MATHIEU Lucas
PUCCI Jérémy

Compte rendu TP Allocateur Mémoire

L'objectif de ce TP est d'implémenter un allocateur mémoire réaliste et de créer des programmes de test permettant de vérifier la bonne implémentation.

Nos choix d'implémentation

- L'allocateur comporte une structure allocator_header permettant de garder en mémoire la taille totale de la mémoire qui est utilisée par l'allocateur, ainsi que le premier bloc libre et la fonction utilisée pour la recherche de bloc libre.

- Chaque bloc est défini par une structure fb située à l'adresse du bloc, elle contient la taille du bloc et le bloc suivant. Si le bloc est un bloc utilisé (donc pas libre), le bloc suivant n'est pas défini. La chaîne de blocs permet de garder une trace des blocs libres.

- Lors de l'initialisation de la mémoire, on crée un unique bloc libre qui est de la taille de la mémoire.

- Afin d'allouer une zone libre, la fonction de recherche du header est utilisée, la fonction mem_fit_first retournant le premier bloc libre qui a la taille exacte demandée ou au moins la taille demandée plus la taille d'une structure fb, car il est nécessaire de créer un nouveau bloc libre avec la taille restante pour ne pas perdre de la mémoire.

- Une fois le bloc choisi, on le retire donc de la chaîne des blocs libres et si nécessaire, on modifie sa taille et on crée un nouveau bloc libre à la suite (que l'on ajoute à la liste des blocs libres).

- Pour libérer un bloc, on le rajoute simplement dans la chaîne des blocs libres. Si c'est possible, on effectue également la fusion des blocs libres, c'est-à-dire que si on libère un bloc à côté d'un autre bloc libre, on les fusionne en un seul bloc libre. Cela permet de ne pas conserver plein de petits blocs libres et ne plus pouvoir allouer une grande quantité de mémoire.

Les limites de notre code

- Notre code n'a qu'une seule fonction de recherche de bloc libre, celle qui choisit le premier bloc convenable. Mais grâce à l'implémentation de la structure allocator_header, il est très simple de changer la fonction de recherche.

- Notre code ne vérifie pas l'adresse lors de la libération d'un bloc, c'est donc à l'utilisateur de faire attention à l'adresse envoyée à cette fonction sinon la mémoire risque d'être corrompue.

- Enfin, notre code ne vérifie pas les dépassements de mémoire, qui corrompent également la mémoire.

Commentaires des tests

```
Test test1 :
? Memoire allouee en 24
? Memoire allouee en 56
? Memoire allouee en 184
? Memoire allouee en 1208
? Fin de la mémoire: 128000
Première zone libre: 11224
Zone occupee, Adresse : 24, Taille : 16
Zone occupee, Adresse : 56, Taille : 112
Zone occupee, Adresse : 184, Taille : 1008
Zone occupee, Adresse : 1208, Taille : 10000
Zone libre, Adresse : 11224, Taille : 116760
?
```

Le premier test est simplement l'allocation successive de taille différentes. Tout est cohérent.

```
Test test2 :
? Memoire allouee en 24
? Memoire allouee en 56
? Memoire allouee en 184
? Memoire allouee en 1208
? Memoire liberee
? Memoire allouee en 184
? Fin de la mémoire: 128000
Première zone libre: 312
Zone occupee, Adresse : 24, Taille : 16
Zone occupee, Adresse : 56, Taille : 112
Zone occupee, Adresse : 184, Taille : 112
Zone libre, Adresse : 312, Taille : 880
Zone occupee, Adresse : 1208, Taille : 10000
Zone libre, Adresse : 11224, Taille : 116760
?
```

Le second test est la suite du test1 avec une libération d'une zone au milieu de deux autres et une réallocation plus petite qui a créé une nouvelle zone. Ici encore, tout est cohérent.

```
Test test3 :
? Echec de l'allocation
?
```

Une simple allocation d'une taille supérieure à la taille maximale.

```
Test test4 :
? Memoire allouee en 24
? Deleting next block 152
Memoire liberee
? Memoire allouee en 24
? Deleting next block 152
Memoire liberee
? Memoire allouee en 24
? Memoire allouee en 152
? Memoire liberee
? Fin de la mémoire: 128000
Première zone libre: 24
Zone libre, Adresse : 24, Taille : 112
Zone occupee, Adresse : 152, Taille : 112
Zone libre, Adresse : 280, Taille : 127704
?
```

L'allocation et la libération successive d'une zone de taille 100 puis deux allocations de taille 100 et une libération.

```
Test test5 :
? Memoire allouee en 24
? Memoire allouee en 56
? Memoire allouee en 88
? Fin de la mémoire: 128000
Première zone libre: 120
Zone occupee, Adresse : 24, Taille : 16
Zone occupee, Adresse : 56, Taille : 16
Zone occupee, Adresse : 88, Taille : 16
Zone libre, Adresse : 120, Taille : 127864
? Memoire liberee
? Fin de la mémoire: 128000
Première zone libre: 56
Zone occupee, Adresse : 24, Taille : 16
Zone libre, Adresse : 56, Taille : 16
Zone occupee, Adresse : 88, Taille : 16
Zone libre, Adresse : 120, Taille : 127864
? Deleting previous block 88
Deleting next block 120
Memoire liberee
? Fin de la mémoire: 128000
Première zone libre: 56
Zone occupee, Adresse : 24, Taille : 16
Zone libre, Adresse : 56, Taille : 127928
?
```

Test du merge des zones libres. On alloue 3 zones de taille 10, on libère celle du milieu puis (en supposant les adresses croissantes vers la droite) la zone la plus à droite. On voit bien le merge entre la zone libre du milieu, la zone que l'on vient de libérer et l'espace mémoire qu'il restait.

Conclusion

Nous avons créé un allocateur mémoire unique et réaliste qui permet à un utilisateur d'allouer et de libérer des zones mémoires. Un système de merge de zone libre a été implémenté pour le rendre plus efficace. Pour vérifier notre bonne implémentation, nous avons créé des programmes de test.