НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Операционные системы

Лабораторная работа №1

Выполнил:

студент группы P33111

Окладников Константин Константинович

Проверил:

Осипов Святослав Владимирович

Санкт-Петербург

2021 г.

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:
    1. До аллокации
    2. После аллокации
    3. После заполнения участка данными
    4. 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

Размер области памяти:	A=197
Начальный адрес:	B=0x513F4598
Создать с помощью этого:	C=mmap
Количество потоков:	D=35
Размер файлов для записи:	E=26
Тип обращения к диску:	F=block
Размер блока I/O:	G=51
Последовательность I/O блоков:	H=random
Число потоков для I из файлов:	I=26
Подсчитать характеристику:	J=avg
Примитив синхронизации	K=sem

Исходный код

```
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>
#include "memory.h"
#include "stdatomic.h"

// A=197;B=0x513F4598;C=mmap;D=35;E=26;F=block;G=51;H=random;I=26;J=avg;K=sem

#define MEM_SIZE 197*1024*1024
#define FIRST_ADDRESS 0x513F4598
#define NUM_THREADS_MEMORY 35
#define NUM_THREADS_FILE 22
#define FILE_SIZE 26*1024*1024
#define BLOCK_SIZE 51
#define RANDOM_FILE_NAME "/dev/urandom"
#define FULL_FILES_COUNT ((MEM_SIZE) / (FILE_SIZE))

typedef struct{
    int thread_num;
    int * memory_region;
    int start_number;
    int end_number;
    int fd;
}memory_fill_params;

typedef struct{
    int * memory_region;
}file_write_params;

typedef struct{
    int file_number;
    int thread_number;
}file_read_params;

int cycle_stop = 0;
long long sum = 0;
```

```
int count = 0;
float avg;
int min_value = INT_MAX;
sem_t semaphore_file[FULL_FILES_COUNT+1];
sem_t main_sem;
atomic_int thread_counter = 0;

void write_to_file(const int * memory_region, int start, int fd, int file_size){
    int num_blocks = file_size / BLOCK_SIZE;
    char *buffer = (char *) malloc(BLOCK_SIZE);
    char *v_memory = (char *) memory_region + start;

    for(int i = 0; i <= num_blocks * 2; i++){
        int block_number = rand() % (num_blocks + 1);
        int start_byte = block_number * BLOCK_SIZE;
        int block_size = block_number == num_blocks ? file_size - (BLOCK_SIZE*num_blocks) : BL
OCK_SIZE;
        memcpy(buffer, v_memory + start_byte, block_size);
        pwrite(fd, buffer, block_size, start_byte);
    }

    free(buffer);
}

void* file_write_thread(void * params_void){
    file_write_params *params = (file_write_params *) params_void;
    printf("[Writer] start\n");

    thread_counter++;
    if(thread_counter == NUM_THREADS_MEMORY+NUM_THREADS_FILE+1){
        sem_post(&main_sem);
    }

    int first_run = 1;
    do{
        //заполняем файлы
        for(int i=0; i<=FULL_FILES_COUNT; i++){
            char filename[16];
            sprintf(filename, "lab1_file_%i.bin", i);
            if(!first_run){
                sem_wait(&semaphore_file[i]); //при первом запуске семафор захвачен заранее
            }
            int fd = open(filename, O_CREAT | O_WRONLY | O_TRUNC, 00666);
            if(fd == -1){
                printf("[Writer] can not open file %i\n", i);
                exit(-1);
            }
            //в последний файл сложим остатки
            int file_size = i == FULL_FILES_COUNT ? MEM_SIZE - (FULL_FILES_COUNT * FILE_SIZE)
: FILE_SIZE;
            write_to_file(params->memory_region, FILE_SIZE/4*i, fd, file_size);
            close(fd);
            sem_post(&semaphore_file[i]);
        }
        if(first_run){
            first_run = 0;
        }
    } while (!cycle_stop);

    printf("[Writer] finish\n");
    return NULL;
}

void read_from_file(int *memory_region, int fd, int file_size){
    int num_blocks = file_size / BLOCK_SIZE;
    char *buffer = (char *) malloc(BLOCK_SIZE);
    char *v_memory = (char *) memory_region;

    for(int i = 0; i <= num_blocks * 2; i++){
```

```
        int block_number = rand() % (num_blocks + 1);
        int start_byte = block_number * BLOCK_SIZE;
        int block_size = block_number == num_blocks ? file_size - (BLOCK_SIZE*num_blocks) : BL
OCK_SIZE;
        pread(fd, buffer, block_size, start_byte);
        memcpy(v_memory + start_byte, buffer, block_size);
    }

    free(buffer);
}

void* file_read_thread(void * params_void){
    file_read_params *params = (file_read_params *) params_void;
    printf("[Reader %i] start with file %i\n", params->thread_number, params->file_number);
    thread_counter++;
    if(thread_counter == NUM_THREADS_MEMORY+NUM_THREADS_FILE+1){
        sem_post(&main_sem);
    }
    do{
        //заполняем файлы
        char filename[16];
        sprintf(filename, "lab1_file_%i.bin", params->file_number);
        sem_wait(&semaphore_file[params->file_number]);
