Mémoire

Questions principales

Décrivez en détail le principe et l'utilité de la segmentation. Comment ce principe est-il mis en oeuvre lors du fonctionnement en mode réel et protégé du processeur ? Le but de la segmentation est de séparer la mémoire en segments. Chaque segment est un espace d'adressage indépendant qui peut s'agrandir à l'exécution. Pour spécifier une adresse quand il y segmentation, il faut donner une adresse en deux parties, une partie donnant le numéro du segment et une autre donnant l'offset par rapport au début du segment. La segmentation apporte une nouvelle sécurité. Si une adresse dépasse l'offset d'un segment et qu'on essaye d'accéder à une zone mémoire hors du segment ou que le segment déborde sur un autre segment, une segmentation fault se produit.

La segmentation avec pagination sous Intel x86 En Intel x86, deux tables de descripteurs de segments sont présentes :

- La LDT (Local Descriptor Table) : chaque process possède une LDT, elle possède les descripteurs de segments des segments locaux au programme.
- La GDT (Global Descriptor Table) : Elle est unique au système, elle décrit les segments du système.

Pour accéder à un segment en x86, il faut d'abord charger un sélecteur de segment dans un registre de segment (CS, DS, SS, etc.). Pendant l'exécution d'un process, les registres de segments possèdent les sélecteurs de segments pour le process. Un sélecteur de segment fait 16 bits.

```
-------------------------------------------------------------------------
| 13 bits : index | 1 bit : 0 = GDT, 1 = LDT | 2 bits : protections (0 à 3) |
-------------------------------------------------------------------------
```

- L'index est l'index dans la table des segments.
- 1 bit, si celui-ci est 0, alors l'index est celui de la GDT, sinon c'est celui de la LDT.
- 2 bits de protections allant d'une valeur de 0 à 3.

Quand le sélecteur de segment est chargé, le descripteur correspondant dans une des deux tables est également lu et stocké dans un registre pour un accès rapide.

Un descripteur de segment consiste en 8 bytes. Le descripteur contient :

- L'adresse de base

- La limite (dernière adresse)
- Bit de granularité : si 0, alors la limite est en bytes. Si 1, la limite est en pages. On utilise ce bit car la limite est seulement sur 20 bytes.
- Un bit qui précise si le segment est en mémoire.
- Les bits de protections.
- et d'autres champs.

Grâce à ce descripteur, on peut donc accéder à une adresse linéaire (pas physique, car les segments sont également paginé, il faudra donc transformer cette adresse linaire en adresse physique grâce à la table des pages), cette adresse linéaire est l'adresse du segment et son offset.

Les segments pouvant être de très grande taille, il est donc nécessaire de les paginer. Chaque segment contient sa propre table des pages elle même paginée. Dans le descripteur de segment, si le segment est en mémoire, on considère donc que table des pages l'est aussi, une fois l'adresse linéaire trouvée, on va donc transformer l'adresse linéaire en adresse physique.

(La suite à vérifier) En mode réel, les adresses font 20 bits, on peut donc avoir seulement 1Mb de mémoire adressable (2^20). En mode réel on peut accéder à n'importe quelle zone mémoire. Le mode réel n'est presque plus utilisé de nos jours, sauf au démarrage, car il est trop dangereux qu'un process puisse avoir accès à n'importe quelle zone mémoire d'un autre process. L'adresse physique en mode réel est : segment * 16 + offset.

Décrivez en détail le principe et l'utilité de la pagination. L'espace d'adressage d'un process est découpé en pages de taille fixe. Seules les pages utiles sont chargées en mémoire à un moment donné pour éviter de trop charger la RAM. Une table en mémoire mémorise la présence et localisation des pages (table des pages). Un programme peut donc dépasser la taille physique de la mémoire. Cela permet aux programmes plus lourd de tourner sur des machines ayant moins de RAM.

Exemple :

- Page de 4Kb
- Adresses 16 bits 64Kb
- 32Kb en mémoire physique (8 cadres de pages).

```
MOV reg, [8192]
MOV reg, [20500]
MOV reg, [32780]
```

```
 cadre  page
-------------
```

1	4
2	9
3	5
4	3
5	7
6	2
7	1
8	6

Trouver les pages virtuelles : (le MMU s'occupe de la traduction)

- 8192 DIV 4Kb = 2
- 8192 MOD 4Kb = 0
- Page 2 Offset 0
- On regarde dans la table des pages : la page 2 se trouve dans le cadre 6 : 6 * 4Kb + 0 = 24576. L'adresse réelle est donc 24576.
- 20500 DIV 4Kb = 5
- 20500 MOD 4Kb = 20
- Page 5 Offset 20
- On regarde dans la table des pages : la pages 5 se trouve dans le cadre 3 : 3 * 4Kb + 20 = 12308. L'adresse réelle est donc 12308.
- 32780 DIV 4Kb = 8
- 32780 MOD 4Kb = 12
- Page 8 Offset 12
- La page 8 ne se trouve pas dans la table des pages : Page Fault Exception (interruption).
- L'OS doit résoudre le problème :
  - Choisir une page peu utilisée.

- Sauvegarder la page victime sur le disque si elle a été modifiée.
- Charger la page demandée à la place de la page victime.
- Mettre à jour la table des pages.
- Recommencer l'instruction interrompue.
- IP = adresse de MOV reg, [32780]
- Imaginons que la page remplacée est la 7, la page 8 se trouve donc dans le cadre 5. 5 * 4Kb + 12 = 20492.

