Université Ibn Tofail
جامعة ابن طفيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⴰⵢⵍ
Ibn Tofail University

Système d'Exploitation
-Gestion de la Mémoire-

Med. AMNAI

Filière SMI-S4

Département d'Informatique

2023-2024

Plan

Plan

Plan

Plan

Plan

Objectifs

Gestion efficace des ressources critiques du système :

- Allouer de la mémoire au processus (éviter le gaspillage) ;
- Connaître les zones libres de la mémoire physique ;
- Récupérer la mémoire à la terminaison d'un processus ;
- Offrir aux processus des services de mémoire virtuelle, de taille supérieure à la mémoire physique technique de va-et-vient (**swapping**) et de **pagination**.

Mémoire et Multiprogrammation

- Contexte :
  - **Nombre arbitraire** de processus ;
  - Plusieurs processus **simultanément** en mémoire ;
  - Une **seule mémoire** pour tout le monde ;
  - **Usage dynamique** de la mémoire ;
- Attentes :
  - Sécurité : seul le processus propriétaire d'une zone mémoire peut en lire le contenu ;
  - **Intégrité** : un processus ne peut modifier (volontairement ou non) la mémoire d'un autre processus ;
  - **Disponibilité** : le système doit satisfaire un maximum de demandes de mémoire.

Translation (relocation)

- Le même programme doit pouvoir s'exécuter à différent endroits de la mémoire.
  - Sans être ré-écrit ni re-compilé (Abstraction).
  - L'**emplacement** mémoire **peut même varier** au cours de l'**exécution** (**swap**)

Fragmentation de la mémoire

- **Qst** : Trouver de l'espace mémoire pour charger un processus ?

- La mémoire est découpée en zones de tailles fixes ;
- Problèmes :
  - **fragmentation** : lorsque la mémoire disponible est inutilisable car non contiguë ;
  - **Fragmentation interne** : certains processus peuvent ne pas utiliser toutes la mémoire qui leur est allouée ;
  - **Fragmentation externe** : ils se peut qu'il n'y est pas de segment continue de mémoire pouvant accueillir un processus alors que la somme de l'espace dans les zones libres serait suffisant ;

Adresses logique / Adresses physiques

- L'utilisation « **optimale** » du système peut requérir plus de processus qu'il n'en tient à un instant donné dans la mémoire ;
  - **Idée** : étendre la mémoire vive en utilisant une mémoire de stockage, plus volumineuse, mais plus lente ;
  - **Problème** : transférer toute la zone mémoire utilisée par un processus peut être coûteux.
- Solutions générale (logiciel, **matériel**)
  - **adresses logiques** vues par le programme / le compilateur ;
  - **adresse physique** vues par le système et le matériel ;
- Solution matériel
  - En mode **utilisateur**, les instructions manipulent des **adresses logiques**. Le **MMU** (Memory Management Unit) qui assure la traduction en adresses physiques.
  - En mode **kernel**, les instructions manipulent directement les **adresses physiques**.

Représentation par Table de bits

Par bitmap : A chaque **unité (ou bloc)** de **mémoire**, on fait correspondre un **bit (0 ou 1)** dans la **table** de bits :

