5 - Gestion de la mémoire

F. Boyer, UJF-Laboratoire Lig
Fabienne.Boyer@imag.fr

- Cours conçu à partir de
  - Cours de E. Berthelot
    - http://www.iie.cnam.fr/%~EBerthelot/
  - Cours de A. Sylberschatz
    - www.sciences.univ-nantes.fr/info/perso/permanents/attiogbe/SYSTEME/CoursSysteme.html
  - Cours de A. Griffaut
    - http://dept-info.labri.fr/~griffault/Enseignement/SE/Cours
  - Cours de H. Bouzourfi, D. Donsez
    - http://www-adele.imag.fr/~donsez/cours/#se

Problématique

- La mémoire centrale est une ressource requise par tout processus
  - Un programme doit être chargé dans la mémoire centrale pour être exécuté
    - *Demarrer_processus(p) ➔ Allouer(taille(p))*
    - *Terminer_processus(p) ➔ Liberer(zone_allouée_à (p))*
  - Un processus a généralement besoin de mémoire dynamique
- Problématique
  - Gérer le partage de la mémoire centrale entre différents processus
  - Gérer l'allocation dynamique de mémoire

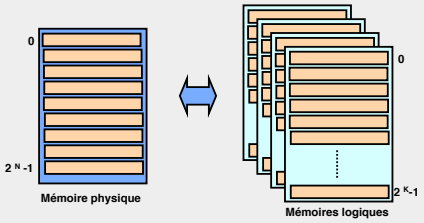
Hiérarchies de mémoire

- Registres (~2 ns)
- Caches (~ 5 ns)
  - L1 on chip (Ko)
  - L2 en multi-cœurs (Mo)
- Mémoire centrale (~ 60 ns)
  - Go
- Mémoires secondaire (~ 12 000 000 ns)
  - To

- Registre: 1 cycle
- L1: 2 à 3 cycles
- L2: dizaine de cycles
- mémoire centrale: centaine au millier de cycles

Fonctions attendues

- **Partage de la mémoire**
  - Fournir une mémoire logique à chaque processus (2^K)
  - Gérer les translations entre les mémoires logiques et la mémoire physique (liaisons d'adresses)



Fonctions attendues

- **Mémoire physique :**
  - Suite contiguë de *mots* (8, 16, 32, 64 bits selon processeur)
  - Adressage direct aléatoire
- **Mémoire logique :**
  - Espace logiquement contigu
  - Adressage direct aléatoire de mots
  - Taille maximum = $2^K - 1$ (k = la capacité d'adressage)

Fonctions attendues

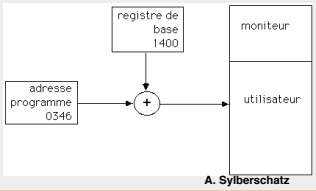
- **Mise en œuvre des mémoires logiques au dessus de la mémoire physique**
  - Swapping (partage de la mémoire dans le temps)
  - Découpage (partitionnement)
  - Multiplexage
- **Protection des espaces logiques**

Concepts

- **Adresse physique**
  - Index d'un mot dans la mémoire physique
- **Adresse logique**
  - Index d'un mot dans la mémoire logique
- **Liaison d'adresses**
  - Transformation d'une adresse logique en une adresse physique
- **La liaison peut avoir lieu à différents moments**
  - À la compilation, au chargement ou à l'exécution
  - Le moment de la liaison d'adresses, ainsi que le type d'adresses sont dépendants des caractéristiques du matériel

Liaison d'adresses

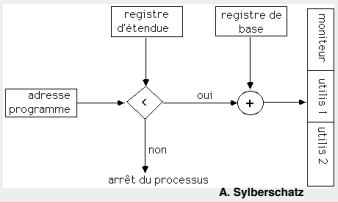
- **Machine avec registre de base**
  - Adresses logiques relatives au début du programme
  - RB contient l'adresse physique du premier mot du programme
  - RB est mis à jour lors de chaque commutation



