LSINF1252
Rapport de projet 1 :
Implémentation de malloc, calloc et free

Aigret Julien (8343-13-00) Vanvyve Nicolas (6590-13-00)

16 mars 2016

1 Gestion de la mémoire

1.1 Initialisation

Nous avons fait le choix d'initialiser notre `heap` à 1MB, taille qui semble raisonnable pour notre utilisation actuelle, mais qui eut être modifiée si besoin est. Cette limite n'est pas modifiée tout au long de l'éxécution de notre code.

1.2 Fragmentation

Dans notre programme, nous avons fait le choix d'allouer immédiatement une zone de mémoire (ou une combinaison de zones de mémoire) si elle est de la taille demandée par l'utilisateur. Dans le cas où il n'y a pas de zone de mémoire de taille précise, deux cas de figure se présentent : soit une ou des zones plus grandes ont été trouvées, ou aucune de taille suffisante n'a été trouvée. Dans le cas où une plus grande zone a été trouvée, nous gardons la plus grande de toutes et y installons notre zone de mémoire, créant un nouveau `block_header` à la fin de celle-ci pointant sur la fin de la "zone libre".

2 Tests unitaires

Nous avons implémenté les tests unitaires décrits ci dessous via CUnit.

- Myfree desalloc: nous vérifions que `alloc` de `block_header` vaut bien 0.
- Mymalloc alloc : idem que ci dessus mais `alloc` de `block_header` doit valoir 1
- Mycalloc alloc : idem que Mymalloc alloc
- Mymalloc size : nous vérifions que l'espace alloué correspond bien au multiple de 4 supérieur ou égal(en byte) de celui demandé.
- Mymalloc NULL : nous vérifions que si 0 byte sont demandé `mymalloc` renvoie `NULL`

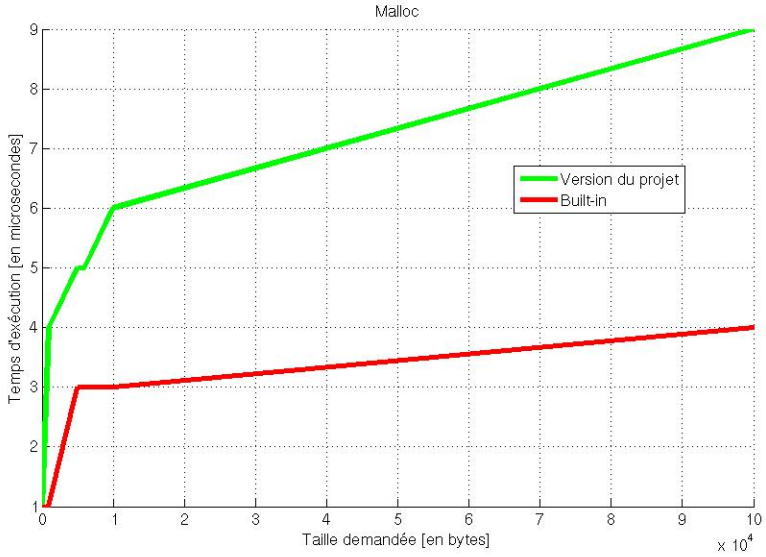

Figure 1: Evolution de la vitesse d'allocation en fonction de la mémoire pour `malloc`

- Mymalloc 2 alloc : nous vérifions que deux allocation successive via `mymalloc` ne pointe pas vers la meme adresse
- Mycalloc initialisation : nous vérifions que après un appel à `mycalloc` la mémoire allouée est bien initialisée à 0

Les test qui s'applique à `mymalloc` aurait pu etre transformés pour `mycalloc`. Mais étant donné que nous avons utilisé `mymalloc` pour implémenter le second cela ne servirait à rien.

3 Tests comparatifs

Nous constatons, comme illustré sur les graphe (fig 1 & 2) que nos fonctions sont plus rapides pour les faibles allocations de mémoires, mais plus la mémoire devient grande et plus elles prennent du temps. Cela est certainement dû aux algorithmes de recherches utilisés par les fonctions `built-in` qui doivent être optimisés pour les grandes zones de mémoire. Notre initialisation de la zone mémoire par `calloc` est aussi très rudimentaire, et prend de plus en plus de temps au fur et à mesure que la taille de la zone à initialiser augmente.

4 Difficultés rencontées

Nous avons eu des difficultés avec la compréhension de la structure de `block_header` comme décrite dans les consignes, ainsi que lors des précisions de consignes de la non-augmentaion du `heap`. Le fonctionnement de `CUnit` a été compliqué à comprendre car mal expliqué dans les ressources du cours.

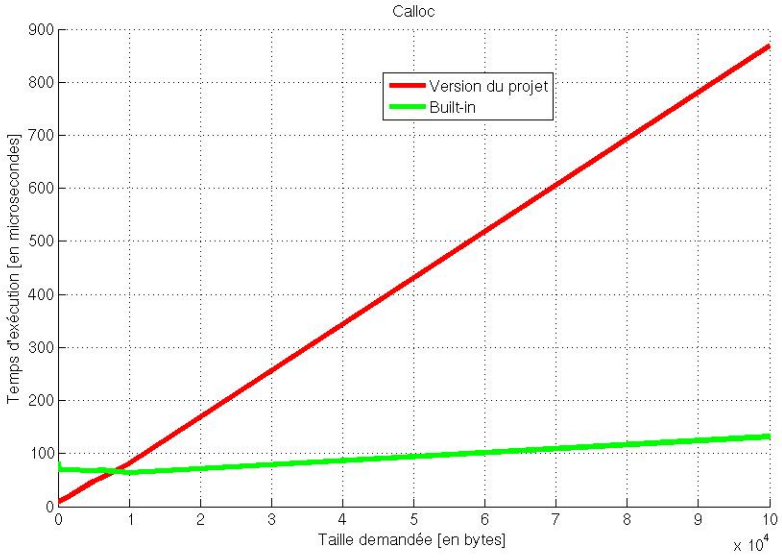

Figure 2: Evolution de la vitesse d'allocation en fonction de la mémoire pour `calloc`

5 Commentaires

Les commentaires du codes sont intégralement dans le code.