Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

(Университет ИТМО)

Факультет Программной инженерии и компьютерной техники

Лабораторная работа №1

“Операционные системы”

Выполнили студенты

Группы P33211

Милеев Константин Андреевич

Кащенко Андрей

Преподаватель

Покид Александр Владимирович

Санкт-Петербург 2020

Задание

# Лабораторная работа №1

Вариант:

A=200;B=0x1D284B2F;C=mmap;D=25;E=46;F=nocache;G=27;H=seq;I=20;J=sum;K=cv

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 программы

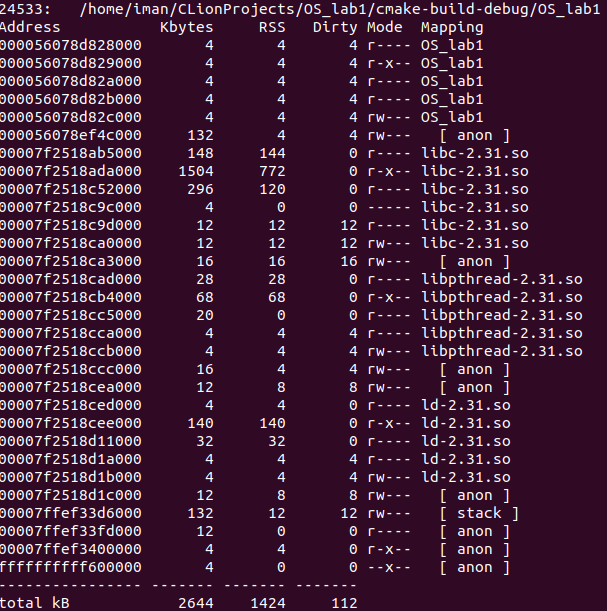
|  |
| --- |
| #include <stdio.h> #include <sys/mman.h> #include <pthread.h> #include <stdlib.h> #include <stdint.h>  *//A=200;B=0x1D284B2F;C=mmap;D=25;E=46;F=nocache;G=27;H=seq;I=20;J=sum;K=cv* #define ALLOCATE\_MEMORY 200 *// A* #define ADDRESS 0x1D284B2F *// B* *//#define mmap // C* #define GENERATED\_THREADS\_AMOUNT 25 *// D* #define FILE\_SIZE 46 *// E* *//#define nocache //F* #define BLOCK\_SIZE\_BYTES 27 *// G* *//#define seq // H* #define READ\_THREADS\_AMOUNT 20 *// I* *//#define sum // J* *//#define cv // K*  #define IN\_BYTES (1024 \* 1024)  #define INTEGER\_GENERATED ((ALLOCATE\_MEMORY \* IN\_BYTES) / GENERATED\_THREADS\_AMOUNT) #define REMAINDER\_GENERATED ((ALLOCATE\_MEMORY \* IN\_BYTES) % GENERATED\_THREADS\_AMOUNT)  #define COUNT\_FILES (ALLOCATE\_MEMORY / FILE\_SIZE)  #define URANDOM "/dev/urandom"  uint8\_t \*address; FILE \*\*files; pthread\_mutex\_t mutexes[COUNT\_FILES]; pthread\_cond\_t conds[COUNT\_FILES]; size\_t check[COUNT\_FILES];  struct GenDataThread {  size\_t start;  size\_t end;  FILE \*urandom; };  struct WritableDataThread {  int threadId;  int sizeMemory; };  struct ReadableDataThread {  int fileId;  int threadId; };  void generateInMemory(); void \*writeInMemory(void \*); void writeInFiles(); void \*writeInFile(void \*); void createFiles(); void readFromFiles(); \_Noreturn void \*readFromFile(void \*);  int main() {  printf("Allocate memory. Press a key to start");  getchar();  address = mmap(  (void\*) ADDRESS,  ALLOCATE\_MEMORY \* IN\_BYTES,  PROT\_READ | PROT\_WRITE,  MAP\_PRIVATE | MAP\_ANONYMOUS,  -1, 0);  if (address == MAP\_FAILED) {  perror("Error address a file");  exit(EXIT\_FAILURE);  }  printf("Write data to memory. Press a key to start");  getchar();  generateInMemory();  printf("Deallocate memory. Press a key to start");  getchar();  munmap((void \*) ADDRESS, ALLOCATE\_MEMORY \* IN\_BYTES);  printf("Infinity writing and reading. Press a key to start");  getchar();  int firstRun = 1;  createFiles();  while (1) {  address = mmap(  (void\*) ADDRESS,  ALLOCATE\_MEMORY \* IN\_BYTES,  PROT\_READ | PROT\_WRITE,  MAP\_PRIVATE | MAP\_ANONYMOUS,  -1, 0);  if (address == MAP\_FAILED) {  perror("Error address a file");  exit(EXIT\_FAILURE);  }  generateInMemory();  writeInFiles();  if (firstRun) {  readFromFiles();  firstRun = 0;  }  }  return 0; }  void generateInMemory() {  FILE \*urandom = fopen(URANDOM, "r");  pthread\_t memoryThreads[GENERATED\_THREADS\_AMOUNT];  struct GenDataThread \*genDataThread = (struct GenDataThread \*) malloc(GENERATED\_THREADS\_AMOUNT \* sizeof(struct GenDataThread));  for (int i = 0; i < GENERATED\_THREADS\_AMOUNT - 1; i++) {  genDataThread[i].start = (size\_t) i \* INTEGER\_GENERATED;  genDataThread[i].end = (size\_t) (i + 1) \* INTEGER\_GENERATED;  genDataThread[i].urandom = urandom;   pthread\_create(&(memoryThreads[i]), NULL, writeInMemory, &(genDataThread[i]));  }   genDataThread[GENERATED\_THREADS\_AMOUNT - 1].start = (size\_t) (GENERATED\_THREADS\_AMOUNT - 1) \* INTEGER\_GENERATED;  genDataThread[GENERATED\_THREADS\_AMOUNT - 1].end = (size\_t) (GENERATED\_THREADS\_AMOUNT - 1) \* INTEGER\_GENERATED + REMAINDER\_GENERATED;  