

Séance 6

Gestion de la mémoire



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons
Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels

- Définition et caractérisation des **deadlocks**
 - Modélisation d'un système avec des processus et des ressources
 - Quatre conditions d'apparition d'un deadlock
- Technique de **gestion** des deadlocks
 - Prévention et détection de deadlocks
 - Techniques avec graphes ou algorithme du banquier
 - Politique de terminaison d'un processus et récupération

Objectifs

- Organisation de la mémoire dans le hardware
 - Addressage des zones mémoire
 - Mémoire physique et logique
- Allocation de la mémoire aux processus
 - Allocation contigüe et segmentation
 - Problèmes de fragmentation
 - Pagination

RUE
DU MALONAT
CARRIERA
DÓU MALOUNAT.

Adressage

Processus et mémoire

- Programme est chargé en mémoire pour en faire un processus

Disque → Mémoire principale

- Algorithmes de gestion de la mémoire

- 1 Gestion au niveau machine
- 2 Pagination et segmentation

- La mémoire est un grand tableau d'octets

Chaque octet est identifié par une adresse

Unité mémoire

- Lors de l'**exécution d'une instruction** lors d'un cycle CPU
 - Instruction de **chargement** depuis la mémoire
 - Instruction de **stockage** vers la mémoire
- L'**unité mémoire** voit défiler un flux d'adresses mémoire

L'unité répond aux demandes, sans interprétation

- Manipulation d'**adresses physiques**

Adresses des octets situés en mémoire principale

Hardware de base

- Le CPU peut directement accéder à **deux types de mémoire**
 - Mémoire principale (RAM)
 - Registres internes au CPU
- **Temps d'accès** mémoire
 - Registres lus en un seul cycle CPU
 - Accès mémoire principale en plusieurs cycles
 - Il faut une transaction sur le bus mémoire
 - Accélération avec cache entre CPU et mémoire principale

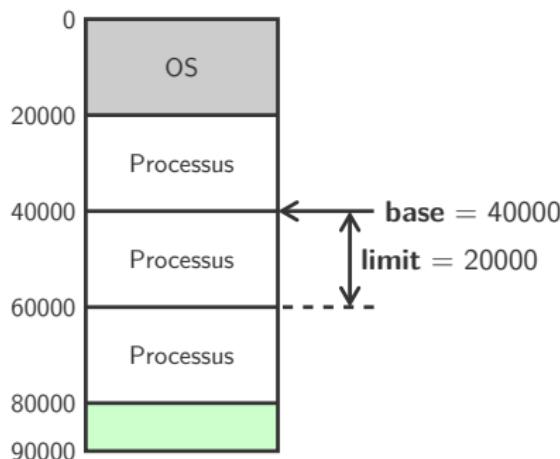
Protection de la mémoire (1)

- **Protection** des zones mémoires en fonction de l'utilisateur
 - Un processus utilisateur ne peut pas accéder à la zone de l'OS
 - Un processus ne peut accéder qu'à sa mémoire
 - Protection des différents utilisateurs si multiusers
- La protection doit se faire au **niveau hardware**

L'OS n'intervient pas dans les demandes du CPU vers la mémoire
- **Plusieurs implémentations** hardware de protection possibles

Protection de la mémoire (2)

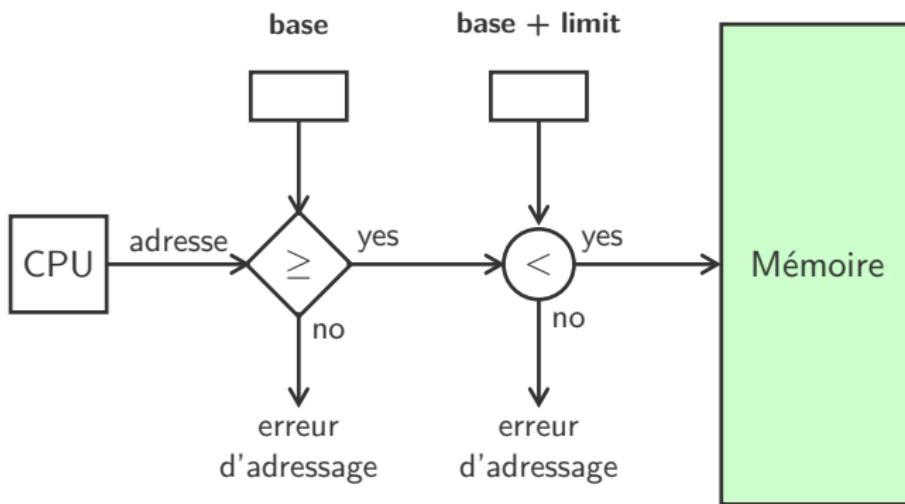
- Deux registres pour **délimiter** la zone mémoire
 - **base** : la plus petite adresse physique légale
 - **limit** : la taille de la zone mémoire



Protection de la mémoire (3)

- Seul l'OS a accès aux registres **base** et **limit**

Utilisation d'une instruction privilégiée, en mode kernel



Liaison des adresses (1)

- Processus déplacé entre disque et mémoire centrale

Processus en attente sont dans la queue d'entrée

- Représentations différentes des adresses d'un programme

- Adresse symbolique dans le code source

Par exemple, une variable count

- Le compilateur crée des adresses relocalisables

Par exemple, 42 octets après le début du module

- L'éditeur de lien lie les adresses relocalisables à des absolues

Par exemple, 72042 si le processus chargé en 72000

Liaison des adresses (2)

- La **liaison des adresses** peut se produire à plusieurs moments

- **Compilation**

Adresse absolue si l'emplacement mémoire est connu lors de la compilation (par exemple les programmes MS-DOS .COM).

