ALMO

Mémoire Virtuelle

Comment exécuter plusieurs programmes simultanément comme s'ils étaient seuls sur l'ordinateur ?

Quelques définitions

Une adresse (pour un processeur) est :

Un segment d'adresse est :

Un espace d'adressage est :

un processus (programme en cours d'exécution) est défini par :

Quelques définitions

Une adresse (pour un processeur) est :

- un numéro désignant une case contenant un nombre ;
- pour le MIPS32, les cases font un octet (8 bits) et les adresses font 32 bits. On peut donc désigner 2^{32} octets (4 GiB).

Un segment d'adresse est :

- un ensemble contiguë d'adresses.

Un espace d'adressage est :

- un ensemble de segments d'adresses.

un processus (programme en cours d'exécution) est défini par :

- un programme (le code) ;
- un espace d'adressage avec au moins 3 segments (code, data, stack) ;
- au moins une activité (nommé thread) défini par son contexte d'exécution ;
- *des fichiers ouverts, des sockets réseau, … .*

Questions

Est-il important de pouvoir exécuter simultanément plusieurs processus sur un ordinateur ?

Comment faire pour avoir plusieurs processus en mémoire ?

Pourquoi n'est-il raisonnable d'avoir un seul espace d'adressage pour tous les processus ? En principe, c'est possible, mais :

Questions

Est-il important de pouvoir exécuter simultanément plusieurs processus sur un ordinateur ?

- Evidemment, les ordinateurs généralistes doivent pouvoir simultanément exécuter un navigateur, un traitement de texte, un terminal, etc.

Comment faire pour avoir plusieurs processus en mémoire ?

- Plusieurs processus simultanément signifie plusieurs segments de code, de data et de stack.

Pourquoi n'est-il raisonnable d'avoir un seul espace d'adressage pour tous les processus ? En principe, c'est possible, mais :

- Cela impose de faire une édition de lien au moment du chargement du programme en mémoire pour le placer dans la place disponible
- Il n'y a pas de sécurité entre les processus (un processus peut accéder aux données des autres)

Solution

Pour pouvoir exécuter plusieurs processus simultanément,
il faut leur donner l'illusion que chacun est seul sur l'ordinateur.

Pour cela, le système d'exploitation va présenter un ordinateur virtuel distinct pour chaque processus.

Chaque processus doit avoir

- son processeur
- sa mémoire
- ses périphériques

Plan

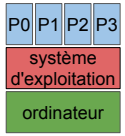
- Un mot sur la virtualisation en informatique
- Virtualisation de la mémoire
- Définition de la MMU (Memory Management Unit)
- Techniques de traduction d'adresses
- Table de pages et cache de traduction d'adresse (TLB)
- Optimisation de la table des pages

Virtualisation

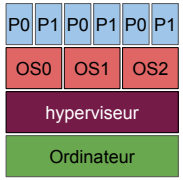
La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique [...] (https://abtec.fr/virtualisation-definition/).

Ici le serveur physique, ici, c'est l'ordinateur.

Nous voulons virtualiser un ordinateur (processeurs, espace d'adressage et périphériques) pour les processus



Mais on peut aussi virtualiser un ordinateur pour avoir plusieurs OS simultanément (cloud computing)



Virtualisation pour un processus

Pour s'exécuter un processus a besoin

- d'un processeur (cœur de calcul)
- de périphériques d'entrées-sortie
- d'un espace d'adressage

Nous avons déjà vu comment virtualiser le cœur,

→ Il suffit d'utiliser le mécanisme d'exécution en temps partagé
Il est même possible de donner une part variable de cœur en changeant la politique d'élection des tâches (équitable ou avec des priorités)

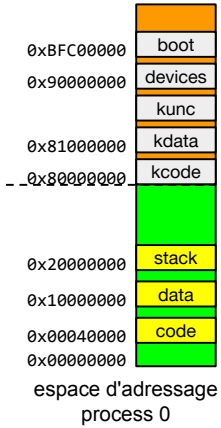
L'accès aux périphériques passent toujours (à de rares exceptions) par l'OS au travers d'une API standardisée (open, read, write, close, etc…)

→ Cette API est une abstraction de la réalité et l'OS est garant de l'exclusion d'usage des périphériques grâce à des verrous. Si deux processus veulent utiliser le même périphérique, l'OS va séquentialiser leur usage.

