

Факультет программной инженерии и компьютерной техники

Операционные системы

Лабораторная работа №1

Вариант № A=130;B=0x126F82A;C=mmap;D=45;E=76;F=nocache;G=103;H=seq;I=84;J=min;K=flock

Преподаватель: Осипов Святослав Владимирович

Выполнил: Кульбако Артемий Юрьевич, Р33112



# 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 <limits.h>

#include <pthread.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <math.h>

#include <fcntl.h>

#define A 130

#define B 0x126F82A

#define C mmap

#define D 45

#define E 76

#define F nocache

#define G 103

#define H seq

#define I 84

#define J min

#define K flock

void\* testThread() { while(1) { printf("next =>\n"); } }

short countIntDigits(int *x*) { return floor(log10(*x*) + 1); }

void fillMemory(void\* *startAddress*, long long int *memorySize*) {

*// до аллокации*

    C((void\*) *startAddress*, *memorySize*, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

*// после аллокации*

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

    void\* threadFunc() { fread((void\*) startAddress, 1, memorySize, urandom); }

    pthread\_t thr;

    pthread\_create(&thr, NULL, threadFunc, NULL);

    pthread\_join(thr, NULL);

    fclose(urandom);

*// после заполнения участка данными*

    munmap((void\*) *startAddress*, *memorySize*);

*// после деаллокации*

}

void fillFile(long long int *fileSize*) {

    int restartWThread = 1;

    void\* writeThread() {

        printf("Write thread awaken\n");

*//fopen для файла с данными не подходит, по причинам: это не системная функция, нет возможности задать флаг, использовать буфер.*

        int flag = O\_WRONLY | O\_CREAT | O\_TRUNC | O\_DIRECT;

        int fd = open("resFile", flag, 0666);

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

*/\**

*Т.к. нам нужно вывести минимальное число из файла, то считываем int из urandom. Изначально кажется, чтобы узнать количество итераций нужно*

*(fileSize / sizeof(int)), но это ошибочный вариант, т.к. в файл будут писаться символы, а не int, а значит, к примеру, 1 и 111 будут весить*

*по разному, а не строго 4 байта. Следовательно, на каждой итерации необходимо подсчитать количество символом в рандомном полученном числе,*

*конвертировать int в char[], проверять не привысило ли сумма всех char[] вес файла, и если да - прервать цикл, иначе записать новые символы в файл.*

*P.S. Полученный файл может незначительно отличаться по размеру в большую сторону, т.к. ОС выделяет место под данные на диске секторами.*

*\*/*

        int weightOfSequence = 0;

        for (;;) {

            struct flock readLock;

            memset(&readLock, 0, sizeof(readLock));

            readLock.l\_type = F\_RDLCK;

            fcntl(fd, F\_SETLKW, &readLock);

            int val;

            fread(&val, sizeof(int), 1, urandom);

            short numberOfDigits = countIntDigits(val) + 1; *// + 1 для пробела: без этого, считывающие потоки не смогут разделить числа*

            weightOfSequence += numberOfDigits \* sizeof(char);

            if (weightOfSequence >= *fileSize*) break;

            else {

                char str[numberOfDigits];

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

                write(fd, str, numberOfDigits);

            }

            readLock.l\_type = F\_UNLCK;

            fcntl(fd, F\_SETLKW, &readLock);

        }

        close(*fd*);

        fclose(*urandom*);

        restartWThread = 1;

    }

    void\* agregateThread() {

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

        int frd = fileno(f);

        struct flock readLock;

        memset(&readLock, 0, sizeof(readLock));

        readLock.l\_type = F\_WRLCK;

        fcntl(frd, F\_SETLKW, &readLock);

        int num;

        int min = INT\_MAX;

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

            if (min > num) min = num;

        }

        readLock.l\_type = F\_UNLCK;

        fcntl(frd, F\_SETLKW, &readLock);

        fclose(f);

*//printf("The final min is %d\n", min); Выводить это нет смысла, т.к. чисел очень много, и почти сразу же будет получено INT\_MIN*

    }

    while (1) {

        if (restartWThread == 1) {

            restartWThread = 0;

            pthread\_t wThr;

            pthread\_create(&wThr, NULL, writeThread, NULL);

