**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Операционные системы»**

**Тема: Файловые системы в Unix-подобных ОС**

| Студентка гр. 2384 |  | Соц Е.А. |
| --- | --- | --- |
| Преподаватель |  | Душутина Е.В. |

Санкт-Петербург

2024

## **Цель работы.**

Проанализировать функциональное назначение структурных элементов дерева ФС. Определить размещение корневого каталога (корневой ФС).

## **Задание.**

1. Ознакомиться с типами файлов исследуемой ФС. Применяя утилиту ***ls***, отфильтровать по одному примеру каждого типа файла используемой вами ФС. Комбинируя различные ключи утилиты рекурсивно просканировать все дерево, анализируя крайнюю левую позицию выходной информации полученной посредством ***ls –l***. Результат записать в выходной файл с указанием полного пути каждого примера. Выполнить задание сначала в консоли построчно, выбирая необходимые сочетания ключей (в командной строке), а затем оформить как скрипт с задаваемым в командной строке именем файла как параметр

Файлы системы могут быть следующих типов:

- Обычный файл

b Специальный файл блочного устройства

c Файл символьного устройства

d Директория

l Символьная ссылка

p FIFO

s Сокет

2. Получить все *жесткие ссылки* на заданный файл, находящиеся в разных каталогах пользовательского пространства (разными способами, не применяя утилиты ***file*** и ***find***). Использовать конвейеризацию и фильтрацию. Оформить в виде скрипта.

3. Проанализировать все возможные способы формирования *символьных ссылок* (ln, link,cp и т.д.), продемонстрировать их экспериментально. Предложить скрипт, подсчитывающий и перечисляющий все полноименные символьные ссылки на файл, размещаемые в разных местах файлового дерева.

4. Получить все символьные ссылки на заданный в качестве входного параметра файл, не используя file (разными способами, не применяя утилиту ***file***).

5. Изучить утилиту ***find***, используя ее ключи получить расширенную информацию о всех типах файлов. Создать *примеры вложенных команд*.

6. Проанализировать *содержимое заголовка файла*, а также файла-каталога с помощью утилит ***od*** и ***\*dump***. Если доступ к файлу-каталогу возможен (для отдельных модификаций POSIX-совместимых ОС), проанализировать изменение его содержимого при различных операциях над элементами, входящими в его состав (файлами и подкаталогами).

7. Определить максимальное количество записей в каталоге. Изменить размер каталога, варьируя количество записей (для этого создать программу, порождающую новые файлы и каталоги, а затем удаляющую их, предусмотрев промежуточный и конечный вывод информации о размере подопытного каталога).

8. Ознакомиться с содержимым ***/etc/passwd, /etc/shadow***, с утилитой ***/usr/bin/passwd***, проанализировать права доступа к этим файлам.

9. Исследовать права владения и доступа, а также их сочетаемость

9.1. Привести примеры применения утилит chmod, chown к специально созданному для этих целей отдельному каталогу с файлами.

9.2. Расширить права исполнения экспериментального файла с помощью флага **SUID**.

9.3. Экспериментально установить, как формируются итоговые права на использование файла, если права пользователя и группы, в которую он входит, различны.

9.4. Сопоставить возможности исполнения наиболее часто используемых операций, варьируя правами доступа к файлу и каталогу.

10. Разработать «программу-шлюз» для доступа к файлу другого пользователя при отсутствии прав на чтение информации из этого файла. Провести эксперименты для случаев, когда пользователи принадлежат одной и разным группам. Сравнить результаты. Для выполнения задания применить подход, аналогичный для обеспечения функционирования утилиты ***/usr/bin/passwd*** (манипуляции с правами доступа, флагом **SUID,** а также размещением файлов).

11. Применяя утилиту df и аналогичные ей по функциональности утилиты, а также информационные файлы типа fstab, получить информацию о файловых системах, *возможных* для монтирования, а также *установленных* на компьютере *реально*.

11.1. Привести информацию об исследованных *утилитах* и *информационных файлах* с анализом их содержимого и *форматов*.

11.2. Привести образ диска с точки зрения состава и размещения всех ФС на испытуемом компьютере, а также образ полного дерева ФС, включая присоединенные ФС съемных и несъемных носителей. Проанализировать и указать формат таблицы монтирования.

11.3. Привести «максимально возможное» дерево ФС, проанализировать, где это указывается

12. Проанализировать и пояснить принцип работы утилиты ***file***.

12.1. Привести алгоритм её функционирования на основе информационной базы, размещение и полное имя которой указывается в описании утилиты в технической документации ОС (как правило, ***/usr/share/file/magic.\****), а также содержимого заголовка файла, к которому применяется утилита. Определить, где находятся магические числа и иные характеристики, идентифицирующие тип файла, применительно к *исполняемым* файлам, а также файлам других типов.

12.2. Утилиту ***file*** выполнить с разными ключами.

12.3. Привести экспериментальную попытку с добавлением в базу собственного типа файла и его дальнейшей идентификацией. Описать эксперимент и привести последовательность действий для расширения функциональности утилиты ***file*** и возможности встраивания дополнительного типа файла в ФС (согласовать содержимое информационной базы и заголовка файла нового типа

## **Выполнение работы.**

1. Ознакомиться с типами файлов исследуемой ФС. Применяя утилиту ***ls***, отфильтровать по одному примеру каждого типа файла используемой вами ФС. Комбинируя различные ключи утилиты рекурсивно просканировать все дерево, анализируя крайнюю левую позицию выходной информации полученной посредством ***ls –l***. Результат записать в выходной файл с указанием полного пути каждого примера. Выполнить задание сначала в консоли построчно, выбирая необходимые сочетания ключей (в командной строке), а затем оформить как скрипт с задаваемым в командной строке именем файла как параметр

Файлы системы могут быть следующих типов:

* - Обычный файл
* b Специальный файл блочного устройства
* c Файл символьного устройства
* d Директория
* l Символьная ссылка
* p FIFO
* s Сокет

Команда ls -l | grep -m1 -E '^-[r, w, x, -]{9}' производит поиск файлов каждого типа (- обычный файл, b специальный файл блочного устройства, с файл символьного устройства, d директория, l символьная ссылка, р FIFO, s сокет).

* grep: Это утилита командной строки, используемая для поиска текста по заданному шаблону. В данном случае она используется для вывода команды фильтра ls -l.
* -m1: Этот параметр ограничивает количество выводимых строк до одной. Это значит, что grep остановится после нахождения первого совпадения с заданным шаблоном.
* -E: Этот параметр указывает grep использовать расширенные регулярные выражения (Extended Regular Expressions, ERE) для поиска шаблона. Это позволяет использовать более сложные шаблоны поиска, чем в базовых регулярных выражениях.
* ‘^-[r, w, x, -]{9}’: Это регулярное выражение, используемое для поиска в выводе ls -l. Оно ищет строки, начинающиеся с символа - (что указывает на тип файла, в данном случае обычный файл), за которым следуют 9 символов, представляющих права доступа к файлу.

С помощью команды readlink -f "name\_of\_file" можно узнать полное имя файла.