Il reste à virtualiser l'espace d'adressage

Virtualisation de l'espace d'adressage

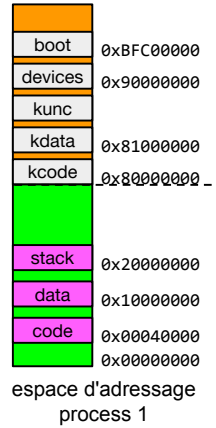
Chaque processus doit disposer d'un espace d'adressage propre.



Le processeur exécute le code du processu dans un espace virtuel, cela signifie que le processeur ne manipule que des adresses virtuelles.

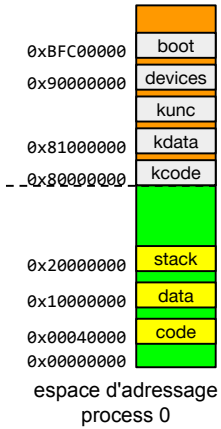
Le processeur ignore qu'il travaille dans un espace virtuel.

L'espace virtuel continue à être découpé en deux, une partie pour le noyau (accessible en mode kernel) et une partie pour l'utilisateur.



Virtualisation de l'espace d'adressage

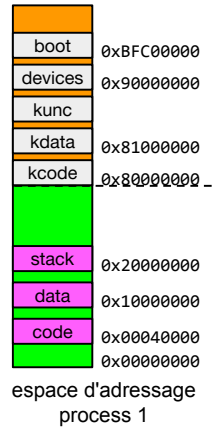
Chaque processus doit disposer d'un espace d'adressage propre.



AVANTAGE

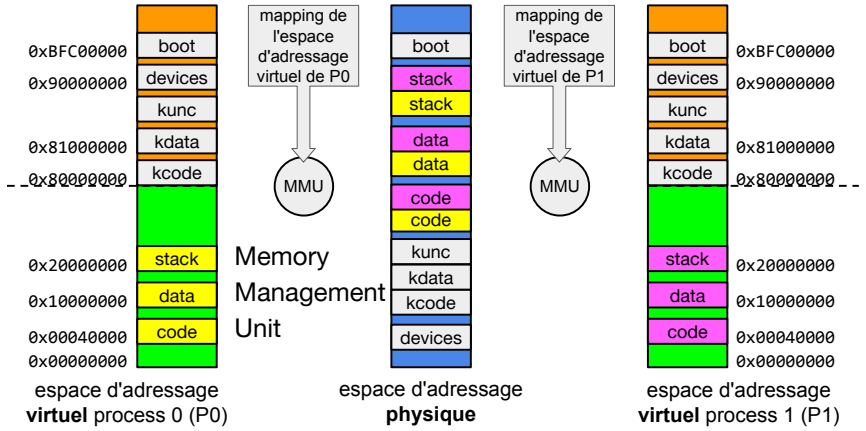
Si tous les processus ont un espace d'adressage vide avec les mêmes segments alors l'éditeur de lien peut le même fichier ldscript.

Ainsi quand on produit un code exécutable (compilé et linké) il n'est pas nécessaire de se soucier de la présence des autres exécutables.



