Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)
Факультет Программной инженерии и компьютерной техники

Лабораторная работа №1
“Операционные системы”

Выполнили: студенты гр. P33113
Асварищ Михаил
Уметов Акбар
Преподаватель:
Покид Александр Владимирович

Санкт-Петербург 2020

Задание

Разработать программу на языке C, которая осуществляет следующие действия

- Создает область памяти размером A мегабайт, начинающихся с адреса B (если возможно) при помощи C=(malloc, mmap) заполненную случайными числами /dev/urandom в D потоков. Используя системные средства мониторинга определите адрес начала в адресном пространстве процесса и характеристики выделенных участков памяти. Замеры виртуальной/физической памяти необходимо снять:
- До аллокации
- После аллокации
- После заполнения участка данными
- После деаллокации
- Записывает область памяти в файлы одинакового размера E мегабайт с использованием F=(блочного, некешируемого) обращения к диску. Размер блока ввода-вывода G байт. Преподаватель выдает в качестве задания последовательность записи/чтения блоков H=(последовательный, заданный или случайный)
- Генерацию данных и запись осуществлять в бесконечном цикле.
- В отдельных I потоках осуществлять чтение данных из файлов и подсчитывать агрегированные характеристики данных - J=(сумму, среднее значение, максимальное, минимальное значение).
- Чтение и запись данных в/из файла должна быть защищена примитивами синхронизации K=(futex, cv, sem, flock).
- По заданию преподавателя изменить приоритеты потоков и описать изменения в характеристиках программы.
- Для запуска программы возможно использовать операционную систему Windows 10 или Debian/Ubuntu в виртуальном окружении.
- Измерить значения затраченного процессорного времени на выполнение программы и на операции ввода-вывода используя системные утилиты.
- Отследить трассу системных вызовов.
- Используя stap построить графики системных характеристик.

Вариант:
A=100;B=0xF56A669C;C=mmap;D=138;E=104;F=block;G=136;H=random;I=56;J=avg;K=cv

Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/file.h>
#include <sys/mman.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdint.h>

//A=100;B=0xF56A669C;C=mmap;D=138;E=104;F=block;G=136;H=random;I=56;J=avg;K=cv

#define MEMORY_SIZE (100 * 1024 * 1024)
#define ADDRESS (void*)0xF56A669C
#define WRITE_THREADS_SIZE 138
#define IO_BLOCK 136
#define ERROR_MMAP_MEMORY -11
#define FILE_SIZE 104 * 1024 * 1024
#define READ_THREADS_AMOUNT 56

void work_with_memory();

void create_for_memory_threads();

void* put_data_in_memory(void* args);

pthread_t* write_to_file();

pthread_t* read_from_file();

char* memory_pointer = NULL;
const char* filename = "file.bin";
pthread_mutex_t* mutex = NULL;
pthread_cond_t* cond = NULL;
int fd = 0;
bool isWriting = false;

void init_file() {
    fd = open(filename, O_RDWR | O_CREAT | O_TRUNC, (mode_t) 0600);
    if (fd == -1) {
        perror("Ошибка при создании файла для записи и чтения\n");
        exit(EXIT_FAILURE);
    }
}

int main(void) {
    cond = malloc(sizeof(pthread_cond_t));
    mutex = malloc(sizeof(pthread_mutex_t));
    pthread_cond_init(cond, NULL);
    pthread_mutex_init(mutex, NULL);

    while (1) {
        printf("Сделайте замер памяти до аллокации. После этого нажмите на Enter.\n");
        getchar();
        work_with_memory();

        init_file();
        pthread_t* write_thread = write_to_file();
        pthread_t* read_threads = read_from_file();