1)поиск блочных устройств

(base) katya@katya:/dev$ ls -l | grep -m1 -E '^b[r, w, x, -]{9}'

brw-rw---- 1 root disk 7, 0 апр 1 14:27 loop0

(base) katya@katya:/dev$ readlink -f "loop0"

/dev/loop0

2)символьные устройства

(base) katya@katya:/dev$ ls -l | grep -m1 -E '^c[r, w, x, -]{9}'

crw-r--r-- 1 root root 10, 235 апр 1 14:27 autofs

(base) katya@katya:/dev$ readlink -f "autofs"

/dev/autofs

3)каталоги

(base) katya@katya:/dev$ ls -l | grep -m1 -E '^d[r, w, x, -]{9}'

drwxr-xr-x 2 root root 820 апр 1 14:27 block

(base) katya@katya:/dev$ readlink -f "block"

/dev/block

4)символьные ссылки

(base) katya@katya:/dev$ ls -l | grep -m1 -E '^l[r, w, x, -]{9}'

lrwxrwxrwx 1 root root 11 апр 1 14:27 core -> /proc/kcore

(base) katya@katya:/dev$ readlink -f "core"

/proc/kcore

5)каналы

(base) katya@katya:/run$ ls -l | grep -m1 -E '^p[r, w, x, -]{9}'

prw------- 1 root root 0 апр 1 14:27 initctl

(base) katya@katya:/run$ readlink -f "initctl"

/run/initctl

6)сокеты

(base) katya@katya:/run$ ls -l | grep -m1 -E '^s[r, w, x, -]{9}'

srw-rw-rw- 1 root root 0 апр 1 14:27 acpid.socket

(base) katya@katya:/run$ readlink -f "acpid.socket"

/run/acpid.socket

7)обычные файлы

(base) katya@katya:/run$ ls -l | grep -m1 -E '^-[r, w, x, -]{9}'

-rw-r--r-- 1 root root 4 апр 1 14:27 acpid.pid

(base) katya@katya:/run$ readlink -f "acpid.pid"

/run/acpid.pid

Разработан скрипт, который выводит все типы файлов, которые есть в введенной директории: find.sh

#!/bin/bash

# Проверяем, что аргумент командной строки был передан

if [ $# -eq 0 ]; then

echo "Пожалуйста, укажите директорию для поиска файлов."

exit 1

fi

# Перебираем все типы файлов

for type in f d l b c s p; do

# Используем find для поиска файлов данного типа

files=$(find "$1" -type "$type")

# Проверяем, есть ли файлы данного типа

if [ -n "$files" ]; then

echo "Тип файла: $type"

(#)echo "$files"

fi

done

Чтобы использовать скрипт, нужно сначала сделать его исполняемым с помощью команды chmod +x find.sh, затем запустить его, указывая путь к директории, в которой будет проводится проверка файлов на тип ./find.sh /path/to/dir.

Примеры выполнения скрипта:

1. c выводом полного пути

(base) katya@katya:~/os/lb2$ ./find.sh /home/katya/piaa/1лаб

Тип файла: f

/home/katya/piaa/1лаб/исследование.xlsx

/home/katya/piaa/1лаб/пиа1.pdf

/home/katya/piaa/1лаб/time.py

/home/katya/piaa/1лаб/пиа1.docx

/home/katya/piaa/1лаб/lb1.py

Тип файла: d

/home/katya/piaa/1лаб

1. без вывода полного пути

(base) katya@katya:~/os/lb2$ ./find.sh /dev

Тип файла: d

Тип файла: l

Тип файла: b

**2**. Получить все *жесткие ссылки* на заданный файл, находящиеся в разных каталогах пользовательского пространства (разными способами, не применяя утилиты ***file*** и ***find***). Использовать конвейеризацию и фильтрацию. Оформить в виде скрипта.

Разработан скрипт, который ищет все жесткие ссылки на файл: hard\_links.sh

#!/bin/bash

if [ $# -lt 1 ]

then

echo $0: error

else

filename=$1

inode=$(ls -i $filename | cut -d ' ' -f 1 | tr -d " ")

tmp=$(ls -lRi /home/katya | grep $inode)

fi

echo $tmp

Пример использования скрипта:

(base) katya@katya:~/os/lb2$ ./hard\_links.sh /home/katya/os/lb1/prog.c

3022534 -rw-rw-r-- 1 katya katya 65 фев 26 18:27 prog.c

(base) katya@katya:~/os/lb2$ ./hard\_links.sh /home/katya/piaa/2лаб/num1.py

3579405 -rw-rw-r-- 1 katya katya 1334 мар 26 22:39 num1.py

**3**. Проанализировать все возможные способы формирования *символьных ссылок* (ln, link,cp и т.д.), продемонстрировать их экспериментально. Предложить скрипт, подсчитывающий и перечисляющий все полноименные символьные ссылки на файл, размещаемые в разных местах файлового дерева.

Формирование символьной ссылки:

1. ln -s /path/to/original /path/to/symlink

(base) katya@katya:~$ ln -s /home/katya/os/lb1/prog.c /home/katya/os/lb2

(base) katya@katya:~/os/lb2$ ls -l

итого 16

-rwxrwxr-x 1 katya katya 641 апр 2 00:56 find.sh

-rwxrwxr-x 1 katya katya 251 апр 2 00:17 find\_types.sh

-rwxrwxr-x 1 katya katya 176 апр 2 19:22 hard\_links.sh

-rw-rw-r-- 1 katya katya 2372 апр 2 19:27 logfile

lrwxrwxrwx 1 katya katya 25 апр 2 19:41 **prog.c -> /home/katya/os/lb1/prog.c**

1. cp -s /path/to/original /path/to/symlink

(base) katya@katya:~$ cp -s /home/katya/piaa/2лаб/num1.py /home/katya/os/lb2

(base) katya@katya:~$ cd os/lb2

(base) katya@katya:~/os/lb2$ ls -l

итого 16

-rwxrwxr-x 1 katya katya 641 апр 2 00:56 find.sh

-rwxrwxr-x 1 katya katya 251 апр 2 00:17 find\_types.sh

-rwxrwxr-x 1 katya katya 176 апр 2 19:22 hard\_links.sh

-rw-rw-r-- 1 katya katya 2372 апр 2 19:27 logfile

lrwxrwxrwx 1 katya katya 32 апр 2 19:56 **num1.py -> /home/katya/piaa/2лаб/num1.py**

lrwxrwxrwx 1 katya katya 25 апр 2 19:41 prog.c -> /home/katya/os/lb1/prog.c

1. Команду link для создания символьных ссылок не получится использовать, потому что с помощью нее создаются жесткие ссылки. Однако, решение может быть таким: создание жесткой ссылки на символьную ссылку, тогда для файла, на который ссылается символьная ссылка, жесткая тоже будет символьной.

Создание скрипта, подсчитывающего и перечисляющего символьные ссылки на заданный файл: links.sh

#!/bin/bash

filename=$1

ls -lRa /home/katya/ | grep $filename | grep ^l > links.txt

#подсчет кол-ва ссылок:

echo -n "total " >> links.txt

wc -l links.txt | cut -c 1 >> links.txt

com=$(cat links.txt)

echo $com

Пример использования скрипта:

(base) katya@katya:~/os/lb2$ ./links.sh /home/katya/os/lb1/prog.c

lrwxrwxrwx 1 katya katya 25 апр 2 20:49 link2.c -> /home/katya/os/lb1/prog.c lrwxrwxrwx 1 katya katya 25 апр 2 19:41 prog.c -> /home/katya/os/lb1/prog.c total 2

**4.** Получить все символьные ссылки на заданный в качестве входного параметра файл, не используя file (разными способами, не применяя утилиту ***file***).

Получить все символьные ссылки на заданный в качестве входного параметра файл можно по принципу, который был использован в скрипте выше: ls -lRa /home/katya/ | grep *your\_file* | grep ^l

(base) katya@katya:~/os/lb2$ ls -lRa /home/katya/ | grep /home/katya/os/lb1/prog.c | grep ^l

lrwxrwxrwx 1 katya katya 25 апр 2 20:49 link2.c -> /home/katya/os/lb1/prog.c

lrwxrwxrwx 1 katya katya 25 апр 2 19:41 prog.c -> /home/katya/os/lb1/prog.c

**5**. Изучить утилиту ***find***, используя ее ключи получить расширенную информацию о всех типах файлов. Создать *примеры вложенных команд*.

