МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»**

**(БГТУ им. В. Г. Шухова)**



Кафедра программного обеспечения вычислительной

техники и автоматизированных систем

**Лабораторная работа №3**

по дисциплине: «Операционные системы»

на тему: «Файловые системы в ОС Linux (Ubuntu): сравнение, области эффективности. Виртуальная файловая система. Пользовательская файловая система»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверили:

доц. Островский Алексей Мичеславович,

асс. Четвертухин Виктор Романович

Белгород, 2024

**Цель работы:** Изучить популярные файловые системы в ОС Linux (ext4, Btrfs, ReiserFS, NTFS, FAT32), определить область эффективности каждой их них, разобраться как осуществляется работа с виртуальной файловой системой (VFS) ОС Linux и выполнить разработку пользовательской файловой системы в соответствии с индивидуальным заданием.

**Условие индивидуального задания:**

Сортировка внешней памяти. Сортировка больших файлов, которые не помещаются в оперативной памяти. Алгоритм: polyphase merge sort.

**Ход выполнения работы**

**Задание 1**

Создаём виртуальный жесткий диск объемом 10 ГБ.

dd if=/dev/zero of=my\_virtual\_disk\_os\_lab3.img bs=1M count=10240

$ (bash ./create\_disk.sh)

10240+0 records in

10240+0 records out

10737418240 bytes (11 GB, 10 GiB) copied, 5.30419 s, 2.0 GB/s

Подключаем my\_virtual\_disk\_os\_lab3.img как виртуальное устройство с использованием losetup:

$ sudo losetup -fP my\_virtual\_disk\_os\_lab3.img

Проверяем, какому loopback - устройству был присвоен диск:

$ losetup -a

/dev/loop16: []: (/home/user/my\_virtual\_disk\_os\_lab3.img)

Команда bash для разбиения диска на 5 разделов для файловых систем ext4, Btrfs, ReiserFS, NTFS, FAT32:

$ sudo parted /dev/loop16 mklabel gpt

sudo parted /dev/loop16 mklabel msdos

sudo parted -a optimal /dev/loop16 mkpart primary ext4 0% 2G

sudo parted -a optimal /dev/loop16 mkpart primary btrfs 2G 4G

sudo parted -a optimal /dev/loop16 mkpart primary reiserfs 4G 6G

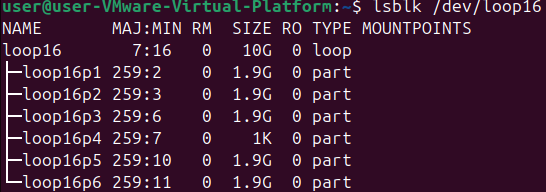
sudo parted -a optimal /dev/loop16 mkpart extended 6G 10G

sudo parted -a optimal /dev/loop16 mkpart logical ntfs 6G 8G

sudo parted -a optimal /dev/loop16 mkpart logical fat32 8G 10G

$ (bash ./parted\_prepare.sh)

$ lsblk /dev/loop16



Установаем соответствующие пакеты, которые содержат инструменты для создания файловых систем Btrfs и ReiserFS.

sudo apt update

sudo apt install btrfs-progs

sudo apt install reiserfsprogs

$(bash ./install\_new\_fs.sh)

Создание файловых систем на разделах, форматирование:

sudo mkfs.ext4 /dev/loop16p1

sudo mkfs.btrfs /dev/loop16p2

sudo mkfs.reiserfs /dev/loop16p3

sudo mkfs.ntfs /dev/loop16p5

sudo mkfs.vfat /dev/loop16p6

$ (bash ./mkfs\_prepare.sh)

Создание директорий для каждого раздела для подготовки к монтированию:

$ sudo mkdir -p /mnt/ext4 /mnt/btrfs /mnt/reiserfs /mnt/ntfs /mnt/fat32

Монтирование:

Система FAT32 не поддерживает права доступа POSIX.

