Университет ИТМО
Факультет ПИиКТ

Операционные системы

Лабораторная работа №1

Выполнила: Наумова Н.А.
Группа P33022
Преподаватель: Осипов С.В.

Санкт-Петербург
2020 г.

Задание:

Разработать программу на языке C, которая осуществляет следующие действия

- Создает область памяти размером A=**84** мегабайт, начинающихся с адреса B=**0x9C516B83** (если возможно) при помощи C=(**malloc**, mmap) заполненную случайными числами /dev/urandom в D=**127** 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=**185** мегабайт с использованием F=(**блочного**, некешируемого) обращения к диску. Размер блока ввода-вывода G=**33** 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=**143** 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:

```
#include <stdlib.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <stdint.h>
#include <sys/file.h>

#define MALLOC_SIZE 84000000 // 84 MB
#define FILL_THREADS 127 // 127
#define ANALYZE_THREADS 143 // 143
#define BLOCK_SIZE 33 // 33
#define FILE_SIZE 185000000
#define FILE_COUNT (MALLOC_SIZE / FILE_SIZE + (MALLOC_SIZE % FILE_SIZE == 0 ?
0 : 1))


struct ThreadsArgs {
    int fd;
    void* address;
    size_t size;
};

// для рандомной записи блоков в файл (отслеживать свободные промежуточки
блоков)
struct LinkedListNode {
    size_t startBlock;
    size_t size;
    struct LinkedListNode* next;
};

void* fillMemory();
void fillMemoryRegion(int, void*, size_t);
void* fillMemoryRegionProxy(void*);
void writeRegionToFile(void*);
void writeFile(int, void*, size_t);
int countNodes(struct LinkedListNode *);
size_t pickRandomBlock(struct LinkedListNode *);
struct LinkedListNode * selectNodeBy(struct LinkedListNode *, int);
void removeNode(struct LinkedListNode *, int);
void readFiles();
void analyzeFile(char*);
void* fileAnalyzeProxy(void*);


int main() {
    while (1) {
        void* regionPointer = fillMemory();
        writeRegionToFile(regionPointer);
        free(regionPointer);
        readFiles();
    }
}

void* fillMemory() {
```

```
    void * regionPointer = malloc(MALLOC_SIZE);

    int fd = open("/dev/urandom", O_RDONLY);
    if (fd < 0) {
        printf("Cannot open /dev/urandom\n");
        return regionPointer;
    }
    pthread_t threads[FILL_THREADS];
    struct ThreadsArgs threadArgs[FILL_THREADS];

    size_t size = MALLOC_SIZE/FILL_THREADS;
    for (int i = 0; i < FILL_THREADS; i++) {
        threadArgs[i].fd = fd;
        if (i == FILL_THREADS - 1)
            threadArgs[i].size = size + MALLOC_SIZE % FILL_THREADS;
        else
            threadArgs[i].size = size;
        threadArgs[i].address = (void*)((uint8_t *)regionPointer + i *
size);
        pthread_create(&threads[i], 0, fillMemoryRegionProxy,
&threadArgs[i]);
    }

    for (int i = 0; i < FILL_THREADS; i++)
        pthread_join(threads[i], NULL);

    close(fd);
    return regionPointer;
}

void writeRegionToFile(void* regionPtr) {
    for (int i = 0; i < FILE_COUNT; i++) {
        char fileName[i + 2];
        for (int c = 0; c < i + 1; c++)
            fileName[c] = 'a';
        fileName[i + 1] = '\0';
        int fd = open(fileName, O_CREAT | O_WRONLY, S_IRWXU | S_IRWXG |
S_IRWXO);
        if (fd < 0) {
            printf("Cannot create file\n");
            return;
        }
        flock(fd, LOCK_EX);
        if (i == FILE_COUNT - 1)
            writeFile(fd, (void*)((uint8_t *)regionPtr + i * FILE_SIZE),
MALLOC_SIZE - FILE_SIZE * (FILE_COUNT - 1));
        else writeFile(fd, (void*)((uint8_t *)regionPtr + i * FILE_SIZE),
FILE_SIZE);
        flock(fd, LOCK_UN);
    }
    printf("The memory area is full\n");
}

void _traceLinkedList(struct LinkedListNode* node) {
    printf("Nodes: ");
    struct LinkedListNode* current = node;
    while (current != NULL) {
        printf("[start=%zu, size=%zu] -> ", current->startBlock,
current->size);
        current = current->next;
    }
```

```
    printf("\n");
}

void writeFile(int fd, void* address, size_t size) {
    int count = size / BLOCK_SIZE + (size % BLOCK_SIZE == 0 ? 0 : 1);
    struct LinkedListNode node;
    node.size = count;
    node.startBlock = 0;
    node.next = NULL;
    int blocksWritten = 0;

    while(1) {