Утилита find в Linux предоставляет мощные возможности для поиска файлов и каталогов по различным критериям, включая тип файла, размер, дату изменения и многое другое. Для получения расширенной информации о всех типах файлов можно использовать различные ключи find:

1. name - поиск по шаблону имени файла
2. iname - поиск по шаблону имени файла, но без учета регистра
3. chmod mode - изменение прав доступа к найденному файлу
4. type file\_type - поиск файлов определенного типа (b, c, d, p, f(regular file), l, n, s)
5. print - вывод полного имени найденного файла
6. ls - вывод имени файла в формате команды ls -l
7. lname file - поиск символических ссылок на определенный файл
8. inode file(n) - поиск файлов с тем же серийным номером, что и определенный файл, или серийным номером н
9. level n - поиск файлов, расположенных в дереве каталогов на н уровней ниже заданного каталога
10. size - поиск файлов по заданному размеру
11. mtime - поиск файлов по дате последнего изменения

Примеры выполнения некоторых команд:

1. Поиск всех файлов в текущем каталоге и его подкаталогах:

(base) katya@katya:~/os/lb2$ find . -type f

./logfile

./links.sh

./find\_types.sh

./hard\_links.sh

./links.txt

./find.sh

1. Поиск всех директорий в текущем каталоге и его подкаталогах:

(base) katya@katya:~/os$ find . -type d

.

./lb2

./lb1

./lb1/prog

./lb1/multicom

1. Поиск файлов определенного типа:

(base) katya@katya:~/os$ find . -type f -name "\*.txt"

./lb2/links.txt

./lb1/dublicate.txt

./lb1/output.txt

./lb1/try.txt

1. Поиск файлов определенного размера(больше 500 Мб)

(base) katya@katya:~$ find . -type f -size +500M

./anaconda3/envs/drones/lib/python3.10/site-packages/nvidia/cudnn/lib/libcudnn\_cnn\_infer.so.8

./anaconda3/envs/drones/lib/python3.10/site-packages/torch/lib/libtorch\_cuda.so

./Загрузки/Anaconda3-2023.\_-CTXuNv.09-0-Linux-x86\_64.sh.part

./.cache/pip/http-v2/c/4/1/2/a/c412a6b66698d2a2a51ad6d9e6392a8f4b38e111ea0cf26dda466924.body

./.cache/pip/http-v2/3/c/e/f/9/3cef90e2f33f3b9a1b50e02cc0736e09cc97714cb8b1101d706d912d.body

1. Поиск файлов, измененных в течении последних н дней ( 7 дней):

(base) katya@katya:~/os/lb2$ find . -type f -mtime -7

./logfile

./links.sh

./find\_types.sh

./hard\_links.sh

./links.txt

./find.sh

В первом задании был написан скрипт, который определяет типы файлов, используя команду find.

Вложенные команды позволяют использовать вывод одной команды в качестве входных данных для другой команды.

Примеры вложенных команд:

1. вывод списка файлов с их размерами:

(base) katya@katya:~/os/lb2$ find . -type f -exec ls -lh {} \; | awk '{print $5, $9}'

3,8K ./logfile

230 ./links.sh

251 ./find\_types.sh

176 ./hard\_links.sh

173 ./links.txt

641 ./find.sh

1. изменение прав доступа к найденным файлам:

(base) katya@katya:~/os/lb2$ find . -type f -name "\*.sh" | xargs chmod +x

1. нахождение файла по содержанию и удаление этого файла:

(base) katya@katya:~/os/lb2/text$ ls

1.txt 2.txt 3.txt 4.txt

(base) katya@katya:~/os/lb2/text$ cat 1.txt && cat 2.txt && cat 3.txt && cat 4.txt

hello

katya sots

hello world

sots sots kakakakakakakakak

(base) katya@katya:~/os/lb2/text$ find . -type f -exec grep -l "sots" {} \; | xargs rm -f

(base) katya@katya:~/os/lb2/text$ ls

1.txt 3.txt

**6**. Проанализировать *содержимое заголовка файла*, а также файла-каталога с помощью утилит ***od*** и ***\*dump***. Если доступ к файлу-каталогу возможен (для отдельных модификаций POSIX-совместимых ОС), проанализировать изменение его содержимого при различных операциях над элементами, входящими в его состав (файлами и подкаталогами).

Od предназначена для отображения содержимого файлов в различных форматах.

Анализ содержимого файла с помощью утилит od:

#создаем пустой файл

(base) katya@katya:~/os/lb2$ od task6.txt

0000000

# редактируем

(base) katya@katya:~/os/lb2$ nano task6.txt

(base) katya@katya:~/os/lb2$ cat task6.txt

katya sots

#вывод в 16ричной системе

(base) katya@katya:~/os/lb2$ od task6.txt

0000000 060553 074564 020141 067563 071564 000012

0000013

#вывод в символьном формате

(base) katya@katya:~/os/lb2$ od -tc task6.txt

0000000 k a t y a s o t s \n

0000013

#редактируем файл

(base) katya@katya:~/os/lb2$ nano task6.txt

(base) katya@katya:~/os/lb2$ cat task6.txt

katya sots

LAB 2 OS

04.04

(base) katya@katya:~/os/lb2$ od -tc task6.txt

0000000 k a t y a s o t s \n L A B 2

0000020 O S \n 0 4 . 0 4 \n

0000032

# редактируем файл

(base) katya@katya:~/os/lb2$ nano task6.txt

(base) katya@katya:~/os/lb2$ cat task6.txt

katya sots

LAB 2 OS

(base) katya@katya:~/os/lb2$ od -tc task6.txt

0000000 k a t y a s o t s \n L A B 2

0000020 O S \n

0000024

Таким образом, od (octal dump) позволяет получить гибридный текстовый и восьмеричный дамп файла. Это полезно для анализа бинарных файлов, так как оно позволяет увидеть как числовые, так и символьные данные в файле.

Флаг -tc выводит содержимое файла в текстовом формате.

Левый столбец при использовании od - количество байт на ячейку. Заметно, что при редактировании этот размер меняется.

\*dump также используется для отображения содержимого файлов, но отображает более детальный вывод, включающий адреса, символы и их 16-ричное представление.

Для получения информации об объектном или исполняемом файле, нужно воспользоваться командой objdump. Основная цель данной команды - помочь в отладке объектного файла, предоставляя информацию о заголовках файлов, содержимом секций, символьной таблице и отладочной информации.

(base) katya@katya:~/os/lb2$ objdump -f a.out

a.out: формат файла elf64-x86-64

архитектура: i386:x86-64, флаги 0x00000150:

HAS\_SYMS, DYNAMIC, D\_PAGED

начальный адрес 0x0000000000001060

* Формат файла: elf64-x86-64. Это означает, что файл a.out является объектным файлом в формате ELF (Executable and Linkable Format) для архитектуры x86-64.
* Архитектура: i386:x86-64. Это указывает на то, что файл предназначен для выполнения на процессорах семейства x86-64, которые поддерживают 64-битную архитектуру.
* Флаги: 0x00000150. Флаги указывают на различные атрибуты файла, такие как наличие символьной таблицы (HAS\_SYMS), динамическую связь (DYNAMIC), и страницы памяти (D\_PAGED).
* Начальный адрес: 0x0000000000001060. Это адрес, с которого начинается исполняемый код в файле.

**7**. Определить максимальное количество записей в каталоге. Изменить размер каталога, варьируя количество записей (для этого создать программу, порождающую новые файлы и каталоги, а затем удаляющую их, предусмотрев промежуточный и конечный вывод информации о размере подопытного каталога).

