Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

“Операционные системы”

Лабораторная работа №6

“malloc/free”

Выполнил:

ст. группы N3246 Цыдыпов А.О.

Проверил:

Ханов А.Р.

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург

2022 г.

Задание:

Протестировать функцию malloc/free и построить график зависимости времени выделения от размера запрашиваемой памяти.
Либо винда, либо линукс

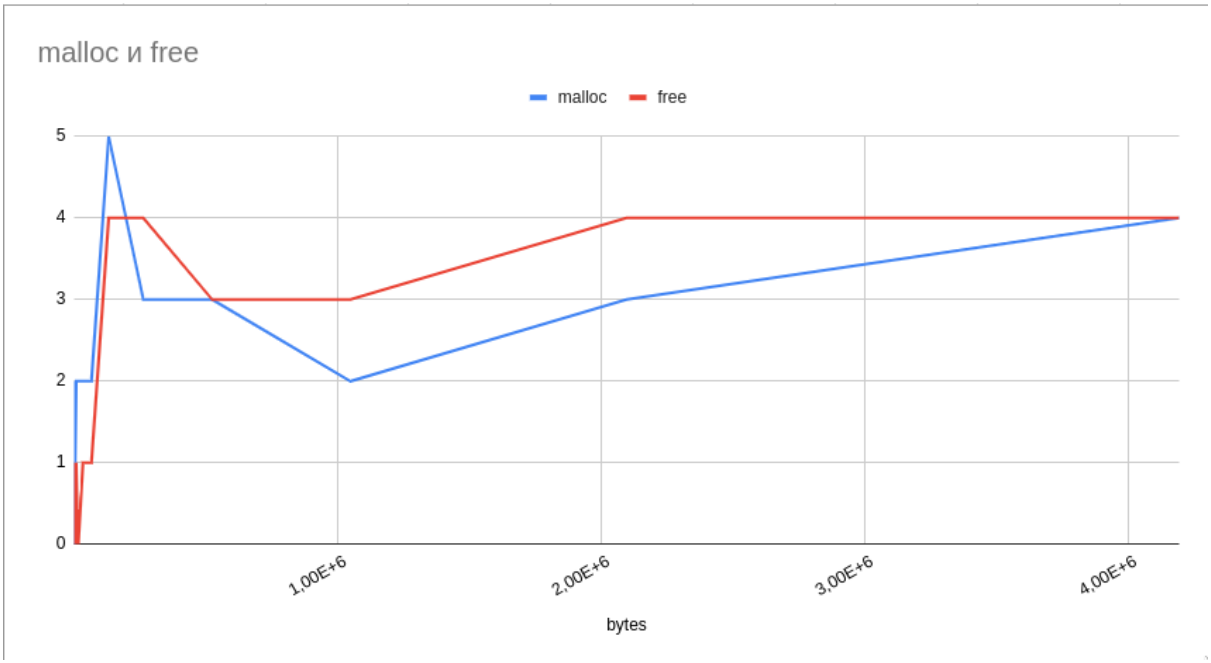
Сложный (или)
1. Сравнить с другими малоками
2. Тестировать на живом процессе

Ход работы:

Напишем программу, которая будет последовательно выделять и очищать память в степенях двойки, параллельно засекая время затрачиваемое на выполнение malloc и free.
./main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define _8G 8589934592

