SALMON Alexandre
GUIVARCH Alan

CPS: Allocateur Mémoire

I. Fonctionement général de l'allocateur

1. Initialisation

L'état de base de la mémoire, celui auquel est initalisé la mémoire est le suivant:
- Les 8 premiers octets de la mémoire contiennent l'adresse du premier bloc libre. En l'occurence à l'initialisation, c'est l'adresse 8.
- Le premier bloc libre a pour taille initiale toute la mémoire privé de l'entête de la mémoire (les 8 octets décrits plus tôt). Le champ "next" du bloc libre est initialisé à NULL.
- On désign la politique de fit.

2. Parcours de tout les blocs

Pour l'affichage, on doit parcourir tout les blocs dans l'ordre, occupés ou non. On peut traiter les blocs comme un liste implicitement chaînée: pour chaque bloc, son pointeur de début plus sa taille indique l'adresse du bloc suivant.

On a donc là une solution pour parcourir tout les blocs, encore faut il savoir si ils sont libre ou non. Plusieur solutions sont possibles:
Garder trace du prochain bloc libre et comparer son adresse à celle du bloc parcouru en est une.
La solution que nous avons implémenté consiste à enregistrer l'état du bloc dans le bit de poids faible de celui-ci (1 voulant dire libre, 0 non). Les tailles étant systématiquement des multiples de 8, les bits de poids faible des tailles sont toujours à 0. Des masques permettent de déterminer la taille d'un bloc ou si il est occupé.

3. Fonctions de Fit et Padding

Les fonctions de fit se basent sur un simple parcours de la liste chainée des blocs libres et renvoie selon les fonctions le premier bloc trouvé, le plus grand bloc ou le plus petit. Pour toute les fonctions le bloc retourné doit être assez grand pour y stocker la taille demandée par l'utilisateur et le padding.

Dans cette implémentation, le padding d'un bloc, requis pour l'aligner sur 16 octets, est situé à la fin de celui-ci. Pour determiner le padding d'un bloc à partir une taille, d'une adresse, on utilise la formule suivante:
adresse & ~(ALIGN - 1) détaillée en commentaire dans le code de la fonction padding.

4. Allocation

Le role de la fonction mem-alloc est de trouver un bloc assez grand pour y stocker le nombre d'octet défini par la variable taille ainsi que le padding requis pour l'alignement.

Une fois ce bloc trouvé la fonction doit réduire ou détruire le bloc libre que l'on utilise, mettre à jour la liste des blocs libre et initialiser la taille d'un block occupé.

Un bloc occupé est représenté par un size_t suivi d'un espace utilisable par l'utilisateur de taille supérieure ou égale à celle demandée. Le pointeur de début de cet espace est celui aligné sur 16 que l'on renvoie à l'utilisateur.

La destruction d'un bloc libre doit résulter su la mise à jour de la liste voire même de la tête de liste si le bloc était le premier bloc de la liste.

5. La libération

La libération d'un zone mémoire ne doit libérer une zone que si l'adresse fournie corresponds à une adresse qui a été allouée plus tot. La libération doit prends plusieur cas de figure en compte:
- La zone libéré est la première zone libre en mémoire, il faut donc la transformer en bloc libre, lui donner comme element "next" le premier bloc libre connu et mettre l'adresse de ce nouveau bloc en début de la mémoire.
- La zone libérée est directement précédée d'un bloc, Il faut donc agrandir le bloc précédent pour qu'il recoure la zone à libérer.
- Dans les autres cas il faut créer un nouveau bloc libre à l'adresse à libérer qui remplace le blococcupé.
- Si un bloc nouvellement créé est suivi directement d'un bloc libre, on fusionne les deux

II. Questions

Question 3

L'utilisateur ne peut pas manipuler toutes les adresses: les adresse de 0 à 7 sont réservées pour le pointeur vers le premier bloc libre. De plus chaque bloc libre ou non est composé d'une entête à laquelle l'utilisateur ne peut pas toucher.

Question 4

Le pointeur retourné par mem_alloc est le pointeur de début de la zone utilisable du bloc occupé, càd le pointeur qui suit les entêtes.

Question 5

Si la taille restant d'un bloc libre après l'allocation est trop petite, on peut agrandir la zone que l'on vient d'allouer afin qu'elle comprenne les chuttes du bloc libre.

III. Extensions

En plus des fonctionalités de base, nous avons mit les variables dont nous avons besoin dans la mémoire, c'est l'entête en début de mémoire.
Nous avons implémenté les fonctions worst_fit et best_fit. Nous n'avons pas effectué de test pour comparer l'efficacité des fonctions de fit. On peut supposer que first_fit est la plus rapide à renvoyer un bloc, c'est la seule à ne pas parcourir tout les blocs. La fragmentation est un facteur à prendre en compte: best_fit est la fonction qui

laisse le moins de rab quand il occupe un bloc, cependant c’est aussi celui qui produit les plus petit restes donc les moins utilisables.
worst_fit produit toujours des restes mais ceux ci sont assez grand pour être utiles.