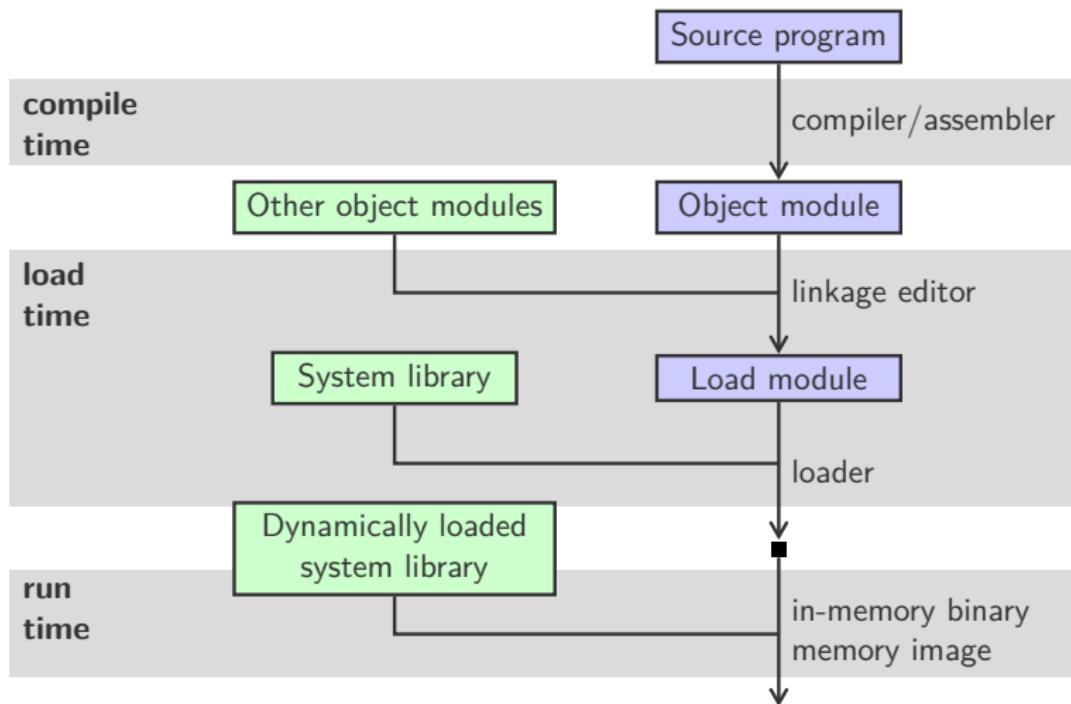
- **Chargement**

Adresse relocisable et liaison faite lors du chargement dans la mémoire, rechargement du code à chaque changement de l'adresse de début.

- **Exécution**

Si le processus peut changer de segment mémoire durant l'exécution, la liaison doit être faite lors de l'exécution.

Liaison des adresses (3)

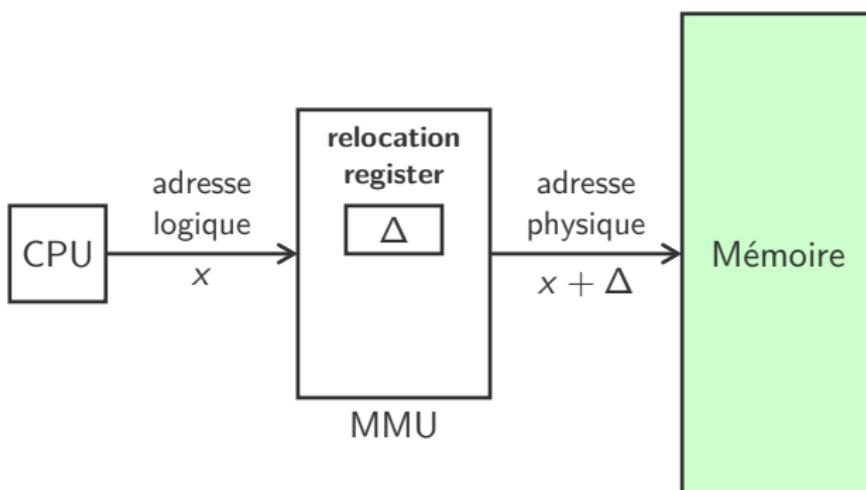


Adresse logique/physique

- Deux **espaces d'adresses** différents
 - Adresse **physique** générée par le CPU
 - Adresse **logique** chargée dans le memory-address register
Adresse lue par l'unité mémoire
- Adresses **physique et logique**
 - Identiques pour liaison à la compilation ou au chargement
 - Différentes pour la liaison à l'exécution
Les adresses logiques sont appelées adresses virtuelles

Unité de gestion mémoire (MMU)

- Système matériel de **conversion** adresse physique → logique
Le programme utilisateur ne manipule que les adresses logiques
- Un **MMU** simple ajoute la valeur du **relocation register**



Chargement dynamique

- Avant, le **programme et ses données** devaient être en mémoire
Limitation de la taille à celle de la mémoire physique
- Chargement **dynamique** à la volée
 - Routine chargée depuis le disque lorsque nécessaire
 - Meilleure utilisation de la mémoire
 - Utile lorsque grande quantité de code peu souvent exécutée
- **Pas d'interventions** de l'OS

