НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Мегафакультет Компьютерных технологий и управления

Лабораторная работа 1

По дисциплине:

«Операционные системы»

Группа: P33112

Студент: Колесников М.В.

Преподаватель: Храмов С.В.

г. Санкт-Петербург

2020

Цель работы

Разработать программу на языке C, которая осуществляет следующие действия

- Создает область памяти размером 142 мегабайт, начинающихся с адреса 0x7FBC24 (если возможно) при помощи malloc заполненную случайными числами /dev/urandom в 44 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 57 мегабайт с использованием некешируемого обращения к диску. Размер блока ввода-вывода 102 байт. Преподаватель выдает в качестве задания последовательность записи/чтения блоков случайный
- Генерацию данных и запись осуществлять в бесконечном цикле.
- В отдельных 147 потоках осуществлять чтение данных из файлов и подсчитывать агрегированные характеристики данных – сумму
- Чтение и запись данных в/из файла должна быть защищена примитивами синхронизации cv.
- 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.

Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>

#define A 142
#define B 0x7FBC24
#define C malloc
#define D 44
#define E 57
#define F nocache
#define G 102
#define H random
#define I 147
#define J max
#define K cv

#define FILE_FLAG O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC
```

```
typedef struct{
    size_t data_size;
    unsigned char* start_address;
    FILE * urandom;
} BufferArgs;

typedef struct{
    pthread_mutex_t * mutex;
    pthread_cond_t * cv;
    unsigned int count_files;
}ThreadArgs;

void* setBuffer(void* set_args);
void writeToFile(unsigned int count_files, unsigned char *buffer);
void* writeToMemory(unsigned char* buffer, unsigned int count_files, pthread_mutex_t * mutex, pthread_cond_t *
cv);
long searchMax(unsigned char *block);
_Noreturn void* readAndExecute(void * args);

int main() {
    printf("До аллокации");
    getchar();
    int count_files = A / E + (A % E > 0 ? 1:0);
    unsigned char* buffer;
    buffer = C(A*1024*1024);

    printf("После аллокации");
    getchar();

    pthread_mutex_t mutex;
    pthread_cond_t cv;

    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&cv,NULL);

    writeToMemory(buffer, count_files, &mutex, &cv);

    printf("После заполнения участка данными");
    getchar();

    free(buffer);
    printf("После деаллокации");
    getchar();

    buffer = C(A*1024*1024);
    pthread_t read_threads[I];

    for(int i =0;i<75;i++){
        ThreadArgs args = {&mutex, &cv, count_files};
        pthread_create(&read_threads[i], NULL, readAndExecute, &args);
    }
    while(1){
        writeToMemory(buffer, count_files, &mutex, &cv);
    }
    return 0;
}


void* setBuffer(void* set_args){
    BufferArgs *args = (BufferArgs*) set_args;

    for(size_t i =0;i<args->data_size;){
        i+=fread(args->start_address + i,1,args->data_size-i,args->urandom);
    }

    return NULL;
}
```

```
void writeToFile(unsigned int count_files, unsigned char *buffer){
    for(int i =0;i<count_files;i++){
        char file_name[14];
        snprintf(file_name,14,"file_%d.bin",i);
        int fd = open(file_name,FILE_FLAG,0666);
        size_t file_size = E * 1024 * 1024;

        for (size_t j = 0;j<file_size;){
            unsigned int block = rand() % (A*1024*1024/G);
            unsigned int block_size = file_size - j < G ? file_size - j : G;
            j+=write(fd, buffer+block*G, block_size);
        }
    }

}

void* writeToMemory(unsigned char* buffer, unsigned int count_files, pthread_mutex_t * mutex, pthread_cond_t *
cv){
    pthread_t threads[D];
    unsigned char* start = buffer;
    FILE * urandom = fopen("/dev/urandom", "rb");
    size_t size_of_data = A*1024*1024/D;

    for(int i = 0;i<D-1;i++){
        BufferArgs args = {size_of_data, start, urandom};
        pthread_create(&threads[i], NULL, setBuffer, &args);
        start+=size_of_data;
    }

    BufferArgs args ={size_of_data + A * 1024 * 1024 % D, start, urandom};
    pthread_create(&threads[D - 1], NULL, setBuffer, &args);

