SCALE FOR PROJECT MALLOC (/PROJECTS/42CURSUS-MALLOC)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere-v2-bg.1337.ma:vogsphere/intra-uuid-8c14a826-a94f-46f3-
```

Introduction

Please respect the following rules:

- Remain polite, courteous, respectful and constructive throughout the correction process. The well-being of the community depends on it.

- Identify with the person (or the group) graded the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.

- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.

Guidelines

You MUST run the requested tests.

Warning: This project is quite complex, the result and it's implementation are subjective. You have to keep in mind the aim of this project:

"This project is about implementing a dynamic memory allocation mechanism."

Attachments

- test4.c (https://cdn.intra.42.fr/document/document/22026/test4.c)
- test3.c (https://cdn.intra.42.fr/document/document/22027/test3.c)
- run_linux.sh (https://cdn.intra.42.fr/document/document/22028/run_linux.sh)
- test2.c (https://cdn.intra.42.fr/document/document/22029/test2.c)
- test0.c (https://cdn.intra.42.fr/document/document/22030/test0.c)
- test5.c (https://cdn.intra.42.fr/document/document/22031/test5.c)
- test1.c (https://cdn.intra.42.fr/document/document/22032/test1.c)
- run_mac.sh (https://cdn.intra.42.fr/document/document/22033/run_mac.sh)
- subject.pdf (https://cdn.intra.42.fr/pdf/pdf/109735/en.subject.pdf)

Preliminaries

Preliminary tests

First check the following elements :

- There is something in the git repository
- A Makefile is present and has all the requested rules
- No cheating (unauthorized functions...)
- 2 globals are authorised : one to manage the allocations, and one to manage the thread-safe

If an element of this list isn't respected, the grading ends.
Use the appropriate flag. You're allowed to debate some more
about the project, but the grading will not be applied.

Yes No

Library compilation

First we will check that the compilation of the library does
generate the requested files by modifying HOSTTYPE:

```
$> export HOSTTYPE=Testing
$> make re
********
$> ln -s libft_malloc_Testing.so libft_malloc.so
$> ls -l libft_malloc.so
lrwxrwxrwx 1 ** ** ** ** ** ** libft_malloc.so -> libft_malloc_Testing.so
$>
```

The Makefile does use HOSTTYPE to define the name of the library
(libft_malloc_$HOSSTYPE.so) and does create a symbolic link
libft_malloc.so pointing towards libft_malloc_$HOSSTYPE.so ?

If that's not the case, the defense stops.

Yes No

Functions export

Check with nm that the library does export the functions malloc,
free, realloc and show_alloc_mem.

```
$> nm -g libft_malloc.so
0000000000000000 T _free
0000000000000000 T _malloc
0000000000000000 T _realloc
0000000000000000 T_show_alloc_mem
                 U _mmap
                 U _munmap
                 U _getpagesize
                 U _write
$>
```

The functions exported by the library are marked with a T, the
used one with a U (adresses have been replaced by 0, they change
from one library to the next, same as the order of the lines).

If the functions are not exported, defense stops.

Yes No

Feature's testing

Please find the attached (MacOS X or Linux) script that will only modifiy the environment variables while you run a test program.

Malloc test

We are first going to make a first test program that does not
use malloc, so that we have a base to compare to.

Use test0.c file attached in the scale
WARNING: If you are using a linux vm, make sure that you are using the time binary that you can get with apt (sudo apt install time), else you won't have access to the -v option.

MAC:

```
$> gcc -o test0 test0.c && /usr/bin/time -l ./test0
```

LINUX:

```
$> gcc -o test0 test0.c && /usr/bin/time -v ./test0
```

We will then add a malloc and write in each allocation to make sure that the memory page is allocated in physical memory by MMU. The system will only really allocate the memory of a page if you write in it, so even if we do a bigger mmap than the malloc request it won't modify the "page reclaims".

For Linux => Major, Minor

For Mac OS X => page reclaims, page faults

Using the test1.c file given as attachment, run the following tests
MAC:

```
$> gcc -o test1 test1.c && /usr/bin/time -l ./test1
```

LINUX:

```
$> gcc -o test1 test1.c && /usr/bin/time -v ./test1
```

Our test1 program requested 1024 times 1024 bytes, so 1Mbyte. We can therefore check by doing the difference with the test0 program:

- either between the "maximum resident set size" lines, we obtain a little more than 1Mbyte
- or between the page reclaims lines that we will multiply by the value of getpagesize(3)

Let's test now both programs with our library:

MAC:

```
$>chmod 777 run_mac.sh


$>./run_mac.sh /usr/bin/time -l ./test0
  ********
  202  page reclaims
  0  page faults
  ********


