ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет безопасности информационных технологий

Кафедра проектирования и безопасности компьютерных систем

Дисциплина:

«Операционные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

Выполнил:

Студент группы N3248

Назаров Максим Вячеславович.

Проверил:

Савков Сергей Витальевич.

Санкт-Петербург 2022г.

Лаб 6.

Протестировать функцию malloc/free и построить график зависимости времени выделения от размера запрашиваемой памяти.

Ход работы:

Была написана программа, которая аллоцирует 100 000 раз ячейки памяти, размерами от 1Mb до 1Gb, выводит время, за которое это происходит.

Представлены реализации программы для malloc, calloc.

Программа использованием malloc:

```
#include <cstdio>
#include <cstdlib>
#include <ctime>

int main() {
    long long *memory;
    clock_t start, end;
    int size;
    for (int i = 1; i <= 30; i++) {
        size = 1 << i;
        start = clock();
        for (int i = 0; i < 100000; i++) {
            memory = (long long *) malloc(size);
            free(memory);
        }
        end = clock();
        double time_in_sec = (double) (end - start) / CLOCKS_PER_SEC;
        printf("Size: %d B  Time:%f sec\n", size, time_in_sec);
    }
    return 0;
}
```

Результат программы для ОС Windows 10:

```
Size: 2 B  Time:0.005000 sec
Size: 4 B  Time:0.004000 sec
Size: 8 B  Time:0.005000 sec
Size: 16 B  Time:0.005000 sec
Size: 32 B  Time:0.004000 sec
Size: 64 B  Time:0.004000 sec
Size: 128 B  Time:0.005000 sec
Size: 256 B  Time:0.005000 sec
Size: 512 B  Time:0.005000 sec
Size: 1024 B  Time:0.004000 sec
Size: 2048 B  Time:0.005000 sec
Size: 4096 B  Time:0.005000 sec
Size: 8192 B  Time:0.005000 sec
Size: 16384 B  Time:0.004000 sec
Size: 32768 B  Time:0.015000 sec
Size: 65536 B  Time:0.014000 sec
Size: 131072 B  Time:0.014000 sec
Size: 262144 B  Time:0.016000 sec
Size: 524288 B  Time:0.016000 sec
Size: 1048576 B  Time:0.923000 sec
Size: 2097152 B  Time:0.984000 sec
Size: 4194304 B  Time:0.953000 sec
Size: 8388608 B  Time:0.995000 sec
Size: 16777216 B  Time:0.978000 sec
Size: 33554432 B  Time:0.980000 sec
Size: 67108864 B  Time:1.054000 sec
Size: 134217728 B  Time:1.120000 sec
Size: 268435456 B  Time:1.213000 sec
Size: 536870912 B  Time:1.594000 sec
Size: 1073741824 B  Time:2.111000 sec
```

Результат программы для ОС Kali Linux:

```
Size: 2 B  Time:0.000826 sec
Size: 4 B  Time:0.000802 sec
Size: 8 B  Time:0.000837 sec
Size: 16 B  Time:0.000855 sec
Size: 32 B  Time:0.000844 sec
Size: 64 B  Time:0.000845 sec
Size: 128 B  Time:0.000858 sec
Size: 256 B  Time:0.000852 sec
Size: 512 B  Time:0.000836 sec
Size: 1024 B  Time:0.000838 sec
Size: 2048 B  Time:0.002536 sec
Size: 4096 B  Time:0.002367 sec
Size: 8192 B  Time:0.002412 sec
Size: 16384 B  Time:0.002515 sec
Size: 32768 B  Time:0.002463 sec
Size: 65536 B  Time:0.002596 sec
Size: 131072 B  Time:0.002559 sec
Size: 262144 B  Time:0.002251 sec
Size: 524288 B  Time:0.002285 sec
Size: 1048576 B  Time:0.002343 sec
Size: 2097152 B  Time:0.002251 sec
Size: 4194304 B  Time:0.002279 sec
Size: 8388608 B  Time:0.002251 sec
Size: 16777216 B  Time:0.002253 sec
Size: 33554432 B  Time:0.306571 sec
Size: 67108864 B  Time:0.314515 sec
Size: 134217728 B  Time:0.334341 sec
Size: 268435456 B  Time:0.370364 sec
Size: 536870912 B  Time:0.473465 sec
Size: 1073741824 B  Time:0.620925 sec
```

