ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина: «Операционные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Выполнил:
Студент группы N3249
Чан Нгок Хуан

Проверил:
Савков Сергей Витальевич

Санкт-Петербург
2022г.

ЗАДАНИЕ

Лаб 6.
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

sudo perf record -ag -e syscalls:sys_enter_mmap -- sleep 30
sudo perf script --header

malloc, free, calloc, realloc — выделить и освободить динамическую память

ОПИСАНИЕ

The **malloc()** function allocates size bytes and returns a pointer to the allocated memory.The memory is not initialized. If size is 0, then **malloc()** returns either NULL, or a unique pointer value that can later be successfully passed to **free().**

The **free()** function frees the memory space pointed to by ptr, which must have been returned by a previous call to **malloc(), calloc(), or realloc().** Otherwise, or if free(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed.

The **calloc()** function allocates memory for an array of nmemb elements of size bytes each and returns a pointer to the allocated memory. The memory is set to zero. If nmemb or size is 0, then **calloc()** returns either NULL, or a unique pointer value that can later be successfully passed to free().

The **realloc()** function changes the size of the memory block pointed to by ptr to size bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will not be initialized. If ptr is NULL, then the call is equivalent to malloc(size), for all values of size; if size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr). Unless ptr is NULL, it must have been returned by an earlier call to **malloc(), calloc()** or **realloc().** If the area pointed to was moved, a free(ptr) is done.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

The **malloc()** and **calloc()** functions return a pointer to the allocated memory, which is suitably aligned for any built-in type. On error, these functions return NULL. NULL may also be returned by a successful call to **malloc()** with a size of zero, or by a successful call to **calloc()** with nmemb or size equal to zero.

The **free()** function returns no value.

The **realloc()** function returns a pointer to the newly allocated memory, which is suitably aligned for any built-in type and may be different from ptr, or NULL if the request fails. If size was equal to 0, either NULL or a pointer suitable to be passed to **free()** is returned. If **realloc()** fails, the original block is left untouched; it is not freed or moved.

