SINF1252 - Rapport - Implémentation de malloc et free

Denauw Antoine
De Carvalho Borges Antonio

16 mars 2016

Table des matières

1 Explications de l'implémentation

malloc : Pour implémenter la fonction my_malloc(), nous parcourons l'espace mémoire initialisée à la recherche d'un block header non-alloué et pouvant contenir la zone mémoire passée en paramètre de la fonction.

calloc : Pour implémenter la fonction my_calloc(), nous faisons appel à la fonction my_malloc, que l'on a implémenté, puis nous mettons toutes les valeurs de la zone mémoire demandée à zéro conformément à la man page de calloc().

free : Pour la fonction my_free(), nous libérons dans un premier temps le bloc que l'ont nous demande de libérer. Dans un second temps, nous parcourons tout l'espace mémoire que l'on nous a donné à l'initialisation. Pendant ce parcours, nous cherchons des blocs non-alloués consécutifs pour les fusionner ensemble.

2 Difficultés rencontrées

2.1 Globales

Les difficultés rencontrées ont été assez nombreuses, tout d'abord, la visualisation du problème était assez complexe. Nous avons eu un peu de mal à comprendre précisément ce qui était demander et nous avons aussi remarqué qu'il était essentiel de comprendre ce que faisais malloc dans les moindre détails pour pouvoir en créer un équivalent.

Une fois compris clairement les consignes et malloc, nous nous sommes attaqués à la structure de tout les fichiers que devait regrouper notre programme car c'est la première fois que nous utilisions CUnit mais aussi un Makefile qui devait être un peu plus complet que dans le projet précédent.

2.2 malloc

Pour l'implémentation de malloc, nous nous sommes réunis en salle infos et nous avons donné nos idées pour les comparer et ensuite nous avons codé pas à pas la fonction malloc. Nous avons eu quelques difficultés au niveau des pointeurs mais aussi pour gérer la mémoire, nous entendons par là les cas où il n'y avait plus de place dans le bloc mémoire demandé que nous avons alloué via la fonction sbrk. Les cas d'appel répété à malloc car nous devions regarder si, dans les blocs qui ont été alloué, et, qui ont été free par après, étaient assez grand pour accueillir l'espace demandé par l'appel suivant de malloc.

2.3 free

La partie la plus sensible au niveau de free à été de rechercher les blocs consécutifs pour les fusionner via la fonction my_fragmentation().

2.4 calloc

La fonction calloc a été de loin la plus facile étant donné que nous avions déjà implémenter my_malloc, nous n'avions qu'à la rapeller pour ensuite initialiser tout les bytes de la zone à zéro. Nous n'avons pas eu de dificultés à l'implémenter et nous nous sommes aidé de la fonction memset qui nous a permit de facilement remplir la zone de zéros.

3 Cas couverts par nos tests unitaires

Pour les tests unitaire, nous avons voulu être le plus complet possible pour chaque branches de nos implémentation que ce soit pour malloc, calloc, free ou de leurs performances par rapport aux vraies fonctions.

Nous avons

4 Performance de notre implémentation

Nos tests unitaire vérifient la vitesse à laquelle s'exécutent nos malloc, free et calloc par rapport aux vraies fonction. Même si cela n'était peut être pas demandé, il nous a semblé important que notre programme soit rapide tout comme efficace.