

static char\* mid\_empty = NULL;

static char\* mid\_half = NULL;

static char\* mid\_full = NULL;

static char\* upper\_border\_empty = NULL;

static char\* upper\_border\_full = NULL;

static char\* between\_tanks = NULL;

static struct termios stored\_settings;

static void \_set\_keypress\_mode();

static void \_reset\_keypress\_mode();

static void \_generate\_pseudo\_graphics\_string();

static void \_output\_tanks\_labels(const oil\_storage\* os);

static void \_output\_tanks\_state(const oil\_storage\* os);

static void \_output\_format\_tanks\_labels(const oil\_storage\* os, char\* name, size\_t real\_size\_name, char\*\* labels);

static void \_output\_characteristics\_tanks(const oil\_storage\* os);

pthread\_t \_read\_chars\_thread;

int continue\_read\_char = 1;

int chars\_buffer[10000];

int last\_write\_char = 0;

static void \_output\_console(oil\_storage\* os);

static void\* \_read\_chars(void\* params);

static char\* \_implement\_command(oil\_storage\* os, char\* command\_line);

void start\_oil\_storage\_interface(oil\_storage\* os) {

\_set\_keypress\_mode();

\_generate\_pseudo\_graphics\_string();

printf("\033[2J");

while (continue\_read\_char) {

printf("\033[0;0H");

\_output\_tanks\_labels(os);

\_output\_tanks\_state(os);

\_output\_characteristics\_tanks(os);

printf("\033[K\n");

\_output\_console(os);

usleep(40 \* 1000);

}

pthread\_join(\_read\_chars\_thread, NULL);

\_reset\_keypress\_mode();

printf("\033[2J\033[0;0H");

fflush(stdin);

}

static void \_set\_keypress\_mode() {

struct termios new\_settings;

tcgetattr(0, &stored\_settings);

new\_settings = stored\_settings;

new\_settings.c\_lflag &= (~ICANON);

new\_settings.c\_cc[VTIME] = 0;

new\_settings.c\_cc[VMIN] = 1;

tcsetattr(0, TCSANOW, &new\_settings);

}

static void \_reset\_keypress\_mode() {

tcsetattr(0, TCSANOW, &stored\_settings);

}

static void \_generate\_pseudo\_graphics\_string() {

static char\* left\_lower\_corner = "└";

static char\* right\_lower\_corner = "┘";

static char\* lower\_empty = "─";

static char\* lower\_full = "▀";

static char\* left\_border = "│";

static char\* right\_border = "│";

static char\* empty = " ";

static char\* half = "▄";

static char\* full = "█";

static char\* left\_upper\_corner = "┌";

static char\* right\_upper\_corner = "┐";

static char\* upper\_empty = "─";

static char\* upper\_full = "▄";

static char\* between\_tanks\_char = " ";

if (lower\_border\_empty != NULL) free(lower\_border\_empty);

if (lower\_border\_full != NULL) free(lower\_border\_full);

if (mid\_empty != NULL) free(mid\_empty);

if (mid\_half != NULL) free(mid\_half);

if (mid\_full != NULL) free(mid\_full);

if (upper\_border\_empty != NULL) free(upper\_border\_empty);

if (upper\_border\_full != NULL) free(upper\_border\_full);

if (between\_tanks != NULL) free(between\_tanks);

lower\_border\_empty = malloc(width\_tank \* 4);

lower\_border\_full = malloc(width\_tank \* 4);

mid\_empty = malloc(width\_tank \* 4);

mid\_half = malloc(width\_tank \* 4);

mid\_full = malloc(width\_tank \* 4);

upper\_border\_empty = malloc(width\_tank \* 4);

upper\_border\_full = malloc(width\_tank \* 4);

between\_tanks = malloc(width\_tank \* 4);

sprintf(lower\_border\_empty, "%s", left\_lower\_corner);

sprintf(lower\_border\_full, "%s", left\_lower\_corner);

sprintf(mid\_empty, "%s", left\_border);

sprintf(mid\_half, "%s", left\_border);

sprintf(mid\_full, "%s", left\_border);

sprintf(upper\_border\_empty, "%s", left\_upper\_corner);

sprintf(upper\_border\_full, "%s", left\_upper\_corner);

for (int i = 1; i + 1 < width\_tank; ++i) {

strcat(lower\_border\_empty, lower\_empty);

strcat(lower\_border\_full, lower\_full);

strcat(mid\_empty, empty);

strcat(mid\_half, half);

strcat(mid\_full, full);

strcat(upper\_border\_empty, upper\_empty);

strcat(upper\_border\_full, upper\_full);

}

strcat(lower\_border\_empty, right\_lower\_corner);

strcat(lower\_border\_full, right\_lower\_corner);

strcat(mid\_empty, right\_border);

strcat(mid\_half, right\_border);

strcat(mid\_full, right\_border);

strcat(upper\_border\_empty, right\_upper\_corner);

strcat(upper\_border\_full, right\_upper\_corner);

between\_tanks[0] = '\0';

for (int i = 0; i < distance\_between\_tanks; ++i) {

strcat(between\_tanks, between\_tanks\_char);

}

}

static void \_output\_tanks\_labels(const oil\_storage\* os) {

static char\* tanks\_label = "резевуар №";

static const size\_t length\_tanks\_label = 10;

size\_t count\_tanks = get\_count\_tanks(os);

printf("%s%s", between\_tanks, between\_tanks);

for (int i = 0; i < count\_tanks; ++i) {

char\* label\_with\_space = malloc(strlen(tanks\_label) + 3 + distance\_between\_tanks + width\_tank);

int num = i + 1;

sprintf(label\_with\_space, "%s%d", tanks\_label, num);

size\_t cur\_sz = length\_tanks\_label + 1;

while (num /= 10) cur\_sz++;

size\_t out\_str\_len = strlen(label\_with\_space);

while (cur\_sz < width\_tank + distance\_between\_tanks) {

label\_with\_space[out\_str\_len++] = ' ';

cur\_sz++;