Mémoire

A	A	A	|||	|||
|||	|||	B	B	B
|||	|||	|||	C	C
C	C	C	|||	|||
D	D	D	D	|||
|||	|||	E	E	E

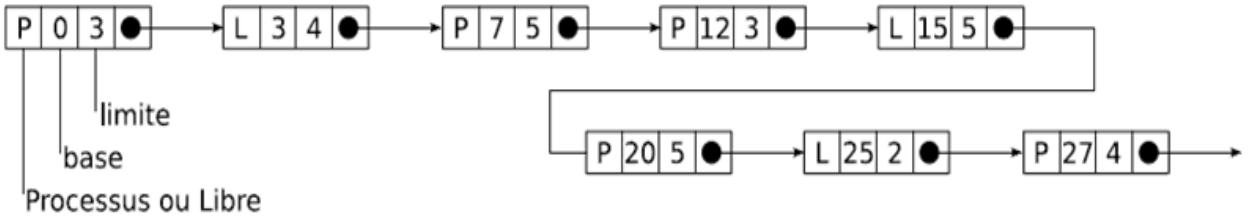
Table de bits

1	1	1	0	0
0	0	1	1	1
0	0	0	1	1
1	1	1	0	0
1	1	1	1	0
0	0	1	1	1

- Chaque unité est un mot(s) ou KO(s) ;
- Dimension de la table de bits = **(taille (M.P))/(taille Unité)**
- La table est découpée en blocs d'allocation généralement de taille 2^n ;

Représentation par Liste Chainée

Listes chaînées : construire et maintenir une liste de blocs de mémoire libres et occupés

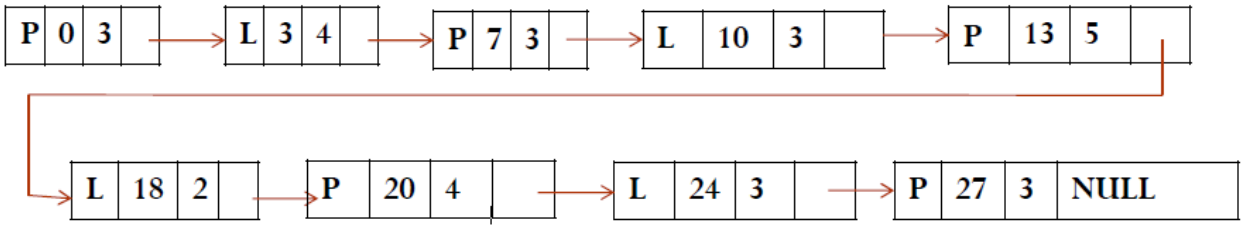


- Chaque bloc est une suite d'adresses consécutives : Toutes occupées par un même processus ou toutes libres.
- Chaque bloc est une structure composé de :
  - **L** (libre) et **P** (occupée).
  - Adresse de début.
  - Taille : longueur.
- Maintenance plus facile et la recherche est moins coûteuse.

Exemple

A	A	A	|||	|||
|||	|||	B	B	B
|||	|||	|||	C	C
C	C	C	|||	|||
D	D	D	D	|||
|||	|||	E	E	E

Cette configuration peut être représentée comme suit :



Algorithmes de recherche

- **First Fit** : Première zone libre.
- **Best Fit** : Meilleur ajustement.
- **Worst Fit** : Plus grand résidu.

First Fit : Première zone libre

- Liste les trous **par adresse** (à partir du début de la mémoire).
- Le gestionnaire de la mémoire (**MMU**) parcourt la liste des segments à la recherche de **la première zone mémoire libre capable** de contenir le processus (dont la taille est supérieure ou égale à la taille du processus).
- Algorithme très **rapide**, il provoque une **fragmentation** interne de la mémoire.
- **RQ** : **Next Fit** a le même principe mais à partir du dernier bloc alloué (crée un peu plus de fragment que Fist Fit).

Best Fit : Meilleur Ajustement

- Liste les trous **par taille croissante**.
- Le gestionnaire de la mémoire (**MMU**) **recherche dans toute la liste la plus petite zone libre capable** de contenir le processus.
- Algorithme **long** et provoque aussi une **fragmentation** de la mémoire (de très petites zones mémoires libres).

Worst Fit : Plus grand résidu.

- Liste les trous par **taille decroissante**.
- Le gestionnaire de la mémoire (**MMU**) parcourt la mémoire à la recherche de **la plus grande zone libre** qui peut contenir le processus.
- Algorithme **long**.
- **RQ** : First Fit est souvent le plus utilisé.