A. Sylberschatz

Liaison d'adresses

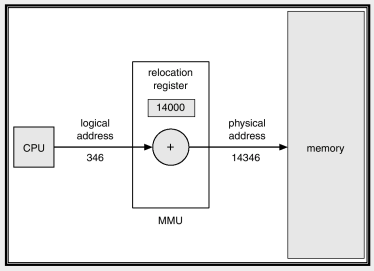
- **Machines avec registre d'étendue**
  - RE permet de contrôler que les accès sont limités à la zone allouée au processus en cours
  - Utile pour la protection



A. Sylberschatz

Liaison d'adresses

- **Machines avec MMU**
  - Dispositif physique rapide

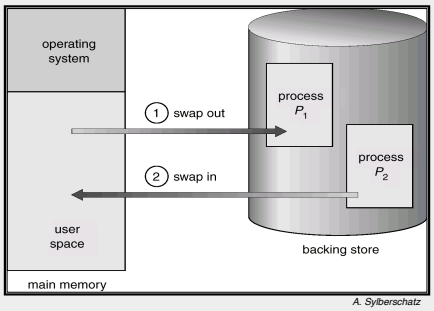


A. Sylberschatz

Swapping / Va et vient

- Technique utilisée en cas de multiprogrammation / temps partagé
- Partage de la mémoire dans le temps
- La mémoire centrale ne peut pas contenir tous les processus en cours d'exécution
- On doit pouvoir vider des processus en mémoire secondaire lorsque ceux-ci ne sont pas actifs

Swapping



A. Sylberschatz

Swapping

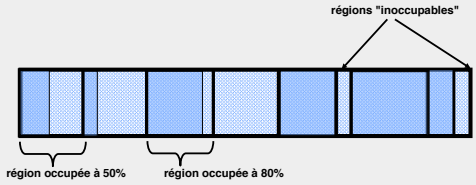
- Le swapping est-il suffisant pour mettre en œuvre les mémoires logiques ?

Les différentes techniques de partitionnement de la mémoire centrale

- **Fonction attendue :**
  - Allouer-zone (t : taille) → adresse
  - Libérer-zone (a : adresse, t : taille)
- **Objectifs :**
  - Optimiser l'utilisation de la mémoire (limiter la fragmentation)
  - Optimiser les algorithmes d'allocation / libération
- **Techniques :**
  - Zones contiguës de tailles fixées
  - Zones contiguës de tailles variables
  - Zones non contiguës de taille fixe (systèmes paginés)
  - Zones non contiguës de taille variable (systèmes segmentés)

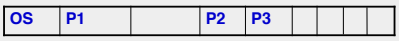
La fragmentation

- Interne : place mémoire laissée libre par un processus dans la région qu'il occupe
- Externe : le taux de régions inoccupables



Allocation en zones contiguës de tailles fixées

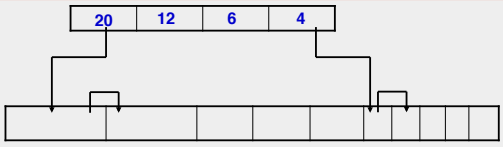
Zones de tailles fixées statiquement:



représentation de la mémoire centrale

→ Le système alloue à P1, P2 et P3 des zones mémoires de tailles supérieure ou égale à leur demande
→Choix des tailles des zones : préfixées ou en fonction d'observations
→Ex: MS-DOS (zones de 64 Ko)

Allocation en zones contiguës de tailles fixées (implantation)



-Tableau de *listes chaînées* de zones libres
-Allouer-zone(t: taille) :
allocation d'une zone de taille la plus proche de t
mise à jour de la liste chaînée
-Libérer-zone (a : adresse, t : taille) :
mise à jour de la liste chaînée

Allocation en zones contiguës de tailles fixées

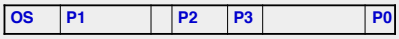
→ Algorithmes d'allocation / libération rapides (si peu de recouvrements)

→ Zones de tailles identiques :
→ Fragmentation interne si taille(processus) << taille(zone)
→ Gestion de recouvrement si taille(processus) >> taille(zone)

→ Zones de tailles différentes :
→ Fragmentation interne si mauvais découpage initial
→ Gestion de recouvrement si mauvais découpage initial

Allocation en zones contiguës de tailles variables

Nombre variable de zones de tailles variables



représentation de la mémoire centrale

→ Le système alloue à P0, P1, P2 et P3 des zones mémoire de tailles égale à leur demande
→ Fragmentation interne?
→ Fragmentation externe?

