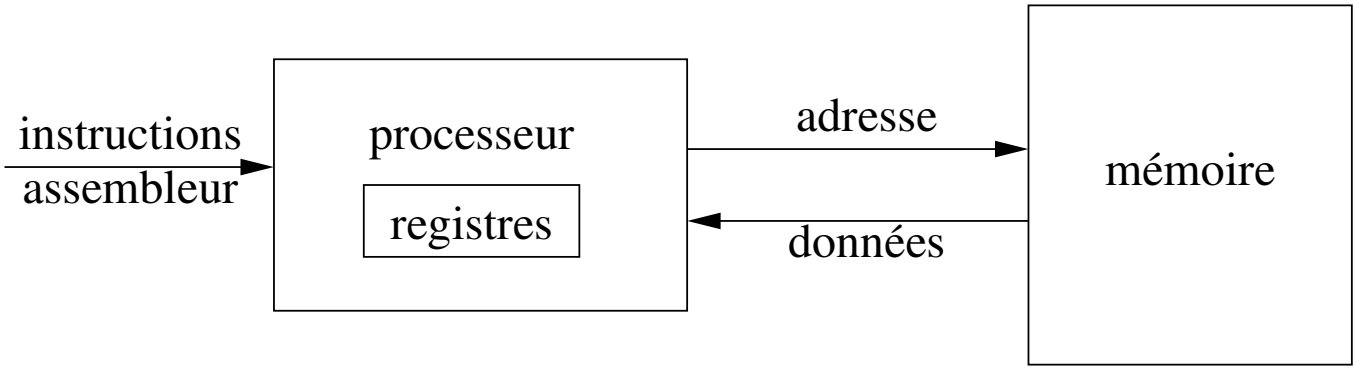
Architecture et Système

Stefan Schwoon

Cours L3, 2025/2026, ENS Paris-Saclay

Vue abstraite d'un ordinateur



Ceci est la machine abstraite proposée au programmeur.

Jusqu'aux années 70/80 : réalisation plus au moins directe de cette architecture.

Depuis : gestion de plus en plus complexe derrière les scènes
(mémoire virtuelle, micro-architecture)

Gestion mémoire/processeur avancée

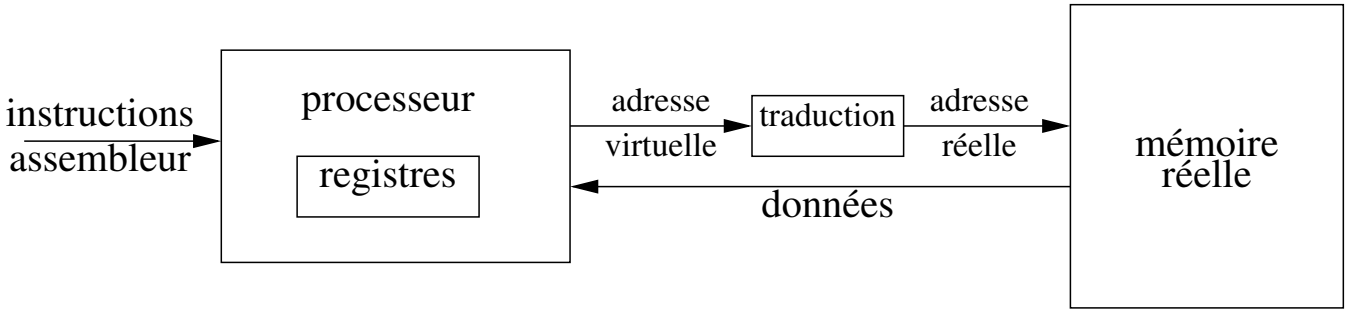
Besoins d'un système d'exploitation :

- partage du processeur entre différents tâches/utilisateurs
- protection et partage mémoire
- gestion de privilèges (mode noyau/mode utilisateur)

Optimisations :

- parallélisation (pour accélérer le calcul)
- mémoire rapide (cache)
- réordonnancement d'instructions / prediction de branches
- réduction de la consommation énergétique

Mémoire virtuelle



Les adresses fournies par les instructions assembleur sont des adresses *virtuelles*.

Une unité dédiée (Memory management unit, MMU) les traduit vers des adresses physiques.

Du coup, un processus n'a accès qu'à la mémoire qui lui appartient.