}

label\_with\_space[out\_str\_len] = '\0';

printf("%s", label\_with\_space);

free(label\_with\_space);

}

printf("\033[K\n");

}

static void \_output\_tanks\_state(const oil\_storage\* os) {

size\_t count\_tanks = get\_count\_tanks(os);

unsigned int\* cur\_levels = malloc(sizeof(unsigned int) \* count\_tanks);

unsigned int\* max\_levels = malloc(sizeof(unsigned int) \* count\_tanks);

for (int i = 0; i < count\_tanks; ++i) {

cur\_levels[i] = get\_current\_level\_tank(os, i);

max\_levels[i] = get\_maximum\_level\_tank(os, i);

}

size\_t count\_segments = height\_tank \* 2 - 2;

for (int i = 0; i < count\_tanks; ++i) {

if (i == 0) printf("%s%s", between\_tanks, between\_tanks);

else printf("%s", between\_tanks);

if (count\_segments \* cur\_levels[i] >= (count\_segments - 1) \* max\_levels[i]) {

printf("%s", upper\_border\_full);

}

else {

printf("%s", upper\_border\_empty);

}

}

printf("\033[K\n");

for (size\_t j = height\_tank - 1; j > 1; --j) {

int segment\_value = (int)(2 \* j - 2);

for (int i = 0; i < count\_tanks; ++i) {

if (i == 0) printf("%s%s", between\_tanks, between\_tanks);

else printf("%s", between\_tanks);

if (count\_segments \* cur\_levels[i] >= segment\_value \* max\_levels[i]) {

printf("%s", mid\_full);

}

else if (count\_segments \* cur\_levels[i] >= (segment\_value - 1) \* max\_levels[i]) {

printf("%s", mid\_half);

}

else {

printf("%s", mid\_empty);

}

}

printf("\033[K\n");

}

for (int i = 0; i < count\_tanks; ++i) {

if (i == 0) printf("%s%s", between\_tanks, between\_tanks);

else printf("%s", between\_tanks);

if (count\_segments \* cur\_levels[i] >= max\_levels[i]) {

printf("%s", lower\_border\_full);

}

else {

printf("%s", lower\_border\_empty);

}

}

printf("\033[K\n");

free(cur\_levels);

free(max\_levels);

}

static void \_output\_format\_tanks\_labels(const oil\_storage\* os, char\* name, size\_t real\_size\_name, char\*\* labels) {

char\* label\_with\_space = malloc(strlen(name) + distance\_between\_tanks \* 2);

sprintf(label\_with\_space, "%s", name);

size\_t label\_with\_space\_len = strlen(label\_with\_space);

while (real\_size\_name++ < 2 \* distance\_between\_tanks) {

label\_with\_space[label\_with\_space\_len++] = ' ';

}

label\_with\_space[label\_with\_space\_len] = '\0';

printf("%s", label\_with\_space);

free(label\_with\_space);

size\_t count\_tanks = get\_count\_tanks(os);

for (int i = 0; i < count\_tanks; ++i) {

char\* chr\_label = malloc(strlen(labels[i]) + distance\_between\_tanks + width\_tank);

sprintf(chr\_label, "%s", labels[i]);

size\_t chr\_label\_len = strlen(chr\_label);

while (chr\_label\_len < distance\_between\_tanks + width\_tank) {

chr\_label[chr\_label\_len++] = ' ';

}

chr\_label[chr\_label\_len] = '\0';

printf("%s", chr\_label);

free(chr\_label);

}

printf("\033[K\n");

}

static void \_output\_characteristics\_tanks(const oil\_storage\* os) {

size\_t count\_tanks = get\_count\_tanks(os);

int\* tanks\_on = malloc(sizeof(int) \* count\_tanks);

unsigned int\* cur\_levels = malloc(sizeof(unsigned int) \* count\_tanks);

unsigned int\* max\_levels = malloc(sizeof(unsigned int) \* count\_tanks);

unsigned int\* min\_levels = malloc(sizeof(unsigned int) \* count\_tanks);

int\* download\_on = malloc(sizeof(int) \* count\_tanks);

unsigned int\* download\_speed = malloc(sizeof(unsigned int) \* count\_tanks);

int\* upload\_on = malloc(sizeof(int) \* count\_tanks);

unsigned int\* upload\_speed = malloc(sizeof(unsigned int) \* count\_tanks);

for (int i = 0; i < count\_tanks; ++i) {

tanks\_on[i] = get\_state\_tank(os, i);

cur\_levels[i] = get\_current\_level\_tank(os, i);

max\_levels[i] = get\_maximum\_level\_tank(os, i);

min\_levels[i] = get\_minimum\_level\_tank(os, i);

download\_on[i] = get\_state\_download\_pump(os, i);

download\_speed[i] = get\_speed\_download\_pump(os, i);

upload\_on[i] = get\_state\_upload\_pump(os, i);

upload\_speed[i] = get\_speed\_upload\_pump(os, i);

}

char\*\* labels = malloc(sizeof(char\*) \* count\_tanks);

for (int i = 0; i < count\_tanks; ++i) labels[i] = malloc(sizeof(char) \* 20);

for (int i = 0; i < count\_tanks; ++i) {

if (tanks\_on[i] == STORAGE\_TANK\_ON) sprintf(labels[i], "ON");

if (tanks\_on[i] == STORAGE\_TANK\_OFF) sprintf(labels[i], "OFF");

}

\_output\_format\_tanks\_labels(os, "Состояние резервуара:", 21, labels);

for (int i = 0; i < count\_tanks; ++i) sprintf(labels[i], "%u", cur\_levels[i]);

\_output\_format\_tanks\_labels(os, "Текущий уровень:", 16, labels);

for (int i = 0; i < count\_tanks; ++i) sprintf(labels[i], "%u", max\_levels[i]);

