

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ
И ИНФОРМАТИКИ

Кафедра технологий программирования

ЛАБОРАТОРНАЯ РАБОТА 5
ПО ДИСЦИПЛИНЕ «СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»

Проектирование и реализация статической и динамической библиотеки для
файловых операций и разработка файловой системы

Методические указания по выполнению лабораторной работы

Минск 2023

СОДЕРЖАНИЕ

Введение	3
1 Методические рекомендации	4
1.1 Основной учебный материал	4
1.2 Сборка программ с помощью системы сборки GNU Autotools ..	4
1.2.1 Утилита make и Makefile	5
1.2.2 Разработка файловой системы	7
1.2.2.1 Основы libsufe	7
1.2.2.2 Использование FUSE	8
2 Методические указания и задания	10
2.1 Методические указания	10
2.1.1 Критерии оценивания	10
2.1.2 Отчет по лабораторной работе	11
2.1.2.1 Требования к репозиторию и файлу README репозито-	
рия	12
2.1.2.2 Защита отчета по лабораторной работе	13
2.2 Задания	14
2.2.1 Задание 1	14
2.2.2 Задание 2	15
2.2.3 Задание 3	17
2.3 Варианты заданий	18
2.4 Контрольные вопросы	24

ВВЕДЕНИЕ

Целью работы является получение навыков разработки программ на языке C для управления файлами и применения Linux API. Для достижения поставленной цели необходимо решить следующие задачи:

- получить навыки разработки приложений, реализующих операции с файлами средствами системных вызовов Linux API на языке C в операционных системах семейства Linux;
- получить навыки разработки и использования статических и динамических библиотек в программах на языке C в операционных системах семейства Linux;
- изучить теоретическую часть лабораторной работы, включая материалы по использованию системы автоматической сборки Autotools и разработке файлов Makefile;
- изучить материалы по разработке файловой системы с применением библиотеки fuse и реализовать собственную файловую систему.

1 Методические рекомендации

В данном разделе представлены рекомендации по выполнению лабораторной работы.

1.1 Основной учебный материал

Учебный материал для выполнения лабораторной работы изложен в источниках:

а) Гунько А.В. Системное программирование в среде Linux: учебное пособие / А.В. Гунько. – Новосибирск: Изд-во НГТУ, 2020. – 235:

1) глава 3, стр. 25;

2) глава 4, стр. 45.

б) Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014. — 448 с.:ил. — (Серия «Бестселлеры O'Reilly»):

1) главы 2-4.

1.2 Сборка программ с помощью системы сборки GNU Autotools

Autotools, или *система сборки GNU*— это набор программных средств, предназначенных для поддержки переносимости исходного кода программ между UNIX-подобными системами.

Перенос кода с одной системы на другую может оказаться непростой задачей. Различные реализации компилятора языка Си могут существенно различаться: некоторые функции языка могут отсутствовать, иметь другое имя или находиться в разных библиотеках. Программист может решить эту задачу, используя макросы и директивы препроцессора, например `#if`, `#ifdef` и прочие.

Но в таком случае пользователь, компилирующий программу на своей системе, должен будет определить все эти макросы, что не так просто, поскольку существует множество разных дистрибутивов и вариаций систем. *Autotools* вызываются последовательностью команд `./configure && make && make install` и решают эти проблемы автоматически.





Система сборки *GNU Autotools* является частью *GNU toolchain* и широко используется во многих проектах с открытым исходным кодом. Средства

сборки распространяются в соответствии с GNU General Public License с возможностью использования их в коммерческих проектах.

- Autoconf;
- Automake;
- Libtool;
- Gnulib;
- Другие средства:
 - make;
 - gettext;
 - pkg-config;
 - gcc;
 - binutils;

На рисунке 1.1 представлена схема работы `autoconf` и `automake`.

Дополнительный учебный материал по применению Autotools, первые 5 из которых изучить перед выполнением задания 2 и 3:

- а)  <https://earthly.dev/blog/autoconf/>.
- б)  https://src_prepare.gitlab.io/devmanual-mirror/general-concepts/autotools/index.html.
- в)  <https://mj-7.medium.com/how-to-package-your-software-in-linux-using-gnu-autotools>.
- г)  <https://www.programmersonsought.com/article/10226793563/>.
- д)  <https://youtu.be/3X00d9Qyc34?si=DrQqrhio270pIoeg>.
- е)  <http://www.h-wrt.com/ru/mini-how-to/autotoolsSimpleProject>.
- ж)  <https://eax.me/autotools/>
- и)  https://help.ubuntu.ru/wiki/using_gnu_autotools
- к)  <https://opensource.com/article/19/7/introduction-gnu-autotools>
- л)  <https://www.gnu.org/software/automake/faq/autotools-faq.html>

1.2.1 Утилита `make` и `Makefile`

Утилита `make` автоматически определяет, какие части большой программы должны быть перекомпилированы, и выполняет необходимые для этого действия. Наиболее часто `make` используется для компиляции С-программ. Отметим, что `make` можно использовать с любым языком программирования. Более того, применение утилиты `make` не ограничивается программами.

