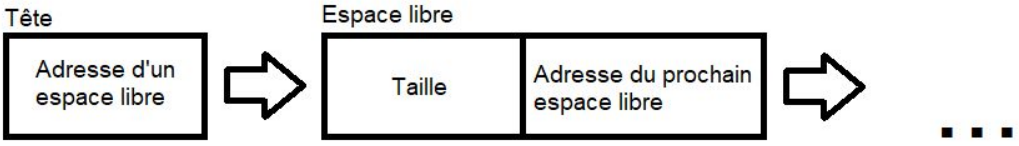
CPS - Allocateur Mémoire

Compte Rendu

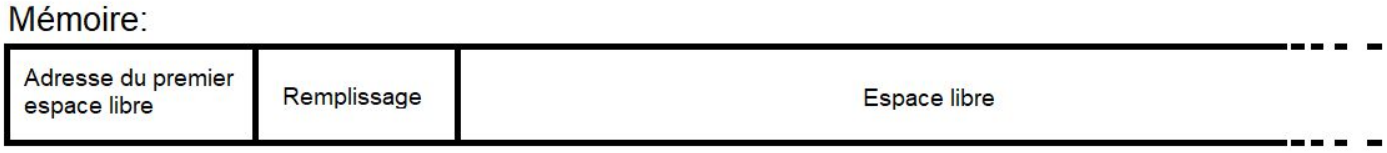
Espaces libres:

Les espaces libres sont gérés comme une liste chaînée par notre allocateur. Nous disposons d'une tête qui pointe sur le premier espace libre (ou NULL si il n'y en a pas). Les espaces libres sont classés de part leurs adresses dans la mémoire par ordre croissant.



Fonction mem_init:

Dans mem_init, on initialise un espace mémoire d'une taille passé en argument. L'initialisation de l'espace mémoire permet de créer le premier espace libre ainsi que de placer au début de l'espace mémoire une adresse sur le premier espace libre (la tête de notre liste chaînée). Donc, à l'initialisation le premier espace libre se trouvera juste après son adresse (techniquement ils ne seront pas forcément côte à côte car il faut prendre en compte l'alignement). On initialise aussi ici quelle politique sera utilisée pour choisir les espaces libres (on a choisi une politique que nous avons imaginé mem_fit_balanced).



Fonction mem_alloc:

La fonction mem_alloc permet d'allouer un espace mémoire d'une taille donnée et renvoie l'adresse de l'espace alloué (ou NULL si l'allocation est impossible) en fonction de notre politique de sélection d'espace libre. On calcul d'abord la taille réelle demandée en prenant en compte l'alignement ainsi qu'un espace pour la taille de l'espace alloué. On récupère ensuite le premier espace libre (dont l'adresse est au début de notre mémoire). Ensuite grâce à notre politique de sélection d'espace libre on obtient (ou non) un espace libre pouvant contenir notre espace mémoire. On sépare ensuite notre programme en deux. Si l'espace libre trouvé est plus gros que la taille demandée, alors on doit allouer notre espace et créer un nouvel espace libre à la suite de celui-ci (on doit aussi supprimer l'ancien espace libre de notre liste d'espace, et aussi rajouter le nouvel espace libre créé à la liste). Sinon, l'espace libre fait la taille demandée et on le supprime simplement en pensant à relier les espaces mémoires qui le liait entre eux. Ensuite,

on ajoute en entête de notre espace mémoire la taille de celui-ci. On renvoie l'adresse situé après la taille du bloc mémoire, que pourra utiliser l'utilisateur.

Fonction mem_free:

Cette fonction libère un bloc mémoire précédemment alloué. On sait que la taille du bloc alloué se trouve pile avant l'adresse du bloc envoyé à l'utilisateur. On peut donc récupérer facilement la taille du bloc. Une fois qu'on a la taille du bloc, on récupère l'espace libre pile avant ce bloc mémoire (si il existe), ainsi que l'espace libre pile après (si il existe). On regarde tout d'abord si l'espace libre avant notre bloc mémoire ne nous colle pas, si il nous colle alors on doit augmenter sa taille pour qu'il contienne maintenant aussi cette espace mémoire. Si le bloc mémoire n'est pas collé avec l'espace libre précédent alors on crée juste un nouvel espace libre. On regarde ensuite si l'espace après notre espace mémoire est collé à celui ci, si c'est la cas alors on le supprime et augmente la taille de l'espace libre précédent.

Fonction mem_show:

La fonction mem_show, nous a fait changer l'implémentation que nous avions utilisé pour notre allocateur mémoire. Au départ nous classions les espaces libres par ordre croissant de taille, mais pour afficher la mémoire dans le bon ordre avec les espaces mémoires utilisé et libre le plus simple était d'avoir les espaces libre classés en fonction de leurs adresses.
Pour afficher la mémoire, notre méthode était de commencer par l'emplacement mémoire situé pile après notre entête à la mémoire, puis de le comparer avec le premier espace libre pour savoir si s'en est un. Comme les tailles sont toujours placées en début d'espace mémoire (pour un espace libre ou occupé), nous avançons simplement de la taille trouvé et si c'était un espace libre on avance aussi dans notre liste chaînée.

Fonction mem_realloc:

Dans mem_alloc, nous commençons par chercher si il y a un espace libre collé à l'espace que l'on veut réallouer. Si il y en a un, on regarde si en l'ajoutant à la taille que l'on a déjà il pourrait contenir la nouvelle taille donnée. Si oui on le supprime de la liste chaînée des espaces libres. Si non, on doit trouver un emplacement mémoire pouvant contenir notre nouvelle espace mémoire. Une fois celui ci trouvé, on copie toutes les données se trouvant dans le premier espace mémoire. Puis nous devons désallouer notre ancien espace occupé, on utilise ici simplement mem_free.

Fonction mem_fit_balanced:

Nous avons commencé par implémenter mem_fit_first, puis mem_fit_worst et mem_fit_best. Nous avons cherché à faire un algorithme qui serait plus optimisé pour choisir un bon emplacement mémoire. Nous avons donc créé mem_fit_balanced qui est un mélange de mem_fit_worst et mem_fit_best. On cherche le meilleur et pire emplacement et si la taille demandée nous semble être une petite taille ne contenant qu'une valeur alors on la place au meilleur emplacement. Sinon, c'est peut être un tableau qui pourra éventuellement grandir et on le place au pire emplacement.