Был написан скрипт, выполняющий задание: 7task.sh

#!/bin/bash

#размер директории

size=$(du -hs $1)

echo "размер директории:" $size

#создаем новые файлы txt

for i in {1..50}

do

name=$1/$i.txt

>$name

#запись информации (для придания файлам "веса")

echo "Sots Ekateryna Andreevna 2384" >> $name

mkdir $1/$i

done

#размер директории после изменений

size=$(du -hs $1)

echo "размер директории после добавления файлов:" $size

#удаляем некоторые файлы

for i in {27..47}

do

name=$1/$i.txt

rm -rf $name $1/$i

done

#размер директории после изменений

size=$(du -hs $1)

echo "размер директории после удаления файлов:" $size

Результаты выполнения программы:

(base) katya@katya:~/os/lb2$ ./7task.sh 7

размер директории: 4,0K 7

размер директории после добавления файлов: 404K 7

размер директории после удаления файлов: 236K 7

**8**. Ознакомиться с содержимым ***/etc/passwd, /etc/shadow***, с утилитой ***/usr/bin/passwd***, проанализировать права доступа к этим файлам.

Для ознакомления с содержимым файлов можно, для начала, просмотреть их содержимое:

1. cat /etc/passwd : содержит информацию о каждом пользователе системы, включая имя пользователя, хеш пароля, UID, GID, домашний каталог, оболочку и другую информацию. Владелец файла имеет права на чтение и запись, а остальные пользователи — только на чтение.

(base) katya@katya:~$ ls -l /etc/passwd

-rw-r--r-- 1 root root 2887 ноя 14 14:19 /etc/passwd

1. sudo cat /etc/shadow: содержит зашифрованные пароли пользователей, информацию о сроке действия паролей, количество дней до смены пароля и другую конфиденциальную информацию. Владелец файла имеет права на чтение и запись, а остальные пользователи — только на чтение. Для доступа к этому файлу требуются права суперпользователя.

(base) katya@katya:~$ sudo ls -l /etc/shadow

-rw-r----- 1 root shadow 1464 ноя 14 14:19 /etc/shadow

1. Утилита /usr/bin/passwd используется для управления паролями пользователей. Она позволяет изменять пароли, устанавливать сроки действия паролей и выполнять другие операции, связанные с управлением учетными записями пользователей. Для использования этой утилиты требуются права суперпользователя. У всех есть доступ к выполнению и чтению, изменять может только владелец. s – setuid бит разрешения, который позволяет пользователю запускать исполняемый файл с правами владельца этого файла.

(base) katya@katya:~$ sudo ls -l /usr/bin/passwd

-rwsr-xr-x 1 root root 59976 фев 6 15:54 /usr/bin/passwd

Пример использования данной утилиты для изменения пароля текущего пользователя:

sudo passwd username, где username - имя пользователя

(base) katya@katya:~$ sudo passwd katya

Новый пароль:

НЕУДАЧНЫЙ ПАРОЛЬ: Пароль должен содержать не менее 8 символов

Повторите ввод нового пароля:

passwd: пароль успешно обновлён

**9.** Исследовать права владения и доступа, а также их сочетаемость.

В Linux каждый файл и каталог имеет права владения и права доступа, которые определяют, кто может читать, записывать или выполнять файл. Эти права разделены на три категории: владелец файла, группа, к которой принадлежит файл, и все остальные пользователи. Права доступа включают чтение (read), запись (write) и выполнение (execute).

Чтобы просмотреть права владения и доступа к файлу, используйте команду ls -l (использовалось в п. 8). Вывод команды будет содержать информацию о типе файла, правах владения и доступа, количестве ссылок, владельце, группе, размере файла, дате последнего изменения и имени файла.

Права владения и доступа могут быть сочетаемыми, что позволяет тонко настраивать доступ к файлам и каталогам. Например, владелец файла может иметь права на чтение, запись и выполнение, в то время как группа и остальные пользователи могут иметь только права на чтение. Это обеспечивает гибкость в управлении доступом к файлам и каталогам, позволяя ограничивать доступ к конфиденциальным данным и системным файлам. Управление правами владения и доступа является ключевым аспектом безопасности в Linux, и его следует выполнять с осторожностью, чтобы избежать несанкционированного доступа к файлам и каталогам.

**9.1.** Привести примеры применения утилит chmod, chown к специально созданному для этих целей отдельному каталогу с файлами.

1. Изменение владельца файла: chown новый\_владелец имя\_файла
2. Изменение группы файла: chgrp новая\_группа имя\_файла
3. Символьный метод:

* chmod u+x имя\_файла - Добавить права на выполнение для владельца
* chmod g-w имя\_файла - Убрать права на запись для группы
* chmod o+r имя\_файла - Добавить права на чтение для всех остальных

1. Числовой метод:

chmod 755 имя\_файла - Установить права на чтение, запись и выполнение для владельца, и только на чтение и выполнение для группы и всех остальных ( 7 - разрешены чтение, запись, исполнение; 6 - разрешены чтение и запись; 5 - разрешены чтение и исполнение, 4 - разрешено только чтение; 0 - ничего не разрешено)

Пример применения утилит:

#создали каталог и два файла, посмотрели их права доступа

(base) katya@katya:~/os/lb2$ ls -l task9\_1

итого 0

-rw-rw-r-- 1 katya katya 0 апр 7 02:48 file1.txt

-rw-rw-r-- 1 katya katya 0 апр 7 02:48 file2.txt

#изменение владельца каталогов и файлов

(base) katya@katya:~$ sudo chown kotik:kotik os/lb2/task9\_1

(base) katya@katya:~$ sudo chown kotik:kotik os/lb2/task9\_1/file1.txt

(base) katya@katya:~$ sudo chown kotik:kotik os/lb2/task9\_1/file2.txt

(base) katya@katya:~$ cd os/lb2/task9\_1

(base) katya@katya:~/os/lb2/task9\_1$ ls -l

итого 0

-rw-rw-r-- 1 kotik kotik 0 апр 7 02:48 file1.txt

-rw-rw-r-- 1 kotik kotik 0 апр 7 02:48 file2.txt

#установка прав доступа

(base) katya@katya:~/os/lb2$ sudo chmod 755 task9\_1

(base) katya@katya:~/os/lb2$ sudo chmod 755 task9\_1/file1.txt task9\_1/file2.txt

(base) katya@katya:~/os/lb2$ ls -l task9\_1

итого 0

-rwxr-xr-x 1 kotik kotik 0 апр 7 02:48 file1.txt

-rwxr-xr-x 1 kotik kotik 0 апр 7 02:48 file2.txt

#теперь владелец имеет права на чтение, запись и выполнение, а остальные пользователи — только на чтение и выполнение.

**9.2.** Расширить права исполнения экспериментального файла с помощью флага **SUID**.

Флаг SUID позволяет файлу выполняться с правами владельца файла, что может быть полезно для предоставления временных привилегий пользователям при выполнении определенных операций.

(base) katya@katya:~/os/lb2/task9\_2$ touch file.txt

(base) katya@katya:~/os/lb2/task9\_2$ ls -l

итого 0

-rw-rw-r-- 1 katya katya 0 апр 7 03:19 file.txt

(base) katya@katya:~/os/lb2/task9\_2$ chmod u+s file.txt

(base) katya@katya:~/os/lb2/task9\_2$ ls -l

итого 0

-rwSrw-r-- 1 katya katya 0 апр 7 03:19 file.txt

появился символ s, это значит, что файл может выполняться с правами владельца файла.

**9.3.** Экспериментально установить, как формируются итоговые права на использование файла, если права пользователя и группы, в которую он входит, различны.