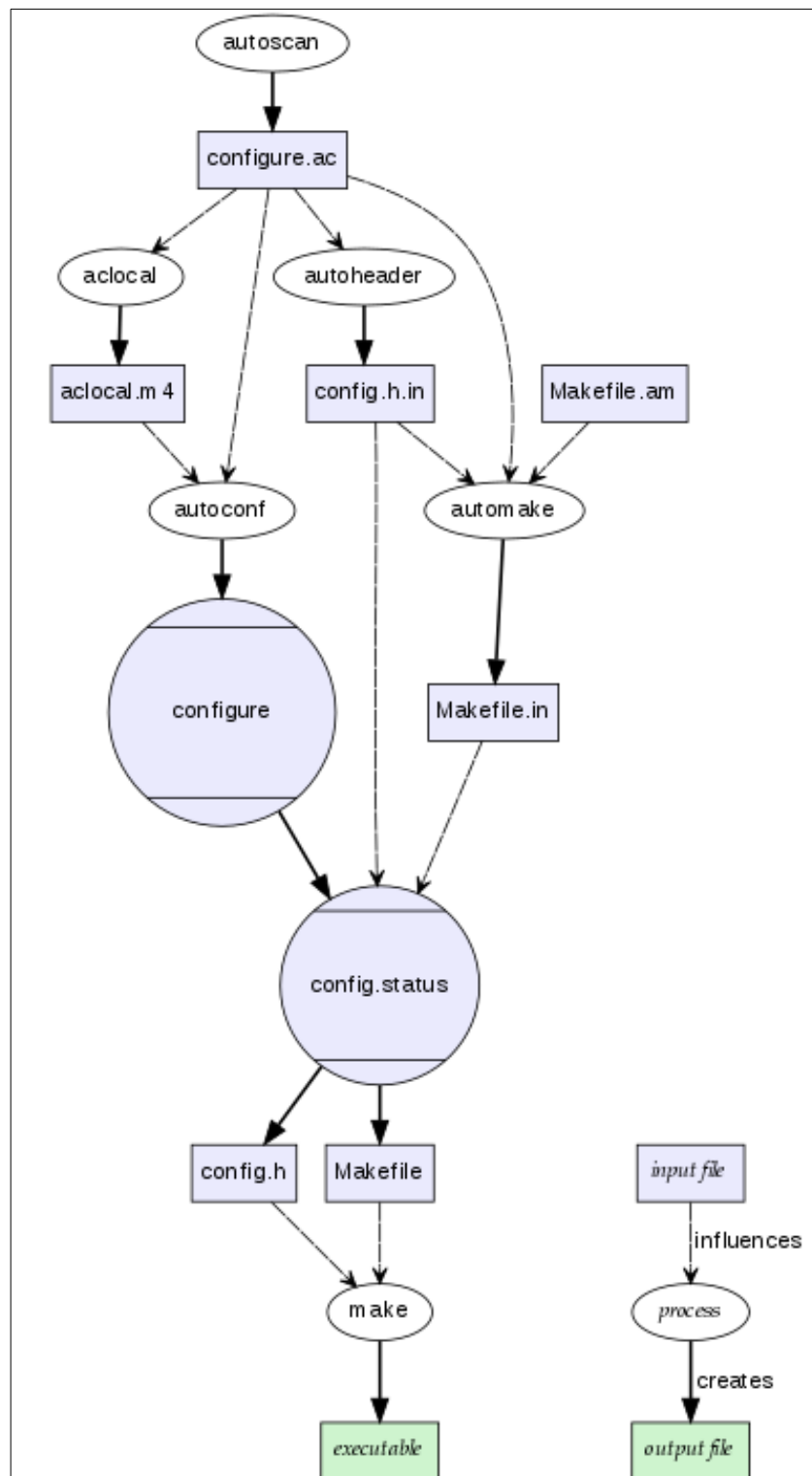





Рисунок 1.1 — Алгоритм сборки пакета с помощью Autotools

Можно использовать её для описания любой задачи, где некоторые файлы должны автоматически порождаться из других всегда, когда те изменяются.

Ознакомьтесь с рекомендациями по разработке файлов Makefile и применению утилиты make:

- а)  <https://parallel.uran.ru/book/export/html/16>.
- б)  http://linux.yaroslav1.ru/docs/prog/gnu_make_3-79_russian_manual.html
- в)  https://www.gnu.org/software/make/manual/html_node/index.html

1.2.2 Разработка файловой системы

Файловая система в пользовательском пространстве (FUSE) — это программный интерфейс для Unix и Unix-подобных компьютерных операционных систем, который позволяет непривилегированным пользователям создавать свои файловые системы без редактирования кода ядра. Это достигается за счет запуска кода файловой системы в пользовательском пространстве, в то время как модуль FUSE предоставляет лишь возможности подключения к реальным интерфейсам ядра.

Модуль FUSE доступен для  Linux, FreeBSD, OpenBSD, NetBSD, OpenSolaris, Minix 3,  macOS и  Windows.

1.2.2.1 Основы libfuse

Для реализации новой файловой системы программа-обработчик должна использовать библиотеку `libfuse`. Эта программа-обработчик должна реализовывать необходимые методы.

Как только файловая система монтируется, обработчик регистрируется в ядре. Когда пользователь вызывает операцию в этой файловой системе, ядро передает эти запросы обработчику.

Модуль FUSE особенно полезен для написания виртуальных файловых систем. В отличие от традиционных файловых систем, которые по существу работают с данными на запоминающих устройствах, виртуальные файловые системы фактически не хранят данные сами по себе. Они действуют как просмотр или подключение существующей файловой системы или устройства хранения.

