Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)

(МАИ)

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра 307 «Цифровые технологии и информационные системы»

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Драйверы устройств»

Оценка задержки реакции на внешние воздействия

Проверил:

ст. преподаватель каф. 307

\_\_\_\_\_\_\_\_\_\_\_\_\_/Максимов А. Н./

Выполнили:

студент группы М3О-414Б-21

\_\_\_\_\_\_\_\_\_\_\_\_\_/Егоров Д.В.

студент группы М3О-414Б-21

\_\_\_\_\_\_\_\_\_\_\_\_/Домникова Е.А.

Цели лабораторной работы

1. Написать символьный драйвер, который будет генерировать с заданной частотой внешние воздействия и принимать реакцию на них;
2. Померить в драйвере время реакции (найти среднее и максимальное);
3. Построить гистограмму времени реакции;
4. Оценить влияние:

а) Приоритета процесса;

б) Дисциплины диспетчеризации;

в) Внешней загрузки (HDD, сеть).

Ход выполнения работы

Были написаны три программы:

1. Символьный драйвер response\_analyzer позволяет измерять время отклика пользовательской программы. Драйвер посылает пользовательской программе сигнал SIGUSR1, которая, после получения, немедленно отправляет уведомление обратно в драйвер через ioctl вызов IOCTL\_RECEIVED\_SIGNAL. Драйвер фиксирует разницу во времени между отправкой сигнала и получением уведомления. (см. приложение А);
2. Приложение, которое взаимодействует с устройством каждую секунду (см. приложение Б);
3. Программа, строящая гистограмму и вычисляющая среднее и максимальное значения (см. приложение В).

**Методы драйвера:**

ioctl:

IOCTL\_PID\_SET: Устанавливает PID процесса, которому будут отправляться сигналы. Принимает аргумент pid\_t - идентификатор процесса.

IOCTL\_RECEIVED\_SIGNAL: Уведомляет драйвер о получении сигнала пользовательской программой.

Обработка сигналов:

send\_sig(SIGUSR1, task\_reference, 0): Отправляет сигнал SIGUSR1 указанному процессу (task\_reference).

send\_sig(SIGINT, task\_reference, 0): Отправляет сигнал SIGINT процессу по завершении времени моделирования.

Сбор статистики:

total\_response\_time: Суммарное время отклика для расчета среднего значения.

max\_res\_time: Максимальное зафиксированное время отклика.

min\_res\_time: Минимальное зафиксированное время отклика.

response\_times\_history: Массив, хранящий историю измеренных времён отклика.

histogram\_data: Массив, в котором хранится гистограмма распределения времени отклика по диапазонам (бинам).

**Параметры:**

DRIVER\_NAME: Имя устройства (/dev/response\_analyzer).

NUMBER\_OF\_BINS: Количество диапазонов (бинов) в гистограмме (по умолчанию 10).

MODELING\_TIME\_SECONDS: Время моделирования в секундах (по умолчанию 5 минут).

timer\_period\_ms: Период срабатывания таймера, который отправляет сигнал SIGUSR1 (по умолчанию 1000 мс).

Сборка драйвера:

make

Загрузка драйвера в ядро:

sudo insmod response\_analyzer.ko

Компиляция пользовательской программы:

gcc main.c -o main

Запуск пользовательской программы от имени суперпользователя:

sudo ./main

Взаимодействие драйвера и пользовательской программы:

1. Пользовательская программа открывает устройство /dev/response\_analyzer.
2. С помощью ioctl команды IOCTL\_PID\_SET передает свой PID драйверу.
3. Драйвер запускает таймер, который с заданной периодичностью (timer\_period\_ms) вызывает функцию response\_timer\_function.
4. Функция response\_timer\_function отправляет сигнал SIGUSR1 пользовательской программе.
5. Пользовательская программа, получив сигнал, обрабатывает его в функции handle\_driver\_signal и отправляет уведомление драйверу с помощью ioctl команды IOCTL\_RECEIVED\_SIGNAL.
6. Драйвер в ioctl\_device при получении IOCTL\_RECEIVED\_SIGNAL фиксирует время получения уведомления, вычисляет разницу во времени и сохраняет ее в response\_times\_history.
7. По истечении времени моделирования (MODELING\_TIME\_SECONDS) драйвер отправляет сигнал SIGINT пользовательскому процессу.
8. Драйвер записывает время ответа в текстовый файл response\_times.txt.
9. Программа, написанная на Python высчитывает максимальное и среднее значения, а также строит гистограммы.

