Florian Barrois
Thomas Duverney

Dossier Allocateur

1 – Les choix d'implémentation

Cette partie décrit nos choix d'implémentation sur la réalisation de l'allocateur mémoire. Notre algorithme de gestion de la mémoire repose sur le principe de chaînage des zones libres. Pour la représentation des zones libres, nous avons gardé la structure fb fournie dans le code initial. Pour pouvoir repérer le début de notre liste chainée, nous avons placé un pointeur (char *), pointant sur la première zone libre de la mémoire. Ce pointeur sert de point d'entrée pour parcourir la chaîne de zones libres, il faut donc veiller à ne pas l'écraser au moment des allocations et libérations.

Pour ce qui est de la représentation des zones occupées, nous avons opté pour une structure appellée « bb_t », elle contient une variable « size » qui représente la taille de la zone occupée. Cette structure est placée en début de mémoire allouée, elle servira pour la suite de l'algorithme notamment lors de la libération.

De plus, nous avons décidé qu'il était possible d'avoir des structures de zone libre qui possèdent une taille de zéro octet puisque l'utilisateur peut tout de même allouer 8 octets du fait que la structure d'un bloc occupé est plus petite de 8 octets que la structure de zone libre.

Dans l'affichage de la taille des zones libres lors de l'exécution du programme, le nombre indiqué à l'utilisateur est la taille maximum qu'il puisse allouer, en prenant en compte la place occupée par les structures.

Exemple : Si le programme affiche un espace vide de taille 50, l'utilisateur peut effectivement allouer 50 octets et non 42 (50 – taille de structure bb_t).

2 – Spécification des fonctions

La fonction « find_prev_free_block » permet de retrouver la dernière zone libre située avant une adresse donnée en paramètre. Cette fonction va donc parcourir la liste chainée de zones libres et retourner l'adresse de la zone libre précédant l'adresse ptrBlock si elle existe, sinon elle retourne NULL.

fb_t find_prev_free_block(char* ptrBlock);*

La fonction «is_on_busy_struct » permet de vérifier si le pointeur passé en paramètre, pointe sur une structure de zone occupée, si c'est le cas elle retourne 1, sinon 0. Cette fonction est utilisée au moment de la libération de mémoire pour vérifier que l'utilisateur

n’essaye pas de libérer en millieu de zone occupée ou à des adresses ne correspondant pas à une zone allouée.

*int is_on_busy_struct(char * ptrZoneToFree);*

3 – Les fonctionnalités et les limites

Toutes les fonctionnalités demandées initialement pour l’allocateur ont été impléméntées. Par contre nous n’avons pas implémenté toutes les fonctions de recherche de zone libre, uniquement la fonction « mem _fit_first » a été réalisée. Les autres n’ont pas été implémentées car nous voulions améliorer le plus possible nos fonctions d’allocation et de recherche.

Pour les fonctionnalités concernant l’allocation, nous avons géré plusieurs choses. Notamement le bourrage des zones occupées, lorsque le futur bloc occupé est inférieur à la zone libre choisie par la fonction « mem_fist_first ». On vérifie que la différence de la taille à allouer et de la taille de la zone libre choisie ne soit pas inférieure à la taille d’une structure fb, en effet si c’est le cas on bourre la zone occupée pour éviter de se retrouver plus rapidement avec de la mémoire fragmentée.

De plus, l’utilisateur est bloqué lorsqu’il veut allouer une taille de mémoire plus grande que la mémoire elle-même. Et lorsqu’il alloue une taille de mémoire trop petite, c’est-à-dire inférieure à la taille d’une structure de zone libre, nous avons décidé de changer la valeur de la taille qu’il souhaite allouer à une valeur égale au minimum à la taille d’une structure bb. Cette fonctionnalité permet d’éviter la fragmentation mémoire.

Pour finir, il y a plusieurs limites concernant notre programme. En effet, nous n’avons mis en place aucune sécurité concernant la gestion de la mémoire, par exemple une sécurité concernant la corruption de la mémoire. Nous n’avons pas non plus géré l’alignement des zones mémoires sur des puissances de 2.

4 – Les tests

Détails importants les fonctions de test réalisées :

- test_alloc_memory_full() : la mémoire est soit complètement pleine, soit morcelée mais sans bloc vide de taille assez grande pour allouer.

- test_alloc_requiring_padding() : vérifie qu’il ne reste pas zone libre de taille inférieure à la taille d’une structure fb après une allocation.

- test_free_address_inside_free_block() : cas de libération de mémoire sur une adresse située dans un bloc libre. Même comportement lorsque l’adresse correspond à une structure de bloc libre.