Программа с использованием calloc:

```
#include <cstdio>
#include <cstdlib>
#include <ctime>

int main() {
    long long *memory;
    clock_t start, end;
    int size;
    for (int i = 1; i <= 30; i++){
        size = 1<<i;
                start = clock();
        for (int i = 0; i < 100000; i++){
            memory = (long long*)calloc(size, sizeof(int));
            free(memory);
        }
        end = clock();
        double time_in_sec = (double)(end-start) / CLOCKS_PER_SEC;
        printf("Size: %d B  Time:%f sec\n", size, time_in_sec);
    }
    return 0;
}
```

Результат программы для ОС Windows 10:

```
Size: 2 B  Time:0.005000 sec
Size: 4 B  Time:0.005000 sec
Size: 8 B  Time:0.005000 sec
Size: 16 B  Time:0.005000 sec
Size: 32 B  Time:0.006000 sec
Size: 64 B  Time:0.006000 sec
Size: 128 B  Time:0.006000 sec
Size: 256 B  Time:0.006000 sec
Size: 512 B  Time:0.008000 sec
Size: 1024 B  Time:0.011000 sec
Size: 2048 B  Time:0.018000 sec
Size: 4096 B  Time:0.031000 sec
Size: 8192 B  Time:0.065000 sec
Size: 16384 B  Time:0.118000 sec
Size: 32768 B  Time:0.217000 sec
Size: 65536 B  Time:0.419000 sec
Size: 131072 B  Time:0.870000 sec
Size: 262144 B  Time:0.946000 sec
Size: 524288 B  Time:0.939000 sec
Size: 1048576 B  Time:0.953000 sec
Size: 2097152 B  Time:0.952000 sec
Size: 4194304 B  Time:0.958000 sec
Size: 8388608 B  Time:0.978000 sec
Size: 16777216 B  Time:1.002000 sec
Size: 33554432 B  Time:1.098000 sec
Size: 67108864 B  Time:1.189000 sec
Size: 134217728 B  Time:1.515000 sec
Size: 268435456 B  Time:2.010000 sec
Size: 536870912 B  Time:2.409000 sec
Size: 1073741824 B  Time:3.069000 sec
```

Результат работы программы для ОС Kali Linux:

```
Size: 2 B  Time:0.002529 sec
Size: 4 B  Time:0.002080 sec
Size: 8 B  Time:0.002163 sec
Size: 16 B  Time:0.001447 sec
Size: 32 B  Time:0.003718 sec
Size: 64 B  Time:0.003125 sec
Size: 128 B  Time:0.003351 sec
Size: 256 B  Time:0.003394 sec
Size: 512 B  Time:0.003832 sec
Size: 1024 B  Time:0.005160 sec
Size: 2048 B  Time:0.008191 sec
Size: 4096 B  Time:0.015386 sec
Size: 8192 B  Time:0.028186 sec
Size: 16384 B  Time:0.054463 sec
Size: 32768 B  Time:0.105328 sec
Size: 65536 B  Time:0.219131 sec
Size: 131072 B  Time:0.491960 sec
Size: 262144 B  Time:1.217115 sec
Size: 524288 B  Time:2.490780 sec
Size: 1048576 B  Time:5.104680 sec
Size: 2097152 B  Time:10.088279 sec
Size: 4194304 B  Time:65.219734 sec
Size: 8388608 B  Time:0.304662 sec
Size: 16777216 B  Time:0.314803 sec
Size: 33554432 B  Time:0.333778 sec
Size: 67108864 B  Time:0.369636 sec
Size: 134217728 B  Time:0.442664 sec
Size: 268435456 B  Time:0.623171 sec
Size: 536870912 B  Time:0.624251 sec
Size: 1073741824 B  Time:0.624466 sec
```

Усложнённый вариант:

тестируем аллокаторы jemalloc и tcmalloc

Jemalloc – это универсальная реализация malloc, в которой делается акцент на предохранении от фрагментации и поддержке масштабируемой конкурентности. **jemalloc** впервые начала использоваться во FreeBSD в 2005 году в качестве аллокатора libc и с тех пор нашла применение во многих приложениях благодаря своему предсказуемому поведению.

tcmalloc - это библиотека управления памятью, открытая Google как альтернатива glibc malloc. Он был использован в известных программах, таких как Chrome и Safari.

