Projet allocateur mémoire

Arthur Millet, Antonia Ivanova - binôme AB

19/12/2023

Nos structures de données

Header

Notre header (bloc de métadonnées), situé au début de la mémoire, est une structure contenant 4 champs qui sont :

- la taille de la mémoire de type size_t
- un pointeur vers la fonction de recherche de zone libre (par défaut : mem_fit_first)
- un pointeur vers la tête de la liste chaînée des zones libres
- un size_t représentant la taille de la plus grande zone mémoire libre (qu'on voulait utiliser pour implémenter des extensions)

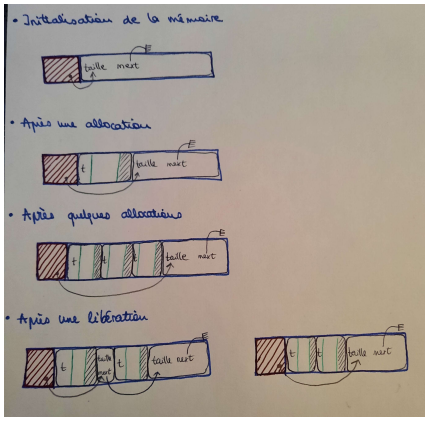
Zone libre

C'est une structure type liste chaînée contenant la taille (= taille de la zone comprenant les métadonnées) de la cellule et un pointeur vers la cellule suivante.

Zone occupée

Contient uniquement un champs représentant sa taille (= la taille de la zone sans le champs taille).

Représentation de la mémoire



mem_init

Dans cette fonction, il faut initialiser les différents champs de notre bloc de métadonnées (header). Pour cela, on crée donc une liste chaînée de zones libres ne contenant qu'une seule zone libre située juste après le header, et la zone libre suivante est donc NULL. Sa taille est la taille totale de la mémoire moins la taille du header. Enfin, on charge la fonction de recherche de zone libre (dans notre cas c'est mem_fit_first()).

mem_show

Cette fonction permet d'afficher les différentes information sur la mémoire, comment elle est occupée. En effet, mem_show affiche chaque zone libre et occupée, son adresse et enfin sa taille. Pour implémenter cette fonction, le plus simple est de parcourir la mémoire et pour chaque zone afficher ses informations grâce à la fonction print().

mem_fit_first

On parcourt la liste chaînée des zones libres jusqu'à trouver la première zone libre de taille suffisante et on retourne la zone.

mem_fit_best

On parcourt la liste chaînée des zones libres et on met à jour une variable locale plus_petite_zl de type zone libre qui a la plus petite taille suffisante. Une fois qu'on a fini de parcourir la liste, on renvoie plus_petite_zone.

mem_fit_worst

Même idée que pour mem_fit_best, sauf qu'on met à jour plus_grande_zone quand la taille de la zone libre pendant le parcours est suffisante et est supérieure à la taille actuelle de plus_grande_zone.

mem_alloc

La première chose importante est de gérer l'alignement. C'est pour cela qu'on a décidé de créer une fonction annexe, qui prend en argument deux entiers, un pour la taille et l'autre pour l'alignement et renvoie la taille aligné (sur 8 bits dans notre cas).

On récupère le pointeur vers la zone libre grâce à la fonction de recherche, et si la taille de la zone est correcte on va chercher la zone libre précédente afin mettre à jour les liens de chaînage et on insère dans la mémoire notre nouvelle zone occupée.

mem_free

La partie la plus importante de cette fonction est de mettre à jour correctement la liste chaînée des zones libres. C'est pour cela qu'on a défini une fonction annexe fusionner_zl(), qui parcourt toute la mémoire et dès qu'elle trouve 2 zones libres adjacentes, elle va les fusionner en une seule zone libre, en mettant ainsi a jour la taille de la zone libre fusionnée.

Voici comment on transforme une zone occupée en zone libre : le type zone libre contient 2 champs alors que le type zone libre ne contient qu'un seul champs. Donc lorsqu'on va libérer la zone occupée, il faut diviser la zone en 2, d'un côté on va mettre le pointeur vers la prochaine zone libre et d'un autre côté, l'utilisateur pourra y stocker les informations qu'il souhaite.

Conclusion

Malheureusement par manque de temps, dû à de nombreux problèmes rencontrés dans l'implémentation de la fonction mem_free(), nous avons eu juste eu le temps d'implémenter les autres politiques : mem_fit_best et mem_fit_worst qui fonctionnent parfaitement !