CANIN Corentin CHAPPAZ Florian BESANTE Joan	Compte rendu API TP3	INFO3

Choix des structures :

Nous avons fait le choix de n'utiliser qu'une seule et même structure, nommée fb, pour la liste des blocs libres mais aussi pour celle des blocs occupés. Cette structure contient la taille du bloc et un pointeur sur le bloc suivant. Ce choix est justifié par le fait que, plusieurs fois dans le programme, nous sommes amenés à faire des opérations ou des comparaison entre pointeur de bloc libre et pointeur de bloc occupé. Si nous avions deux structures différentes, un cast aurait été nécessaire pour chaque test ou opération, ce qui ajoute une source d'erreur et rend le programme moins lisible.

Pour ce qui est de l'allocator_header, en plus des propriétés size et mem_fit_function, on ajoute deux pointeurs : l'un pointe sur le premier bloc de la liste des blocs libres, l'autre pointe sur le premier bloc de la liste des blocs occupés.

Exemple de mémoire :

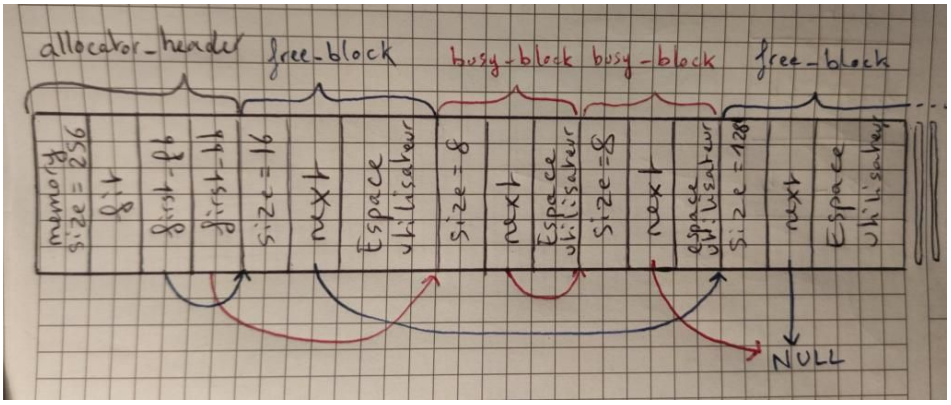

Figure 1 : Schéma d'un exemple de mémoire

Sur ce schéma, on peut voir un état exemple de la mémoire selon notre modèle. Une largeur de 2 carreaux correspond à un bloc de taille 8.

On observe bien l'entête "allocator_header" de la mémoire et ses quatre champs suivie d'une succession de blocs libres ("free_block") ou occupés ("busy_block"). Ce sont en réalité des instances d'un même type structure. On les différencie par leur présence dans l'une ou l'autre des listes chaînées qui commencent par les champs first_fb et first_bb de l'entête. Lorsqu'un bloc est le dernier de son type, on passe NULL à son champ next.

On peut noter que l'espace utilisateur correspond en réalité à l'espace entre deux blocs et n'est pas véritablement implanté dans le type structure fb.

Implémentation de mem_init :

La fonction initialise simplement l'allocator_header avec les arguments donnés. Il initialise aussi la fonction fit qui sera utilisée. Enfin, il initialise un unique bloc libre qui occupe toute la mémoire réservée.

Implémentation de mem_alloc :

Retourne l'adresse à laquelle l'utilisateur peut écrire (donc pas l'adresse de l'en-tête). On respecte également un alignement de 8. La mémoire disponible pour l'utilisateur peut donc être légèrement plus grande que celle demandée. La fonction s'occupe alors d'ajouter ce bloc alloué à la liste des blocs occupés, et si le bloc libre qu'on est en train d'utiliser a une taille supérieure à celle demandée, on fait en sorte que la fin de ce bloc soit un nouveau bloc libre à part entière (NB: Si ce qui reste à la fin du bloc est d'une taille globale inférieure à 16, il n'y a pas suffisamment d'espace pour en faire un nouveau bloc libre, on offre alors cet espace à l'utilisateur).

Implémentation de mem_free :

A l'aide de multiples pointeurs qui parcourent nos deux listes chaînées, on cherche tout d'abord à voir si l'adresse donnée correspond bien à un bloc occupé, on ne fait rien sinon. Ensuite, on transforme ce bloc en bloc libre (en faisant les repointages nécessaires dans chacune des listes). On fusionne les éventuels blocs libres juxtaposés à ce bloc en déterminant si les blocs juste avant et juste après sont libres ou non.

Implémentation de mem_get_size :

Si l'adresse donnée en argument est valide, on retourne la taille du bloc pointé par cette adresse.

Implémentation de mem_show :

La fonction utilise deux pointeurs : l'un va parcourir la liste des blocs libres, l'autre celle des blocs occupés. A chaque itération, on fait appel à la fonction print (passée en argument) pour afficher le bloc à l'adresse la plus petite, en comparant simplement les deux pointeurs.

Implémentation de mem_fit_first :

La fonction utilise un pointeur qui va parcourir la liste de blocs libres donnée en argument. Si à un moment ce pointeur pointe sur un bloc dont la taille est supérieure ou égale à la taille donnée en argument, le pointeur sur ce bloc est immédiatement retourné. Si aucun bloc adapté n'est trouvé, la fonction retourne NULL. mem_fit_best et mem_fit_worst sont assez similaires, nous ne les détaillerons pas.

Tests :

En addition à test_init.c, nous avons écrit un jeu de 8 tests dans le fichier tests_user.c. Il teste plusieurs cas particuliers et est automatiquement exécuté par make tests. Les tests sont assez explicites dans leur affichage et sont commentés pour plus de précision.

Extensions :

Nous avons implémenté d'autres modes de recherche de bloc libre : mem_fit_best et mem_fit_worst. Pour changer de mode, il faut modifier la fonction mem_init.

Limites :

Après discussion avec d'autres groupes, il semblerait que certaines de nos fonctions sont très longues. Cependant cela donne un code probablement plus clair, étape par étape.

Nos structures sont également assez grandes et prennent donc beaucoup de place. Cependant nous pensons qu'elles sont plus simples d'utilisation.