Монтирование FAT32 с разрешением записи и чтения:

$ sudo mount -t vfat /dev/loop16p5 /mnt/fat32 -o uid=$(id -u),gid=$(id -g),umask=0022

sudo mount /dev/loop16p1 /mnt/ext4

sudo mount /dev/loop16p2 /mnt/btrfs

sudo mount /dev/loop16p3 /mnt/reiserfs

sudo mount -t ntfs /dev/loop16p5 /mnt/ntfs

sudo mount -t vfat /dev/loop16p6 /mnt/fat32

$ (bash ./mount\_disks.sh)

Проверка монтирования:

$ lsblk -f /dev/loop16



# Назначить владельца текущему пользователю

sudo chown -R $USER:$USER /mnt/ext4

sudo chown -R $USER:$USER /mnt/btrfs

sudo chown -R $USER:$USER /mnt/reiserfs

sudo chown -R $USER:$USER /mnt/ntfs

# Установить права доступа для чтения, записи и выполнения

sudo chmod -R 755 /mnt/ext4

sudo chmod -R 755 /mnt/btrfs

sudo chmod -R 755 /mnt/reiserfs

sudo chmod -R 755 /mnt/ntfs

$ (bash ./rules\_set.sh)

Установка необходимых пакетов для визуализации:

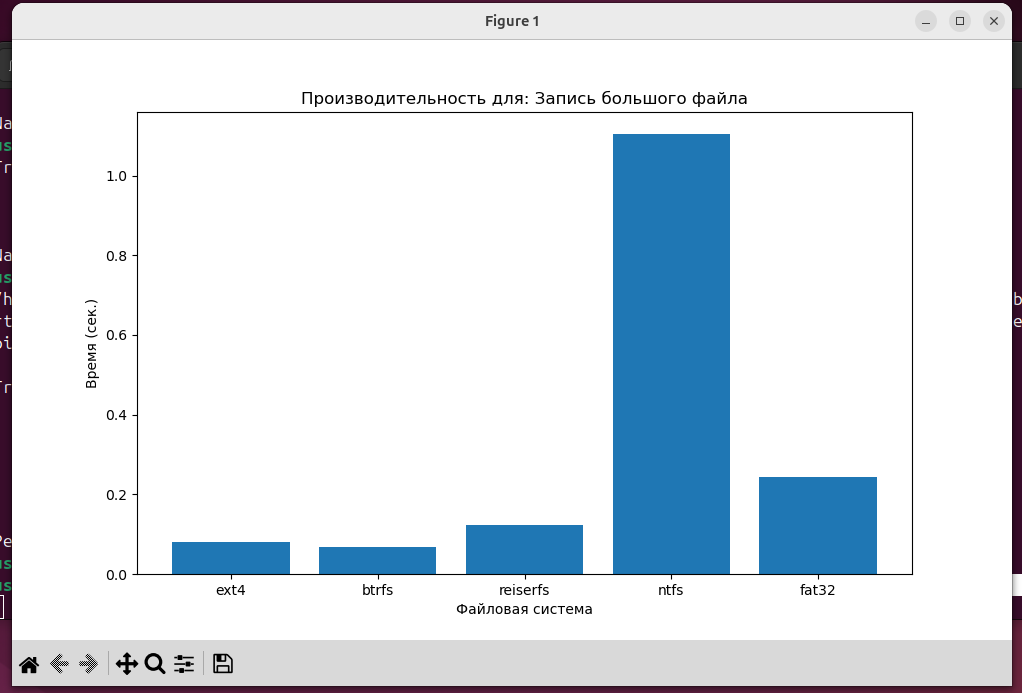
$ sudo apt update && sudo apt install python3-pip -y