I. Функции malloc() и free()

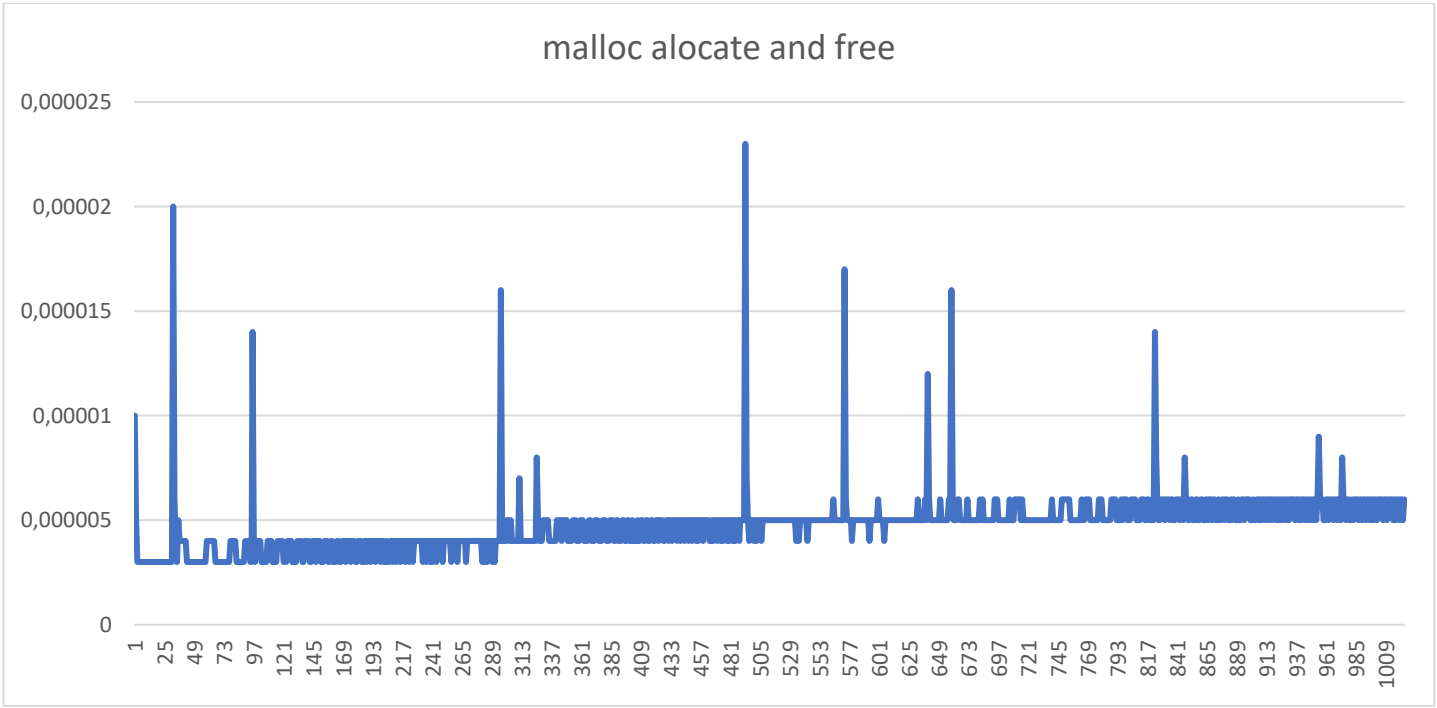
1. Time allocate and free

Код программы

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

int main(){
    FILE *f;
    f = fopen("mallocfree.txt", "w");
    int size = 1024*1024;
    for (int i = 1; i <= 1024; i++){
        int *a;
        clock_t begin = clock();
        a = (int*)malloc(i*size);
        free(a);
        clock_t end = clock();
        double time = (double)(end - begin)/CLOCKS_PER_SEC;
        fprintf(f, "%d %f\n", i, time);
    }
    fclose(f);
}
```

График