        size_t i = pickRandomBlock(&node);
        if (i == -1)
            break;
        lseek(fd, i * BLOCK_SIZE, SEEK_SET);
        if (i == count - 1)
            write(fd, (void*)((uint8_t *)address + i * BLOCK_SIZE), size -
BLOCK_SIZE * (count - 1));
        else write(fd, (void*)((uint8_t *)address + i * BLOCK_SIZE),
BLOCK_SIZE);
        blocksWritten++;
        printf("Writing %d blocks of %d\r", blocksWritten, count);
        fflush(stdout);
    }
    printf("\n");
}

size_t pickRandomBlock(struct LinkedListNode * node) {
    size_t partitionCount = countNodes(node);
    size_t selectedNodeIdx = rand() % partitionCount;
    struct LinkedListNode * selectedNode = selectNodeBy(node,
selectedNodeIdx);
    if (partitionCount == 1 && selectedNode->size == 0)
        return -1;
    size_t randomBlockNumber = rand() % selectedNode->size;
    size_t result = selectedNode->startBlock + randomBlockNumber;
    if (randomBlockNumber == selectedNode->size - 1) {
        selectedNode->size --;
        if (selectedNode->size == 0)
            removeNode(node, selectedNodeIdx);
    } else if (randomBlockNumber == 0) {
        selectedNode->startBlock ++;
        selectedNode->size --;
    } else {
        size_t size = selectedNode->size - 1;
        selectedNode->size = randomBlockNumber;
        struct LinkedListNode * newNode = (struct LinkedListNode
*)malloc(sizeof(struct LinkedListNode));
        newNode->startBlock = selectedNode->startBlock + randomBlockNumber +
1;
        newNode->size = size - selectedNode->size;
        struct LinkedListNode * prevNextNode = selectedNode->next;
        selectedNode->next = newNode;
        newNode->next = prevNextNode;
    }
    return result;
}

void removeNode(struct LinkedListNode * node, int index) {
    if (index == 0) {
```

```
        if (node->next == NULL) {
            node->next = 0;
            node->size = 0;
            node->startBlock = 0;
            return;
        }

        node->startBlock = node->next->startBlock;
        node->size = node->next->size;
        struct LinkedListNode * toBeDeleted = node->next;
        node->next = node->next->next;
        free((void *)toBeDeleted);
    } else {
        struct LinkedListNode * prevDeletedElement = selectNodeBy(node,
index - 1);
        struct LinkedListNode * toBeDeleted = prevDeletedElement->next;
        prevDeletedElement->next = prevDeletedElement->next->next;
        free((void *)toBeDeleted);
    }
}

struct LinkedListNode * selectNodeBy(struct LinkedListNode * node, int
position) {
    for (int i = 0; i < position; i ++)
        node = node->next;
    return node;
}

int countNodes(struct LinkedListNode * node) {
    int count = 1;
    if (node == NULL)
        return 0;
    while (node->next != NULL) {
        count ++;
        node = node->next;
    }
    return count;
}

void readFiles() {
    pthread_t threads[ANALYZE_THREADS];
    for (int i = 0; i < ANALYZE_THREADS; i++) {
            pthread_create(&threads[i], NULL, fileAnalyzeProxy, NULL);
    }
}

void* fileAnalyzeProxy (void* argsPointer) {
    for (int i = 0; i < FILE_COUNT; i++) {
        char fileName[i + 2];
        for (int c = 0; c < i + 1; c++)
            fileName[c] = 'a';
        fileName[i + 1] = '\0';
        analyzeFile(fileName);
    }
    return 0;
}

void analyzeFile(char* filename) {
    int fd = open(filename, O_RDONLY);
    if (fd < 0) {
        printf("Cannot open file\n");
```

```
        return;
    }
    flock(fd, LOCK_SH);
    off_t fileSize = lseek(fd, 0, SEEK_END);
    //back to the beginning of the file
    lseek(fd, 0, SEEK_SET);
    int64_t* fileData = (int64_t*) malloc(fileSize);
    read(fd, fileData, fileSize);
    flock(fd, LOCK_UN);
    uint64_t min = fileData[0];
    for (size_t i = 0; i < fileSize/sizeof(uint64_t); i++)
        if (fileData[i] < min) min = fileData[i];
    printf("Analysis completed. Min = %lld\n", min);
    free(fileData);
}

void* fillMemoryRegionProxy(void* argsPointer) {
    struct ThreadsArgs* args = argsPointer;
    fillMemoryRegion(args->fd, args->address, args->size);
    return NULL;
}

void fillMemoryRegion(int fd, void* address, size_t size) {
    read(fd, address, size);
}
```