Prise en charge par le programmeur lors du design du programme

Liaison dynamique

- Liaison dynamique de librairies lors de l'exécution

Aussi appelée librairies partagées

- Inclusion d'un **stub** pour faire une référence à la routine
 - Vérifie si la librairie est déjà chargée ou non
 - Se remplace par l'adresse de la routine
 - Routine chargée une seule fois pour tous les processus
- Facilite la **gestion des correctifs** de bugs
- L'**OS doit vérifier** que la routine est dans l'espace du processus

Swapping



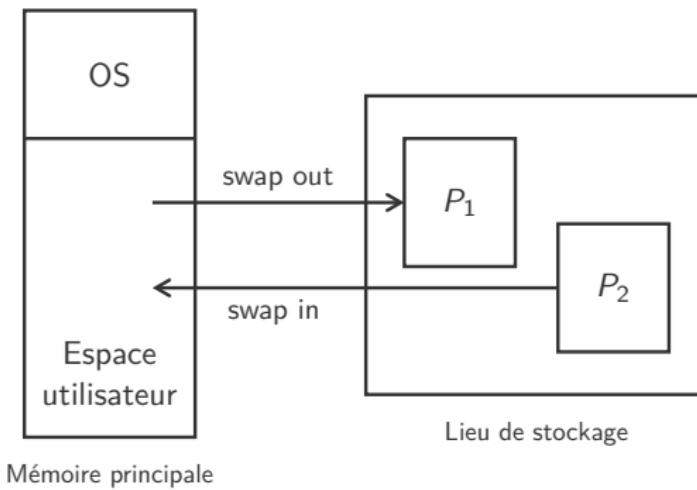
Swapping

- Processus **swappé** hors de la mémoire temporairement
Processus déplacé sur le disque dur par exemple
- Augmentation du degré de **multiprogrammation** d'un système
Espace d'adresses physiques total plus grand que le réel
- **Dispatcher** en charge des swaps
 - Changements de contextes prennent du temps !
 - 2 s si processus de 100 Mio et taux de transfert de 50 Mio/s

Swapping standard

- Utilisation d'un **disque rapide** et suffisamment grand

Accès direct aux images mémoires



Contraintes

- Vitesse de transfert pour les swaps in/out

Temps de transfert total \propto à la quantité de mémoire swappée

- On ne peut swapper qu'un processus **idle**...

Pas d'opérations E/S en cours si périphérique a besoin mémoire...

- ...ou alors techniques de **double buffering**

Buffers de l'OS pour les résultats des opérations d'E/S

Appareils mobiles

- Pas de swap utilisé
 - Mémoire flash en plus faible quantité
 - Nombre limité d'opérations d'écriture pour la vie du disque
- iOS demande aux app de **libérer** volontairement de la mémoire

Une app n'ayant pas assez libéré peut se voir terminée
- Android **sauve l'état** de l'app avant de la terminer

Cela permet une restauration rapide de l'app

Allocation de la mémoire



Allocation contigüe

- Mémoire partagée entre l'OS et les processus utilisateurs

Bon choix de répartition à trouver

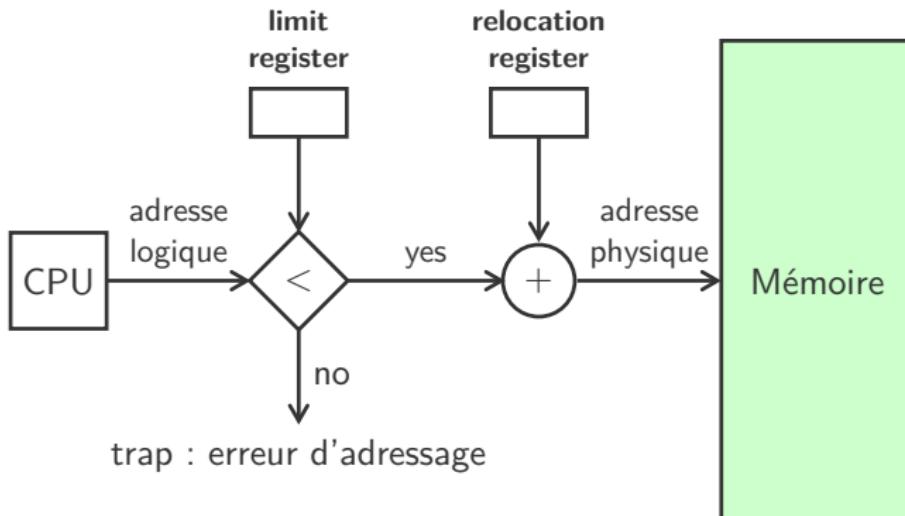
- L'allocation doit être faite de manière efficace

- OS réside aux adresses basses avec le vecteur d'interruption
- Processus utilisateurs dans les adresses hautes

- Processus placés dans des sections contigües de la mémoire

Placement l'un à côté de l'autre en mémoire

Protection de la mémoire (4)



- Deux **registres chargées** par le dispatcher

Permet un changement dynamique de la taille de l'OS

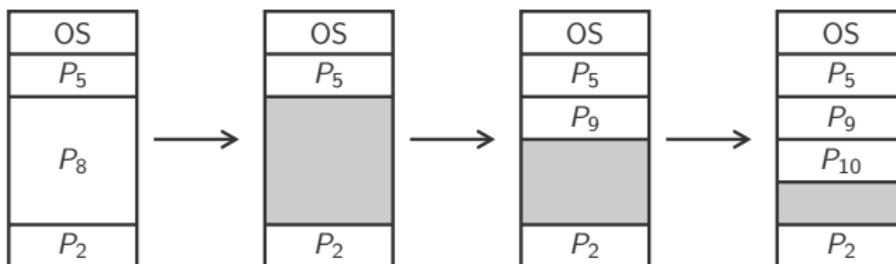
Allocation de la mémoire

- Méthode **multi-partition**

- Plusieurs partitions de tailles fixes en mémoire
- Un processus par partition

- Méthode **partition variable**

- Liste de la mémoire libre et occupée
- Un processus terminé libère un trou



Allocation de mémoire dynamique

Comment satisfaire une requête de mémoire de taille n à partir d'une liste de trous ?