Allocation en zones contiguës de taille variable (Implantations)

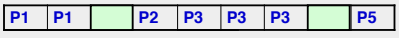
- **Liste chainée (ordonnée) des zones libres**
  - + Pas de mémoire supplémentaire
  - - Temps d'accès aux blocs libres d'une taille ≥ k pour allocation / libération
  - - Libération : calcul d'adresses pour fusionner (peut être fait en asynchrone)
- **Vecteur de bits**
  - + éclatement / fusion automatiques
  - - recherche de k bits consécutifs
  - - Taille du vecteur

Allocation en zones contiguës de taille variable (stratégies d'allocation)

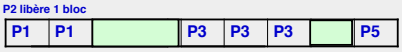
- First-Fit: liste non ordonnée, ou ordonnée par adresse
- Best-Fit: liste ordonnée par tailles croissantes
- Worst-Fit: liste ordonnée par tailles décroissantes
- Exemple:

On suppose que chaque processus demande une zone de taille exprimée en nombre de blocs

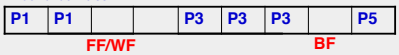
Etat courant: P1: 2 blocs, P2: 1 bloc, P3: 3 blocs, P5: 1 bloc



P2 libère 1 bloc



P4 demande 1 bloc



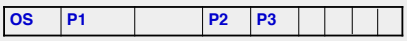
FF/WF BF

Allocation en zones contiguës de taille variable (conclusion)

- En général, First-Fit et Best-Fit meilleurs que Worst-Fit
- Globalement :
  - (+) Simplicité
  - (+) Pas de fragmentation interne
  - (-) Fragmentation externe
  - (-) Temps de gestion des fusions / éclatements

Allocation en zones contiguës de tailles variables contrôlées

Zones de tailles variables contrôlées



représentation de la mémoire centrale

Objectif = accélérer les temps de gestion des fusions / éclatements en utilisant une gestion par subdivision

→Allocation : si aucun bloc libre de taille approximativement égale à la taille demandée, éclater un bloc libre en deux sous-blocs de tailles contrôlées

→Libération : fusionner éventuellement le bloc libéré avec le(s) bloc(s) contigu(s)

→Règles d'éclatement et règles de fusionnement

Allocation en zones contiguës de tailles variables contrôlées

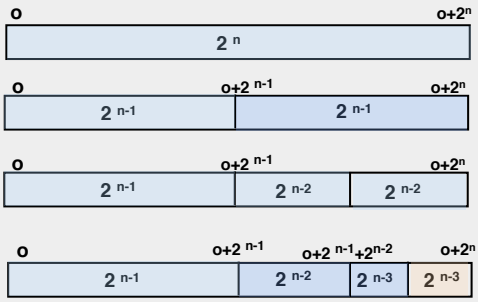
Algorithme Buddy:
éclatement d'un bloc en deux sous-blocs de taille égale (blocs *siamois*) : $2^k = 2^{k-1} + 2^{k-1}$

Origine o — o+2 n-1 — o+2 n-1+2n-2 — o+2n

Adresse du siamois (bloc de taille 2^k et d'adresse a) = a XOR 2^k

48 XOR 8 = 56

Allocation en zones contiguës de tailles variables contrôlées



Algorithme buddy

- Allocation et libération rapides (pas de parcours, le calcul d'adresse du bloc précédent / suivant est direct)
- Fragmentation interne limitée (tailles arrondies à une puissance de 2)
- Fragmentation externe (limitée)

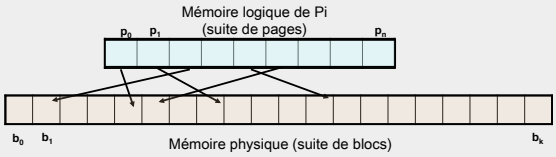
- Variante: fibonacci (Free BSD)

Allocation en zones contiguës (remarques générales)

