LOUGHRAN Nicholas
ROBERT François

Compte rendu TP Allocateur Mémoire

Choix d'implémentation :

Nous avons décidé de créer plusieurs structures pour différencier les blocs alloués des blocs libres.
La structure fb, représentant les « free bloc » : elle contient un champ renseignant la **taille** de la zone libre (en comptant la taille que cette même structure prend dans cette zone), et **un pointeur sur le prochain « free_bloc ».**
Ceci nous permet de parcourir la liste des zones libres facilement, quand on arrive en bout de liste le pointeur sur le prochain « free bloc » à la valeur NULL.
Pour représenter les zones occupées nous avons une structure ab (pour allocated bloc) : Elle renseigne uniquement la **taille** du bloc occupé (sans compter la taille de cette structure, qui est placé avant le début de la vraie zone occupée, voir schéma ci dessous).
Nous avons également une structure d'en tête de mémoire : cette structure contient un pointeur sur la fonction qui décrit la stratégie d'allocation à utiliser, elle contient également un pointeur sur le premier élément de la liste des zones libres. Cet en tête de mémoire est rempli à l'initialisation de la mémoire dans la fonction mem_init(). Cette structure permet d'accéder de façon plus claire à la tête de la liste des zones libre dans notre code par exemple.

Nous n'avons pas créé de liste de bloc alloué comme pour les bloc libre, car nous utilisons le fait qu'on connaît où sont les blocs libre, leur taille, de plus nous pouvons retrouver la taille de chaque zone occupée en début de cette dites zone, ce mécanisme est utilisé dans la procédure :
void mem_free(void* zone)

Pour la taille des zones alouées : la taille des zones allouées respecte un alignement défini dans la variable globale MEM_ALIGN, nous avons mit cette variable à 8 dans mem.c.

Fonctionnalités de l'allocateur :

Initialisation de la mémoire avec la fonction void mem_init()

Définition de la stratégie d'allocation avec void mem_fit(mem_fit_function_t* mff)

Affichage des zones en mémoire avec void mem_show(void (*print)(void *, size_t, int free))

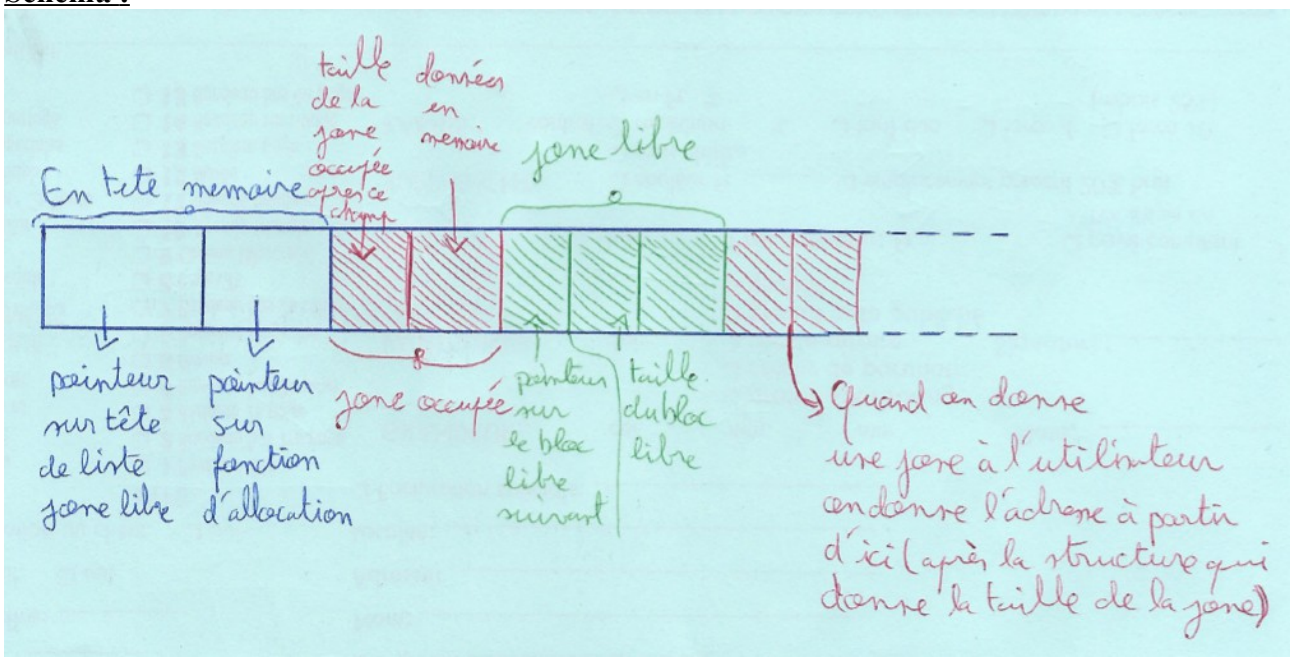
Libération d'une zone mémoire occupée avec void mem_free(void* zone)

Allocation d'une zone de taille demandée avec void* mem_alloc(size_t size)

Tests :

Notre allocateur n'arrive pas à passer les tests proposés mis à part test_init.
Les tests : test_base , test_cheese et test_fusion donnent une erreur de segmentation que nous n'avons pas réussi à corriger malgré une analyse de l'exécution avec l'outil gdb

Schéma :


taille de la zone occupée par ce champ
données en mémoire
zone libre
En tête mémoire
pointeur sur tête de liste zone libre
pointeur sur fonction d'allocation
zone occupée
pointeur sur le bloc libre suivant
taille du bloc libre
Quand on donne une zone à l'utilisateur on donne l'adresse à partir d'ici (après la structure qui donne la taille de la zone)