В принципе, любой ресурс, доступный реализации с помощью FUSE, можно представить как файловую систему.

1.2.2.2 Использование FUSE

FUSE позволяет выполнять монтирование без прав `root` и используется в следующих задачах:

- автоматическое монтирование флешек и внешних дисков при подключении к Linux;
- монтирование файловой системы ОС Android при подключении к ОС Linux;
- монтирование сетевой файловой системы **SSHFS**, которая обеспечивает доступ к файлам на внешнем (удалённом) компьютере по протоколу **SSH**.

Для монтирования (подключения) и размонтирования (отключения) файловых систем **FUSE** применяется программа **fusermount** или **fusermount3**. Для данной программы установлен специальный бит **setuid**, который обеспечивает выполнение программы с правами владельца файла.

Реализации файловой системы **FUSE** накладываются ограничения:

- Пользователь может подключаться только к точке монтирования, для которой у него есть разрешение на запись.
- Точка монтирования не является каталогом, который не принадлежит пользователю.
- Никакой другой пользователь не может получить доступ к содержимому смонтированной файловой системы.


Для монтирования файловой системы **FUSE**, реализация которой представлена исполняемым файлом **hello** в каталоге **example** (см. рис. 1.2), необходимо создать каталог **/tmp/fuse** и выполнить команду:

```
1 $ mkdir /tmp/fuse
2 $ example/hello -d -s -f /tmp/fuse
```

Размонтирование файловой системы выполняется командой:

```
1 $ sudo fusermount -u /tmp/fuse
```

Подробнее о файловой системе **FUSE**:

-  <https://www.maastaar.net/fuse/linux/filesystem/c/2016/05/21/writing-a-simple-file-system-using-fuse/>

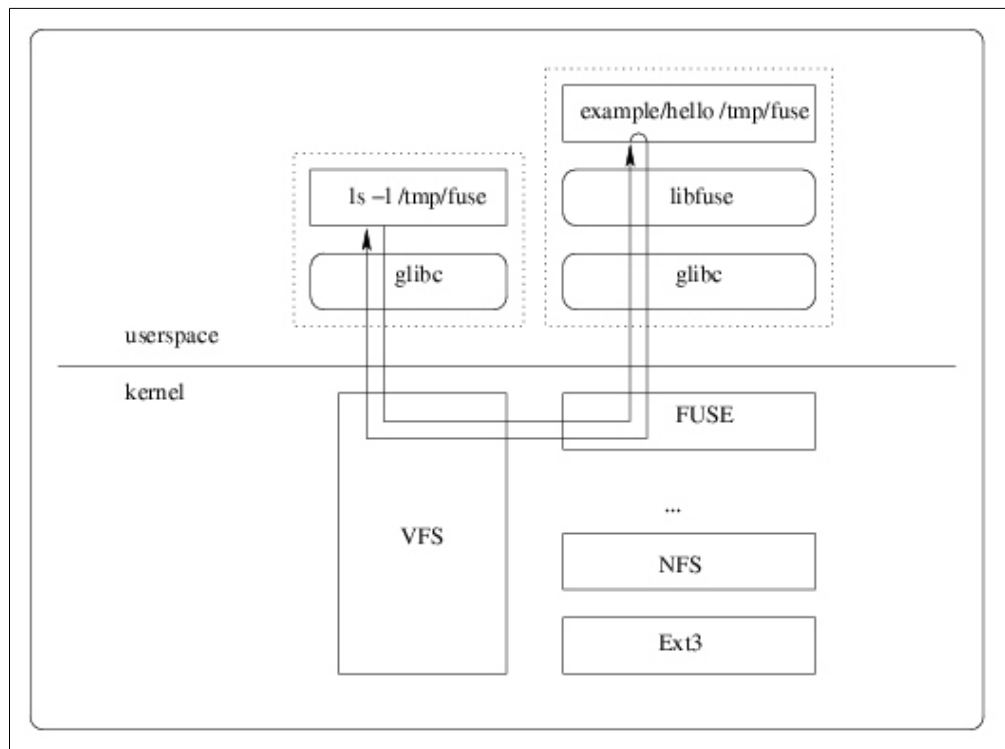


Рисунок 1.2 — Схема работы FUSE

- б) <https://engineering.facile.it/blog/eng/write-filesystem-fuse/>
- в) [Writing a FUSE Filesystem: a Tutorial](#)
- г) <https://georgesims21.github.io/posts/fuse/>
- д) <https://youtu.be/sLR17lUjTpc?feature=shared>
- е) <https://github.com/libfuse/libfuse/tree/master>
- ж) <https://wiki.archlinux.org/title/FUSE>
- и) https://en.wikipedia.org/wiki/Filesystem_in_Userspace#Applications

2 Методические указания и задания

2.1 Методические указания

Общие рекомендации по выполнению заданий лабораторной работы:

а) Проект для любого из заданий может быть реализован в виде консольного приложения в среде ОС Ubuntu, CentOS или других средствами компилятора gcc версии не ниже 4.

б) Проект для заданий 1-2 должен быть реализован с применением системных вызовов для открытия, чтения, записи и закрытия файла. Ввод исходных данных производится исключительно через аргументы командной строки. Вывод сообщений об ошибках и результатах работы программы может производиться средствами стандартной библиотеки языка C.