\_output\_format\_tanks\_labels(os, "Максимальный уровень:", 21, labels);

for (int i = 0; i < count\_tanks; ++i) sprintf(labels[i], "%u", min\_levels[i]);

\_output\_format\_tanks\_labels(os, "Минимальный уровень:", 20, labels);

for (int i = 0; i < count\_tanks; ++i) {

if (download\_on[i] == PUMP\_ON) sprintf(labels[i], "ON");

if (download\_on[i] == PUMP\_OFF) sprintf(labels[i], "OFF");

}

\_output\_format\_tanks\_labels(os, "Насос закачки:", 14, labels);

for (int i = 0; i < count\_tanks; ++i) sprintf(labels[i], "%u", download\_speed[i]);

\_output\_format\_tanks\_labels(os, "Скорость закачки:", 17, labels);

for (int i = 0; i < count\_tanks; ++i) {

if (upload\_on[i] == PUMP\_ON) sprintf(labels[i], "ON");

if (upload\_on[i] == PUMP\_OFF) sprintf(labels[i], "OFF");

}

\_output\_format\_tanks\_labels(os, "Насос откачки:", 14, labels);

for (int i = 0; i < count\_tanks; ++i) sprintf(labels[i], "%u", upload\_speed[i]);

\_output\_format\_tanks\_labels(os, "Скорость откачки:", 17, labels);

for (int i = 0; i < count\_tanks; ++i) free(labels[i]);

free(labels);

free(upload\_speed);

free(upload\_on);

free(download\_speed);

free(download\_on);

free(min\_levels);

free(max\_levels);

free(cur\_levels);

free(tanks\_on);

}

static void \_output\_console(oil\_storage\* os) {

const size\_t console\_log\_max\_size = 200;

static const size\_t console\_string\_max\_len = 500;

static char\* console\_log[200];

static size\_t console\_log\_size = 0;

static int is\_initial = 0;

static int current\_ptr = 0;

if (!is\_initial) {

for (int i = 0; i < console\_log\_max\_size; ++i) {

console\_log[i] = malloc(sizeof(char) \* console\_string\_max\_len);

memset(console\_log[i], 0, sizeof(char) \* console\_string\_max\_len);

}

console\_log[0][0] = '>';

is\_initial = 1;

pthread\_create(&\_read\_chars\_thread, NULL, \_read\_chars, NULL);

}

int start\_i = 0;

if (console\_log\_size > 20) start\_i = console\_log\_size - 20;

for (; start\_i < console\_log\_size; ++start\_i) printf("%s\033[K\n", console\_log[start\_i]);

printf("\033[s\n\033[K\n\033[K\n\033[K\033[u");

while (current\_ptr < last\_write\_char) {

char c = chars\_buffer[current\_ptr++];

if (c == '\n') {

char\* ans = \_implement\_command(os, console\_log[console\_log\_size] + 1);

printf("%s\033[K\n", console\_log[console\_log\_size]);

console\_log\_size++;

strcpy(console\_log[console\_log\_size], ans);

printf("%s\033[K\n", console\_log[console\_log\_size]);

console\_log\_size++;

console\_log[console\_log\_size][0] = '>';

}

else {

size\_t cl\_len = strlen(console\_log[console\_log\_size]);

if (c == 127) {

if (cl\_len > 1) console\_log[console\_log\_size][cl\_len - 1] = '\0';

}

else {

console\_log[console\_log\_size][cl\_len] = c;

}

}

}

printf("%s\033[K", console\_log[console\_log\_size]);

fflush(stdout);

}

static void\* \_read\_chars(void\* params) {

while (continue\_read\_char) {

chars\_buffer[last\_write\_char] = getchar();

last\_write\_char++;

}

return NULL;

}

static char\* \_implement\_command(oil\_storage\* os, char\* command\_line) {

char command[100];

sscanf(command\_line, "%s", command);

if (strcmp(command, "exit") == 0) {

continue\_read\_char = 0;

return "ok (press any key)";

}

unsigned int number = strtol(command\_line + strlen(command) + 1, &command\_line, 10) - 1;

if (strcmp(command, "turn\_on\_tank") == 0) {

turn\_on\_tank(os, number);

return "ok";

}

if (strcmp(command, "turn\_off\_tank") == 0) {

turn\_off\_tank(os, number);

return "ok";

}

if (strcmp(command, "set\_minimum\_level\_tank") == 0) {

unsigned int min\_level = strtol(command\_line + 1, NULL, 10);

set\_minimum\_level\_tank(os, number, min\_level);

return "ok";

}

if (strcmp(command, "set\_maximum\_level\_tank") == 0) {

unsigned int max\_level = strtol(command\_line + 1, NULL, 10);

set\_maximum\_level\_tank(os, number, max\_level);

return "ok";

}

if (strcmp(command, "turn\_on\_download\_pump") == 0) {

turn\_on\_download\_pump(os, number);

return "ok";

}

if (strcmp(command, "turn\_off\_download\_pump") == 0) {

turn\_off\_download\_pump(os, number);

return "ok";

}

if (strcmp(command, "set\_speed\_download\_pump") == 0) {

unsigned int speed = strtol(command\_line + 1, NULL, 10);

set\_speed\_download\_pump(os, number, speed);

return "ok";

}

if (strcmp(command, "turn\_on\_upload\_pump") == 0) {

turn\_on\_upload\_pump(os, number);

return "ok";

}

if (strcmp(command, "turn\_off\_upload\_pump") == 0) {

turn\_off\_upload\_pump(os, number);

return "ok";

}

if (strcmp(command, "set\_speed\_upload\_pump") == 0) {

unsigned int speed = strtol(command\_line + 1, NULL, 10);

set\_speed\_upload\_pump(os, number, speed);

return "ok";

