Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master GSI
Semestre 1

Systèmes d'exploitation 2

A. ABBAS
abbasakli@gmail.com

Préambule

- **Pré-requis**:
  - Langage de programmation
  - Introduction générale aux systèmes d'exploitation
  - Architecture des ordinateurs
- **Volume horaire hebdomadaire**: 1.5H Cours + 1.5H TD.
- **Évaluation**: continu + Examen
- **Coefficient** : 2, Crédit: 4

Objectifs de l'enseignement

Le but du cours est de présenter le fonctionnement interne d'un système d'exploitation de type Unix.

Pour rendre les choses plus concrètes et à fin de faire la part à l'expérimentation pratique, nous utiliserons principalement le système Linux (une variante d'Unix dont les sources sont libres et très largement diffusés sur Internet).

D'autre part, ce cours propose aussi une introduction aux systèmes temps réels avec l'utilisation de RTLinux.

Contenu du cours

1. **Architecture des Systèmes d'Exploitation**
   a. **Présentation générale** (Rôle du système d'exploitation, Structure générale du système Linux, Organisation du noyau)
   b. **Processus** (Notions de base, Ordonnancement, Clonage)
   c. **Le système de fichiers** (Organisation des fichiers, I-noeuds, Verrouillage de fichiers, Système virtuel de fichiers)
   d. **Gestion de la mémoire** (Allocation mémoire, Espace d'adressage des processus, la swap)
   e. **Signaux** (Emission d'un signal, détournement des signaux, implémentation des signaux :envoie / réception / blocage / détournement)
   f. **Gestion de périphériques** (Principes, mode bloc / mode caractère)
   g. **Modules chargeables** (Principes, Exemples de modules chargeables)
2. **Linux Temps Réel**
   a. **Unix et le temps réel**
   b. **RTLinux**

Partie 1:

Architecture des Systèmes d'Exploitation

Chapitre 1:

Présentation générale des Systèmes d'Exploitation

Présentation générale des Systèmes d'Exploitation

Malgré les différences des points de vue forme, taille et type, les ordinateurs se composent de matériel et de logiciels.

Matériel :

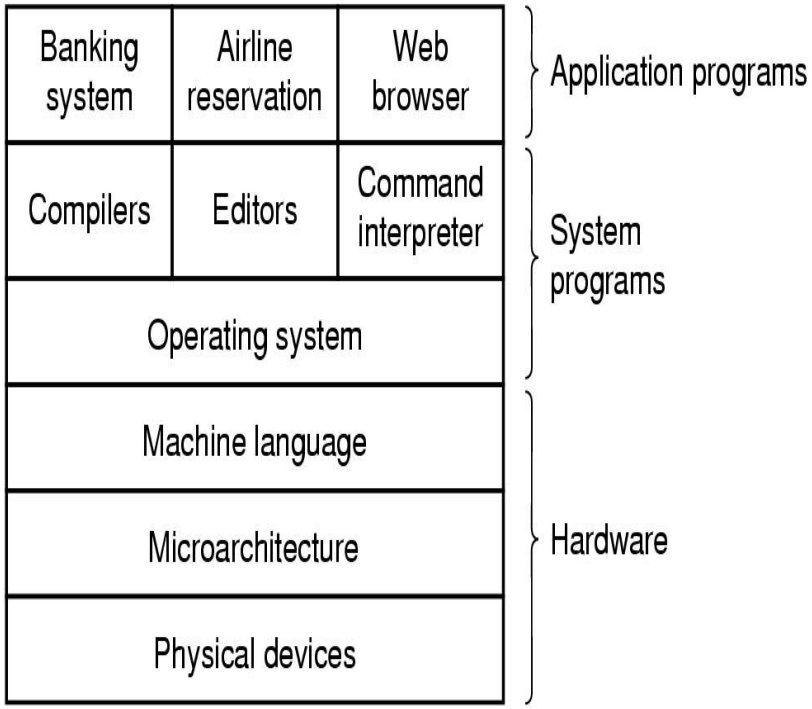
Ecran, clavier, Disque, CPU, Mémoire, Bus, …

Les logiciels :

– Les programmes système :

- les utilitaires (compilateurs, éditeurs, interpréteurs de commandes) ;
- le système d'exploitation

– Les programmes d'application.



Présentation générale des Systèmes d'Exploitation

Le système d'exploitation :

- gère et contrôle les composants de l'ordinateur et
- fournit une base (machine virtuelle) sur laquelle seront construits les programmes d'application et les utilitaires :

services = {appels système}

But :

Développer des applications sans se soucier des détails de fonctionnement et de gestion du matériel.

Présentation générale des Systèmes d'Exploitation

Interface avec le matériel

• Chaque composant (processeurs, mémoires et périphériques) de l'ordinateur a son propre code (câblé ou logiciel) qui assure son fonctionnement et les interactions avec les autres.

• Le système d'exploitation gère et coordonne l'ensemble de ces composants au moyen, notamment, de signaux interruptions.

• Les interruptions permettent au système d'exploitation de reprendre le contrôle :

– Interruptions matérielles :
  - • Horloges (pour gérer l'allocation des processeurs)
  - • Périphériques (pour signaler la fin d'E/S)

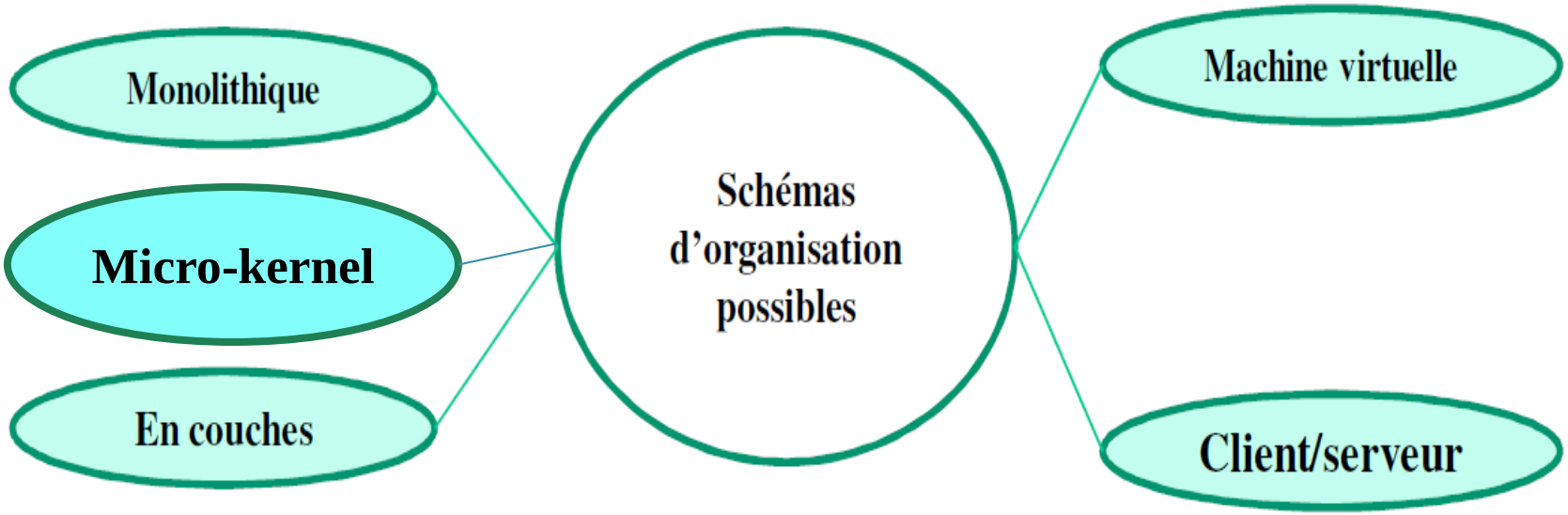
– Interruptions logicielles :
  - • Erreurs arithmétiques (division par zéro)
  - • Données non disponibles en mémoire (défaut de page)
  - • Appels système (invocation du système d'exploitation).

Présentation générale des Systèmes d'Exploitation

Structure d'un système d'exploitation

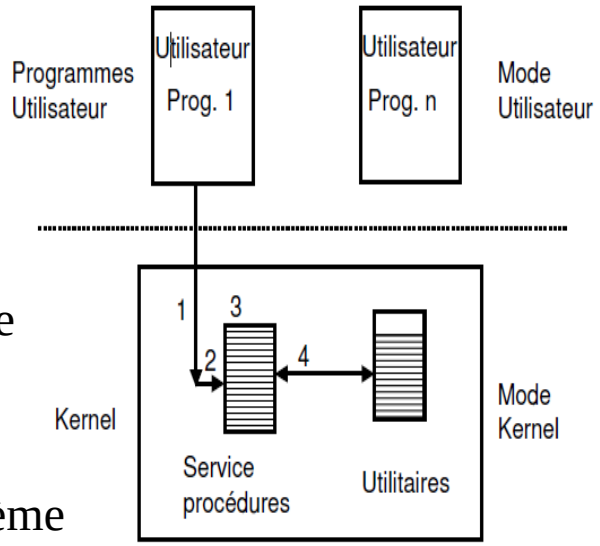
Le système d'exploitation, appelé **noyau** ou **kernel , gère le matériel et** fournit aux programmes une interface d'appels système.

Les appels système permettent aux programmes de créer et de gérer des processus et des fichiers.



Présentation générale des Systèmes d'Exploitation

I. Les systèmes monolithiques



Le SE monolithique est un **ensemble de procédures, chacune pouvant appeler toute autre à tout instant.**

(1) A chaque appel système correspond une procédure de bibliothèque que l'utilisateur peut appeler (bibliothèque standard)

Chaque procédure dépose les paramètres de l'appel système dans un endroit prédéfini comme les registres du processeur, et provoquer une **interruption logicielle** pour passer du **mode utilisateur** au **mode noyau** et activer ainsi le système d'exploitation

(2, 3) Lorsque le système d'exploitation prend le contrôle, il vérifie la validité des paramètres et en déduit la procédure de service à activer pour satisfaire la requête.

(4) Procédure de service appel utilitaires, et puis retourner au mode utilisateur.

Présentation générale des Systèmes d'Exploitation

1. Les systèmes monolithiques

un SE monolithique est organisé principalement en **3 couches :**

C1- Une **procédure principale dans la couche supérieure, qui identifie la procédure de service requise.**

C2- Des **procédures de service dans la couche inférieure à la précédente qui exécutent les appels** système.

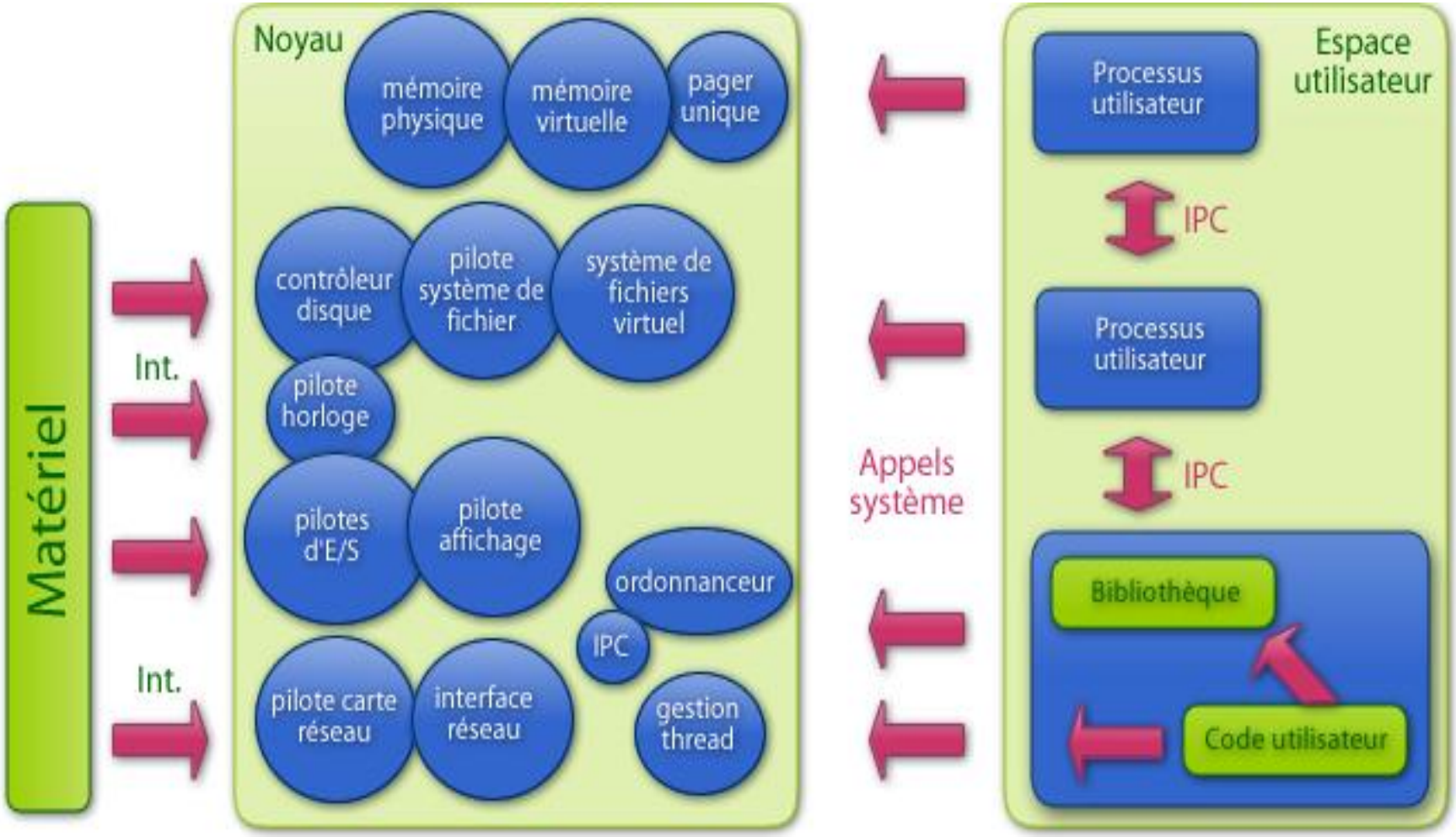
C3- Des **procédures utilitaires dans la couche basse qui assistent les procédures système. Une procédure** utilitaire peut être appelée par plusieurs procédures systèmes.

Exemple :

Linux < 1.2, vieux Unix, OS/360…

Présentation générale des Systèmes d'Exploitation

I. Les systèmes monolithiques



Présentation générale des Systèmes d'Exploitation

1. Les systèmes monolithiques

Certains systèmes d'exploitation, comme les anciennes versions de **Linux, ou certains vieux Unix ont un noyau** monolithique. C'est-à-dire que l'ensemble des fonctions du système et des pilotes sont regroupés dans **un seul bloc de** code et un seul bloc binaire généré à la compilation.

Au fur et à mesure de leurs développements, le code de ces noyaux monolithiques a augmenté en taille et il s'est avéré difficile de les maintenir.

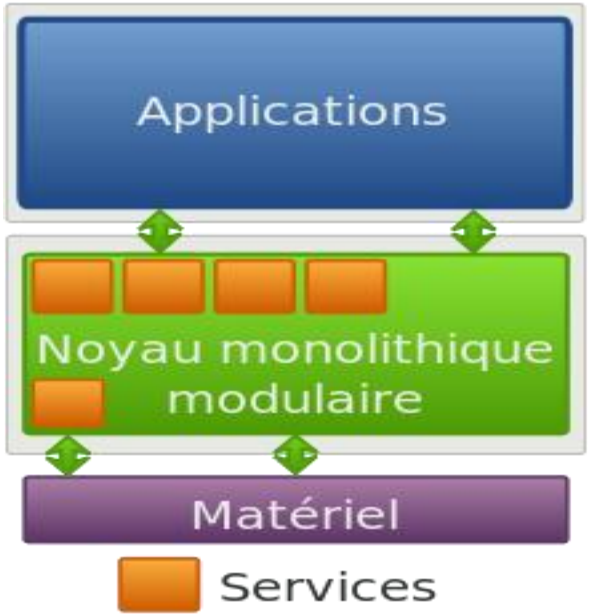
Pour résoudre les problèmes évoqués ci-dessus, les noyaux monolithiques sont devenus modulaires.

Présentation générale des Systèmes d'Exploitation

1. Les systèmes monolithiques modulaires

Dans ce type de noyau, seules les parties fondamentales du système sont regroupées dans un bloc de code unique (monolithique).

Les autres fonctions, comme les pilotes matériels, sont regroupées en différents modules qui peuvent être séparés tant du point de vue du code que du point de vue binaire.



Par exemple avec le noyau Linux, certaines parties peuvent être non compilées ou compilées en tant que modules chargeables directement dans le noyau.

Une erreur dans un module met en danger la stabilité de tout le système.

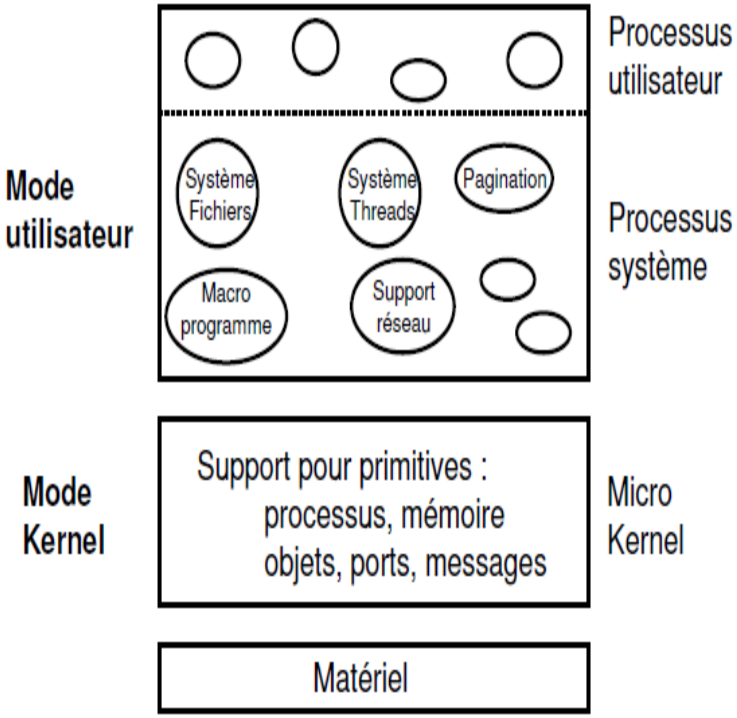
Exemple :

Linux > 1.2, OS/2, Unix SysVr4 / SunOS 5

Présentation générale des Systèmes d'Exploitation

2. Les systèmes à Micro-kernel

Minimiser les fonctionnalités dépendantes du noyau en plaçant la plus grande partie des services du système d'exploitation à l'extérieur de ce noyau

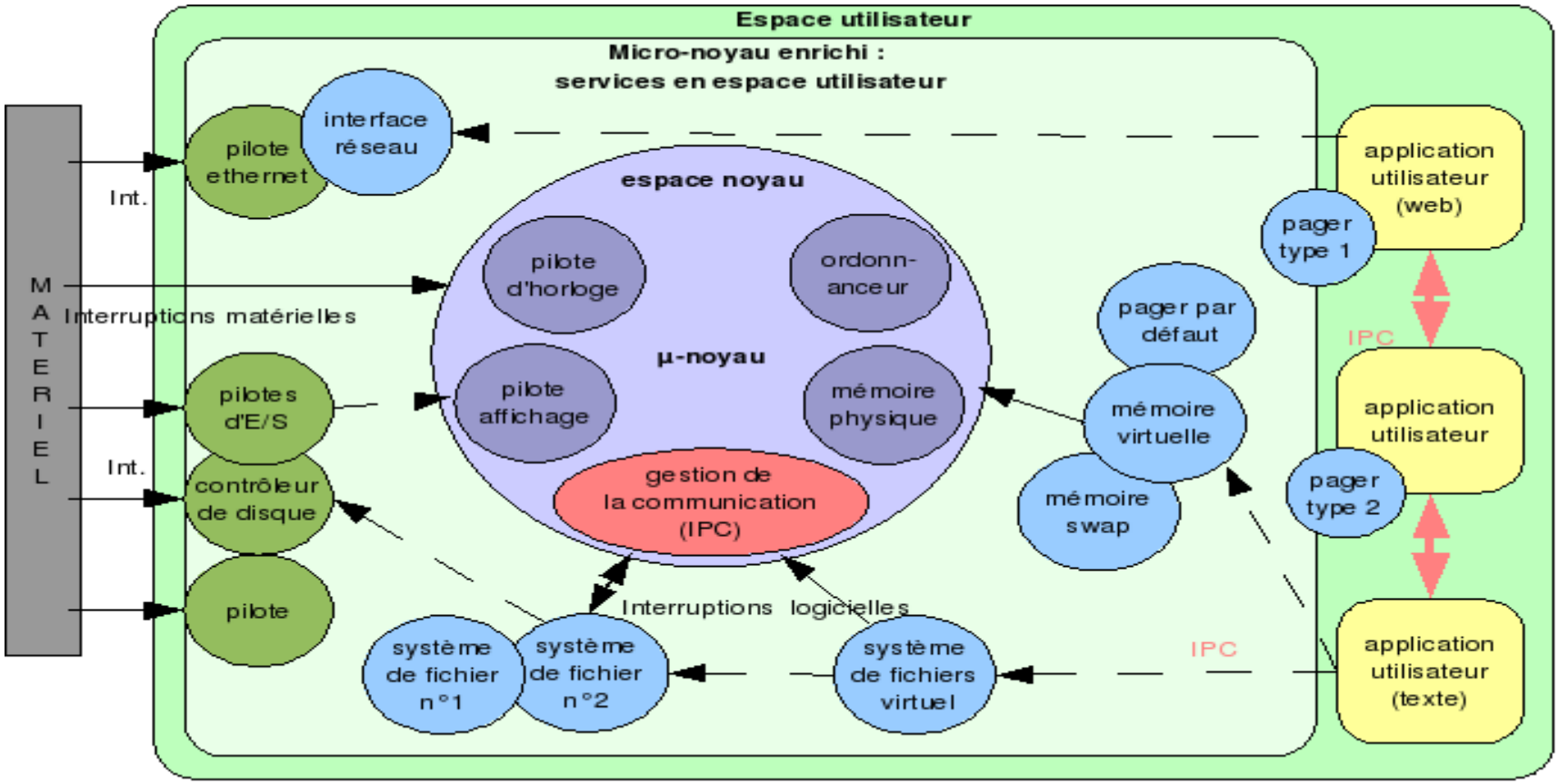


l'architecture micro-kernel divise les services du Système d'exploitation, en « **haut-niveau** » implantés dans le domaine de l'utilisateur et « **bas-niveau** » implantés dans l'espace du mode superviseur.

Exemple :
Mac OS X, Darwin, GNU/HURD,GNU/Mklinux

Présentation générale des Systèmes d'Exploitation

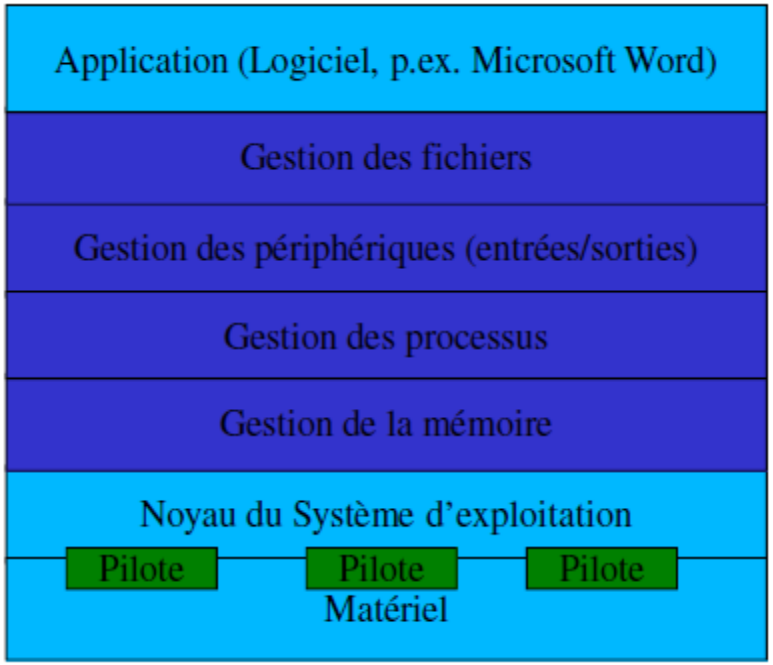
2. Les systèmes à Micro-kernel



Présentation générale des Systèmes d'Exploitation

3. Les systèmes en couches

Chaque couche utilise les fonctions des couches inférieures.



Exemple : SE de Dijkstra (Premier système d'exploitation en couches – 1968) : système de traitements par lots (batch)

Présentation générale des Systèmes d'Exploitation

4. L'architecture client/serveur

Dans le modèle **client/serveur le système d'exploitation est composé** d'un noyau et d'un ensemble de serveurs.

Le noyau gère la communication entre les **clients et les serveurs.**

Les clients (processus utilisateur) sont les demandeurs de services.

Processus serveur effectue le travail et renvoie une réponse.

Les **serveurs** s'exécutent en mode utilisateur. Ils ne peuvent donc pas accéder directement au matériel. Par conséquent, une erreur ou un bogue dans le serveur n'affectera pas, en général, l'ensemble de la machine.

Ce modèle est bien adapté aux **systèmes distribués.**

Présentation générale des Systèmes d'Exploitation

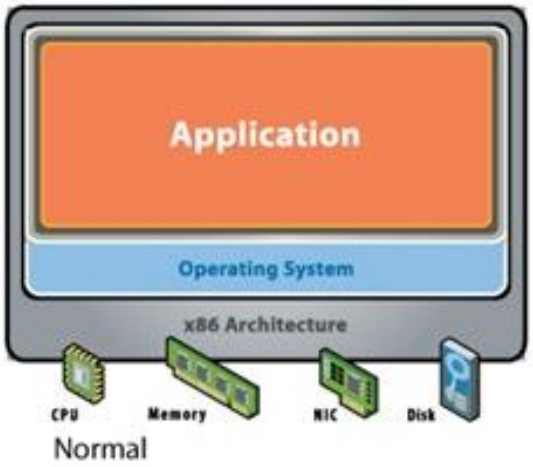
5. Les machines virtuelles

Une **machine virtuelle (*virtual machine) est une illusion*** d'un appareil informatique créée par un logiciel d'émulation.

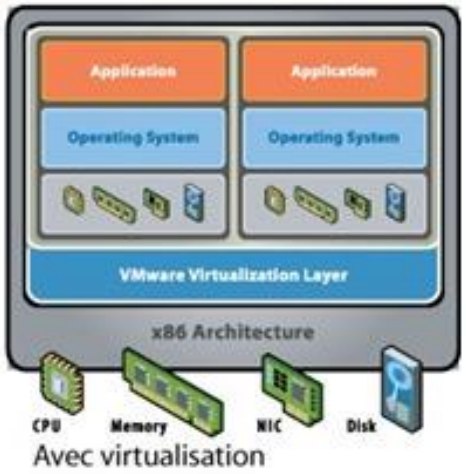
Le logiciel d'émulation **simule la présence de** ressources matérielles et logicielles permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine simulée

Des systèmes d'exploitation différents peuvent tourner au dessus d'une machine virtuelle.

Exemples de machines virtuelles :
JVM (pg. Java),
VMWare (Windows, Linux, OS/2)



Normal



Avec virtualisation

Concepts du processus

Définition

- Un processus est l'entité dynamique représentant l'exécution d'un programme sur un processeur

- Un processus est l'activité résultant de l'exécution d'un programme séquentiel, avec ses données, par un processeur.