в) Проект для каждого из заданий должен предусматривать обработку исключительных ситуаций (отсутствие или неверное количество входных параметров, ошибки открытия входного и/или выходного файла, ошибки чтения и записи).

г) Проект для заданий 1-2 может содержать системные вызовы для блокировки файлов (входного – на чтение, выходного – на запись). Наличие функций блокировки файлов соответствует заданию повышенной сложности.

д) Проект для задания 2 является модификацией задания 1 основная бизнес-логика которого (обработка содержимого входного файла) реализована в виде функции, помещенной в статическую/динамическую библиотеку.

е) Проект для задания 3 представляет собою файловую систему FUSE, реализованную на языке C с использованием библиотеки `libfuse`.

2.1.1 Критерии оценивания

Оценка 4-5

Выполнено задания 1-2. Структура программы в задании 2 соответствует *КИС*, содержит `Makefile` для сборки, созданный вручную. В `Makefile` продемонстрировать использование переменных, автоматических переменных, шаблонных имен и, при необходимости, поиск пререквизитов по каталогам, абстрактные цели. Представлен отчет, исходный код проекта в

git-репозитории. Отчет, файлы протоколов команд и меток времени без ошибок. Лабораторная работа сдана с задержкой в 1-2 недели.

Оценка 6-7

Выполнено задания 1-3, задание 3 без реализации дополнительной функции. Структура программы соответствует *КИС* и для сборки применяется система сборки **Autotools** с автоматической генерацией **Makefile**, а так же должен быть создан пакет проекта с помощью команды **make distcheck**.

Проект по каждому из заданий опубликовать в отдельной ветке и в отдельном каталоге. Для заданий 1 и 2 обязательны тесты. Выполнить сборку проектов по заданиям 1-2 с помощью *Github Actions*.

Представлен отчет, ответы на контрольные вопросы, исходные коды скриптов в git-репозитории. Отчет, исходный код может содержать незначительные ошибки. Лабораторная работа сдана с задержкой в 1 неделю.

Оценка 8-9

Выполнены задания 1-3 на отличном уровне. Структура программы соответствует *КИС* и для сборки применяется система сборки **Autotools** с автоматической генерацией **Makefile**, а так же должен быть создан пакет проекта с помощью команды **make distcheck**.

Проект по каждому из заданий опубликовать в отдельной ветке и в отдельном каталоге. Для заданий 1 и 2 обязательны тесты. Выполнить сборку проектов по заданиям 1-3 с помощью *Github Actions*.


Представлен отчет, ответы на контрольные вопросы, исходные коды скриптов в git-репозитории. Отчет, исходный код не содержат ошибок. Лабораторная работа сдана в срок.

2.1.2 Отчет по лабораторной работе

Отчет по лабораторной работе должен быть опубликован в репозитории и отвечать требованиям:

1. Отчет по лабораторной работе состоит из письменного отчета и кода программ, опубликованных в репозиторий
2. Письменный отчет содержит цель работы.
3. Письменный отчет включает вариант задания.
4. Письменный отчет добавить описание ключевых моментов реализации и тестов.

5. Исходный код программ для каждого задания опубликовать в подкаталоге `/src` соответствующего каталога проекта (задания) и в соответствующей ветке репозитория.

Ссылка на репозиторий для лабораторной работы 5 доступна в курсе  «Системное программирование».




В файле `README` в корневом каталоге проекта на *github* должна быть ссылка на отчёт и краткие сведения о выполненных заданиях (проектах).

Отчет опубликовать во внешнем хранилище или в репозитории в каталоге `/docs`. **! Отчёт должен результаты тестов по каждой программе и ответы на контрольные вопросы.**

2.1.2.1 Требования к репозиторию и файлу `README` репозитория

Корневой каталог репозитория должен включать:

1. файл `README`, содержащий ссылку на отчет;
2. при публикации отчета в репозитории, разместить его в папке `/docs`;
3. каждое задание находится в каталоге проекта, структура которого соответствует требованиям задания;

4. в корневом каталоге репозитория и папках проектов должен быть добавлен файл `.gitignore`, в котором определены правила для исключения временных, исполняемых, объектных файлов и т.д. В качестве примера для проекта на языке C рекомендуется использовать  шаблон `.gitignore` из библиотеки шаблонов  A collection of `.gitignore` templates на  *Github*.

Пример оформления файла `README` может быть таким:

```
1 # Overview
2
3 Report on LabRabota1.
4
5 # Usage
6
7 // Заменить <<link>> и <<folder>> на соответствующие ссылки и названия /
8 To check, please, preview report by <<link>> and script files in <<folder>>.
9
10 # Author
11
12 Your name and group number.
13
14 # Additional Notes
```

15	
16	// СКОПИРОВАТЬ И ВСТАВИТЬ ССЫЛКУ НА СВОЙ РЕПОЗИТОРИЙ НА ПРИМЕР
17	https://github.com/mariyad/lab5-task1-gr16-david

2.1.2.2 Защита отчета по лабораторной работе

Каждая лабораторная работа содержит тексты задач и контрольные вопросы, ответы на которые проверяются преподавателем при приёме работы у студента.

Выполнение студентом лабораторной работы и сдача её результатов преподавателю происходит следующим образом:

1. Студент выполняет разработку программ.

В ходе разработки студент обязан следовать указаниям к данной задаче (в случае их наличия). Исходные тексты программ следует разрабатывать в соответствии с требованиями к оформлению для программ на языке C.