        pthread_join(*write_thread, NULL);
        for (int i = 0; i < READ_THREADS_AMOUNT; i++) {
```

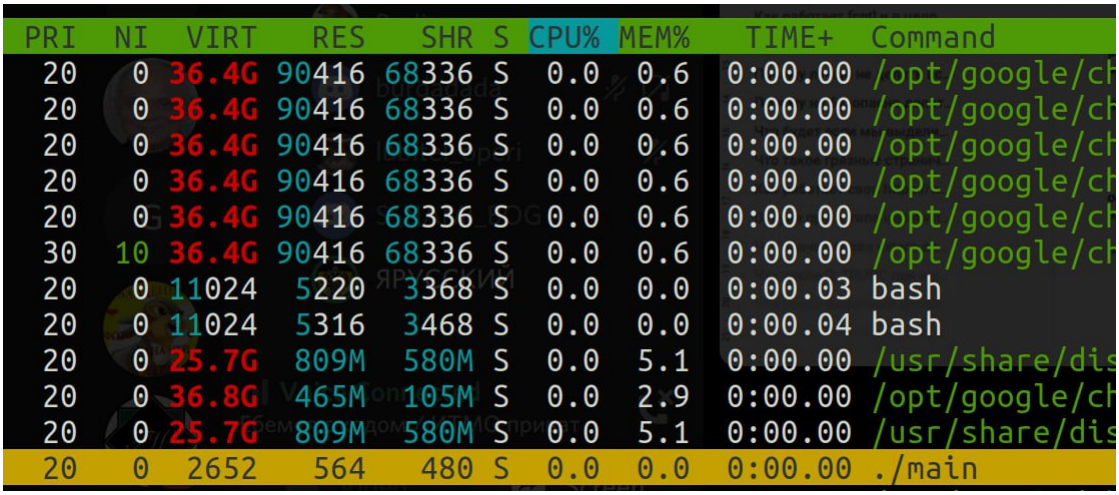
```
            pthread_join(read_threads[i], NULL);
        }

        free(write_thread);
        free(read_threads);
        munmap(memory_pointer, MEMORY_SIZE);
    }

    return 0;
}

void work_with_memory() {
    memory_pointer = mmap(ADDRESS, MEMORY_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, 0, 0);
    if (memory_pointer == MAP_FAILED) {
        printf("При аллокации произошла ошибка, пожалуйста попробуйте снова чуть
позже.\n");
        exit(ERROR_MMAP_MEMORY);
    }

    printf("Сделайте замер паяти после аллокации. После этого нажмите на Enter.\n");
    printf("Адресс начала аллоцируемой памяти = %p\n", memory_pointer);
    getchar();

    create_for_memory_threads();
    printf("Сделайте замер памяти после заполнения аллоцируемой памяти. После этого
нажмите на Enter.\n");
    getchar();

    printf("Работа с памятью закончена.\nСделайте замер памяти после деаллокации. После
этого нажмите на Enter.\n");
    getchar();
}

void create_for_memory_threads() {
    pthread_t threads[WRITE_THREADS_SIZE];
    int threads_id[WRITE_THREADS_SIZE];
    for (int i = 0; i < WRITE_THREADS_SIZE; i++) {
        threads_id[i] = i;
        pthread_create(&threads[i], NULL, put_data_in_memory, (void*) &threads_id[i]);
    }
    for (int i = 0; i < WRITE_THREADS_SIZE; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* put_data_in_memory(void* args) {
    int id = *(int*) args;
    int block = MEMORY_SIZE / WRITE_THREADS_SIZE;
    char* start_of_block = memory_pointer + block * id;
    char* end_of_block = (id == WRITE_THREADS_SIZE) ? (memory_pointer + MEMORY_SIZE) :
(memory_pointer + block * (id + 1));

    int urandom;
    int tmp = IO_BLOCK;
    char buffer[IO_BLOCK];

    if ((urandom = open("/dev/urandom", O_RDONLY)) < 0) {
        printf("Невозможно открыть /dev/urandom\n");
        exit(EXIT_FAILURE);
    }

    for (char* current_address = start_of_block; current_address < end_of_block;
current_address++) {
        if (tmp == IO_BLOCK) {
```

```
            read(urandom, buffer, IO_BLOCK);
            tmp = 0;
        }
        *current_address = buffer[tmp++];
    }

    close(urandom);

    return NULL;
}

void* write_to_file_thread() {
    printf("Поток номер %lu начал запись в файл\n", (uint64_t) pthread_self());
    char* io_block = malloc(sizeof(char) * IO_BLOCK);

    for (size_t offset = 0; offset < MEMORY_SIZE; offset += IO_BLOCK) {
//        printf("%lu/%d\n", offset, MEMORY_SIZE);
        if (offset + IO_BLOCK >= MEMORY_SIZE) {
            memcpy(io_block, memory_pointer + offset, MEMORY_SIZE - offset);
        } else {
            memcpy(io_block, memory_pointer + offset, IO_BLOCK);
        }


        pthread_mutex_lock(mutex);
        isWriting = true;

        ssize_t bytesWritten = write(fd, io_block, IO_BLOCK);
        if (bytesWritten == -1) {
            perror("Ошибка при записи в файл");
            return NULL;
        }

        isWriting = false;
        pthread_cond_signal(cond);
        pthread_mutex_unlock(mutex);
    }

    free(io_block);
    printf("Поток номер %lu закончил запись в файл\n", (uint64_t) pthread_self());
    return NULL;
}

pthread_t* write_to_file() {
    pthread_t* write_thread = malloc(sizeof(pthread_t));
    pthread_create(write_thread, NULL, write_to_file_thread, NULL);
    return write_thread;
}

void* read_from_file_thread() {
    float sum = 0;
    float n = 0;
    printf("Поток номер %lu начал читать из файла\n", (uint64_t) pthread_self());

    pthread_mutex_lock(mutex);
    lseek(fd, 0, SEEK_SET);
    pthread_mutex_unlock(mutex);

    char* io_block = malloc(sizeof(char) * IO_BLOCK);
    for (int i = 0; i < FILE_SIZE / IO_BLOCK; i++) {
        pthread_mutex_lock(mutex);
        while (isWriting) {
            pthread_cond_wait(cond, mutex);
        }
```