        int fd = open(filename, O_RDONLY);
        if(fd == -1){
            sem_post(&semaphore_file[params->file_number]);
            printf("[Reader %i] can not open file %i\n", params->thread_number, params-
>file_number);
            return NULL;
        }

        //последний файл короче остальных
        int file_size = params-
>file_number == FULL_FILES_COUNT ? MEM_SIZE - (FULL_FILES_COUNT * FILE_SIZE) : FILE_SIZE;
        int *memory = (int *) malloc(file_size);
        read_from_file(memory, fd, file_size);
        close(fd);
        sem_post(&semaphore_file[params->file_number]);

        for(int i=0; i<file_size/4; i++){
            sum += memory[i];
            count++;
        }

        free(memory);
    } while (!cycle_stop);
    printf("[Reader %i] finish\n", params->thread_number);
    return NULL;
}

void* fill_memory_thread(void * params_void){
    memory_fill_params *params = (memory_fill_params *) params_void;
    printf("[Generator %i] start\n", params->thread_num);
    thread_counter++;
    if(thread_counter == NUM_THREADS_MEMORY+NUM_THREADS_FILE+1){
        sem_post(&main_sem);
    }
    char *memory_segment = (char *) params->memory_region + (params->start_number * 4);
    int size = (params->end_number - params->start_number) * 4;

    do{
        pread(params->fd, memory_segment, size, 0);
    } while (!cycle_stop);

    printf("[Generator %i] finish\n", params->thread_num);
    return NULL;
}

void fill_memory(int * memory_region){
```

```
    cycle_stop=0;
    sem_init(&main_sem, 0, 0);

    int fd = open(RANDOM_FILE_NAME, O_RDONLY);
    pthread_t *memory_fillers = (pthread_t*) malloc(NUM_THREADS_MEMORY * sizeof(pthread_t));
    memory_fill_params *memory_data = (memory_fill_params *) malloc(NUM_THREADS_MEMORY * sizeo
f(memory_fill_params));
    int segment_size = (MEM_SIZE / 4 / NUM_THREADS_MEMORY) + 1;

    for(int i=0; i<NUM_THREADS_MEMORY; i++){
        memory_data[i].thread_num=i;
        memory_data[i].start_number= i * segment_size;
        memory_data[i].end_number = i != NUM_THREADS_MEMORY - 1 ? (i + 1) * segment_size : MEM
_SIZE / 4;
        memory_data[i].memory_region = memory_region;
        memory_data[i].fd = fd;
        pthread_create(&(memory_fillers[i]), NULL, fill_memory_thread, &memory_data[i]);
    }

    //file writing params
    for(int i=0; i<=FULL_FILES_COUNT; i++){
        sem_init(&semaphore_file[i], 0, 1);
        sem_wait(&semaphore_file[i]); //блокируем все файлы, пока они не будут созданы
    }

    pthread_t *file_writer = (pthread_t *) malloc(sizeof(pthread_t));
    file_write_params *file_write_data = (file_write_params *) malloc(sizeof(file_write_params
));
    file_write_data->memory_region = memory_region;
    pthread_create(file_writer, NULL, file_write_thread, (void *) file_write_data);

    //file reading params
    file_read_params *file_read_data = (file_read_params *) malloc(NUM_THREADS_FILE * sizeof(f
ile_read_params));
    pthread_t *file_readers = (pthread_t*) malloc(NUM_THREADS_FILE * sizeof(pthread_t));

    for(int i=0; i<NUM_THREADS_FILE; i++){
        file_read_data[i].thread_number=i;
        file_read_data[i].file_number = i % FULL_FILES_COUNT;
        pthread_create(&(file_readers[i]), NULL, file_read_thread, &(file_read_data[i]));
    }

    sem_wait(&main_sem);
    printf("Press Enter to interrupt infinite loop\n");
    getchar();
    printf("Waiting for all threads finish\n");
    cycle_stop=1;

    //wait for all threads
    for(int i=0; i<NUM_THREADS_MEMORY; i++){
        pthread_join(memory_fillers[i], NULL);
    }
    pthread_join(*file_writer, NULL);
    for(int i=0; i<NUM_THREADS_FILE; i++){
        pthread_join(file_readers[i], NULL);
    }

    //free memory
    free(memory_fillers);
    free(memory_data);
    free(file_writer);
    free(file_write_data);
    free(file_read_data);
    free(file_readers);

    sem_destroy(&main_sem);
    for(int i=0; i<=FULL_FILES_COUNT; i++){
        sem_destroy(&semaphore_file[i]);
    }
```

```
    close(fd);
}


int main() {
    printf("Pause before memory allocation. Press Enter to continue\n");
    getchar();
    printf("Memory allocation...\n");

    int *memory_region = mmap((void*) FIRST_ADDRESS, MEM_SIZE, PROT_READ | PROT_WRITE, MAP_ANO
NYMOUS | MAP_PRIVATE, -1, 0);

    printf("Pause after memory allocation. Press Enter to continue\n");
    getchar();

    printf("Memory filling...\n");
    fill_memory(memory_region);

    printf("Pause after memory filling. Press Enter to continue\n");
    getchar();
    avg = (float)sum / (float)count;
    printf("Average value is %f\n", avg);

    munmap((void*) FIRST_ADDRESS, MEM_SIZE);
    return 0;
}
```

