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

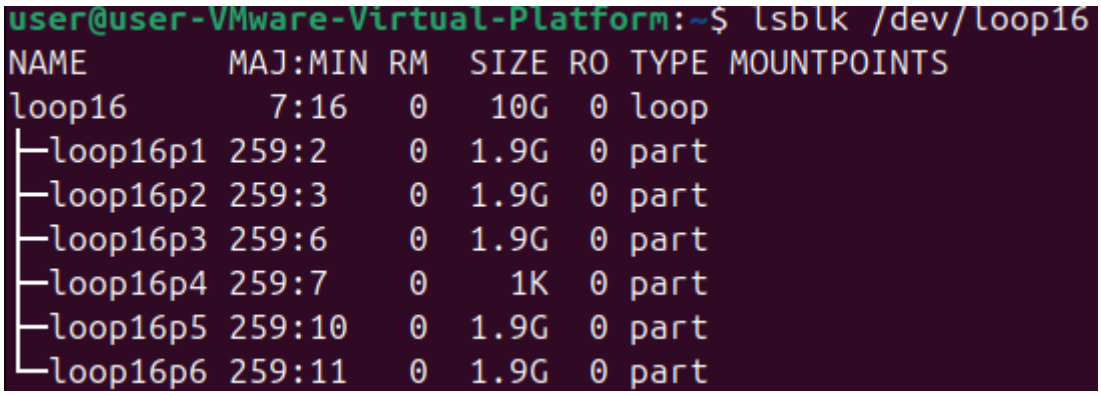
sudo parted -a optimal /dev/loop16 mkpart extended 6G 10G

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

```
user@user-VMware-Virtual-Platform:~$ lsblk -f /dev/loop16
NAME       FSTYPE   FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
loop16
├─loop16p1 ext4     1.0         ee20f4fe-276f-49d8-9a3a-7050d712cc04    1.7G     0% /mnt/ext4
├─loop16p2 btrfs                153f9790-194e-41fc-93c8-1504eb06694f    1.7G     0% /mnt/btrfs
├─loop16p3 reiserfs 3.6         eec3cf3d-0ebf-49d3-b716-75c7ba591f28    1.8G     2% /mnt/reiserfs
├─loop16p4
├─loop16p5 ntfs                 159A6B7A7B9B5FDC                        1.9G     1% /mnt/ntfs
└─loop16p6 vfat     FAT32       B57D-DB18                               1.9G     0% /mnt/fat32
```

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:

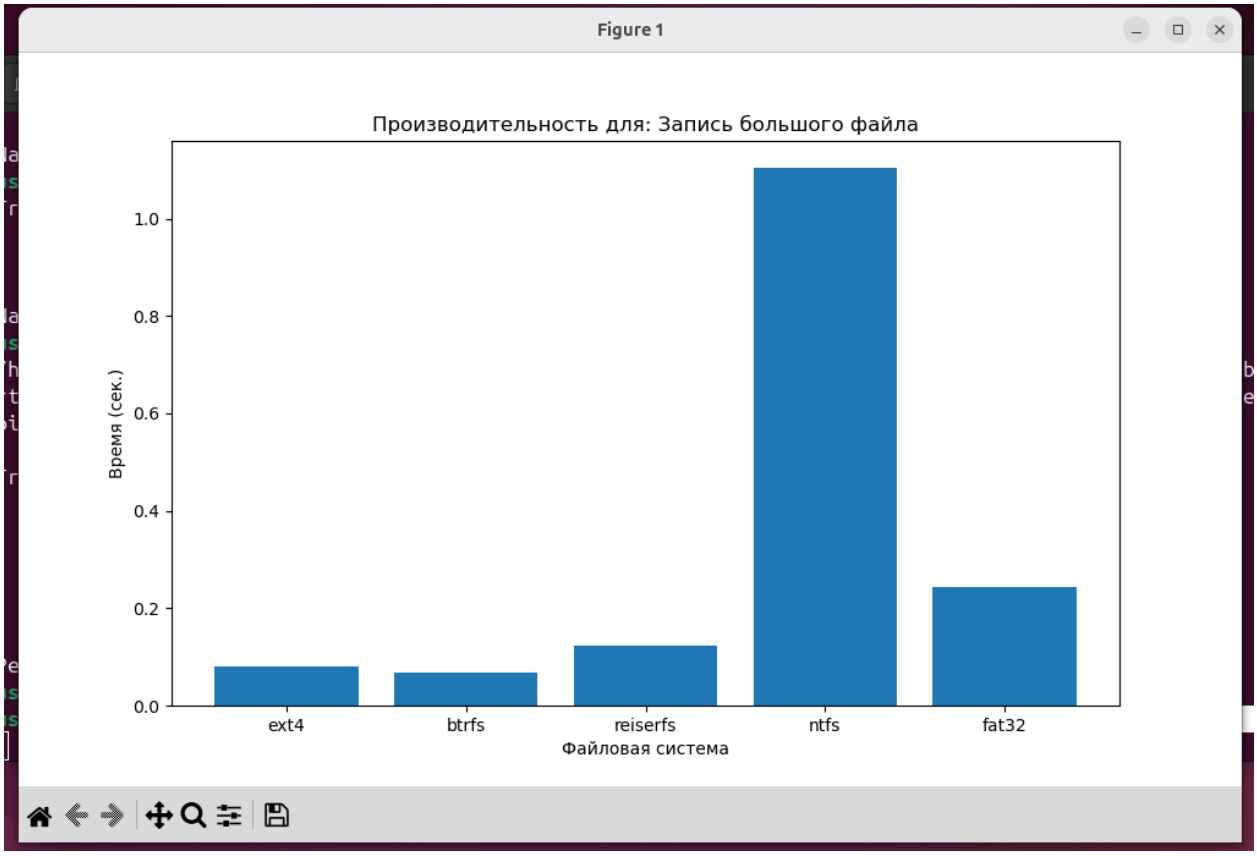
$ sudo apt update && sudo apt install python3-pip -y