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

    fclose(urandom);
    pthread_mutex_lock(mutex);
    writeToFile(count_files, buffer);
    pthread_cond_broadcast(cv);
    pthread_mutex_unlock(mutex);

}

long searchMax(unsigned char *block){
    int max = INT16_MIN;

    for (int i =0;i<G/sizeof(int);i+=sizeof(int)){
        int num = 0;

        for(int j = 0; j<sizeof(int);j++){
            num = (num << 8) + block[i+j];
        }

        if (num > max)
            max = num;
    }

    return max;
}

_Noreturn void* readAndExecute(void * args){
    ThreadArgs * arg = (ThreadArgs*) args;
    char file_name[14];
    snprintf(file_name,14,"file_%d.bin",arg->count_files-1);

    while(1){
```

```
        int max = INT16_MIN;
        pthread_mutex_lock(arg->mutex);
        pthread_cond_wait(arg->cv,arg->mutex);
        printf("%ld в состоянии ожидания\n", pthread_self());
        FILE *file = fopen(file_name,"rb");
        unsigned char block_size[G];

        for(unsigned int i = 0;i<2*E*1024*1024/G;i++){
            unsigned int block = rand()%(2*E*1024*1024/G);
            fseek(file,block*G,0);
            if(fread(&block_size,1,G,file) !=G)
                continue;
            else
                max = searchMax(block_size);

        }

        printf("Максимум %lld\n",max);
        fclose(file);
        pthread_mutex_unlock(arg->mutex);
    }
}
```

Переосмысление кода

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>

#define A 142
#define B 0x7FBC24
#define C malloc
#define D 44
#define E 57
#define F nocache
#define G 102
#define H random
#define I 147
#define J max
#define K cv

#define FILE_FLAG O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC

typedef struct{
    size_t data_size;
    unsigned char* start_address;
    FILE * urandom;
} BufferArgs;

typedef struct{
    pthread_mutex_t * mutex;
    pthread_cond_t * cv;
    unsigned int count_files;
}ThreadArgs;

void* setBuffer(void* set_args);
void writeToFile(unsigned int count_files, unsigned char *buffer);
void* writeToMemory(unsigned char* buffer, unsigned int count_files, pthread_mutex_t * mutex, pthread_cond_t *
cv);
long searchMax(unsigned char *block);
_Noreturn void* readAndExecute(void * args);

#define IM printf("До аллокации");getchar();int count_files = A / E + (A % E > 0 ? 1:0);unsigned char* buffer;buffer =
C(A*1024*1024);printf("После аллокации");getchar();pthread_mutex_t mutex;pthread_cond_t cv;
```

```
#define WANNA  pthread_mutex_init(&mutex,NULL);pthread_cond_init(&cv,NULL);writeToMemory(buffer,
count_files, &mutex, &cv);printf("После заполнения участка данными");getchar();free(buffer);printf("После
деаллокации");getchar();buffer = C(A*1024*1024);pthread_t read_threads[I];
#define DIE for(int i =0;i<75;i++){ThreadArgs args = {&mutex, &cv, count_files};pthread_create(&read_threads[i],
NULL, readAndExecute, &args);}while(1){writeToMemory(buffer, count_files, &mutex, &cv);}return 0;

int main() {
    IM
    WANNA
    DIE
}


void* setBuffer(void* set_args){
    BufferArgs *args = (BufferArgs*) set_args;

    for(size_t i =0;i<args->data_size;){
        i+=fread(args->start_address + i,1,args->data_size-i,args->urandom);
    }

    return NULL;
}

void writeToFile(unsigned int count_files, unsigned char *buffer){
    for(int i =0;i<count_files;i++){
        char file_name[14];
        snprintf(file_name,14,"file_%d.bin",i);
        int fd = open(file_name,FILE_FLAG,0666);
        size_t file_size = E * 1024 * 1024;

        for (size_t j = 0;j<file_size;){
            unsigned int block = rand() % (A*1024*1024/G);
            unsigned int block_size = file_size - j < G ? file_size - j : G;
            j+=write(fd, buffer+block*G, block_size);
        }
    }

}

void* writeToMemory(unsigned char* buffer, unsigned int count_files, pthread_mutex_t * mutex, pthread_cond_t *
cv){
    pthread_t threads[D];
    unsigned char* start = buffer;
    FILE * urandom = fopen("/dev/urandom", "rb");
    size_t size_of_data = A*1024*1024/D;

    for(int i = 0;i<D-1;i++){
        BufferArgs args = {size_of_data, start, urandom};
        pthread_create(&threads[i], NULL, setBuffer, &args);
        start+=size_of_data;
    }

    BufferArgs args ={size_of_data + A * 1024 * 1024 % D, start, urandom};
    pthread_create(&threads[D - 1], NULL, setBuffer, &args);

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

    fclose(urandom);
    pthread_mutex_lock(mutex);
    writeToFile(count_files, buffer);
    pthread_cond_broadcast(cv);
    pthread_mutex_unlock(mutex);

}

long searchMax(unsigned char *block){
```

```
    int max = INT16_MIN;

    for (int i =0;i<G/sizeof(int);i+=sizeof(int)){
        int num = 0;

        for(int j = 0; j<sizeof(int);j++){
            num = (num << 8) + block[i+j];
        }

        if (num > max)
            max = num;
    }

    return max;
}

_Noreturn void* readAndExecute(void * args){
    ThreadArgs * arg = (ThreadArgs*) args;
    char file_name[14];
    snprintf(file_name,14,"file_%d.bin",arg->count_files-1);

    while(1){
        int max = INT16_MIN;
        pthread_mutex_lock(arg->mutex);
        pthread_cond_wait(arg->cv,arg->mutex);
        printf("%ld в состоянии ожидания\n", pthread_self());
        FILE *file = fopen(file_name,"rb");
        unsigned char block_size[G];

        for(unsigned int i = 0;i<2*E*1024*1024/G;i++){
            unsigned int block = rand()%(2*E*1024*1024/G);
            fseek(file,block*G,0);
            if(fread(&block_size,1,G,file) !=G)
                continue;
            else
                max = searchMax(block_size);

        }

        printf("Максимум %lld\n",max);
        fclose(file);
        pthread_mutex_unlock(arg->mutex);
    }
}
```