Замер характеристик

Замер ОП с помощью free

1. До аллокации

```
               total        used        free      shared  buff/cache   available
Mem:        19951364      950964    16724772        5120     2275628    18634968
Swap:        4194300           0     4194300
```

2. После аллокации

```
               total        used        free      shared  buff/cache   available
Mem:        19951364      951132    16724584        5140     2275648    18634784
Swap:        4194300           0     4194300
```

3. После заполнения участка данными

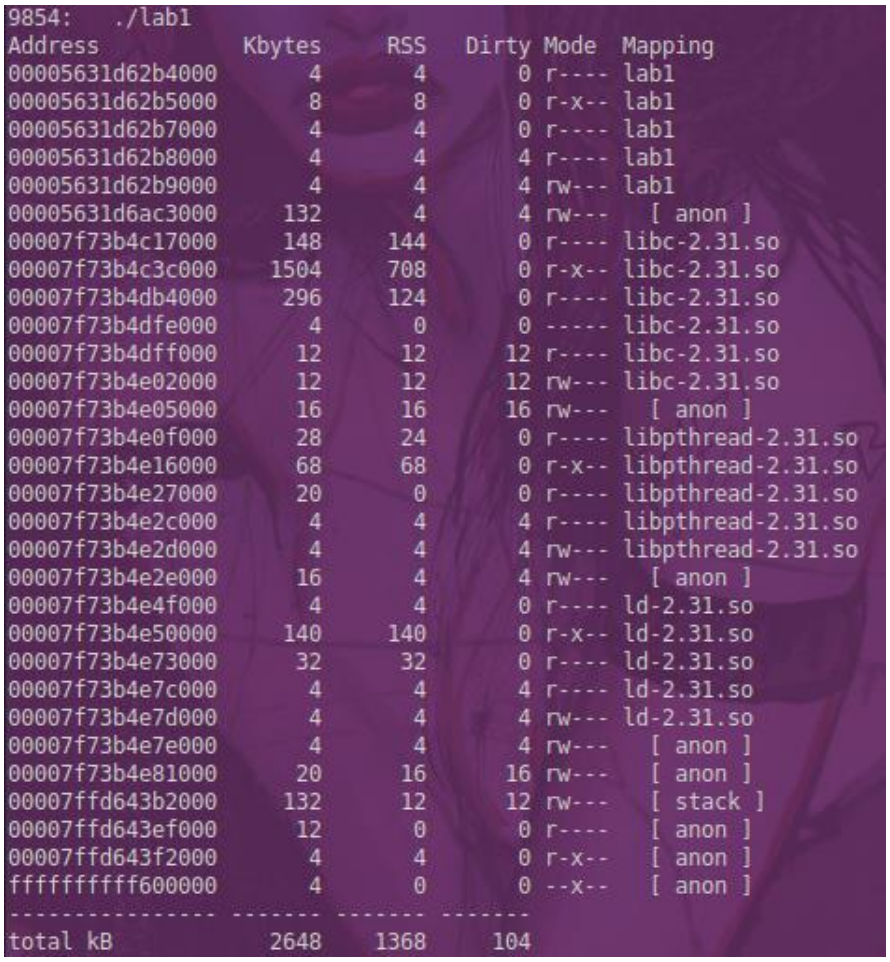
```
               total        used        free      shared  buff/cache   available
Mem:        19951364     1259476    16416160        5164     2275728    18326408
Swap:        4194300           0     4194300
```

