Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Операционным системам»

Выполнил:

Студент группы P33111

Лесонен М.О.

Преподаватель:

Покид А.В.

Санкт-Петербург

2020

**Цель работы**:

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(systentap) построить графики системных характеристик.

A=18;B=0x37C28270;C=mmap;D=93;E=135;F=nocache;G=75;H=random;I=17;J=avg;K=sem

**Код**:

[#include](https://vk.com/im?sel=317826&st=%23include) <pthread.h>  
[#include](https://vk.com/im?sel=317826&st=%23include) <stdlib.h>  
[#include](https://vk.com/im?sel=317826&st=%23include) <stdio.h>  
[#include](https://vk.com/im?sel=317826&st=%23include) <memoryapi.h>  
[#include](https://vk.com/im?sel=317826&st=%23include) <semaphore.h>

[#define](https://vk.com/im?sel=317826&st=%23define) A 18 // область памяти  
[#define](https://vk.com/im?sel=317826&st=%23define) B 0x37C28270 // адрес начала выделенной памяти  
[#define](https://vk.com/im?sel=317826&st=%23define) D 93 // кол-во потоков  
[#define](https://vk.com/im?sel=317826&st=%23define) E 135 // размер файла  
[#define](https://vk.com/im?sel=317826&st=%23define) G 75 //размер блока ввода-вывода  
[#define](https://vk.com/im?sel=317826&st=%23define) I 17 //потоки чтения  
int threadsActivated = 1;  
  
  
  
//Рандом чисел  
int randRightVersion() {  
  
int retval;  
int [\*myInt](https://vk.com/club71954440) = (int \*) malloc(sizeof(int));  
\*(((char \*) myInt) + 0) = (rand());  
\*(((char \*) myInt) + 1) = rand();  
\*(((char \*) myInt) + 2) = rand();  
\*(((char \*) myInt) + 3) = rand() % 127;  
retval = [\*myInt](https://vk.com/club71954440);  
free(myInt);  
return retval;  
}  
  
//данные для потока генератора памяти  
typedef struct {  
char \*memoryRegion;  
int numCount;  
} generatorParams;  
  
//гененрирует значения в памяти  
void \*threadGenerator(void \*funcParams) {  
generatorParams \*params = (generatorParams \*) funcParams;  
int first\_run = 1;  
do {  
for (int i = 0; i < params->numCount \* 4; ++i) {  
params->memoryRegion[i] = rand() % 256;  
}  
if (first\_run) {  
first\_run = 0;  
}  
} while (threadsActivated);  
return NULL;  
}  
  
//параметры для потока записи в файл  
typedef struct {  
int filesNumber;  
sem\_t\* semaphore;  
int [\*start](https://vk.com/id52435);  
int futexCompare;  
} fileWriterParams;  
  
  
//поток записи  
void \*threadFileWriter(void \*funcParams) {  
fileWriterParams \*params = (fileWriterParams \*) funcParams;  
char \*thread\_start\_write = (char \*) params->start;  
do {  
for (int i = 0; i < params->filesNumber; i++) {  
sem\_wait(params->semaphore + i);  
char filename[8] = "malab\_\0";  
filename[5] = 'A' + i;  
int blocksAmount = A \* 1024 \* 1024 / G;  
FILE \*currentFile;  
fopen\_s(&currentFile, filename, "w +");  
for (int d = 0; d < E \* 1024 \* 1024 / (G); d++) {  
fwrite(thread\_start\_write + ((randRightVersion() % blocksAmount) \* G), G, 1, currentFile);  
}  
fclose(currentFile);  
  
sem\_post(params->semaphore+i);  
}  
} while (threadsActivated);  
return NULL;  
}  
  
//параметры для потока чтения файлов  
typedef struct {  
int id;  
int fileNumber;  
sem\_t\* semaphore;  
int futexCompare;  
} fileReaderParams;  
  