Согласно официальному отчету о тестировании, ptmalloc требуется около 300 наносекунд для выполнения malloc и освобождения на машине P4 с частотой 2,8 ГГц (для небольших объектов). Та же операция в версии TCMalloc занимает всего около 50 наносекунд.

Установка tcmalloc:
$ sudo apt install libtcmalloc-minimal4

$ export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libtcmalloc_minimal.so.4

$ sudo ln -s /usr/lib/x86_64-linux-gnu/libtcmalloc_minimal.so.4 /usr/lib/x86_64-linux-gnu/libtcmalloc_minimal.so

Компиляция: g++ -ltcmalloc_minimal jemalloc.c

Установка jemalloc:
$ sudo apt-get install libjemalloc-dev

$ export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libjemalloc.so

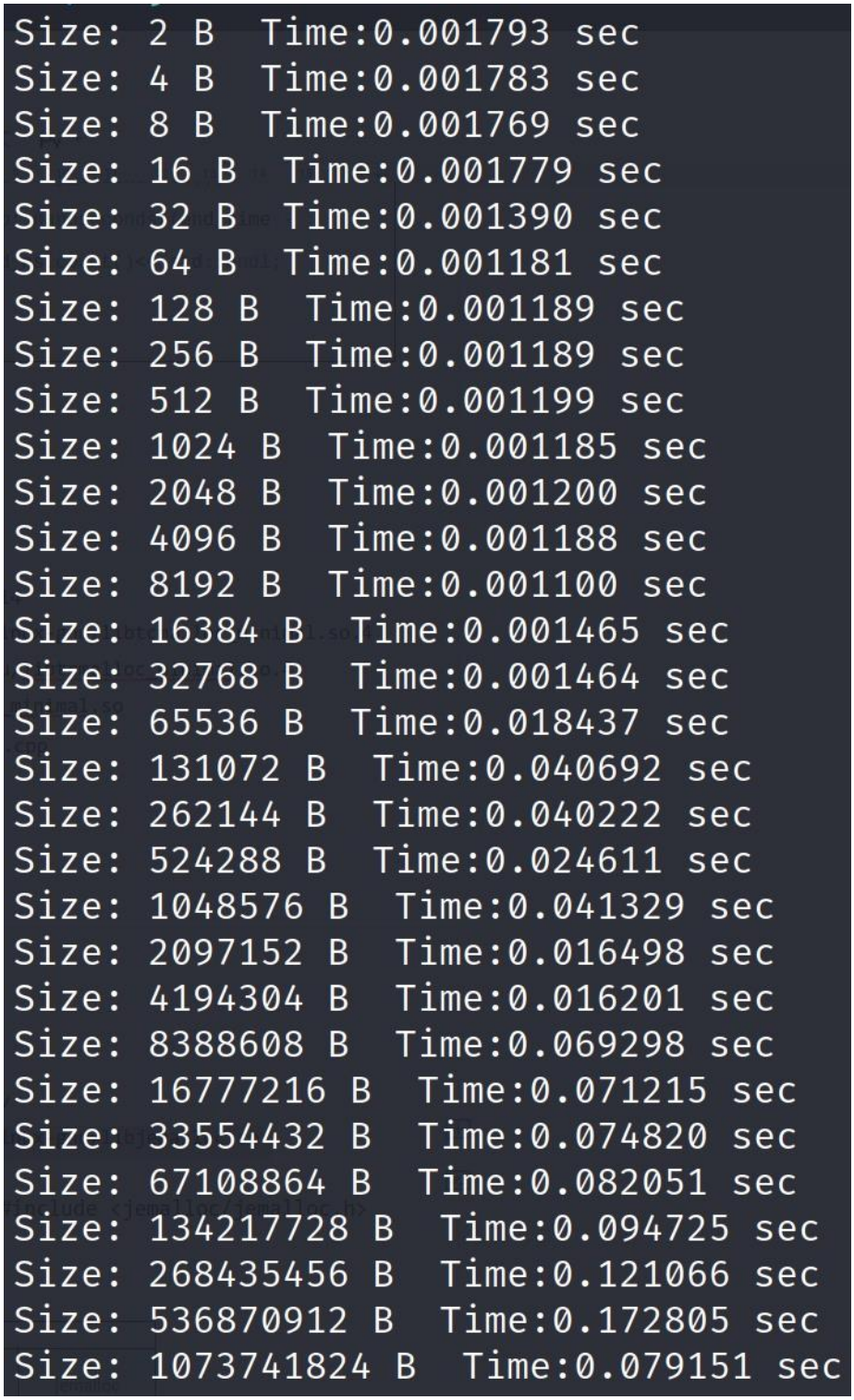
При этом мы добавляем заголовочный файл jemalloc/jemalloc.h
Для компиляции мы используем: g++ -ljemalloc tcmalloc.c