Exemples

- Si une liste des blocs libres contient les blocs de tailles **10k, 4k, 20k, 18k, 7k, 9k, 12k et 19k**.
- Quelle sera la zone allouée pour les demandes de tailles 12k(A), 10k(B), 9k(C) et 8k(D) en utilisant les algorithmes (First Fit, Best Fit, Worst Fit) ?
- Solution :

	A (12K)	B (10K)	C (9K)	D (8K)
First Fit	20K	10K	10K	10K
Best Fit	12K	10K	9K	9K
Worst Fit	20K	20K	20K	20K

Problématique

Demande de page

Dans le cas de la présence d'un processus dont la taille est supérieure à la taille de la mémoire physique disponible.

- **Solution** : Mémoire virtuelle (pagination, segmentation, segmentation avec pagination).
- **Remarque** : **La taille (MV) >= 2*taille(MP)**

Principe de la Pagination

Demande de page

- La mémoire (**virtuelle et physique**) est divisée en zones de taille fixe, appelées **pages**.
- L'**unité** d'allocation est une **page**.
- La **taille** d'une **page** varie entre **512 octets** et quelques **kilos octets**

Exemple de pages

Demande de page

- Etant donner :
  - une mémoire physique de taille 32ko ;
  - une mémoire virtuelle de taille 64ko ;
  - la taille d'une page est de 4 ko.
- Alors :
  - le nombre de page de la MP = (32ko)/(4ko)= 8 pages
  - le nombre de page de la MV = (64ko)/(4ko)= 16 pages

Représentation de la Pagination

Représentation des mémoires physique et virtuelle avec pagination.

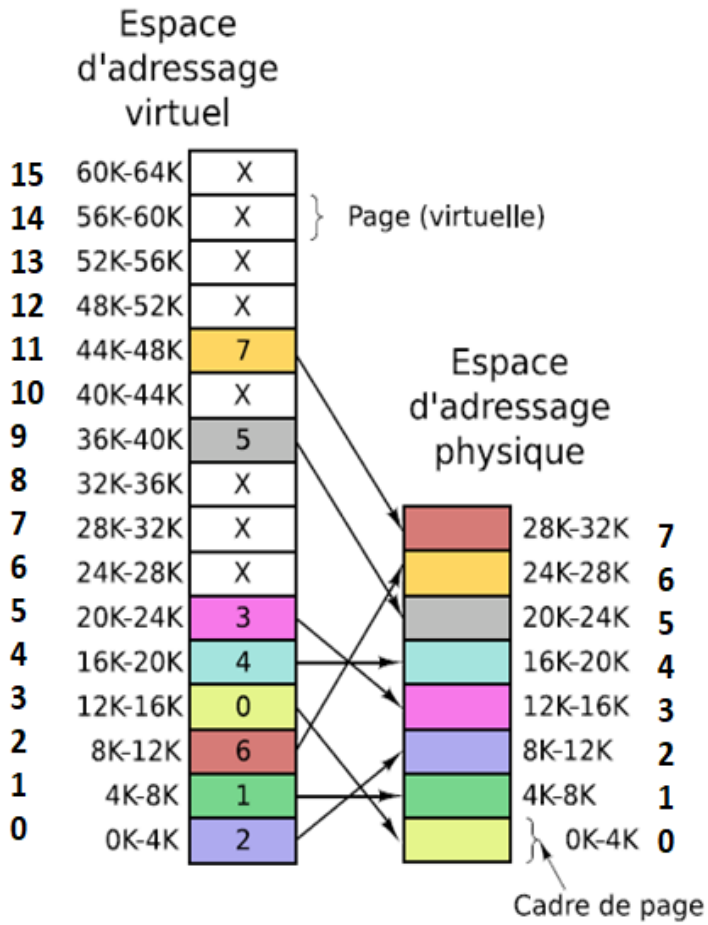
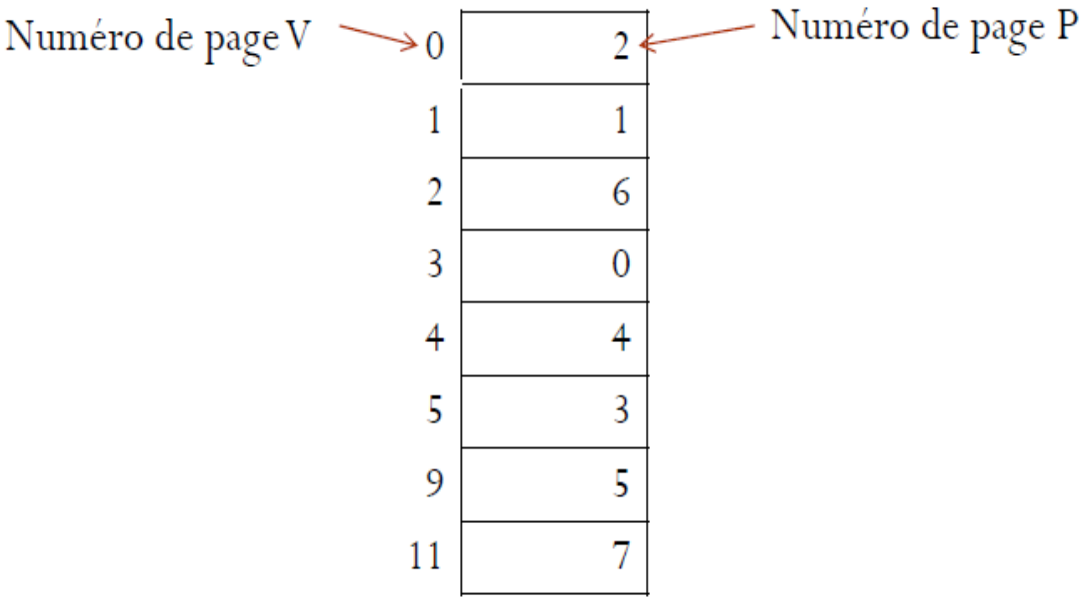