Traduction mémoire virtuelle $\leftrightarrow$ réelle

Exemple : traduction sur un processeur Intel 64-bit

Espace virtuel :

adressage avec 48 bits (donc théroriquement 256 Tera)

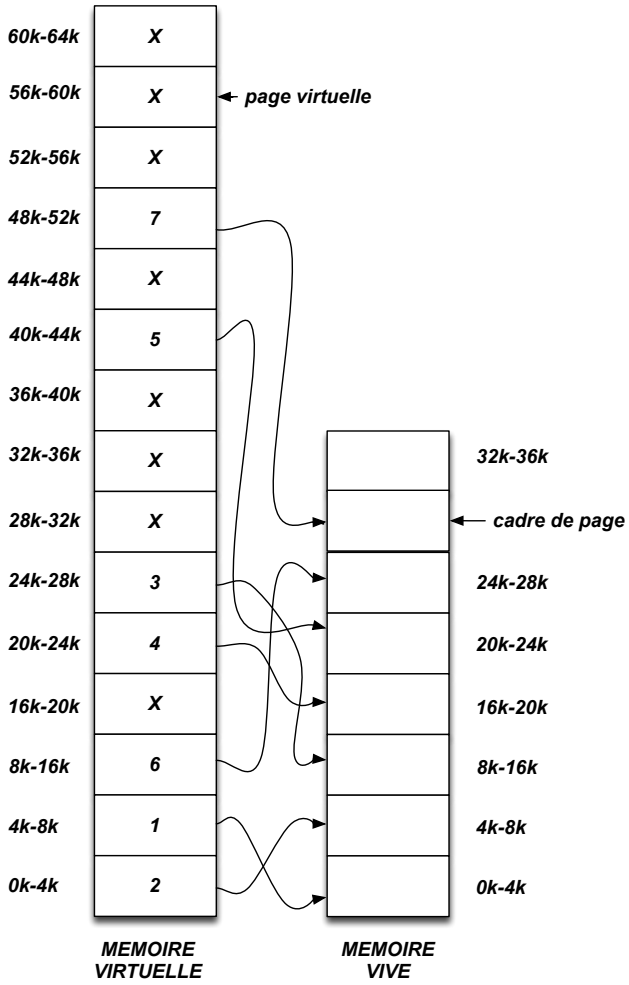
découpé en *pages* (bloc de mémoire contiguë)

taille d'une page : 1 GB, 2 MB, 4KB, selon le cas

une adresse se découpe donc en:
– numéro de page (les bits les plus significatifs)
– écart/offset (les bits les moins significatifs)

Visualisation

Une mémoire virtuelle traduit donc le numéro d'une page virtuelle vers une page réelle, en gardant l'écart:



Propriétés de la traduction:

partielle, injective

réalisée à l'aide d'un *tableau de pages* propre à chaque processus,
créé et maintenu par le noyau

Un registre dédié (p.ex. CR3 dans les Intel i7) contient le pointer vers le tableau du processus actuel. Lorsque le système bascule entre deux processus, c'est ce registre qui change de valeur.

Remarques

Lors de l'exécution du processus, les accès mémoire se font indépendamment du noyau, le matériel s'en occupe.

Certains appels système (p.ex. `malloc`, `brk`) modifient le tableau de traduction.

Traitement d'un accès mémoire illégal (à une adresse non affectée) :

- le processeur déclenche une interruption ;
- le système rattrape cette exception et envoye un signal (`SIGSEGV`) au processus ;
- le signal termine le processus (sauf s'il le rattrape).

Détails de la pagination (à l'exemple des x86)

Traduction réalisée à l'aide d'un *trie* (tableau à plusieurs niveaux).

Une partie des pages réelles (de taille 4 KB) est utilisée pour stocker les tableaux – 4 KB contiennent $512 = 2^9$ adresses à 64 bits.

Dans un premier temps, on découpe l'espace virtuel en $512 = 2^9$ blocs de 512 GB ($= 2^{39}$ octets).

Un premier tableau est alors utilisé pour indiquer si un tel bloc est soit entièrement inutilisé, soit l'adresse d'une autre page qui décrit son découpage.