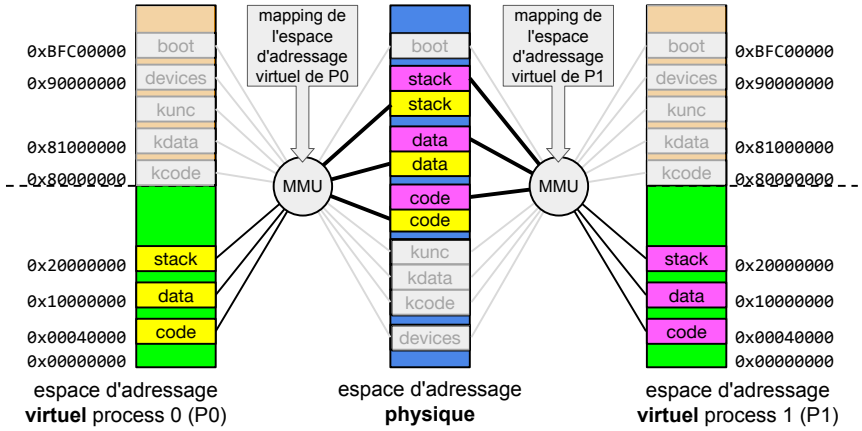
Virtualisation de la mémoire

Les segments de l'espace d'adressage virtuel de chaque processus sont mappés dans des segments de l'espace d'adressage physique.
Ce sont les composants MMU qui se chargent de la traduction d'adresses



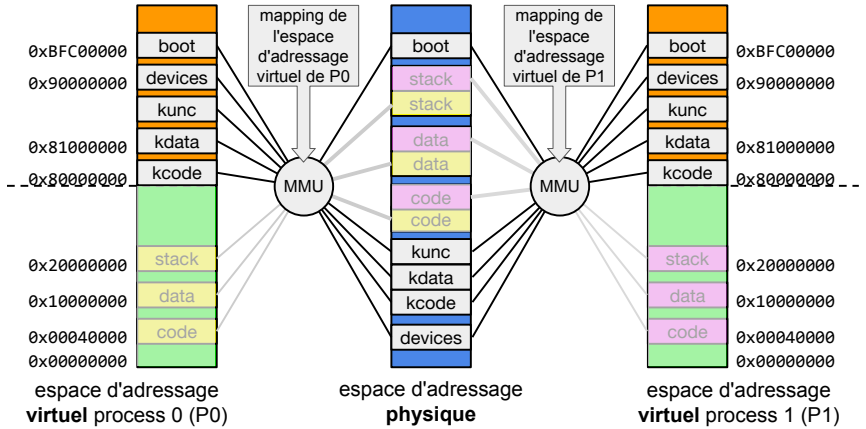
Virtualisation de la mémoire

Les segments de l'utilisateur sont évidemment mappés à des adresses différentes pour chaques processus.



Virtualisation de la mémoire

Les segments virtuels utilisés par l'OS sont mappés dans les mêmes segments physiques pour tous les processus.

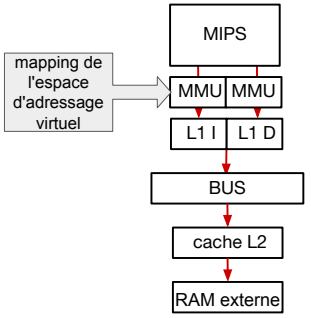


Position de la MMU dans l'architecture

L'unité de gestion de la mémoire MMU est placée entre le processeur et le système mémoire.

Toutes les adresses des programmes sont virtuelles et sont traduites en adresses physiques avant d'être envoyées vers la mémoire

Comme il y a deux accès à la mémoire pour chaque cœur (instruction et donnée), il y a deux MMUs séparées mais les deux utilisent le même mapping.



Principe de la table de mapping

La table de mapping est la table qui donne la correspondance entre les adresses virtuelles et les adresses physiques.

- Les processus manipulent des segments.
- Un segment est défini par une adresse de début et une taille : 2 mots de 32b
- Pour faire la correspondance virtuelle-physique, il suffit de 3 mots :
  - adresse de début du segment dans l'espace virtuel
  - adresse de début du segment dans l'espace physique
  - taille du segment (la même dans les deux segments)
- Si on alloue la mémoire physique par segment, c'est-à-dire que pour chaque processus, on a 3 segments pour l'utilisateur et 5 segments pour le noyau, alors pour la table de mapping se compose de 8 x 3 mots de 32b.
  (il peut y avoir quelques segments en plus mais cela reste faible)
- Pour faire la traduction, la MMU qui reçoit une adresse virtuelle, commence par trouver le segment virtuel auquel appartient l'adresse et fait une translation (elle additionne la différence entre les adresses de début des segments)
- Si elle ne trouve pas le segment virtuel, c'est une erreur de segmentation, le segment physique est sensé exister si l'OS fait bien son travail.