2. Студент выполняет самостоятельную проверку исходного текста каждой разработанной программы и правильности её работы, а также свои знания по теме лабораторной работы.

Исходные тексты программ должны соответствовать требованиям к оформлению, приведённым в приложении. Недопустимо отсутствие в тексте программы следующих важных элементов оформления: спецификации программного файла и подпрограмм, а также отступов, показывающих структурную вложенность языковых конструкций.

Для проверки правильности работы программы студенту необходимо разработать набор тестов и на их основе провести тестирование программы. Тестовый набор должен включать примеры входных и выходных данных, приведённые в тексте задачи, а также тесты, разработанные студентом самостоятельно.

Самостоятельная проверка знаний по теме лабораторной работы выполняется с помощью контрольных вопросов и заданий, приведённых в конце текста лабораторной работы.

3. Студент защищает разработанные программы. Защита заключается в том, что студент должен ответить на вопросы преподавателя, касающиеся разработанной программы, и контрольные вопросы.

К защите необходимо представить исходные тексты программ, оформленных в соответствии с требованиями, и протоколы тестирования каждой программы, подтверждающие правильность ее работы.

Протокол тестирования включает в себя тест (описание входных данных и соответствующих им выходных данных), описание выходных данных, полученных при запуске программы на данном тесте, и отметку о прохождении теста. Тест считается пройденным, если действительные результаты работы программы совпали с ожидаемыми.

Пример оформления протокола тестирования программы на определение количества вхождений слов в строке представлен в таблице 2.1.

Программы, не прошедшие тестирование, к защите не принимаются. В случае неверной работы программы хотя бы на одном тесте студент обязан выполнить отладку программы для поиска и устранения ошибки.



Таблица 2.1 — Пример протокола тестирования задачи на определение количества вхождений слова в строке

№ п/п	Входные данные	Ожидаемые выходные данные	Действительные выходные данные	Тест пройден
1	foo bar foo bar	bar: 2 foo: 2	bar: 2 foo: 2	Да
2	test record created	test: 1 record: 1 created: 1	test: 1 record: 1 created: 1	Да
3	1 2 3 4 5 1 2 3 4 5 6 7 8 9	1: 2 2: 2 3: 2 4: 2 5: 2	1: 2 2: 2 3: 2 4: 2 5: 2	Да
4	Hello World	Hello: 1 World: 1	Hello: 1 newline World: 1	Да

2.2 Задания

2.2.1 Задание 1

Структура проекта по модели КИС не требуется. Выполнить следующие этапы:

1. Изучить главу 3, с. 25  <https://disk.yandex.ru/i/1h2bFAEfYwZ5nw> и главу 2, с. 54  <https://disk.yandex.ru/i/NSY1w0nBr-IJFg>.

2. Изучить примеры кода  <http://gun.cs.nstu.ru/ssw/files/>.

3. Написать и отладить программу согласно варианту, получающую в аргументах командной строки имя существующего текстового файла, имя выходного файла, который будет переписан при его наличии и в который будет помещен результат работы программы и символ, слово или число, используемый (ое) для обработки файла.


4. Результатом работы программы является выходной текстовый файл, содержащий текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или `-1` в случае ошибки.

5. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.


6. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.

7. Представить тесты, детализацию реализации в README репозитория.

2.2.2 Задание 2

Вариант соответствует варианту задания 1. Необходимо изменить проект из задания 1 и реализовать согласно модели *КИС*. Для отладки использовать отладчик `gdb`  <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Требования к проекту программы:

— Структура программы должна соответствовать модели *КИС*  <https://www.opennet.ru/docs/RUS/zlp/002.html>.



— Сборка выполняться с помощью утилиты `make`. При написании `Makefile` продемонстрировать использование шаблонов, переменных, префиксов **для оценки 4-5**.

— Для сборки использовать систему сборки `Autotools` с автоматической генерацией `Makefile` **для оценки 6-9**.

— Сформировать пакет с открытыми исходными кодами в формате `.tgz` (`.tar.gz`) **для оценки 6-9**.

— Продемонстрировать автоматическую сборку с *Github Actions* и результаты выполнения приложения (для оценки 6-9).


Порядок выполнения задания:

1. Проект рекомендуется реализовать в 3 этапа:
 - Вынесение операций обработки содержимого входного файла в функцию, описанную в этом же файле программы.
 - Вынесение функции в отдельный файл, формирование статической библиотеки, применение статической библиотеки.
 - Формирование разделяемой библиотеки и модификация головной программы для динамической загрузки и выгрузки библиотеки.
2. Изучить главу 4, с. 45  <https://disk.yandex.ru/i/1h2bFAEfYwZ5nw>.
3. Изучить примеры кода:  <http://gun.cs.nstu.ru/ssw/libs/>.
4. Модифицировать программу из задания № 1, вынося операции обработки входного файла в функцию.
5. Сформировать статическую библиотеку, подключить к головной программе, продемонстрировать работоспособность программы преподавателю.
6. Сформировать разделяемую библиотеку, модифицировать головную программу для динамической загрузки разделяемой библиотеки, продемонстрировать работоспособность программы преподавателю.
7. Результатом работы каждой версии программы является выходной текстовый файл, содержащий текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или -1 в случае ошибки.
8. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.
9. Результатом выполнения лабораторной работы считается демонстрация работы двух вариантов программы (со статической и динамической библиотеками) и обработки исключительных ситуаций преподавателю. Каждая версия программы может быть представлена в отдельной ветке репозитория и собрана в виде пакета согласно требованиям выше.