int main(){
  clock_t mallocB, mallocA, freeB, freeA;
  FILE * logFile = fopen("log.csv", "w");
  fprintf(logFile, "bytes,malloc,free\n");
  for(long long i = 1; i < _8G; i*=2){
    mallocB = clock();
    void* tmp = malloc(i);
    mallocA = clock();
    freeB = clock();
    free(tmp);
    freeA = clock();
    fprintf(logFile, "%lld,%d,%d\n", i, (mallocA - mallocB), (freeA -
freeB));
  }
  return 0;
}
```

Получим следующий график:



Легко заметить, что график не растет линейно и имеет свои пики. Происходит это возможно, из-за того что на самом деле выделяется больше памяти, чем мы требуем, чтобы сократить число вызываний функции malloc в будущем.

Усложненный вариант:

Изначально, я хотел сделать это с помощью LD_PRELOAD, однако написав библиотеку и запустив ее, я понял что логается слишком много строк и мне стало лень чистить выходные данные от огромных пиков.

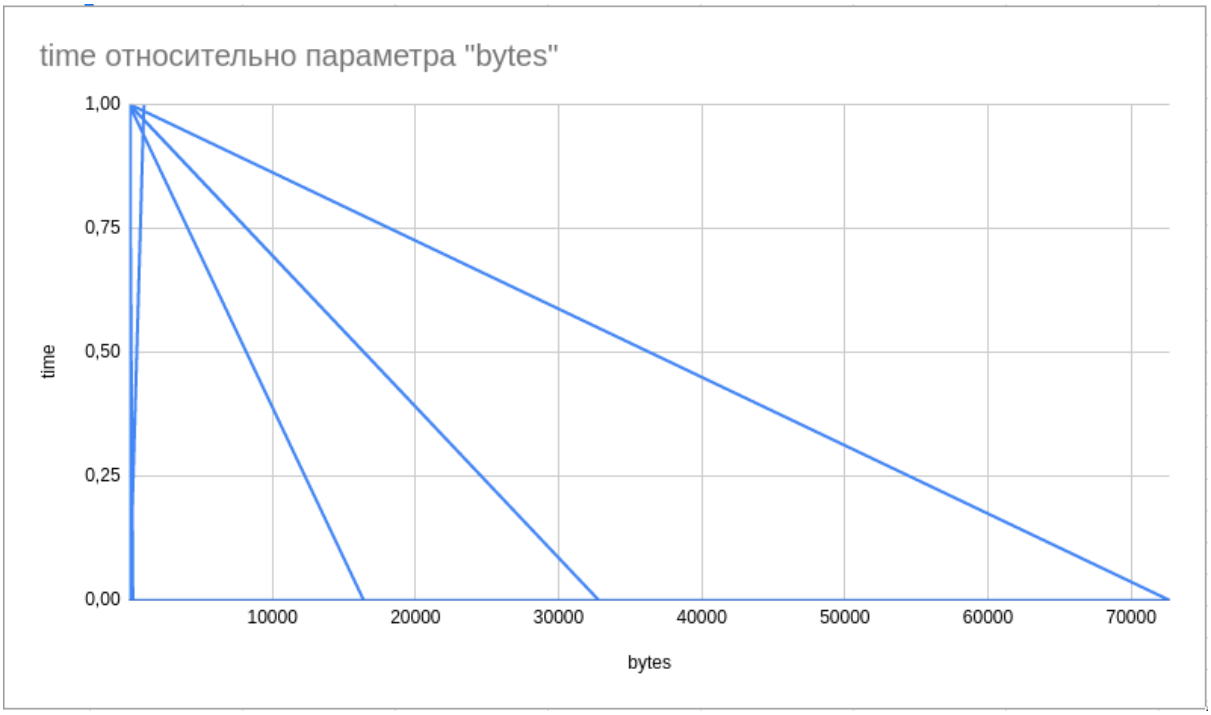
Попытка 1:

./hook.c

```c
#define _GNU_SOURCE

#include <stdio.h>
#include <dlfcn.h>
#include <time.h>

static void* (*real_malloc)(size_t)=NULL;
clock_t b,a;

static void mtrace_init(void)
{
    real_malloc = dlsym(RTLD_NEXT, "malloc");
    if (NULL == real_malloc) {
        fprintf(stderr, "Error in `dlsym`: %s\n", dlerror());
    }
}
```