Системный мониторинг

ps -A | grep main — узнаём PID

Данная система – Ubuntu 20.04.1. Запущена в Oracle VM Virtual Box со следующими параметрами системы: выделенная оперативная память – 4096 мб, 2 ядра

Использующиеся средства мониторинга: стандартная утилита top для мониторинга количества занимаемой виртуальной и физической памяти в моменты: до аллокации, после аллокации,после заполнения участка данными и после деалокации; также данная утилита использовалась для замера %CPU во время работы программы.

Для мониторинга подсистемы I/O была использована команда iostat.

Для трассировки – strace.

Замеры памяти top

Период замера	VIRT(вирт.память)	RES(физ.память)
До аллокации	3102	806
После аллокации	194676	812
После заполнения участка данными	340012	167184
После деаллокации	141620	2811

```
1871 maxim     20   0    3102    806    512 t  56,0   0,1   0:00.00 main
1871 maxim     20   0  194676    812    512 t  56,0   0,1   0:51.12 main
1871 maxim     20   0  340012 167184    512 t  56,0   0,1  28:02.92 main
1871 maxim     20   0  141620   2811    512 t  56,0   0,1  28:52.44 main
```

Замеры %CPU & IO

```
1871 maxim     20   0  861072   5652   1552 S  56,0   0,1  10:48.88 main
```

```
sda              691,86       827,33      4676,72         0,00    1322408     747
5273             0
```

