

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

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

Факультет безопасности информационных технологий

Дисциплина:


«Операционные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

«Функции malloc/free»

Выполнила:

Студентка группы N32511

Синюта А.А. 

Проверил:

Ханов А.Р. _____

Санкт-Петербург

2023г.

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

Код на C, выполняющий выполняет цикл с выделением памяти разных размеров разными способами и сохранением результатов времени выделения в файл "allocation_times.txt".

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
```

```
#define FILENAME "allocation_times.txt"
#define NUM_TESTS 10
```

```
long long get_current_time() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000000 + tv.tv_usec;
}
```

```
long long test_malloc(size_t size) {
    void* ptr;
    long long start_time, end_time, total_time = 0;

    for (int i = 0; i < NUM_TESTS; i++) {
        start_time = get_current_time();
        ptr = malloc(size);
        free(ptr);
        end_time = get_current_time();
        total_time += end_time - start_time;
    }

    return total_time / NUM_TESTS;
}
```

```
long long test_calloc(size_t size) {
    void* ptr;
    long long start_time, end_time, total_time = 0;

    for (int i = 0; i < NUM_TESTS; i++) {
```

```

        start_time = get_current_time();
        ptr = calloc(1, size);
        free(ptr);
        end_time = get_current_time();
        total_time += end_time - start_time;
    }

    return total_time / NUM_TESTS;
}

long long test_realloc(size_t size) {
    void* ptr;
    long long start_time, end_time, total_time = 0;

    for (int i = 0; i < NUM_TESTS; i++) {
        ptr = malloc(size / 2);
        start_time = get_current_time();
        ptr = realloc(ptr, size);
        free(ptr);
        end_time = get_current_time();
        total_time += end_time - start_time;
    }

    return total_time / NUM_TESTS;
}

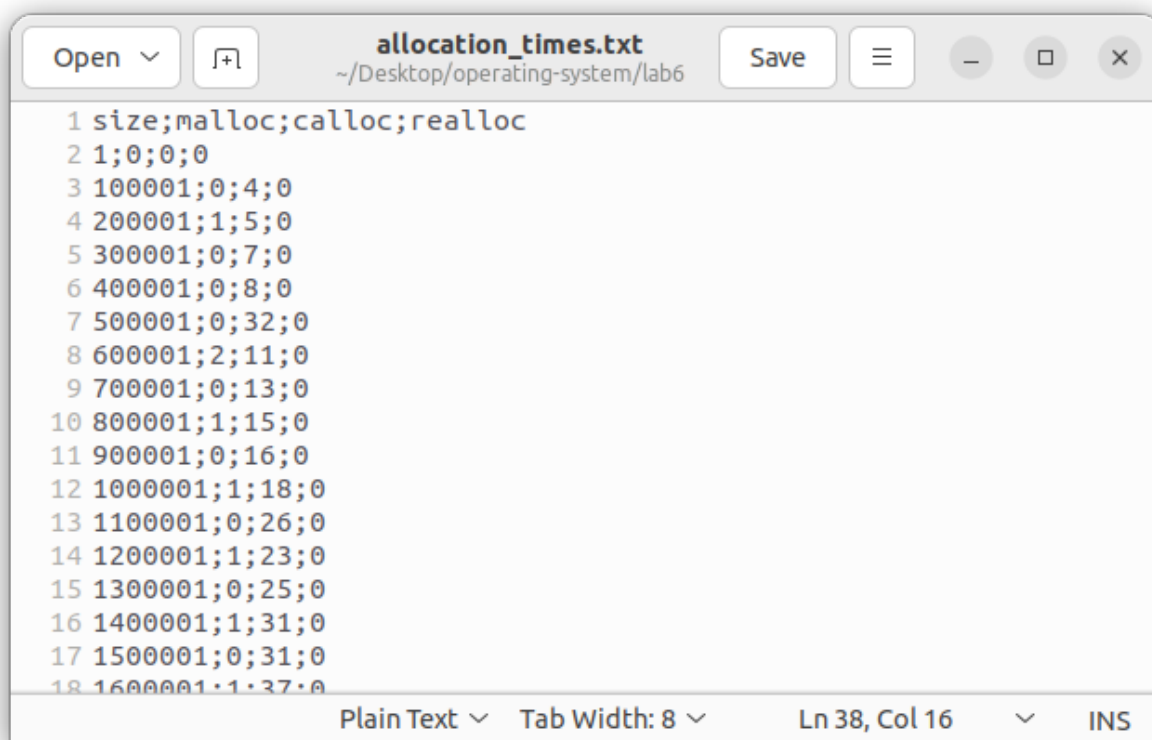
int main() {
    FILE* file = fopen(FILENAME, "w");
    if (file == NULL) {
        printf("Failed to open file for writing.\n");
        return 1;
    }
    fprintf(file, "size;malloc;calloc;realloc\n");