Результаты измерений:

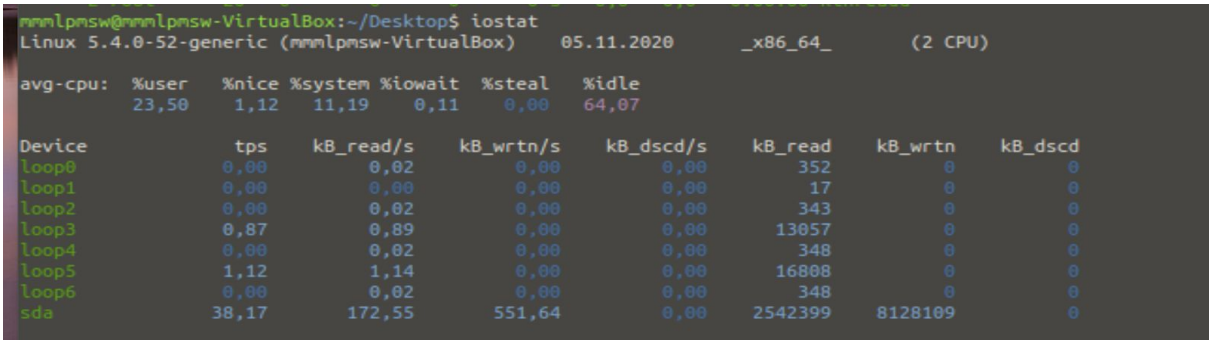
Выполняем остановку программы в нужных местах с помощью GDB, вызываем top и смотрим результаты.

	VIRT	RES
До аллокации	2512	588
После аллокации	84676	588
После заполнения участка данными	117856	84016
После деаллокации	35824	2216

%CPU min = 67

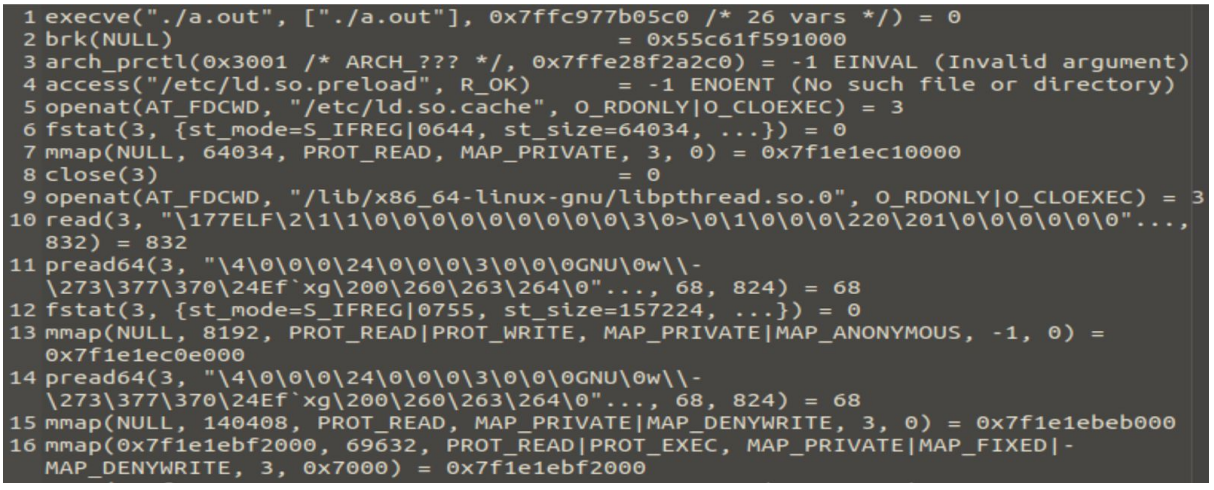
%CPU max = 107

Чтение/запись (iostat)