        }

        pthread\_t aThrs[I];

        for (int i = 0; i < I; i++) {

            pthread\_create(&aThrs[i], NULL, agregateThread, NULL);

        }

        for (int i = 0; i < I; i++) { *// здесь необходимо ждать завершения поток, иначе будет segmentation fault*

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

        }

    }

}

int main() {

    fillMemory((void \*) B, 130 \* 1024 \* 1024);

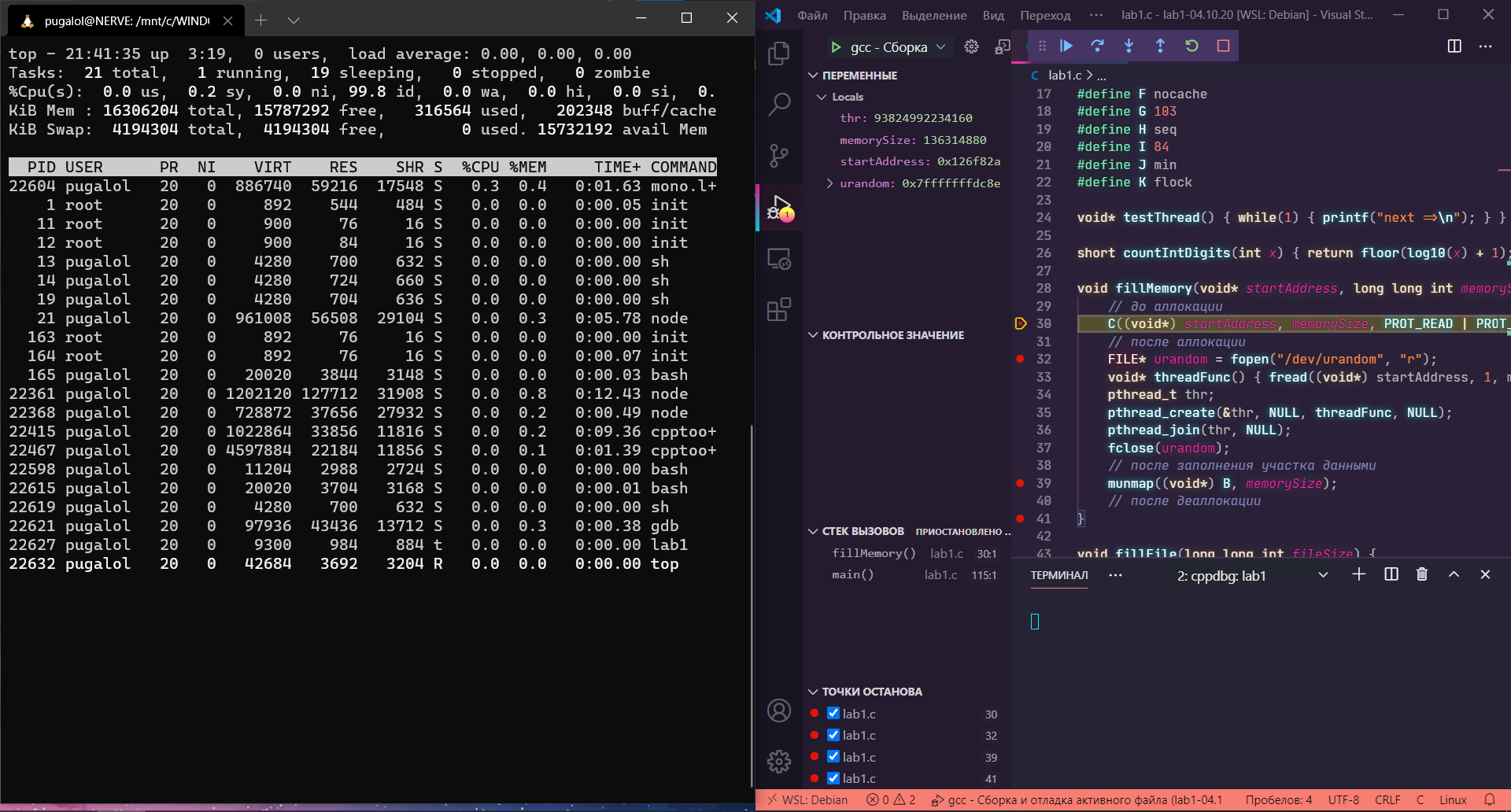
    fillFile(E \* 1024 \* 1024);

    return 0;

