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

Группы P33211

Просолович М.А.

Тайц Ю.М.

Преподаватель

Покид Александр Владимирович

Санкт-Петербург 2020

Задание

# Лабораторная работа №1

Вариант:

A=276;B=0x33F2678F;C=mmap;D=77;E=37;F=block;G=120;H=random;I=91;J=max;K=cv

Разработать программу на языке С, 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.

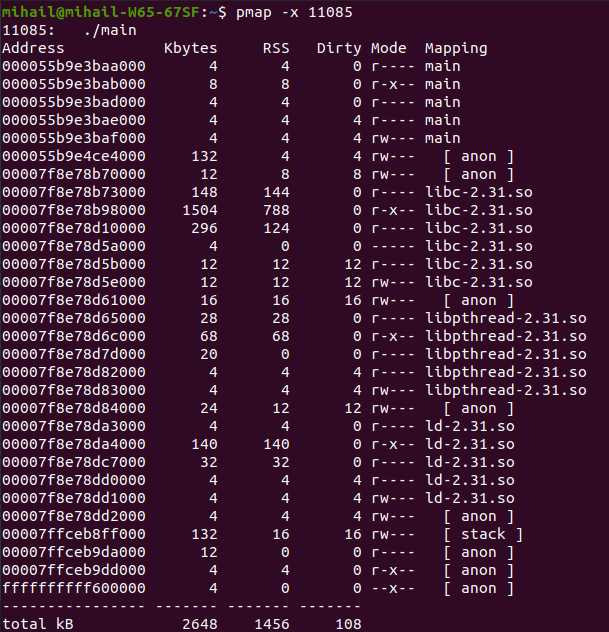
Используя stap построить графики системных характеристик.

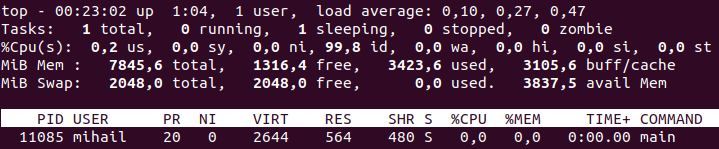
Код программы

#include <pthread.h>  
#include <sys/mman.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
//A=276;B=0x33F2678F;C=mmap;D=77;E=37;F=block;G=120;H=random;I=91;J=max;K=cv  
#define ALLOCATED\_MEMORY\_SIZE\_MB 276  
#define START\_ADDRESS 0x33F2678F  
#define WRITE\_IN\_MEMORY\_THREADS\_CNT 77  
#define FILES\_SIZE\_MB 37  
#define IO\_BUFFER\_SIZE\_B 120  
#define READ\_FILES\_THREADS\_CNT 91  
  
#define INT\_MIN -2147483648  
  
typedef struct {  
 FILE\* src;  
 int length;  
 unsigned char\* start;  
} WriteInMemoryProps;  
  
typedef struct {  
 unsigned char\* src;  
 char\* fileName;  
 pthread\_mutex\_t\* mutex;  
 pthread\_cond\_t\* cv;  
} WriteInFileProps;  
  
typedef struct {  
 char\* fileName;  
 pthread\_mutex\_t\* mutex;  
 pthread\_cond\_t\* cv;  
} ReadFileProps;  
  
void\* writeInMemory(void\* props) {  
 WriteInMemoryProps\* data = props;  
 unsigned char\* start = data->start;  
 int length = data->length;  
 FILE\* src = data->src;  
 size\_t i = 0;  
 while (i < length) {  
 i += fread(start + i, 1, length - i, src);  
 }  
 return NULL;  
}  
  
//writing data from /dev/urandom in memory beginning from start address  
void writeInMemoryFromUrandom(unsigned char\* startAddress) {  
 FILE\* urandom = fopen("/dev/urandom", "r");  
  
 int wrn\_part = ALLOCATED\_MEMORY\_SIZE\_MB \* 1024 \* 1024 / WRITE\_IN\_MEMORY\_THREADS\_CNT;  
  
 pthread\_t writeInMemoryThreads[WRITE\_IN\_MEMORY\_THREADS\_CNT];  
  
 for (int i = 0; i < WRITE\_IN\_MEMORY\_THREADS\_CNT - 1; i++) {  
 WriteInMemoryProps\* props = malloc(sizeof(WriteInMemoryProps));  
 props->start = startAddress;  
 props->length = wrn\_part;  
 props->src = urandom;  
 pthread\_create(&(writeInMemoryThreads[i]), NULL, writeInMemory, props);  
 startAddress += wrn\_part;  
 }  
  
