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

\_\_\_\_\_\_\_\_\_\_\_\_\_/Евпатов П. А./

студент группы М3О-414Б-21

\_\_\_\_\_\_\_\_\_\_\_\_/Плюснин С. Г./

Задания 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).

Выполнение задания

Написаны три программы:

1. Модуль ядра (символьный драйвер), который считает время реакции (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).

Были запущены следующие тесты:

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

./make

Mean = 5282460.49

Max = 10912608

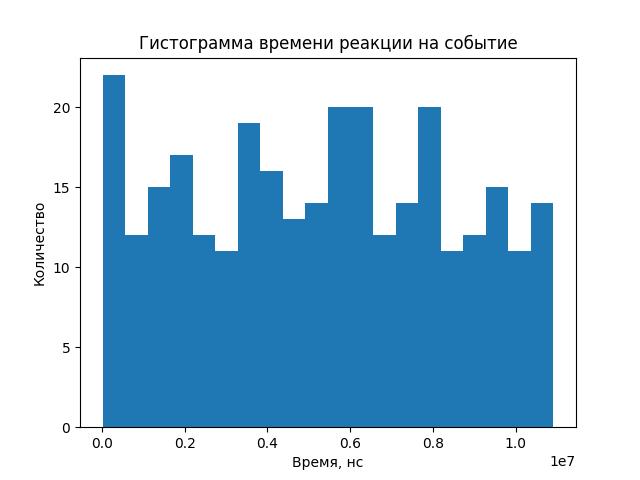


Рисунок 1 — Гистограмма времени реакции на событие в обычном режиме

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

sudo nice -n -20 ./main

Mean = 5360921.886666667

Max = 10918775

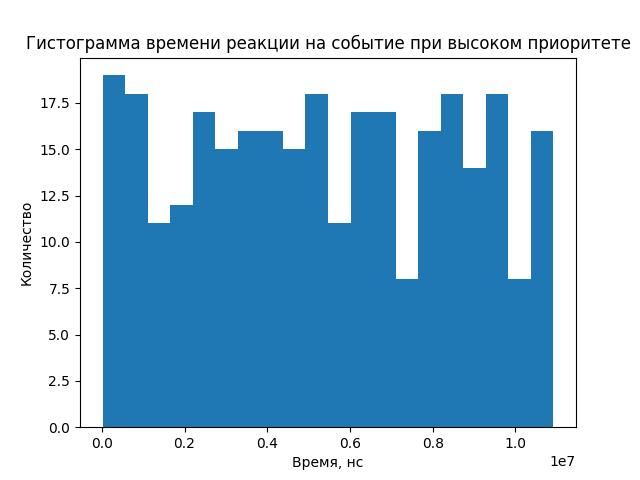


Рисунок 2 — Гистограмма времени реакции на событие при высоком приоритете

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

sudo nice -n 20 ./main

Mean = 5431225.633333334

Max = 11127158

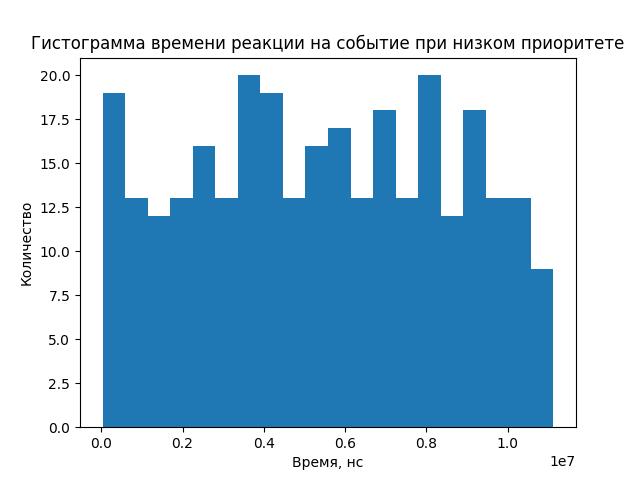


Рисунок 3 — Гистограмма времени реакции на событие при низком приоритете

1. Дисциплина диспетчеризации SCHED\_BATCH (см. рис. Рисунок 4):