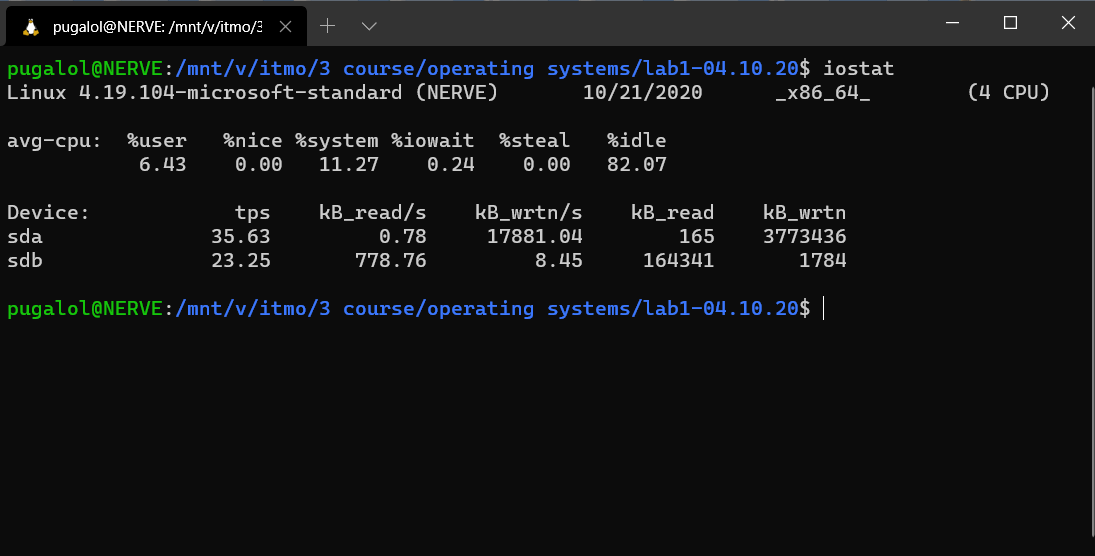
}

# Ход выполнения

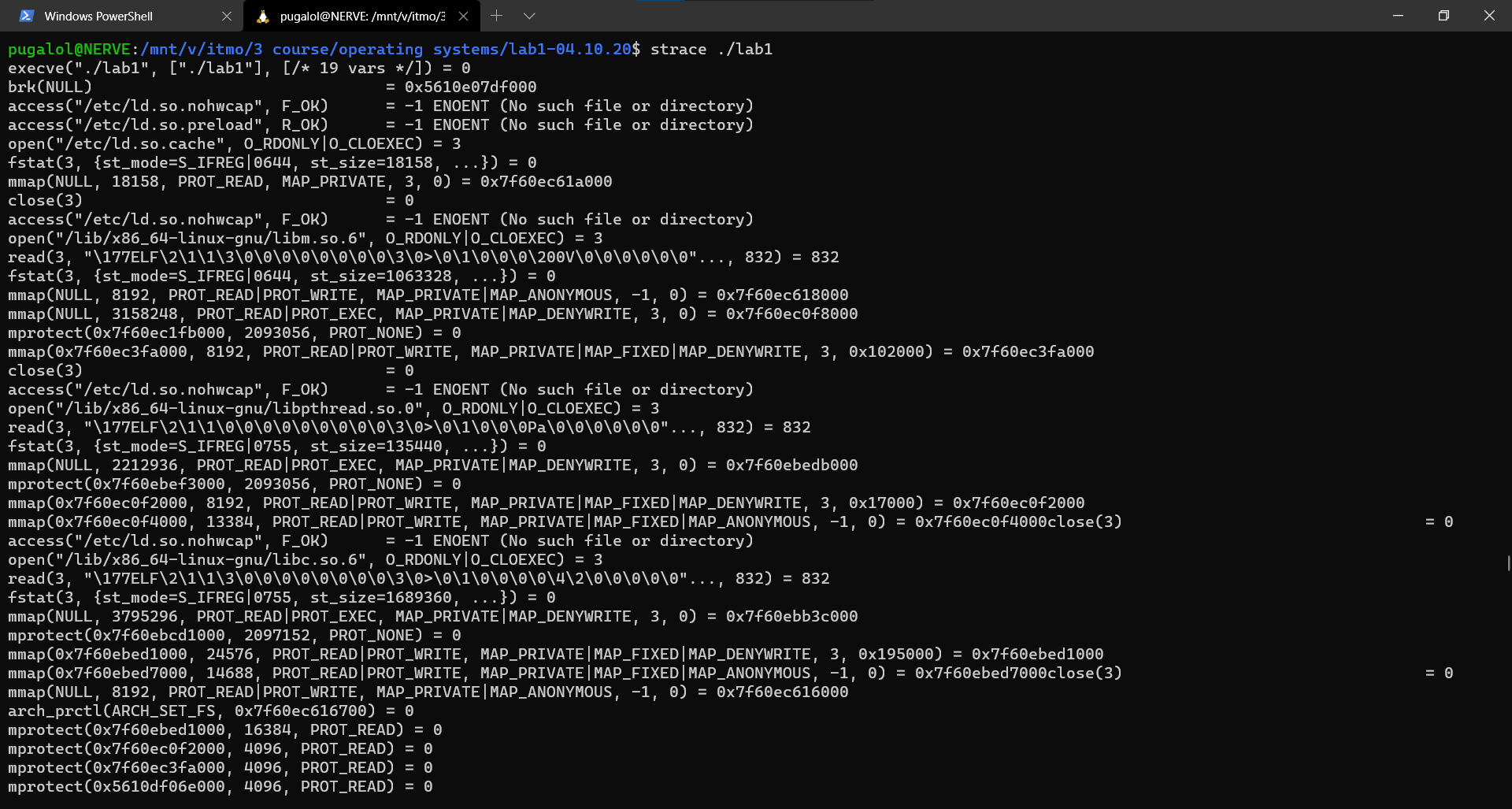
Так как при выполнение лабораторной использовалась WSL2, я имел возможность проводить замеры утилитами средствами Windows и Linux одновременно, и сравнить результаты (т.к. системы и их процессы живут в разных адресных пространствах, но «крутятся» на одном гипервизоре, имеет смысл сравнивать только данные об общей загрузке компьютера). Для замеров памяти воспользуемся отладчиком gdb, VS Code и утилитой top. Установим точки останова в необходимых местах, и посмотрим потребление памяти процессом lab1. Для параметров io: iostat. В случае Windows – Диспетчер задач.



|  |  |  |  |
| --- | --- | --- | --- |
| Заполнение памяти | | | |
|  | VIRT (вирт. п.) | RES (физ. п.) | |
| До аллокации | 9300 | 984 | |
| После аллокации | 142420 | 984 | |
| После заполнения участка данными | 216284 | 134240 | |
| После деаллокации | 216284 | 134240 | |
| Общее потребление | | | |
|  | Debian | | Windows 10 |
| %CPU | 50 | | 100 (если смотреть на потребление суммы всех процессов, то выходит ≈ 50%, а остальные, видимо, как раз отъедает “невидимый” для диспетчера процесс lab1 из wsl). |
| io | 778 КБ/с чтение;  8.5 КБ/с запись | | 0 КБ/с чтение;  12 КБ/с запись (не знаю, как объяснить подобную разницу… Может быть, гипервизор отдаёт каждой из ОС данные исключительно по их обращениям к диску). |



Для отслеживания системных вызовов воспользуюсь strace. Запустим её для программы командой strace ./lab1 и получим приблизительный вывод:



# Вывод

В процессе выполнения лабораторной работы я получил базовые навыки программирования на С, научился вызывать системные функции Linux в своих программах, а также наблюдать за нагрузкой на компьютер средствами операционной системы. Эти навыки будут полезны в процессе оптимизации и тестирования написанного мной кода.