genDataThread[GENERATED\_THREADS\_AMOUNT - 1].urandom = urandom;   pthread\_create(&(memoryThreads[GENERATED\_THREADS\_AMOUNT - 1]),  NULL,  writeInMemory,  &(genDataThread[GENERATED\_THREADS\_AMOUNT - 1]));   for (int i = 0; i < GENERATED\_THREADS\_AMOUNT; i++) {  pthread\_join(memoryThreads[i], NULL);  }   fclose(urandom); }  void \*writeInMemory(void \*dataThread) {  struct GenDataThread \*genDataThread = (struct GenDataThread \*) dataThread;  for (size\_t i = genDataThread->start; i < genDataThread->end; i++) {  uint8\_t number = 0;  fread(&number, sizeof(uint8\_t), 1, genDataThread->urandom);  address[i] = number;  }  return NULL; }  void writeInFiles() {  pthread\_t \*filesThreads = malloc(COUNT\_FILES \* sizeof(pthread\_t));  struct WritableDataThread \*writableDataThread = (struct WritableDataThread \*) malloc(COUNT\_FILES \* sizeof(struct WritableDataThread));  for (int i = 0; i < COUNT\_FILES; i++) {  writableDataThread[i].threadId = i;  writableDataThread[i].sizeMemory = ((i + 1) \* FILE\_SIZE) < ALLOCATE\_MEMORY  ? FILE\_SIZE \* IN\_BYTES  : (ALLOCATE\_MEMORY - i \* FILE\_SIZE) \* IN\_BYTES;   pthread\_create(&(filesThreads[i]), NULL, writeInFile, &(writableDataThread[i]));  }   for (int i = 0; i < COUNT\_FILES; i++) {  pthread\_join(filesThreads[i], NULL);  }   free(filesThreads); }  void \*writeInFile(void \*dataThread) {  struct WritableDataThread \*writableDataThread = (struct WritableDataThread \*) dataThread;  int threadId = writableDataThread->threadId;  int sizeMemory = writableDataThread->sizeMemory;  pthread\_mutex\_t \*mutex = &(mutexes[threadId]);  pthread\_cond\_t \*cond = &(conds[threadId]);  FILE \*file = files[threadId];   fseek(file, 0, SEEK\_SET);  size\_t offsetMinor = threadId \* FILE\_SIZE \* IN\_BYTES;  for (int i = 0; i < sizeMemory; i += BLOCK\_SIZE\_BYTES) {  pthread\_mutex\_lock(mutex);  check[threadId] = 1;   int nelts = (i + BLOCK\_SIZE\_BYTES) < sizeMemory? BLOCK\_SIZE\_BYTES: (sizeMemory - i);  size\_t offsetMajor = offsetMinor + i;  fwrite((address + offsetMajor), sizeof(uint8\_t), nelts, file);  fflush(file);   check[threadId] = 0;  pthread\_cond\_signal(cond);  pthread\_mutex\_unlock(mutex);  }   return NULL; }  void createFiles() {  files = (FILE \*\*) malloc(COUNT\_FILES \* sizeof(FILE \*));  char filename[20];  for (int i = 0; i < COUNT\_FILES; i++) {  sprintf(filename, "file%d", i);  files[i] = fopen(filename, "w+b");   pthread\_mutex\_init(&(mutexes[i]), NULL);  pthread\_cond\_init(&(conds[i]), NULL);  } }  void readFromFiles() {  pthread\_t \*readFiles = (pthread\_t \*) malloc(READ\_THREADS\_AMOUNT \* sizeof(pthread\_t));  struct ReadableDataThread \*readableDataThread = (struct ReadableDataThread \*) malloc(READ\_THREADS\_AMOUNT \* sizeof(struct ReadableDataThread));  for (int i = 0; i < READ\_THREADS\_AMOUNT; i++) {  readableDataThread[i].threadId = i;  readableDataThread[i].fileId = i % COUNT\_FILES;   pthread\_create(&(readFiles[i]), NULL, readFromFile, &(readableDataThread[i]));  }  }  \_Noreturn void \*readFromFile(void \*dataThread) {  struct ReadableDataThread \*readableDataThread = (struct ReadableDataThread \*) dataThread;  int fileId = readableDataThread->fileId;  pthread\_mutex\_t \*mutex = &(mutexes[fileId]);  pthread\_cond\_t \*cond = &(conds[fileId]);  FILE \*file = files[fileId];   while (1) {  fseek(file, 0, SEEK\_SET);  unsigned long long sum = 0;  for (int i = 0; i < FILE\_SIZE \* IN\_BYTES; i++) {  pthread\_mutex\_lock(mutex);   while (check[fileId]) {  pthread\_cond\_wait(cond, mutex);  }   uint8\_t number;  fread(&number, sizeof(uint8\_t), 1, file);  sum += number;   pthread\_mutex\_unlock(mutex);  }  printf("Sum in file %d with thread %d is equal %llu\n", fileId, readableDataThread->threadId,sum);  } } |

