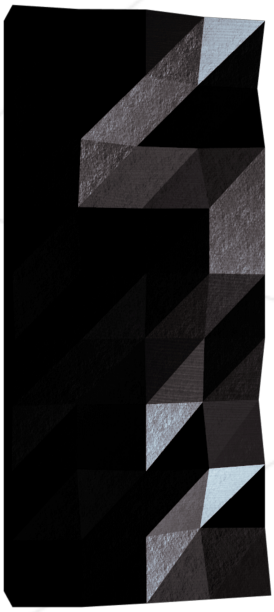

Projet Environnement UNIX II

malloc

42 staff staff@42.fr

Résumé: Ce projet consiste à implémenter un mécanisme d'allocation dynamique de la mémoire.

Table des matières

Chapitre I

Préambule

Voici ce que Wikipédia a à dire au sujet de la mémoire sensorielle

On appelle registre sensoriel, ou « mémoire sensorielle » la structure qui garde pendant un très court laps de temps (quelques millisecondes) l'information sensorielle, c'est-à-dire, les sons, les images, les odeurs, etc, qui nous atteint la plupart du temps inconsciemment.

L'enregistrement sensoriel est ce qui nous met en contact avec le monde extérieur. En effet, à chaque instant nous sommes exposés à une multitude de stimuli, dont pour la plupart nous ne prenons pas conscience. Cependant, même s'il est nécessaire de porter notre attention sur ces stimuli pour en saisir le sens, cela ne veut pas dire qu'ils ne sont pas enregistrés par notre organisme. Au contraire, toutes les informations qui atteignent nos sens sont enregistrées.

On compte la mémoire olfactive comme étant la meilleure. En effet, des études [réf. souhaitée] ont mené au constat qu'une odeur, bien qu'on n'ait pas pris conscience de cette odeur, reste enregistrée pour toute ou presque toute sa vie.

Un auteur s'est penché sur cette idée de mémoire olfactive, Marcel Proust. En effet, celui-ci a remarquablement évoqué le parfum des petites madeleines trempées dans le thé, analysé les souvenirs qui s'y rattachaient et repéré les mécanismes de la mémoire olfactive. « Et tout d'un coup le souvenir m'est apparu. Ce goût, c'était celui du petit morceau de madeleine que le dimanche matin à Combray ma tante Léonie m'offrait . La vue de la petite madeleine ne m'avait rien rappelé avant que je n'y eusse goûté. Mais, quand d'un passé ancien rien ne subsiste, après la mort des êtres, après la destruction des choses, seules, plus frêles mais plus vivaces, plus immatérielles, plus persistantes, plus fidèles, l'odeur et la saveur restent encore longtemps, comme des âmes, à porter sans fléchir l'édifice immense du souvenir. »

Chapitre II

Sujet - Partie obligatoire

Ce miniproj consiste à écrire une librairie de gestion de l'allocation dynamique de la mémoire. Afin de pouvoir la faire utiliser par des programmes déjà existant sans les modifier ni les recompiler, vous devrez réécrire les fonctions malloc(3), free(3), et realloc(3) de la libc

Vos fonctions seront prototypées comme celles du système :

```
#include <stdlib.h>

void       free(void *ptr);
void       *malloc(size_t size);
void       *realloc(void *ptr, size_t size);
```

- La fonction malloc() alloue "size" octets de mémoire et retourne un pointeur sur la mémoire allouée.
- La fonction realloc() essaye de modifier la taille de l'alloction pointée par "ptr" à "size" octets, et retroune "ptr". Si il n'y a pas assez de place à l'emplacement mémoire pointé par "ptr", realloc() crée une nouvelle allocation, y copie autant de données de l'ancienne allocation que possible dans la limite de la taille de la nouvelle allocation, libère l'ancienne allocation et retourne un pointeur sur cette nouvelle allocation.
- La fonction free() libère l'allocation de la mémoire pointée par "ptr". Si "ptr" vaut NULL, free() ne fait rien.
- En cas d'erreur, les fonctions malloc() et realloc() retournent un pointeur NULL ;
- Vous devez utiliser les syscall mmap(2) et munmap(2) pour réclamer et rendre des zones mémoire au système.
- Vous devez gérer vos propres allocations mémoires pour le fonctionnement interne de votre projet sans utiliser la fonction malloc de la libc.
- Vous devez dans un soucis de performance limiter le nombre d'appel à mmap(), mais aussi à munmap(). Vous devrez donc "pré-allouer" des zones mémoire pour y stocker vos "petits" et "moyens" malloc.
- La taille de ces zones devra impérativement être un multiple de getpagesize().