4. После деаллокации

```
               total        used        free      shared  buff/cache   available
Mem:        19951364      950988    16724624        5188     2275752    18634880
Swap:        4194300           0     4194300
```

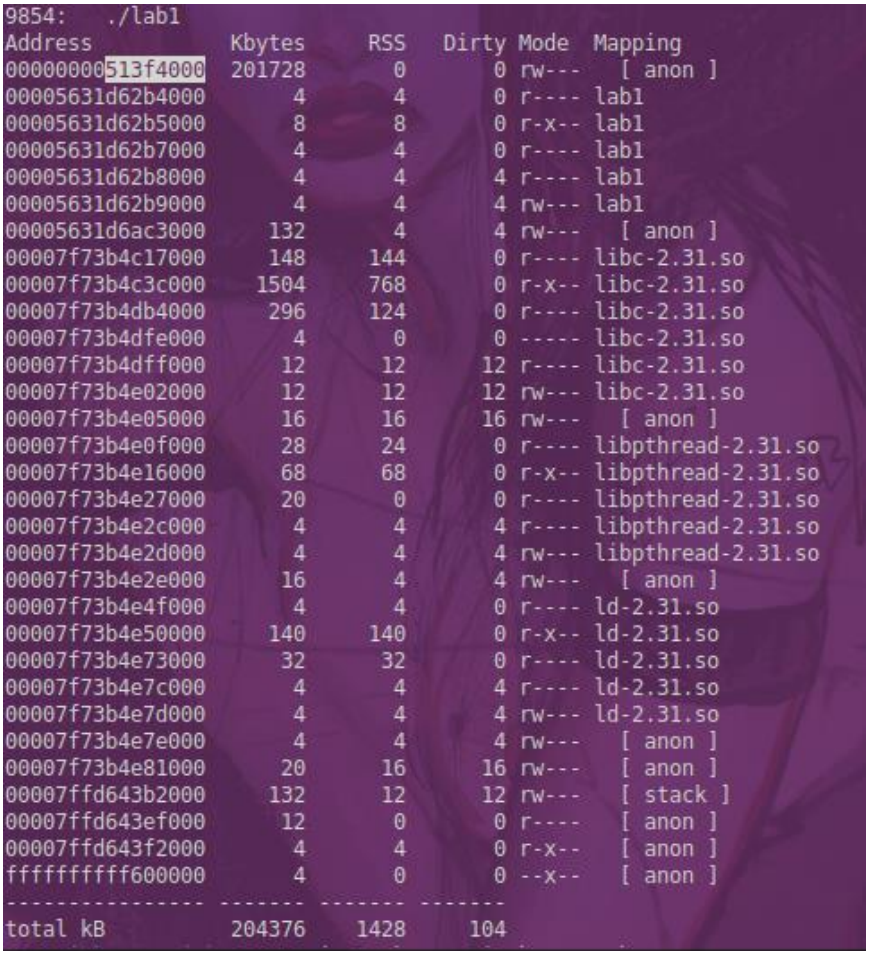
Исследование ОП с помощью pmap

1. До аллокации



```
9854:   ./lab1
Address           Kbytes     RSS   Dirty Mode  Mapping
00005631d62b4000       4       4       0 r---- lab1
00005631d62b5000       8       8       0 r-x-- lab1
00005631d62b7000       4       4       0 r---- lab1
00005631d62b8000       4       4       4 r---- lab1
00005631d62b9000       4       4       4 rw--- lab1
00005631d6ac3000     132       4       4 rw---   [ anon ]
00007f73b4c17000     148     144       0 r---- libc-2.31.so
00007f73b4c3c000    1504     708       0 r-x-- libc-2.31.so
00007f73b4db4000     296     124       0 r---- libc-2.31.so
00007f73b4dfe000       4       0       0 ----- libc-2.31.so
00007f73b4dff000      12      12      12 r---- libc-2.31.so
00007f73b4e02000      12      12      12 rw--- libc-2.31.so
00007f73b4e05000      16      16      16 rw---   [ anon ]
00007f73b4e0f000      28      24       0 r---- libpthread-2.31.so
00007f73b4e16000      68      68       0 r-x-- libpthread-2.31.so
00007f73b4e27000      20       0       0 r---- libpthread-2.31.so
00007f73b4e2c000       4       4       4 r---- libpthread-2.31.so
00007f73b4e2d000       4       4       4 rw--- libpthread-2.31.so
00007f73b4e2e000      16       4       4 rw---   [ anon ]
00007f73b4e4f000       4       4       0 r---- ld-2.31.so
00007f73b4e50000     140     140       0 r-x-- ld-2.31.so
00007f73b4e73000      32      32       0 r---- ld-2.31.so
00007f73b4e7c000       4       4       4 r---- ld-2.31.so
00007f73b4e7d000       4       4       4 rw--- ld-2.31.so
00007f73b4e7e000       4       4       4 rw---   [ anon ]
00007f73b4e81000      20      16      16 rw---   [ anon ]
00007ffd643b2000     132      12      12 rw---   [ stack ]
00007ffd643ef000      12       0       0 r----   [ anon ]
00007ffd643f2000       4       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
total kB            2648    1368     104
```