```
        ssize_t readBytes = read(fd, io_block, IO_BLOCK);
        if (readBytes == -1) {
            perror("Ошибка при записи в файл");
            return NULL;
        }

        for (int j = 0; j < IO_BLOCK; j++) {
            sum += (unsigned char) io_block[j];
            n += 1;
        }

        pthread_mutex_unlock(mutex);
    }
    float avg = sum / n;
    printf("Среднее значение в файле: %f\n", avg);

    free(io_block);
    printf("Поток номер %lu закончил читать из файла\n", (uint64_t) pthread_self());
    return NULL;
}

pthread_t* read_from_file() {
    pthread_t* read_threads = malloc(READ_THREADS_AMOUNT * sizeof(pthread_t));

    for (int i = 0; i < READ_THREADS_AMOUNT; i++) {
        pthread_create(&read_threads[i], NULL, read_from_file_thread, NULL);
    }

    return read_threads;
}
```

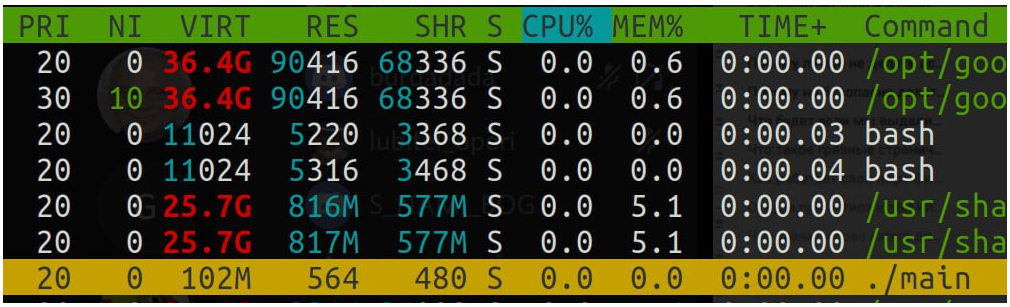
Замеры памяти

До аллокации:



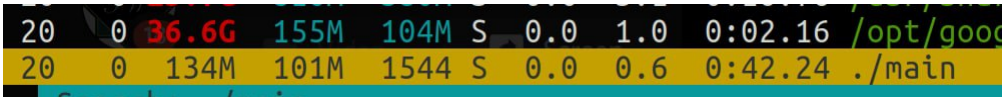
PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
30	10	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/google/ch
20	0	11024	5220	3368	S	0.0	0.0	0:00.03	bash
20	0	11024	5316	3468	S	0.0	0.0	0:00.04	bash
20	0	25.7G	809M	580M	S	0.0	5.1	0:00.00	/usr/share/dis
20	0	36.8G	465M	105M	S	0.0	2.9	0:00.00	/opt/google/ch
20	0	25.7G	809M	580M	S	0.0	5.1	0:00.00	/usr/share/dis
20	0	2652	564	480	S	0.0	0.0	0:00.00	./main

После аллокации:



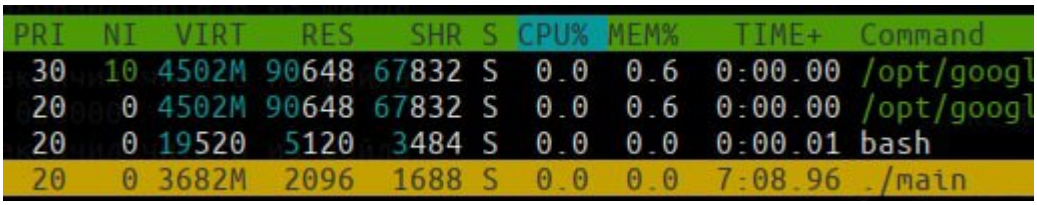
PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
20	0	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/goo
30	10	36.4G	90416	68336	S	0.0	0.6	0:00.00	/opt/goo
20	0	11024	5220	3368	S	0.0	0.0	0:00.03	bash
20	0	11024	5316	3468	S	0.0	0.0	0:00.04	bash
20	0	25.7G	816M	577M	S	0.0	5.1	0:00.00	/usr/sha
20	0	25.7G	817M	577M	S	0.0	5.1	0:00.00	/usr/sha
20	0	102M	564	480	S	0.0	0.0	0:00.00	./main

После заполнения данными:



20	0	36.6G	155M	104M	S	0.0	1.0	0:02.16	/opt/goo
20	0	134M	101M	1544	S	0.0	0.6	0:42.24	./main

После деаллокации:



PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
30	10	4502M	90648	67832	S	0.0	0.6	0:00.00	/opt/googl
20	0	4502M	90648	67832	S	0.0	0.6	0:00.00	/opt/googl
20	0	19520	5120	3484	S	0.0	0.0	0:00.01	bash
20	0	3682M	2096	1688	S	0.0	0.0	7:08.96	./main

Карта памяти процесса

f56a6000-fbaa6000 rw-p 00000000 00:00 0
55e665d1c000-55e665d1d000 r--p 00000000 08:13 3417686 /home/akbar/itmo/os/lab/main
55e665d1d000-55e665d1e000 r-xp 00001000 08:13 3417686
/home/akbar/itmo/os/lab/main
55e665d1e000-55e665d1f000 r--p 00002000 08:13 3417686 /home/akbar/itmo/os/lab/main
55e665d1f000-55e665d20000 r--p 00002000 08:13 3417686 /home/akbar/itmo/os/lab/main