- Chaque zone doit pouvoir contenir au moins 100 allocations.
  - Les mallocs “TINY”, de 1 à n octets, seront stockés dans des zones de N octets
  - Les mallocs “SMALL”, de (n+1) à m octets, seront stockés dans des zones de M octets
  - Les mallocs “LARGE”, de (m+1) octets et plus, seront stockés hors zone, c’est à dire simplement avec un mmap(), ils seront en quelquesorte une zone à eux tout seul.
- C’est à vous de définir la taille de n, m, N et M afin de trouver un bon compromis entre vitesse (économie d’appel système) et économie de mémoire.

Vous devez également écrire une fonction permettant d’afficher l’état des zones mémoires allouées. Elle devra être prototypée comme suit :

```
void      show_alloc_mem();
```

L’affichage sera formaté par adresse croissante comme dans l’exemple suivant :

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

Chapitre III

Sujet - Partie bonus

Les bonus ne seront évalués que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc ... Concrètement, cela signifie que si votre partie obligatoire n'obtient pas TOUS les points à la notation, vos bonus seront intégralement IGNORÉS.

Des idées de bonus :

- Gérer des variables d'environnement de debug de malloc. Vous pouvez imiter celles du malloc du système ou inventer les votres.
- Créer une fonction show_alloc_mem_ex() qui permet d'afficher plus de détails, par exemple un historique des allocations, ou un dump hexa des zones allouées.
- "Défragmenter" la mémoire libérée.
- Gérer l'utilisation de votre malloc dans un programme en multi-thread (donc être "thread safe", et ce avec la lib pthread).

Chapitre IV

Consignes

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- La librairie doit se nommer `libft_malloc_$HOSTTYPE.so`
- Vous devez rendre un Makefile. Il devra compiler la librairie, et contenir les règles habituelles. Il ne doit recompiler la librairie qu'en cas de nécessité.
- Votre Makefile devra se charger de vérifier l'existence de la variable d'environnement $HOSTTYPE. Si elle est vide ou inexistante, lui assigner la valeur suivante : `‘uname -m‘_‘uname -s‘`

```
ifeq ($(HOSTTYPE),)
    HOSTTYPE := $(shell uname -m)_$(shell uname -s)
endif
```

- Votre Makefile devra créer un lien symbolique `libft_malloc.so` pointant vers `libft_malloc_$HOSTTYPE.so` donc par exemple :
libft_malloc.so -> libft_malloc_intel-mac.so
- Si vous êtes malin et que vous utilisez votre biliothèque `libft` pour votre `malloc`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépôt de rendu. Votre `Makefile` devra compiler la librairie, en appelant son `Makefile`, puis compiler votre projet.
- Vous pouvez avoir une variable globale pour gérer vos allocations et une pour le thread-safe.
- Votre projet doit être à la Norme.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...).
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
$>
```

- Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes :
  - mmap(2)
  - munmap(2)
  - getpagesize(3)
  - getrlimit(2)
  - les fonctions autorisées dans le cadre de votre libft (write(2) par exemple ;-))
  - les fonctions de la libpthread
- Vous avez l'autorisation d'utiliser d'autres fonctions dans le cadre de vos bonus, à condition que leur utilisation soit dûment justifiée lors de votre correction. Soyez malins.
- Vous pouvez poser vos questions sur le forum dans l'intranet.

Chapitre V

Notation

- La notation de `malloc` s'effectue en deux temps :
  - En premier lieu, votre partie obligatoire sera testée. Elle sera notée sur 20 points.
  - Ensuite, vos bonus seront évalués. Ils seront notés sur 10 points.
    - Ils ne seront évalués que si votre partie obligatoire est PARFAITE (Tout doit fonctionner comme attendu, et la gestion d'erreur doit être sans faille).
    - Vous obtiendrez entre 1 et plusieurs points par fonctionnalité bonus distincte et correctement réalisée (À la discrétion de vos correcteurs)
    - Également, l'optimisation de la qualité de certains éléments de votre code seront évalués et pourront donner lieu à des points supplémentaires dans cette partie bonus.
- Bon courage à tous !

Chapitre VI

Rions un peu

Dans un pas si lointain passé, le projet malloc se faisait avec brk(2) et sbrk(2) en lieu et place de mmap(2) et munmap(2). Voici ce que le man de brk(2) et sbrk(2) a à dire sur l'époque des dinosaures :

```
$> man 2 brk
...
DESCRIPTION
The brk and sbrk functions are historical curiosities left over from earlier days before the advent of
   virtual memory management.
...
4th Berkeley Distribution      December 11, 1993     4th Berkeley Distribution
$>
```

De cette description découle très certainement la concision de l'implémentation de brk(2) sur Mac Os X :

```
void *brk(void *x)
{
  errno = ENOMEM;
  return((void *)-1);
}
```