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”

Выполнил: студент гр. P33113
Прикота Виталий Александрович
Преподаватель:
Покид Александр Владимирович

Санкт-Петербург 2020

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:
- До аллокации
- После аллокации
- После заполнения участка данными
- 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.

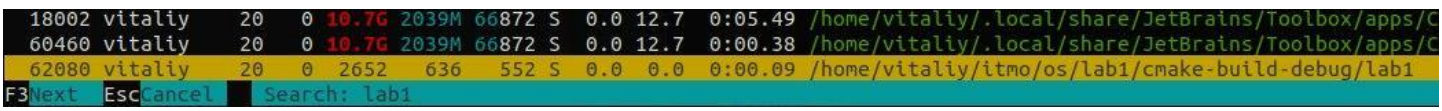
Вариант:
A=276;B=0x28B070E0;C=mmap;D=74;E=47;F=nocache;G=36;H=random;I=139;J=max;K=sem

Код программы

https://github.com/rbetik12/OS-memory-allocator

Замеры памяти

До аллокации:



После аллокации (из /proc/status):

```
VmPeak:   272184 kB
VmSize:   272184 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:       560 kB
VmRSS:       560 kB
```

После заполнения данными (из pmap -x):

```
6631:   /home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000028b07000  269532  269532       0 rw-s- output
000055789ab0e000       4       4       0 r---- lab1
000055789ab0f000       8       8       0 r-x-- lab1
000055789ab11000       4       4       0 r---- lab1
000055789ab12000       4       4       4 r---- lab1
000055789ab13000       4       4       4 rw--- lab1
```

После деаллокации:

```
 66785 vitaliy    20   0  418M  1960  1712 S  0.0  0.0  1:45.60 /home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
 67001 vitaliy    20   0  568M 61172 49424 S  0.0  0.4  0:00.00 /opt/google/chrome/chrome --type=utility --utility-su
```

Карта памяти процесса

```
28b07000-3923e000 rw-s 00000000 08:13 5775723
/home/vitaliy/itmo/os/lab1/cmake-build-debug/output
55789ab0e000-55789ab0f000 r--p 00000000 08:13 5775363
/home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
55789ab0f000-55789ab11000 r-xp 00001000 08:13 5775363
/home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
55789ab11000-55789ab12000 r--p 00003000 08:13 5775363
/home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
55789ab12000-55789ab13000 r--p 00003000 08:13 5775363
/home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
55789ab13000-55789ab14000 rw-p 00004000 08:13 5775363
/home/vitaliy/itmo/os/lab1/cmake-build-debug/lab1
55789ab14000-55789ab16000 rw-p 00000000 00:00 0
55789af68000-55789af89000 rw-p 00000000 00:00 0                          [heap]
7fc278000000-7fc278021000 rw-p 00000000 00:00 0
7fc278021000-7fc27c000000 ---p 00000000 00:00 0
7fc27e6c4000-7fc27e6c5000 ---p 00000000 00:00 0
7fc27e6c5000-7fc27eec5000 rw-p 00000000 00:00 0
7fc27eec5000-7fc27eec6000 ---p 00000000 00:00 0
7fc27eec6000-7fc27f6c6000 rw-p 00000000 00:00 0
7fc27f6c6000-7fc27f6c7000 ---p 00000000 00:00 0
7fc27f6c7000-7fc27fec7000 rw-p 00000000 00:00 0
7fc27fec7000-7fc27fec8000 ---p 00000000 00:00 0
7fc27fec8000-7fc2806c8000 rw-p 00000000 00:00 0
7fc29c000000-7fc29c021000 rw-p 00000000 00:00 0
7fc29c021000-7fc2a0000000 ---p 00000000 00:00 0
7fc2a370e000-7fc2a3711000 rw-p 00000000 00:00 0
7fc2a3711000-7fc2a3736000 r--p 00000000 08:13 12060802                   /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7fc2a3736000-7fc2a38ae000 r-xp 00025000 08:13 12060802                   /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7fc2a38ae000-7fc2a38f8000 r--p 0019d000 08:13 12060802                   /usr/lib/x86_64-linux-
gnu/libc-2.31.so
```

```
7fc2a38f8000-7fc2a38f9000 ---p 001e7000 08:13 12060802                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7fc2a38f9000-7fc2a38fc000 r--p 001e7000 08:13 12060802                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7fc2a38fc000-7fc2a38ff000 rw-p 001ea000 08:13 12060802                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7fc2a38ff000-7fc2a3903000 rw-p 00000000 00:00 0
7fc2a3903000-7fc2a390a000 r--p 00000000 08:13 12060815                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7fc2a390a000-7fc2a391b000 r-xp 00007000 08:13 12060815                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7fc2a391b000-7fc2a3920000 r--p 00018000 08:13 12060815                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7fc2a3920000-7fc2a3921000 r--p 0001c000 08:13 12060815                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7fc2a3921000-7fc2a3922000 rw-p 0001d000 08:13 12060815                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7fc2a3922000-7fc2a3928000 rw-p 00000000 00:00 0
7fc2a393d000-7fc2a393e000 r--p 00000000 08:13 12058697                    /usr/lib/x86_64-linux-
gnu/ld-2.31.so
7fc2a393e000-7fc2a3961000 r-xp 00001000 08:13 12058697                    /usr/lib/x86_64-linux-
gnu/ld-2.31.so
7fc2a3961000-7fc2a3969000 r--p 00024000 08:13 12058697                    /usr/lib/x86_64-linux-
gnu/ld-2.31.so
7fc2a396a000-7fc2a396b000 r--p 0002c000 08:13 12058697                    /usr/lib/x86_64-linux-
gnu/ld-2.31.so
7fc2a396b000-7fc2a396c000 rw-p 0002d000 08:13 12058697                    /usr/lib/x86_64-linux-
gnu/ld-2.31.so
7fc2a396c000-7fc2a396d000 rw-p 00000000 00:00 0
7fff7350b000-7fff7352c000 rw-p 00000000 00:00 0                           [stack]
7fff735d5000-7fff735d8000 r--p 00000000 00:00 0                           [vvar]
7fff735d8000-7fff735d9000 r-xp 00000000 00:00 0                           [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                   [vsyscall]
```