2. После аллокации



```
9854:   ./lab1
Address           Kbytes     RSS   Dirty Mode  Mapping
00000000513f4000  201728       0       0 rw---   [ anon ]
00005631d62b4000       4       4       0 r---- lab1
00005631d62b5000       8       8       0 r-x-- lab1
00005631d62b7000       4       4       0 r---- lab1
00005631d62b8000       4       4       4 r---- lab1
00005631d62b9000       4       4       4 rw--- lab1
00005631d6ac3000     132       4       4 rw---   [ anon ]
00007f73b4c17000     148     144       0 r---- libc-2.31.so
00007f73b4c3c000    1504     768       0 r-x-- libc-2.31.so
00007f73b4db4000     296     124       0 r---- libc-2.31.so
00007f73b4dfe000       4       0       0 ----- libc-2.31.so
00007f73b4dff000      12      12      12 r---- libc-2.31.so
00007f73b4e02000      12      12      12 rw--- libc-2.31.so
00007f73b4e05000      16      16      16 rw---   [ anon ]
00007f73b4e0f000      28      24       0 r---- libpthread-2.31.so
00007f73b4e16000      68      68       0 r-x-- libpthread-2.31.so
00007f73b4e27000      20       0       0 r---- libpthread-2.31.so
00007f73b4e2c000       4       4       4 r---- libpthread-2.31.so
00007f73b4e2d000       4       4       4 rw--- libpthread-2.31.so
00007f73b4e2e000      16       4       4 rw---   [ anon ]
00007f73b4e4f000       4       4       0 r---- ld-2.31.so
00007f73b4e50000     140     140       0 r-x-- ld-2.31.so
00007f73b4e73000      32      32       0 r---- ld-2.31.so
00007f73b4e7c000       4       4       4 r---- ld-2.31.so
00007f73b4e7d000       4       4       4 rw--- ld-2.31.so
00007f73b4e7e000       4       4       4 rw---   [ anon ]
00007f73b4e81000      20      16      16 rw---   [ anon ]
00007ffd643b2000     132      12      12 rw---   [ stack ]
00007ffd643ef000      12       0       0 r----   [ anon ]
00007ffd643f2000       4       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
total kB          204376    1428     104
```