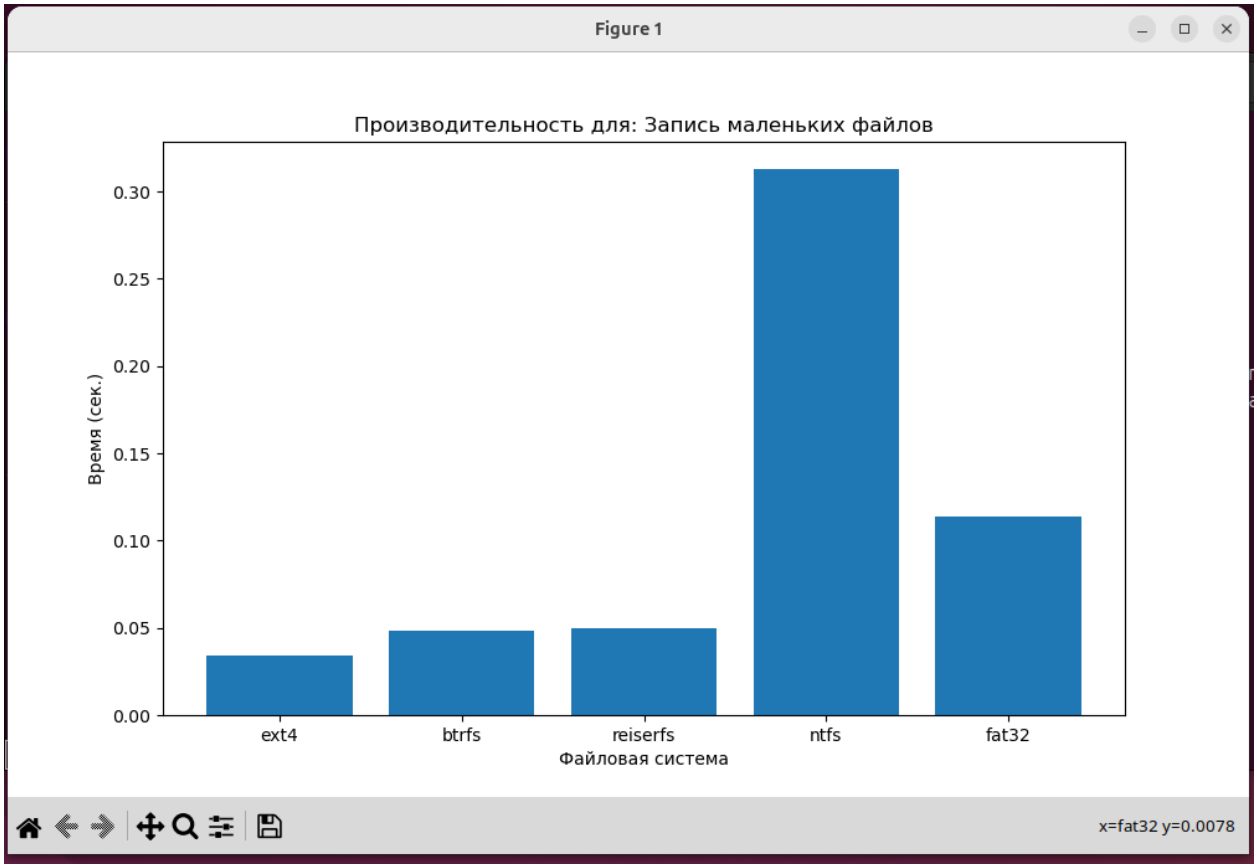
$ python3 -m pip install matplotlib --break-system-packages