- **Fragmentation externe**
- **Quelque soit l'algorithme utilisé, il est possible que la place libre devienne insuffisante**
  - Libérer une zone en renvoyant en mémoire secondaire les données qu'elle contient
  - Réorganiser entièrement l'espace mémoire (compactage)
  - Utiliser un ramasse-miettes dans le cas ou des blocs alloués ont pu devenir inaccessibles

Allocation en zones non contiguës de taille fixe (pagination)

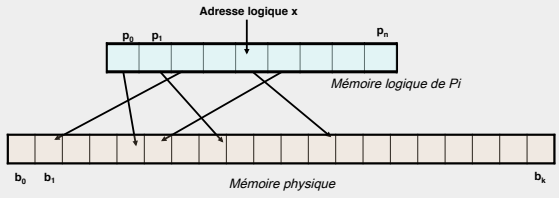
- **Mémoire logique (image mémoire) d'un processus tronçonnée en zones non contiguës de taille fixe (pages)**
- **Objectif : pas de fragmentation externe et fragmentation interne minimale (dernière page)**



Allocation en zones non contiguës de taille fixe (pagination)

- **Accès adresse logique x**

→ Comment savoir où est x dans la mémoire physique ?



- Déterminer à quelle page est associée l'adresse x (p_i)
- Déterminer à quel endroit se trouve x dans la page (déplacement)
- Déterminer dans quel bloc est chargée cette page (bj)

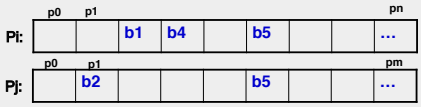
Pagination : adressage

Décomposition d'une adresse logique

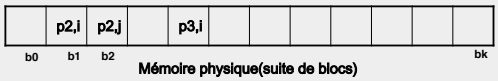
- Mémoire logique de 2^m octets
- Pages de 2^n octets
- Adresses sur m bits

No page sur *m-n* bits de poids fort	Déplacement sur *n* bits faibles

Pagination : partage de la mémoire entre différents processus



Mémoires logiques des processus (suites de pages)



Mémoire physique(suite de blocs)

Librairies

- **Librairies**
  - Définissent des fonctions utilisables par plusieurs programmes
- **Librairies statiques**
  - Suffixées par .a (Unix)
  - Les fonctions utilisées sont inclues dans le binaire exécutable à la compilation
- **Librairies dynamiques (partagées)**
  - Suffixées par .so (Unix)
  - La librairie est placée dans l'espace d'adressage et liée aux autres librairies et au binaire exécutable au moment du chargement (Unix / Linux, DLL Windows)

Librairies partagées (Unix)

- **Solution 1**
  - Tout accès à une variable globale ou à une fonction est effectué au travers d'un adressage absolu
  - Alors, une librairie dynamique L doit toujours être placée à la même adresse virtuelle dans les différents processus qui l'utilisent
- **Solution 2**
  - Tout accès à une variable globale ou à une fonction est effectué au travers d'un adressage relatif (code PIC)
  - Alors, une librairie partagée L peut être placée à différentes adresses virtuelles dans les différents processus qui l'utilisent
  - Mais les accès vers des fonctions *externes* doivent être interprétés en fonction du processus en cours (table d'indirection)

Pagination : implantation

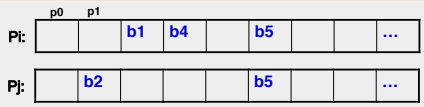


Table des pages de Pi

page	bloc
2	1
3	4
5	5

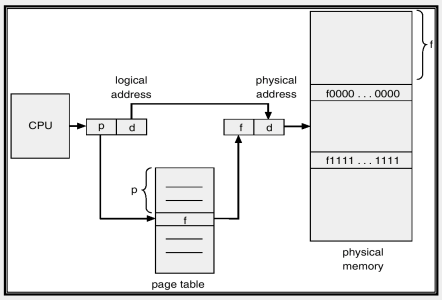
Table globale

bloc	proc	écr	mod
0			
1	j	x	x
2	i	x	
3	i		
4	i		
5	i,j		

Pagination : implantation de la table des pages

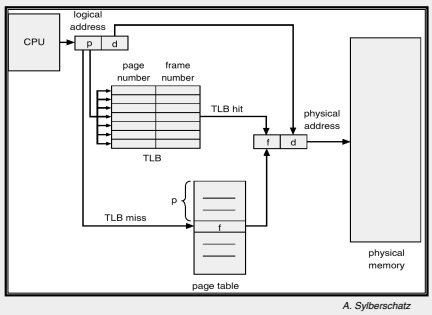
- En mémoire avec un registre pointeur de la table des pages (solution lente)
- Par un ensemble de registres spécialisés changés à chaque commutation de processus (solution rapide)
- En utilisant des registres associatifs (cache d'adresses hardware - TLB)