#создание файла

(base) katya@katya:~/os/lb2/task9\_3$ touch file9\_3.txt

#установка прав владельца на чтение, запись и выполнение

(base) katya@katya:~/os/lb2/task9\_3$ chmod 700 file9\_3.txt

(base) katya@katya:~/os/lb2/task9\_3$ ls -l

итого 0

-rwx------ 1 katya katya 0 апр 7 15:47 file9\_3.txt

#изменение группы владельца файла

(base) katya@katya:~/os/lb2/task9\_3$ chgrp adm file9\_3.txt

(base) katya@katya:~/os/lb2/task9\_3$ ls -l

итого 0

-rwx------ 1 katya adm 0 апр 7 15:47 file9\_3.txt

#владелец может читать, записывать и выполнять файл, а члены группы могут читать и выполнять файл, но не могут записывать в него

(base) katya@katya:~/os/lb2/task9\_3$ chmod 750 file9\_3.txt

(base) katya@katya:~/os/lb2/task9\_3$ ls -l

итого 0

-rwxr-x--- 1 katya adm 0 апр 7 15:47 file9\_3.txt

Вывод показывает, что права владельца установлены на rwx, а права группы — на r-x.

Этот пример демонстрирует, как формируются итоговые права на использование файла, учитывая различия в правах пользователя и группы.

При отсутствии каких-либо прав у пользователя, но при наличии прав у группы, файл становится недоступен. При обратной же ситуации, права будут. Можно сделать вывод о том, что итоговые права формируются на основании прав доступа пользователя.

**9.4.** Сопоставить возможности исполнения наиболее часто используемых операций, варьируя правами доступа к файлу и каталогу.

|  | Файл | Каталог |
| --- | --- | --- |
| Чтение | отображение информации в файле, возможность копирования | отображение файлов и директорий, которые содержатся в этом каталоге |
| Запись | возможность удалять файл, изменять и сохранять | добавление или удаление файлов в этом каталоге |
| Выполнение | запуск файла, если он исполняемый | переход в этот каталог |

Важные моменты:

* Право на исполнение для каталога не дает возможности выполнять файлы внутри каталога напрямую. Для этого файлам также должно быть предоставлено право на исполнение.
* Право на запись в каталог позволяет добавлять новые файлы и подкаталоги, но не дает возможности изменять содержимое уже существующих файлов, если у них нет соответствующих прав.
* Право на чтение для каталога позволяет просматривать содержимое каталога, но не дает возможности просматривать содержимое файлов внутри каталога, если у файлов нет прав на чтение.

**10.** Разработать «программу-шлюз» для доступа к файлу другого пользователя при отсутствии прав на чтение информации из этого файла. Провести эксперименты для случаев, когда пользователи принадлежат одной и разным группам. Сравнить результаты. Для выполнения задания применить подход, аналогичный для обеспечения функционирования утилиты ***/usr/bin/passwd*** (манипуляции с правами доступа, флагом **SUID,** а также размещением файлов).

#создание и проверка шлюз-программы

(base) katya@katya:~/os/lb2$ gcc task10.c -o task10.out

(base) katya@katya:~/os/lb2$ ./task10.out

Sots Ekateryna

(base) katya@katya:~/os/lb2$ ls -l task10.out

-rwxrwxr-x 1 katya katya 16136 апр 7 16:44 task10.out

#добавление исполняемому файлу программы разрешение SUID

(base) katya@katya:~/os/lb2$ chmod +s task10.out

(base) katya@katya:~/os/lb2$ ls -l task10.out

-rwsrwsr-x 1 katya katya 16136 апр 7 16:44 task10.out

#проверка прав доступа у текстового файла

(base) katya@katya:~/os/lb2$ ls -l task10.txt

-rw-rw-r-- 1 katya katya 15 апр 7 16:42 task10.txt

#удаление прав чтения у остальных пользователей

(base) katya@katya:~/os/lb2$ chmod o-r task10.txt

(base) katya@katya:~/os/lb2$ ls -l task10.txt

-rw-rw---- 1 katya katya 15 апр 7 16:42 task10.txt

Теперь, сменив пользователя-владельца, можно убедиться, что вывести содержимое текстового файла не удастся, однако, запустив программу, можно вывести содержимое файла. Это происходят благодаря SUID у исполняемого файла.

**11.** Применяя утилиту df и аналогичные ей по функциональности утилиты, а также информационные файлы типа fstab, получить информацию о файловых системах, *возможных* для монтирования, а также *установленных* на компьютере *реально*.

1. Утилита df позволяет найти тип файловой системы для всех смонтированных устройств хранения и разделов ( -Т - вывод типа файловой системы, -h - вывод размера файловой системы):

(base) katya@katya:~/os/lb2$ df -Th

Файл.система Тип Размер Использовано Дост Использовано% Cмонтировано в

tmpfs tmpfs 1,6G 2,6M 1,5G 1% /run

/dev/nvme0n1p6 ext4 86G 23G 59G 28% /

tmpfs tmpfs 7,6G 64M 7,5G 1% /dev/shm

tmpfs tmpfs 5,0M 4,0K 5,0M 1% /run/lock

efivarfs efivarfs 128K 14K 110K 12% /sys/firmware/efi/efivars

/dev/nvme0n1p7 ext4 90G 29G 57G 34% /home

/dev/nvme0n1p2 vfat 96M 31M 66M 33% /boot/efi

tmpfs tmpfs 1,6G 144K 1,6G 1% /run/user/1000

tmpfs tmpfs 1,6G 176K 1,6G 1% /run/user/1001

tmpfs tmpfs 1,6G 76K 1,6G 1% /run/user/128

1. Файл /etc/fstab содержит информацию о дисковых разделах и их монтировании.

(base) katya@katya:~$ cat /etc/fstab

# /etc/fstab: static file system information.

#

# Use 'blkid' to print the universally unique identifier for a

# device; this may be used with UUID= as a more robust way to name devices

# that works even if disks are added and removed. See fstab(5).

#

# <file system> <mount point> <type> <options> <dump> <pass>

# / was on /dev/nvme0n1p6 during installation

UUID=2054832e-ce3f-4e91-a84e-401d87746dd2 / ext4 errors=remount-ro 0 1

# /boot/efi was on /dev/nvme0n1p2 during installation

UUID=7C6B-67C8 /boot/efi vfat umask=0077 0 1

# /home was on /dev/nvme0n1p7 during installation

UUID=b8953e34-00fc-44c4-be86-df184d799104 /home ext4 defaults 0 2

1. Команда lsblk отображает информацию о блочных устройствах, включая физические диски и их разделы, -f выводит информацию о файловой системе