2. Time allocate and time free

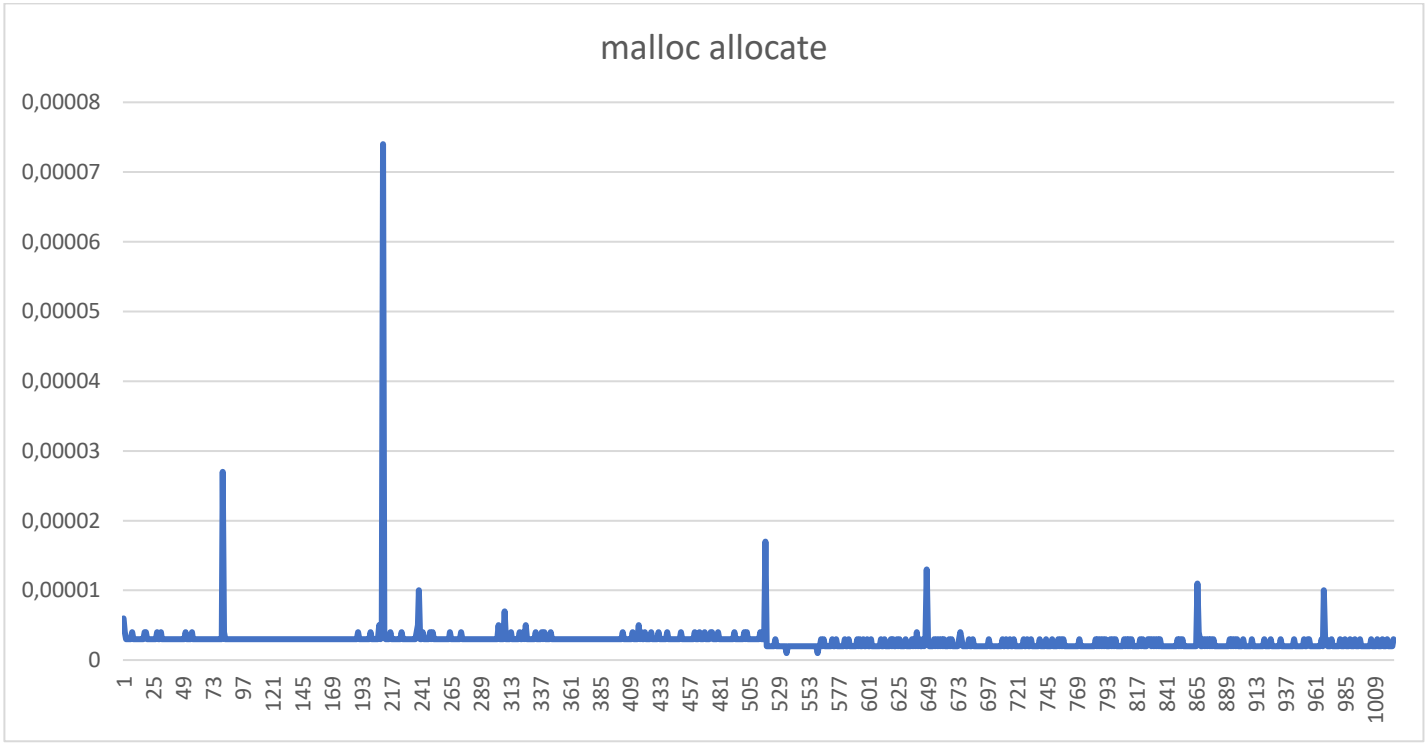
Код программы

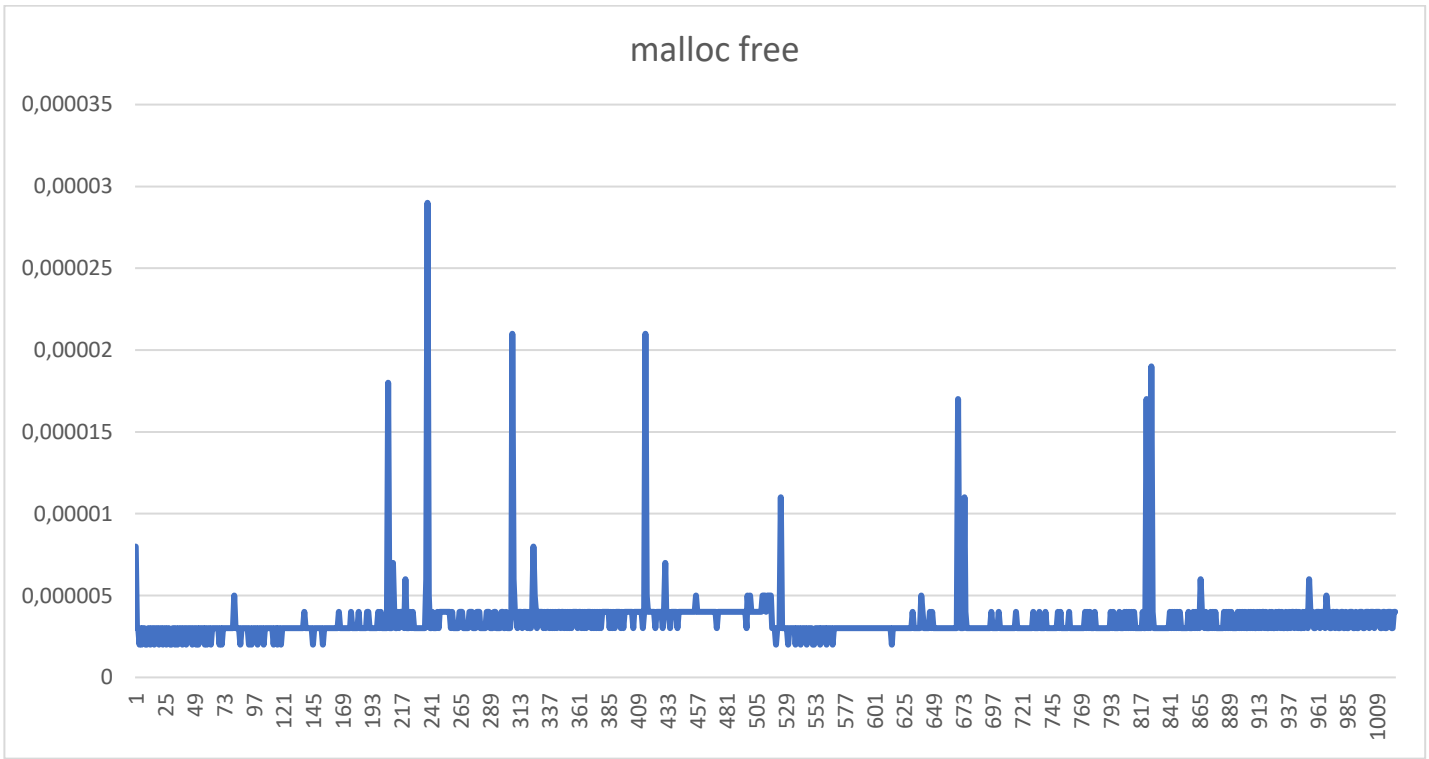
```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(){
    FILE *f;
    f = fopen("malloc.txt", "w");
    FILE *fd;
    fd = fopen("free1.txt", "w");
    int size = 1024*1024;
    for (int i = 1; i <= 1024; i++){
        int *a;
        clock_t begin1 = clock();
        a = (int*)malloc(i*size);
        clock_t end1 = clock();
        double time1 = (double)(end1 - begin1)/CLOCKS_PER_SEC;
        fprintf(f, "%d %f\n", i, time1);

