Système d'exploitation – TD3
Mémoire virtuelle (en salle machine)

1 Organisation de l'espace virtuel des processus

1.a) Déduire de `/proc/self/maps` comment les différentes régions mémoire des processus sont organisées en mémoire virtuelle sous Linux. Pourquoi la pile et le tas sont-ils placés ainsi ?

1.b) Où est située la mémoire du noyau ? Pourquoi fait-elle partie de l'espace virtuel utilisateur du processus alors qu'il n'y a pas accès ?

1.c) Quelles variations observe-t-on d'un processus à l'autre, même s'ils sont lancés exactement de la même façon ? Pourquoi est-il utile de faire varier le mapping en espace utilisateur ? Pourquoi est-il utile de ne pas faire varier le mapping noyau ?

1.d) Faire quelques `mmap` privés et partagés et retrouver les entrées correspondantes dans `/proc/self/maps`.

1.e) Ecrire un programme qui fait plein de petits `malloc` (quelques kilooctets ou dizaines de kilooctets) et expliquer l'évolution du (ou des) tas. Faire maintenant des gros `malloc` (de l'ordre du mégaoctet) et expliquer également. En déduire le comportement du système vis-à-vis d'un éventuel débordement du tas.

1.f) Ecrire un programme récursif qui va faire déborder la pile. Pour savoir à quel moment il a planté, on pourra lui faire afficher `/proc/self/maps` régulièrement. On pourra également essayer de rattrapper le signal SIGSEGV pour afficher l'adresse exacte où le débordement a eu lieu (cela nécessitera probablement `sigaltstack`). Pourquoi le système ne peut-il pas gérer les débordements de pile comme ceux du tas ?

1.g) Pour cette question, connectez-vous à une machine 64bits (pour avoir des résultats intéressants). Utilisez par exemple `uname -m` pour savoir si c'est le cas.

Déterminez la taille maximale que l'on peut projeter en mémoire virtuelle d'un seul coup avec `mmap`. On utilisera flags `MAP_ANONYMOUS|MAP_PRIVATE` et le descripteur `-1` pour obtenir une projection anonyme (sans fichier associé). Essayez à nouveau avec `MAP_NORESERVE` et expliquez.

Faites une boucle de gros `mmap` jusqu'à ce que le système refuse de projeter de nouvelles zones. Déduisez-en une approximation de la taille maximale de l'espace d'adressage du processus. Comparez cette taille à la notion d'architecture 64bits, en allant notamment jeter un oeil dans `/proc/cpuinfo`.

1.h) Essayez maintenant de mapper une page à l'adresse 0x1000 (seconde page virtuelle). Que se passe-t-il ? Pourquoi ? Essayez maintenant de mapper une page à l'adresse NULL. Comment faire et que se passe-t-il ? A quoi sert tout ceci ?

2 Manipulation de la mémoire virtuelle

2.a) Créez un fichier de 8ko. Ecrivez un programme qui projette (`mmap`) trois fois ce fichier dans un même processus de manière publique (`MAP_SHARED`). Ecrivez `1111` au début d'un mapping, puis `2222` dans second mapping (décalé de 4 octets), et observez le résultat depuis le troisième mapping.

Comme on n'a pas écrit de `\0` à la fin des chaînes, on ne peut pas afficher avec `%s` comme d'habitude. On implémentera une fonction affichant individuellement 8 caractères avec `%c`.

2.b) On utilise maintenant un mapping public et un mapping **privé** vers le même fichier. Écrivez `1111` au début du premier puis `2222` dans le second, et le contraire dans la deuxième page (`2222` décalé de 4096+4 dans le second, puis `1111` décalé de 4096 dans le premier). En réfléchissant à l'implémentation du copy-on-write, expliquez le phénomène.

Que se passerait-il si on écrivait à cheval sur les deux pages ? Comment pourrait-on utiliser ce phénomène pour mesurer la taille des pages ?