 WriteInMemoryProps\* lastThreadProps = malloc(sizeof(WriteInMemoryProps));  
 lastThreadProps->start = startAddress;  
 lastThreadProps->length = wrn\_part + ALLOCATED\_MEMORY\_SIZE\_MB \* 1024 \* 1024 % WRITE\_IN\_MEMORY\_THREADS\_CNT;  
 lastThreadProps->src = urandom;  
 pthread\_create(&(writeInMemoryThreads[WRITE\_IN\_MEMORY\_THREADS\_CNT - 1]), NULL, writeInMemory, lastThreadProps);  
  
 for (int i = 0; i < WRITE\_IN\_MEMORY\_THREADS\_CNT; i++)  
 pthread\_join(writeInMemoryThreads[i], NULL);  
  
 fclose(urandom);  
}  
  
\_Noreturn void\* infiniteGenerating(void\* startAddress) {  
 while (1) {  
 writeInMemoryFromUrandom(startAddress);  
 }  
}  
  
double generateRandom() {  
 return (double)rand() / (double)RAND\_MAX ;  
}  
  
void writeInFile(unsigned char\* src, char\* fileName, pthread\_mutex\_t\* mutex, pthread\_cond\_t\* cv) {  
 pthread\_mutex\_lock(mutex);  
 FILE\* file = fopen(fileName, "wb");  
 int file\_size = FILES\_SIZE\_MB \* 1024 \* 1024;  
 for (int j = 0; j < file\_size / IO\_BUFFER\_SIZE\_B; j++) {  
 int blockNumber = (int) (generateRandom() \* file\_size / IO\_BUFFER\_SIZE\_B);  
 int blockSize = IO\_BUFFER\_SIZE\_B;  
 if (blockNumber == (file\_size / IO\_BUFFER\_SIZE\_B) && file\_size % IO\_BUFFER\_SIZE\_B != 0) {  
 //this is a last block, and we should adjust block size  
 blockSize = file\_size % IO\_BUFFER\_SIZE\_B;  
 }  
 fwrite(src + blockNumber \* IO\_BUFFER\_SIZE\_B, 1, blockSize, file);  
 }  
 fclose(file);  
 printf("Data generated for %s\n", fileName);  
 pthread\_cond\_broadcast(cv);  
 pthread\_mutex\_unlock(mutex);  
}  
  
\_Noreturn void\* writeFromMemoryToFile(void\* props) {  
 WriteInFileProps\* args = props;  
 while (1) {  
 writeInFile(args->src, args->fileName, args->mutex, args->cv);  
 }  
}  
  
int find\_max(int\* begin, int length) {  
 int max = INT\_MIN;  
 for (int\* i = begin; i < begin + length; i++) {  
 int val = \*i;  
 if (val > max)  
 max = val;  
 }  
 return max;  
}  
  
\_Noreturn void\* readFile(void\* props) {  
 ReadFileProps\* args = props;  
 char\* fileName = args->fileName;  
 while (1) {  
 int max = INT\_MIN;  
 pthread\_mutex\_lock(args->mutex);  
 printf("Waiting on cv %s \n", fileName);  
 pthread\_cond\_wait(args->cv, args->mutex);  
 FILE\* file = fopen(fileName, "rb");  
 unsigned char buf[IO\_BUFFER\_SIZE\_B];  
 int file\_size = FILES\_SIZE\_MB \* 1024 \* 1024;  
 for (int i = 0; i < file\_size / IO\_BUFFER\_SIZE\_B; i++) {  
 int blockNumber = (int) (generateRandom() \* file\_size / IO\_BUFFER\_SIZE\_B);  
 int blockSize = IO\_BUFFER\_SIZE\_B;  
 if (blockNumber == (file\_size / IO\_BUFFER\_SIZE\_B) && file\_size % IO\_BUFFER\_SIZE\_B != 0) {  
 //this is a last block, and we should adjust block size  
 blockSize = file\_size % IO\_BUFFER\_SIZE\_B;  
 }  
 fseek(file, blockNumber \* IO\_BUFFER\_SIZE\_B, SEEK\_SET);  
 fread(&buf, 1, blockSize, file);  
 int localMax = find\_max((int\* ) &buf[0], IO\_BUFFER\_SIZE\_B / 4);  
 if (localMax > max)  
 max = localMax;  
 }  
 printf("Max in %s: %d\n", fileName, max);  
 fclose(file);  
 pthread\_mutex\_unlock(args->mutex);  
 }  
}  
  