```
55e665d20000-55e665d21000 rw-p 00003000 08:13 3417686
/home/akbar/itmo/os/lab/main
55e666e8d000-55e666eae000 rw-p 00000000 00:00 0                  [heap]
7f40842e3000-7f40842e4000 ---p 00000000 00:00 0
7f40842e4000-7f4084ae4000 rw-p 00000000 00:00 0 s
7f4084ae4000-7f4084ae5000 ---p 00000000 00:00 0
7f4084ae5000-7f40852e5000 rw-p 00000000 00:00 0
7f40852e5000-7f40852e6000 ---p 00000000 00:00 0
7f40852e6000-7f4085ae6000 rw-p 00000000 00:00 0
7f4085ae6000-7f4085ae7000 ---p 00000000 00:00 0
7f4085ae7000-7f40862e7000 rw-p 00000000 00:00 0
7f40c936d000-7f40c9370000 rw-p 00000000 00:00 0
7f40c9370000-7f40c9395000 r--p 00000000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c9395000-7f40c950d000 r-xp 00025000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c950d000-7f40c9557000 r--p 0019d000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c9557000-7f40c9558000 ---p 001e7000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c9558000-7f40c955b000 r--p 001e7000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c955b000-7f40c955e000 rw-p 001ea000 08:13 12068232           /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f40c955e000-7f40c9562000 rw-p 00000000 00:00 0
7f40c9562000-7f40c9569000 r--p 00000000 08:13 12068245           /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f40c9569000-7f40c957a000 r-xp 00007000 08:13 12068245           /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f40c957a000-7f40c957f000 r--p 00018000 08:13 12068245           /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f40c957f000-7f40c9580000 r--p 0001c000 08:13 12068245           /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f40c9580000-7f40c9581000 rw-p 0001d000 08:13 12068245           /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f40c9581000-7f40c9587000 rw-p 00000000 00:00 0
