

Университет ИТМО

“Операционные системы”

Лабораторная работа №1

Вариант:

A=97;B=0x1C57CB42;C=malloc;D=19;E=63;F=nocache;G=48;H=random;I=133;J=avg;K=flock

Работу выполнил:

Тарасов А.С.

Группа:

P33112

Преподаватель:

Осипов С.

г. Санкт – Петербург

2020 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 N=(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 star построить графики системных характеристик.

Код программы:

```

#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>
#include <stdint.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <inttypes.h>

//C=malloc;D=19;E=63;F=nocache;G=48;H=random;I=133;J=avg;K=flock

#define A 97
#define MEMSIZE_B A*1024*1024
#define B 0x1C57CB42
#define D 19
#define E 63
#define FILESIZE_B E*1024*1024
#define I 133
#define FILES_NUMBER (A/E)+1;

void * start_address;

typedef struct fillMemoryArgs {
    void *memory_pointer;
    size_t memory_size;
    FILE *urandom;
} fillArgs;

void * fill_memory_from_thread(void * args){
    fillArgs *fill_args = (fillArgs *) args;
    int c = fread((void *) fill_args->memory_pointer, sizeof(int),
(fill_args->memory_size)/sizeof(int), fill_args->urandom);
    if(c <= 0){
        perror("Память не заполняется");
        _exit(1);
    }

    // int * num = fill_args->memory_pointer;
    // printf("%p:  %d\n", fill_args->memory_pointer,  *num );

    return NULL;
}

void fill_memory(size_t memory_size){
    FILE *urandom = fopen("/dev/urandom", "r");
    pthread_t threads[D];
    size_t block_size = memory_size/D;

```

```

    fillArgs args = {start_address, block_size, urandom};

    for(int i = 0; i < D-1; i++){
        pthread_create(&threads[i], NULL, fill_memory_from_thread, (void *)
&args);
        args.memory_pointer += (memory_size/D);
    }

    for (int i = 0; i < D-1; i++){
        pthread_join(threads[i], NULL);
    }
    // printf("Область памяти заполнена\n");
}

void write_from_memory_to_file(int file, void* memory_pointer, uint64_t
size){
    write(file, memory_pointer, size);
    close(file);
}

const char* make_filename(uint8_t name){
    switch (name) {
        case 0:
            return "./first.txt";
        case 1:
            return "./second.txt";
        case 2:
            return "./third.txt";
        default:
            return "./over.txt";
    }
}

void fill_files(void* memory_pointer){
    // printf("Начало работы с файлами...\n");
    uint64_t files_count = (A / E) +1;
    size_t size_to_write = FILESIZE_B;
    size_t size_left = MEMSIZE_B;

    for (uint8_t i = 0; i < files_count; i++){
        const char* filename = make_filename(i);
        int out = open(filename, O_CREAT | O_WRONLY | O_TRUNC | O_DIRECT,
0666);

        // printf("Начало записи...\n");
        if (size_left < FILESIZE_B) {
            size_to_write = size_left;
            // printf("size_to_write was changed, теперь = %ld\n",
size_left);
        }
        write_from_memory_to_file(out, memory_pointer, size_to_write);
        size_left -= FILESIZE_B;
        memory_pointer += FILESIZE_B;
    }
    // printf("Запись в файлы окончена\n");
}