Агрегированное значение

Максимум значений, записанных в файлы: 255. Так как программа оперирует одно байтовыми беззнаковыми целыми числами.

Время затраченное на ввод-вывод

Подсчет производился с помощью конструкции:

```
struct timespec start, finish;
double elapsed;
clock_gettime(CLOCK_MONOTONIC, &start);
```

…Код, время исполнения которого мы считаем…

```
clock_gettime(CLOCK_MONOTONIC, &finish);

elapsed = (finish.tv_sec - start.tv_sec);
elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
```

Переменная elapsed хранит время исполнения участка кода

Время затраченное на вывод в файлы: 0.7 секунд

Время затраченное на чтение из файлов: 6.3 секунды

Трассировка системных вызовов

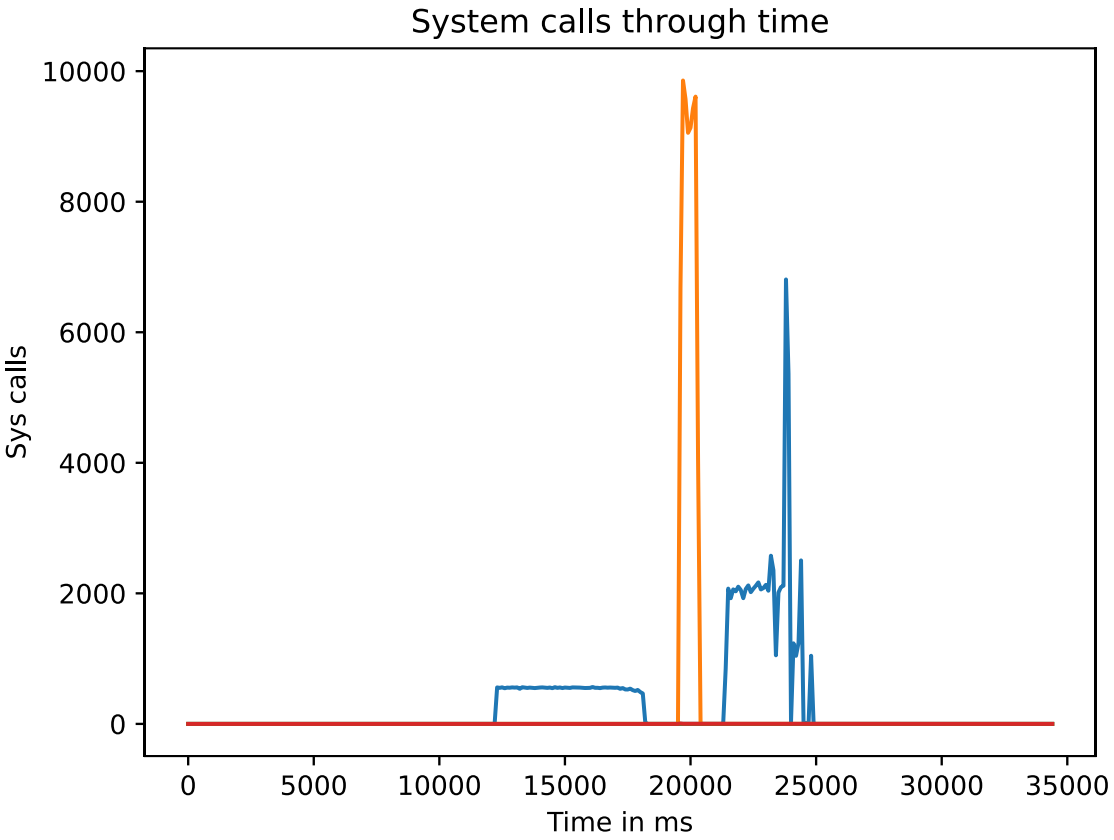
https://raw.githubusercontent.com/rbetik12/OS-memory-allocator/master/kernel.svg

График системных характеристик, посчитанных с помощью stap

Оранжевая линия – вызовы write

Синия линия – вызовы read

Красная и зеленая – вызовы open и close



Код скрипта stap (производит измерения раз в 100 мс):

```
global read, write, start, open, close

probe begin {
   start = gettimeofday_s()
}
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(100) {
   printf("%d\t%d\t%d\t%d\t%16d\n", read, write, open, close, task_stime())
   read=0
   write=0
   open = 0
```

```
   close = 0
}
```

Вывод

В ходе выполнения лабораторной работы я познакомился с новыми инструментами мониторинга ОС и процессов, например system tap или pmap.