    size_t min_size = 1; // Minimum size of memory to allocate
    size_t max_size = 100000000; // Maximum size of memory to allocate
    size_t step_size = 100000; // Size increment

    for (size_t size = min_size; size <= max_size; size += step_size) {
        long long m = test_malloc(size);
        long long c = test_calloc(size);
        long long r = test_realloc(size);
        fprintf(file, "%zu;%lld;%lld;%lld\n", size, m, c, r);
    }
    fclose(file);
    printf("Results saved to file %s\n", FILENAME);
    return 0;
}

```

В результате получаем такой файл с 1000 записями:

A screenshot of a text editor window titled 'allocation_times.txt' with a path of '~/.Desktop/operating-system/lab6'. The window contains a list of 18 lines of data, each representing a memory allocation record. The data is formatted as 'size;malloc;calloc;realloc'. The 'size' values range from 1 to 1600001 in increments of 100000. The 'malloc', 'calloc', and 'realloc' values are integers representing time in seconds. The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 38, Col 16', and 'INS' mode.

```
1 size;malloc;calloc;realloc
2 1;0;0;0
3 100001;0;4;0
4 200001;1;5;0
5 300001;0;7;0
6 400001;0;8;0
7 500001;0;32;0
8 600001;2;11;0
9 700001;0;13;0
10 800001;1;15;0
11 900001;0;16;0
12 1000001;1;18;0
13 1100001;0;26;0
14 1200001;1;23;0
15 1300001;0;25;0
16 1400001;1;31;0
17 1500001;0;31;0
18 1600001;1;37;0
```

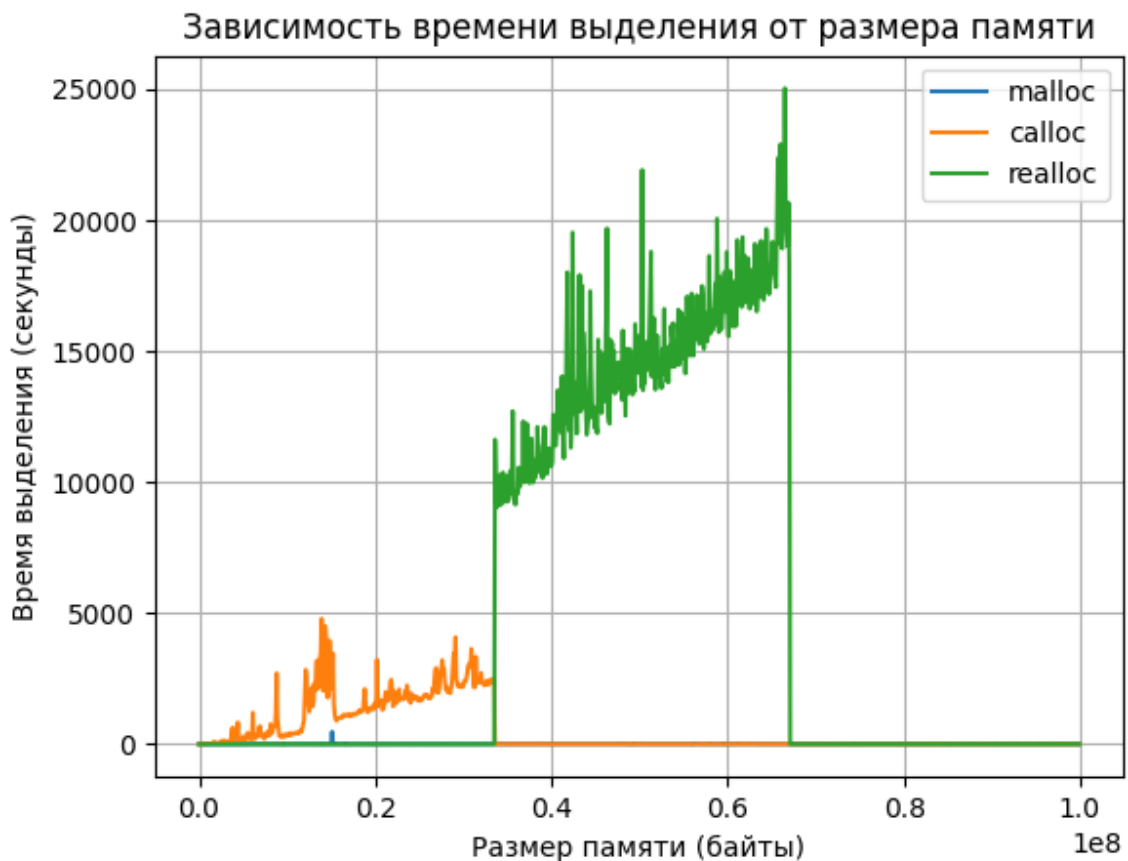
На основе этих данных построим график с помощью matplotlib Python3.