- **Trois stratégies** différentes possibles
 - **First-fit** prend le premier trou libre qui est assez grand
 - **Best-fit** prend le plus petit trou libre qui est assez grand
 - **Worst-fit** prend le plus grand trou libre
- **First-fit et best-first** meilleurs que worst-fit

En terme de temps et utilisation disque (et first-fit plus rapide)

Fragmentation

- Deux types de **fragmentation** possibles dans la mémoire
 - Fragmentation **externe** avec first-fit et best-fit

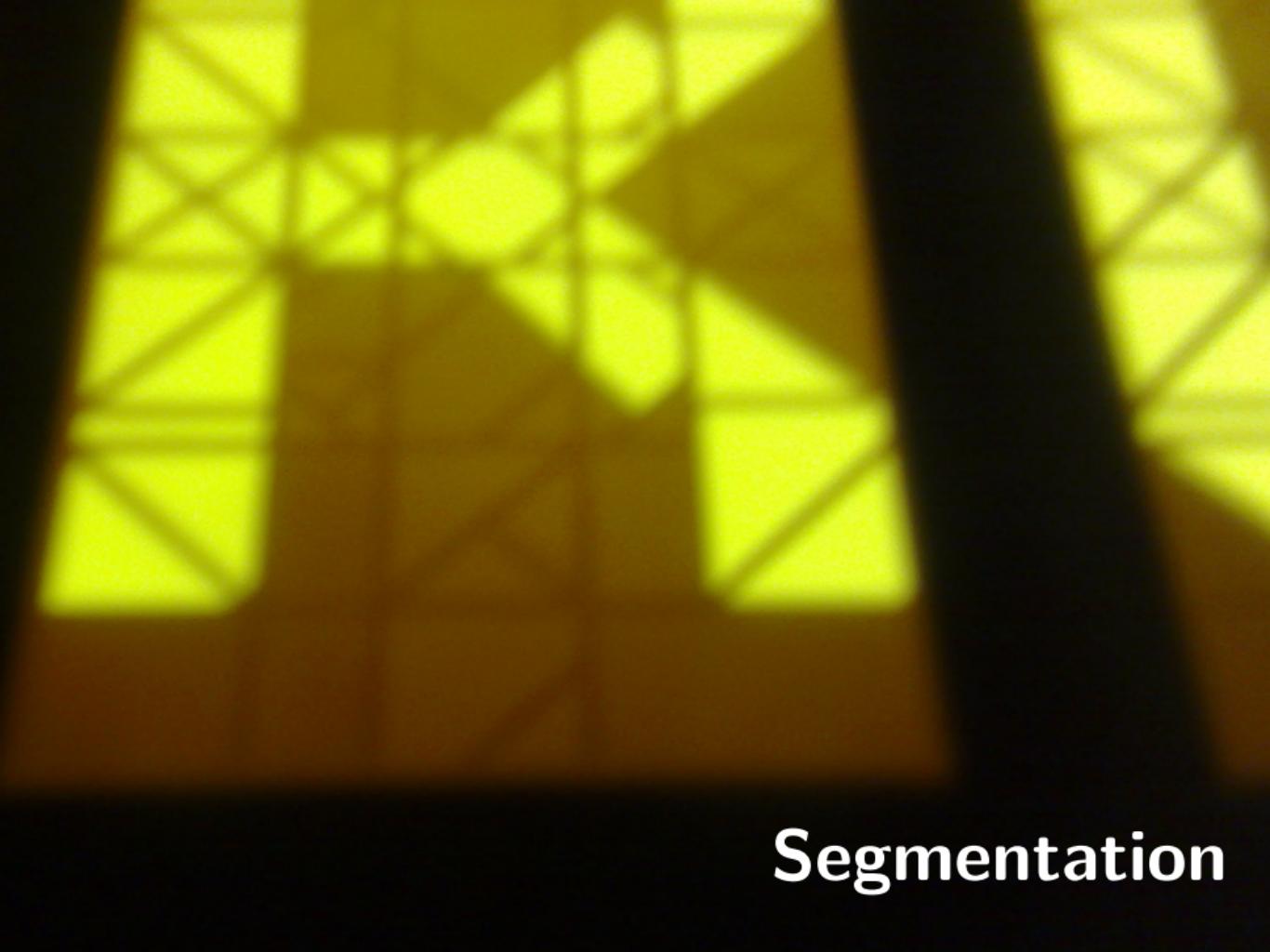
*Découpe de l'espace restant en petits bouts
Assez de place mémoire pour une requête, mais pas contigu*
 - Fragmentation **interne**

*Mémoire allouée plus grande que mémoire demandée
Différence interne dans les blocs de taille fixe pas utilisée*
- Règle des **50 pourcents** pour first-fit (statistique)

Pour N blocs alloués, il y a $0.5N$ blocs perdus par fragmentation

Compactage

- Résolution de la fragmentation par **compactage**
 - Déplacer le contenu mémoire pour avoir des grands blocs libres
Seulement possible si la relocalisation est dynamique
 - Possible seulement avec relocalisation dynamique d'adresses
 - Tous ces déplacements mémoire peuvent couter cher
- Autoriser l'espace d'adresses logique d'être **non-contigu**
Utilisation de la segmentation et de la pagination