Implantation de la table des pages



A. Sylberschatz

Implantation de la table des pages avec TLB



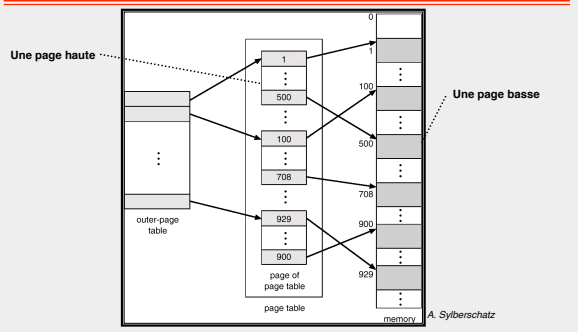
A. Sylberschatz

Pagination : techniques alternatives

- Techniques alternatives de gestion des tables de pages :
  - Tables des pages à plusieurs niveaux
  - Mémoires associatives
  - Tables inversées

➔ Le problème est de limiter l'espace consommé par la table des pages

Pagination : table des pages à 2 niveaux (exemple)



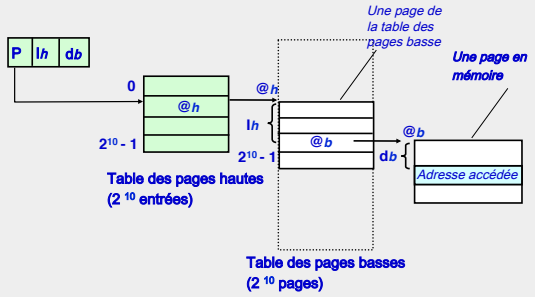
Pagination : table des pages à 2 niveaux (exemple)

- Adresse logique : (no page, déplacement)
  n° page basse sur 20 bits, dépl. sur 12 bits
- Table des pages basse de 2^{20} entrées occupe 4 Mo (2^{10} pages)
- Table des pages haute contient 2^{10} entrées
- N° page haute sur 10 bits
- Décomposition
  index page haute sur 10 bits, index page basse sur 10 bits

Page number		Page offset
P	*lh*	*db*
10	10	12

P= index dans la table des pages hautes

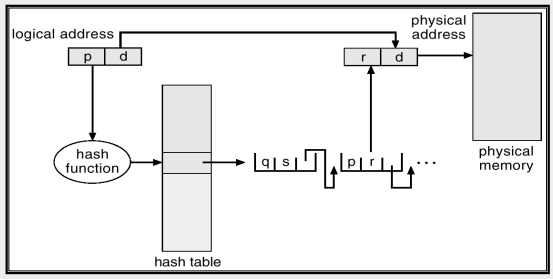
Pagination : table des pages à 2 niveaux (exemple)



Pagination : table de page hachées

- Utilisé lorsque l'espace d'adressage > 32 bits.
- Le numéro de page est haché dans une table des pages dont le nombre d'entrées est réduit.
- Chaque entrée de la table des pages contient une liste chainée.

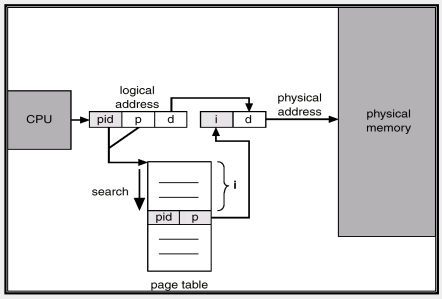
Pagination : table de pages hachées



Pagination : table de pages inversées

- Une entrée par bloc physique
- Pour chaque entrée
  - no page logique, id processus
- Une adresse virtuelle
  - id processus, no page logique, dépl
- Accès mémoire
  - chercher la page correspondante dans la table inversée
- Utilise une table de hashage pour accélérer la recherche de la page correspondante
- Utilisé dans certaines architectures 64 bits (IBM, HP)