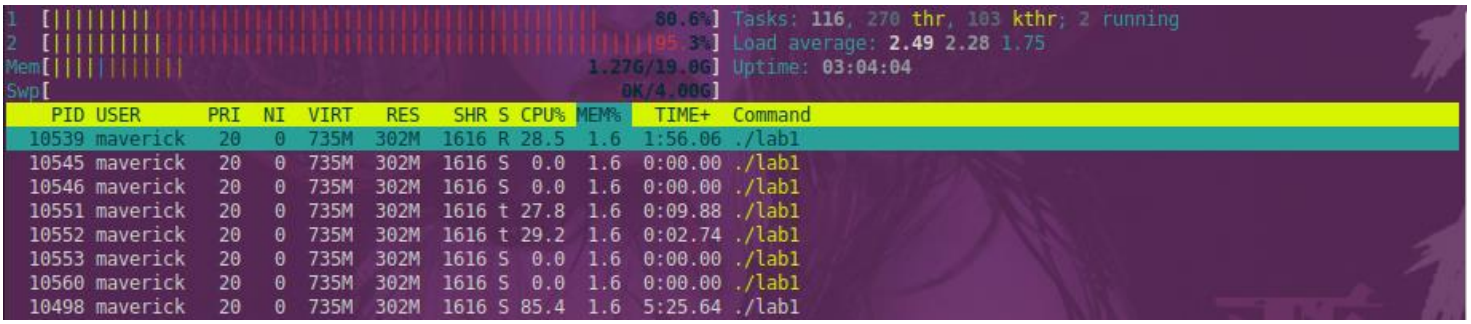
3. После заполнения участка данными

```
9854:   ./lab1
Address           Kbytes     RSS   Dirty Mode  Mapping
00000000513f4000  201728  201728  201728 rw---   [ anon ]
00005631d62b4000       4       4       0 r---- lab1
00005631d62b5000       8       8       0 r-x-- lab1
00005631d62b7000       4       4       0 r---- lab1
00005631d62b8000       4       4       4 r---- lab1
00005631d62b9000       4       4       4 rw--- lab1
00005631d6ac3000     132      24      24 rw---   [ anon ]
00007f738c000000   26628   26628   26628 rw---   [ anon ]
00007f738da01000   38908       0       0 -----   [ anon ]
00007f73937ff000       4       0       0 -----   [ anon ]
00007f7393800000    8192       8       8 rw---   [ anon ]
00007f7394000000     132       4       4 rw---   [ anon ]
00007f7394021000   65404       0       0 -----   [ anon ]
00007f73983de000       4       0       0 -----   [ anon ]
00007f73983df000    8192       8       8 rw---   [ anon ]
00007f7398bdf000       4       0       0 -----   [ anon ]
00007f7398be0000    8192       8       8 rw---   [ anon ]
00007f73993e0000       4       0       0 -----   [ anon ]
00007f73993e1000    8192       8       8 rw---   [ anon ]
00007f73a8000000   26628   26628   26628 rw---   [ anon ]
00007f73a9a01000   38908       0       0 -----   [ anon ]
00007f73b0000000   26628   26628   26628 rw---   [ anon ]
00007f73b1a01000   38908       0       0 -----   [ anon ]
00007f73b4c17000     148     144       0 r---- libc-2.31.so
00007f73b4c3c000    1504    1020       0 r-x-- libc-2.31.so
00007f73b4db4000     296     124       0 r---- libc-2.31.so
00007f73b4dfe000       4       0       0 ----- libc-2.31.so
00007f73b4dff000      12      12      12 r---- libc-2.31.so
00007f73b4e02000      12      12      12 rw--- libc-2.31.so
00007f73b4e05000      16      16      16 rw---   [ anon ]
00007f73b4e0f000      28      24       0 r---- libpthread-2.31.so
00007f73b4e16000      68      68       0 r-x-- libpthread-2.31.so
00007f73b4e27000      20       0       0 r---- libpthread-2.31.so
00007f73b4e2c000       4       4       4 r---- libpthread-2.31.so
00007f73b4e2d000       4       4       4 rw--- libpthread-2.31.so
00007f73b4e2e000      16       4       4 rw---   [ anon ]
00007f73b4e4f000       4       4       0 r---- ld-2.31.so
00007f73b4e50000     140     140       0 r-x-- ld-2.31.so
00007f73b4e73000      32      32       0 r---- ld-2.31.so
00007f73b4e7c000       4       4       4 r---- ld-2.31.so
00007f73b4e7d000       4       4       4 rw--- ld-2.31.so
00007f73b4e7e000       4       4       4 rw---   [ anon ]
00007f73b4e81000      20      16      16 rw---   [ anon ]
00007ffd643b2000     132      12      12 rw---   [ stack ]
00007ffd643ef000      12       0       0 r----   [ anon ]
00007ffd643f2000       4       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
total kB          499304  283348  281772
```

Замеры времени с помощью time ./lab1

```
real    1m0.999s
user    0m15.121s
sys     0m49.797s
```

Взгляд на запущенные потоки через htop



Скриншот был создан, когда отрабатывали потоки чтения из файлов.

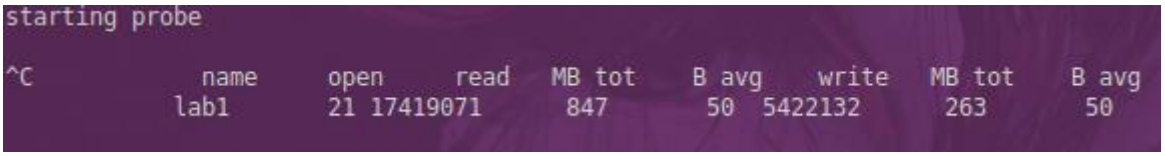
Замер времени для ввода-вывода (sudo strace -c -fp [pid])

