TP : Allocateur mémoire

El Yandouzi Elias Salmon Amad

Polytech Grenoble - INFO3 - API

Objectif Nous allons dans ce TP chercher à réaliser un allocateur mémoire. Nous l'implémenterons en utilisant la méthode *first_fit* pour trouver les zones libres. Toute la difficulté de ce TP réside dans la compréhension des pointeurs, de leurs usages et de leurs arithmétiques.

1 Choix d'implémentation

1.1 Représentation de la mémoire

D'un point de vue architecturale, nous considérerons les zones de mémoires comme des blocs contenant en en-tête de la méta donnée suivie du contenu alloué. De ce fait trois structures viennent en tête assez rapidement :

- *first_bloc*, comme son nom l'indique est le premier bloc de métadonnées. Il permet de structurer l'ensemble de la mémoire et donc doit nécessairement être bien implémenté. Un cast sur une structure *first_bloc* donnera accès à la taille globale de la mémoire (*sizeG*) ainsi qu'à l'adresse du prochaine bloc libre (*pbloc_free*).

- *free_bloc*, avec une structure *pfree_bloc* qui n'est autre qu'un pointeur sur une structure du même nom, permet de connaître l'ensemble de blocs libres avec leurs tailles respective. Tous les blocs libres sont liées grâce à chaînage.

- *bloc_used* est la denière structure pour l'organisation de notre mémoire. Elle indique les blocs qui sont utilisés en précisant seulement leur taille. Aucun chaînage n'est present entre bloc du même type. Toute fois, avec de simple calcul, il est facile de retrouver un bloc occupé.

1.2 Allocation de la mémoire

Pour l'allocation de mémoire, le seul paramètre demandé à l'utilisateur est la taille de la mémoire à allouée. Une fois que la taille est connue, le programme cherche une zone libre assez graour contenir un bloc de la taille demandée avec ses métadonnées. Avant de lancer la recherche on effectue aussi un alignement sur 16 octets pour faciliter l'accès au processeur au zone mémoire et donc avoir un gain de temps et de performance non négligeable.
Ici, la recherche d'espace libre se fait suivant le principe de *mem_fit_first*, soit au premier bloc rencontré suffisamment grand. Si aucun bloc ne peut accueillir l'allocation, alors **NULL** est renvoyé avec un message précisant l'échec de l'allocation.

1.3 Libération de la mémoire

Le processus de libération de mémoire (a.k.a *free* dans la libc) demande à l'utilisateur l'adresse de la zone mémoire à libérer. De là, plusieurs cas se distingue et nous n'en citerons qu'une partie dans ce compte rendu :

- *ZO* **ZO** *ZO*, pour libérer la partie en gras, il faut trouver l'élément libre qui le précède ainsi que celle qui la suit. Il faudra donc faire pointer la précédente zone sur cette **ZO** et donner pour suivant à celle-ci, celle qui la suivait.

- *ZL* **ZO** *ZO*, ce cas de figure demande le moins de travail, la *ZL* qui la précède doit juste augmenter sa taille, aucun travail n'est requis sur les pointeurs de la liste chaîné.

- ... (bien évidemment tout les cas sont traités dans le fichier *mem.c*)

1.4 Affichage de la mémoire

Afin d'afficher la mémoire dans son entièreté, le programme parcourt toute la zone mémoire de bloc en bloc en partant de l'adresse de départ. Ensuite, des calculs utilisant l'arithmétique de pointeurs permettent de déterminer si le bloc courant est libre ou occupé.
Selon la nature du bloc, la méthode *mem_show* affiche les données appropriées.

2 Tests

Nous avons fourni dans le fichier test.c, un programme qui permet d'effectuer une batterie de tests d'allocations de mémoire couvrant plusieurs points anguleux possibles, notamment ceux pouvant apparaître lors de la libération de mémoire. Ce programme teste ces cas-ci :

- "ZO ZL $\rightarrow$ ZL",
- "ZL ZO $\rightarrow$ ZL"
- "ZL ZO ZL $\rightarrow$ ZL",

Ainsi qu'une mémoire pleine complètement et que l'on veut *free*.
Pour réaliser cette batterie de tests, il suffit d'exécuter *./test* dans un terminal après avoir executé *make*.