System

- Numero: 5
- Prof: Laskar Gabriel
- Date: 07 Novembre 2017

Rappels

On a vu comment on passe d'une adresse virtuelle a une adresse physique.

- PDIdx: 10 bytes (Page Table index)
- PTIdx: 10 bytes (Page index)
- Offset: 12 bytes

Page Meta Data

Dans chaque page on stocke:

- Present
- R / W
- S / U
- NX (64b / PAE)
- AVL
- PGE (Page global ou pas)
- Dirty
- Access
- Cache (CPU ou memoire)

Precisions

Cache

Les caches peuvent avoir plusieur politiques:

- Nope (On le met jamais en cache)
- Writethrough (En lecture je la met en cache en ecriture je la propage directement en memoire principale)
- Writeback (En lecture je la met en cache en ecriture aussi, c'est seulement apres que l'on va le mettre en memoire principale, il y a donc des problemes de synchronisation).

On veut desactiver le cache pour certains cas. Exemple les devices qui ne sont pas vraiment de la memoire.

AVL (Available)

Permet de stocker une information supplementaire sur le mapping.

Access

Passe a `1` une fois qu'il a lu la page. Permet de savoir si on a deja acceder a la page.

Dirty

Comme access mais sur l'ecriture. Permet de savoir si on a modifier une page.

Access et **Dirty** sont utile pour:

- On demand paging
- Swap
- File Mapping
- Shared Memory
- Copy on Write (Fork)

PageFault Handler

Dans quelle cas on pagefault ?

- Non Mappe
- Mauvaise precision
- Stack
  - [RIP, CS]
  - [RSP, SS]
  - [eflags]
  - Error cache
    * P: Pas present
    * U/S
    * R/W
    * I/D

Comment on gere les cas normaux

P (Pas present)

- Si en write $\rightarrow$ on va mapper et initialiser a `0`. On initialise le `mmap` a `0` car le but c'est de faire tourner du code meme s'il est completement buguer.
- Si en read $\rightarrow$ on va lui donner un page physique en READONLY plein de `0`. Comme ca on gagne de la RAM (Puis l'user a fait de la merde donc bon...)
- Si on write sur une zone readonly. $\rightarrow$ on va copier la page dans une nouvelle en RW. `map(RW, copy(pg))`

Shared Memory

On va creer des fichiers qui sont backer par de la memoire au lieu d'etre backer par du disque. Donc d'un ponit de vue du Page mapping, *shared memory* et *file mapping* c'est le meme combat.

Swap

L'idee est que je n'ai plus de RAM mais j'ai dis a un user qu'il m'en reste. On va donc voler de la memoire au disque. On va ensuite *swap* des pages dans dans la la memoire vole. On va selectionner des pages peu utilise. Pour indiquer que la page a ete swap on passe le bit *Present* a 0 mais on laisse l'adresse valide. On peut aussi utiliser le NX.

Si on a plus de swap. On veut continuer a tourner… Une solution c'est de tuer quelqu'un pour faire de la place. On utilise le **OOMKiller**. On doit deja selectionner la personne a tuer. On va prendre un candidat chiant qui prend beaucoup de place. De plus on va eviter de tuer un programme system (pour pas se suicider). En pratique il va tuer le browser, le serveur graphique ou gcc. Ca implique a garder des stats en interne sur tous les process et ca va permettre de calculer un score sur chaque process celui qui en a plus se fait kill.

Comment on gere les cas louches

En userland quand on utilise un debugueur.

Ou stocker l'erreur ?

- CR2:
  - Fault address

```
struct task_struct
{
  struct mm_struct *mm;
  // Some stuff
};

struct mm_struct {
  pgd;
  struct vm_area *vmas;
};
```

Sur linux il y a deux fonctions `oups` et `panic` qui sont utilise en cas de merde…

- `panic` c'est vraiment la mort.
- `oups` c'est gerable (On peut tuer le thread et passer a un autre.) Si on a plus qu'un thread se transforme en `panic`.

En cas d'erreur l'addresse de l'instruction qui n'a pas marche va sur la stack. Car on ne peut pas revenir en arriere (Du fait que les instructions sont de taille variable).