```
import matplotlib.pyplot as plt

filename = 'allocation_times.txt'
sizes = []
mallocs = []
callocs = []
reallocs = []
with open(filename, 'r') as file:
    label0, label1, label2, label3 = file.readline().strip().split(';')
    for line in file:
        size, malloc, calloc, realloc = line.strip().split(';')
        sizes.append(int(size))
        mallocs.append(int(malloc))
        callocs.append(int(calloc))
        reallocs.append(int(realloc))
plt.plot(sizes, mallocs, label=label1)
plt.plot(sizes, callocs, label=label2)
plt.plot(sizes, reallocs, label=label3)
plt.xlabel('Размер памяти (байты)')
plt.ylabel('Время выделения (секунды)')
plt.title('Зависимость времени выделения от размера памяти')
plt.grid(True)
plt.legend()
plt.savefig('chart.png')
plt.show()
```

```
vboxuser@ubuntu64: ~/Desktop/operating-system/lab6
vboxuser@ubuntu64:~/Desktop/operating-system/lab6$ make
gcc -o lab6 lab6.c
vboxuser@ubuntu64:~/Desktop/operating-system/lab6$ ./lab6
Results saved to file allocation_times.txt
vboxuser@ubuntu64:~/Desktop/operating-system/lab6$ python3 chart.py
vboxuser@ubuntu64:~/Desktop/operating-system/lab6$
```

Получаем такой график:



Вывод

Исходя из полученного графика, можно сделать следующие выводы относительно производительности трех функций выделения памяти: malloc(), calloc() и realloc().

При анализе графика видно, что на начальных этапах размера выделяемой памяти функция calloc() демонстрировала наихудшие показатели производительности, требуя больше времени на выделение памяти по сравнению с другими функциями. Однако, по мере увеличения размера выделяемой памяти, наихудшей функцией стала realloc(). Это может быть связано с необходимостью копирования данных при изменении размера памяти. Однако, при дальнейшем увеличении размера запроса памяти, время выполнения всех трех функций сравнялось.

Сравнивая все три функции выделения памяти, можно сделать вывод, что `malloc()` показал лучшие результаты производительности во всех рассмотренных случаях. Она демонстрировала более низкое время выполнения по сравнению с `calloc()` и `realloc()`. Таким образом, для данного набора тестов и размеров выделяемой памяти, функция `malloc()` является наиболее эффективным выбором.