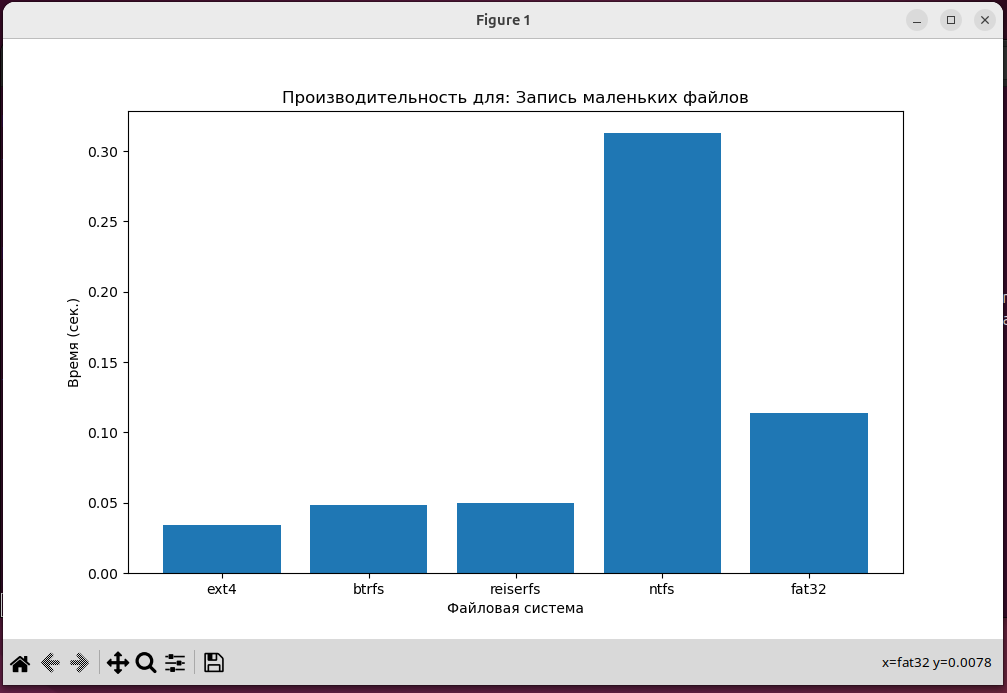
$ python3 -m pip install matplotlib --break-system-packages

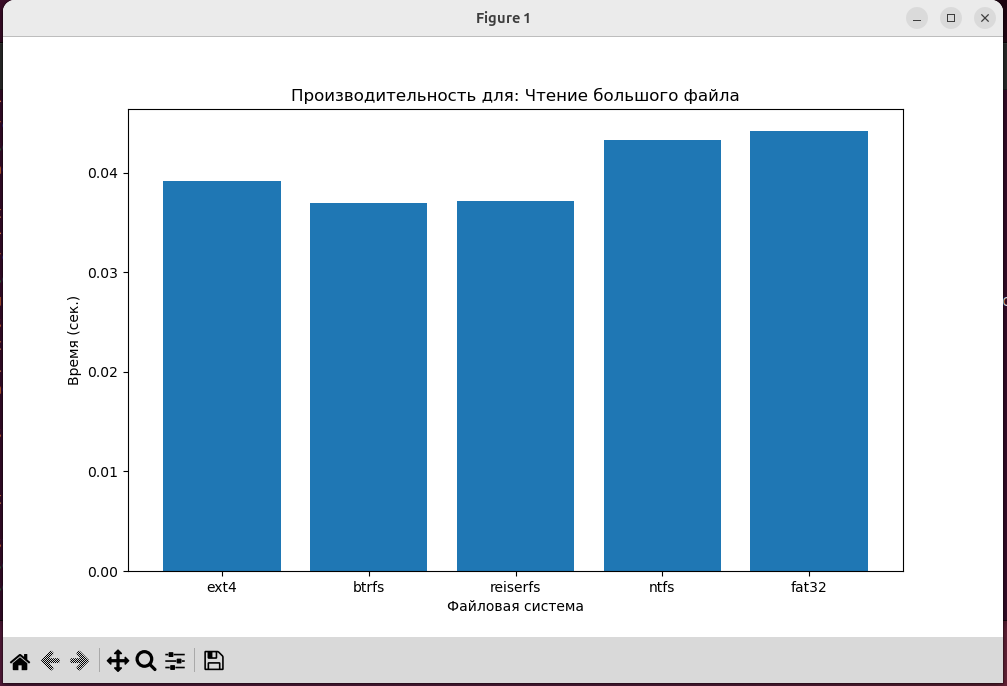
$ sudo apt install python3-pil python3-pil.imagetk python3-tk -y