}

if (strcmp(command, "get\_state\_tank") == 0) {

int state = get\_state\_tank(os, number);

if (state == STORAGE\_TANK\_ON) return "ON";

return "OFF";

}

if (strcmp(command, "get\_minimum\_level\_tank") == 0) {

unsigned int min\_level = get\_minimum\_level\_tank(os, number);

char\* min\_level\_str = malloc(sizeof(char) \* 9);

sprintf(min\_level\_str, "%u", min\_level);

return min\_level\_str;

}

if (strcmp(command, "get\_maximum\_level\_tank") == 0) {

unsigned int max\_level = get\_maximum\_level\_tank(os, number);

char\* max\_level\_str = malloc(sizeof(char) \* 9);

sprintf(max\_level\_str, "%u", max\_level);

return max\_level\_str;

}

if (strcmp(command, "get\_current\_level\_tank") == 0) {

unsigned int cur\_level = get\_current\_level\_tank(os, number);

char\* cur\_level\_str = malloc(sizeof(char) \* 9);

sprintf(cur\_level\_str, "%u", cur\_level);

return cur\_level\_str;

}

if (strcmp(command, "get\_state\_download\_pump") == 0) {

int state = get\_state\_download\_pump(os, number);

if (state == PUMP\_ON) return "ON";

return "OFF";

}

if (strcmp(command, "get\_speed\_download\_pump") == 0) {

unsigned int speed = get\_speed\_download\_pump(os, number);

char\* speed\_str = malloc(sizeof(char) \* 9);

sprintf(speed\_str, "%u", speed);

return speed\_str;

}

if (strcmp(command, "get\_state\_upload\_pump") == 0) {

int state = get\_state\_upload\_pump(os, number);

if (state == PUMP\_ON) return "ON";

return "OFF";

}

if (strcmp(command, "get\_speed\_upload\_pump") == 0) {

unsigned int speed = get\_speed\_upload\_pump(os, number);

char\* speed\_str = malloc(sizeof(char) \* 9);

sprintf(speed\_str, "%u", speed);

return speed\_str;

}

return "Unknown command";

}

Файл oil\_storage\_interface.h

#ifndef OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_INTERFACE\_H

#define OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_INTERFACE\_H

#include "oil\_storage.h"

/\*\*

\* Запустить интерфейс

\* @param os нефтрехранилище для которого реализуется интерфейс

\*/

void start\_oil\_storage\_interface(oil\_storage\* os);

#endif //OIL\_STORAGE\_MANAGE\_SYSTEM\_OIL\_STORAGE\_INTERFACE\_H

Файл pump.c

#include "pump.h"

#include "oil\_storage\_def.h"

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

/\*\*

\* Функция, в которой осуществляется работа насоса

\* @param p\_ptr указатель на насос

\* @return NULL;

\*/

static void\* \_pump\_work(void\* p\_ptr);

/\*\*

\* насос для перекачки нефтeпродуктов

\*/

struct \_pump {

/\*\*

\* текущее значение нефти (величина, которая будет изменяться)

\*/

int\* value;

/\*\*

\* скорость перекачки (величина, на которую будет изменять уровень нефти)

\*/

int delta;

/\*\*

\* состояние работы насоса(PUMP\_ON - включен, PUMP\_OFF выключен)

\*/

int state;

/\*\*

\* поток, в котором осуществляется работа насоса

\*/

pthread\_t work\_thread;

};

pump\* create\_pump(int\* value, int delta\_per\_unit\_time) {

pump\* p = malloc(sizeof(pump));

if (p != NULL) {

p->value = value;

p->delta = delta\_per\_unit\_time;

p->state = PUMP\_OFF;

}

return p;

}

void turn\_on\_pump(pump\* p) {

if (p->state == PUMP\_OFF) {

p->state = PUMP\_ON;

pthread\_create(&p->work\_thread, NULL, \_pump\_work, p);

}

}

void turn\_off\_pump(pump\* p) {

if (p->state == PUMP\_ON) {

p->state = PUMP\_OFF;

pthread\_join(p->work\_thread, NULL);

}

}

int get\_state\_pump(const pump\* p) {

return p->state;

}

void set\_delta\_pump(pump\* p, int delta\_per\_unit\_time) {

p->delta = delta\_per\_unit\_time;

}

int get\_delta\_pump(const pump\* p) {

return p->delta;

}

void finalize\_pump(pump\* p) {

turn\_off\_pump(p);

free(p);

}

void\* \_pump\_work(void\* p\_ptr) {

pump\* p = p\_ptr;

while (p->state == PUMP\_ON) {

\*p->value += p->delta;

usleep(TIME\_UNIT \* 1000);

}

return NULL;

}

Файл pump.h

#ifndef OIL\_STORAGE\_MANAGE\_SYSTEM\_PUMP\_H

#define OIL\_STORAGE\_MANAGE\_SYSTEM\_PUMP\_H

/\*\*

\* насос для перекачки нефтeпродуктов

\*/

struct \_pump;

typedef struct \_pump pump;

/\*\*

\* создать насос

\* @param value текущее значение нефти (величина, которая будет изменяться)

\* @param delta\_per\_unit\_time скорость перекачки (величина, на которую будет изменять уровень нефти)

\* @return указатель на насос

\*/

pump\* create\_pump(int\* value, int delta\_per\_unit\_time);

/\*\*

\* включить насос

\* @param p указатель на насос

\*/

void turn\_on\_pump(pump\* p);

/\*\*

\* выключить насос

\* @param p указатель на насос

\*/

void turn\_off\_pump(pump\* p);

/\*\*

\* получить состояние работы насоса

\* @param p указатель на насос

\* @return PUMP\_OFF - насос выключен, PUMP\_ON - насос включен

\*/

int get\_state\_pump(const pump\* p);