(base) katya@katya:~$ lsblk -f

NAME FSTYPE FSVER LABEL UUID FSAVAIL FSUSE% MOUNTPOINTS

loop0

squash 4.0 0 100% /snap/bare/5

loop1

squash 4.0 0 100% /snap/clion/265

loop2

squash 4.0 0 100% /snap/code/155

loop3

squash 4.0 0 100% /snap/code/156

loop4

squash 4.0 0 100% /snap/core18/2796

loop5

squash 4.0 0 100% /snap/core18/2812

loop6

squash 4.0 0 100% /snap/core20/2105

loop7

squash 4.0 0 100% /snap/core20/2182

loop8

squash 4.0 0 100% /snap/core22/1033

loop9

squash 4.0 0 100% /snap/core22/1122

loop10

squash 4.0 0 100% /snap/curl/1754

loop11

squash 4.0 0 100% /snap/discord/181

loop12

squash 4.0 0 100% /snap/discord/182

loop13

squash 4.0 0 100% /snap/firefox/3836

loop14

squash 4.0 0 100% /snap/firefox/4033

loop15

squash 4.0 0 100% /snap/gnome-3-38-2004/143

loop16

squash 4.0 0 100% /snap/gnome-42-2204/141

loop17

squash 4.0 0 100% /snap/gnome-42-2204/172

loop18

squash 4.0 0 100% /snap/gtk-common-themes/1535

loop19

squash 4.0 0 100% /snap/pycharm-community/374

loop20

squash 4.0 0 100% /snap/pycharm-community/378

loop21

squash 4.0 0 100% /snap/snap-store/1113

loop22

squash 4.0 0 100% /snap/snap-store/959

loop23

squash 4.0 0 100% /snap/snapd/20671

loop24

squash 4.0 0 100% /snap/snapd/21184

loop25

squash 4.0 0 100% /snap/snapd-desktop-integration/83

loop27

squash 4.0 0 100% /snap/telegram-desktop/5741

loop28

squash 4.0 0 100% /snap/zoom-client/218

loop29

squash 4.0 0 100% /snap/zoom-client/225

loop30

0 100% /snap/snapd-desktop-integration/157

loop31

squash 4.0 0 100% /snap/telegram-desktop/5767

nvme0n1

│

├─nvme0n1p1

│ ntfs Восстановить

│ 2C2E698C2E69503E

├─nvme0n1p2

│ vfat FAT32 7C6B-67C8 65,2M 32% /boot/efi

├─nvme0n1p3

│

├─nvme0n1p4

│ ntfs Windows 10

│ AA486C36486C0403

├─nvme0n1p5

│ ntfs win doc

│ 4498517D98516E84

├─nvme0n1p6

│ ext4 1.0 2054832e-ce3f-4e91-a84e-401d87746dd2 58,9G 26% /var/snap/firefox/common/host-hunspell

│ /

└─nvme0n1p7

ext4 1.0 b8953e34-00fc-44c4-be86-df184d799104 56,1G 32% /home

MOUNTPOINT: Каталог, в котором смонтировано устройство хранения/раздел (файловая система) (если смонтировано).

FSTYPE: Тип файловой системы устройства хранения/раздела.

LABEL: Метка файловой системы устройства хранения/раздела.

UUID: UUID (универсальный уникальный идентификатор) файловой системы устройства хранения/раздела. 20

FSUSE%: Процент дискового пространства, используемого на устройстве хранения/разделе

FSAVAIL: Объем свободного места на диске устройства хранения/раздела -e7 – прячет вывод петлевых (loop) устройств.

(base) katya@katya:~$ lsblk -f -e7

NAME FSTYPE FSVER LABEL UUID FSAVAIL FSUSE% MOUNTPOINTS

nvme0n1

│

├─nvme0n1p1

│ ntfs Восстановить

│ 2C2E698C2E69503E

├─nvme0n1p2

│ vfat FAT32 7C6B-67C8 65,2M 32% /boot/efi

├─nvme0n1p3

│

├─nvme0n1p4

│ ntfs Windows 10

│ AA486C36486C0403

├─nvme0n1p5

│ ntfs win doc

│ 4498517D98516E84

├─nvme0n1p6

│ ext4 1.0 2054832e-ce3f-4e91-a84e-401d87746dd2 58,9G 26% /var/snap/firefox/common/host-hunspell

│ /

└─nvme0n1p7

ext4 1.0 b8953e34-00fc-44c4-be86-df184d799104 56,1G 32% /home

**11.1.** Привести информацию об исследованных *утилитах* и *информационных файлах* с анализом их содержимого и *форматов*.

1. Squash - это файловая система только для чтения. Она используется для упаковки файлов и директорий в единый файл, который потом может быть смонтирован как файловая система.

Основные характеристики SquashFS:

* Сжатие: SquashFS использует различные алгоритмы сжатия, такие как zlib, lz4, lzo, xz и zstandard, для сжатия файлов, inode и каталогов. Это позволяет существенно уменьшить размер файловой системы, сохраняя при этом возможность чтения.
* Блочные размеры: Поддерживаются блочные размеры от 4KiB до 1MiB, что обеспечивает дополнительное сжатие. Размер блока по умолчанию составляет 128K.
* Использование: SquashFS предназначен для использования в системах с ограниченными ресурсами, где требуется минимальное накладное расхождение, а также для общих целей чтения файловых систем и в архивации.

Особенности SquashFS включают в себя поддержку сжатия данных и управления файловыми атрибутами, такими как права доступа и временные метки. Она также поддерживает использование "множества" файловых систем, где одна файловая система может содержать ссылки на другие файловые системы.

1. NTFS (New Technology File System) — это файловая система, разработанная Microsoft для использования в операционных системах семейства Windows. Поддержка NTFS в Linux является важным шагом в обеспечении совместимости между операционными системами. Несмотря на предыдущие ограничения, теперь Linux-пользователи могут полноценно работать с NTFS-разделами, что упрощает совместное использование данных между Windows и Linux-системами.
2. Vfat (Virtual File Allocation Table) - это файловая система, которая была разработана Microsoft и используется для хранения файлов на накопителях с файловой системой FAT32. В Linux, VFAT является одной из наиболее распространенных файловых систем для сменных носителей, таких как флешки, карты памяти и другие устройства. VFAT поддерживает стандартные функции файловых систем FAT, такие как длинные имена файлов, структура каталогов, а также поддерживает файлы размером более 4 ГБ. VFAT также поддерживает стандарты кодирования символов для работы с различными языками и национальными символами.
3. Ext4 (Fourth Extended Filesystem) - это расширенная файловая система, которая является стандартной файловой системой в большинстве современных дистрибутивов Linux. Основные преимущества Ext4 перед своими предшественниками включают в себя повышенную производительность и более быстрое время монтирования. Ext4 также поддерживает большие файлы и директории, а также более быстрое создание и удаление файлов и директорий.

**11.2.** Привести образ диска с точки зрения состава и размещения всех ФС на испытуемом компьютере, а также образ полного дерева ФС, включая присоединенные ФС съемных и несъемных носителей. Проанализировать и указать формат таблицы монтирования.

mount выводит список текущих смонтированных файловых систем (название, точка монтирования, тип, опции монтирования, например, rw - режим доступа чтение/запись, nosuid - запрещает SUID, noexec - запрещает исполнение любых двоичных файлов фс, relatime-обновляет время доступа к файлу только в случае его редактирования).

(base) katya@katya:~$ mount

sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)

proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)

udev on /dev type devtmpfs (rw,nosuid,relatime,size=7837200k,nr\_inodes=1959300,mode=755,inode64)

devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)

tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1575100k,mode=755,inode64)

/dev/nvme0n1p6 on / type ext4 (rw,relatime,errors=remount-ro)

securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)

tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)

tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)

cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory\_recursiveprot)

pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)

efivarfs on /sys/firmware/efi/efivars type efivarfs (rw,nosuid,nodev,noexec,relatime)

bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)

systemd-1 on /proc/sys/fs/binfmt\_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe\_ino=26655)

hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)

mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)

debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)

tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)

fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)

configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)

ramfs on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)