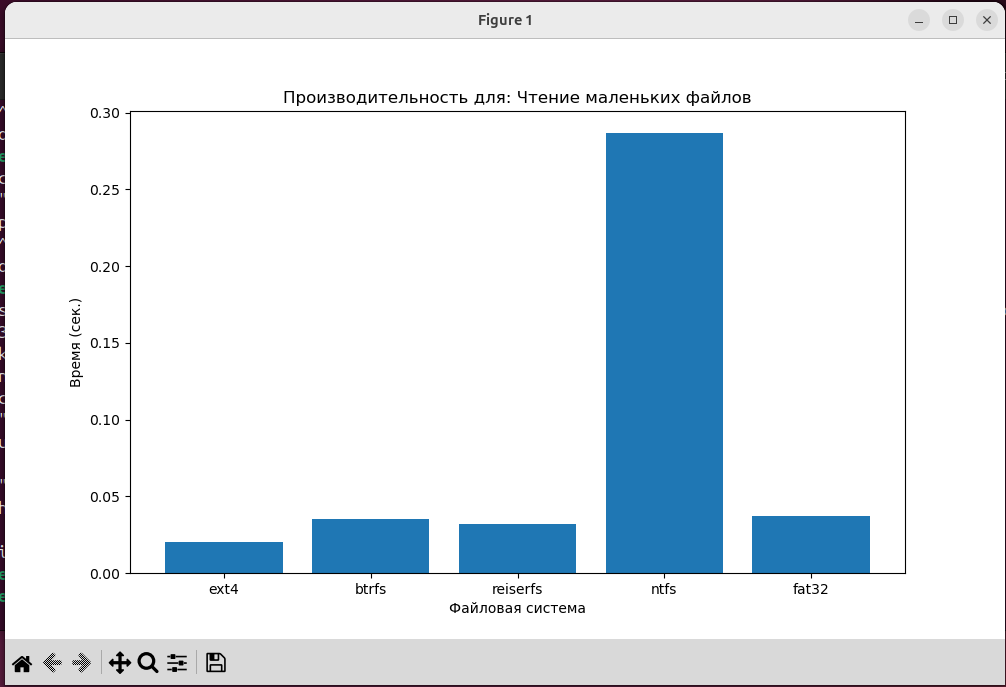
Замеры с использованием прилагаемого кода:

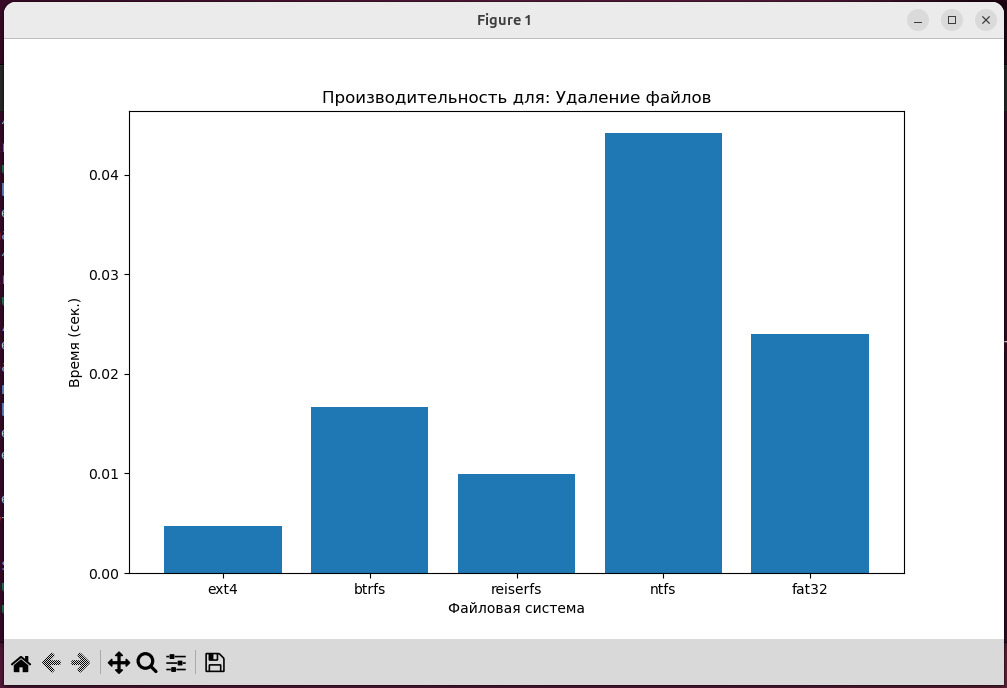
$ sudo python3 stats\_prepare.py











Отключение разделов (после завершения всех тестов):

sudo umount /mnt/ext4

sudo umount /mnt/btrfs

sudo umount /mnt/reiserfs

sudo umount /mnt/ntfs

sudo umount /mnt/fat32

$ (bash ./umount\_disks.sh)

Отключение loopback-устройства (после завершения всех тестов):

$ sudo losetup -d /dev/loop16

**Задание 2**

**Код программы индивидуального задания:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 1024

#define MEMORY\_LIMIT 2500 // Ограничение памяти

// Функция для чтения строк из файла и записи их в массив

int read\_lines(FILE \*file, char \*\*lines, int max\_lines) {

int count = 0;

char buffer[MAX\_LINE\_LENGTH];

while (fgets(buffer, MAX\_LINE\_LENGTH, file) != NULL && count < max\_lines) {

lines[count] = strdup(buffer);

count++;

}

return count;

}

// Функция для сортировки массива строк

void sort\_lines(char \*\*lines, int count) {

for (int i = 0; i < count - 1; i++) {

for (int j = i + 1; j < count; j++) {

if (strcmp(lines[i], lines[j]) > 0) {

char \*temp = lines[i];

lines[i] = lines[j];

lines[j] = temp;

}

}

}

}

// Функция для записи отсортированных строк в файл

void write\_lines(FILE \*file, char \*\*lines, int count) {

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

fputs(lines[i], file);

free(lines[i]);

}

}

// Функция для разделения файла на части и сортировки каждой части

int split\_and\_sort(const char \*input\_filename, const char \*output\_prefix) {

FILE \*input\_file = fopen(input\_filename, "r");

if (!input\_file) {

perror("Ошибка открытия входного файла");

exit(1);

}

char \*\*lines = malloc(MEMORY\_LIMIT \* sizeof(char \*));

int part\_number = 0;

while (1) {

int count = read\_lines(input\_file, lines, MEMORY\_LIMIT);

if (count == 0) break;

sort\_lines(lines, count);

char output\_filename[1024];

sprintf(output\_filename, "%s%d.txt", output\_prefix, part\_number++);

FILE \*output\_file = fopen(output\_filename, "w");

if (!output\_file) {

perror("Ошибка создания временного файла");

exit(1);

}

write\_lines(output\_file, lines, count);

fclose(output\_file);

}

free(lines);

fclose(input\_file);

return part\_number;

}

// Функция для слияния отсортированных файлов (polyphase merge sort)

void polyphase\_merge(const char \*output\_prefix, int part\_count, const char \*output\_filename) {

FILE \*output\_file = fopen(output\_filename, "w");

if (!output\_file) {

perror("Ошибка создания выходного файла");

exit(1);

}

FILE \*\*input\_files = malloc(part\_count \* sizeof(FILE \*));

for (int i = 0; i < part\_count; i++) {

char filename[1024];

sprintf(filename, "%s%d.txt", output\_prefix, i);

input\_files[i] = fopen(filename, "r");

if (!input\_files[i]) {

perror("Ошибка открытия временного файла");

exit(1);

}

}

char \*\*buffers = malloc(part\_count \* sizeof(char \*));

int \*active = malloc(part\_count \* sizeof(int));

for (int i = 0; i < part\_count; i++) {

buffers[i] = malloc(MAX\_LINE\_LENGTH);

if (fgets(buffers[i], MAX\_LINE\_LENGTH, input\_files[i])) {

active[i] = 1;