/\*\*

\* Установить скорость перекачки нефти

\* @param p указатель на насос

\* @param deltainjection\_pump\_per\_unit\_time скорость перекачки нефти

\*/

void set\_delta\_pump(pump\* p, int delta\_per\_unit\_time);

/\*\*

\* Получить скорость перекачки нефти

\* @param p указатель на насос

\* @return скорость перекачки нефти

\*/

int get\_delta\_pump(const pump\* p);

/\*\*

\* Уничтожить насос

\* @param p указатель на насос

\*/

void finalize\_pump(pump\* p);

#endif //OIL\_STORAGE\_MANAGE\_SYSTEM\_PUMP\_H

Файл storage\_tank.c

#include "storage\_tank.h"

#include "oil\_storage\_def.h"

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

/\*\*

\* Функция котроля уровня нефтпродуктов в резервуаре

\* @param st\_ptr указатель на резервуар

\* @return NULL

\*/

static void\* \_control\_level(void\* st\_ptr);

/\*\*

\* резервуар для хранения нефтепродуктов

\*/

struct \_storage\_tank {

/\*\*

\* минимальный уровень нефтепродуктов в резервуаре

\*/

unsigned int minimum\_level;

/\*\*

\* максимальный уровень нефтепродуктов в резервуаре

\*/

unsigned int maximum\_level;

/\*\*

\* уровень нефтепродуктов в резервуаре

\*/

int current\_level;

/\*\*

\* состояния работы резервуара (STORAGE\_TANK\_ON - влючен, STORAGE\_TANK\_OFF - выключен)

\*/

int state;

/\*\*

\* насос для закачивания нефтепродуктов

\*/

pump\* injection\_pump;

/\*\*

\* насос для откачивания нефтепродуктов

\*/

pump\* pumping\_pump;

/\*\*

\* поток, для контроля уровня нефтепродутов в резервуаре

\*/

pthread\_t control\_thread;

};

storage\_tank\* create\_storage\_tank(unsigned int min\_level, unsigned int max\_level, unsigned int speed\_injection\_pump, unsigned int speed\_pumping\_pump) {

storage\_tank\* st = malloc(sizeof(storage\_tank));

st->minimum\_level = min\_level;

st->maximum\_level = max\_level;

st->current\_level = min\_level;

st->state = STORAGE\_TANK\_OFF;

st->injection\_pump = create\_pump(&st->current\_level, speed\_injection\_pump);

st->pumping\_pump = create\_pump(&st->current\_level, -speed\_pumping\_pump);

return st;

}

void turn\_on\_storage\_tank(storage\_tank\* st) {

if (st->state == STORAGE\_TANK\_OFF) {

st->state = STORAGE\_TANK\_ON;

pthread\_create(&st->control\_thread, NULL, \_control\_level, st);

}

}

void turn\_off\_storage\_tank(storage\_tank\* st) {

if (st->state == STORAGE\_TANK\_ON) {

st->state = STORAGE\_TANK\_OFF;

turn\_off\_pump(st->injection\_pump);

turn\_off\_pump(st->pumping\_pump);

pthread\_join(st->control\_thread, NULL);

}

}

int get\_state\_storage\_tank(const storage\_tank\* st) {

return st->state;

}

void set\_minimum\_level\_storage\_tank(storage\_tank\* st, unsigned int min\_level) {

st->minimum\_level = min\_level;

}

unsigned int get\_minimum\_level\_storage\_tank(const storage\_tank\* st) {

return st->minimum\_level;

}

void set\_maximum\_level\_storage\_tank(storage\_tank\* st, unsigned int max\_level) {

st->maximum\_level = max\_level;

}

unsigned int get\_maximum\_level\_storage\_tank(const storage\_tank\* st) {

return st->maximum\_level;

}

unsigned int get\_current\_level\_storage\_tank(storage\_tank\* st) {

if (st->current\_level < 0) {

st->current\_level = 0;

}

return st->current\_level;

}

void finalize\_storage\_tank(storage\_tank\* st) {

turn\_off\_storage\_tank(st);

finalize\_pump(st->injection\_pump);

finalize\_pump(st->pumping\_pump);

free(st);

}

void turn\_on\_injection\_pump(storage\_tank\* st) {

if (st->state == STORAGE\_TANK\_OFF) {

turn\_on\_storage\_tank(st);

}

if (st->current\_level < st->maximum\_level) {

turn\_on\_pump(st->injection\_pump);

}

}

void turn\_off\_injection\_pump(storage\_tank\* st) {

turn\_off\_pump(st->injection\_pump);

}

int get\_state\_injection\_pump(const storage\_tank\* st) {

return get\_state\_pump(st->injection\_pump);

}

void set\_speed\_injection\_pump(storage\_tank\* st, unsigned int speed) {

set\_delta\_pump(st->injection\_pump, speed);

}

unsigned int get\_speed\_injection\_pump(const storage\_tank\* st) {

get\_delta\_pump(st->injection\_pump);

}

void turn\_on\_pumping\_pump(storage\_tank\* st) {

if (st->state == STORAGE\_TANK\_OFF) {

turn\_on\_storage\_tank(st);

}

if (st->current\_level > st->minimum\_level) {

turn\_on\_pump(st->pumping\_pump);

}

}

void turn\_off\_pumping\_pump(storage\_tank\* st) {

turn\_off\_pump(st->pumping\_pump);

}

int get\_state\_pumping\_pump(const storage\_tank\* st) {

get\_state\_pump(st->pumping\_pump);

}

void set\_speed\_pumping\_pump(storage\_tank\* st, unsigned int speed) {

set\_delta\_pump(st->pumping\_pump, -speed);

}

unsigned int get\_speed\_pumping\_pump(const storage\_tank\* st) {

return -get\_delta\_pump(st->pumping\_pump);

}

static void\* \_control\_level(void\* st\_ptr) {

storage\_tank\* st = st\_ptr;

while (st->state == STORAGE\_TANK\_ON) {

if (st->current\_level <= st->minimum\_level) {

turn\_off\_pump(st->pumping\_pump);