Программа с использованием jemalloc:

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
```

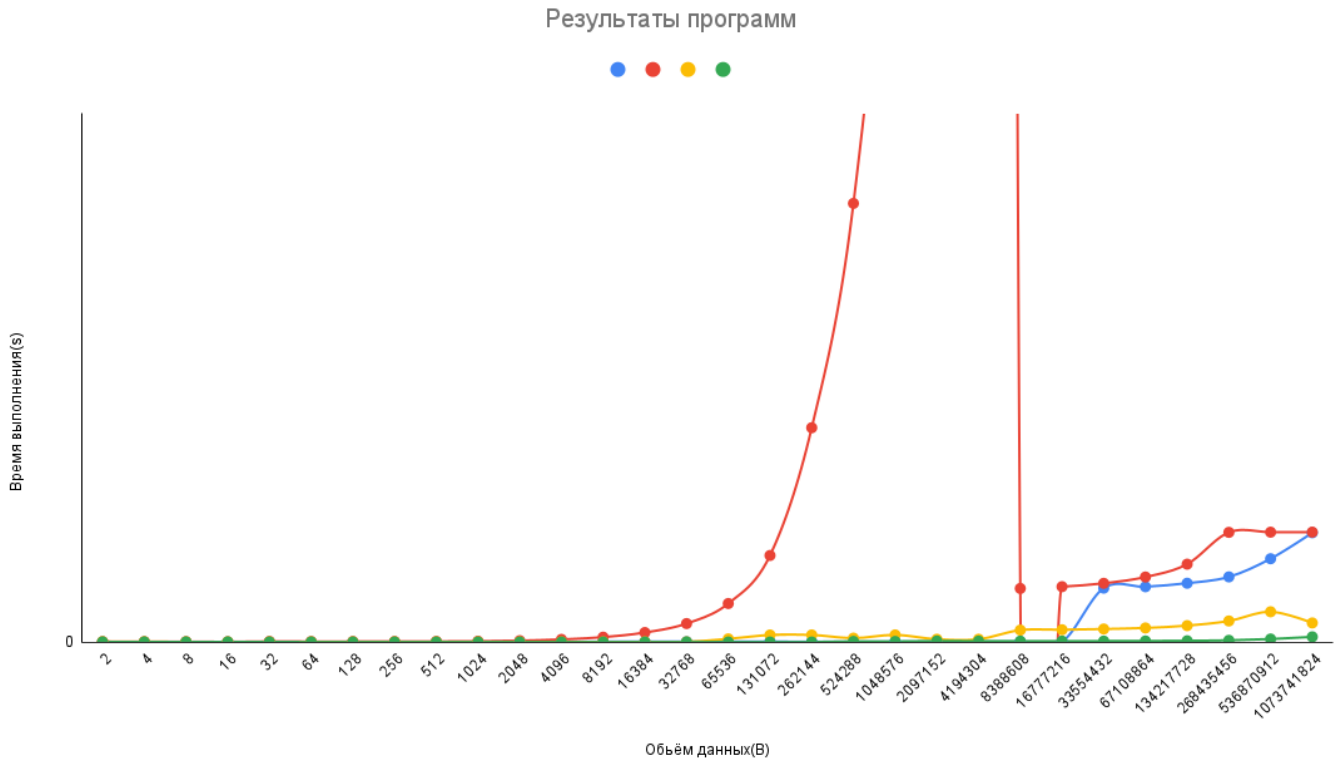
```c
#include <jemalloc/jemalloc.h>



