SALAMANI Léandre
SAULLE Laura

TP Allocation mémoire

Ce TP avait pour but de réaliser un allocateur mémoire contigüe. Aussi il était nécessaire de faire toutes les fonctions de l'interface de l'allocateur.

Il a fallut définir différentes structures avant de commencer :

- ***struct fb*** représente une zone libre. Elle contient la taille de la zone (structure comprise), ainsi qu'un pointeur vers la zone libre suivante.
- ***struct bb*** représente une zone allouée. Elle contient uniquement la taille de la zone (structure comprise aussi).
  Nous avons choisi de ne pas faire une liste chaînée des zones occupées car nous aurions perdu de l'espace de stockage, et nous n'aurions pas gagné beaucoup en temps d'exécution.
- ***global_s*** est une structure qui est placée en début de mémoire, afin de retrouver les informations capitale pour la gestion de la mémoire : un pointeur vers une fonction d'allocation, et un pointeur vers la première zone libre.

Fonctionnalités :

- ***void mem_init() ;***

Cette procédure initialise la mémoire avec une structure global_s. La stratégie d'allocation définit par défaut est le First Fit ; la liste des zones libres commence à l'octet suivant la stucture.

- ***void mem_fit(mem_fit_function_t* mff) ;***

Cette procédure modifie la fonction d'allocation dans la structure global_s.

- ***struct fb* mem_first_fit(struct fb* head, size_t size) ;***
- ***struct fb* mem_best_fit(struct fb* head, size_t size) ;***
- ***struct fb* mem_worst_fit(struct fb* head, size_t size) ;***

Ces procédures correspondent aux différents algorithmes de stratégies d'allocation.

- ***void mem_show(void (*print)(void *, size_t, int free)) ;***

Cette procédure affiche l'étât de la mémoire avec la fonction donnée en paramètre.
Comme elle est appelée quelque soit l'affichage voulu (zones libres, occupées, ou toutes), on doit parcourir toute la mémoire par ordre croissant d'adresse. On lit donc zone par zone jusqu'à ce qu'on atteigne la fin de la mémoire.

- ***void* mem_alloc(size_t size) ;***

Cette fonction alloue un espace de taille *size* dans la mémoire. Elle renvoie l'adresse du début de la zone libre si elle réussit, NULL sinon.
Nous avons décidé qu'une allocation d'une taille 0 renverrait NULL. En effet, nous aurions perdu ces zones en les allouant, car une *stucture fb* ne peut pas être stockée dans une zone de taille de *structure bb*.
Une allocation est faite en trois étapes :

- une zone est cherchée avec la stratégie d'allocation ;
- elle est supprimée de la liste des zones vides (avec ***suppr_zone***)
- si elle est trop grande, le surplus est réinséré dans la liste. (avec ***add_zone***)

Ce choix est coûteux car il nécessite deux parcours de liste, alors qu'il est possible de faire cela en un seul. Cependant nous avons préféré cette implémentation car elle était plus compréhensible.

- ***void mem_free(void* zone) ;***

Cette procédure libère la zone mémoire pointée par *zone*. Le pointeur vaut NULL à la fin.
La libération se fait en deux étapes :

- la zone mémoire est ajoutée dans la liste des zones libres
- un parcours de la liste fusionne les zones si plusieurs sont devenues contigües (avec ***merge_zone***)

Cette stratégie demande deux parcours de la liste des zones vides, mais est plus simple à appréhender.

- ***struct fb* add_zone(struct fb* new_zone, struct fb* head) ;***
- ***struct fb* suppr_zone(struct fb* old_zone, struct fb* head) ;***

Ces deux fonctions permettent de gérer la liste des zones libres. Elles renvoient toutes les deux l'adresse de la liste modifiée.
Elles permettent d'ajouter ou de supprimer un maillon en conservant toujours un ordre croissant des adresses.

- ***struct fb* merge_zone(struct fb* zone, struct fb* head) ;***

Cette fonction fusionne *zone* avec les zones libres avant et après, s'il y en a.
Elle permet de reconstruire la mémoire, afin qu'il n'y ait pas plusieurs zones libres contigües.

Tests :

Nous avons modifié les tests existant pour inclure les différentes stratégies d'allocations : les tests *test_base*, *test_fusion* et *test_cheese* effectuent des tests sur les 3 stratégies d'allocation. Les tests fonctionnent bien, excepté une erreur avec la stratégie *worst_fit* sur le test *test_cheese* (erreur de test assert()), que nous n'avons pas réussi à débugger.

Nous n'avons pas créer de nouveaux tests par manque de temps, mais aussi parce que nous pensons que les tests fournis étaient assez complets.

Pour aller plus loin :

Nous avons commencé à implémenter ***mem_realloc***. Mais la partie qui consiste à agrandir la taille est complexe, et nous avons manqué de temps. Un ébauche est en commentaire dans le fichier **mem.c**.