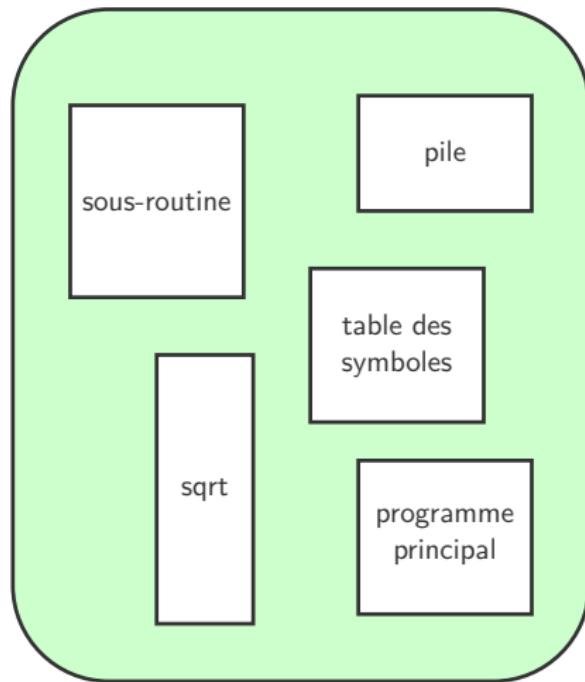


Segmentation

Segmentation

- Gestion de la mémoire qui correspond à la **vue programmeur**
- Un espace d'adresses logiques est une collection de **segments**
Un segment possède un nom et une longueur
- **Identification d'une adresse** par deux éléments
 - Un nom de segment
 - Un décalage dans le segment

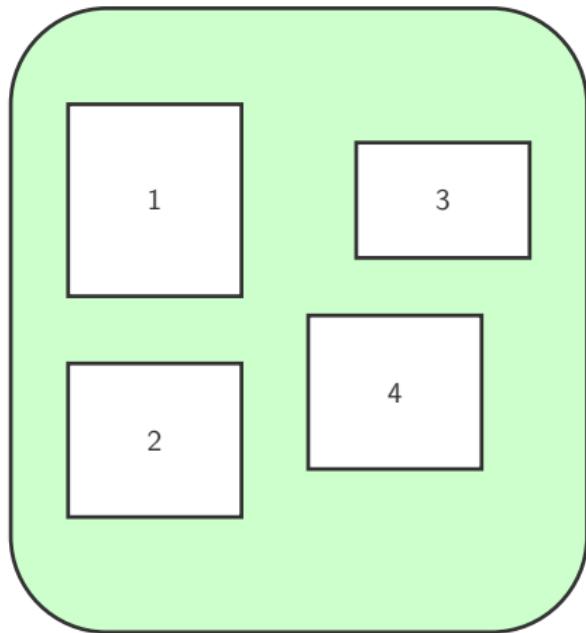
Vue programmeur d'un programme



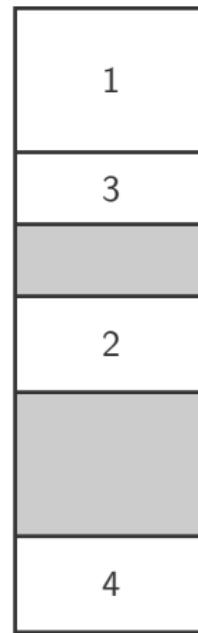
Exemple en C

- Segments construits par le compilateur
- Exemple en C
 - 1 Code
 - 2 Variables globales
 - 3 Tas
 - 4 Pile
 - 5 Librairie standard C
- Les bibliothèques liées à la compilation dans un segment à part

Vue logique de la segmentation



Espace utilisateur

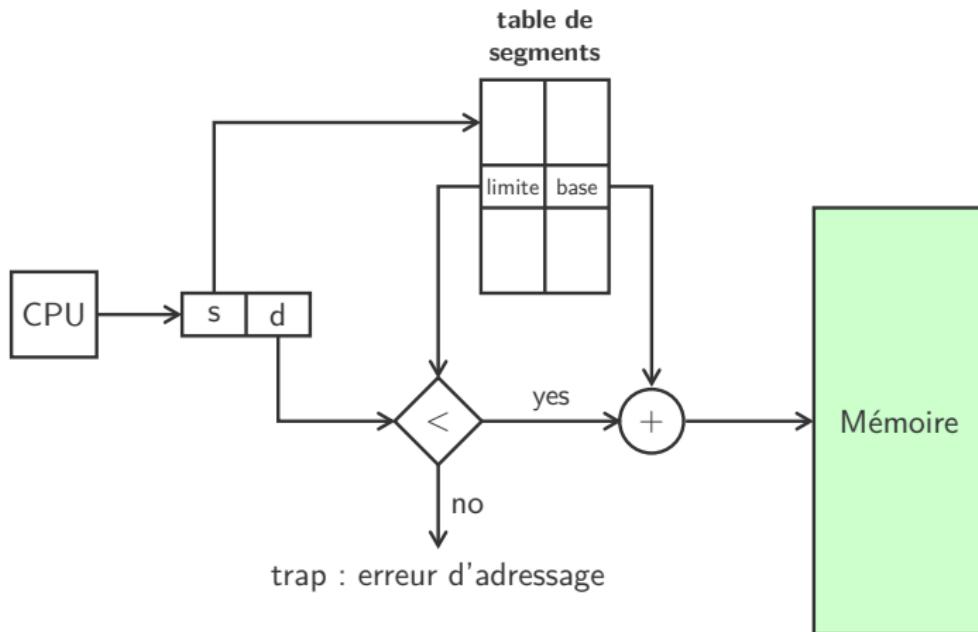


Espace mémoire physique

Calcul des adresses (1)