Problème de l'allocation par segments

Le principe de fonctionnement de la MMU avec une table de correspondance de segments n'est pas efficace.

1. Elle provoque une perte de mémoire. Si on mappe dans l'espace d'adressage physique (EAP) plusieurs programmes et qu'on en efface quelques-uns, cela va laisser des trous disjoints dans l'EAP et inutilisables pour d'autres segments.
   Il s'agit d'un problème de fragmentation externe.
2. Quand on mappe un segment, il faut l'allouer en totalité dès le début. Si vous créez une variable globale table de 100MB dans votre programme alors vous devrez avoir un segment physique de 100MB dès le début, même si votre programme n'utilise en réalité que quelques cases de ce tableau.
3. Il est impossible de faire grandir un segment virtuel si les segments physiques sont créés contiguës (pour économiser de l'espace).
4. La traduction virtuelle-physique de la MMU n'est pas si simple, il faut faire une recherche associative pour trouver le segment virtuel dans lequel se trouve l'adresse virtuelle à traduire.

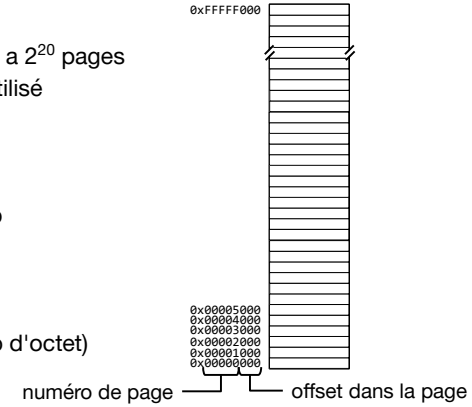
Mémoire paginée

Une page est un segment d'adresse, aligné en mémoire, de taille fixe.
Typiquement, la taille d'une page est de 4kB = 2^{12}Bytes

- L'espace d'adressage physique
  est découpé en pages
  Si une page fait 2^{12}Bytes alors il y a 2^{20} pages
- La page est l'élément atomique utilisé
  pour le mapping.

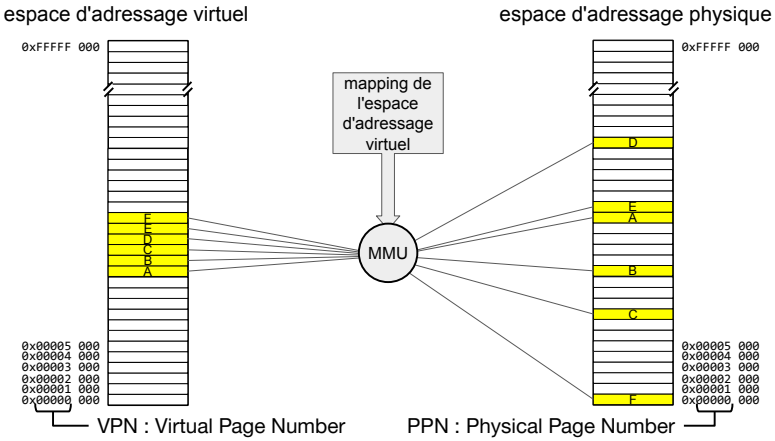
Le dessin représente le découpage
en page d'un espace d'adressage 32b

- Les 20 bits de poids forts
  sont le numéro de page
- Les 12 bits de poids faible
  sont l'offset dans la page (numéro d'octet)



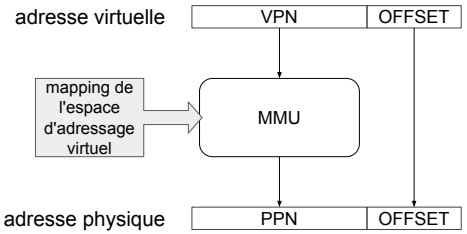
Mapping de l'espace virtuel paginé

Chaque segment d'adresse virtuel est aligné sur une page et sa taille est un nombre entier de page, chaque segment virtuel est formé d'un nombre entier de pages virtuelles.
Le mapping consiste à mapper chaque page virtuelle dans une page physique.