```c
void *malloc(size_t size)
{
    if(real_malloc==NULL) {
        mtrace_init();
    }
    b = clock();
    void *p = NULL;
    // fprintf(stderr, "malloc(%d) = ", size);
    a = clock();
    p = real_malloc(size);
    fprintf(stderr, "%ld,%ld\n", size, (a-b));
    return p;
}
```

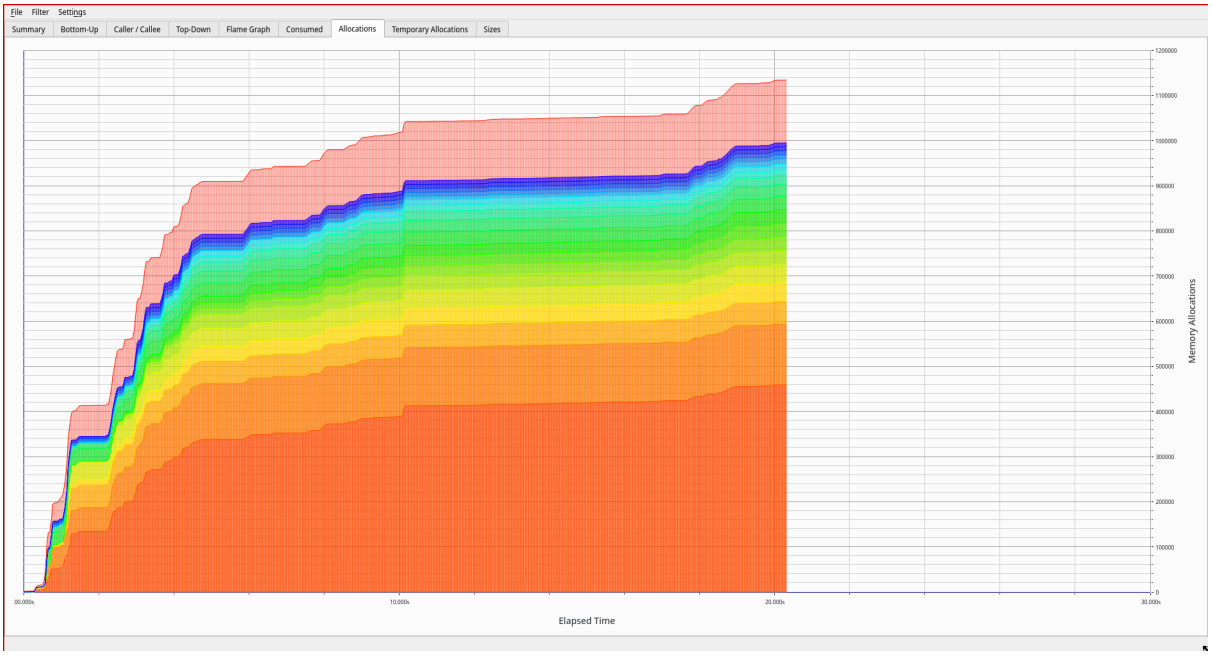
И вот, что вышло.

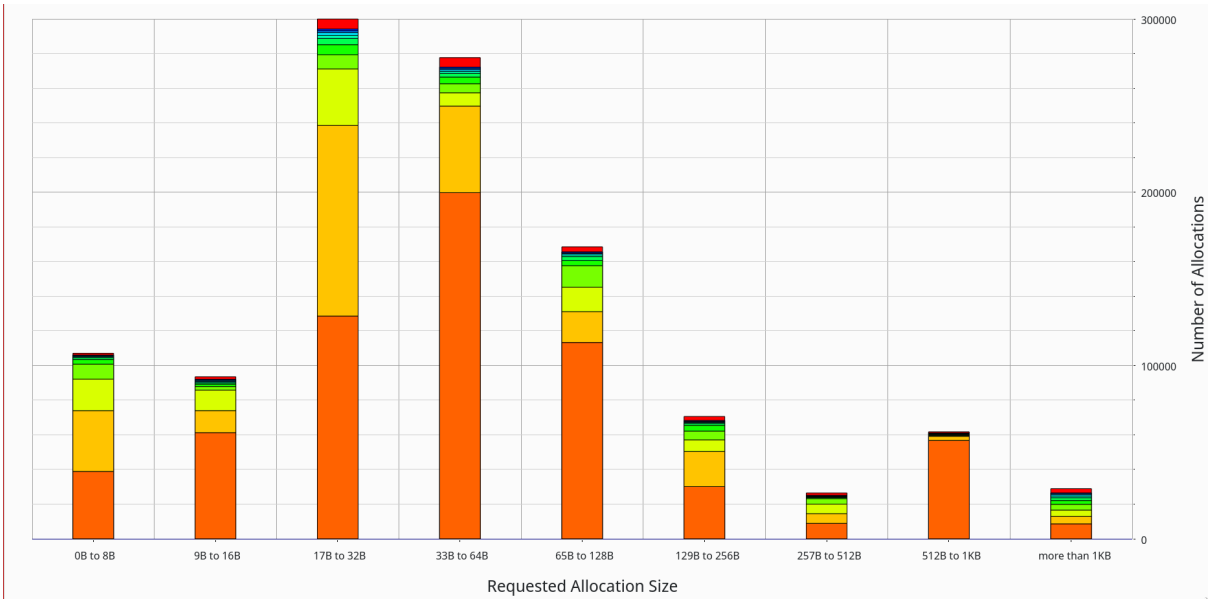


Попытка 2: Анализ работы telegram-desktop в runtime с помощью утилиты heaptrack.

Heaptrack - очень удобная утилита для профилирования приложений.

```
[mertz@arch 6_lab]$ heaptrack telegram-desktop
heaptrack output will be written to "/home/mertz/dev/uni/os_class/6_lab/heaptrack.telegram-desktop.5608.zst"
/usr/lib/heaptrack/libheaptrack_preload.so
```





Вывод:

Как мы видим, приложение telegram-desktop чаще всего выделяет участки памяти размером от 17 до 32 байт. Поискав в исходниках, я не нашел никаких

маллоков, кроме как связанных с FFmPEG (av_malloc())

Что в целом не удивительно, так как проект написан на плюсах, и в открытом виде маллоки используются редко.