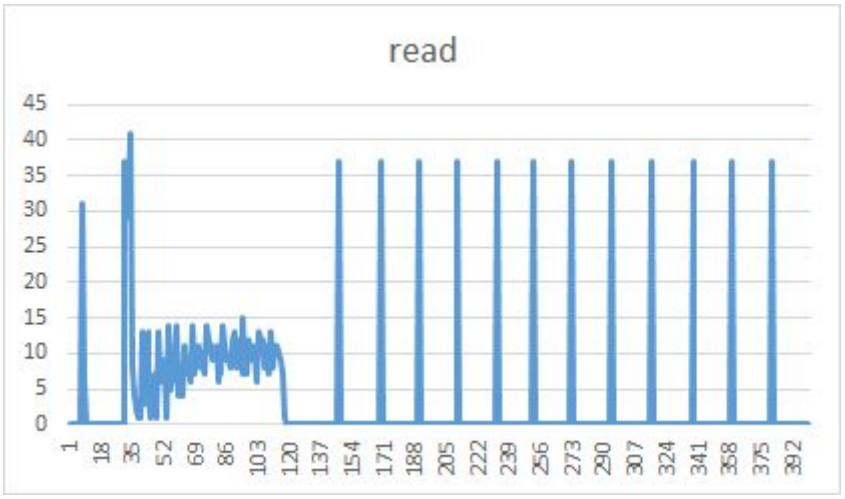
```
mmmlpmsw@mmmlpmsw-VirtualBox:~/Desktop$ iostat
Linux 5.4.0-52-generic (mmmlpmsw-VirtualBox)    05.11.2020      _x86_64_       (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          23,50    1,12   11,19    0,11    0,00   64,07

Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
loop0             0,00         0,02         0,00         0,00        352          0          0
loop1             0,00         0,00         0,00         0,00         17          0          0
loop2             0,00         0,02         0,00         0,00        343          0          0
loop3             0,87         0,89         0,00         0,00      13057          0          0
loop4             0,00         0,02         0,00         0,00        348          0          0
loop5             1,12         1,14         0,00         0,00      16808          0          0
loop6             0,00         0,02         0,00         0,00        348          0          0
sda              38,17       172,55       551,64         0,00    2542399    8128109          0
```

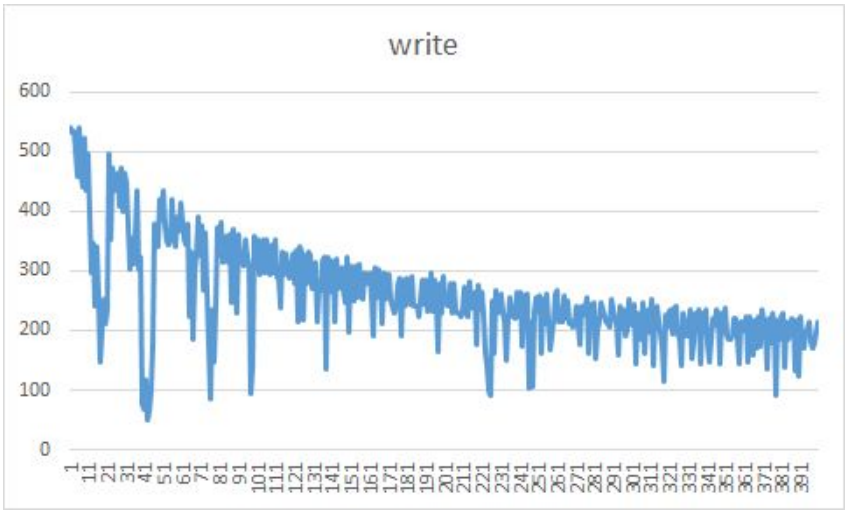
strace ./a.out (для отслеживания системных вызовов)