- **Adresse logique** : \langle numéro de segment, décalage \rangle
- **Table de segments** : chaque entrée contient deux éléments
 - **base** : adresse physique du début du segment
 - **limite** : longueur du segment
- Registre de table de segments
Segment-Table Base Register (STBR)
- Registre de longueur de table de segments
Segment-Table Length Register (STLR)

Calcul des adresses (2)



Pagination



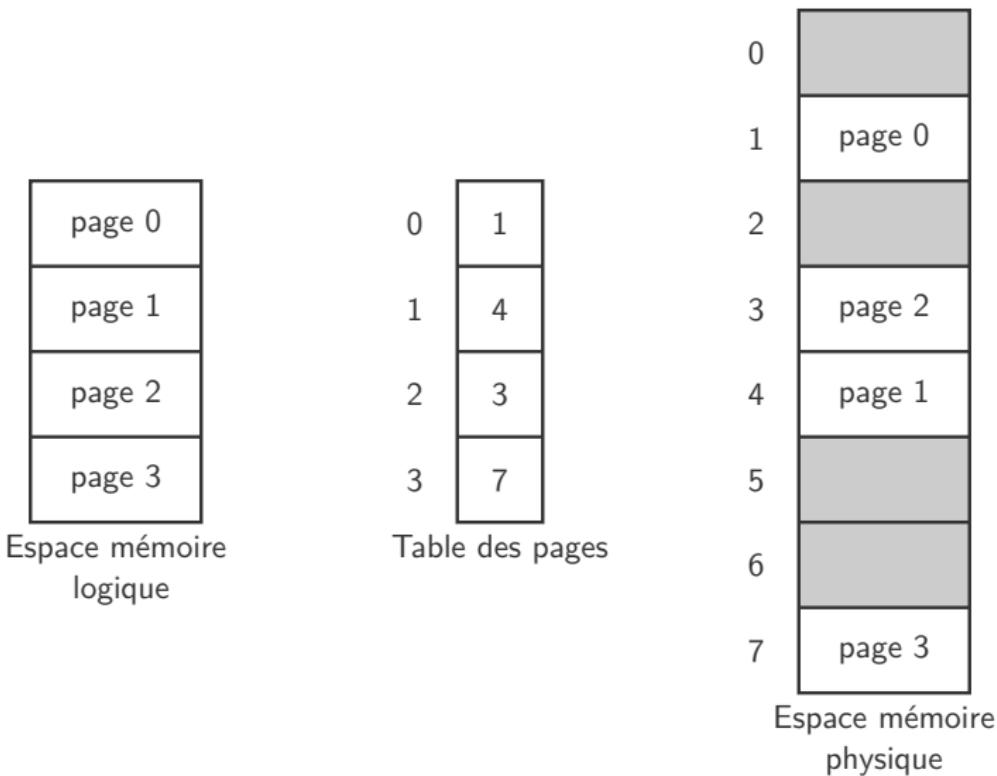
Pagination

- Permet aussi à l'espace mémoire alloué de **ne pas être contigu**
- **Évite** la fragmentation externe et la nécessité de compactage
Mais de la fragmentation interne peut arriver...
- Blocs de **tailles fixes** et identiques facilitent le swap
Blocs sur stockage externe même taille qu'en mémoire

Concepts de base

- Mémoire physique découpée en **cadres** de tailles fixes
La taille est une puissance de 2 (entre 512 octets et 1 Gio)
- Mémoire logique découpée en **pages** de même taille
Pour faciliter le swapping, notamment
- Chargement d'une **page vers un cadre**

Vue du modèle de pagination



Calcul des adresses (3)

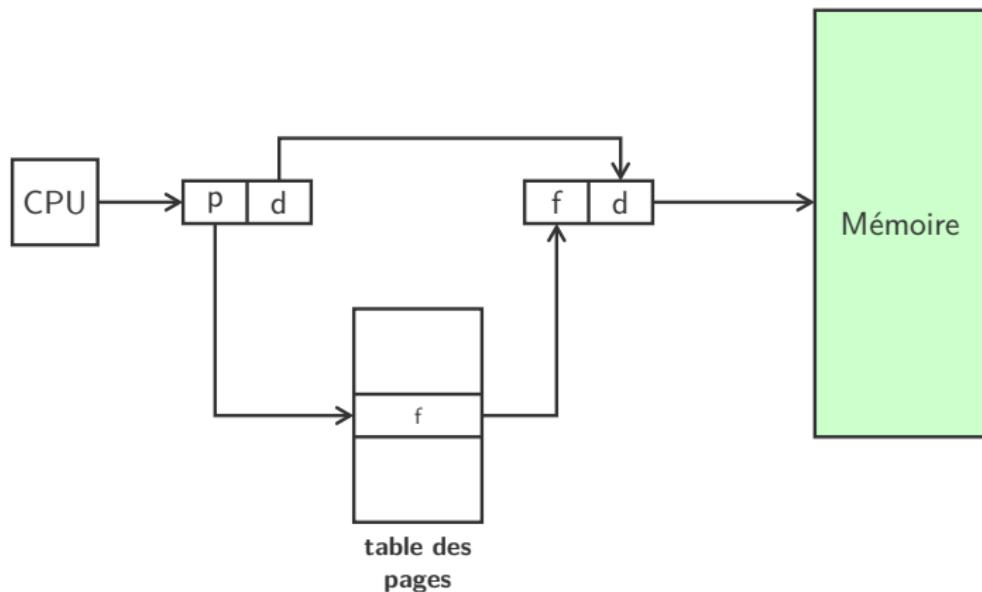
- **Adresse logique** : \langle numéro de page, décalage \rangle
- **Table de pages** : entrées avec adresse physique de la page
- Registre de table des pages