sudo chrt -b 0 ./main

Mean = 5545536.44

Max = 10967494

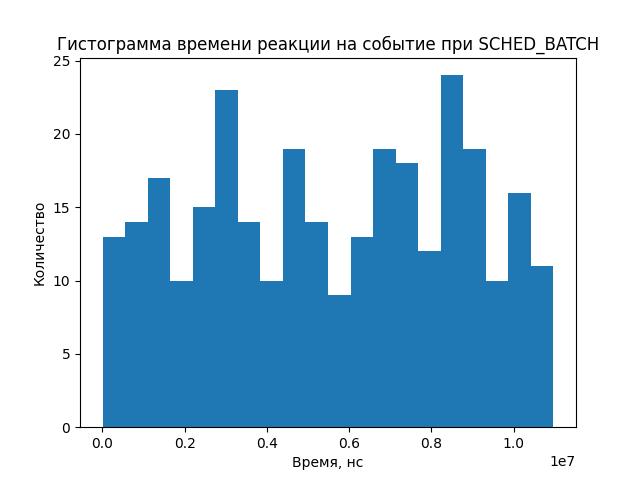


Рисунок 4 — Гистограмма времени реакции на событие при SCHED\_BATCH

1. Дисциплина диспетчеризации SCHED\_FIFO (см. рис. Рисунок 5):

sudo chrt -f 99 ./main

Mean = 5236244.243333333

Max = 11675149

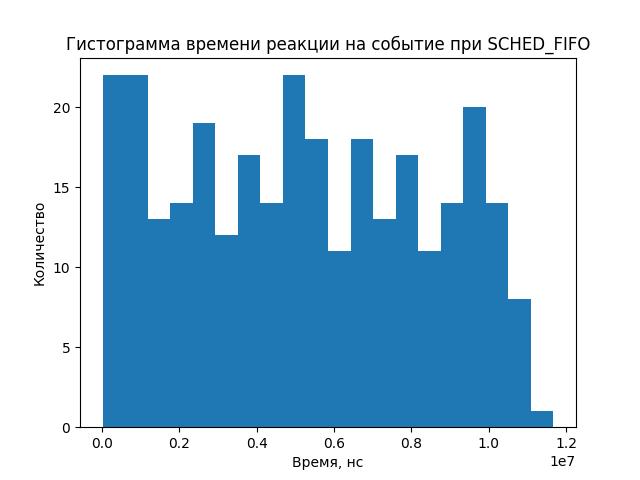


Рисунок 5 — Гистограмма времени реакции на событие при SCHED\_FIFO

1. Дисциплина диспетчеризации SCHED\_RR (см. рис. Рисунок 6):

sudo chrt -r 99 ./main

Mean = 5304392.723333334

Max = 10885460

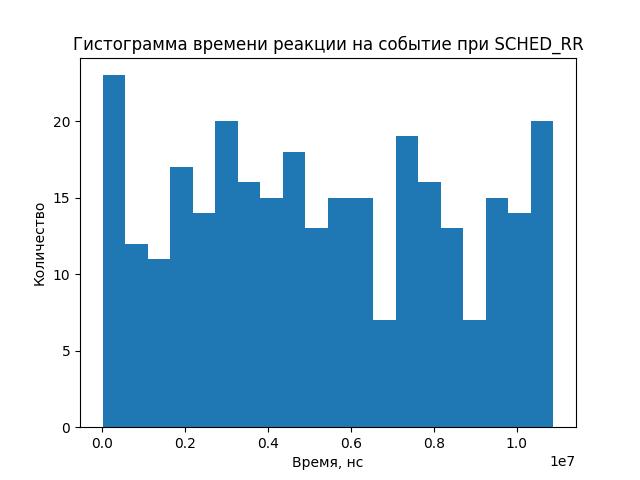


Рисунок 6 — Гистограмма времени реакции на событие при SCHED\_RR

1. Дисциплина диспетчеризации SCHED\_IDLE (см. рис. Рисунок 7):