% time	seconds	usecs/call	calls	errors	syscall
85.45	4430.585617	144691	30621	10294	futex
10.94	567.464420	24	23521125		pread64
3.50	181.290565	22	8100768		pwrite64
0.09	4.670943	1167735	4		read
0.01	0.281788	4858	58		set_robust_list
0.00	0.240551	1955	123		write
0.00	0.089713	1546	58		madvise
0.00	0.055006	873	63	1	munmap
0.00	0.014784	254	58		clone
0.00	0.012890	415	31		openat
0.00	0.002356	35	66		mmap
0.00	0.002107	30	68		mprotect
0.00	0.001103	35	31		close
100.00	5184.711843		31653074	10295	total

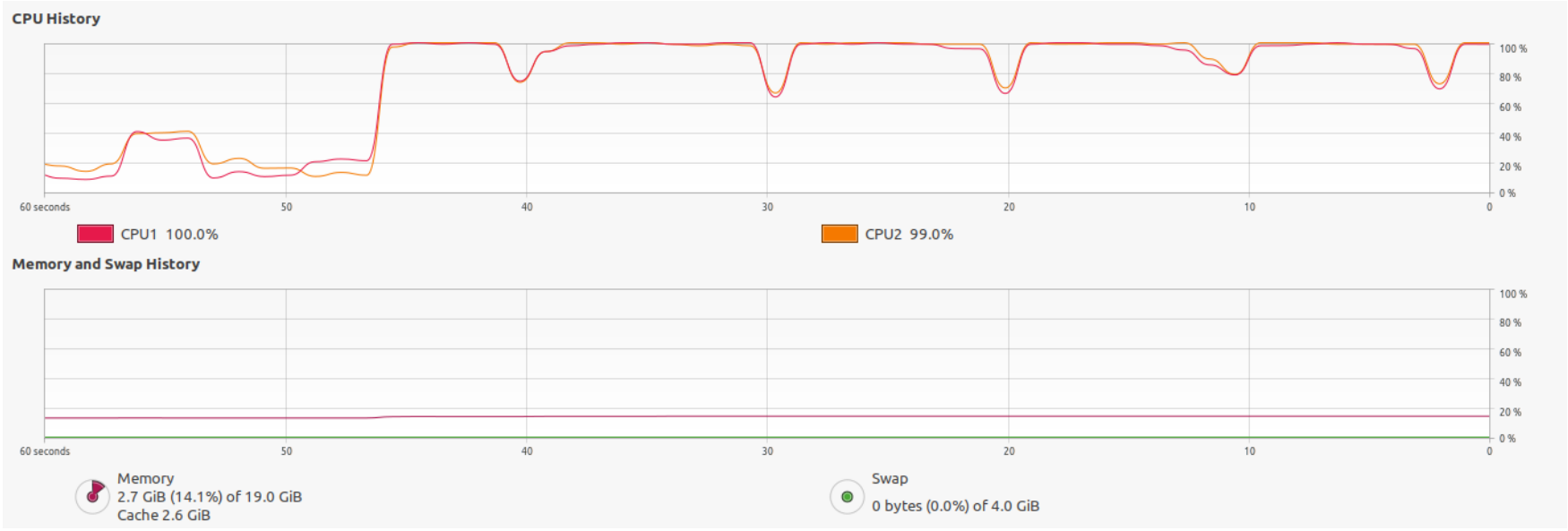
Трасса системных вызовов

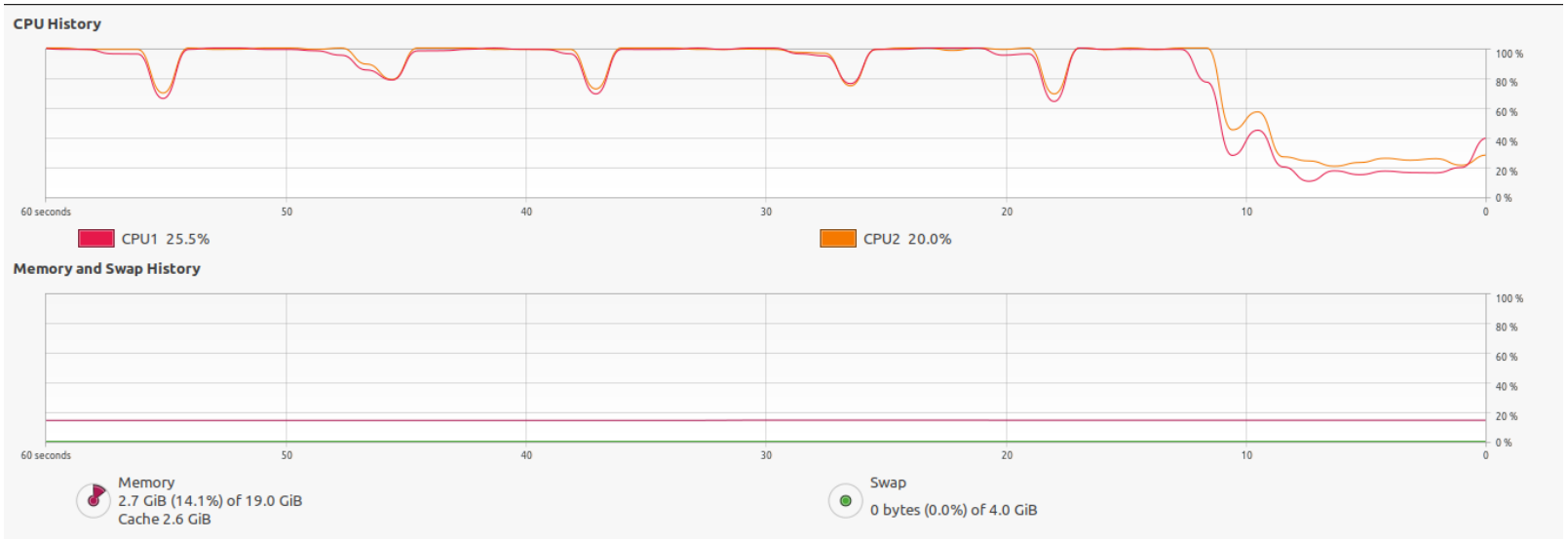
```
strace: Process 12281 attached
read(0, "\n", 1024)                     = 1
write(1, "Memory allocation...\n", 21)  = 21
mmap(0x513f4598, 206569472, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x513f4000
write(1, "Pause after memory allocation. P"..., 55) = 55
read(0, "\n", 1024)                     = 1
write(1, "Memory filling...\n", 18)     = 18
openat(AT_FDCWD, "/dev/urandom", O_RDONLY) = 3
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f881a16e000
mprotect(0x7f881a16f000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7f881a96dfb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTIDstrace: Process 12288 attache
d
 <unfinished ...>
[pid 12288] set_robust_list(0x7f881a96e9e0, 24 <unfinished ...>
[pid 12281] <... clone resumed>, parent_tid=[12288], tls=0x7f881a96e700, child_tidptr=0x7f881a96e9d0) = 12288
[pid 12288] <... set_robust_list resumed>) = 0
[pid 12281] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid 12288] write(1, "[Generator 0] start\n", 20 <unfinished ...>
[pid 12281] <... mmap resumed>)         = 0x7f881996d000
[pid 12288] <... write resumed>)        = 20
[pid 12281] mprotect(0x7f881996e000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid 12288] pread64(3,  <unfinished ...>