Были запущены следующие тесты:

1. В обычном режиме (см. рис. Рисунок 1):

Mean = 5282460.49

Max = 10912608

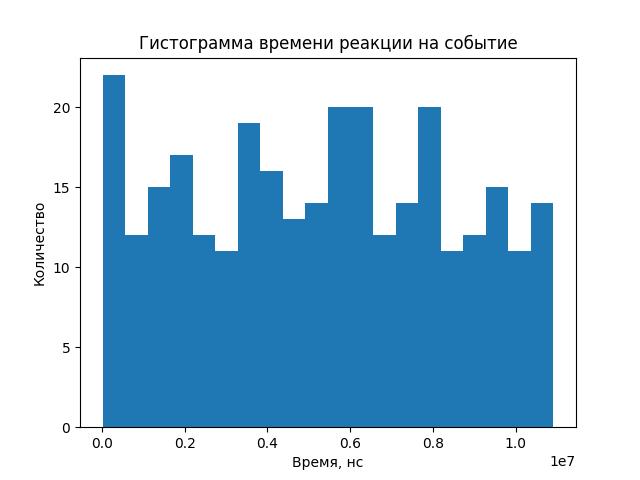


Рисунок 1 — Гистограмма времени реакции на событие в обычном режиме

1. Высокий приоритет (см. рис. Рисунок 2):

Mean = 5360921.886666667

Max = 10918775

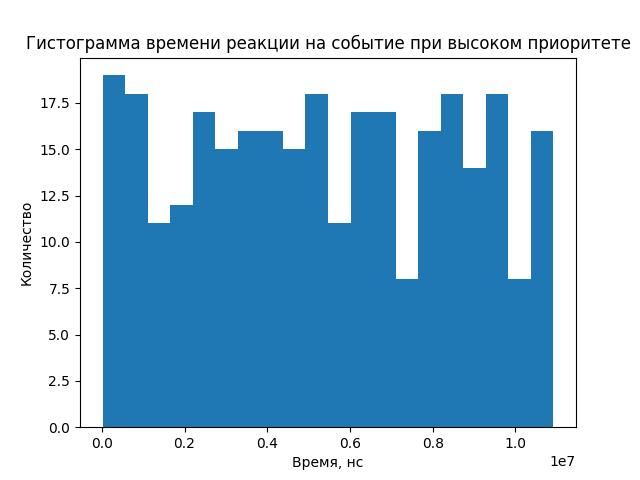


Рисунок 2 — Гистограмма времени реакции на событие при высоком приоритете

1. Низкий приоритет (см. рис. Рисунок 3):

Mean = 5431225.633333334

Max = 11127158

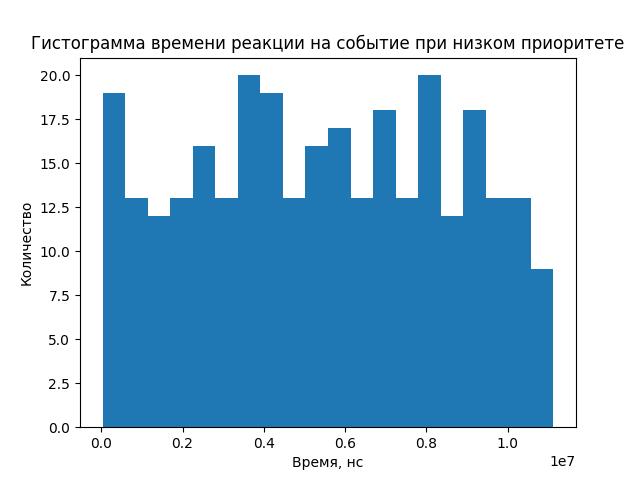


Рисунок 3 — Гистограмма времени реакции на событие при низком приоритете

Для проверки возможных дисциплин диспетчеризации необходимо выполнить

cat /sys/block/sda/queue/scheduler

В результате выполнения данной команды получено:

none [mq-deadline]

Данный результат говорит о том, что на устройстве, на котором ведется тестирование, доступна только одна дисциплина диспетчеризации.