sudo chrt -i 0 ./main

Mean = 5228909.083333333

Max = 10890074

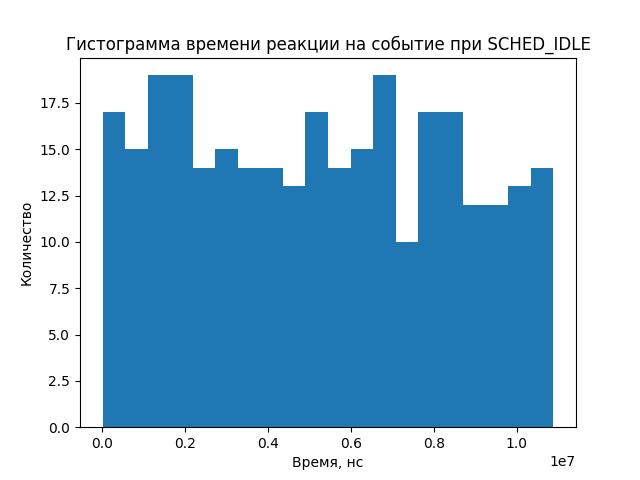


Рисунок 7 — Гистограмма времени реакции на событие при SCHED\_IDLE

1. Дисциплина диспетчеризации SCHED\_OTHER (см. рис. Рисунок 8):

sudo chrt -o 0 ./main

Mean = 5158550.03

Max = 10815034

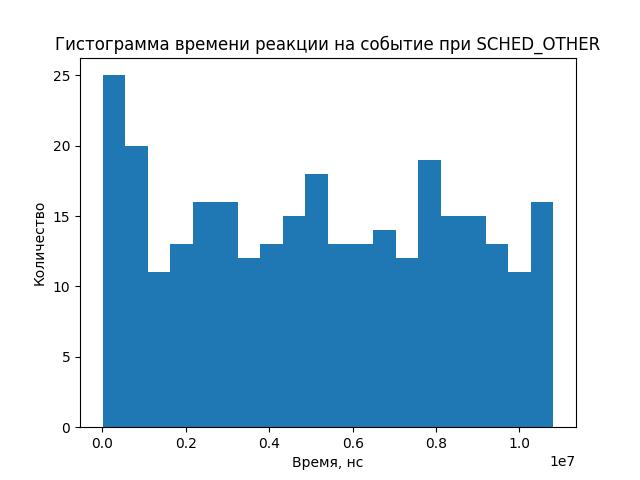


Рисунок 8 — Гистограмма времени реакции на событие при SCHED\_OTHER

1. Под нагрузкой HDD (см. рис. Рисунок 9):

while true; do dd if=/dev/urandom count=10M bs=1 | bzip2 -9 > /tmp/random.bin ; rm -f /tmp/random.bin ; done

Mean = 5578212.3

Max = 11209446

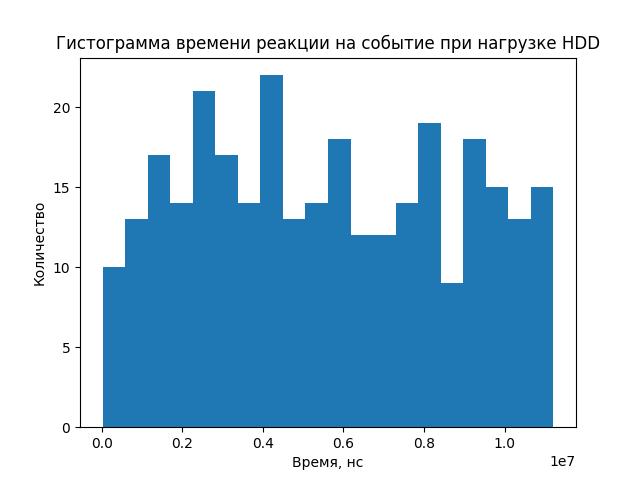


Рисунок 9 — Гистограмма времени реакции на событие при нагрузке HDD

1. Под нагрузкой сети (см. рис. Рисунок 10)

ncat -l 127.0.0.1 12345 -k -c 'xargs -n1 -i echo "Response: {}"'