$>./run_mac.sh /usr/bin/time -l ./test1
  ********
  525  page reclaims
  0  page faults
  ********
$>
```

LINUX:

```
$>chmod 777 run_linux.sh


$>./run_linux.sh /usr/bin/time -v ./test0
  ********
  Major (requiring I/O) page faults: 0
  Minor (reclaiming a frame) page faults: 64
  ********
```

```
$>./run_linux.sh /usr/bin/time -v ./test1
  ********
  Major (requiring I/O) page faults: 0
  Minor (reclaiming a frame) page faults: 323
  ********
```

Count the number of pages used in addition to the real malloc and adjust the score accordingly:

- less pages than the real malloc, allocated memory is insufficiant: 0
- 181 pages and over, malloc works but the overhead is very too big: 1
- between 91 pages and 180 pages, malloc works but the overhead is too big: 2
- between 51 pages and 90 pages, malloc works but the overhead is very big: 3
- between 21 pages and 50 pages than real malloc, malloc works but the overhead is big: 4
- between 0 pages and 20 pages than real malloc, malloc works and the overhead is fine: 5

Rate it from 0 (failed) through 5 (excellent)

Pre-allocated zones

Check inside the source code that the pre-allocated zones for the different malloc sizes allow to store at least 100 times the maximum size for this type of zone. Check also that the size of the zones is a multiple of getpagesize().

If one of these points is missing, click NO.

Yes No

Tests of free

We will simply add a free to our test program:

```
$> cat test2.c
```

We will compare the number of "page reclaims / page faults" to those in test0 and test1. If there are as many or more "page reclaims" than test1, the free doesn't work.

MAC:

```
$> gcc -o test2 test2.c && ./run_mac.sh /usr/bin/time -l ./test2
```

LINUX:

```
$> gcc -o test2 test2.c && ./run_linux.sh /usr/bin/time -v ./test2
```

Does the free function? (less "page reclaims / page faults" than test1)

Yes No

Quality of the free function

Run test0 and test2. Test2 should not have more than 10 page reclaims compared to test0.

Yes No

Realloc test

Using test3.c file given as attachment, test the following:

MAC:

```
$> gcc -o test3 test3.c -L. -lft_malloc && ./run_mac.sh ./test3
Hello world!
Hello world!
$>
```

LINUX:

```
$> gcc -o test3 test3.c -L. -lft_malloc && ./run_linux.sh ./test3
Hello world!
Hello world!
$>
```

The test must print out "Hello world!" two times.

Does it work as expected?

Yes No

Show_alloc_mem test

Using test4.c file given as attachment, test the following:

MAC:

```
$> gcc -o test4 test4.c -L. -lft_malloc && ./run_mac.sh ./test4
$>
```

LINUX:

```
$> gcc -o test4 test4.c -L. -lft_malloc && ./run_linux.sh ./test4
$>
```

Does the display corresponds the subject and the TINY/SMALL/LARGE allocation of the project?

Yes No

Alignement test

Using test5.c file given as attachment, test the following:

MAC:

```
$> gcc -o test5 test5.c -L. -lft_malloc && ./run_mac.sh ./test5
$>
```

LINUX:

```
$> gcc -o test5 test5.c -L. -lft_malloc && ./run_linux.sh ./test5
$>
```

You have no alignment errors.

Yes No

Bonus

Competitive access

The project manages the competitive access of the threads with the support of the pthread library and with mutexes.

Count the applicable cases:

- a mutex prevents multiple threads to simulteanously enter inside the malloc function
- a mutex prevents multiple threads to simulteanously enter inside the free function
- a mutex prevents multiple threads to simulteanously enter inside the realloc function
- a mutex prevents multiple threads to simulteanously enter inside the show_alloc_mem function

Rate it from 0 (failed) through 5 (excellent)

Additional bonuses

If there are more bonuses, grade them here. Bonuses must be 100%
functional and a minimum useful. (up to the grader)

Bonus example:

- During a free, the projet "defragments" the free memory while regrouping the available simultaneous blocks.
- Malloc has debugging environnement variables
- A function allows to make an hexadecimal dump of the allocated zones
- A fonction allows to display an history of the memory allocations done.
- If there are other bonuses, add them up here. Bonuses must be 100% functional and a minimum useful (at the corrector's discretion)

Rate it from 0 (failed) through 5 (excellent)

Ratings

Don’t forget to check the flag corresponding to the defense

✔ Ok ★ Outstanding project

Empty work | Incomplete work | No author file | Invalid compilation | Norme | Cheat

Crash | Forbidden function | Can’t support / explain code

Conclusion

Leave a comment on this evaluation

Finish evaluation