        clock_t begin2 = clock();
        free(a);
        clock_t end2 = clock();
        double time2 = (double)(end2 - begin2)/CLOCKS_PER_SEC;
        fprintf(fd, "%d %f\n", i, time2);
    }
    fclose(f);
}
```

График


malloc allocate
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 25 49 73 97 121 145 169 193 217 241 265 289 313 337 361 385 409 433 457 481 505 529 553 577 601 625 649 673 697 721 745 769 793 817 841 865 889 913 937 961 985 1009


malloc free
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
1 25 49 73 97 121 145 169 193 217 241 265 289 313 337 361 385 409 433 457 481 505 529 553 577 601 625 649 673 697 721 745 769 793 817 841 865 889 913 937 961 985 1009

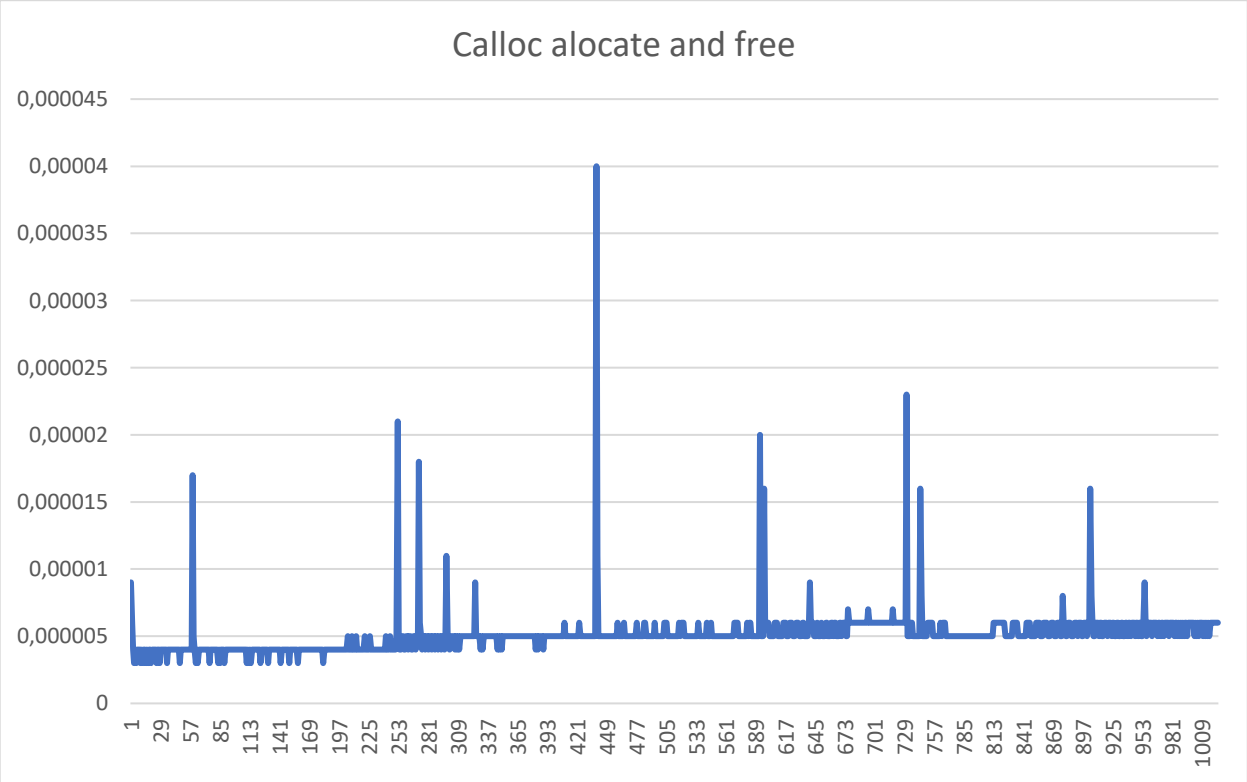
II. Функции calloc() и free()

Код программы

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

int main(){
    FILE *f;
    f = fopen("callocfree.txt", "w");
    int size = 1024*1024;
    for (int i = 1; i <= 1024; i++){
        int *a;
        clock_t begin = clock();
        a = (int*)calloc(i, size);
        free(a);
        clock_t end = clock();
        double time = (double)(end - begin)/CLOCKS_PER_SEC;
        fprintf(f, "%d %f\n", i, time);
    }
    fclose(f);
}
```