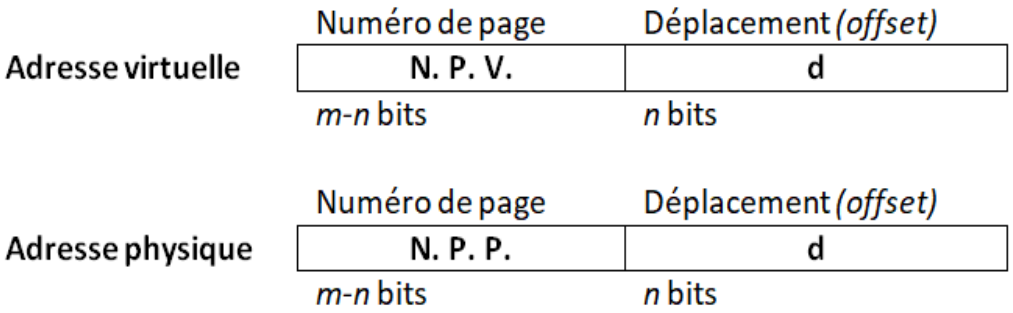
Table de Pagination

Table de pages décrivant la correspondance entre les adresses virtuelles et physiques (selon l'exemple précédent).



Adresses Physique/Virtuelle

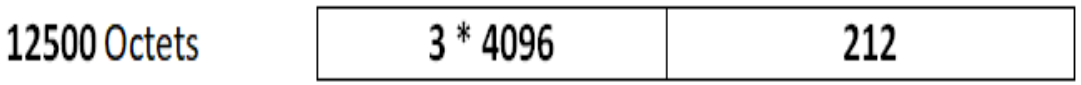
- Une **adresse virtuelle** est composée du numéro de **page virtuelle plus le déplacement** dans celle-ci.
- Pour chaque adresse virtuelle est associée une **adresse physique** composée du **numéro de page physique et d'un déplacement**.
- Le **déplacement** dans la page **physique** est **égale** au **déplacement** dans la page **virtuelle**.



- **Problématique** : Comment trouver l'équivalent d'une page virtuelle vers une page physique ?