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

Замеры с использованием прилагаемого кода:

$ sudo python3 stats_prepare.py





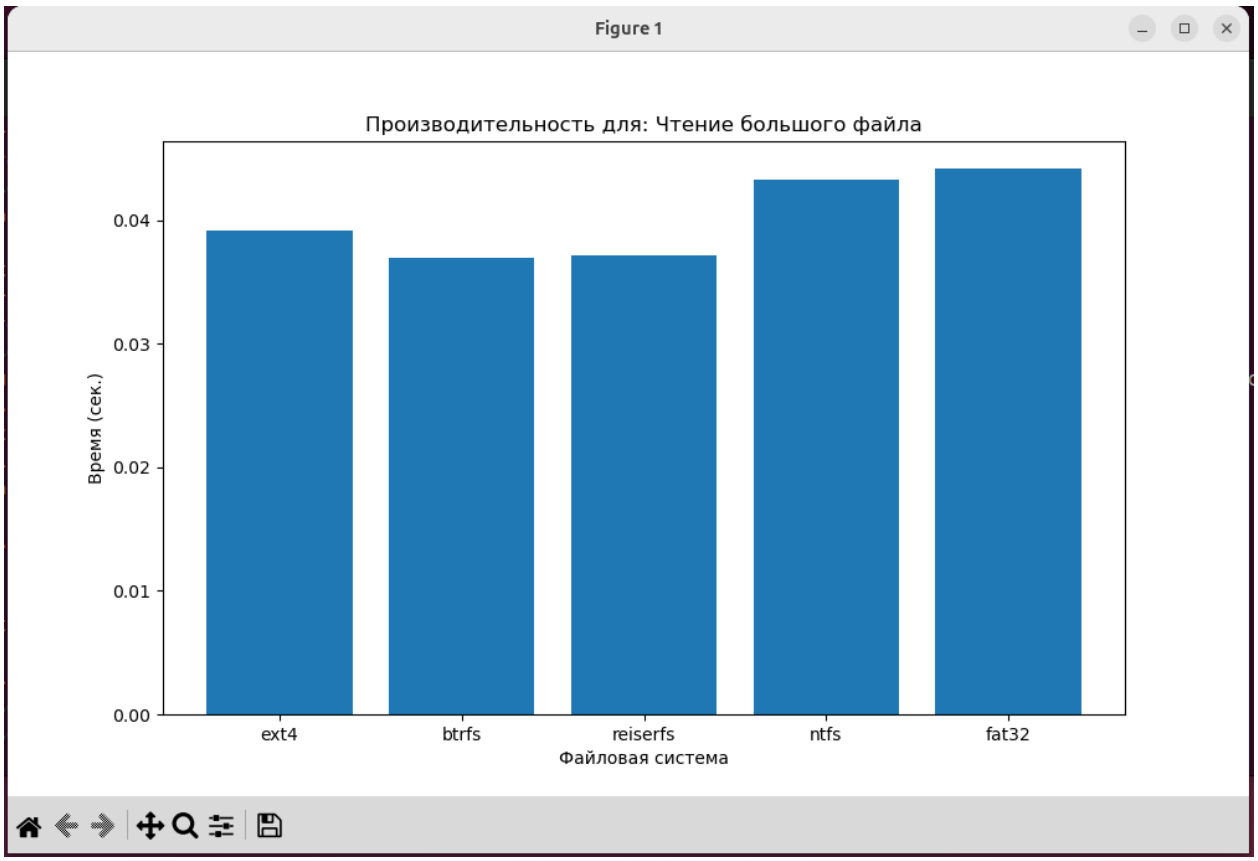


Figure 1
Производительность для: Чтение большого файла
Время (сек.)
0.00
0.01
0.02
0.03
0.04
ext4
btrfs
reiserfs
ntfs
fat32
Файловая система