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           8,90    0,86    4,40   16,47    0,00   69,37
```

Измеренный размер %CPU: 56.0%
Измеренный I/O kb_read/s: 827.3
Измеренный I/O kb_write/s: 4676.72

Для всех выведенных iostat'ом параметров значения нормальные кроме %idle, так как у нас большое количество потоков, каждый из которых в определенный период времени захватывает mutex файла, и остальные потоки находятся в состоянии ожидания и ждут, пока он завершит свою работу и 'отпустит' mutex данного файла.

Часть трассировки

Sudo strace -p pid

write(3, "\323U.\231k\373@I\210\246*\"\303\331=W\340\306\2610'O:\257G\
345\256\356\225\306\f\244"..., 4096) = 4096
write(3, "w\22\371\264\336\260\266\304&\35\7\374E\203kG\207\232o\24+\3123#4t\
324\231]\322\360\317"..., 4096) = 4096
write(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"...,
4096) = 4096
write(3, "\17{<\313A\10\235\1\37+\364\275r%-80c\342\221+\3724Y?*F\
213\371\252\311\273"..., 4096) = 4096
write(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"...,
4096) = 4096
write(3, "\316\343\367\324B=\262\177\330[\254\344\340\367\275\213\34w\2=\211&\
354\222\240.\257\255\20\373f\241"..., 4096) = 4096
write(3, "\322\340\366\273cuQ3\357WI\212\222\257\324\340\36\4F\270c\

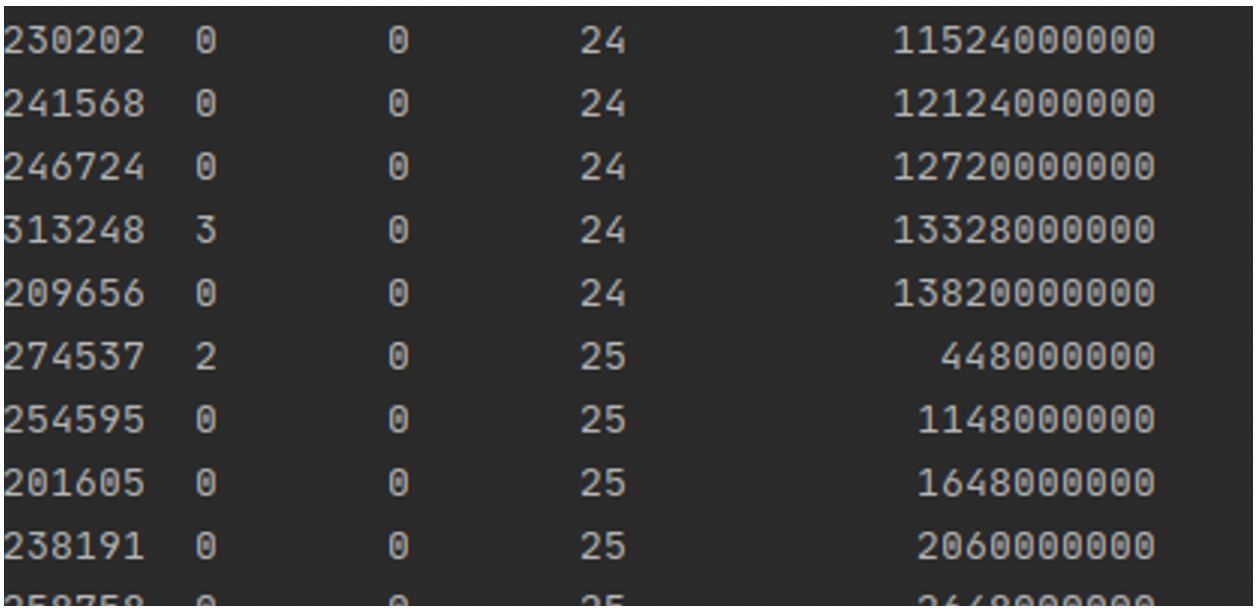
30\1\372\304\260\265y\371\363\322\260"..., 4096) = 4096

Stap (SystemTap)

sudo stap -x $(pidof main) system_monitor.stp

system_monitor.stp:

```
global read,close,write,open
probe begin {
        start = gettimeofday_s()
        open = 0
        close = 0
}

probe syscall.write {
        if (pid() == target()) {
                write += 1
        }
}

probe syscall.read {
        if (pid() == target()) {
                read += 1
        }
}

probe syscall.open {
        if (pid() == target()) {
                open += 1
        }
}

probe syscall.close {
        if (pid() == target()) {
                close += 1
        }
}

probe timer.ms(1000) {
        printf("%d\t%d\t%d\t%d\t%16d\n", read, write, open, close, task_stime())
        read = 0
        write = 0
}
```



Вывод

В ходе выполнения данной лабораторной работы я поставил 3 операционную систему, получил незабываемый опыт программирования на языке C, а также опробовал возможность многопоточного выполнения программы в ОС Linux. Также вспомнил некоторые средства мониторинга для UNIX-os и опробовал новые средства мониторинга (system tap).