График



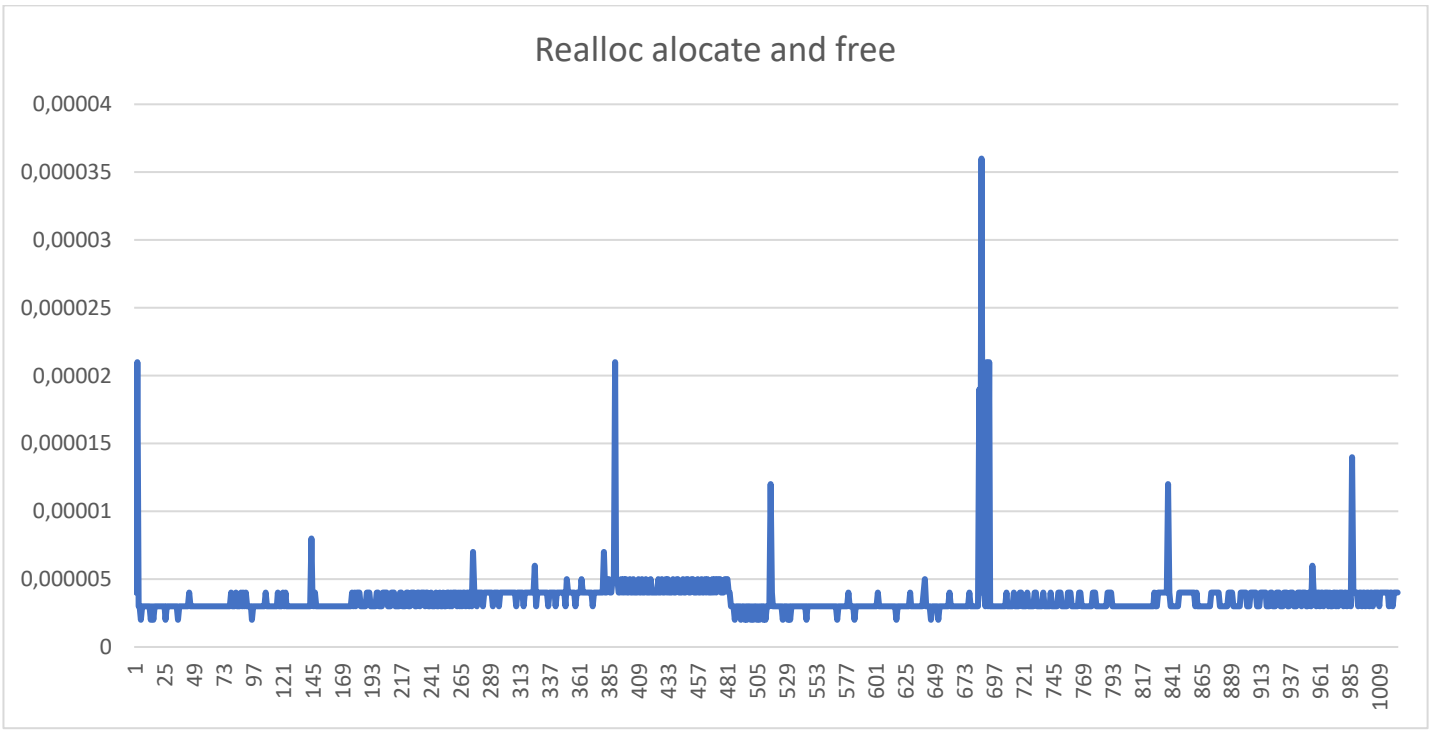
III. Функция realloc() и free()

Код программы

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

int main(){
    FILE *f;
    f = fopen("reallocfree.txt", "w");
    int size = 1024*1024;
    for (int i = 1; i <= 1024; i++){
        int *a;
        a = (int*)malloc(i*size);
        clock_t begin = clock();
        a = (int*)realloc(a,i*size);
        free(a);
        clock_t end = clock();
        double time = (double)(end - begin)/CLOCKS_PER_SEC;
        fprintf(f, "%d %f\n", i, time);
    }
    fclose(f);
}
```

График



	Malloc and free (us)	Calloc and free (us)	Realloc and free (us)
среднее	4.77	5.14	3.59

IV. Вывод

- Мы видим, что malloc, calloc и realloc отличаются по времени:
  Calloc > malloc > realloc
- Освобождение памяти занимает больше времени при увеличении размера выделяемого/очищаемого блока памяти.