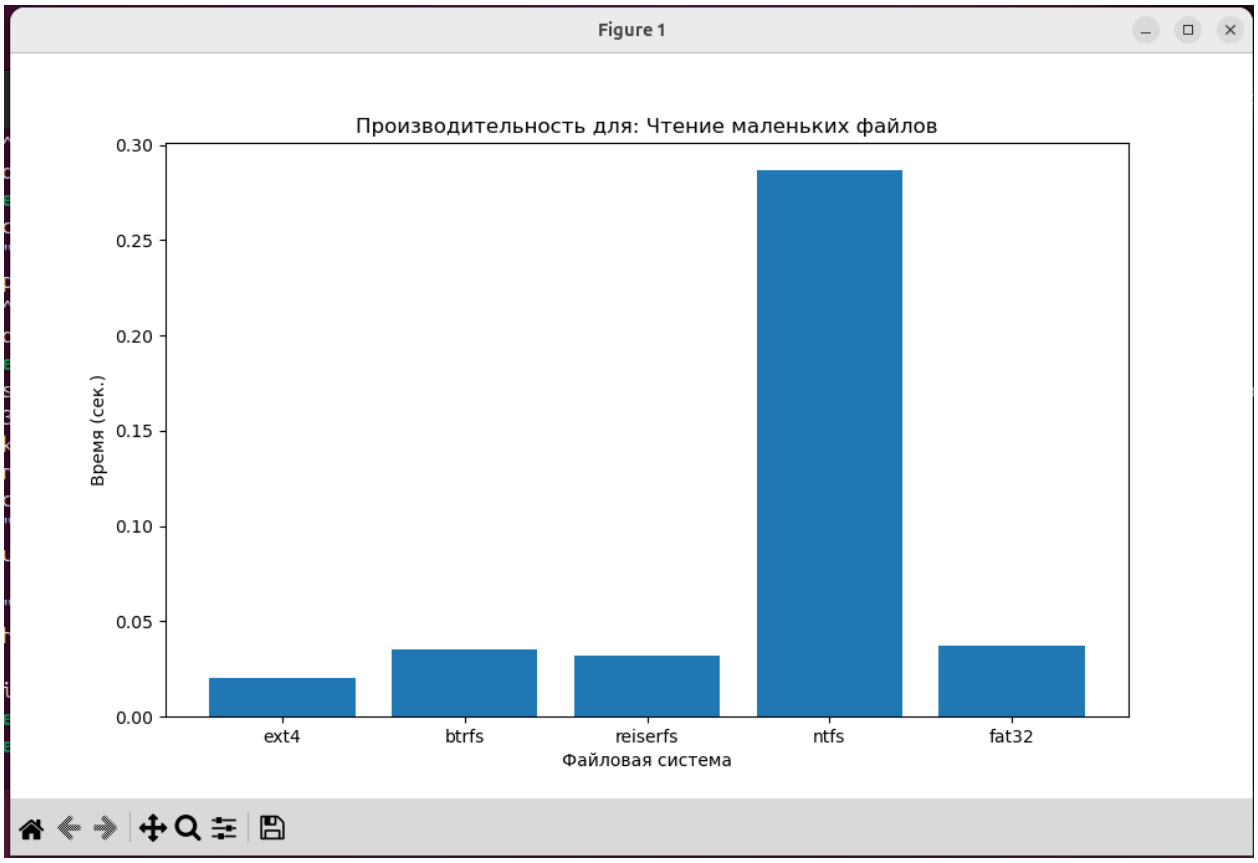
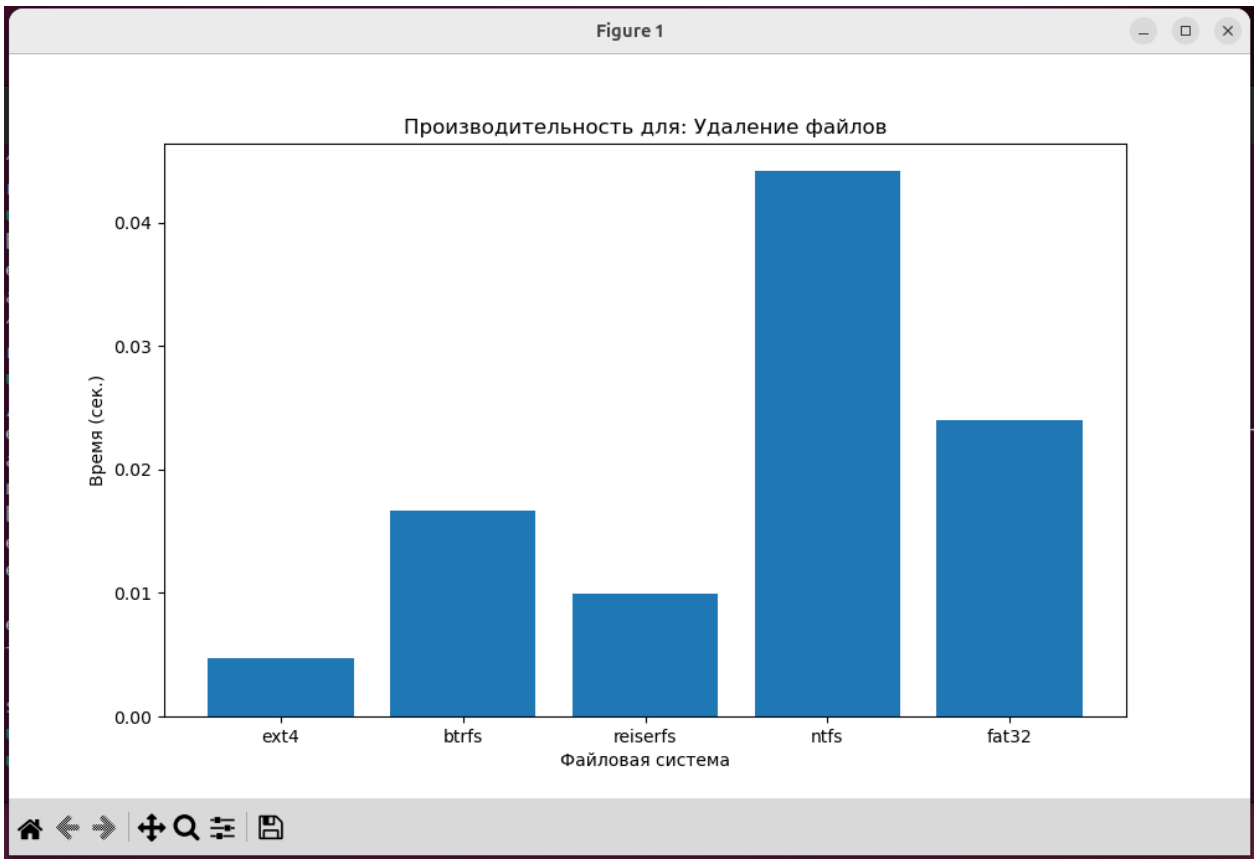