Memory Management Unit : MMU

Le travail de la MMU :
→ traduire le numéro de page virtuelle en numéro de page physique



Le mapping de l'espace virtuel est donc la correspondance des numéros de page virtuel en numéro de page physique

Table des pages

- Le mapping est représenté par un tableau à une dimension indexé par le numéro de page virtuelle et contenant un numéro de page physique.
- Avec des adresses numéros de pages sur 20 bits, cette table fait 2^{20} cases de 20 bits, c'est-à-dire 1Mi cases (1024*1024)
- En plus des numéros de pages physiques, on ajoute des drapeaux
  - valid : 1 = la page virtuelle est mappée, sinon elle n'est pas mappée
  - r,w,x : droits d'accès en lecture (r), écriture (w), exécution (x)
  - dirty : la page a été modifiée
  - kernel : 1 = la page physique appartient au noyau
  - etc…
- On aligne les cases de la table de pages sur des mots de 32bits. Ça fait 20bits pour le numéro de page physique, et 12bits pour les drapeaux
- La table de page fait donc 4MB
- Le mapping paginé est donc beaucoup plus gros que le mapping segmenté

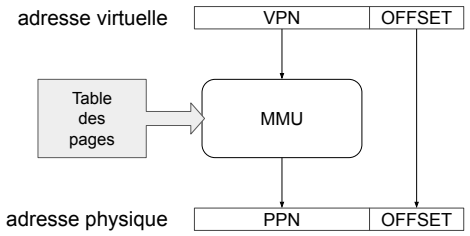


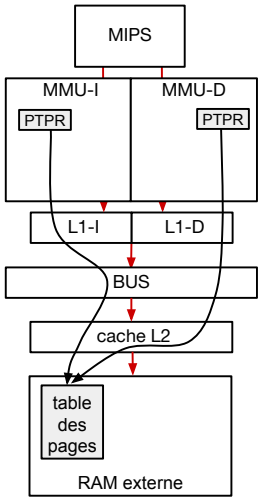
Table de page dans la MMU

En raison de sa taille (4MiB), la table des pages ne peut pas être mise dans la MMU.
⇒ Elle est placée dans la mémoire.

Pour chaque processus, on a une table de page. C'est la même pour la MMU instruction et pour la MMU data.

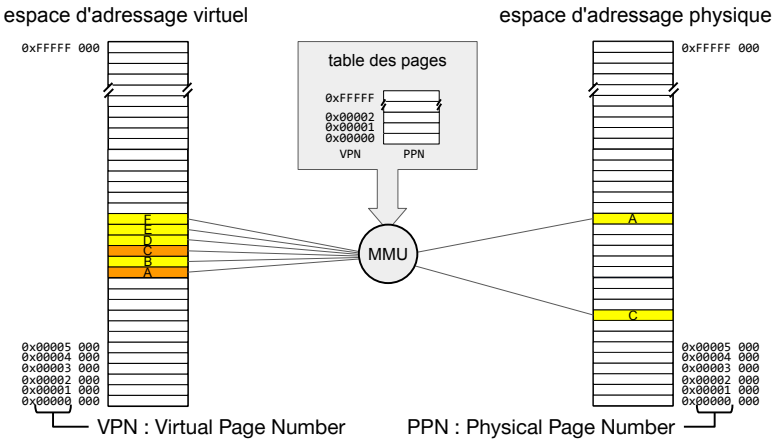
Chaque MMU doit avoir un pointeur sur la table de page en cours :
PTPR = Page Table Pointer Register

Pour chaque accès à la mémoire avec une adresse virtuelle, il faut faire une traduction vers une adresse physique, donc potentiellement un accès à la table de page, est-ce possible ?
Evidemment NON !