Pagination : table de pages inversées



Pagination à la demande

- Placement partiel des images mémoire des processus
- Pages chargées au moment de leur accès (si nécessaire)
- Pages non chargées stockées sur disque (mémoire de réserve)

➔ Gestion du défaut de page

Pagination à la demande

- **Matériel nécessaire**
  - Un bit de présence dans la table des pages
  - Un disque rapide
- **Gestion de défauts de pages**
  - Stopper temporairement l'instruction
  - Trouver un bloc libre (algorithme de remplacement)
  - Charger la page
  - Relancer l'instruction

Algorithmes de remplacement

- **Remplacement**
  - Sélectionner une page victime
  - Enregistrer la page victime dans le disque si nécessaire
  - Modifier la table des pages
- **Algorithmes de sélection de la page à libérer**
  - OPTIMUM
  - RANDOM
  - LRU (Least-Recently Used)
  - FIFO
  - Seconde chance

→ algorithmes classiques de gestion de cache

Principe de localité

- **Propriété des 90%-10%**
  - Un programme passe 90% de son temps à exécuter seulement 10% de ses instructions
- **Propriétés de localité :**
  - Temporelle
  - Spatiale
- **Propriétés exploitées pour**
  - Algorithmes de remplacement
  - Algorithmes de pré-chargement

Exemple algorithme Optimum

Pages adressées :

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Mémoire de 3 blocs :

b1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
b2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0
b3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1

Exemple FIFO

Pages adressées :

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Mémoire de 3 blocs :

b1	7	7	7	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
b2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	0	0
b3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	1

Exemple LRU

Pages adressées :

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Mémoire de 3 blocs :

b1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
b2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
b3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Algorithme LRU (implantation)

- Page → date (tableau de n bits)
- Accès (page) → décalage vers la droite de toutes les dates (1 pour la page accédée, 0 pour les autres)
- Date min = page la moins récemment accédée

Accès	Date Page0	Date Page1	Date Page2	Ordre pages /date
	000	000	000	
Page 0	100	000	000	P0,P1=P2
Page 1	010	100	000	P1,P0,P2
Page 2	001	010	100	P2,P1,P0
Page 1	000	101	010	P1,P2,P0

Classement des algorithmes

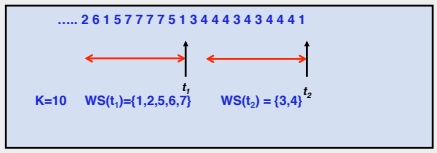
- OPTIMUM
- LRU
- Seconde Chance
- FIFO
- RANDOM

Remplacement global

- La page à déchargée est prise à un processus quelconque
- Trashing du système si :
  - Trop de processus présents en mémoire centrale
  - Les processus n'ont pas assez de place chacun
  - Ils font beaucoup de fautes de pages → diminution du taux d'utilisation du processeur (attente des E/S)
  - Diminution du taux d'utilisation du processeur → augmentation du nombre de processus en mémoire centrale
  - etc

Remplacement local

- La page à décharger est prise au processus qui a fait la faute
- Allocation d'un nombre de blocs constant à un processus
- Détermination du nombre de blocs : notion de *working set*
  - Indices des pages adressées lors des *K* derniers accès
  - Exemple :



Optimisation par pré-chargement de pages

- Une page peut être chargée sans qu'elle soit demandée
- Principe de localité spatiale
- Swapping-in d'un processus

Optimisation par l'usage d'un pool de blocs libres

- Accélère le traitement d'un défaut de page
- Libération de blocs dès que le nombre de blocs libres est proche d'un seuil donné (démon)

Mémoire virtuelle

- Mémoire logique beaucoup plus grande que la mémoire physique
- Taille de la mémoire virtuelle déterminée par la taille des adresses
  - Adresses sur n bits → Mémoire de 2^n bits
- Requiert la pagination à la demande
- Fournit une mémoire virtuelle par processus

Mémoire virtuelle