GUYOT Sacha
MANISSADJIAN Gabriel

Compte rendu TP1 CPS

Question 1:

Lorsqu'on est dans une zone occupée, on a pas besoin de savoir quelles sont les autres zones occupées adjacentes. Si l'on veut connaître les zones de la mémoire qui sont occupées, il suffit de parcourir toute la mémoire de zone en zone avec un pointeur en ajoutant la taille de la zone courante au pointeur, et de parcourir en même temps les zones libres (qui sont liées entre elles) pour tester si le pointeur de la zone courante est égal à l'adresse d'une zone vide. De ce fait, il est inutile d'avoir une liste de zones occupées.

Question 2:

La seule information nécessaire pour une zone allouée est sa taille, pour connaître la place qu'elle prend dans la mémoire. Nous avons tout de même ajouté un 2ème champ qui contient la taille exacte demandée par l'utilisateur eu plus du champ qui contient la taille réellement occupée par la zone (taille demandée plus longueur du padding).

Les conventions que nous avons choisies sont :

Le début de la partie data de chaque zone occupée est alignée sur une adresse multiple de ALIGNMENT. Leurs métadonnées (= taille de la zone) sont situées avant le début de la zone, elles dépassent donc de 12 octets (taille de size_t + int).

Les métadonnées des zones libres contiennent la taille de la zone ainsi qu'un pointeur sur la zone libre suivante (la taille avant le pointeur). Seule la taille de la zone dépasse avant l'alignement (de 8 octets aussi).

Puisque les zones occupées ne contiennent pas de pointeurs sur la zone suivante, on peut comparer les tailles des zones à partir de l'alignement lorsqu'on veut savoir si une zone libre a une taille suffisante pour être allouée.

Lorsqu'on alloue une partie d'une zone vide en tant que zone occupée, on alloue à partir de la fin de la zone, pour ne pas avoir à modifier le pointeur de la zone libre à chaque allocation. On a simplement à recalculer sa taille et à créer la structure de zone occupée.

Question 3:

Les adresses auxquelles l'utilisateur peut accéder sans risque sont toutes comprises entre 16 et la taille de la mémoire allouée. Il n'est pas censé accéder à une adresse qui ne lui a pas été explicitement retournée.

Question 4:

Lorsqu'on alloue une zone, on retourne l'adresse du début de la mémoire utilisable, alignée sur un multiple de ALIGNMENT.

Question 5:

Notre stratégie d'alignement des zones nous permet de ne jamais avoir de zone trop petite pour contenir ses métadonnées. À l'allocation d'une zone occupée, on réduit la taille de la zone libre où elle est ajoutée de la taille de la zone occupée plus un ALIGNMENT, ce qui

laisse la place de stocker les métadonnées de la zone occupée (on perd donc 4 octets lorsqu'on alloue une zone occupée dans une zone vide strictement plus grande qu'elle). Si les zones sont de la même taille, les métadonnées de la zone occupée sont écrites par dessus celles de la zone vide (après avoir supprimé la zone vide de la liste des zones vides).
Pour supprimer une zone libre, on parcourt la liste des zones libres jusqu'à trouver celle dont le pointeur next pointe sur celle qu'on veut supprimer, et on change la valeur de ce pointeur afin de pointer sur la zone vide suivant la zone qu'on supprime. Lorsque la zone à supprimer est la première de la liste, on change le pointeur sur la première zone vide (stocké à l'adresse 0) pour qu'il pointe vers la deuxième zone vide.

La principale limite de notre code est le fait qu'ALIGNMENT doit impérativement être supérieur ou égal à 16 et que nous perdons 4 à 19 octets à chaque allocation pour un ALIGNMENT de 16 ((taille demandée)%ALIGNMENT + 4)

Nous avons testé les cas suivants :

- allouer et libérer une zone de taille maximale
- allouer plusieurs petites zones, de tailles multiples d'ALIGNMENT ou non, et les libérer
- allouer et libérer des zones de manière à créer des zones libres consécutives pour tester leur fusion
- tous les tests précédents à la suite et dans n'importe quel ordre afin de vérifier que les zones libres sont correctement fusionnées.