```

```

}

void file_lock(int fd) {
    int flockRc;

    if (flock(fd, LOCK_SH) != 0) {
        printf("Ошибка при блокировании, errno = %d\n", errno);
        exit(1);
    }
}

void file_unlock(int fd) {
    int flockRc = flock(fd, LOCK_UN);
    if (flockRc != 0) {
        printf("Ошибка при разблокировке, errno = %d\n", errno);
        exit(1);
    }
}

void analyze_file(const char* fileName) {
    int fd = open(fileName, O_CREAT | O_RDONLY, S_IRWXU | S_IRGRP | S_IROTH);
    file_lock(fd);
    off_t size = lseek(fd, 0, SEEK_END);
    lseek(fd, 0, SEEK_SET);
    int* data = (int*) malloc(size);

    read(fd, data, size);

    file_unlock(fd);
    close(fd);
    __uint64_t sum = 0;
    __uint64_t amount = size/sizeof(int);
    long double avg = 0;
    for (size_t i = 1; i < amount; i++) {
        sum += data[i];
    }
    avg = sum/amount;

    // printf("Расчётная сумма: '%s' = %ld\n", fileName, sum);
    printf("Расчётное среднее: '%s' = %Lf\n", fileName, avg);
    free(data);
}

void* read_from_file_thread(void* vargPtr) {
    // while(1) {
        for (int i = 0; i < (A/E)+1; i++) {
            const char* fileName = make_filename(i);
            analyze_file(fileName);
        }
    // }
    return NULL;
}

void read_from_files(){
    printf("Читаем файлы в %d потоков...\n", I);
    pthread_t threads[I];
    for (int i = 0; i < I; i++) {

```

```

        pthread_create(&threads[i], NULL, read_from_file_thread, NULL);
    }
    for (int i = 0; i < I; i++){
        pthread_join(threads[i], NULL);
    }
}

int main(void){
    printf(" всего элементов: %ld\n",  MEMSIZE_B/sizeof(int));

    printf("снять до аллокации\n");
    getchar();
    printf("снять до аллокации - готово\n");

    start_address = malloc(MEMSIZE_B);

    printf("снять после аллокации\n");
    getchar();
    printf("снять после аллокации - готово\n");

    printf("Начальный адрес: %p\n", start_address);
    while(1){
        fill_memory(MEMSIZE_B);
        printf("снять после заполнения памяти\n");
        getchar();
        printf("снять после заполнения памяти - готово\n");

        free(start_address);

        printf("снять после деаллокации памяти\n");
        getchar();
        printf("снять после деаллокации - готово\n");

        fill_files(start_address);
        read_from_files();

    }

    // while(1);
    return 0;
}

```

Мониторинг:

1. Замеры физической и виртуальной памяти:

```
olex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$ ps -eo pid,vsz,rss,comm | grep main
10960 2640 628 main
olex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$ ps -eo pid,vsz,rss,comm | grep main
10960 101972 628 main
olex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$ ps -eo pid,vsz,rss,comm | grep main
10960 200292 80180 main
olex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$ ps -eo pid,vsz,rss,comm | grep main
10960 100960 1744 main
olex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$
```