Page-Table Base Register (PTBR)

- Registre de longueur de table des pages

Page-Table Length Register (PTLR)

Calcul des adresses (4)



Calcul des adresses (5)

- Puissance de 2 pour la taille des pages

Traduction des adresses logiques plus facile

- Espace logique de 2^m octets et pages de 2^n octets
 - $m - n$ bits de poids forts pour le **numéro de la page p**
 - n bits de poids faibles pour le **décalage d**

Table des cadres

- Maintient d'une liste des **cadres libres**

Maintient en fait une entrée par cadre dans une table

- Au lancement d'un **nouveau processus**

- Recherche de cadres libres pour les pages demandées
- Création d'une table des pages
- Remplissage de la table des pages

- L'OS maintient une **copie de la table des pages**

Pour traduire les adresses lors d'appel système

Stockage page des tables

- Une table des pages **par processus**

Pointeur vers la table des pages dans un registre et dans le PCB

- Table des pages stockées dans un **ensemble de registres**

Seulement possible pour des petites tables

- Table des pages en **mémoire principale**

Avec un registre qui pointe sur son adresse de début

Page-Table Base Register (PTBR)

Translation Look-aside Buffer (TLB)

- Mémoire associative rapide

Séquence d'entrées de la forme ⟨clé, valeur⟩

- Comparaison simultanée d'un élément avec toutes les clés

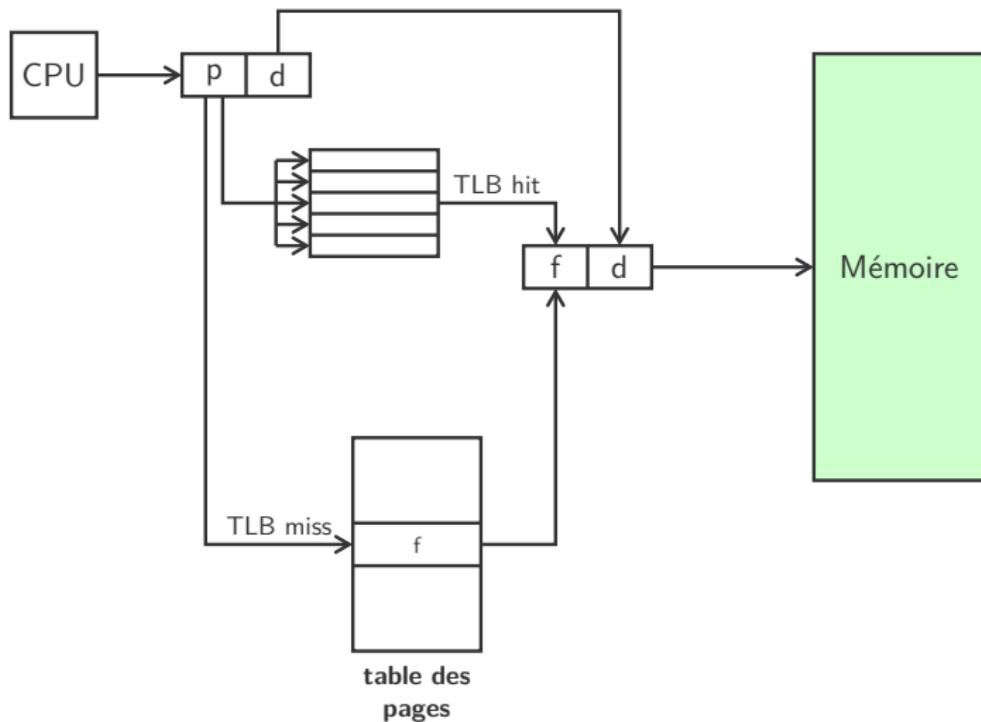
- Taille petite du TLB avec peu d'entrées

Entre 32 et 1024 entrées

- Recherche dans TLB, puis dans table des pages si nécessaire

Accès direct versus deux accès mémoire si en mémoire principale

Calcul des adresses (6)



Temps d'accès effectif

- Un accès mémoire (TLB) contre deux (table de pages)
- Taux de succès (TS)

Probabilité de trouver p dans la TLB

- Temps d'accès effectif (TAE) (si cycle mémoire de 100 ns)

$$TS \times 100 + (1 - TS) \times 200$$

- Entrées du TLB peuvent être modifiées pour **limiter les miss**

Certaines entrées sont wired down, pour kernel code par exemple

Protection de la mémoire

- Des **bits de protection** par cadre stocké en table des pages
- Bit de **mode** : read-write et read-only
Vérification lors du calcul de l'adresse physique
- Bit **valide/invalid** pour autorisation d'accès
Détermine si la page appartient ou non au processus