int main() {
    long long *memory;
    clock_t start, end;
    int size;
    for (int i = 1; i <= 30; i++){
        size = 1<<i;
                start = clock();
        for (int i = 0; i < 100000; i++){
            memory = (long long*)malloc(size, sizeof(int));
            free(memory);
        }
        end = clock();
        double time_in_sec = (double)(end-start) / CLOCKS_PER_SEC;
        printf("Size: %d B  Time:%f sec\n", size, time_in_sec);
    }
    return 0;
}
```

Результат программы с использованием jemalloc на ОС Kali Linux:



Код для программы с tcmalloc ровно такой же, отличаются лишь предзагруженные динамические библиотеки.

График для malloc/calloc/jemalloc/tcmalloc:



На графике не отмечено одно значение которое выдала программа, а именно 65,219734 секунд у calloc

Обьём данных(B)	malloc	calloc	jemalloc	tcmalloc
	0,0008	0,0025	0,0017	0,0011
	0,0008	0,002	0,0017	0,0010
	0,0008	0,0021	0,0013	0,0010
	0,0008	0,0014	0,0011	0,0010
	0,0008	0,0037	0,0011	0,0007
	0,0008	0,0031	0,0011	0,0005
1	0,0008	0,0033	0,0011	0,0005
2	0,0008	0,0033	0,0011	0,0005
5	0,0008	0,0038	0,0011	0,0005
10	0,0008	0,005	0,0011	0,0006
20	0,0025	0,0081	0,0011	0,0006
40	0,0023	0,0153	0,0011	0,000
81	0,0024	0,0281	0,0011	0,0006
163	0,0025	0,0544	0,0014	0,0006
327	0,0024	0,1053	0,0014	0,0006
655	0,0025	0,2191	0,018	0,0006
1310	0,0025	0,491	0,0404	0,0005
2621	0,0022	1,2171	0,0407	0,0005
5242	0,0022	2,490	0,0227	0,0040
10485	0,0023	5,104	0,0409	0,0037
20971	0,0022	10,0882	0,0162	0,0062
41943	0,0022	65,2197	0,0164	0,0056
83886	0,0022	0,3046	0,0682	0,0056
167772	0,0022	0,3148	0,0702	0,0057
335544	0,3065	0,3337	0,0744	0,0060
671088	0,3145	0,3696	0,0804	0,00

1342177	0,3343	0,4426	0,0935	0,0075
2684354	0,3703	0,6231	0,1200	0,0101
5368709	0,4734	0,6242	0,17	0,0176
10737418	0,6209	0,6244	0,1088	0,0299

Вывод: Засчёт использование многопоточности для небольших значений и хорошей оптимизации(на моей машине 6-ть ядер) tcmalloc справляется с поставленной задачей намного лучше чем другие исследованные аллокаторы. Стоит отметить, что jemalloc также хорошо справился с задачей на фоне malloc и calloc, скорее всего это связано с использованием красно-черного дерева для большого значения аргумента(хотя стоит заметить что идея с “аренами” для каждого потока достаточно интересна). Отметим также, что malloc справляется с поставленной задачей лучше на меньших обьёмах данных по сравнению с calloc-ом(первоначальная инициализация calloc-ом безусловно даёт о себе знать). Такой резкий скачок может быть вызван фрагментацией памяти из-за отсутствия «осведомленности о потоках» в вызовах calloc, реализованных по умолчанию.