Allocation des pages physiques

- La table des pages n'a pas besoin d'être complète dès le début. Les pages physiques sont allouées à la demande.
- Sur le schéma, on suppose que seules les pages A et C sont utilisées, ce seront donc les seules à être mappées



Translation Look-aside Buffer : TLB

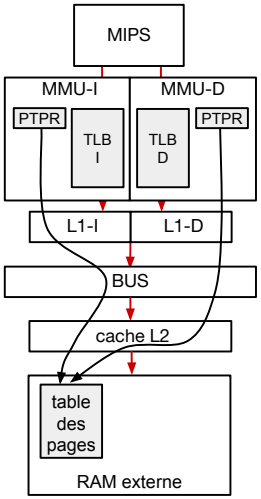
Pour accélérer la traduction d'adresse,
on place dans les MMU des caches de traductions d'adresses.
Les TLB sont des extraits de la table de page.

Ici, une TLB à 4 entrées contenant les 4 dernières traductions lues dans la table de pages.

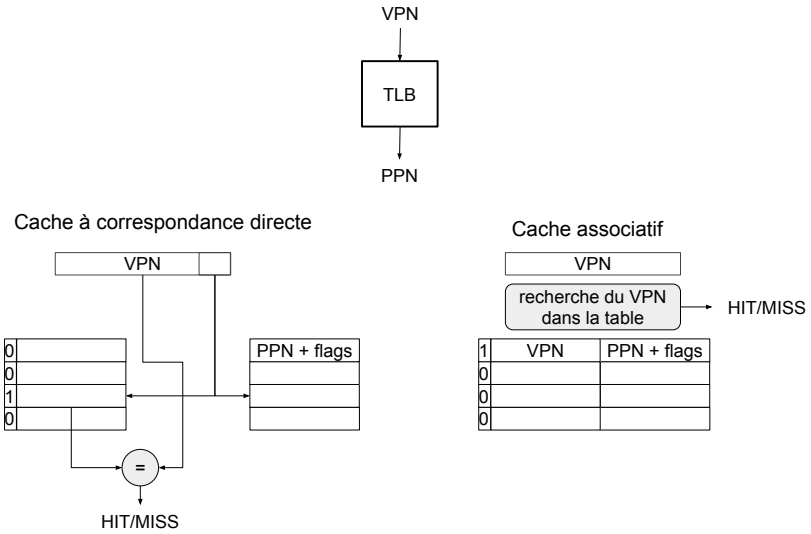
	VPN	PPN + flags
1		
0		
0		
0		

Pour chaque entrée, il y a un bit de validité et un couple VPN-PPN

Ce cache est petit (moins de 64 entrées) et le plus souvent associatif (n'importe quelle case peut servir à n'importe quelle traduction). Nous verrons en TME, les conséquences d'un cache à correspondance directe.



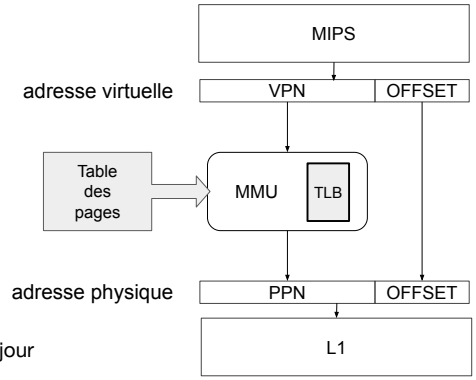
Réalisation de la TLB



Principe de fonctionnement de la MMU

Chaque MMU est placée entre le cœur et un cache.

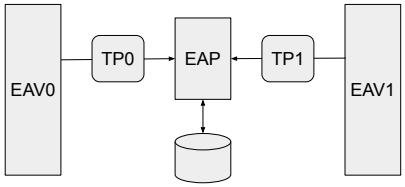
- commence par couper l'adresse en 2
  - Le numéro de VNP
  - L'offset
- Lit la TLB pour savoir si elle connaît déjà le numéro de page physique
  oui remplace le VPN par le PPN
  non c'est un **MISS TLB**,
  il faut lire la table de page
- En cas de MISS TLB, on lit la table des pages, Il y a deux cas possibles.
  1) la page virtuelle est déjà mappée
     ⇒ On lit la table et on met la TLB à jour
  2) La page n'est pas mappée,
     c'est un **page fault**
     ⇒ la MMU envoie une exception pour que l'OS trouve une page



Avantage de la pagination de l'espace d'adressage

L'intérêt de la pagination est un meilleur usage de l'espace d'adressage physique.