} else {

active[i] = 0;

}

}

while (1) {

int min\_index = -1;

for (int i = 0; i < part\_count; i++) {

if (active[i] && (min\_index == -1 || strcmp(buffers[i], buffers[min\_index]) < 0)) {

min\_index = i;

}

}

if (min\_index == -1) break;

fputs(buffers[min\_index], output\_file);

if (!fgets(buffers[min\_index], MAX\_LINE\_LENGTH, input\_files[min\_index])) {

active[min\_index] = 0;

}

}

for (int i = 0; i < part\_count; i++) {

free(buffers[i]);

fclose(input\_files[i]);

}

free(buffers);

free(active);

free(input\_files);

fclose(output\_file);

}

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

if (argc != 4) {

fprintf(stderr, "Использование: %s <входной\_файл> <выходной\_файл> <папка для временных файлов>\n", argv[0]);

return 1;

}

const char \*input\_filename = argv[1];

const char \*output\_filename = argv[2];

char output\_prefix[60];

strcpy(output\_prefix, argv[3]);

strcat(output\_prefix, "sorted\_part\_");

int part\_count = split\_and\_sort(input\_filename, output\_prefix);

polyphase\_merge(output\_prefix, part\_count, output\_filename);

for (int i = 0; i < part\_count; i++) {

char filename[1024];

sprintf(filename, "%s%d.txt", output\_prefix, i);

remove(filename);

}

return 0;

}

**Изменённый код программы для вывода графиков:**

import os

import time

import matplotlib

import matplotlib.pyplot as plt

from pathlib import Path

import subprocess

MOUNT\_POINTS = {

"ext4": "/mnt/ext4",

"btrfs": "/mnt/btrfs",

"reiserfs": "/mnt/reiserfs",

"ntfs": "/mnt/ntfs",

"fat32": "/mnt/fat32",

}

matplotlib.use('TkAgg')

LARGE\_FILE\_SIZE\_MB = 100

SMALL\_FILES\_COUNT = 1000

SMALL\_FILE\_SIZE\_KB = 10

def run\_sort\_program(input\_file, output\_file, temp\_dir, mount\_point):

start\_time = time.time()

subprocess.run([

"/home/user/os\_lab3", input\_file, output\_file, temp\_dir

], cwd=mount\_point, check=True)

return time.time() - start\_time

def generate\_test\_file(directory, size\_mb):

filepath = os.path.join(directory, "input.txt")

with open(filepath, 'w') as f:

for \_ in range(size\_mb \* 1024):

f.write("Lorem ipsum dolor sit amet, consectetur adipiscing elit.\n")

return filepath

results = {fs: {} for fs in MOUNT\_POINTS.keys()}

# Compile the sorting program

subprocess.run(["gcc", "-o", "os\_lab3", "os\_lab3.c"], check=True)

for fs, path in MOUNT\_POINTS.items():

os.makedirs(path, exist\_ok=True)

# Prepare test files

input\_file = generate\_test\_file(path, LARGE\_FILE\_SIZE\_MB)

output\_file = os.path.join(path, "output.txt")

temp\_dir = os.path.join(path, "temp\_files/")

os.makedirs(temp\_dir, exist\_ok=True)

# Run and measure the sorting program

results[fs]['Скорость сортировки'] = run\_sort\_program(input\_file, output\_file, temp\_dir, path)

# Clean up

os.remove(input\_file)

os.remove(output\_file)

for temp\_file in Path(temp\_dir).glob("\*"):

os.remove(temp\_file)

os.rmdir(temp\_dir)

# Plot results

plt.figure(figsize=(10, 6))

times = [results[fs]['Скорость сортировки'] for fs in MOUNT\_POINTS.keys()]

plt.bar(MOUNT\_POINTS.keys(), times)

plt.title("Производительность сортировки внешней памяти")

plt.xlabel("Файловая система")

plt.ylabel("Время (сек.)")

plt.show()

**Протоколы, логи, скриншоты, графики.**

**Основная идея работы алгоритма:**

Алгоритм сортировки внешней памяти основан на принципе polyphase merge sort. Его ключевая идея заключается в следующем:

1. Из входного файла выделяется максимально возможное количество строк, которое укладывается в заданное ограничение по памяти. Эти строки сортируются и записываются в несколько временных файлов.
2. После этого запускается процесс слияния: для каждого временного файла открывается поток чтения, из которых извлекается минимальная строка. Эта строка записывается в результирующий файл. Процесс повторяется, пока все строки не будут обработаны.

