TD3 : Formatted I/O Library

Master M1 MOSIG, Grenoble Universities

Lenka Kuníková Lina Marsso

26/10/2014

1 Introduction

In this project we have to implement a I/O library. And evaluate their performances in different applications.

Firstly in this report we give answers of questions in the subject. Then we present you our implementation. Then we speak about the safety checks implementation. Finnaly, we analyse the performance and conclude this project.

2 Subject questions

2.1 Question 1

3 Implementation

1. We initialize the memory with the function memory_init
2. We alloc the new block with mem_alloc like that :
   - We find the address of new block allocated with one between the strategy 3 (First fit, Worse fit, best fit). We can choose. Basically, it's first fit.
   - We allocate the full block and update list of freeblock
3. We can free a block with free_block :
   - We find an adjacent block to the one to be freed and try to merge then
   - If no adjacent block : create a new block and insert it in the list

4 Safety checks

5 Evaluation

For evaluate our library, we use one test he copie one file in an other file buffered for our library and we write the same test but using the standard open, read and write.

We evaluate in the begining the time for copy two file with our library, and with the standard open, read and write. We have also one ratio :

$$\left(\frac{My\_stdio\_time\_execution}{Standar\_Read\_Write\_time\_execution}\right) \tag{1}$$

We calculate the time of the execution with a script time.sh. We can see

Library	Time (microsecondes)	Size of file
libmy_sdio	4	8.0K
Standar	36	8.0K

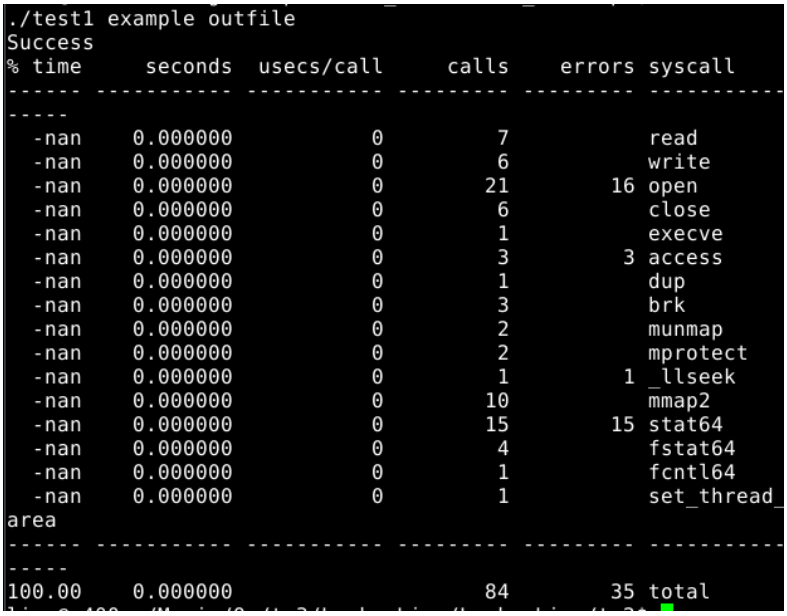

Figure 1: Number of system call of our library

5.1 Table with fragmentation level of differents programs

program	worst_fit	best_fit	first_fit
ls	1.83	1.83	2.48
wc	0.96	0.96	0.96
ps	1.37	1.36	1.37
hostname	2.42	2.42	3.23

6 Conclusion

The goal of this project was to get familiar with the concept of memory managment and I think we succeeded. We managed to implement a simple version of memory allocator which allows users to allocate and free memory.

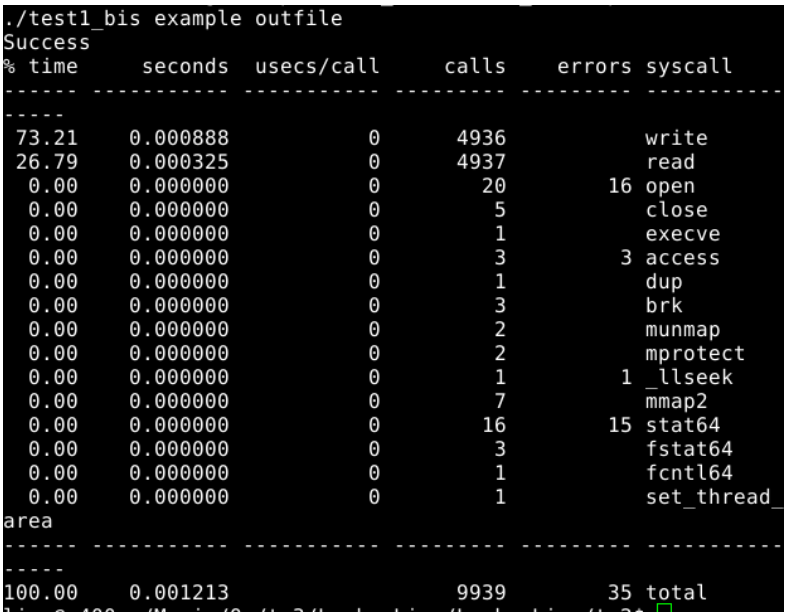

Figure 2: Number of system call with read and write

Figure 3: Memory requested and memory used with best fit strategy

During the implementation we had to deal with several problems. For example we needed to find a way on how to determine positions of free blocks when we do not have linked lists of them, or think about the algorithm that helps us merge contiguous blocks of free space. Moreover, we implemented the three most common algorithms for finding an appropriate free block (first-fit, best-fit and worse-fit). We are now able to describe pros and cons for each of them.

In the second part, we added some other features to our memory allocator. All of them related to security checks. Now we know that implementing functions for free and alloc memory is not enough and we also need to deal with situations where users (on purpose or by chance) do not act the way we expect. The control we are making is not perfect but it helps us to deal with many cases that may happen.

Finnaly, we can see an application of our allocator. We can analyse it's performance with many apllications like ls, hostname... It was a very interesting project.