}

if (st->current\_level < 0) {

st->current\_level = 0;

}

if (st->current\_level >= st->maximum\_level) {

turn\_off\_pump(st->injection\_pump);

}

usleep(TIME\_UNIT);

}

return NULL;

}

Файл storage\_tank.h

#ifndef OIL\_STORAGE\_MANAGE\_SYSTEM\_STORAGE\_TANK\_H

#define OIL\_STORAGE\_MANAGE\_SYSTEM\_STORAGE\_TANK\_H

#include "pump.h"

/\*\*

\* резервуар для хранения нефтепродуктов

\*/

struct \_storage\_tank;

typedef struct \_storage\_tank storage\_tank;

/\*\*

\* Создать резервуар

\* @param min\_level минимально допустиммый уровень нефти

\* @param max\_level максимально допустимый уровень нефти

\* @param speed\_injection\_pump скорость закачки нефти

\* @param speed\_pumping\_pump скорость откачки нефти

\* @return указатель на созданный резервуар

\*/

storage\_tank\* create\_storage\_tank(unsigned int min\_level, unsigned int max\_level, unsigned int speed\_injection\_pump, unsigned int speed\_pumping\_pump);

/\*\*

\* переключить резервуар в рабочее состояние

\* @param st указатель на резервуар

\*/

void turn\_on\_storage\_tank(storage\_tank\* st);

/\*\*

\* переключить резервуар в нерабочее состояние

\* @param st указатель на резервуар

\*/

void turn\_off\_storage\_tank(storage\_tank\* st);

/\*\*

\* получить состояние работы резервуара

\* @param st указатель на резервуар

\* @return STORAGE\_TANK\_OFF - нерабочее состояние, STORAGE\_TANK\_ON - рабочее состояние

\*/

int get\_state\_storage\_tank(const storage\_tank\* st);

/\*\*

\* установить мимнимальный уровень нефти

\* @param st указатель на резервуар

\* @param min\_level минимальный уровень нефти

\*/

void set\_minimum\_level\_storage\_tank(storage\_tank\* st, unsigned int min\_level);

/\*\*

\* получить минимальный уровень нефти

\* @param st указатель на резервуар

\* @return минимальный уровень нефти

\*/

unsigned int get\_minimum\_level\_storage\_tank(const storage\_tank\* st);

/\*\*

\* установить максимальный уровень нефти

\* @param st указатель на резервуар

\* @param min\_level максимальный уровень нефти

\*/

void set\_maximum\_level\_storage\_tank(storage\_tank\* st, unsigned int max\_level);

/\*\*

\* получить максимальный уровень нефти

\* @param st указатель на резервуар

\* @return максимальный уровень нефти

\*/

unsigned int get\_maximum\_level\_storage\_tank(const storage\_tank\* st);

/\*\*

\* получить уровень нефти

\* @param st указатель на резервуар

\* @return уровень нефти

\*/

unsigned int get\_current\_level\_storage\_tank(storage\_tank\* st);

/\*\*

\* уничтожить резевуар

\* @param st указатель на резервуар

\*/

void finalize\_storage\_tank(storage\_tank\* st);

/\*\*

\* включить насос закачки нефтепродуктов

\* @param st указатель на резервуар

\*/

void turn\_on\_injection\_pump(storage\_tank\* st);

/\*\*

\* выключить насос закачки нефтепродуктов

\* @param st указатель на резервуар

\*/

void turn\_off\_injection\_pump(storage\_tank\* st);

/\*\*

\* получить состояние работы насоса закачки

\* @param st указатель на резервуар

\* @return PUMP\_OFF - нерабочее состояние, PUMP\_ON - рабочее состояние

\*/

int get\_state\_injection\_pump(const storage\_tank\* st);

/\*\*

\* установить скорость закачки нефтепродуктов

\* @param st указатель на резервуар

\* @param speed скорость закачки

\*/

void set\_speed\_injection\_pump(storage\_tank\* st, unsigned int speed);

/\*\*

\* получить скорость закачки нефтепродуктов

\* @param st указатель на резервуар

\* @return скорость закачки нефтепродуктов

\*/

unsigned int get\_speed\_injection\_pump(const storage\_tank\* st);

/\*\*

\* включить насос откачки нефтепродуктов

\* @param st указатель на резервуар

\*/

void turn\_on\_pumping\_pump(storage\_tank\* st);

/\*\*

\* выключить насос откачки нефтепродуктов

\* @param st указатель на резервуар

\*/

void turn\_off\_pumping\_pump(storage\_tank\* st);

/\*\*

\* получить состояние работы насоса откачки

\* @param st указатель на резервуар

\* @return PUMP\_OFF - нерабочее состояние, PUMP\_ON - рабочее состояние

\*/

int get\_state\_pumping\_pump(const storage\_tank\* st);

/\*\*

\* установить скорость откачки нефтепродуктов

\* @param st указатель на резервуар

\* @param speed скорость откачки

\*/

void set\_speed\_pumping\_pump(storage\_tank\* st, unsigned int speed);

/\*\*

\* получить скорость откачки нефтепродуктов

\* @param st указатель на резервуар

\* @return скорость откачки нефтепродуктов

\*/

unsigned int get\_speed\_pumping\_pump(const storage\_tank\* st);