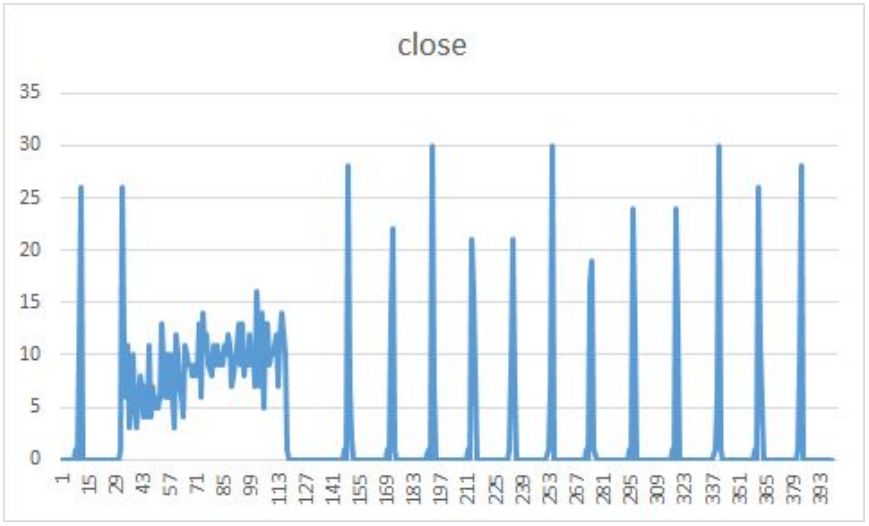
```
 1 execve("./a.out", ["./a.out"], 0x7ffc977b05c0 /* 26 vars */) = 0
 2 brk(NULL)                               = 0x55c61f591000
 3 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe28f2a2c0) = -1 EINVAL (Invalid argument)
 4 access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
 5 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
 6 fstat(3, {st_mode=S_IFREG|0644, st_size=64034, ...}) = 0
 7 mmap(NULL, 64034, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1e1ec10000
 8 close(3)                                = 0
 9 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
10 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
11 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0w\\-
   \273\377\370\24Ef`xg\200\260\263\264\0"..., 68, 824) = 68
12 fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0
13 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
   0x7f1e1ec0e000
14 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0w\\-
   \273\377\370\24Ef`xg\200\260\263\264\0"..., 68, 824) = 68
15 mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1e1ebeb000
16 mmap(0x7f1e1ebf2000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|-
   MAP_DENYWRITE, 3, 0x7000) = 0x7f1e1ebf2000
```

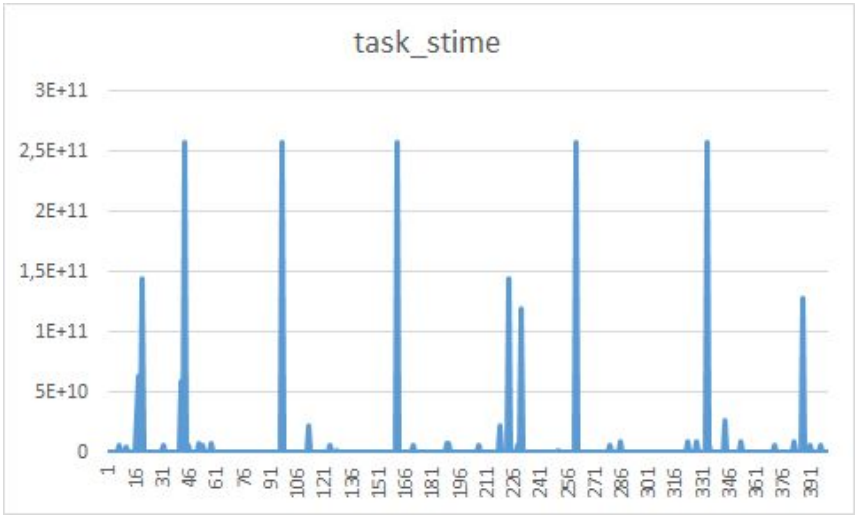
stap


read
45
40
35
30
25
20
15
10
5
0
1
18
35
52
69
86
103
120
137
154
171
188
205
222
239
256
273
290
307
324
341
358
375
392


write
600
500
400
300
200
100
0
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181
191
201
211
221
231
241
251
261
271
281
291
301
311
321
331
341
351
361
371
381
391


open
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393


close
35
30
25
20
15
10
5
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393


task_stime
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
1
16
31
46
61
76
91
106
121
136
151
166
181
196
211
226
241
256
271
286
301
316
331
346
361
376
391

Вывод:

сделав эту лабораторную работу, я, хочется в это верить, научилась писать на C хоть что-то, что перестало кидать мне segmentation fault'ы, взаимодействовать с операционной системой при помощи системных вызовов, с gdb и измерять потребление памяти и использование процессорного времени.