[pid 12281] <... mprotect resumed>)     = 0
[pid 12281] clone(child_stack=0x7f881a16cfb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTIDstrace: Process 1
2289 attached
 <unfinished ...>
[pid 12289] set_robust_list(0x7f881a16d9e0, 24 <unfinished ...>
[pid 12281] <... clone resumed>, parent_tid=[12289], tls=0x7f881a16d700, child_tidptr=0x7f881a16d9d0) = 12289
[pid 12289] <... set_robust_list resumed>) = 0
[pid 12281] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid 12289] write(1, "[Generator 1] start\n", 20 <unfinished ...>
[pid 12281] <... mmap resumed>)         = 0x7f881916c000
[pid 12289] <... write resumed>)        = 20
[pid 12281] mprotect(0x7f881916d000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid 12289] pread64(3,  <unfinished ...>
[pid 12281] <... mprotect resumed>)     = 0
[pid 12281] clone(child_stack=0x7f881996bfb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[122
90], tls=0x7f881996c700, child_tidptr=0x7f881996c9d0) = 12290
[pid 12281] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f881896b000
[pid 12281] mprotect(0x7f881896c000, 8388608, PROT_READ|PROT_WRITE) = 0
[pid 12281] clone(child_stack=0x7f881916afb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[122
91], tls=0x7f881916b700, child_tidptr=0x7f881916b9d0) = 12291
[pid 12281] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f881816a000
[pid 12281] mprotect(0x7f881816b000, 8388608, PROT_READ|PROT_WRITE) = 0
[pid 12281] clone(child_stack=0x7f8818969fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[122
```

Сбор информации о процессах с помощью stap



Графические показания потребления ресурсов компьютера





Вывод

В ходе работы я ознакомился со взаимодействием программы на Си и операционной систем (семафоры, работа с файлами) и средствами мониторинга операционной системы.