

Факультет программной инженерии и компьютерной техники

Операционные системы

Лабораторная работа №1

Вариант A=81;B=0x8230E78E;C=mmap;D=73;E=168;F=nocache;G=76;H=seq;I=137;J=sum;

K=futex

Преподаватель: Покид Александр Владимирович

Выполнил: Буланов Кирилл Сергеевич, Р33122

# Задание

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

##define \_GNU\_SOURCE

#include <sys/mman.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <linux/futex.h>

#include <syscall.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <math.h>

#define A 81

#define B 0x8230E78E

#define D 73

#define E 168

#define I 137

**typedef** **struct** fillMemoryArgs {

**void** \*startAddress;

size\_t memorySize;

FILE \*urandom;

} fillArgs;

**void** \*fillMemoryFromThread(**void** \*args) {

fillArgs \*inArgs = (fillArgs \*) args;

**int** c = fread((**void** \*) inArgs->startAddress, 1, inArgs->memorySize, inArgs->urandom);

**if**(c <= 0){

perror("Память не заполняется");

\_exit(1);

}

**return** 0;

}

**void** fillMemory(**void** \*startAddress, size\_t memorySize) {

startAddress = mmap(startAddress, memorySize, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANON, -1, 0);

**if**(startAddress == MAP\_FAILED){

perror("Не мапится");

\_exit(1);

}

FILE \*urandom = fopen("/dev/urandom", "r");

pthread\_t threads[D];

fillArgs args = {startAddress, memorySize, urandom};

**for**(**int** i = 0; i < D; ++i)

pthread\_create(&threads[i], NULL, fillMemoryFromThread, (**void** \*) &args);

**for**(**int** i = 0; i < D; ++i)

pthread\_join(threads[i], NULL);

fclose(urandom);

munmap(startAddress, memorySize);

}

\_Noreturn **void** \*writeThread(**void** \*fut) {

**int** out = open("output", O\_CREAT | O\_WRONLY | O\_TRUNC | O\_DIRECT, 0700);

**if**(out == -1){

perror("Файл не открыть");

\_exit(1);

}

**while** (1) {

FILE \*urandom = fopen("/dev/urandom", "r");

**int** isURandomEmpty = 0;

**while** (!isURandomEmpty) {

syscall(SYS\_futex , (**int** \*) fut, FUTEX\_WAIT, 1, NULL, NULL, 0);

\*((**int** \*) fut) = 1;

**int** val;

**if** (fread(&val, **sizeof**(**int**) - 1, 1, urandom) < 0) {

isURandomEmpty = 1;

printf("Рандом пуст");

} **else** {

size\_t inputSize = floor(log10(abs(val)) + 2);

**char** str[inputSize];

sprintf(str, "%d ", abs(val));

**int** c = write(out, str, inputSize);

**if**(c <= 0){

perror("Нельзя записать в файл");

\_exit(1);

}

}

\*((**int** \*) fut) = 0;

syscall(SYS\_futex , (**int** \*) fut, FUTEX\_WAKE, 1, NULL, NULL, 0);

}

fclose(urandom);

}

close(out);

}

\_Noreturn **void** \*statisticsThread(**void** \*fut) {

**while**(1) {

FILE \*f = fopen("output", "r");

**if**(f == NULL){

**continue**;

}

syscall(SYS\_futex , (**int** \*) fut, FUTEX\_WAIT, 1, NULL, NULL, 0);

\*((**int** \*) fut) = 1;

**int** num;

**long** **long** sum = 0;

**while** (fscanf(f, "%d ", &num) > 0)

sum += (**long** **long**) num;

printf("Сумма: %lld\n", sum);

\*((**int** \*) fut) = 0;

syscall(SYS\_futex , (**int** \*) fut, FUTEX\_WAKE, 1, NULL, NULL, 0);

fclose(f);

}

}

**void** fillFile(**long** **long** fileSize) {

**int** futTemp = 1;

**int** \*fut = &futTemp;

pthread\_t wThread;

pthread\_create(&wThread, NULL, writeThread, (**void** \*) fut);

\*fut = 0;

pthread\_t statThreads[I];

**for** (**int** i = 0; i < I; ++i)

pthread\_create(&statThreads[i], NULL, statisticsThread, (**void** \*) fut);

pthread\_join(wThread, NULL);

**for** (**int** i = 0; i < I; i++)

pthread\_join(statThreads[i], NULL);

}

**int** main() {

fillMemory((**void** \*) B, A \* 1024 \* 1024);

fillFile(E \* 1024 \* 1024);

**return** 0;

}

# Ход выполнения

Замеры производились на Ubuntu 18.04. Для данных о заполнении памяти и потребления CPU использовалась утилита top, запущенная во время работы программы.