Карта памяти:

```
oLex@oLex-VivoBook-ASUSLaptop-X571GT-X571GT:~$ pmap -x 13466
13466: ./main
Адрес      Кб      RSS      Dirty Mode Mapping
000055cf0cd9f000      4        4        0 r---- main
000055cf0cda0000      4        4        0 r-x-- main
000055cf0cda1000      4        4        0 r---- main
000055cf0cda2000      4        4        4 r---- main
000055cf0cda3000      4        4        4 rw--- main
000055cf0cf15000     132       44       44 rw--- [ anon ]
00007fee5e675000    129032    129032    129032 rw--- [ anon ]
00007fee66477000      4         0         0 ----- [ anon ]
00007fee66478000     72708     64524     64524 rw--- [ anon ]
00007fee6ab79000      4         0         0 ----- [ anon ]
00007fee6ab7a000     72708     64524     64524 rw--- [ anon ]
00007fee6f27b000      4         0         0 ----- [ anon ]
00007fee6f27c000     72708     64524     64524 rw--- [ anon ]
00007fee7397d000      4         0         0 ----- [ anon ]
00007fee7397e000     72708     64524     64524 rw--- [ anon ]
00007fee7807f000      4         0         0 ----- [ anon ]
00007fee78080000     72708     64524     64524 rw--- [ anon ]
00007fee7c781000      4         0         0 ----- [ anon ]
00007fee7c782000     72708     64524     64524 rw--- [ anon ]
00007fee80e83000      4         0         0 ----- [ anon ]
00007fee80e84000     72708     64524     64524 rw--- [ anon ]
00007fee85585000      4         0         0 ----- [ anon ]
00007fee85586000     72708     64524     64524 rw--- [ anon ]
00007fee89c87000      4         0         0 ----- [ anon ]
00007fee89c88000     72708     64524     64524 rw--- [ anon ]
00007fee8e389000      4         0         0 ----- [ anon ]
00007fee8e38a000     72708     64524     64524 rw--- [ anon ]
00007fee92a8b000      4         0         0 ----- [ anon ]
00007fee92a8c000     72708     64524     64524 rw--- [ anon ]
00007fee9718d000      4         0         0 ----- [ anon ]
00007fee9718e000     72708     64524     64524 rw--- [ anon ]
00007fee9b88f000      4         0         0 ----- [ anon ]
00007fee9b890000     72708     64524     64524 rw--- [ anon ]
00007fee9ff91000      4         0         0 ----- [ anon ]
00007fee9ff92000     72708     64524     64524 rw--- [ anon ]
00007feea4693000      4         0         0 ----- [ anon ]
00007feea4694000     72708     64524     64524 rw--- [ anon ]
00007feea8d95000      4         0         0 ----- [ anon ]
00007feea8d96000     72708     64524     64524 rw--- [ anon ]
00007feead497000      4         0         0 ----- [ anon ]
00007feead498000     72708     64524     64524 rw--- [ anon ]
00007feeb1b99000      4         0         0 ----- [ anon ]
00007feeb1b9a000     72708     64524     64524 rw--- [ anon ]
00007feeb629b000      4         0         0 ----- [ anon ]
00007feeb629c000     72708     64524     64524 rw--- [ anon ]
00007feeba99d000      4         0         0 ----- [ anon ]
00007feeba99e000     72708     64524     64524 rw--- [ anon ]
00007feebf09f000      4         0         0 ----- [ anon ]
00007feebf0a0000     72708     64524     64524 rw--- [ anon ]
00007feec37a1000      4         0         0 ----- [ anon ]
00007feec37a2000     72708     64524     64524 rw--- [ anon ]
00007feec7ea3000      4         0         0 ----- [ anon ]
00007feec7ea4000     72708     64524     64524 rw--- [ anon ]
```


Затраченное процессорное время на выполнение программы:

```
time ./main
```

```
real    0m6,817s
user    0m11,881s
sys     0m11,377s
```

Затраченное процессорное время на операции ввода-вывода используя системные утилиты:

```
sudo strace -c -fp [ps -C main]
```

time	seconds	usecs/call	calls	errors	syscall
91,00	1150,286973	2561886	449	4	read
5,28	66,787819	70600	946	49	futex
1,44	18,176959	24899	730		munmap
0,69	8,739386	11560	756		mmap
0,31	3,912666	8694	450		write
0,31	3,874940	14090	275		madvise
0,29	3,670177	3972	924		flock
0,18	2,328136	2476	940		lseek
0,17	2,141219	4675	458		close
0,15	1,837360	3860	476		openat
0,08	1,057988	3503	302		clone
0,06	0,712238	2490	286		mprotect
0,04	0,552005	1827	302		set_robust_list
0,00	0,000101	50	2	2	ioctl
0,00	0,000073	36	2		fstat
00.00	1264,078040		7298	55	total

```
lex@olex-VivoBook-ASUSLaptop-X571GT-X571GT:~$
```

Вывод

Выполняя данную работу я познакомился с основами программирования на С под Linux. Также я узнал об устройстве потоков и процессов, научился выполнять код в разных потоках/процессах. Узнал о системных вызовах и о средствах мониторинга.

При выполнении столкнулся с проблемой записи данных в память во множестве потоков - каждый поток записывал свои данные поверх данных, записанных другим потоком. КПД программы стремительно снижалось. Эта проблема решилась путем распределения памяти между потоками: для каждого потока считался свой начальный адрес и записывался лишь некоторый блок общей области.