2.2.3 Задание 3

Написать на языке C программу с помощью библиотеки FUSE, которая подключает виртуальную файловую систему, дерево каталогов которой полученное с помощью команды `tree` задано согласно варианту.

Требования к проекту программы:

- Структура программы должна соответствовать модели КИС  <https://www.opennet.ru/docs/RUS/zlp/002.html>.

- Сборка выполняться с помощью утилиты `make`. При написании `Makefile` продемонстрировать использование шаблонов, переменных, пре-реквизитов **для оценки 4-5**.

- Для сборки использовать систему сборки `Autotools` с автоматической генерацией `Makefile` **для оценки 6-9**.

- Сформировать пакет с открытыми исходными кодами в формате `.tgz` (`.tar.gz`) **для оценки 6-9**.

- Продемонстрировать автоматическую сборку с *Github Actions* и результаты выполнения приложения (для оценки 6-9).

Порядок выполнения:

1. Изучите примеры реализации файловой системы FUSE на языке C, представленные в методических рекомендациях в подпункте «1.2.2.2. Использование FUSE». **! В статьях приведены примеры, код из которых разобрать и использовать при реализации собственной файловой системы.**

2. Реализовать файловую систему, которая содержит 4 каталога: `foo`, `bar`, `baz` и `bin`,— а также 4 файла, из которых 3 — текстовые файлы: `example`, `readme.txt`, `test`,— и 1 бинарный. Содержимое бинарного файла должно быть взято из соответствующей стандартной системной утилиты (скопировать бинарный файл в файл проекта), название которой соответствует названию файла: `ls`, `grep`, `pwd`,... в зависимости от задания.

3. Содержимое остальных файлов:

`readme.txt`: *Student <имя и фамилия>, <номер зачетки>*

`test.txt`: *<Любой текст на ваш выбор с количеством строк равным последним двум цифрам номера зачетки>*

`example`: *Hello world! Student <имя и фамилия>, group <номер группы>, task <вариант>*

4. Файловая система должна монтироваться в папку `/mnt/fuse/`, после чего должна быть возможность осуществить листинг ее каталогов и просмотр содержимого файлов. При обращении к файловой системе должны проверяться права доступа (маска прав указана в дереве директорий через прямую косую черту (слэш) после имени файла). Владелец всех файлов должен быть текущий пользователь, который выполняет монтирование системы.

2.3 Варианты заданий

Вариант 1

1) *Для заданий 1-2:* Удалить из текста заданный символ. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `mkdir` (см. рис. 2.1).

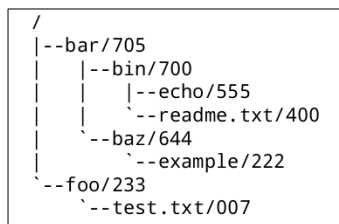


Рисунок 2.1 — Вариант 1 к заданию 3

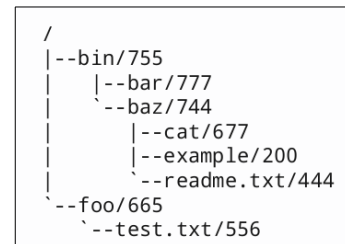


Рисунок 2.2 — Вариант 2 к заданию 3

Вариант 2

1) *Для заданий 1-2:* В конце каждой строки вставить заданный символ. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `rmdir` (см. рис. 2.2).

Вариант 3

1) *Для заданий 1-2:* Заменить цифры на пробелы. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `symlink` (см. рис. 2.3).

Вариант 4

1) *Для заданий 1-2:* Заменить знаки на заданный символ. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.



Рисунок 2.3 — Вариант 3 к заданию 3

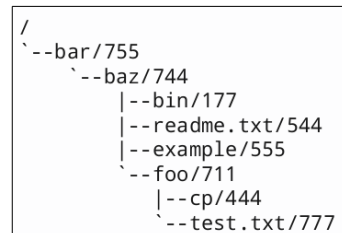


Рисунок 2.4 — Вариант 4 к заданию 3

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **rename** (см. рис. 2.4).

Вариант 5

1) *Для заданий 1-2:* Заменить каждый пробел на два. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **write** (см. рис. 2.5).

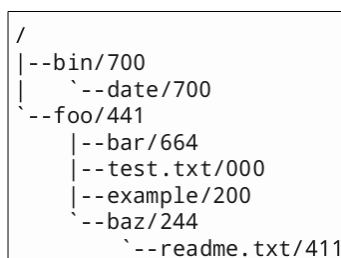


Рисунок 2.5 — Вариант 5 к заданию 3

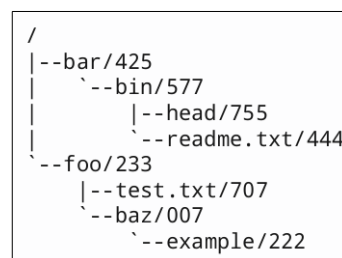


Рисунок 2.6 — Вариант 6 к заданию 3

Вариант 6

1) *Для заданий 1-2:* После каждой точки вставить символ '\n'. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **rmdir** (см. рис. 2.6).