unsigned char\* allocate\_memory() {  
 return mmap((void\* ) START\_ADDRESS, ALLOCATED\_MEMORY\_SIZE\_MB \* 1024 \* 1024, PROT\_READ | PROT\_WRITE,  
 MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);  
}  
  
void allocateDeallocateMemory() {  
 printf("Before memory allocation.");  
 getchar();  
 unsigned char\* allocatedStartAddress = allocate\_memory();  
 printf("After memory allocation.");  
 getchar();  
 writeInMemoryFromUrandom(allocatedStartAddress);  
 printf("After memory filling.");  
 getchar();  
 munmap(allocatedStartAddress, ALLOCATED\_MEMORY\_SIZE\_MB \* 1024 \* 1024);  
 printf("After memory deallocation.");  
 getchar();  
}  
  
char\* fileName;  
char\* getFileNameByNumber(int fileNumber) {  
 fileName = malloc(sizeof(char) \* 11);  
 snprintf(fileName, 11, "file\_%d.bin", fileNumber);  
 return fileName;  
}  
  
int main() {  
 allocateDeallocateMemory();  
  
 unsigned char\* allocatedStartAddress = allocate\_memory();  
  
 const unsigned int filesCnt = ALLOCATED\_MEMORY\_SIZE\_MB / FILES\_SIZE\_MB + 1;  
 pthread\_mutex\_t mutexes[filesCnt];  
 pthread\_cond\_t cvs[filesCnt];  
 for (int i = 0; i < filesCnt; i++) {  
 pthread\_mutex\_init(&(mutexes[i]), NULL);  
 pthread\_cond\_init(&(cvs[i]), NULL);  
 }  
  
 pthread\_t readThreads[READ\_FILES\_THREADS\_CNT];  
 int fileNumber = 0;  
 for (int i = 0; i < READ\_FILES\_THREADS\_CNT; i++) {  
 if (fileNumber == filesCnt)  
 fileNumber = 0;  
 ReadFileProps\* props = malloc(sizeof(ReadFileProps));  
 props->fileName = getFileNameByNumber(fileNumber);  
 props->mutex = &(mutexes[fileNumber]);  
 props->cv = &(cvs[fileNumber]);  
 pthread\_create(&readThreads[i], NULL, readFile, props);  
 fileNumber++;  
 }  
  
 pthread\_t writeInMemory;  
 pthread\_create(&writeInMemory, NULL, infiniteGenerating, allocatedStartAddress);  
  
 pthread\_t fileThreads[filesCnt];  
 for (int i = 0; i < filesCnt; i++) {  
 WriteInFileProps\* props = malloc(sizeof(WriteInFileProps));  
 props->src = allocatedStartAddress;  
 props->fileName = getFileNameByNumber(i);  
 props->mutex = &(mutexes[i]);  
 props->cv = &(cvs[i]);  
 pthread\_create(&fileThreads[i], NULL, writeFromMemoryToFile, props);  
 }  
  
 for (int i = 0; i < READ\_FILES\_THREADS\_CNT; i++)  
 pthread\_join(readThreads[i], NULL);  
 pthread\_join(writeInMemory, NULL);  
 for (int i = 0; i < filesCnt; i++) {  
 pthread\_join(fileThreads[i], NULL);  
 }  
  