1. Под нагрузкой HDD (см. рис. 4):

Mean = 5578212.3

Max = 11209446

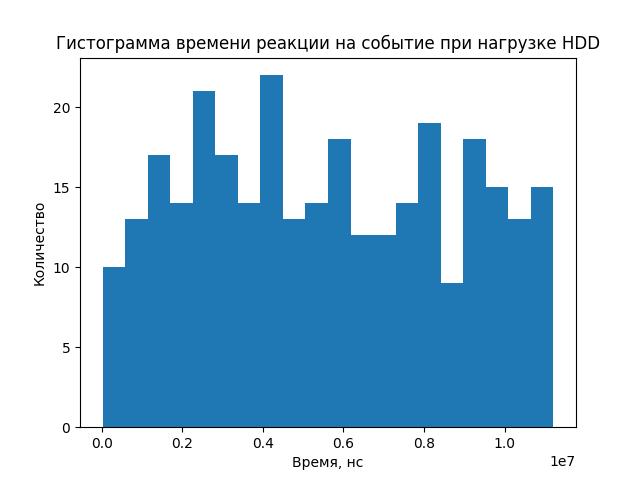


Рисунок 4 — Гистограмма времени реакции на событие при нагрузке HDD

1. Под нагрузкой сети (см. рис. 5)

Mean = 5390781.713333333

Max = 11546087

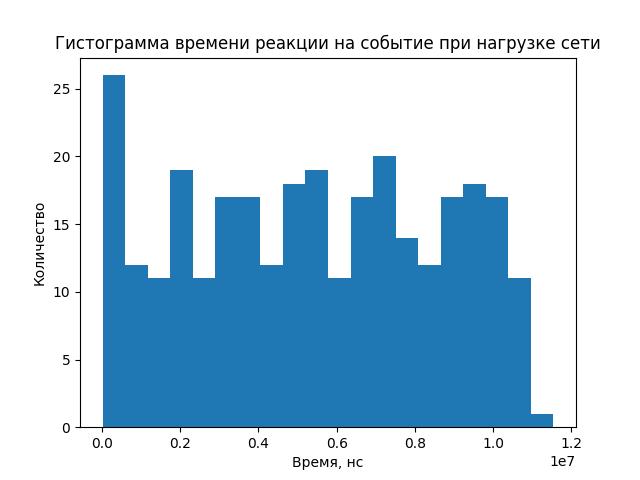


Рисунок 5 — Гистограмма времени реакции на событие при нагрузке сети

Вывод

В лабораторной работе №2 разработан символьный драйвер для оценки нагрузки сети. Приоритет процесса влияет на время реакции прямо пропорционально. Среднее время под нагрузкой HDD и сети больше, чем не под нагрузкой, причём для HDD оно больше, чем для сети.

Приложение А. Код драйвера

Код программы символьного драйвера написан на языке C.

#include <linux/module.h>

#include <linux/fs.h>

#include <linux/uaccess.h>

#include <linux/sched.h>

#include <linux/pid.h>

#include <linux/signal.h>

#include <linux/timer.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/cdev.h>

#include <linux/fs.h>

#include <linux/ktime.h>

#include <linux/file.h>

#include <linux/path.h>

#include <linux/namei.h>

#define DRIVER\_NAME "response\_analyzer"

#define IOCTL\_PID\_SET \_IOW('a', 'b', pid\_t)

#define IOCTL\_RECEIVED\_SIGNAL \_IO('a', 'd')

#define IOCTL\_EXIT \_IO('a', 'e')

#define MODELING\_TIME\_SECONDS 60\*5 // 5 минут

static pid\_t process\_id = -1;

static int device\_usage\_counter = 0;

static int major\_number;

static struct class \*device\_class;

struct timer\_list response\_timer;

unsigned long timer\_period\_ms = 1000;

ktime\_t signal\_sent\_time;

// Имя файла для сохранения данных

static char \*output\_file\_path = "/tmp/response\_times.txt";

module\_param(output\_file\_path, charp, 0000);