  
//поток чтения  
void \*threadFileReader(void \*funcParams) {  
fileReaderParams \*params = (fileReaderParams \*) funcParams;  
char filename[8] = "malab\_\0";  
filename[5] = 'A' + params->fileNumber;  
  
do {  
sem\_wait(params->semaphore);  
double sum = 0;  
long file\_size = 0;  
FILE \*current\_file;  
do {  
fopen\_s(&current\_file, filename, "r");  
} while (current\_file == NULL);  
fseek(current\_file, 0, SEEK\_END);  
file\_size = ftell(current\_file);  
//int blocks = file\_size/16/ (G);  
int blocks = 10;  
char\* buffer = (char\*) malloc(file\_size\*2);  
for (int i = 0; i < blocks\*2;i++){  
int x = randRightVersion()%10;  
  
fseek(current\_file,x%(blocks)\*G,SEEK\_SET);  
fread(buffer+G\*i,G,1,current\_file);  
for(int d = 0; d < G; d++){  
sum += buffer[d];  
  
}  
}  
fclose(current\_file);  
sum/=(G \* blocks\*2); //avg  
printf("%d %.0f (0x%08x) random sum %s\n", params->id, sum, &sum, filename);  
free(buffer);  
sem\_post(params->semaphore);  
} while (threadsActivated);  
return NULL;  
}  
  
int main() {  
randRightVersion();  
  
printf("Before memory allocation\n");  
getchar();  
int [\*start](https://vk.com/id52435) = (int \*) VirtualAlloc(B, A \* 1024 \* 1024, MEM\_COMMIT, PAGE\_READWRITE); //выделение по адресу  
if (start == NULL) {  
  
start = (int \*) VirtualAlloc(NULL, A \* 1024 \* 1024, MEM\_COMMIT, PAGE\_READWRITE); // выделение памяти  
}  
printf("After memory allocation");  
getchar(); //ждем символ  
//выделение памяти под потоки  
pthread\_t [\*generators](https://vk.com/id27029174) = (pthread\_t \*) VirtualAlloc(NULL, D \* sizeof(pthread\_t), MEM\_COMMIT, PAGE\_READWRITE);  
generatorParams \*generatorsData = (generatorParams \*) VirtualAlloc(NULL, D \* sizeof(generatorParams), MEM\_COMMIT,  
PAGE\_READWRITE);  
pthread\_t [\*writers](https://vk.com/club92965792) = (pthread\_t \*) VirtualAlloc(NULL, 1 \* sizeof(pthread\_t), MEM\_COMMIT, PAGE\_READWRITE);  
fileWriterParams \*writersParams = (fileWriterParams \*) VirtualAlloc(NULL,sizeof(fileWriterParams), MEM\_COMMIT,  
PAGE\_READWRITE);  
pthread\_t [\*readers](https://vk.com/club24832977) = (pthread\_t \*) VirtualAlloc(NULL, I \* sizeof(pthread\_t), MEM\_COMMIT, PAGE\_READWRITE);  
fileReaderParams \*readersParams =

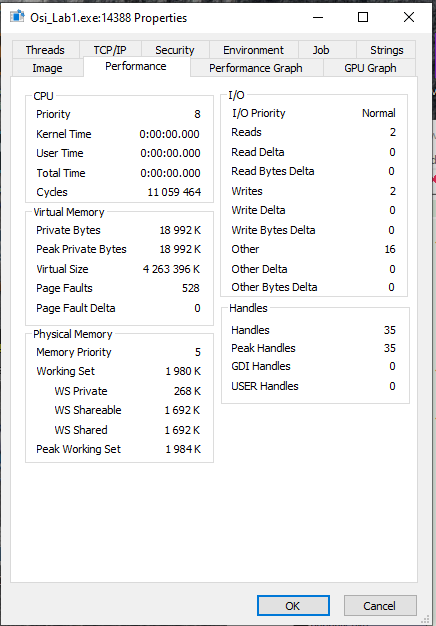
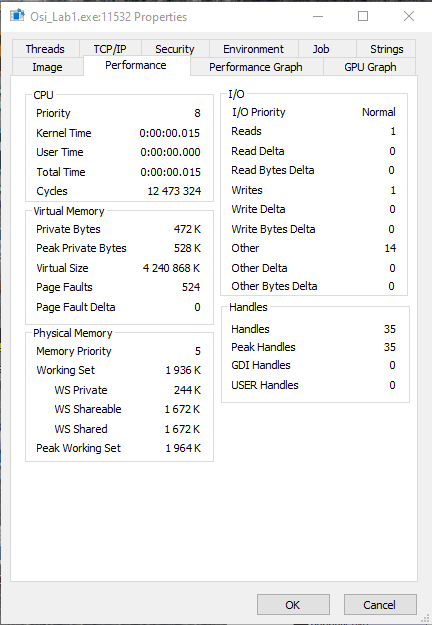
(fileReaderParams \*) VirtualAlloc(NULL,I \* sizeof(fileReaderParams), MEM\_COMMIT,  
PAGE\_READWRITE);  
  
