Pratique du C
Projet

Ma bibliothèque d'allocation mémoire

Auteur :
Djebien Tarik
Groupe : **2**
Date: **Décembre 2010**

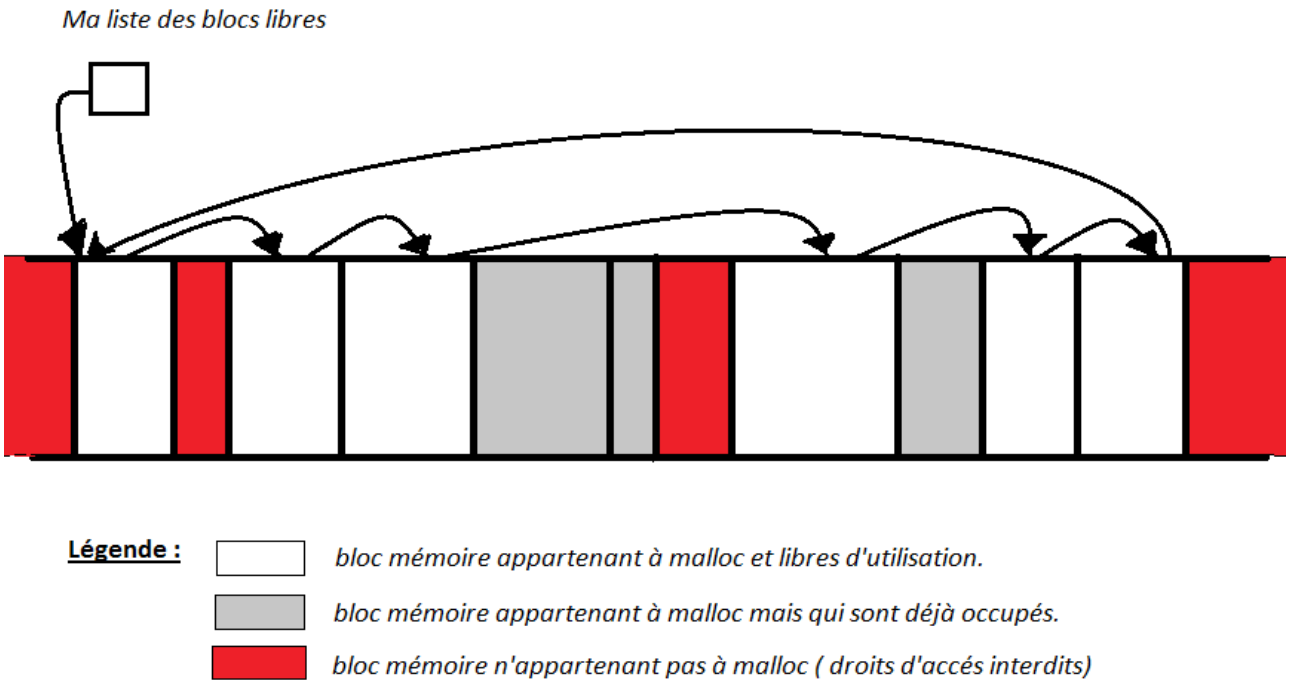
Modélisation du projet

Introduction:

Nous avons vu précédemment dans **BAM_Simple.c** que notre système d'allocation mémoire implémenté avec un tableau qui fonctionner comme une FIFO restait relativement très limité dans le contexte d'une utilisation réelle dans la pratique lors d'une utilisation dans un programme en langage C.
Il est donc nécessaire d'implémenter une nouvelle version avec moins de restrictions et plus d'optimisation que **BAM_Simple.c**

Cahier des charges du projet :

Premièrement, l'ordre des appels aux fonctions ***malloc*** et ***free*** pourra se dérouler dans un ordre quelconque.
Deuxièmement, au lieu d'allouer un tableau de taille fixée lors de la compilation, ***malloc*** demandera de l'espace mémoire au système d'exploitation en cas de nécessité supplémentaire.
De plus, puisque d'autres fonctions du programme pourront également demander de l'espace sans passer par la gestion dynamique de mémoire, on devra implémenter une structure de donnée qui fera en sorte que l'espace géré par ***malloc*** ne soit pas forcément contigu.
On conservera donc un espace libre sous la forme d'une liste de blocs libres. Chaque bloc contient une taille, un pointeur sur le prochain bloc libre ainsi que l'espace disponible proprement dit. Pour respecter le sujet, on conserve les blocs dans l'ordre croissant des adresses en mémoire, et le dernier bloc pointe sur le premier.(Liste circulaire avec pour dernier élément le bloc correspondant à la plus grande adresse mémoire).



Si une demande est effectuée au système, il suffira alors de parcourir la liste des blocs libres jusqu'à atteindre un bloc d'une taille suffisamment grande.

Cet algorithme est appelé «first fit»:
on choisit le premier bloc satisfaisant le critère de taille qui nous tombe sous la main, pas le meilleur, juste le premier parmi tous les choix possibles.

- Si le bloc trouvé possède exactement la taille demander, on le retire de la liste et on le retourne à l'utilisateur.
- Si le bloc est trop grand, on le coupe et on retourne à l'utilisateur un des blocs divisés ayant la taille adéquate, et on ajoute le reste du bloc initial dans la liste des blocs libres.
- Dernière possibilité, si on ne trouve aucun bloc de taille suffisante correspondant à la demande, on n'a pas le choix, la seule solution est de demander au système d'exploitation un blocs de mémoire plus gros en l'ajoutant dans la liste des blocs libres.

La libération d'un espace implique également un parcours de la liste des blocs libres afin de rechercher et trouver la position où il faut insérer le bloc désalloué parmi ceux déjà libres. Si le bloc libéré est adjacent à un bloc libre droit ou gauche, on les fusionne pour former un bloc de taille plus grande. Ceci permet d'optimiser la mémoire pour y accéder plus rapidement. (contrainte du sujet: principe de fragmentation de la mémoire).
La relation d'ordre sur les blocs établie précédemment nous permet de déterminer si deux blocs sont adjacents ou non.

Pour reprendre les même notation que dans les cours d'Algorithmique et Programmation impérative, nous allons appeler «en tête» toutes les informations se trouvant dans chacune des cellules de la liste. Pour faciliter l'alignement du tas mémoire, on fera en sorte que les blocs ont une taille qui soit un multiple de la taille d'un en tête, celui ci sera aligné convenablement dans le tas.

Implémentation du projet

- Implémentation réalisés:
  malloc et **free**
- Voir fichier source :
  bam.c