Mean = 5390781.713333333

Max = 11546087

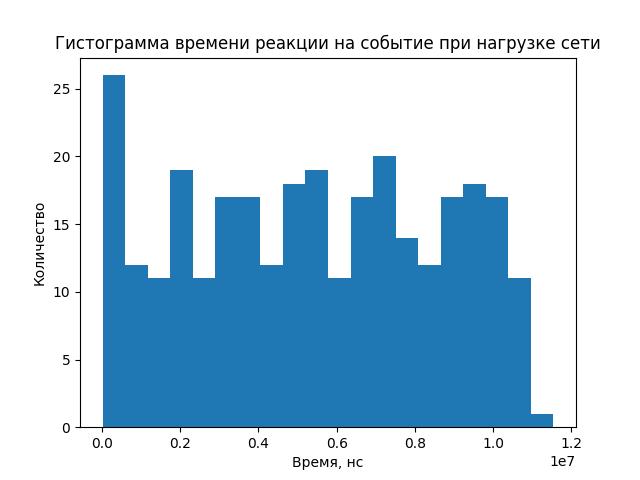


Рисунок 10 — Гистограмма времени реакции на событие при нагрузке сети

Вывод

В лабораторной работе №2 разработан символьный драйвер для оценки нагрузки сети. Приоритет процесса влияет на время реакции прямо пропорционально. Из всех дисциплин диспетчеризации наиболее лучшая SCHED\_OTHER. 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/kernel.h>

#include <linux/fs.h>

#include <linux/timer.h>

#include <linux/uaccess.h>

#include <linux/slab.h>

#include <linux/sched.h>

#include <linux/ktime.h>

#define DEVICE\_NAME "reaction\_time"

#define INITIAL\_BUFFER\_SIZE 10

static struct timer\_list my\_timer;

static int frequency = 100; // Частота в Гц

static ktime\_t event\_time;

static ktime\_t \*response\_time;

static int buffer\_size = INITIAL\_BUFFER\_SIZE;

static int index = 0;

static int device\_open(struct inode \*inode, struct file \*file) {

index = 0; // Сброс индекса при открытии устройства

return 0;

}

static ssize\_t device\_read(struct file \*file, char \_\_user \*buffer, size\_t len, loff\_t \*offset) {

if (index == 0) {

return 0; // Нет данных для чтения

}

// Ограничиваем количество байтов для чтения

size\_t count = min(len, sizeof(ktime\_t) \* index);

// Копируем данные времени реакции в пользовательское пространство

if (copy\_to\_user(buffer, response\_time, count)) {

return -EFAULT;

}

// Возвращаем количество байтов

return count;

}

static ssize\_t device\_write(struct file \*file, const char \_\_user \*buffer, size\_t len, loff\_t \*offset) {

if (index >= buffer\_size) {

// Увеличиваем размер буфера

buffer\_size \*= 2; // Увеличиваем вдвое

response\_time = krealloc(response\_time, sizeof(ktime\_t) \* buffer\_size, GFP\_KERNEL);

if (!response\_time) {

return -ENOMEM; // Ошибка при выделении памяти

}

}

ktime\_t current\_time = ktime\_get();

ktime\_t reaction\_time = ktime\_sub(current\_time, event\_time); // Рассчитываем время реакции

response\_time[index++] = reaction\_time; // Записываем время реакции

// Выводим время реакции в лог ядра

printk(KERN\_INFO "Reaction Time: %lld ns\n", ktime\_to\_ns(reaction\_time));

return len; // Возвращаем количество записанных байтов

}

void timer\_callback(struct timer\_list \*timer) {

event\_time = ktime\_get(); // Сохраняем время события

// Перезапускаем таймер

mod\_timer(&my\_timer, jiffies + msecs\_to\_jiffies(1000 / frequency));