2.c) Modifiez votre programme pour ne créer qu'un seul mapping public de 1Mo (et aucun privé). Ajoutez un argument en ligne de commande pour terminer après une des étapes intermédiaires :

— avant le mmap,
— après le mmap,
— après une lecture d'un octet, d'une page, ou de tout le mapping,
— après une écriture d'un octet, d'une page, ou de tout le mapping,
— jusqu'à la fin du programme.

Utilisez `/usr/bin/time` pour mesurer les défauts de pages de votre processus selon l'étape à laquelle vous vous arrêtez. Expliquez les chiffres.

Système d'exploitation – TD3 (bis)
Mémoire virtuelle (en salle TD)

3 Tables de pages

On considère un système de mémoire virtuelle équipé d'une MMU. La capacité d'adressage du processeur est 32 bits. La taille des pages est 4ko.

3.a) De combien de pages est fait un espace virtuel ? Quelle est la taille des adresses virtuelles ?

3.b) Quelle est la taille de la table des pages si elle est représentée de façon linéaire et si chaque entrée contient le numéro d'un cadre physique sur 48 bits, des droits sur 3 bits, un bit VALID, un bit DIRTY et 11 bits AGE ?

3.c) Quelle information permet de savoir si un accès à une page est valide ? Comment savoir quelles pages risquent d'être beaucoup utilisées dans le futur ? Quelle information permet de savoir si une page est présente en mémoire ? Quelle est la taille des adresses physiques ? Qu'en pensez-vous ?

3.d) Est-il envisageable de stocker la table de pages dans la MMU ? Est-il envisageable de la stocker en mémoire physique ? Que se passe-t-il si on a 200 processus ?

On considère maintenant une table des pages à 3 niveaux. Les niveaux 1 et 2 contiennent des tables de 4ko remplies de pointeurs (32bits) vers les niveaux suivants. Le niveau 3 contient des tableaux de 4ko contenant les entrées de la table linéaire sus-citée.

3.e) Expliquez par un schéma comment traduire une adresse virtuelle en adresse physique en parcourant cette table. On expliquera notamment comment chacun des bits de l'adresse virtuelle est utilisé.

3.f) Comparez l'occupation de cette table à celle de la table linéaire sus-cité quand un processus n'utilise qu'1ko, 100ko, 10Mo, 1Go puis tout l'espace disponible. Quid d'une table couvrant 1000 fois 1ko dispersés dans tout l'espace virtuel ?

4 Etats des entrées de la table de pages

4.a) Décrivez l'état des bits de droits, VALID et DIRTY d'une entrée de la table de pages dans les cas suivants :

- Adresse invalide
- Adresse valide pointant vers page en lecture seule, présente en mémoire
- Adresse valide pointant vers page en écriture seule, présente en mémoire, et récemment modifiée
- Adresse valide pointant vers page en lecture-écriture, mais actuellement en copy-on-write
- Adresse valide pointant vers page absente en mémoire car swappée sur le disque

4.b) Dans quel cas le système va-t-il devoir rapatrier une page depuis le disque vers la mémoire physique ? Dans quel contexte l'opération est-elle effectuée ? Où sont stockées les informations nécessaires à localiser la page sur le disque ?

5 Translation Lookaside Buffer

Notre processeur est maintenant équipée d'un TLB mais pas de MMU.

5.a) De quelles informations le TLB a-t-il besoin pour travailler ? De quelles informations peut-il se passer ? Pourquoi va-t-on effectivement ne placer dans le TLB qu'une part des informations de gestion de la mémoire virtuelle ? Comparez cette situation à celle d'une MMU.
Que va-t-il se passer quand le TLB n'aura pas la traduction d'une adresse demandée ? Distinguer le cas où l'accès est valide, invalide, et copy-on-write. Là aussi, comparer avec le cas MMU.

5.b) De quelles instructions doit disposer le processeur pour manipuler le TLB ?

5.c) Réfléchir au cas d'un processeur multicoeur. Combien de TLB peut-on ou doit-on avoir ? Comment interagissent-ils ?