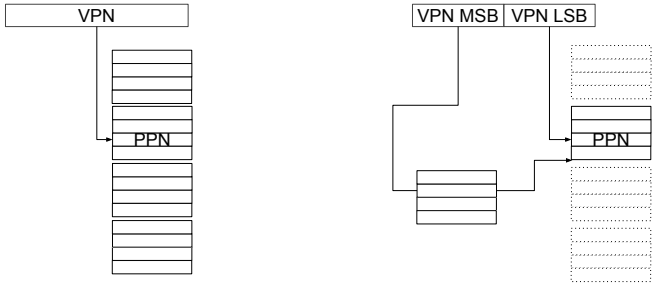
- Il n'y a plus de fragmentation externe, maintenant la fragmentation est interne mais c'est moins grave.
- On n'est pas obligé de mapper l'intégralité des pages d'un segment virtuel seule les pages virtuelles.
- Il est possible de faire grandir les segments virtuels
- La traduction virtuelle-physique est coûteuse mais simple.
- On peut aussi utiliser le disque dur pour stocker temporairement des pages physiques rarement utilisées (c'est le swap). il suffit de le dire dans la table des pages.



Optimisation de la table des pages

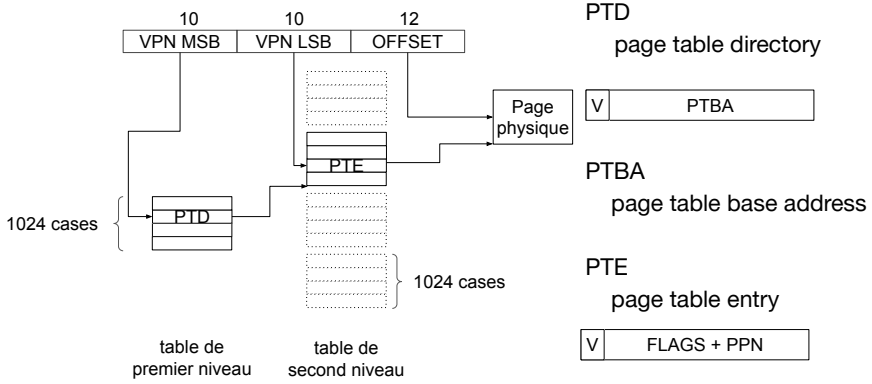
La table des pages est un tableau à une dimension de 2^{20} cases, c'est énorme.
C'est une table creuse, peu de cases sont occupées ⇒ on peut la réduire

Dans ce schéma, un tableau à 16 cases décomposé en 2 niveaux



Optimisation de la table des pages

La table des pages est un tableau à une dimension de 2^{20} cases, c'est énorme.
C'est une table creuse, peu de cases sont occupées ⇒ on peut la réduire



PTD
page table directory

PTBA
page table base address

PTE
page table entry

V | FLAGS + PPN

En résumé, nous avons vu :

- pourquoi il est nécessaire d'avoir un espace d'adressage indépendant par application
- le concept de virtualisation en informatique
- le principe de virtualisation de la mémoire
- le rôle du composant MMU et sa place dans le système mémoire
- la mémoire segmentée et la table des segments
- l'optimisation de la mémoire paginée et la table des pages
- Le principe de fonctionnement de la MMU
- La définition de la table des pages
- Le concept de cache TLB et le principe de fonctionnement
- le gain en place de la table des pages grâce à une table à 2 niveaux

Prochaine séance : l'examen :-)

Pour aller plus loin,

- Architecture matérielle
  - processeurs multi-threadés, exécution spéculatives, 64bits, etc.
  - caches write-back, n-associatifs, etc.
  - architectures hétérogènes, many-cores, etc.
  - périphériques complexes, réseau, cryptoprocesseur, GPU
  - etc.
- Système d'exploitation
  - créations dynamiques d'application et de ressources
  - noyaux modulaires, micro-noyau, multi-noyau
  - gestion des processus
  - etc.
- Application
  - vraie libc, POSIX, etc.
  - exploitation du parallélisme
  - etc.