}

static struct file\_operations fops = {

.owner = THIS\_MODULE,

.open = device\_open,

.read = device\_read,

.write = device\_write,

};

static int \_\_init reaction\_time\_init(void) {

int result;

// Регистрируем символьное устройство

result = register\_chrdev(0, DEVICE\_NAME, &fops);

if (result < 0) {

printk(KERN\_ALERT "Failed to register character device\n");

return result;

}

// Выделяем память для хранения времени реакции

response\_time = kmalloc(sizeof(ktime\_t) \* buffer\_size, GFP\_KERNEL);

if (!response\_time) {

unregister\_chrdev(result, DEVICE\_NAME);

return -ENOMEM; // Ошибка при выделении памяти

}

// Настраиваем таймер

timer\_setup(&my\_timer, timer\_callback, 0);

mod\_timer(&my\_timer, jiffies + msecs\_to\_jiffies(1000 / frequency));

printk(KERN\_INFO "Reaction Time Driver Initialized\n");

return 0;

}

static void \_\_exit reaction\_time\_exit(void) {

del\_timer(&my\_timer);

kfree(response\_time); // Освобождаем память

unregister\_chrdev(0, DEVICE\_NAME);

printk(KERN\_INFO "Reaction Time Driver Exited\n");

}

module\_init(reaction\_time\_init);

module\_exit(reaction\_time\_exit);

MODULE\_LICENSE("GPL");

Приложение Б. Код программы

Код программы приложения написан на языке C.

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <time.h>

#include <sys/ioctl.h>

#include <errno.h>

#define DEVICE\_NAME "/dev/reaction\_time"

void test\_reaction\_time\_device() {

int fd;

ssize\_t bytes\_written;

ssize\_t bytes\_read;

struct timespec \*response\_times;

size\_t response\_count = 300; // Количество ожидаемых времен реакции

// Открываем устройство

fd = open(DEVICE\_NAME, O\_RDWR);

if (fd < 0) {

perror("Failed to open device");

return;

}

printf("Device opened successfully.\n");

// Имитируем события реакции

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

// Записываем событие (здесь мы просто пишем что-то в устройство)

bytes\_written = write(fd, "1", 1);

if (bytes\_written < 0) {

perror("Failed to write to device");

close(fd);

return;

}

sleep(1); // Ждем 1 секунду перед следующим событием

}

// Читаем времена реакции

response\_times = malloc(sizeof(struct timespec) \* response\_count);

if (!response\_times) {

perror("Failed to allocate memory for response times");

close(fd);

return;

}

bytes\_read = read(fd, response\_times, sizeof(struct timespec) \* response\_count);

if (bytes\_read < 0) {

perror("Failed to read from device");

free(response\_times);

close(fd);

return;

}

printf("Read %zd bytes from device:\n", bytes\_read);

// Выводим времена реакции

for (int i = 0; i < bytes\_read / sizeof(struct timespec); i++) {

printf("Reaction Time %d: %ld ns\n", i\*2 +1,

response\_times[i].tv\_sec);

printf("Reaction Time %d: %ld ns\n", i \* 2 + 2,

response\_times[i].tv\_nsec);

}

// Освобождаем память и закрываем устройство

free(response\_times);

close(fd);

}

int main() {

test\_reaction\_time\_device();

return 0;

}

Приложение В. Код обработки данных

Код обработки данных написан на языке Python.

import numpy as np

from matplotlib import pyplot as plt

textfile = open("input.txt", 'r',encoding='cp1251')

text=textfile.read()

text = text.split('\n')

data = []

for i in range(len(text)):

str = text[i].split()

data.append(int(str[3]))

plt.hist(data, bins=20)

plt.title("Гистограмма времени реакции на событие при нагрузке сети")

plt.xlabel("Время, нс")

plt.ylabel("Количество")

plt.show()

print(f"Mean = {np.mean(data)}\nMax = {max(data)}")