Timothee Rognon
Besante Joan

Compte rendu TP allocateur memoire

I/ Résumé

Nous avons implémenté toute les fonctions de base demandé, plus les fonctions ralloc, checkoverflow, aligne, et calloc, que nous détaillerons plus loin. Les trois stratégies fonctionnent, et nous avons effectué des tests de performance dessus.

II/ Structure d'un block

Nous utilisons deux structures, mem_free_block_t et mem_busy_block_t. Ces deux structures désignent en réalité la même structure, mais distinguer ces deux type de block permet de rendre le code plus lisible. Toute fois, cela nous contraint a faire régulièrement des cast lorsque l'on veut faire des opérations ou comparaisons sur les pointeurs des structures.
Cette structure contient la taille du bloc et un pointeur sur le bloc suivant, ainsi que deux balise, markerpre et markerpost, qui sont placé au début et a la fin de chaque bloque afin de résoudre certain problème d'overflow.

III/ Principe

Notre mémoire est donc représenter par une liste chaîné de busy_block et une de free_block.
L'allocateur possède dès l'initialisation un free et busy block virtuel(de taille égale a 0) qui repésente les 1er élément de nos listes chainé. Nous les récupérerons respectivement à l'adresse mem_space_get_addr() et mem_space_get_addr() + sizeof(mem_free_block).
La taille du block alloué lors de l'allocation peut être légèrement plus grande pour deux raison:On doit respecter un alignement de 64bit, et si l'espace restant après le block créer est inférieur a sizeof(mem_free_block_t), on l'inclue dans le block.

IV/ Implémentation

Implémentation de mem_free :
Si l'adresse n'est pas dans l'allocateur mémoire, ou qu'elle ne correspond pas a l'adresse d'un block on ne fait rien. Si l'utilisateur tente de free deux fois une même adresse, on affiche un message d'erreur.

Implémentation de mem_fit :
Les stratégie worst et best étant extrêmement similaire, nous les avons réunie dans la fonction general_fit pour évité la recopie de code. general fit change l'opérateur de comparaison dans le code en fonction de la stratégie demandé.

Implémentation de realloc
Cette fonction est naïve, elle copie le contenu d'un busy block dans un nouveau block de la taille demandé puis supprime l'ancien block. Pour augmenter les performance de cette fonction, nous pourrions vérifier si il est possible de redimensionner le block existant avant d'en créer un nouveau.

Implémentation de checkOverflow
Cette fonction permet de détecter si des donnée ont été corrompu par un overflow.
Chaque block a deux balise qui encadre l'entête. Si l'utilisateur dépasse lors de l'écriture, il va écrire sur la balise markerPre du block suivant, ce qui seras détecté lorsque la fonction checkOverflow vérifieras l'intégrité de l'entête.
On aurais pus mettre une seul balise afin de diminuer la taille de l'entête, mais deux nous permettent de détecter de plus petit overflow.

Toutefois, si l'overflow écrit a une adresse supérieur a la balise, on ne le détecte pas.
De plus, un hack peut modifier l'entête en récrivant la balise.
On considère que lorsque l'on alloue une taille plus grande que demandé par l'utilisateur, un débordement sur cette espace n'est pas un overflow.
La balise est de taille size_t afin de maximiser la tailler occuper par la balise car si on met une + petite, la place est perdu a cause de l'alignement fait par le compilateur.
On ne détecte pas l'overflow si c'est le dernier block et qu'il n'y pas de block vide après.

Implémentation de aligned
Avoir des bite aligné augmente la rapidité des accès mémoire (voir bus de donné).
Notre header est aligné sur et 64 et 32, donc il suffit d'arrondir au multiple supérieur de 32 ou 64 la taille alloué. Le type de la machine est modifiable avec la valeur de IS64BIT dans mem_os. (64 par défaut).

Implémentation calloc
Cette fonction permet d'allouer un tableau. Il alloue count cellule de taille size en fixant tout l'espace allouer a 0.

V/ Performance

Afin d'évaluer la performance de nos trois stratégie, nous allons compter le nombre de block libre a la fin de trois cas d'utilisation différent. Le nombre de block libre n'est pas le seul critère pour évaluer les performance, il y a aussi la taille de ces block, mais c'est le plus simple a tester.
Nous montrons un scénario ou best fit est le plus performant, un ou worst fit est meilleur, et un ou c'est first_fit. Ces tests ont pour but de montrer que chaque stratégie peut être pertinente en fonction des cas.

VI/ Test

En plus de passer tout les test fourni avec le TP, nous avons construit des test afin de test les ca classique de chaque fonction. Ceux ci commence tous par perso_test.

VII/ Compilation

Il suffit de faire make dans le premier dossier pour compiler le code. Il est ensuite possible de lancer les tests dans le fichier tests ou bien perf pour visionner les performance des stratégie.