**Условия тестирования:**

Для проверки производительности алгоритма был использован тестовый файл:

* Объём файла: 500 тысяч строк.
* Размер каждой строки: 10 символов.
* Ограничение памяти: 1000 строк.

В процессе выполнения алгоритма:

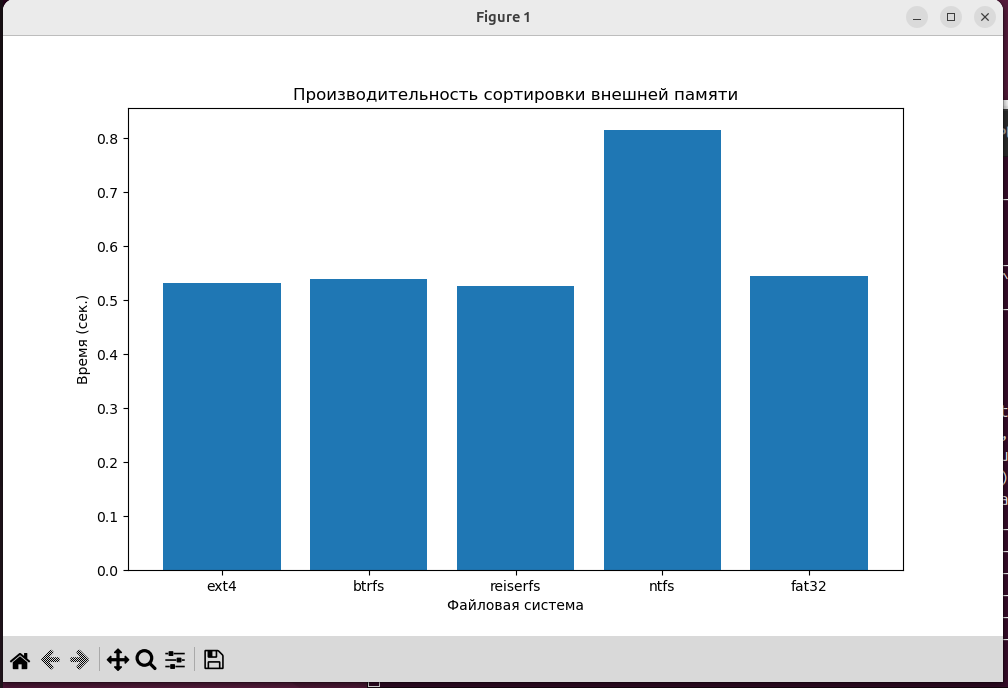
* Исходный файл был разделён на 500 временных файлов, каждый размером 12.3 kB.
* После завершения сортировки все временные файлы удаляются.

**Результаты:**

Производительность алгоритма тестировалась на разных файловых системах: ext4, btrfs, ntfs, fat32, и reiserfs. Замеры включали:

* Скорость выполнения сортировки для каждого файлового типа.
* Влияние размера временных файлов и количества строк на общую производительность.

**График производительности:**



**Логи выполнения:**

Во время выполнения алгоритма были собраны время выполнения полного цикла сортировки каждой файловой системы.

NTFS в данном тесте, вероятно, показала лучшую производительность благодаря оптимизации для последовательной обработки данных, эффективной работе с временными файлами. Однако это не обязательно означает, что NTFS всегда быстрее — в других сценариях (например, с большими файлами или многопоточными операциями) ext4 или btrfs могут показать лучшие результаты.

**Вывод:** на этой лабораторной работе провели тестирование файловых систем в ОС Linux (ext4, Btrfs, ReiserFS, NTFS, FAT32).