7f40c95a1000-7f40c95a2000 r--p 00000000 08:13 12068220           /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f40c95a2000-7f40c95c5000 r-xp 00001000 08:13 12068220           /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f40c95c5000-7f40c95cd000 r--p 00024000 08:13 12068220           /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f40c95ce000-7f40c95cf000 r--p 0002c000 08:13 12068220           /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f40c95cf000-7f40c95d0000 rw-p 0002d000 08:13 12068220           /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f40c95d0000-7f40c95d1000 rw-p 00000000 00:00 0
7ffed9818000-7ffed983a000 rw-p 00000000 00:00 0                  [stack]
7ffed98c8000-7ffed98cc000 r--p 00000000 00:00 0                  [vvar]
7ffed98cc000-7ffed98ce000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0          [vsyscall]
```

Агрегированное значение

Максимум значений, записанных в файлы: 256. Так как программа оперирует одно байтовыми беззнаковыми целыми числами.

Время затраченное на ввод-вывод

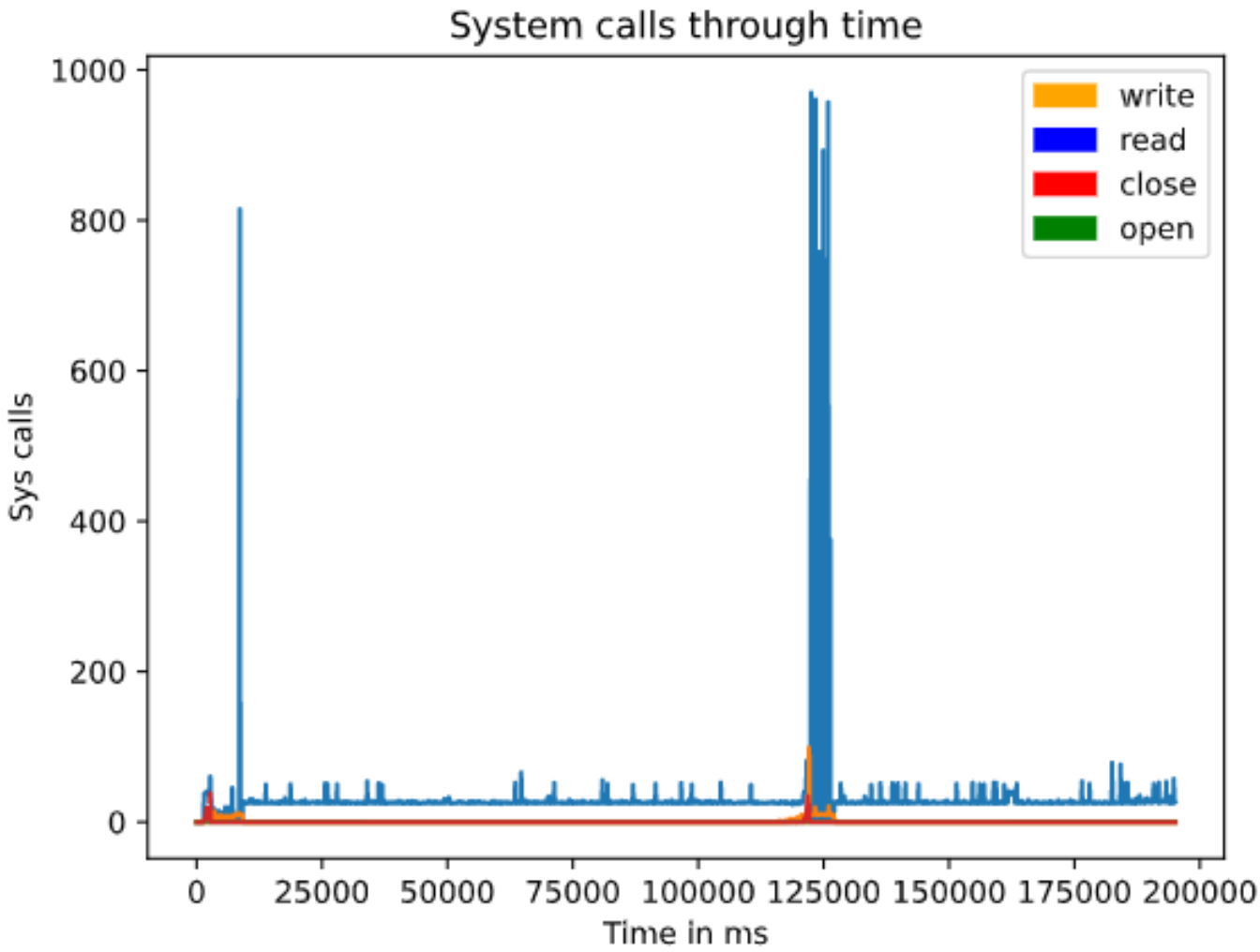
Подсчет производился с помощью time.

Время затраченное на вывод в файлы: 0.8 секунд Время

затраченное на чтение из файлов: 3.6 секунды

График системных характеристик, посчитанных с помощью stap

Оранжевая линия - вызовы write
Синяя линии – вызовы read
Зеленая и красная – вызовы open и close



Код скрипта:

```
global read, write, start, open, close

probe begin {
   start = gettimeofday_s()
}
probe syscall.write {
   if (pid() == target())
       write += 1
}

probe syscall.read {
   if (pid() == target())
       read += 1
}

probe syscall.open {
   if (pid() == target())
       open += 1
}

probe syscall.close {
   if (pid() == target())
       close += 1
}

probe timer.ms(100) {
   printf("%d\t%d\t%d\t%d\t%16d\n", read, write, open, close, task_stime())
   read=0
   write=0
   open = 0
   close = 0
}
```

Вывод

В ходе выполнения лабораторной работы я познакомился с новыми инструментами мониторинга ОС и процессов, например system tap или pmap.