 return 0;  
}

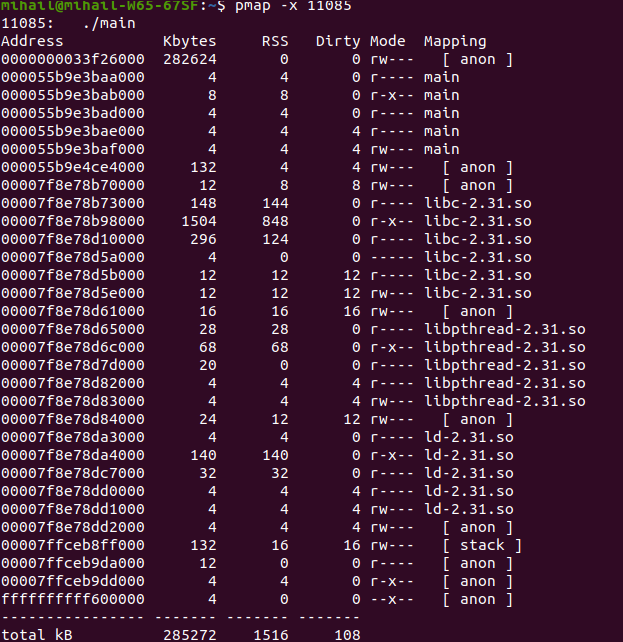
Сбор статистики

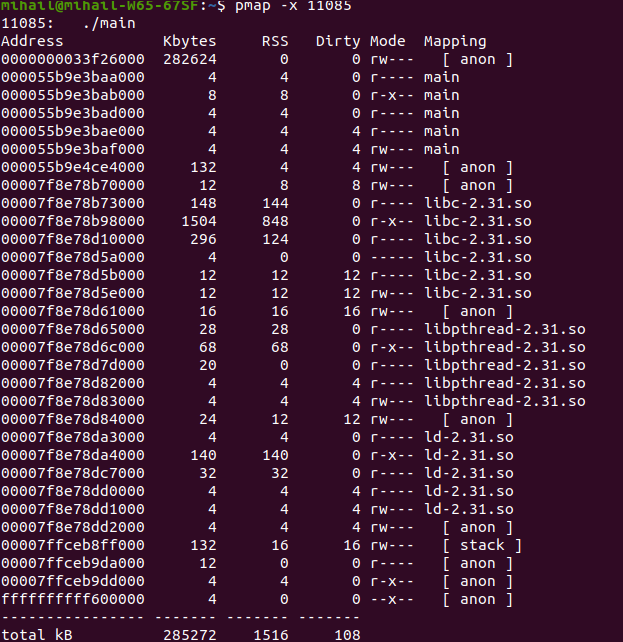
Before allocation

**

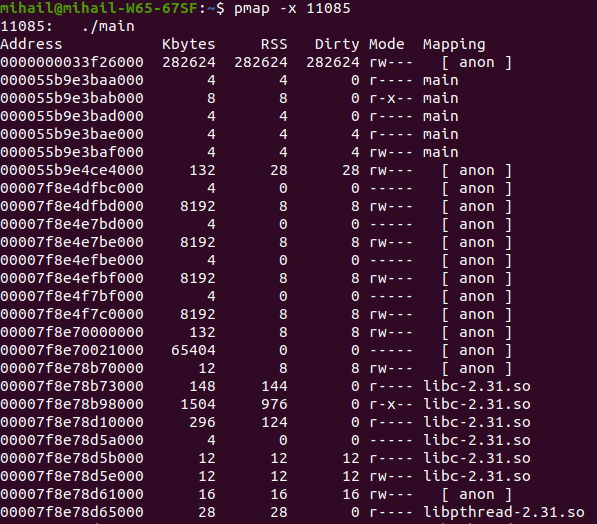


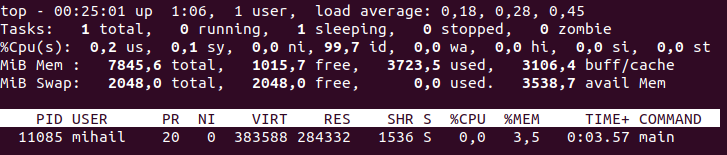
After allocation



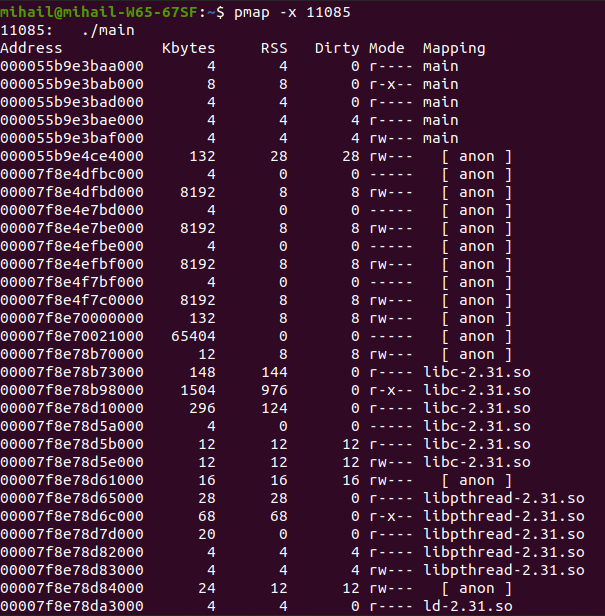
**

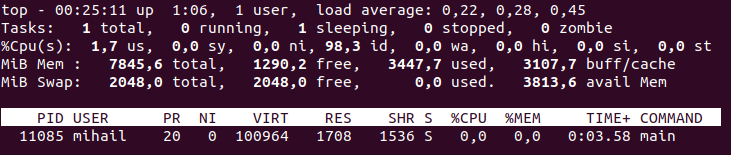
After filling

**

**

After deallocation



**

Strace:

execve("./main", ["./main"], 0x7ffff5e1ed40 /\* 60 vars \*/) = 0

brk(NULL) = 0x557655443000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffd5fa6670) = -1 EINVAL (Invalid argument)