Exemple

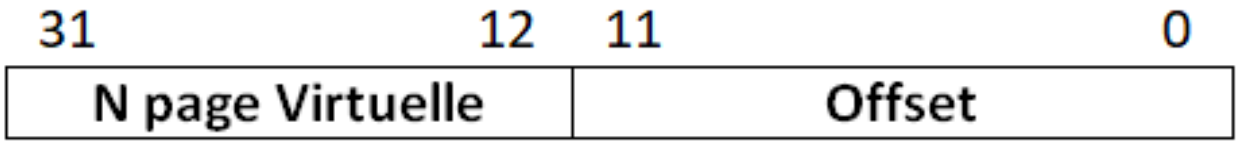
Calculer l'adresse physique qui correspond à l'adresse virtuelle 12500 Octets avec comme taille de page 4Ko ?



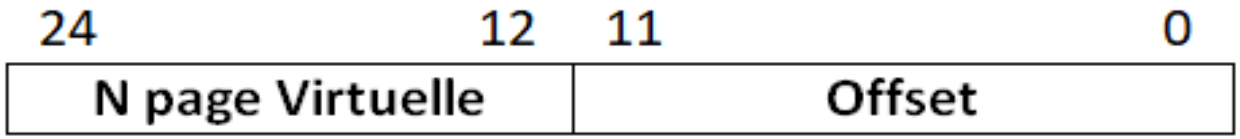
- ==> 12500 = n * 4k + offset (n ?, offset ?)
- n= 12500/4k = 3
- offset = 12500 - (n*4k) = 212

Ecriture des Adresses

- Une adresse virtuelle de **32 bits** pourrait être découpée en un champ de **20 bits** et un champ de **12 bits** :
  - Ceci représente 1 M (2^{20}) pages de 4 Ko (2^{12} O).
  - Ce qui donne un total de 4 Go (2^{32} O).



- Supposons que notre ordinateur a une mémoire physique de 32 Mo (2^{25}) (nombre de pages ?).
- Il aura donc (32M/4k) 8 K (2^{13}) pages de **4 Ko**. L'adresse physique aura donc la forme suivante :



Exemple

Demande de page

- si la taille (Mémoire Virtuelle) = 64 KO = 2^6 x 2^{10} O = 2^{16}O. d'où l'utilisation de 2^{16} adresses virtuelles.
- chacune peut être codée sur 16 bits.
- si la taille (Page Virtuelle) = 4 KO = 2^2 x 2^{10} O = 2^{12} O, ce qui donne 2^{12} déplacement.
- chacun d'entre eux peut être codé sur 12 bits de la façon suivante :

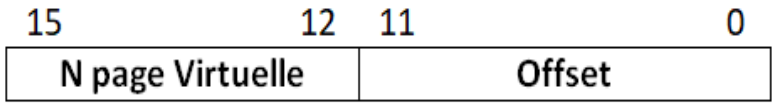


Table de Pages

La table de pages permet de trouver le correspondant d'une page virtuelle vers une page physique.

N° de pages Virtuelle	N page physique	Bits complémentaires
0	2	
1	1	
2	6	
3	0	
4	4	
5	X	
..	..	
15	X	

- **Bits complémentaires** : **présence** (chargéé ou non), **modification** (M), **référence** (R) (utilisée ou non) et les bits de **protection**.

Table de Pages (1)

- Pour chaque index (N de pages Virtuelles) on fait correspondre le **N de pages Physiques** + des **bits complémentaires** + des **bits de protection**.
- Bits complémentaires composés d'**un bit de présence** (**1** si la page virtuelle est chargée en mémoire physique, **0** sinon).
- Les **bits de protection** définissent les droits de *lecture*, *écriture* et *exécution*.

Table de Pages (2)

- Généralement, le nombre de pages Virtuelles est égale au moins 2 fois le nombre de pages physiques ;
- Donc quelques pages Virtuelles **ne seront pas mappées** (n'auront pas de correspondance en mémoire physique) ;
- Si une **page virtuelle** n'est pas présente en **mémoire physique**, il se produit **un défaut de page** ;
- Dans ce cas le système procède comme suit :
  - Repère une page physique peu utilisée ;
  - Recopie son contenu sur le disque, si elle a été modifiée ;
  - Place la page demandée dans la mémoire physique (c-a-d la case libérée) ;
  - Modifier les indicateurs de présence dans la table des pages ;
  - Exécution de l'instruction.