#endif //OIL\_STORAGE\_MANAGE\_SYSTEM\_STORAGE\_TANK\_H

# Приложение B – Результаты тестирования

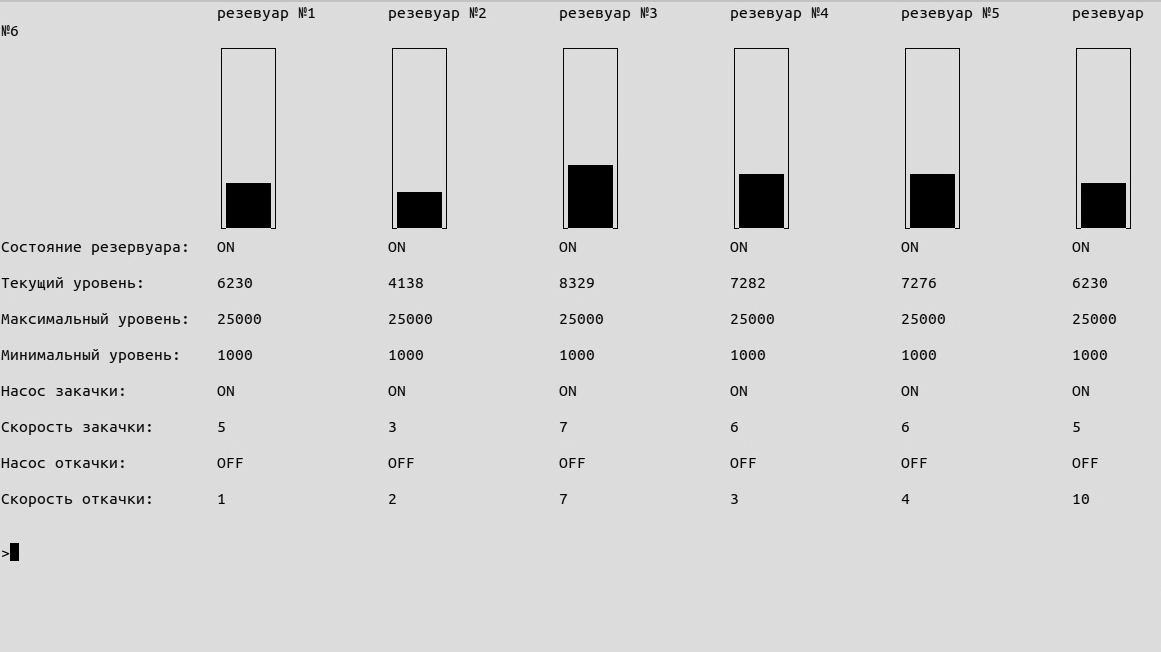


Рисунок B.1 – Запуск моделирования хранилища нефти с 6 резервуарами

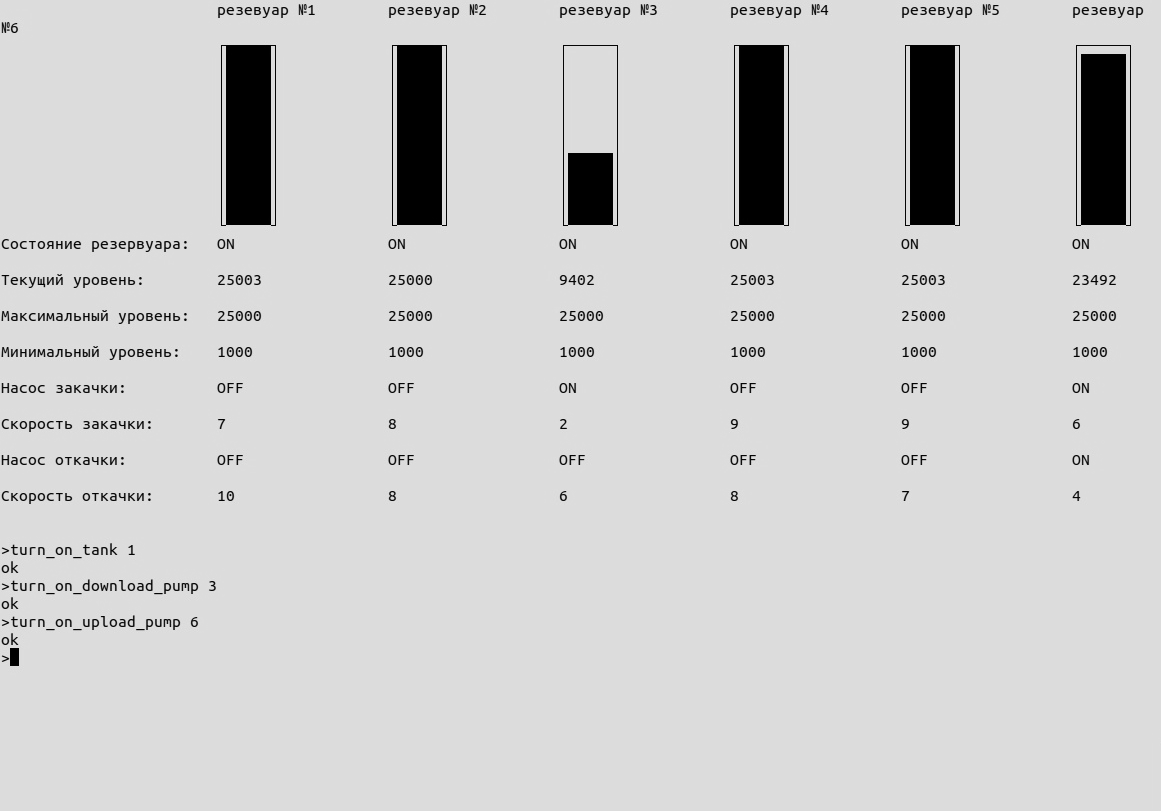