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=98934, ...}) = 0

mmap(NULL, 98934, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f9920025000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=157224, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f9920023000

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f9920000000

mmap(0x7f9920007000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f9920007000

mmap(0x7f9920018000, 20480, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7f9920018000

mmap(0x7f992001d000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7f992001d000

mmap(0x7f992001f000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f992001f000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f991fe0e000

mprotect(0x7f991fe33000, 1847296, PROT\_NONE) = 0

mmap(0x7f991fe33000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f991fe33000

mmap(0x7f991ffab000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f991ffab000

mmap(0x7f991fff6000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f991fff6000

mmap(0x7f991fffc000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f991fffc000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f991fe0b000

arch\_prctl(ARCH\_SET\_FS, 0x7f991fe0b740) = 0

mprotect(0x7f991fff6000, 12288, PROT\_READ) = 0

mprotect(0x7f992001d000, 4096, PROT\_READ) = 0

mprotect(0x557654d93000, 4096, PROT\_READ) = 0

mprotect(0x7f992006b000, 4096, PROT\_READ) = 0

munmap(0x7f9920025000, 98934) = 0

set\_tid\_address(0x7f991fe0ba10) = 16150

set\_robust\_list(0x7f991fe0ba20, 24) = 0

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f9920007bf0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7f99200153c0}, NULL, 8) = 0

clone(child\_stack=0x7f991ee07fb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[16153], tls=0x7f991ee08700, child\_tidptr=0x7f991ee089d0) = 16153

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f991de07000

mprotect(0x7f991de08000, 8388608, PROT\_READ|PROT\_WRITE) = 0

clone(child\_stack=0x7f991e606fb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[16154], tls=0x7f991e607700, child\_tidptr=0x7f991e6079d0) = 16154

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f991d606000

mprotect(0x7f991d607000, 8388608, PROT\_READ|PROT\_WRITE) = 0

..............................................................

futex(0x7f991fe0a9d0, FUTEX\_WAIT, 16151, NULL) = 0

futex(0x7f991f6099d0, FUTEX\_WAIT, 16152, NULL) = 0

futex(0x7f991ee089d0, FUTEX\_WAIT, 16153, NULL) = 0

futex(0x7f991e6079d0, FUTEX\_WAIT, 16154, NULL) = 0

futex(0x7f991de069d0, FUTEX\_WAIT, 16155, NULL) = 0

munmap(0x7f991f60a000, 8392704) = 0

futex(0x7f991d6059d0, FUTEX\_WAIT, 16156, NULL) = 0

munmap(0x7f991ee09000, 8392704) = 0

futex(0x7f991ce049d0, FUTEX\_WAIT, 16157, NULL) = 0

munmap(0x7f991e608000, 8392704) = 0

futex(0x7f9917fff9d0, FUTEX\_WAIT, 16158, NULL) = 0

munmap(0x7f991de07000, 8392704) = 0

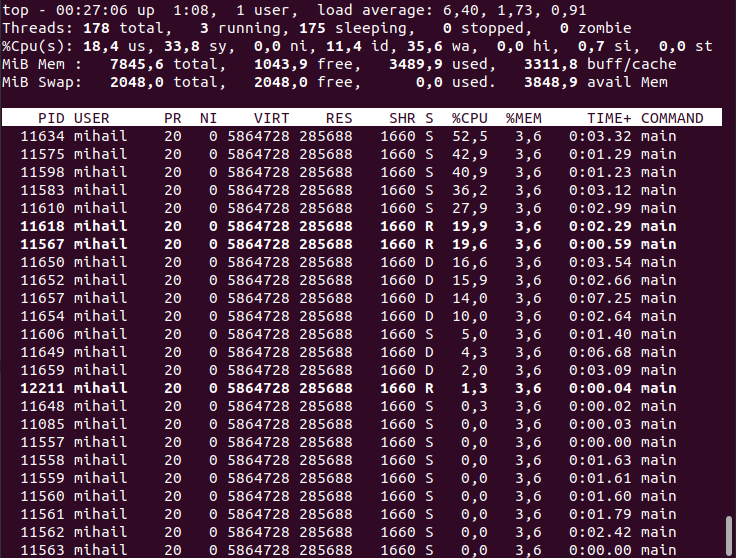
futex(0x7f99177fe9d0, FUTEX\_WAIT, 16159, NULL) = 0