Taille de la Table de Pages

- Taille (TP) = **NB de pages Virtuelle** * (**NB de bits** pour coder un N de **page physique** + bits **complémentaires**).
- **Exemple**
  - une mémoire virtuelle de **64Mo**.
  - une mémoire physique de **32Mo**.
  - a taille d'une page est de **4Ko**.
- **Donner la taille de la table des pages ?**
  - Déterminer le nombre de pages virtuelles : **64Mo** / **4Ko** = 64 x 2^{20} / 4 x 2^{10} = 2^{26} / 2^{12} = **2^{14} P.V**.
  - Déterminer le nombre de pages physiques : **32Mo** / **4Ko** = 32 x 2^{20} / 4 x 2^{10} = 2^{25} / 2^{12} = **2^{13} P.Ph.**
  - Déterminer le nombre de bits nécessaires pour stocker Adresse physique : log2(2^{13}) = **13** bits.
- **La taille de la table des pages** (sans bits complémentaires) : **2^{14} x 13** = 2^4 x 13 x 2^{10} = 208 x 2^{10} = **208Kbits = 26Ko**

Demande de page : 1- Algorithme Optimal

- À chaque fois qu'une **page** est accédée, elle est **étiquetée** avec un **label** donnant le **nombre d'instructions à exécuter avant le prochain accès** à cette page (impossible de connaitre les accès futures ! ! ! !) ;
- Lors d'une demande de remplacement de page, **la page avec le label le plus élevé** est choisie pour être déchargée de la mémoire principale ;
- Impossible à implanter : Solution théorique.

2- Algorithme NRU (Not Recently Used)(1)

- Utilise 2 bits : **R** page été **utilisée**, et **M** (*dirty*) page **modifiée** ;
- SE, **régulièrement** (**timer** qui génère une interruption) le bit **R est remis à 0** ;
- Retirer les pages dont **R** est à **0** car elles n'ont pas été utilisées récemment (depuis la dernière mise à zéro).

2- Algorithme NRU (Not Recently Used) (2)

- Cas possibles :
  - *Non accédée*, *non modifiée* (**R=0,M=0**) : il sélectionne une page et la retire.
  - *Non accédée*, *modifiée* (**R=0,M=1**) : il sélectionne une page et la retire (une sauvegarde sur disque de la page retirée).
  - *Accédée*, *non modifiée* (**R=1,M=0**) : il sélectionne une page et la retire.
  - *Accédée*, *modifiée* (**R=1,M=1**) : une sauvegarde sur disque de la page retirée est nécessaire.
- Facile à implanter et assez efficace.

3- Algorithme FIFO

- Liste des pages en mémoire principale suivant leur **ordre d'arrivée** ;
- La page **retirée** est la plus **ancienne** ;
- Très basique et peut **remplacer** des pages **plus utilisées** juste parce qu'elles sont vieilles ;

4- Algorithme de la seconde chance (amélioration FIFO)

- Tenir compte du bit **R** ;
- Si bit **R** à **1** (**accès récent**), le bit **R** est **remis à zéro (0)** et la page est **réinsérée en fin de la liste** des pages ;
- Si tous les bits **R** sont à **1** alors il dégénère en FIFO ;

5- Algorithme LRU (Least Recently Used)

- Enlever la page **la moins récemment utilisée**.
- La page la plus **anciennement utilisée** est **retirée** de la mémoire.
- Un peu plus coûteux, mais excellente approximation de l'optimal.
- Liste chaînée des pages, à mettre à jour à chaque accès à une page.
- Ou bien compteur à incrémenter à chaque accès (matériel).
- **RQ** : Solution la plus performante et la plus utilisée.