Рисунок B.2 – Выполнение команд включения резервуаров и насосов

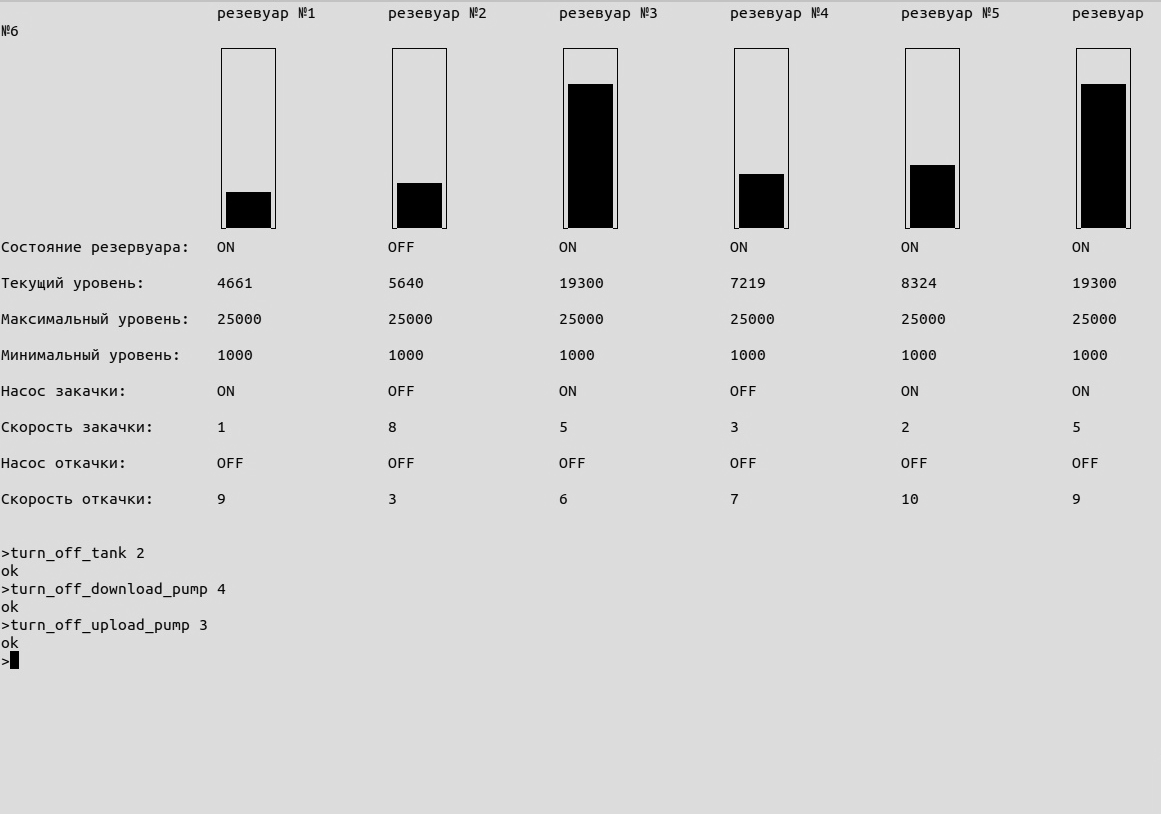


Рисунок B.3 – Выполнение команд выключения резервуаров и насосов

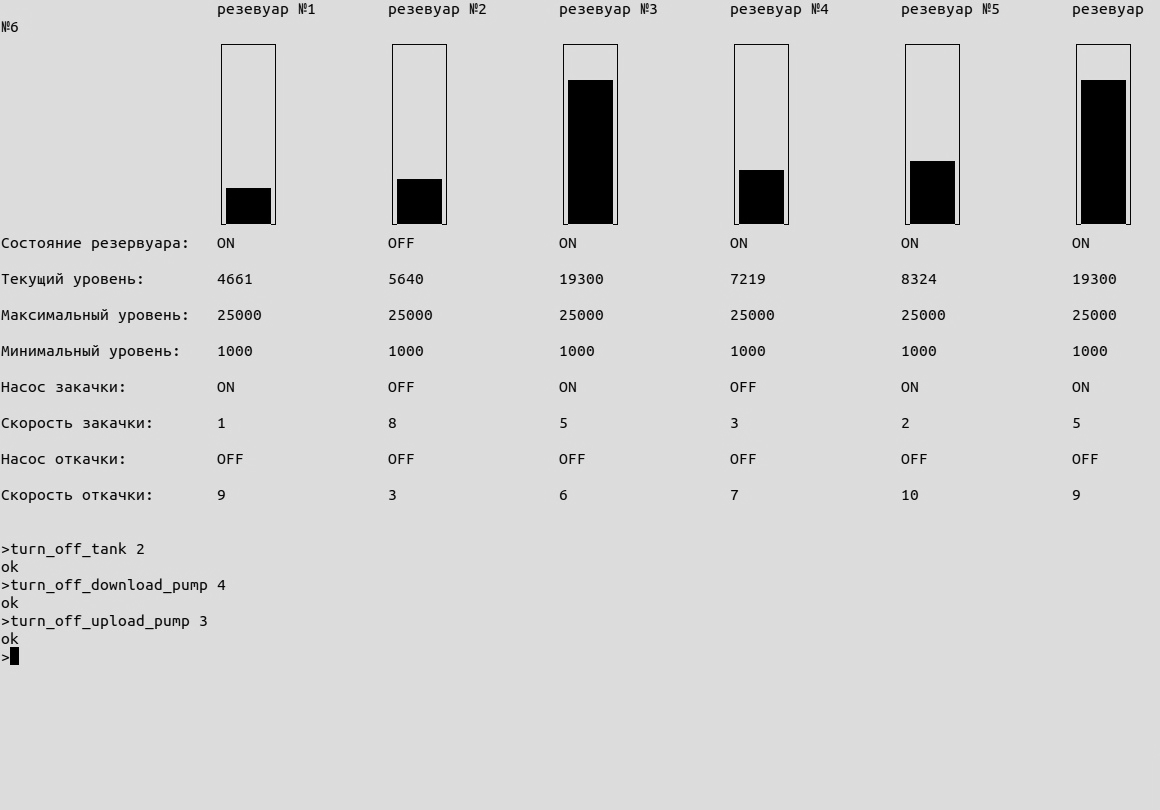


Рисунок B.4 – Выполнение команд для изменения параметров системы