Page partagée

- Partager du code commun entre plusieurs processus

Ne charger qu'une seule fois la page en mémoire

- Seulement possible pour du code réentrant

Code qui ne se modifie pas lui-même

- Les données propre à chaque processus resteront séparées

Structure de la page de table

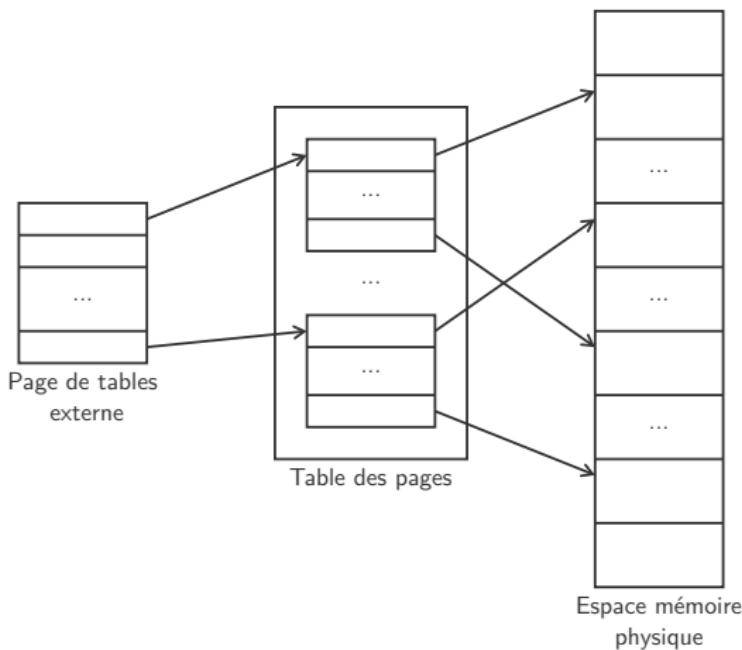
- Trois manières possibles pour structurer la table des pages
 - 1 Pagination hiérarchique
 - 2 Table des pages hachée
 - 3 Table des pages inversée
- Très large espace d'adresses logiques : 2^{32} à 2^{64}

La table des pages devient elle-même énorme !

Pagination hiérarchique (1)

- Table des pages découpées en **deux niveaux**

La table des pages est elle-même paginée



Pagination hiérarchique (2)

- Adresses physique sur 32 bits et pages de 4 Kio

Du coup p sur 20 bits et d sur 12 bits

- Table des pages elle-même paginée

Du coup, p_1 sur 10 bits et p_2 sur 10 bits

- p_1 : index dans la page des tables externe
- p_2 : décalage dans les pages de la table des pages

Table des pages hachée

- Trois éléments dans **chaque entrée** de la table de hachage
 - 1 Le numéro de page virtuel
 - 2 Le numéro du cadre
 - 3 Un pointeur du prochain élément de la liste chainée

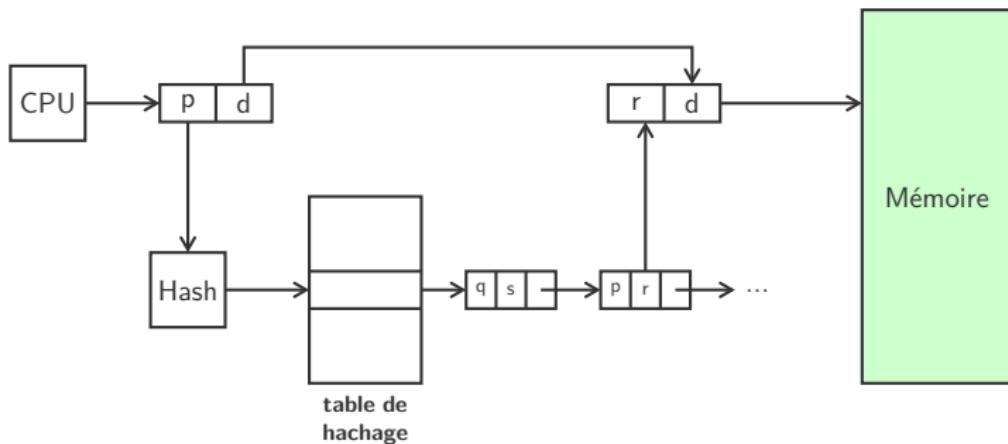
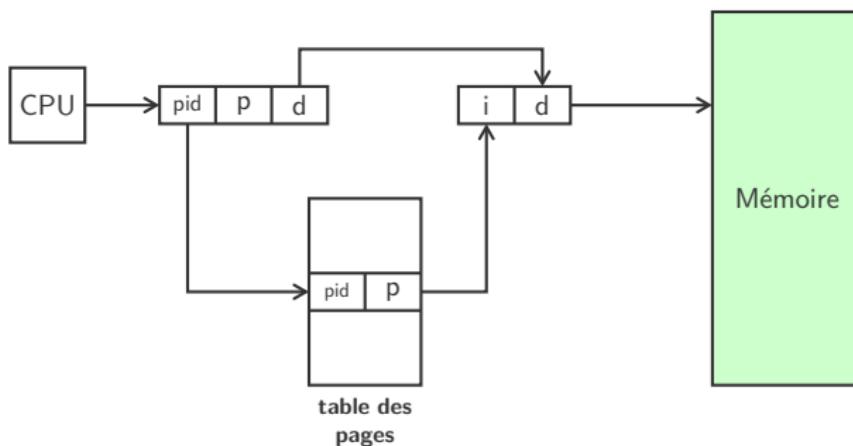


Table des pages inversée

- Classiquement, une **table des pages** par processus
 - Une entrée pour chaque page utilisée par le processus*
- Par **inversion**, une entrée pour chaque cadre



Crédits

- <https://www.flickr.com/photos/94439743@N05/8600304460>
- <https://www.flickr.com/photos/alinashea/4379718680>
- <https://www.flickr.com/photos/wirewiping/6159572150>
- <https://www.flickr.com/photos/hackaday/2062947011>
- <https://www.flickr.com/photos/kimberlykoppen/5858521608>