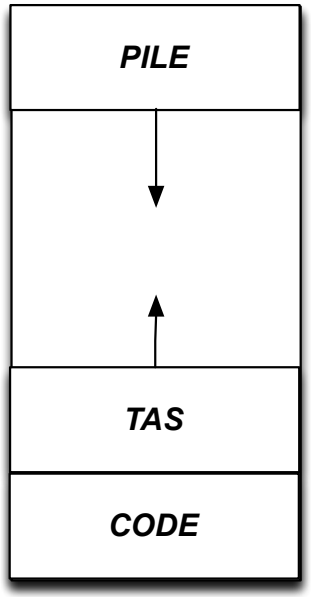
Tout bloc de 512 GB est découpé en 512 blocs de 1 GB ($= 2^{30}$ octets), et de suite. Après 4 niveaux ($4 \times 9 = 36$) on obtient l'adresse d'un bloc de 4 KB ($4096 = 2^{12}$).

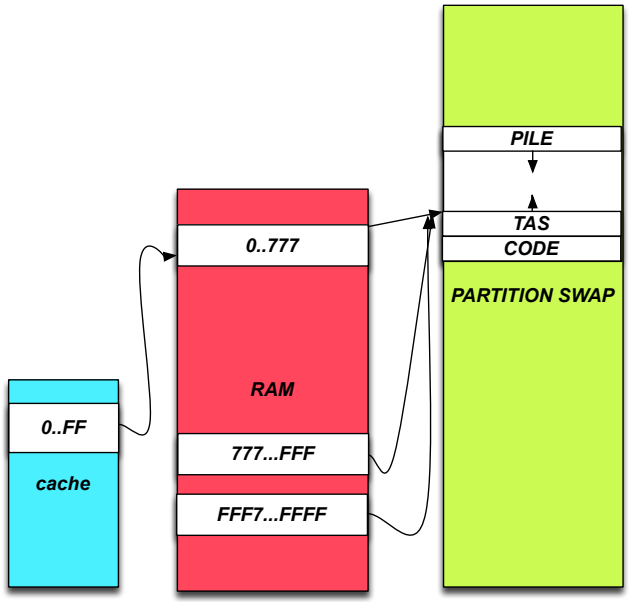
Organisation de la mémoire virtuelle

Les 16 bits non-utilisés stockent des informations sur les *permissions*: accès en écriture, page exécutable, accès privilégié (en mode noyau seulement), . . .

Sous Linux, un processus vit dans la moitié basse de la mémoire virtuelle (bit 47 = 0). La partie haute est reservé au noyau.

Organisation en code, tas et pile:





Gestion du tas

Une grande partie de l'espace virtuelle ne correspond à aucune page réelle (et les accès à ces adresses déclenchent une violation de segment).

Un processus peut élargir la taille de son tas avec `brk`.

`malloc` / `free` : Fonctions en mode utilisateur qui organisent le tas (font appel à `brk` si nécessaire).

Ces fonctions utilisent un partie du tas pour organiser les allocations.

Les bogues (débordement de mémoire) peuvent corrompre cette information, ménant à des effets secondaires non prévisibles.

Translation lookaside buffer (TLB)

Avec le principe évoqué précédemment, chaque accès mémoire virtuel se traduit en cinq accès réels $\rightarrow$ inefficace

Du coup, on mémorise les pages utilisées le plus souvent dans le TLB (mémoire associative mais limitée).

Attention, une même adresse virtuelle correspond à plusieurs adresses réelles, en fonction du processus:

Certains processeurs récents offrent un régistre stockant l'identifiant du processus actuel, pris en compte par le TLB.

Dans d'autres processuers, le TLB doit être invalidé quand on bascule vers un autre processus.

Failles Spectre et Meltdown

Failles découvertes en 2017 / 2018

Principe de Spectre (rappel):

Un attaquant fournit un paramètre hors limite à une fonction victime.

La fonction victime teste si le paramètre est bon et le rejète sinon.

Pourtant, la prédiction de branche fait une exécution spéculative avec le mauvais paramètre.

Le mauvais paramètre fait accèder la fonction à une valeur sécrète.

En fonction de la valeur sécrète la performance du cache sera différente.
$\rightarrow$ la valeur sécrète peut être découverte

Failles Spectre et Meltdown

Failles découvertes en 2017 / 2018

Principe de Meltdown:

Le noyau garde toute la mémoire physique dans une partie de la mémoire virtuelle. Le descripteur de ces pages ne permet que des accès par le noyau.

Un attaquant y accède quand même.

Cet accès correspond à plusieurs micro-instructions:
récupérer le descripteur de pages, tester le résultat, obtenir la case mémoire ou rejeter l'accès, ...

Effet de cache et d'exécution spéculative: on récupère la case mémoire spéculativement.

L'effet dans le cache peut être mésuré comme dans Spectre.
$\rightarrow$ l'attaque casse la protection mémoire