Сбор статистики

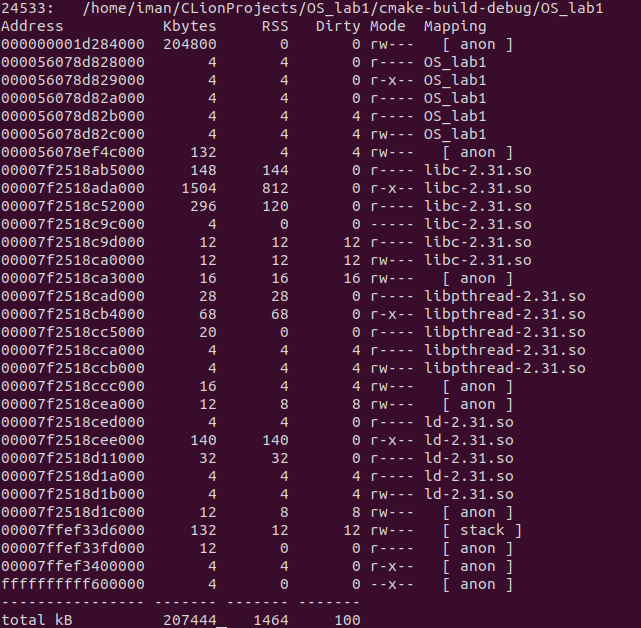
**ps -A | grep <name>**

**pmap -x <PID>**

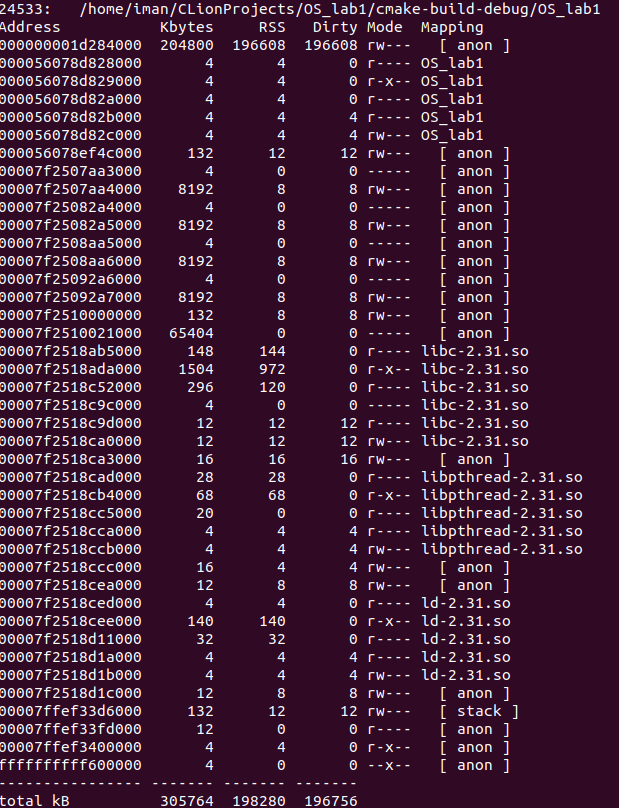
*Before allocation*

**

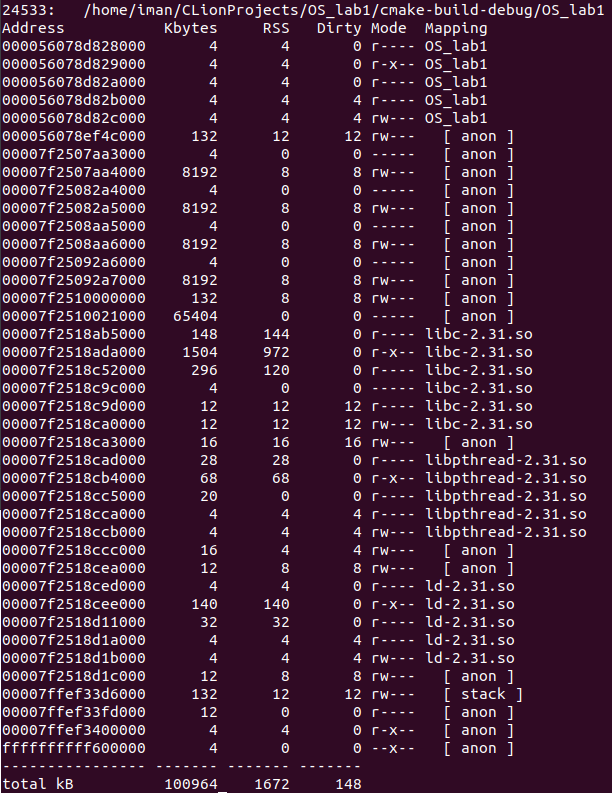
*After allocation*

**

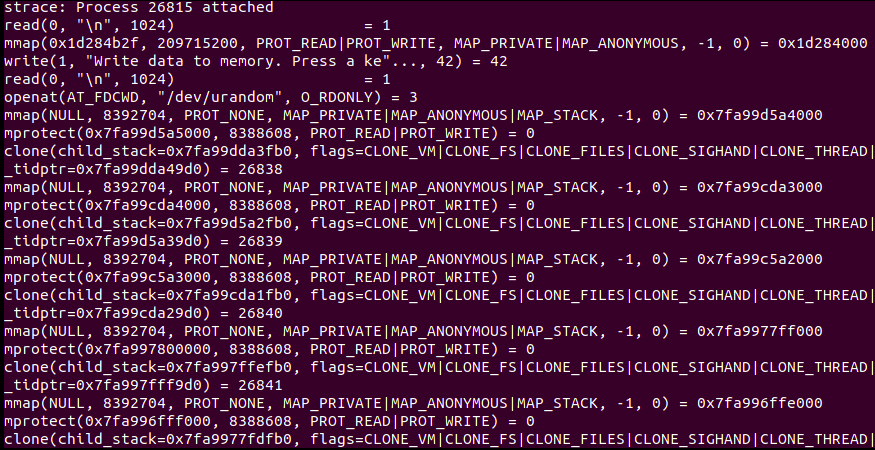