Pour les techniques de changement de page, voir dernière question.

Questions secondaires

(segmentation) Comment la notion de ring est elle exploitée avec cette segmentation en mode protégée ?

(segmentation) Donnez un exemple d'instruction qui peut provoquer une erreur de segmentation

(segmentation) Comment ne pas perdre trop de temps en raison des plus nombreux accès à la mémoire ?

(pagination) Détaillez comment le processeur 32 bits a implémenté cette pagination.

(pagination) Quel est le rôle des interruptions dans cette pagination ? Page Fault Exception -> page non trouvée. Cette interruption est envoyé par le MMU lorsqu'il ne trouve pas la page.

(pagination) Comment ne pas perdre trop de temps en raison des plus nombreux accès à la mémoire ?

(pagination) Comment ne pas perdre trop de place mémoire en raison de la présence de la table des pages ? La solution est de paginer la table des pages pour que toute la table des pages ne réside pas en mémoire en même temps.

(pagination) Que faire si la mémoire vive de l'ordinateur est saturée ? Le MMU vérifie qu'une page est en mémoire, une adresse de 16 bits est composée de 4 bits de poids fort qui correspondent au numéro de la page, et des 12 bits de poids faibles qui correspondent à l'offet. Les 12 bits à 1 = 4095, 4Kb = 4096. Donc les 12 bits MOD 4096 donne l'offset.

L'exemple ci-dessus prend en compte des pages de taille 4Kb, le nombre de bits spécifiant l'offset se calcule en fonction de la taille de la page.

Un mécanisme hardware permet donc de vérifier si une page est présente dans la table des pages.

Si une page n'est pas en mémoire, voici les solutions possibles :

- Au hasard : le système prend une page au hasard et l'élimine.
  - Très mauvaise technique car il faudra peut-être recharger la page très vite ensuite. Trop de défauts de page et donc d'interruptions.
- Not Recently Used (NRU) : Algorithme simple à performance acceptable : On associe deux bits à la page, le bit `R` qui est remis à 0 régulièrement et au chargement de la page et mis à 1 lorsqu'on fait appel à la page. Et le bit `M` qui est initialement à 0 et qui est mis à 1 si l'on modifie la page. Pourquoi remettre `R` à 0 ? Car on pourrait avoir une page "utilisée" mais il y a longtemps.
  - Lorsqu'il faut supprimer une page, le système regarde les bits `RM` et cherche à supprimer celle où la valeur est la plus petite, `R` étant le bit de poids fort et `M` le bit de poids faible.
  - L'inconvénient est qu'on pourrait supprimer une page souvent utilisée juste après que son bit `R` ait été remis à 0.
  - Ces bits sont gérés au niveau hardware car le système n'a pas forcément la main.
  - Si `M` est à 1 lors de la suppression, la page est sauvegardée sur le disque.
- First In First Out (FIFO) : Algorithme simple à performances médiocres. Au chargement de la page, on lui associe l'heure courante. Lorsqu'il faut retirer une page, on choisit la plus ancienne.
  - Inconvénient majeur : suppression de table importante.
- First Int First Out amélioré : En plus de l'heure de chargement, on ajoute un bit `R` pour l'utilisation récente. Si la page la plus ancienne a été récemment utilisée (bit `R` = 1), alors on remet son bit `R` à 0 et on met à jour son heure à l'heure courante pour devenir la page "la plus récente". On recherche la page la plus ancienne, etc.

- Least Recently Used (LRU) : Le processeur possède un compteur qui est incrémenté à chaque instruction. À chaque fois qu'une page est utilisée, on lui associe la valeur de ce compteur. La page qui a la valeur la plus petite est celle qui est retirée de la mémoire.
  - Algorithme couteux en temps.
- Least Recently Used software : En plus du LRU, un bit `R` est associé à la page. À chaque fois que l'on remet `R` à zéro, on mémorise le bit `R` dans une mémoire associée à la page grâce à l'instruction `shr` (Shift Right, décalage de bit vers la droite).
  - Cette technique ne permet pas de savoir si une page a été beaucoup ou peu utilisée, on sait juste qu'elle a été utilisée au moins une fois.
- LRU avec matrice de bits : On utilise une matrice `N * N` avec `N` = nombre de pages. Lorsqu'on référence une page `i`, on met la ligne `i` à 1, la colonne `i` à zéro. Lors du changement de page, la page dont la ligne correspond à la plus petite valeur binaire est supprimée.

Il existe également un démon de pagination qui libère les pages peu utilisées à intervalles réguliers pour limiter les défauts de page.

brk() et sbrk()

```
#include <unistd.h>
int brk(const void *endds);
void *sbrk(int incr);
```

Les deux appels permettent de changer la taille du processus. L'adresse manipulée par les deux appels est la première adresse qui est en dehors du processus.

- Augmentations de la taille du processus avec des appels `sbrk` et on utilise
- adresses retournées par `sbrk` pour les appels `brk` pour réduire la taille du processus.

On utilisera de préférence pour les appels `sbrk` des valeurs de `incr` qui sont des multiples de la taille de page. Le système réalisant des déplacement du point de rupture par nombre entier de pages (ce qui est logique dans un système de mémoire paginé. A ne pas utiliser en conjonction avec les fonctions d'allocation standard malloc, calloc, realloc, free.