MODULE\_PARM\_DESC(output\_file\_path, "Path to the output file for response times");

int time\_measurements\_count = 0;

void response\_timer\_function(struct timer\_list \*t) {

if (process\_id != -1) {

struct pid \*pid\_reference = find\_get\_pid(process\_id);

if (pid\_reference) {

struct task\_struct \*task\_reference = pid\_task(pid\_reference, PIDTYPE\_PID);

if (task\_reference) {

send\_sig(SIGUSR1, task\_reference, 0);

signal\_sent\_time = ktime\_get\_real();

} else {

pr\_alert("Task not found for PID: %d\n", process\_id);

}

put\_pid(pid\_reference);

} else {

pr\_alert("Invalid PID: %d\n", process\_id);

}

}

mod\_timer(&response\_timer, jiffies + msecs\_to\_jiffies(timer\_period\_ms));

}

static void create\_and\_write\_to\_file(const char \*filename, const char \*data){

struct file \*file;

struct path path;

loff\_t pos = 0;

int err;

// Получаем путь к файлу по его имени

err = kern\_path(filename, LOOKUP\_FOLLOW, &path);

if (err) {

pr\_err("Failed to get path for %s\n", filename);

return;

}

// Открываем файл для записи (O\_CREAT | O\_WRONLY | O\_APPEND)

file = file\_open\_root(path.dentry, path.mnt, O\_CREAT | O\_WRONLY | O\_APPEND, 0644);

if (IS\_ERR(file)) {

pr\_err("Failed to open file %s\n", filename);

path\_put(&path);

return;

}

// Записываем данные в файл

kernel\_write(file, data, strlen(data), &pos);

// Закрываем файл

fput(file);

// Освобождаем ресурсы

path\_put(&path);

}

static int open\_device(struct inode \*inode\_ptr, struct file \*file\_ptr) {

if (device\_usage\_counter)

return -EBUSY;

device\_usage\_counter++;

pr\_alert("Device opened\n");

return 0;

}

static int close\_device(struct inode \*inode\_ptr, struct file \*file\_ptr) {

if (device\_usage\_counter <= 0)

return -EBUSY;

device\_usage\_counter--;

process\_id = -1;

pr\_alert("Device closed\n");

return 0;

}

static long ioctl\_device(struct file \*file\_ptr, unsigned int ioctl\_command, unsigned long ioctl\_param) {

switch (ioctl\_command) {

case IOCTL\_PID\_SET:

copy\_from\_user(&process\_id, (pid\_t \*)ioctl\_param, sizeof(pid\_t));

break;

case IOCTL\_RECEIVED\_SIGNAL:

{

ktime\_t signal\_received\_time = ktime\_get\_real();

s64 time\_difference = ktime\_to\_ns(ktime\_sub(signal\_received\_time, signal\_sent\_time));

// Запись времени отклика в файл

char data\_buffer[128];

int len = snprintf(data\_buffer, sizeof(data\_buffer), "%lld\n", time\_difference);

create\_and\_write\_to\_file(output\_file\_path, data\_buffer);

time\_measurements\_count++;

if (time\_measurements\_count == MODELING\_TIME\_SECONDS) {

if (process\_id != -1) {

struct pid \*pid\_reference = find\_get\_pid(process\_id);

if (pid\_reference) {

struct task\_struct \*task\_reference = pid\_task(pid\_reference, PIDTYPE\_PID);

if (task\_reference) {

send\_sig(SIGINT, task\_reference, 0);

} else {

pr\_alert("Task not found for PID: %d\n", process\_id);

}

put\_pid(pid\_reference);

} else {

pr\_alert("Invalid PID: %d\n", process\_id);

}

}

}

}

break;

default:

return -EINVAL;

}

return 0;

}

static struct file\_operations file\_ops = {

.open = open\_device,

.release = close\_device,

.unlocked\_ioctl = ioctl\_device,

};

static int \_\_init response\_analyzer\_init(void) {

// Регистрация старшего номера устройства

major\_number = register\_chrdev(0, DRIVER\_NAME, &file\_ops);

if (major\_number < 0) {

pr\_alert("register\_chrdev() failed: %d\n", major\_number);

return -EINVAL;