/var/lib/snapd/snaps/bare\_5.snap on /snap/bare/5 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/clion\_265.snap on /snap/clion/265 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/code\_155.snap on /snap/code/155 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/code\_156.snap on /snap/code/156 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core18\_2796.snap on /snap/core18/2796 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core18\_2812.snap on /snap/core18/2812 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core20\_2105.snap on /snap/core20/2105 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core20\_2182.snap on /snap/core20/2182 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core22\_1033.snap on /snap/core22/1033 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/core22\_1122.snap on /snap/core22/1122 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/curl\_1754.snap on /snap/curl/1754 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/discord\_181.snap on /snap/discord/181 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/discord\_182.snap on /snap/discord/182 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/firefox\_3836.snap on /snap/firefox/3836 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/firefox\_4033.snap on /snap/firefox/4033 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/gnome-3-38-2004\_143.snap on /snap/gnome-3-38-2004/143 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/gnome-42-2204\_141.snap on /snap/gnome-42-2204/141 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/gnome-42-2204\_172.snap on /snap/gnome-42-2204/172 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/gtk-common-themes\_1535.snap on /snap/gtk-common-themes/1535 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/pycharm-community\_374.snap on /snap/pycharm-community/374 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/pycharm-community\_378.snap on /snap/pycharm-community/378 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/snap-store\_1113.snap on /snap/snap-store/1113 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/snap-store\_959.snap on /snap/snap-store/959 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/snapd\_21184.snap on /snap/snapd/21184 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/snapd\_20671.snap on /snap/snapd/20671 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/snapd-desktop-integration\_83.snap on /snap/snapd-desktop-integration/83 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/telegram-desktop\_5741.snap on /snap/telegram-desktop/5741 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/dev/nvme0n1p6 on /var/snap/firefox/common/host-hunspell type ext4 (ro,noexec,noatime,errors=remount-ro)

/var/lib/snapd/snaps/zoom-client\_218.snap on /snap/zoom-client/218 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/zoom-client\_225.snap on /snap/zoom-client/225 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/dev/nvme0n1p7 on /home type ext4 (rw,relatime)

/dev/nvme0n1p2 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)

binfmt\_misc on /proc/sys/fs/binfmt\_misc type binfmt\_misc (rw,nosuid,nodev,noexec,relatime)

tmpfs on /run/snapd/ns type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1575100k,mode=755,inode64)

nsfs on /run/snapd/ns/snapd-desktop-integration.mnt type nsfs (rw)

tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=1575096k,nr\_inodes=393774,mode=700,uid=1000,gid=1000,inode64)

gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user\_id=1000,group\_id=1000)

portal on /run/user/1000/doc type fuse.portal (rw,nosuid,nodev,relatime,user\_id=1000,group\_id=1000)

nsfs on /run/snapd/ns/snap-store.mnt type nsfs (rw)

/var/lib/snapd/snaps/snapd-desktop-integration\_157.snap on /snap/snapd-desktop-integration/157 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

/var/lib/snapd/snaps/telegram-desktop\_5767.snap on /snap/telegram-desktop/5767 type squashfs (ro,nodev,relatime,errors=continue,threads=single,x-gdu.hide)

nsfs on /run/snapd/ns/telegram-desktop.mnt type nsfs (rw)

tmpfs on /run/user/1001 type tmpfs (rw,nosuid,nodev,relatime,size=1575096k,nr\_inodes=393774,mode=700,uid=1001,gid=1001,inode64)

gvfsd-fuse on /run/user/1001/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user\_id=1001,group\_id=1001)

portal on /run/user/1001/doc type fuse.portal (rw,nosuid,nodev,relatime,user\_id=1001,group\_id=1001)

tmpfs on /run/user/128 type tmpfs (rw,nosuid,nodev,relatime,size=1575096k,nr\_inodes=393774,mode=700,uid=128,gid=134,inode64)

gvfsd-fuse on /run/user/128/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user\_id=128,group\_id=134)

portal on /run/user/128/doc type fuse.portal (rw,nosuid,nodev,relatime,user\_id=128,group\_id=134)

После подключения флешки была добавлена строка:

/dev/sda1 on /media/katya/UBUNTU 22\_0 type vfat (rw,nosuid,nodev,relatime,uid=1000,gid=1000,fmask=0022,dmask=0022,codepage=437,iocharset=iso8859-1,shortname=mixed,showexec,utf8,flush,errors=remount-ro,uhelper=udisks2)

Команда fdisk -l выводит информацию о всех дисках:

(base) katya@katya:~$ sudo fdisk -l

[sudo] пароль для katya:

Диск /dev/loop0: 4 KiB, 4096 байт, 8 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop1: 946,18 MiB, 992145408 байт, 1937784 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop2: 308,55 MiB, 323534848 байт, 631904 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop3: 310,8 MiB, 325898240 байт, 636520 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop4: 55,66 MiB, 58363904 байт, 113992 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop5: 55,66 MiB, 58363904 байт, 113992 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop6: 63,91 MiB, 67014656 байт, 130888 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop7: 63,91 MiB, 67010560 байт, 130880 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/nvme0n1: 476,94 GiB, 512110190592 байт, 1000215216 секторов

Disk model: INTEL SSDPEKNU512GZ

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Тип метки диска: gpt

Идентификатор диска: 0AEEF433-2A11-4ACF-8894-B6F1822ED34F

Устр-во начало Конец Секторы Размер Тип

/dev/nvme0n1p1 2048 1085439 1083392 529M Среда для восст

/dev/nvme0n1p2 1085440 1290239 204800 100M EFI

/dev/nvme0n1p3 1290240 1323007 32768 16M Зарезервированн

/dev/nvme0n1p4 1323008 342695935 341372928 162,8G Microsoft basic

/dev/nvme0n1p5 716804096 1000214527 283410432 135,1G Microsoft basic

/dev/nvme0n1p6 342695936 525694975 182999040 87,3G Файловая систем

/dev/nvme0n1p7 525694976 716804095 191109120 91,1G Файловая систем

Элементы таблицы разделов упорядочены не так, как на диске.

Диск /dev/loop8: 74,11 MiB, 77713408 байт, 151784 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop9: 74,21 MiB, 77819904 байт, 151992 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop10: 6,41 MiB, 6725632 байт, 13136 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop11: 97,71 MiB, 102453248 байт, 200104 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop12: 106,51 MiB, 111681536 байт, 218128 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop13: 266,63 MiB, 279584768 байт, 546064 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop14: 268,25 MiB, 281276416 байт, 549368 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop15: 349,7 MiB, 366682112 байт, 716176 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop16: 496,98 MiB, 521121792 байт, 1017816 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop17: 504,15 MiB, 528642048 байт, 1032504 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop18: 91,69 MiB, 96141312 байт, 187776 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop19: 643,61 MiB, 674877440 байт, 1318120 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop20: 643,61 MiB, 674877440 байт, 1318120 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop21: 12,93 MiB, 13553664 байт, 26472 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop22: 12,32 MiB, 12922880 байт, 25240 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop23: 40,43 MiB, 42393600 байт, 82800 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop24: 39,1 MiB, 40996864 байт, 80072 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop25: 452 KiB, 462848 байт, 904 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop27: 417,66 MiB, 437952512 байт, 855376 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop28: 376,67 MiB, 394969088 байт, 771424 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop29: 366,6 MiB, 384409600 байт, 750800 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop30: 476 KiB, 487424 байт, 952 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Диск /dev/loop31: 417,66 MiB, 437948416 байт, 855368 секторов

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

После подключения флешки:

Диск /dev/sda: 57,73 GiB, 61991813632 байт, 121077761 секторов

Disk model: DataTraveler 3.0

Единицы: секторов по 1 \* 512 = 512 байт

Размер сектора (логический/физический): 512 байт / 512 байт

Размер I/O (минимальный/оптимальный): 512 байт / 512 байт

Тип метки диска: dos

Идентификатор диска: 0x000fce75

Устр-во Загрузочный начало Конец Секторы Размер Идентификатор Тип

/dev/sda1 \* 2048 121077760 121075713 57,7G c W95

**11.3.** Привести «максимально возможное» дерево ФС, проанализировать, где это указывается

Максимально возможное дерево файловой системы (ФС) в контексте Linux и Unix-подобных операционных систем обычно относится к структуре каталогов и файлов, которая может быть создана внутри файловой системы. Это дерево начинается с корневого каталога (/) и расширяется вниз через подкаталоги и файлы, которые могут быть созданы пользователями и системными процессами.

