MULLER Matthieu
MERMET Arthur

ALLOCATEUR MÉMOIRE

Choix pour l'implémentation

mem_free : Pour l'implémentation du free et pour rendre notre code plus simple et clair, on libère d'abord le bloc en le chainant aux autres blocs libres puis on appelle la fonction **fusion_fb** pour permettre de regrouper les zones libres adjacentes.

struct ub et **struct fb** : Ces structures représentent les partitions occupées et libres de la mémoire. Pour **fb**, on implémente la taille et le prochain bloc libre pour parcourir facilement cette liste. Pour **ub** cependant, pour économiser de la mémoire, on implémente seulement sa taille.

mem_alloc : Pour éviter tout problème lors de la libération des zones, on fait seulement des allocations supérieures à la structure la plus lourde (**fb**) (en agrandissant si besoin si le paramètre **taille** est trop faible) . De plus, toutes les allocations sont, comme dans le malloc de la librairie standard C, alignées sur un padding de 16 octets. Enfin, pour que l'utilisateur puisse écrire directement sur cette zone, la fonction retourne un pointeur sur le début de la zone allouée.

mem_init : On initialise la mémoire et une fonction particulière parmi **mem_fit_first, mem_fit_best** et **mem_fit_worst** qui représente chacune une manière différente de gérer dans quelle zone libre on alloue de la mémoire. Par défaut, on utilise **mem_fit_best** .

Utilisation

mem_init(mem_addr, taille_zone)
Initialise le système mémoire de **taille_zone** à l'adresse **mem_addr**

mem_alloc(taille)
Renvoie l'adresse d'une zone ou l'on peut écrire des données de taille allant jusqu'à **taille** au maximum

mem_free(adresse)
Libère le bloc mémoire à l'adresse **adresse** si il était occupé, ne fait rien sinon. L'adresse passée doit être l'une de celles retournée par **mem_alloc.**

mem_show(print)
Affiche à l'écran les zones libres et occupées de la mémoir en utilisant la fonction d'adresse "**print**"

Tests effectués

test_alloc.c :
Première allocation de 10 zones de tailles aléatoire mais sans débordement

MULLER Matthieu
MERMET Arthur

Réinitialisation de la mémoire puis une allocation de taille nulle
Réinitialisation de la mémoire puis une allocation de taille supérieure à la taille disponible. La mémoire affiché pour les deux derniers tests doit être une unique zone libre de la taille de la mémoire

Test free :
allocations de plusieurs zones de tailles aléatoires et stockage des adresses retournées par **mem_alloc.** On libère une zone sur deux en alternant, puis les zones restantes en s'assurant à la fin qu'il ne reste plus qu'un bloc libre qui recouvre toute la mémoire.

On a également testé la libération d'une zone libre puis la libération d'une adresse aléatoire dans notre système mémoire grâce à **memshell**

Makefile

- make
Pour compiler **memshell.c** et tous les tests sans les exécuter.

- make test
Pour compiler tous les tests et les exécuter.

- make clean
Pour effacer les fichiers compilés

- make test_ls
Pour tester notre implémentation sur la fonction shell **ls** a la place de la librairie par défaut

Limites de l'implémentation :

- Le padding de 16 octet peut représenter une grosse perte de mémoire si toutes les zone alloué ont une taille congrue à 1 modulo 16. En effet, sur les 16 octets alloués, un seul sera utilisé.

- De plus notre implémentation permet plusieurs manières de sélectionner les zones à allouer mais ne permet pas à l'utilisateur de choisir cette dernière sans intervenir dans le code même.

\0