*After generate*

**

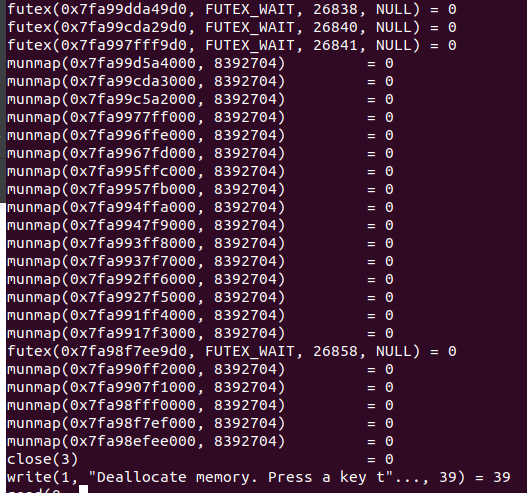
*After deallocation*

**

*Системные вызовы:*

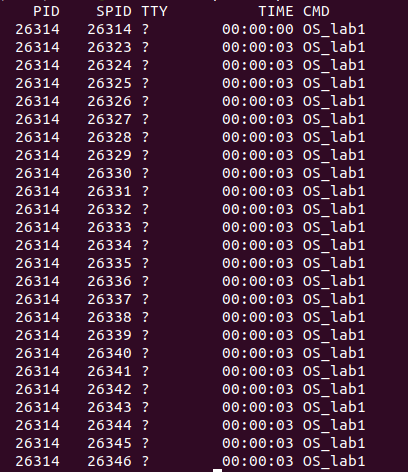
**

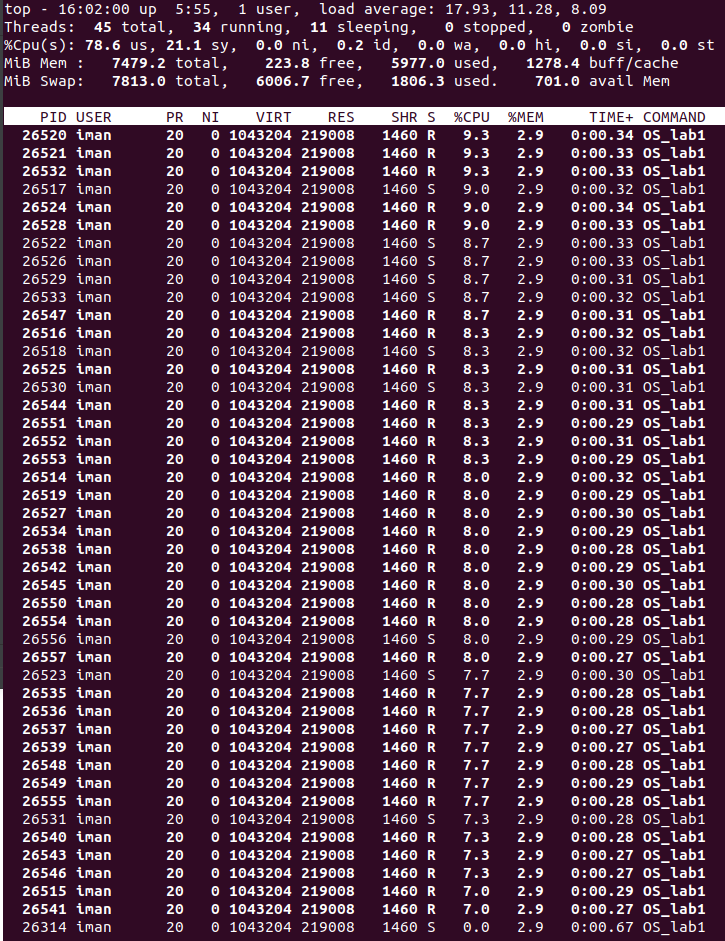
*…*

**

*Threads процессы*

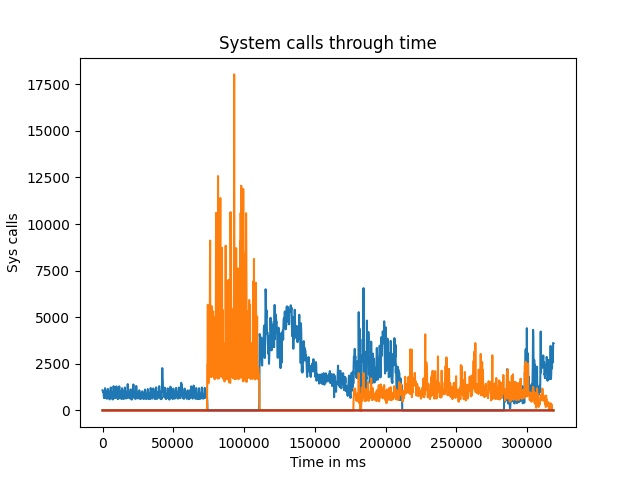
*ps -T -p <pid>*

**

**

*Запуск скрипта systemtap во время бесконечно цикла*

**



Оранжевая - write

Синяя - read

Красная - open

Зеленая - close

Вывод

В ходе выполнения лабораторной работы мы написали многопоточную программу на языке C и поработали с инструментами мониторинга ОС и процессов.