TP Système – Allocateur de mémoire

Thomas MEDARD
Antoine MERCIER-PRONCHERY

Introduction

Ce TP nous a permis de mieux nous familiariser avec le fonctionnement d'une mémoire interne. Nous avons mis au point un allocateur de mémoire robuste (selon nos tests) capable de tenir sur la durée sans rencontrer de bug bloquant.

Généralité et allocation de la mémoire

➔ Initialisation du programme

L'initialisation de l'allocateur va initialiser toutes les variables globales avec les informations sur la mémoire qui lui sont fournis.

➔ Demande d'allocation

Lors de l'allocation de la mémoire et afin d'imiter les fonctions du vrai Malloc, si la taille demandée à allouer est de zéro, notre programme retournera NULL à l'utilisateur. De même, si aucun bloc mémoire à été trouvé, le pointeur retourné vaudra NULL.

Si la taille est supérieur à 0 notre programme va tenter de trouver un bloc disponible/ La taille sera arrondi pour faciliter le traitement matériel. Si on ne trouve pas de bloc avec cette taille, on refait une recherche avec la taille non arrondie.

Si toute la mémoire est occupée, alors la tête de liste vaudra NULL.

➔ Libération mémoire

Lors de la libération de la mémoire par l'utilisateur, notre programme va être capable de chaîner correctement, si nécessaire, la liste des blocs libres.

Il y a plusieurs cas particuliers que nous avons pris en compte dont la liste non exhaustive pourrait contenir :

- Libération d'une zone mémoire pleine
- Libération d'une zone accolée à une zone libre (avant et/ou après).

Libération des espaces mémoire

Concernant la libération des espaces mémoire préalablement alloué, nous avons couvert tous les cas possibles que nous avons trouvé afin de recomposer des blocs vides de manière valide.

En effet en fonction de quel espace mémoire est libérée et de son emplacement certaines précautions doivent être prise afin d'assurer l'intégrité de la mémoire.

Ainsi notre algorithme peut correctement s'occuper de différents cas de libération de mémoire :

- Un bloc mémoire libéré entre deux blocs libres
- Un bloc mémoire libéré après un bloc libre mais juste avant un bloc alloué (aucun espace entre le bloc libéré et le prochain bloc)
- Un bloc mémoire libéré se situe après un bloc occupé et avant un bloc libre

Pour tous ces cas, des exceptions supplémentaires sont possible :

- Un bloc mémoire libéré est le seul bloc disponible de la mémoire
- Un bloc mémoire libéré se situe avant tous les autres blocs libres

Il nous a donc fallut réfléchir à ces différentes possibilités afin d'obtenir une méthode la libération de mémoire alloué la plus flexible possible.

Pour chaque cas, lors de la libération, nous rechercons au préalable le bloc précédent et le bloc suivant afin d'avoir toutes les informations pour effectuer les diverses opérations sur les pointeurs.

Parcours et recherche de la mémoire

Notre fonction de parcours mémoire se contente d'itérer à travers notre structure.

Enfin, nous montrons à l'utilisateur la taille totale stocké dans la mémoire à la fin des opérations.

Batterie de test

Test d'endurance

Ce test consiste à simuler la vie de l'allocateur mémoire dans un programme utilisateur.

Pour ça, le test va, de manière aléatoire, soit éffectuer une allocation, soit éffectuer une libération. Le choix est influencé par le nombre d'allocation déjà effectuées (plus on fait d'allocation et plus on a de chance de réaliser une libération).

Les tailles allouées sont aussi aléatoires.

Le nombre d'allocation à éffectuer est fixe.

Ce test est utile pour repérer des bugs non spécifiques qui peuvent arriver durant une longue période d'activité du programme. En revanche, nous avons du tester nous même les cas particuliers.

Test Initial

Ceci est le test fournis. Ce test tente d'initaliser la mémoire et d'allouer la plus grande zone possible. Ce test est réaliser plusieurs fois en réinitialisant la mémoire à chaque fois.
Ce test peut être utile pour simplement tester l'initialisation.

Commentaires

Nous n'avons pas réalisé de module supplémentaire, nous n'avons donc que la fonction fit_first, trouvant le premier bloc libre trouvé lors de la recherche.

La fonction fit_best aurait été intéressante car elle aurait permis de trouver, pour chaque allocation, la place la plus approprié dans la mémoire, réduisant ainsi les espaces de mémoires inoccupés, trop petit pour accueillir une nouvelle donnée (la fragmentation).

De même, la fonction worst_fit aurait aussi pu limiter cette fragmentation.