writersParams->filesNumber = A/E; //считаем кол-во файлов  
if(A%E > 0)writersParams->filesNumber++;  
  
sem\_t \*sems = (sem\_t\*)VirtualAlloc(NULL, writersParams->filesNumber\*sizeof(sem\_t), MEM\_COMMIT, PAGE\_READWRITE );//если один поток занят то другие стопяться  
  
for(int i = 0; i < writersParams->filesNumber; i++) {  
sem\_init(sems[i], 0, 0);  
}  
int \*tmpointer = start;  
int amount = A \* 1024 \* 1024 / D / sizeof(int);  
//Генераторы для разных кусков памяти  
for (int i = 0; i < D; i++) {  
  
generatorsData[i].memoryRegion = (char \*) tmpointer;  
tmpointer += amount;  
generatorsData[i].numCount = amount;  
}  
generatorsData[D - 1].numCount += (A \* 1024 \* 1024 / sizeof(int)) % D;  
  
  
writersParams->semaphore = &sems[0];  
writersParams->start = start;  
for(int i = 0; i < I; i++){  
readersParams[i].fileNumber = (i+1) % writersParams->filesNumber ;  
readersParams[i].semaphore = &sems[readersParams[i].fileNumber];  
readersParams[i].id = i;  
}  
for (int i = 0; i < D; i++) {  
pthread\_create(&generators[i], NULL, threadGenerator, &generatorsData[i]);  
}  
//создание потков  
pthread\_create(writers, NULL, threadFileWriter, writersParams);  
for (int i = 0; i < I; i++) {  
pthread\_create(&readers[i], NULL, threadFileReader, &readersParams[i]);  
}  
getchar();// потоки работают и данные заполняються  
printf("Deactivate\n");  
threadsActivated = 0;  
for (int i = 0; i < D; i++)  
pthread\_join(generators[i], NULL);  
for (int i = 0; i < D; i++) {  
sem\_post(\*readersParams[i].semaphore);  
pthread\_join(readers[i], NULL);  
}  
pthread\_join(writers, NULL);  
VirtualFree(start, A \* 1024 \* 1024, MEM\_RELEASE);  
VirtualFree(generators, D \* sizeof(pthread\_t), MEM\_RELEASE);  
VirtualFree(generatorsData, D \* sizeof(generatorParams), MEM\_RELEASE);  
VirtualFree(writers, 1 \* sizeof(pthread\_t), MEM\_RELEASE);  
VirtualFree(writersParams, 1 \* sizeof(fileWriterParams), MEM\_RELEASE);  
VirtualFree(readers, 1 \* sizeof(pthread\_t), MEM\_RELEASE);  
VirtualFree(readersParams, 1 \* sizeof(fileReaderParams), MEM\_RELEASE);  
VirtualFree(sems, I\*sizeof(sem\_t), MEM\_RELEASE); }

Используя системные средства мониторинга, определите :

VMMap - Адрес начала области памяти размером 18МБ в адресном пространстве процесса



До аллокации, после аллокации (process explorer)



После заполнения участка данными, после деаллокации:

