

# Mémoire

## Swapping - Adressage Virtuel - Pagination

Programmation Système — R3.05

C. Raïevsky



Département Informatique  
BUT Informatiques 2<sup>ème</sup> année

2023

## Structuration initiale de la mémoire

Rappel État initial

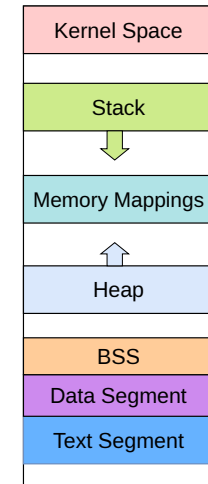
### Zones mémoire prédéfinies par l'OS :

“Statiques” :

- ▶ Text segment (ro)
- ▶ Data segment
- ▶ BSS segment

“Dynamiques” :

- ▶ Heap (Tas, free-store)
- ▶ Stack (Pile)
- ▶ Memory Mapping segment



2 / 25

## Heap != Stack

Rappel Heap & Stack

### Tas – Heap ou Free Store

- ▶ Allocation dynamique de mémoire
- ▶ En cours d'exécution
- ▶ Explicitement par le programme
- ▶ en utilisant `malloc` et *al.*

### Pile – Stack

- ▶ Utilisée pour les appels de fonction
- ▶ Pas de manipulation explicite par le programme (du moins en C)
- ▶ Instructions processeur dédiées
- ▶ Allouée et agrandit par l'OS

3 / 25

## Problème d'espace mémoire

Mémoire physique

Rappel Épuisement de l'espace disponible

### Épuisement de l'espace mémoire disponible

- ▶ Si un processus prend trop de mémoire ou
- ▶ s'il y a trop de processus actifs

### Solutions

- ▶ Va-et-vient – *Swapping*
- ▶ Mémoire virtuelle

4 / 25

## Va-et-vient – *Swapping*

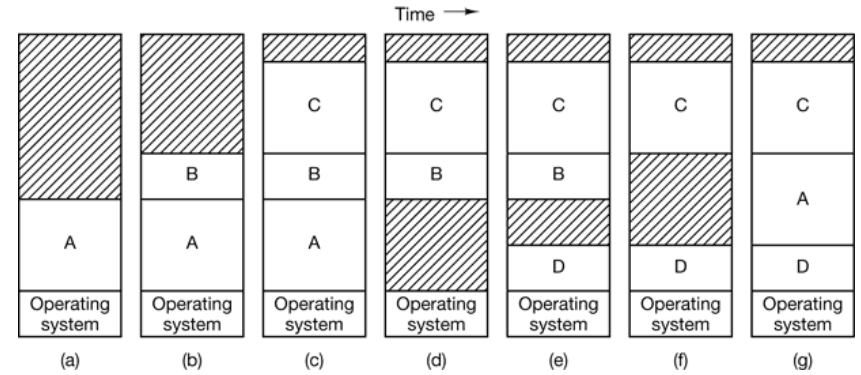
Lorsque un processus est choisi pour s'exécuter, 2 cas possibles :

- ▶ Suffisamment de mémoire disponible : chargement en mémoire
- ▶ Sinon :
  - ▶ déchargement d'un ou plusieurs processus de la mémoire vers le disque dur ;
  - ▶ puis chargement du processus choisi

Un processus est :

- ▶ soit complètement chargé en mémoire
- ▶ soit sur le disque

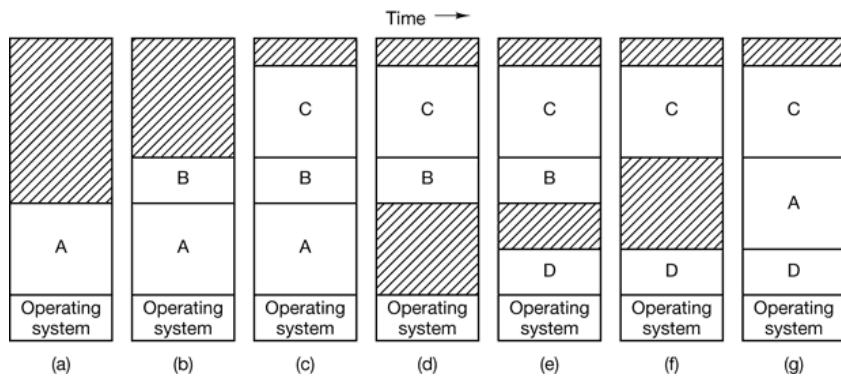
## Exemple de séquence d'exécution de 4 processus



Swapping aux transitions :

- ▶ (c) → (d) → (e) ⇒ A sort, D rentre
- ▶ (e) → (f) → (g) ⇒ B sort, A rentre

## Exemple de séquence d'exécution de 4 processus



Comment gérer les espaces libres et occupés ?

- ▶ tableau de bits
- ▶ liste chaînée

## Gestion de la mémoire libre

Tableau de bit – *bitmap*

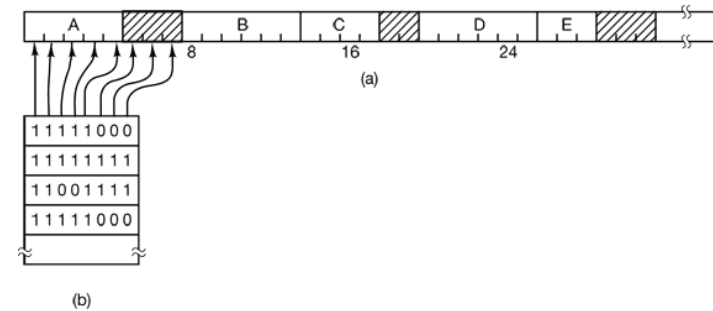
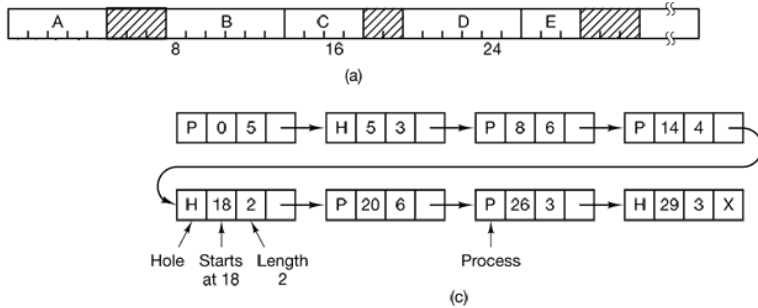


Tableau de bit – *bitmap*

- ▶ Mémoire divisée en blocs
- ▶ Bit à 1 si le bloc est occupé par un processus
- ▶ **Problème** : recherche d'un espace libre longue

## Gestion de la mémoire libre

## Liste chaînée



## Liste chaînée

- ▶ Chaque segment mémoire est représenté par un élément de la liste
- ▶ Amélioration de la recherche d'espace libre
- ▶ **Un problème persiste** : quel espace libre choisir ?

9 / 25

## Choix d'un espace libre

(1/2)

## Principaux Algorithmes

Premier espace libre – *First Fit*

- ▶ Le premier espace libre suffisamment grand pour contenir le processus est choisi

Espace libre suivant – *Next Fit*

- ▶ Le premier espace libre suffisamment grand pour contenir le processus est choisi
- ▶ La prochaine recherche commence à l'adresse de cet espace
- ▶ Meilleures performances que *first fit* en simulation

10 / 25

## Choix d'un espace libre

(2/2)

## Principaux Algorithmes

Espace libre le plus adéquat – *Best Fit*

- ▶ L'espace libre dont la taille est la plus proche de celle demandée est choisi
- ▶ Examen de tous les espaces libres à chaque fois
- ▶ Entraîne plus de fragmentation que les 2 autres à cause de la sciure – *Saw Dust*

## Optimisations

- ▶ Utilisation de plusieurs listes – *Quick Fit*
  - ▶ suivant le type (occupé, libre)
  - ▶ suivant la taille
- ▶ Classement des listes par taille pour accélérer la recherche

11 / 25

## Problème d'espace, encore. . .

Cette technique (*swapping*) conserve des problèmes :

- ▶ Si un processus prend plus de mémoire que disponible
- ▶ Charger et décharger tout un processus prend beaucoup de temps

## Solution :

Mémoire virtuelle + pagination

12 / 25

## Exemple d'adresses mémoire virtuelles

```

1 int g;
2
3 int main(int argc, const char *argv[])
4 {
5     printf("&g: %#010x\n", &g);
6     int j;
7     int k;
8     printf("&j: %#010x\n", &j);
9     printf("&k: %#010x\n", &k);
10    printf("&j-&k: %d\n", &j - &k);
11
12    // Memory allocation on the heap
13    int* px1 = malloc(100 * sizeof(int));
14    printf("px1: %#010x\n", px1);
15
16    int* px2 = malloc(42 * sizeof(int));
17    printf("px2: %#010x\n", px2);
18
19    printf("px2-px1: %d\n", px2 - px1);
20
21    return 0;
22 }
    
```

Comment être sûr que les adresses ne seront pas les mêmes ?

### 2 exécutions

```

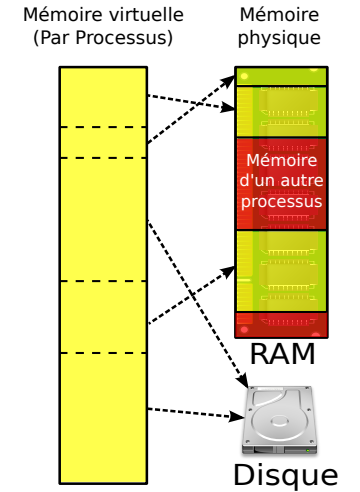
&g : [0xb]
&j : [0x1]
&k : [0x1]
&j - &k:
px 1 : [0]
px 2 : [0]
px2 - px1
px1 - &g:
-----
&g : [0x2]
&j : [0x0]
&k : [0x0]
&j - &k:
px 1 : [0]
px 2 : [0]
px2 - px1
px1 - &g:
    
```

## Adressage mémoire

Mémoire Virtuelle

### Chaque processus a un espace d'adressage VIRTUEL

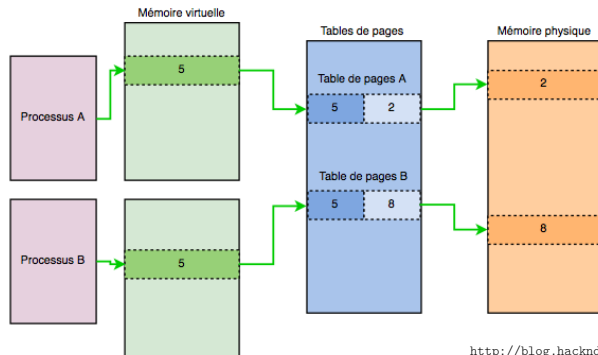
- ▶ Évite les conflits d'adresses
- ▶ Facilite grandement la multiprogrammation
- ▶ Permet de mettre en place des mécanismes de protection
- ▶ Nécessite un mécanisme de mise en correspondance



## Adressage Mémoire

Correspondance adresse virtuelle ↔ adresse physique

- ▶ Mémoire virtuelle et physique découpées en pages
- ▶ Chaque processus possède une **TABLE DE PAGE**
- ▶ qui contient la correspondance



<http://blog.hackndo.com/gestion-de-la-memoire/>

## Pourquoi des pages ?

### Si correspondance directe adresse virtuelle – adresse physique

- ▶ Chaque mise en correspondance consomme l'espace d'une adresse virtuelle et l'espace d'une adresse physique
- ▶ ⇒ La table de correspondance consomme 2/3 de la mémoire. . .

De nombreux périphériques sont accédés par blocs, qui sont mis en correspondance avec les pages

### On parle de :

- ▶ "Page" pour les pages virtuelles
- ▶ "Cadre de page" – "Page frame" pour les pages en mémoire physique

## Table des pages

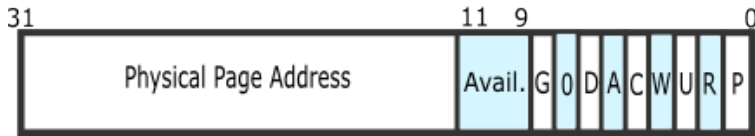
Représentation simpliste

Chaque adresse virtuelle comporte :

- ▶ Un numéro de page virtuelle
- ▶ Un décalage par rapport au début de cette page (*offset*)

Chaque entrée dans la table des pages comporte :

- ▶ Le numéro de page physique auquel elle correspond
- ▶ Un indicateur (*flag*) de validité
- ▶ Des indicateurs d'accès
  - ▶ Est-il possible d'écrire dans cette page ?
  - ▶ Contient elle du code exécutable ?



Source : [wiki.osdev.org/Paging](http://wiki.osdev.org/Paging)

## Exercice sur la table des pages

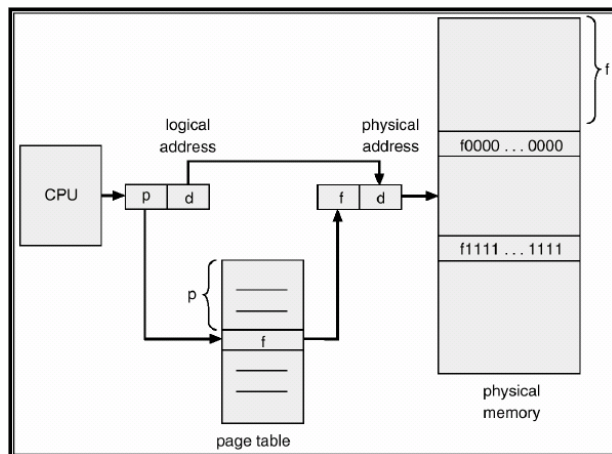
Schématisez un accès mémoire par le cpu utilisant cette représentation

Commencez par :

1. Représenter les deux parties de l'adresse virtuelle
2. Représenter la table des pages
3. Représenter les deux parties de l'adresse physique
4. Représenter la mémoire physique
5. Faites correspondre les différentes parties des adresses aux différents décalages

## Table des pages

Représentation simpliste



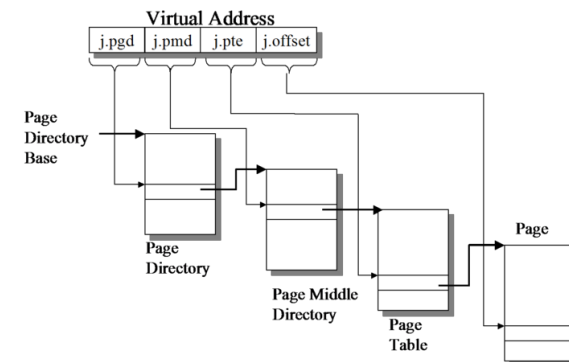
Source : [courses.teresco.org/cs432\\_f02/lectures/11-memory/11-memory.html](http://courses.teresco.org/cs432_f02/lectures/11-memory/11-memory.html)

## Table des pages

Sous Linux

Décomposition hiérarchique des adresses virtuelles

- ▶ 3 décalages dans des tables intermédiaires
- ▶ Un décalage (*offset*) dans la page correspondante



## Défaut de page – Page fault

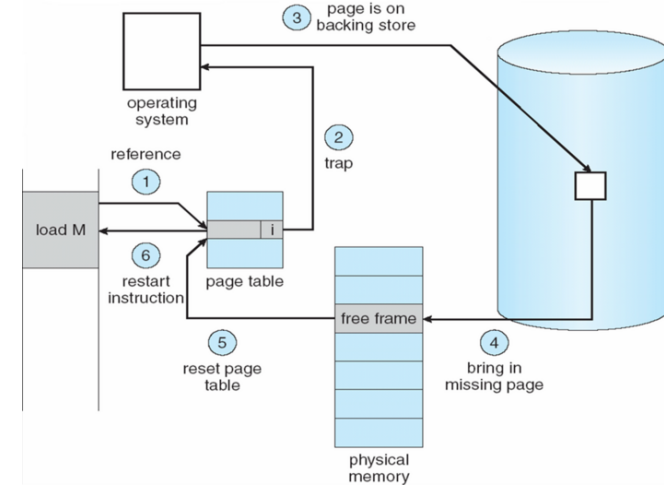
Accès à une page virtuelle qui n'est pas associée à une page physique

Un défaut de page entraîne :

- ▶ Une interruption, l'exécution du processus ne peut pas continuer
- ▶ Le noyau se charge de
  - ▶ charger la page virtuelle accédée
  - ▶ décharger au préalable une page virtuelle associée à un cadre de page, s'il n'y a pas de **cadre de page libre**

Un processus qui cause trop de défauts de page fait du "trashing"

## Défaut de page – Page fault



## Remplacement de pages

Quelle page décharger en cas de *page fault* ?

### The Optimal Page Replacement Algorithm

- ▶ Facile à décrire, impossible à implémenter
- ▶ ↔ Décharger la page qui sera accédée dans le plus longtemps
- ▶ Implique de connaître le futur...

### Not Recently Used – NRU

- ▶ Utilisation de deux bits pour classer les pages :
  1. non accédées, non modifiée
  2. non accédées, modifiées
  3. accédées, non modifiées
  4. accédées, modifiées
- ▶ remise à zéro du bit "accédée" régulièrement
- ▶ Sélection aléatoire dans la classe la plus basse

## Remplacement de pages

Quelle page décharger en cas de *page fault* ?

### FIFO

- ▶ L'OS stocke une liste chaînée des pages chargée
- ▶ dans l'ordre de chargement
- ▶ la tête de la liste est déchargée en cas de *page fault*

Problème : ne tient pas compte de l'utilisation effective des pages

### FIFO avec seconde chance

Identique à FIFO sauf que l'OS examine le bit "accédée" de la page la plus ancienne :

- ▶ Si elle n'a **pas** été accédée, elle est déchargée (FIFO)
- ▶ Si elle a été accédée,
  - ▶ elle est replacée en tête de liste
  - ▶ la recherche continue

## Remplacement de pages

Quelle page décharger en cas de *page fault* ?

### Clock Replacement Algorithm

Idem à FIFO avec seconde chance mais en utilisant une liste circulaire

### Least Recently Used

- ▶ Déchargement de la page la moins utilisée
- ▶ Complicé et coûteux à mettre en œuvre
- ▶ Approximations implémentées : *aging*

### Working Sets

- ▶ Préchargement d'ensembles de pages
- ▶ Basé sur une analyse au runtime des ensembles de pages