Заполнение памяти:

До аллокации — VIRT: 10264, RES: 1216

После аллокации — VIRT: 93208, RES: 1220

После заполнения участка данными — VIRT: 167072, RES: 84288

После деаллокации — VIRT: 167072, RES: 84288

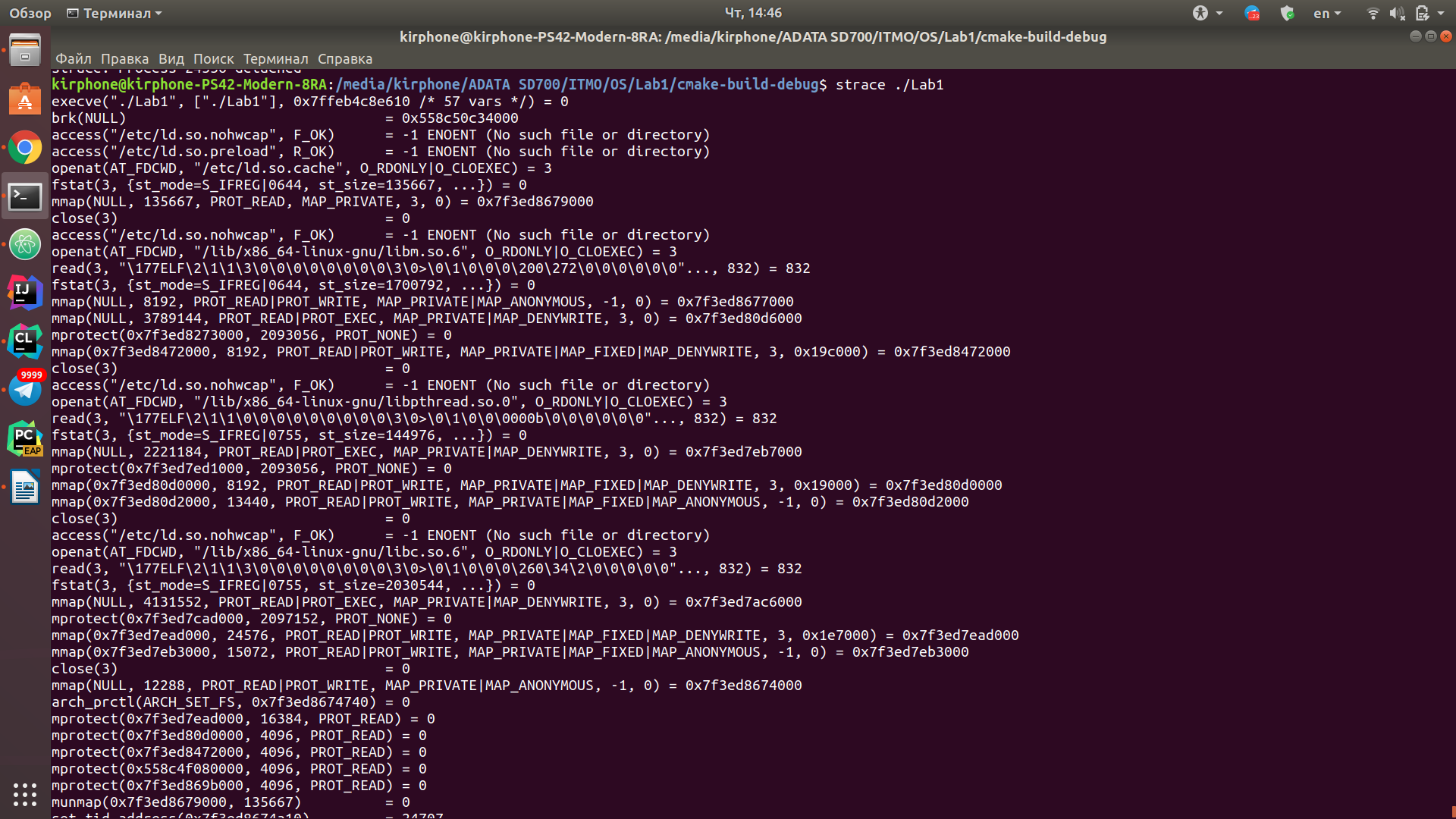
Процент загрузки процессора – 755%

Для теста скорости записи и чтения использовалась утилита iostat.

51 КБ/с чтение;

204 КБ/с запись

Для отслеживания системных вызовов используем strace. Запускаем её командой strace ./Lab1.



Stap скрипт

global countOfRead, countOfWrite  
global uticks, kticks, ticks  
  
probe begin{  
 printf("START\n")  
}  
  
probe vfs.read{  
 if(target() == pid()){  
 countOfRead++;  
 }  
}  
  
probe vfs.write{  
 if(target() == pid()){  
 countOfWrite++;  
 }  
}  
  
probe perf.sw.cpu\_clock!, timer.profile{  
 if(target() == pid()){  
 if (user\_mode())  
 uticks <<< 1  
 else  
 kticks <<< 1  
 }  
 ticks <<< 1  
}  
  
probe end{  
 printf("Количество вызовов чтения файла: %d\n", countOfRead)  
 printf("Количество вызовов записи в файл: %d\n", countOfWrite)  
 allticks = @count(ticks)  
 printf ("%7s %7s (из %d тактов)\n",  
 "пользовательские", "в режиме ядра", allticks)  
  
 uscaled = @count(uticks)\*10000/allticks  
 kscaled = @count(kticks)\*10000/allticks  
 printf ("%3d.%02d%% %3d.%02d%%\n",  
 uscaled/100, uscaled%100, kscaled/100, kscaled%100)  
 printf("\n")  
  
 delete uticks  
 delete kticks  
 delete ticks  
 delete countOfRead  
 delete countOfWrite  
}

Результат работы:

Количество вызовов чтения файла: 850074

Количество вызовов записи в файл: 134290

пользовательские в режиме ядра (из 1385620 тактов)

2.54% 12.31%

# Вывод

В процессе выполнения лабораторной работы я изучил основы многопоточного программирования на С, узнал способы работы с одной памятью в разных потоках, а также узнал, как собирать статистику по работе программы. Довольно интересно было узнать отличие практик многопоточной работы на C от других языков, особенно, языков более высокого уровня.