munmap(0x7f991d606000, 8392704) = 0

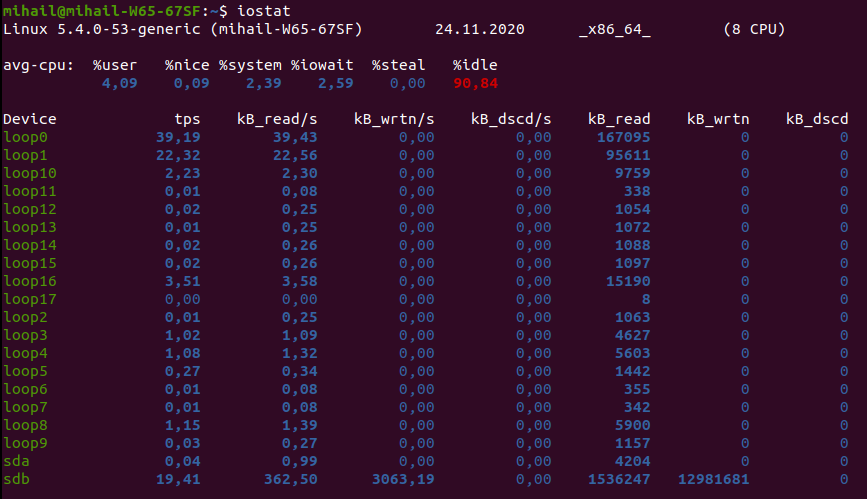
futex(0x7f9916ffd9d0, FUTEX\_WAIT, 16160, NULL) = 0

.............................................................................................................

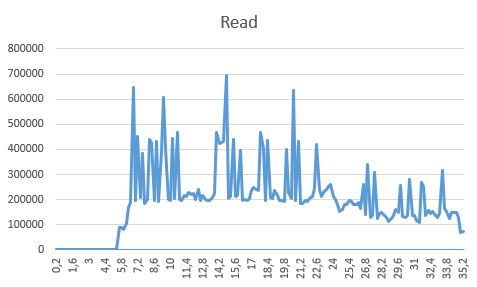
Загрузка процессора во время выполнения:

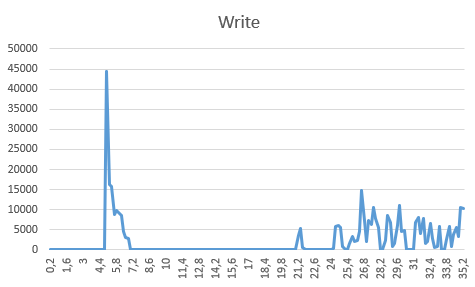


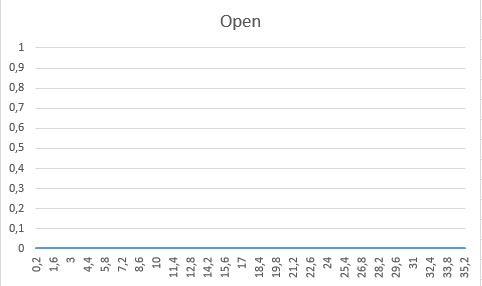
Статистика по вводу-выводу:

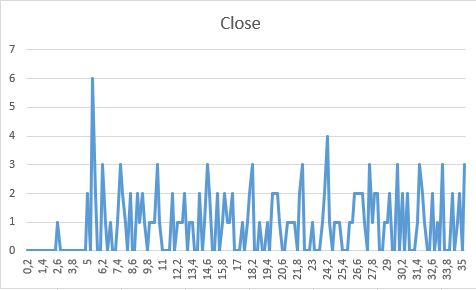


Графики системных вызовов, полученные с помощью stap.









Вывод

В процессе работы мы разработали многопоточную программу на языку С, поработали с примитивами синхронизации. Поставили виртуальную машину на Windows и узнали основные команды для мониторинга системы в Linux. Написали скрипт для systemtap, с помощью которого построили графики системных характеристик.