Вариант 7

1) *Для заданий 1-2:* Удалить из текста все пробелы. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **chown** (см. рис. 2.7).

Вариант 8

1) *Для заданий 1-2:* Заменить заданные символы на пробелы. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.

```

/
|--bin/022
|--bar/555
|--baz/744
|   |--readme.txt/644
|   |--example/777
|--foo/771
|   |--pwd/777
|   |--test.txt/000

```

Рисунок 2.7 — Вариант 7 к заданию 3

```

/
|--bin/700
|   |--which/700
|--foo/441
|   |--bar/664
|   |--test.txt/000
|   |--example/200
|--baz/244
|   |--readme.txt/411

```

Рисунок 2.8 — Вариант 8 к заданию 3

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **rename** (см. рис. 2.8).

Вариант 9

1) *Для заданий 1-2:* После каждого пробела вставить точку. Параметры командной строки: 1. Имя входного файла; 2. Количество вставок точки.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **chown** (см. рис. 2.9).

```

/
|--bin/022
|--bar/555
|--baz/744
|   |--readme.txt/644
|   |--example/777
|--foo/771
|   |--pwd/777
|   |--test.txt/000

```

Рисунок 2.9 — Вариант 9 к заданию 3

```

/
|--bar/755
|   |--bin/700
|   |   |--paste/555
|   |   |--readme.txt/400
|--foo/333
|   |--test.txt/707
|--baz/644
|   |--example/211

```

Рисунок 2.10 — Вариант 10 к заданию 3

Вариант 10

1) *Для заданий 1-2:* Заменить все пробелы первым символом текста. Параметры командной строки: 1. Имя входного файла; 2. Максимальное количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **chmod** (см. рис. 2.10).

Вариант 11

1) *Для заданий 1-2:* Во всех парах одинаковых символов второй символ заменить на пробел. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **mkdir** (см. рис. 2.11).

Вариант 12

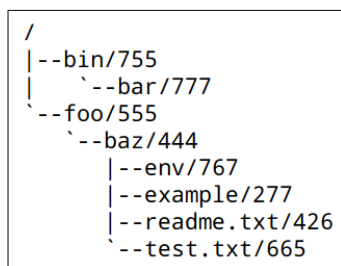


Рисунок 2.11 — Вариант 11 к заданию 3

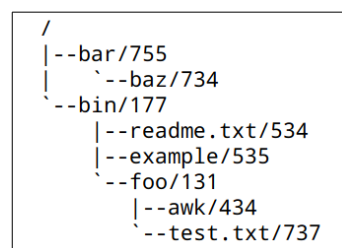


Рисунок 2.12 — Вариант 12 к заданию 3

1) *Для заданий 1-2:* Заменить на пробелы все символы, совпадающие с первым символом в строке. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **rename** (см. рис. 2.12).

Вариант 13

1) *Для заданий 1-2:* Заменить заданную пару букв на символы **#@**. Параметры командной строки: 1. Имя входного файла; 2. Заданная пара букв.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **chown** (см. рис. 2.13).

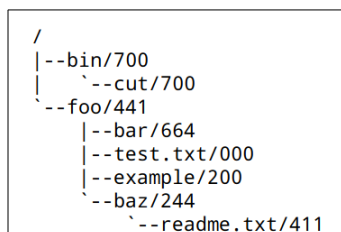


Рисунок 2.13 — Вариант 13 к заданию 3

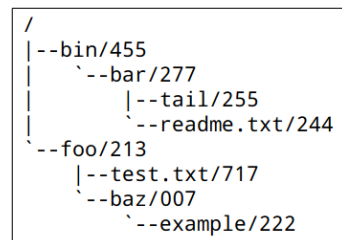


Рисунок 2.14 — Вариант 14 к заданию 3

Вариант 14

1) *Для заданий 1-2:* Заменить все цифры заданным символом. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция **rmdir** (см. рис. 2.14).

Вариант 15

1) *Для заданий 1-2:* Заменить на пробел все символы, совпадающие с последним символом в строке. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) Для задания 3: Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `mkdir` (см. рис. 2.15).

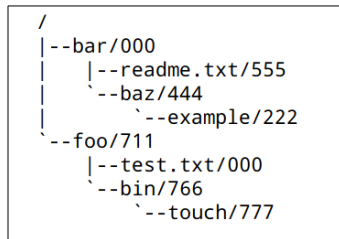


Рисунок 2.15 — Вариант 15 к заданию 3

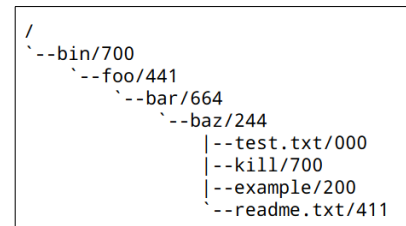


Рисунок 2.16 — Вариант 16 к заданию 3

Вариант 16

1) Для заданий 1-2: Заменить все символы с кодами меньше 48 на пробелы. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) Для задания 3: Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `symlink` (см. рис. 2.16).

Вариант 17

1) Для заданий 1-2: Заменить все символы с кодами больше 48 на пробелы. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) Для задания 3: Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `mkdir` (см. рис. 2.17).

