Système d'Exploitation - L3, S6
TP 1 - Mémoire

Les questions avec * sont facultatives, mais nous vous encourageons vivement à les faire en dehors des heures de TP.

Exercice 1.1 *Echauffement*
Téléchargez et désarchivez le fichier **memoire.tar.gz**.

1. Lisez le module **memoire** et expliquez en quoi il permet de pister les fuites mémoire.
2. Permet-t-il de réparer les fuites mémoire ?

Exercice 1.2 *Gestion mémoire simple*
L'objectif c'est d'implémenter un mécanisme d'allocation/libération de mémoire.

On va supposer que la mémoire est divisée en mots consécutifs, de même taille t (en octets) et qu'il y en a b (numéroté de 1 à b). Allouer de la mémoire de taille l, c'est fournir $\lceil \frac{l}{t} \rceil$ mots consécutifs. Pour gérer la mémoire, on va la diviser en blocs, chaque bloc étant représenté par une adresse de début (un numéro de mot), une taille (en mots) et un drapeau indiquant si le bloc est utilisé ou pas. Les blocs sont chaînés dans une liste.

1. Téléchargez le module memoire2 et écrivez la fonction creerMemoire qui permet de créer une mémoire avec un seul bloc libre composé de tous les mots.
2. Ecrivez la fonction `allocationMemoire` qui alloue de la mémoire de taille demandée et qui retourne l'adresse du bloc alloué. Il faudra gérer les cas où il n'y a pas de plage mémoire consécutive de taille demandée.
3. Ecrivez la fonction `liberationMemoire` qui libère la mémoire allouée. Il faudra gérer les cas où l'adresse donnée n'est pas celle d'un bloc alloué. Pensez à regrouper les blocs libres.
4. * Implémentez un algorithme de défragmentation de mémoire qui consiste à déplacer tous les blocs utilisés vers le début (ou fin) de mémoire.

Exercice 1.3 *Pagination*

L'objectif ici est de proposer un module qui gère la mémoire virtuelle. On rappelle que dans le mécanisme de mémoire virtuelle, chaque processus a son espace mémoire divisé en pages et à chaque instant, pour qu'une page virtuelle soit utilisée il faut qu'elle soit en correspondance avec une page physique. Pour faire simple, on va considérer qu'un programme a une taille et on calcule sa table des pages à partir de cette taille. On supposera par défaut des pages physiques de taille $4Ko$ et une mémoire de taille $64Ko$, mais ces deux valeurs doivent être des paramètres du module, que l'on prend lors de l'exécution si elles sont fournies). Une page virtuelle sera identifiée par son numéro, le numéro de page physique (ou -1 si pas en mémoire) et deux drapeaux signalant si la page a été lue et modifiée.

L'objectif maintenant c'est de créer le module **page**.

1. Créez les fichiers .h et .c du module **page**. Réfléchissez d'abord à la structure d'une page.
2. Écrivez une fonction qui prend un entier n et qui crée les pages virtuelles d'un programme de taille n.
3. Écrivez une fonction qui permet d'affecter une page physique à une page virtuelle. Lorsqu'il n'y a pas de page physique libre, vous devez utiliser un algorithme de remplacement d'une page pour choisir une page virtuelle à éjecter. Pour le remplacement d'une page virtuelle, vous implémenterez l'algorithme modifié FIFO (qui commencera par regarder parmi les pages non accédées, sinon par celles non modifiées).
4. Une adresse virtuelle sera alors composée d'un numéro de page virtuelle et du décalage dans la page physique. Écrivez la fonction qui permet de transformer une adresse virtuelle en adresse physique. Lorsque la page virtuelle n'est pas en correspondance avec une page physique, il y a défaut de page et vous devez dans ce cas faire d'abord appel à la fonction de chargement de page, avant de faire la conversion.
5. Testez votre implémentation. Écrivez d'abord une structure **instruction** qui contient une adresse virtuelle et le type d'instruction (lecture ou écriture), et un programme sera un tableau d'instructions. Pour créer un programme, vous pouvez choisir une taille aléatoire, et remplir les instructions aléatoirement. L'exécution du programme sera juste une boucle où on exécutera une à une les instructions.

   Pistez dans le test les défauts de pages et expliquez. Vous pouvez par exemple modifier votre algorithme pour afficher 1 message à chaque défaut de page.
6. Implémentez l'algorithme de remplacement de pages NFU (Not Frequently Used). Cet algorithme consiste à mettre dans chaque page virtuelle un compteur **age** de 8 bits et à l'incrémenter à chaque fois qu'une page a été accédée. Pour incrémenter un compteur : on commence d'abord par tout décaler vers la droite et ensuite on remplace le bit de gauche (qui est le bit de poids fort) par le bit d'accés. Dans l'algorithme NFU on remplace toujours la page ayant la valeur **age** la plus petite, s'il y a égalité on applique FIFO.

   (a) Ecrivez la fonction qui permet d'incrémenter le compteur **age** d'1 page.

   (b) Ecrivez une fonction qui permet de remplacer une page en utilisant l'algorithme NFU.

   (c) Relancez votre fonction de test avec l'algorithme NFU à la place du FIFO. Comparez les deux implémentations. On considérera que chaque tour de boucle est un tick de l'horloge et la modification de la variable **age** de toutes les pages est faite à chaque tick de l'horloge.
7. * Modifiez votre implémentation pour prendre en compte la pagination à 2 niveaux.