Ограничения на максимально возможное дерево ФС:

* Ограничение на глубину: В Linux и Unix-подобных системах существует ограничение на максимальную глубину вложенности каталогов, которое определяется параметром fs.inotify.max\_user\_watches в ядре. Это ограничение может быть изменено, но изменение этого параметра может повлиять на производительность системы.
* Ограничение на количество файлов: Каждый файл и каталог в файловой системе имеет уникальный идентификатор (inode), и общее количество inode в файловой системе ограничено. Это ограничение зависит от типа файловой системы и ее размера.

Файл /usr/include/linux/limits.h содержит определения максимальных значений для различных системных ограничений в Linux.

(base) katya@katya:~$ cat /usr/include/linux/limits.h

/\* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note \*/

#ifndef \_LINUX\_LIMITS\_H

#define \_LINUX\_LIMITS\_H

#define NR\_OPEN 1024

#define NGROUPS\_MAX 65536 /\* supplemental group IDs are available \*/

#define ARG\_MAX 131072 /\* # bytes of args + environ for exec() \*/

#define LINK\_MAX 127 /\* # links a file may have \*/

#define MAX\_CANON 255 /\* size of the canonical input queue \*/

#define MAX\_INPUT 255 /\* size of the type-ahead buffer \*/

#define NAME\_MAX 255 /\* # chars in a file name \*/

#define PATH\_MAX 4096 /\* # chars in a path name including nul \*/

#define PIPE\_BUF 4096 /\* # bytes in atomic write to a pipe \*/

#define XATTR\_NAME\_MAX 255 /\* # chars in an extended attribute name \*/

#define XATTR\_SIZE\_MAX 65536 /\* size of an extended attribute value (64k) \*/

#define XATTR\_LIST\_MAX 65536 /\* size of extended attribute namelist (64k) \*/

#define RTSIG\_MAX 32

#endif

Таким образом, наибольшее количество символов пути файла равняется 4096. Так как имя каталога содержит как минимум два символа (/ + имя), то максимальный уровень вложенности равен 2047 директорий ((2^12) / 2 -1).

**12**. Проанализировать и пояснить принцип работы утилиты ***file***.

**12.1.** Привести алгоритм её функционирования на основе информационной базы, размещение и полное имя которой указывается в описании утилиты в технической документации ОС (как правило, ***/usr/share/file/magic.\****), а также содержимого заголовка файла, к которому применяется утилита. Определить, где находятся магические числа и иные характеристики, идентифицирующие тип файла, применительно к *исполняемым* файлам, а также файлам других типов.

Утилита file в Unix и Unix-подобных операционных системах, включая Linux, используется для определения типа данных, содержащихся в файле, основываясь на магических числах и других характеристиках, идентифицирующих тип файла. Эти магические числа — это уникальные последовательности байтов в начале файла, которые позволяют системе различать и распознавать различные типы файлов без необходимости знать их расширение.

Для определения типа файла выполняются разные тесты:

1. Filesystem test основан на анализе кода возврата системного вызова stat(). Файл проверяется на пустоту и принадлежность к одному из типов специальных файлов. Известные типы файлов распознаются, если они определены в системном файле /usr/include/x86\_64-linux-gnu/sys/stat.h
2. Magic number test используется для проверки файлов, данные которых записаны в определенном формате. В начале таких файлов записано магическое число, которое помогает ОС определить тип файла. Все известные ОС магические числа по умолчанию хранятся в /usr/share/file/magic.
3. Language test используется для анализа языка, на котором написан файл, если этот файл в формате ASCII: выполняется поиск стандартных строк, которые могут соответствовать определенному языку.

Таким образом типы файлов можно разделить на 3 группы: текстовые (файл содержит только ASCII символы и может быть прочитан на терминале), исполняемые (файл содержит результаты компилирования программы в форме понятной ядру ОС) и данные (все, что не подходит под первые две группы)

**12.2.** Утилиту ***file*** выполнить с разными ключами.

1. вызов команды без ключа: определение типа файла

(base) katya@katya:~/os/lb2$ file task10.txt

task10.txt: ASCII text

(base) katya@katya:~/os/lb2$ file task10.out

task10.out: setuid, setgid ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=57ad18a1611ae2c7d18cb3c4b4cd9e292aba23a5, for GNU/Linux 3.2.0, not stripped

(base) katya@katya:~/os/lb2$ file find.sh

find.sh: Bourne-Again shell script, Unicode text, UTF-8 text executable

(base) katya@katya:~/os/lb2$ file 7

7: directory

(base) katya@katya:~/os/lb2$ file prog.c

prog.c: symbolic link to /home/katya/os/lb1/prog.c

1. file \* - отображение типов всех файлов в данном каталоге

7: directory

7task.sh: Bourne-Again shell script, Unicode text, UTF-8 text executable

a.out: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=9bbf10a67443902a065ac730caffc18022a31bac, for GNU/Linux 3.2.0, not stripped

find.sh: Bourne-Again shell script, Unicode text, UTF-8 text executable

find\_types.sh: Bourne-Again shell script, ASCII text executable

hard\_links.sh: Bourne-Again shell script, ASCII text executable

link2.c: symbolic link to /home/katya/os/lb1/prog.c

links.sh: Bourne-Again shell script, Unicode text, UTF-8 text executable

links.txt: Unicode text, UTF-8 text

logfile: Unicode text, UTF-8 text

num1.py: symbolic link to /home/katya/piaa/2лаб/num1.py

prog.c: symbolic link to /home/katya/os/lb1/prog.c

task10.c: C source, ASCII text

task10.out: setuid, setgid ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=57ad18a1611ae2c7d18cb3c4b4cd9e292aba23a5, for GNU/Linux 3.2.0, not stripped

task10.txt: ASCII text

task6.txt: empty

task9\_1: directory

task9\_2: directory

task9\_3: directory

text: directory

1. -L - отображение типа файла по ссылке

(base) katya@katya:~/os/lb2$ file -L link2.c

link2.c: C source, ASCII text

**12.3.** Привести экспериментальную попытку с добавлением в базу собственного типа файла и его дальнейшей идентификацией. Описать эксперимент и привести последовательность действий для расширения функциональности утилиты ***file*** и возможности встраивания дополнительного типа файла в ФС (согласовать содержимое информационной базы и заголовка файла нового типа

Был создан файл task12.sots с содержимым “123”, добавлена строчка “0 string 123 katya” в файл magic” (123 - магическое число, katya - тип файла)

#редактирование файла magic

(base) katya@katya:~$ sudo nano /etc/magic

#определение типа

(base) katya@katya:~$ cd os/lb2

(base) katya@katya:~/os/lb2$ file task12.sots

task12.sots: katya

(base) katya@katya:~/os/lb2$ cat task12.sots

123

Таким образом новый тип файла был успешно добавлен.

## **Вывод.**

В ходе лабораторной работы было проанализировано функциональное назначение структурных элементов дерева ФС. Определить размещение корневого каталога (корневой ФС).

Для выполнения работы были изучены типы файлов, утилиты find, od, \*dump, df, file и другие.

В результате выполнения заданий было написано несколько bash-скриптов и добавлен собственный тип файлов.

## **Список литературы.**

1. “Системное программное обеспечение. Межроцессные взаимодействия в операционных системах” Душутина Е.В.
2. "Linux File Systems" by Moshe Bar - книга, посвященная различным файловым системам в Linux.
3. "Linux man pages" - официальная документация, содержащая описание команд и системных вызовов в Linux, включая команды для работы с файловыми системами.