TD3 – Gestion de la mémoire

Objectifs

- Comprendre l'organisation mémoire d'un processus.
- Manipuler l'**allocation dynamique** (malloc, free).
- Observer les **erreurs mémoire** (fuites, accès illégaux).
- Découvrir la **mémoire virtuelle** et les mécanismes de segmentation / pagination.
- Introduire **mmap** pour le mapping mémoire.

Rappel théorique

- Un processus a une mémoire découpée en zones :
  - **Code** (instructions),
  - **Données globales** (variables statiques),
  - **Tas (heap)** → mémoire dynamique (malloc/free),
  - **Pile (stack)** → variables locales, appels de fonctions.
- L'OS fournit une **mémoire virtuelle** : chaque processus croit avoir son propre espace mémoire (protégé par la **MMU**).
- La mémoire est une ressource critique → risques :
  - **Fuite mémoire** : oublier de libérer (malloc sans free).
  - **Segfault** : accès à une zone non autorisée.

1. Exercices

Exercise 1 – Allocation simple

Écrire un programme qui :

1. Alloue dynamiquement un tableau de 10 entiers avec malloc.
2. Remplit le tableau avec des valeurs.
3. Affiche son contenu.
4. Libère la mémoire avec free.

Question : que se passe-t-il si on oublie le free() ?

Exercice 2 – Fuite mémoire volontaire

1. Écrire une fonction qui fait un malloc sans jamais free.
2. Appeler cette fonction dans une boucle 1000 fois.

3. Exécuter le programme avec **valgrind** :

```
valgrind ./a.out
```

Observer les messages indiquant la fuite.

Exercice 3 – Segmentation fault

1. Créer un pointeur non initialisé :

```
int *p;
*p = 42;
```

2. Exécuter le programme → crash attendu.
3. Observer le message **Segmentation fault (core dumped)**.

Discussion : pourquoi l'OS interdit d'écrire à cet endroit ?

Exercice 4 – mmap

1. Créer un fichier texte data.txt avec du contenu.
2. Écrire un programme qui utilise mmap pour **mapper le fichier en mémoire**.
3. Lire et afficher le contenu directement depuis la mémoire mappée.

Question : quelle différence avec read() ?

Exercice 5 – Pagination

1. Utiliser getpagesize() pour afficher la taille des pages mémoire.
2. Allouer un grand tableau et observer la différence entre **taille allouée** (via sizeof) et **taille réellement occupée en RAM** (via top ou ps).
3. Discuter du rôle du **lazy allocation** (l'OS n'alloue réellement qu'à la première utilisation).

3. Outils d'observation

- **valgrind** → détecter fuites mémoire, invalid writes, use-after-free.
- **gdb** → traquer un segfault (commande bt).
- **pmap** → afficher la carte mémoire d'un processus.
- **/proc//maps** → fichiers mappés et zones mémoire.
- **top / htop** → surveiller l'usage mémoire en temps réel.

4. Points de discussion

- Pourquoi l'OS empêche-t-il un processus d'accéder à la mémoire d'un autre ?
- Quelle est la différence entre pile et tas ?
- Pourquoi mmap peut être plus efficace que read ?

- Que se passe-t-il si plusieurs processus mappent le même fichier en mémoire ?