}

pr\_info("major = %d\n", major\_number);

// Создание класса устройств

device\_class = class\_create(DRIVER\_NAME);

if (IS\_ERR(device\_class)) {

pr\_alert("class\_create() failed: %ld\n", PTR\_ERR(device\_class));

return -EINVAL;

}

// Создание устройства

struct device\* dev = device\_create(device\_class, NULL, MKDEV(major\_number, 0), NULL, DRIVER\_NAME);

if (IS\_ERR(dev)) {

pr\_alert("device\_create() failed: %ld\n", PTR\_ERR(dev));

return -EINVAL;

}

pr\_info("/dev/%s created\n", DRIVER\_NAME);

// Инициализация и запуск таймера

timer\_setup(&response\_timer, response\_timer\_function, 0);

mod\_timer(&response\_timer, jiffies + msecs\_to\_jiffies(timer\_period\_ms));

return 0;

}

static void \_\_exit response\_analyzer\_exit(void) {

// Удаление таймера

del\_timer(&response\_timer);

// Удаление устройства

device\_destroy(device\_class, MKDEV(major\_number, 0));

// Удаление класса устройств

class\_destroy(device\_class);

// Отмена регистрации старшего номера устройства

unregister\_chrdev(major\_number, DRIVER\_NAME);

}

module\_init(response\_analyzer\_init);

module\_exit(response\_analyzer\_exit);

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("Response time measurement driver");

MODULE\_AUTHOR("Dima");

Приложение Б. Код программы

Код программы приложения написан на языке C.

#include <stdio.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <unistd.h>

#include <signal.h>

#include <stdlib.h>

#define DEVICE\_PATH "/dev/response\_analyzer"

#define IOCTL\_PID\_SET \_IOW('a', 'b', pid\_t)

#define IOCTL\_RECEIVED\_SIGNAL \_IO('a', 'd')

#define IOCTL\_EXIT \_IO('a', 'e')

int device\_file\_descriptor;

void handle\_driver\_signal(int signal\_number) {

printf("Received signal from driver!\n");

// Сигнализируем драйверу о приеме

if (device\_file\_descriptor < 0) {

perror("Device failed to open");

return;

}

ioctl(device\_file\_descriptor, IOCTL\_RECEIVED\_SIGNAL);

}

void handle\_program\_exit(int signal\_number) {

printf("Exit!\n");

close(device\_file\_descriptor);

exit(0);

}

int main() {

// Регистрация обработчика SIGINT

signal(SIGINT, handle\_program\_exit);

// Открытие файла устройства

device\_file\_descriptor = open(DEVICE\_PATH, O\_RDWR);

if (device\_file\_descriptor < 0) {

perror("Failed to open device");

return -1;

}

// Получение PID текущего процесса

pid\_t process\_id = getpid();

// Регистрация обработчика сигнала от драйвера

signal(SIGUSR1, handle\_driver\_signal);

// Установка PID процесса в драйвере

ioctl(device\_file\_descriptor, IOCTL\_PID\_SET, &process\_id);

// Бесконечный цикл ожидания сигнала

while (1) {

pause(); // Приостановка выполнения до получения сигнала

}

// Закрытие файла устройства (этот код никогда не выполнится в данном случае)

close(device\_file\_descriptor);

return 0;

}

Приложение В. Код обработки данных

Код обработки данных написан на языке Python.

import numpy as np

import matplotlib.pyplot as plt

import numpy as np

def calculate\_stats\_and\_plot\_histogram(filepath, bins=10):

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

data = [int(line.strip()) for line in f]

mean\_value = np.mean(data)

max\_value = max(data)

min\_value = min(data)

print(f"Mean: {mean\_value:.2f}")

print(f"Max: {max\_value}")

print(f"Min: {min\_value}")

plt.hist(data, bins=bins, edgecolor='black')

plt.xlabel("время (нс)")

plt.ylabel("количество")

plt.title("Data Distribution")

plt.legend()

plt.show()

# Путь к файлу с данными

file\_path = 'result.txt'

# Вызов функции для вычисления статистики и построения гистограммы

calculate\_stats\_and\_plot\_histogram(file\_path, bins=20)