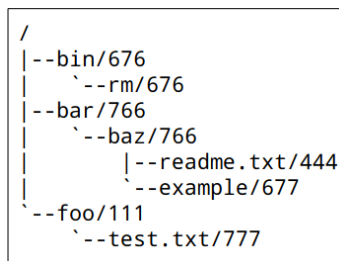


Рисунок 2.17 — Вариант 17 к заданию 3

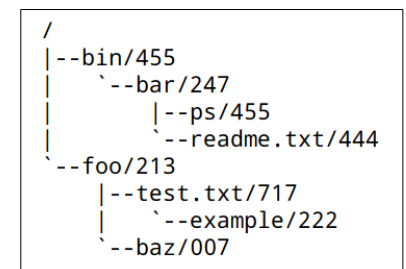


Рисунок 2.18 — Вариант 18 к заданию 3

Вариант 18

1) Для заданий 1-2: Заменить каждый третий символ на пробел. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) Для задания 3: Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `symlink` (см. рис. 2.18).

Вариант 19

1) *Для заданий 1-2:* Заменить все пробелы на заданный символ. Параметры командной строки: 1. Имя входного файла; 2. Заданный символ.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `chown` (см. рис. 2.19).

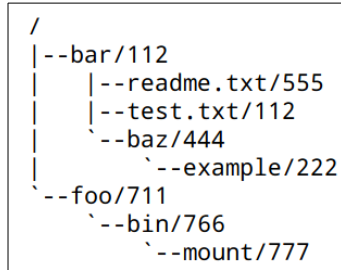


Рисунок 2.19 — Вариант 19 к заданию 3

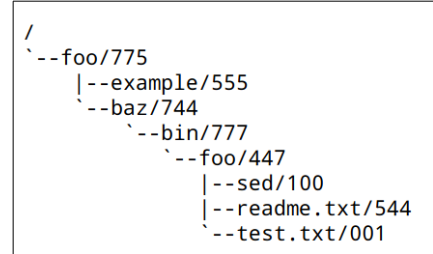


Рисунок 2.20 — Вариант 20 к заданию 3

Вариант 20

1) *Для заданий 1-2:* Заменить все пары одинаковых символов на пробелы. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `chmod` (см. рис. 2.20).

Вариант 21

1) *Для заданий 1-2:* Заменить запятую на две точки. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `mkdir` (см. рис. 2.21).

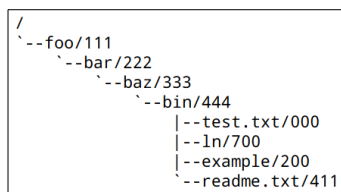


Рисунок 2.21 — Вариант 21 к заданию 3

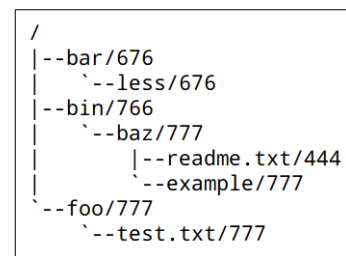


Рисунок 2.22 — Вариант 22 к заданию 3

Вариант 22

1) *Для заданий 1-2:* После каждой запятой вставить символ '\n'. Параметры командной строки: 1. Имя входного файла; 2. Количество замен.

2) *Для задания 3:* Дополнительно к условию задания 3 (общее описание) должна быть реализована функция `symlink` (см. рис. 2.22).

2.4 Контрольные вопросы

1. Перечислите стандартные дескрипторы файлов. Чем они отличаются от дескрипторов обычных файлов?
2. В чем заключается универсальность модели ввода-вывода UNIX?
3. В чем отличия вызова функции `open()` для создания нового файла и открытия существующего?
4. Перечислите форматы и значения третьего аргумента функции `open()`.
5. Перечислите дополнительные (кроме режима доступа) флаги функции `open()`.
6. Каковы основные ошибки, могущие возникнуть при открытии файла?
7. Каковы особенности работы функции `read()`?
8. Каковы особенности работы функции `write()`?
9. Почему нужно явно вызывать функцию `close()`?
10. Для чего служит функция `lseek()`? Какие ее допустимые аргументы? К каким типам файлов ее нельзя применять?
11. Какие виды блокировки файла существуют в Linux?
12. Приведите описание структуры блокировки файла, опишите ее поля.
13. Чем отличается применение функций блокировки `fcntl()` и `lockf()`?
14. Возможно ли сочетать функции блокировки POSIX с файловыми функциями стандартной библиотеки C (`<stdio.h>`)? Почему?
15. Каковы особенности функций блокировки файлов средствами стандартной библиотеки C (`<stdio.h>`)?
16. В чем состоит необходимость организации библиотек объектов?
17. Дайте определение статической и динамической библиотеки. С помощью какой команды создаются и редактируются статические библиотеки? Перечислите варианты подключения статических библиотек.
18. Назовите особенности разделяемых библиотек и опишите процесс их создания. В чем особенности их применения?
19. Опишите команды `objdump`, `readelf`, `nm`.

20. Что такое динамически загружаемые библиотеки?
21. Перечислите функции интерфейса `dlopen`.
22. В чем заключаются особенности применения функции `dlopen()`.
Какие ее флаги используются?
23. Каковы особенности применения функции `dlsym()`?
24. Каковы особенности файловой системы `FUSE`?
25. Опишите схему работы `FUSE`.
26. Какие функции реализованы в `fuse_operation`?
27. Перечислить основные этапы, которые необходимо выполнить с помощью системы сборки *Autotools* для генерации скрипта `./configure`.
28. Для чего предназначен сценарий `./configure`?