Figure 1
Производительность для: Чтение маленьких файлов
Время (сек.)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
ext4
btrfs
reiserfs
ntfs
fat32
Файловая система



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:

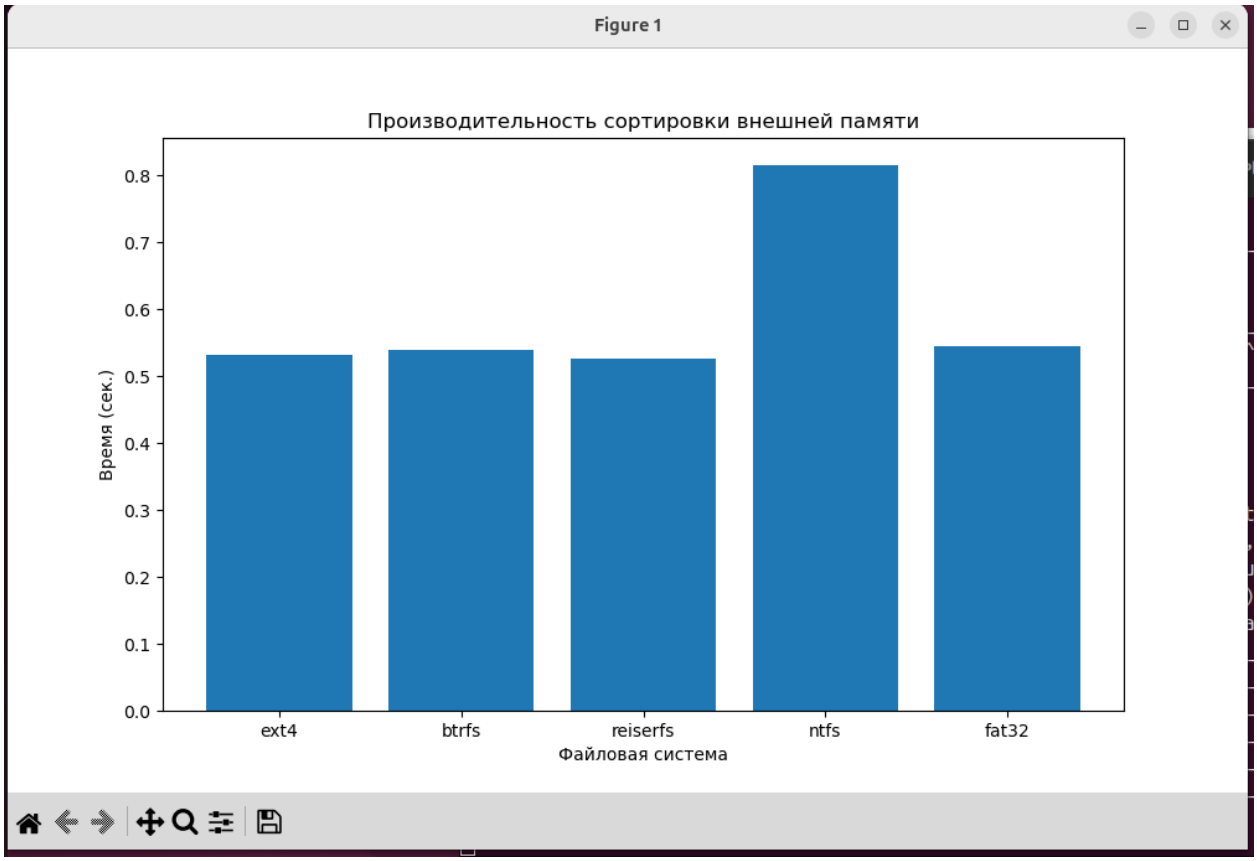
- Исходный файл был разделён на 500 временных файлов, каждый размером 12.3 kB.
- После завершения сортировки все временные файлы удаляются.

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).