Mémoire virtuelle : Linux et la pagination

Ensimag 2A

7 décembre 2018

Le but de cet exercice est de travailler autour d'un certain nombre de mécanismes lié à la mémoire virtuelle et en particulier à la pagination.

Nous prenons ici l'exemple du noyau Linux. Il utilise le même modèle générique pour toutes les architectures, mais en l'adaptant à la réalité du matériel. Il y a une table par processus. Il travaille avec des tables de pages à quatre niveaux : PGT, PUD, PMD et PTE (exemple fig. 1). PUD est un ajout "récent", début 2005, dans versions 2.6.11. Le but était alors de coller aux 4 niveaux de l'architecture x86_64.

Il est à noter que certains niveaux ne correspondent pas à une réalité physique sur certaines architectures. Par exemple sur x86 (PC en 32 bits), les tables PUD et PMD ne contiennent qu'une seule entrée et disparaissent du code vraiment généré pendant la compilation.

De même, chaque type de processeur, ou presque, possède une TLB, c'est-à-dire une mémoire associative (un dictionnaire) qui mémorise les résultats des recherches récentes dans la table des pages. Le but est le même que celui des caches de données et d'instructions : ne pas avoir à faire de nombreux accès mémoire pour lire la table de pages en mémoire, à chaque instruction. Suivant les cas, cette TLB n'est pas forcément directement accessible (architecture x86) ou au contraire est le seul moyen de contrôle sur la traduction lorsque le processeur ne connaît pas la table des pages (architecture SPARC ou MIPS).

Chaque page en mémoire est rangée dans une *case*. Pour chaque case, une structure `struct page` contient l'ensemble des informations utiles sur la page. On dispose d'une fonction `allocFrame()`, qui renvoie une case disponible.

1 Le cas de l'architecture x86_64

le but de cette partie est de faire un peu attention à l'influence du matériel sur les capacités du système. La figure 1 et la table 1 donnent des détails sur l'implantation de la table des pages et sur l'utilisation des bits de l'adresse virtuelle pour la traduction.

Question 1 : Quelle est la taille de l'espace de mémoire virtuelle utilisable par un processus avec l'architecture x86 ? et x86_64 ?

Question 2 : On peut noter que dans la figure 1, les tables et les pages font toutes 4 Ko. Dans chaque table, chacune des 512 entrées utilise donc 8

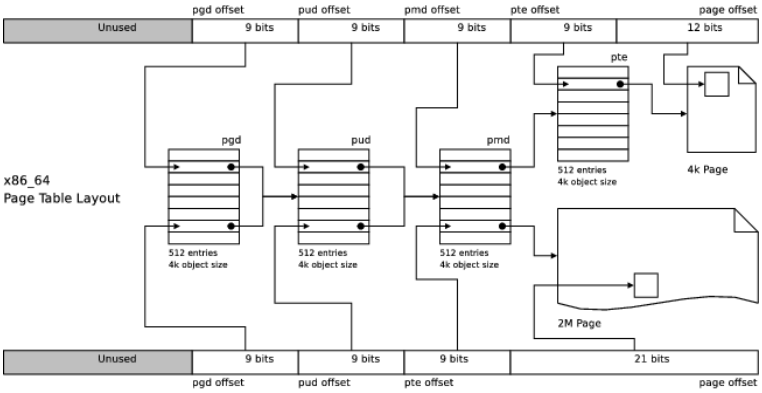

FIGURE 1 – Table des pages de l'architecture x86_64 (mode normal et mode "grande page" [http ://linux-mm.org/PageTableStructure])

Bits used	PGD	PUD	PMD	PTE
i386	22-31			12-21
x86-64	39-47	30-38	21-29	12-20

TABLE 1 – Table d'utilisation des bits en x86 et x86_64 (node normal) (Les bits utilisés sont numérotés à partir du bit 0)

octets. En admettant que les tables et les pages sont alignées en mémoire sur des multiples de 4Ko, combien de bits par entrée sont disponibles pour décrire les propriétés de la page (présence en mémoire, droits de lecture-écriture, etc.) ?

Question 3 : Quel est le coût de stockage de la table des pages d'un processus si elle est entièrement utilisée ?

Question 4 : Chaque processus possède un pointeur vers sa propre table des pages. Le registre du processeur pointant vers la table des pages est donc mis-à-jour dans le processeur à chaque changement de contexte. Que ce passe-t-il alors automatiquement pour la TLB ? Pourquoi ?

Question 5 : Pour chaque case de la mémoire physique, il faut stocker une *struct page*. Comment faire ce stockage et comment retrouver cette structure à partir d'une adresse virtuelle ?

1.1 Copy-on-Write

Lors d'un fork, il serait coûteux de vraiment copier toute la mémoire du processus appelant, surtout qu'au début elle est complètement identique pour le processus père et le processus fils. L'idée est donc de la copier uniquement lorsqu'elle sera modifiée.

Question 6 : Que faut-il comme information supplémentaire dans la structure gérant la page pour pouvoir implanter le copy-on-write ?

Question 7 : Indiquer succinctement comment vous implanter cette fonctionnalité en utilisant les protections possibles pour la page.

Question 8 : Proposer le ou les codes des traitants implantant cette partie. Vous pouvez définir de nouvelles fonctions de haut niveau pourvu que vous documentiez ce qu'elles font.