Project rather UNIX

Malloc

42 staff staff@42.fr

Summary: This project involves the implementation of a dynamic allocation mechanism for

memory.

Page 2

Contents

1

Page 3

Chapter I

Preamble

Here is what Wikipedia has to say about sensory memory

Sensory register, or "sensory memory", is the structure that keeps
A very short period of time (a few milliseconds) the sensory information, that is to say,
Sounds, images, smells, etc., which reaches us most of the time unconsciously.

Sensory recording is what brings us into contact with the outside world. In
At every moment, we are exposed to a multitude of stimuli.
Most of us do not realize. However, even if it is necessary to
Our attention to these stimuli in order to grasp their meaning, this does not mean that they are
Not registered by our organization. On the contrary, all the information that
Our senses are recorded.

The olfactory memory is counted as the best. Indeed, studies [ref.
Wishes] led to the observation that an odor, although one was not aware of this
Odor, remains recorded for all or most of his life.

An author has examined this idea of olfactory memory, Marcel Proust. Indeed,
This one remarkably evoked the perfume of the little madeleines soaked in the
Tea, analyzed the memories associated with it and spotted the mechanisms of memory
Olfactory. "And all of a sudden the memory appeared to me. This taste was that of the
A piece of Madeleine that Sunday morning at Combray, my aunt Leonie offered me. The
The sight of little Madeleine had reminded me of nothing before I had tasted it. But,
When from an ancient past nothing remains, after the death of beings, after the destruction
Things, more fragile but more perennial, more immaterial, more persistent, more
The smell and flavor remain for a long time, like souls, to wear without
The immense edifice of remembrance. "

Chapter II

Subject

This miniproj consists of writing a library of management of the dynamic allocation of memory. In order to be able to use it by programs already existing without the
You must rewrite the malloc (3), free (3), and realloc (3)
Of libc

Your functions will be prototyped like those of the system:

```
#include <stdlib.h>

Void        Free (void * ptr);
Void        * Malloc (size_t size);
Void        * Realloc (void * ptr, size_t size);
```

- The malloc () function allocates "size" bytes of memory and returns a pointer to
  The allocated memory.
- The realloc () function tries to change the size of the alloction pointed to by "ptr"
  To "size" bytes, and finds "ptr". If there is not enough space at the location
  Memory pointed to by "ptr", realloc () creates a new allocation, copies as many
  Of the old allocation as possible within the limit of the size of the
  New allocation, releases the old allocation and returns a pointer to that allocation.
  New allocation.
- The free () function frees the allocation of the memory pointed to by "ptr". If "ptr" is equal to
  NULL, free () does nothing.
- If an error occurs, the malloc () and realloc () functions return a NULL pointer;
- You must use the syscall mmap (2) and munmap (2) to claim and return
  Memory to the system.
- You must manage your own memory allocations for internal operation
  Of your project without using the malloc function of the libc.
- You must in a performance concern limit the number of calls to mmap (),
  But also to munmap (). You will therefore need to "pre-allocate" memory areas for
  Store there your "small" and "medium" malloc.
- The size of these fields must imperatively be a multiple of getpagesize ().

Page 5

- Each zone must contain at least 100 allocations.
  - Mallocs "TINY", from 1 to n bytes, will be stored in zones of N bytes
  - The "SMALL" mallocs, from (n + 1) to m bytes, will be stored in M bytes
  - The "LARGE" mallocs, of (m + 1) bytes and more, will be stored outside the zone; To say simply with a mmap (), they will be in a few a zone to them alone.
- It is up to you to define the size of n, m, N and M in order to find a good compromise Between speed (system call saving) and memory savings.

You must also write a function to display the status of the Allocated funds. It should be prototyped as follows:

```
Void        Show_alloc_mem ();
```

The display will be formatted by increasing address as in the following example:

```
TINY: 0xA0000
0xA0020 - 0xA004A: 42 bytes
0xA006A - 0xA00BE: 84 bytes
SMALL: 0xAD000
0xAD020 - 0xADEAD: 3725 bytes
LARGE: 0xB0000
0xB0020 - 0xBBEEF: 48847 bytes
Total: 52698 bytes
```

4

Page 6

Chapter III

Topic - Bonus Game

Bonuses will be counted only if your compulsory part
Is PERFECT. Per PERFECT, one understands of course that it
Is fully realized, and it is not possible to
Its faulty behavior, even in the event of such a vicious error
Misuse, etc ... Concretely, this
Means that if your mandatory part does not pass, your bonuses
Will be completely IGNORED.

Bonus ideas:

- Manage malloc debug environment variables. You can copy
  Of the malloc of the system or invent yours.
- Create a function show_alloc_mem_ex () which allows to display more details,
  For example a history of allocations, or a hex dump of the allocated zones.
- "Defragment" the freed memory.

• Manage the use of your malloc in a multi-threaded program
"Thread safe", and this with the lib pthread).

Page 7

Chapter IV

Instructions

• This project will only be corrected by humans. You are therefore free to organize and
Naming your files as you wish, while respecting the constraints
Listed here.

• The library must be named libft_malloc_ $ HOSTTYPE.so

• Make a Makefile. It will compile the library, and contain the rules

Usual. It should only recompile the library if necessary.

- Your Makefile will have to check the existence of the environment variable $ HOSTTYPE. If it is empty or nonexistent, assign it the following value: 'Uname -m' _ 'uname -s'

```
Ifeq ($ (HOSTTYPE),)
    HOSTTYPE: = $ (shell uname -m) _ $ (shell uname -s)
Endif
```

- Your Makefile will create a symbolic link libft_malloc.so pointing to Libft_malloc_ $ HOSTTYPE.so for example: Libft_malloc.so -> libft_malloc_intel-mac.so

- If you are smart and use your libft library for your malloc, You must copy the sources and the associated Makefile into a named folder Libft that should be at the root of your render repository. Your Makefile will Compile the library, by calling its Makefile, then compile your project.

- You can have a global variable to manage your allocations and one for the Thread-safe.

- Your project must comply with the Standard.

- You must manage the errors reasonably. In no case your program Must not leave unexpectedly (Segmentation fault, etc ...).

- At the root of your repository, you must return an authoring file containing Your login followed by a '\ n':

```
$> Cat -e author
Xlogin $
$>
```


Page 8

- As part of your mandatory part, you have the right to use the functions The following:
  - mmap (2)
  - munmap (2)
  - getpagesize (3)
  - getrlimit (2)
  - the functions allowed as part of your libft (write (2) for example ;-))

    - functions of the libpthread
- You have permission to use other features as part of your bonus, Provided that their use is duly justified in your correction. The Smart.

- You can ask your questions on the forum on the Intranet.

Good luck to all !

Chapter V

Let's laugh a little

In a past so far past, the malloc project was done with brk (2) and sbrk (2) instead And place of mmap (2) and munmap (2). Here is what the man of brk (2) and sbrk (2) has to say on The time of the dinosaurs:

```
$> Man 2 brk
...
DESCRIPTION
The brk and sbrk functions are historical curiosities
    Virtual memory management.
...
4th Berkeley Distribution        December 11, 1993    4th Berkeley Distribution
$>
```

Of course, this description is the result of the concise implementation of Brk (2) on Mac Os X:

```
Void * brk (void * x)
{
  Errno = ENOMEM;
  Return ((void *) - 1);
}
```

8