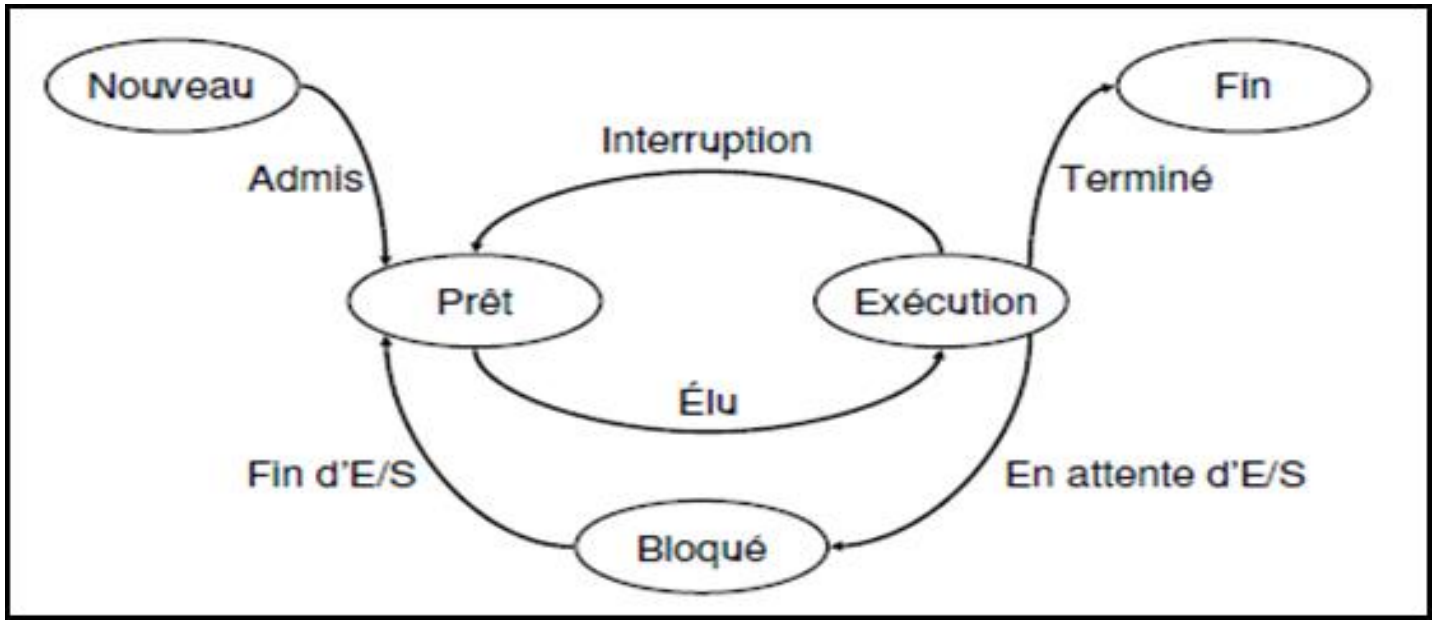
Un processus est caractérisé par:

1. Un numéro d'identification unique (PID);
2. Un espace d'adressage (code, données, piles d'exécution);
3. Un état principal (prêt, en cours d'exécution (élu), bloqué, …);
4. Les valeurs des registres lors de la dernière suspension (CO, sommet de Pile...);
5. Une priorité;
6. Les ressources allouées (fichiers ouverts, mémoires, périphériques …);
7. Les signaux à capter, à masquer, à ignorer, en attente et les actions associées;
8. Autres informations indiquant le processus père, les processus fils, le groupe, les variables d'environnement, les statistiques et les limites d'utilisation des ressources....

Concepts du processus

États d'un processus

Plusieurs processus peuvent se trouver simultanément en cours d'exécution (multiprogrammation et temps partagé), si un système informatique ne comporte qu'un seul processeur, alors, à un instant donné, un seul processus aura accès à ce processeur. En conséquence, un programme en exécution peut avoir plusieurs états. Ces états peuvent être résumés comme suit :



Concepts du processus

États d'un processus

Nouveau : création d'un processus dans le système

Prêt : le processus est placé dans la file d'attente des processus prêts, en attente d'affectation du processeur.

En exécution : Le processus est en cours d'exécution.

Bloqué : Le processus attend qu'un événement se produise, comme l'achèvement d'une opération d'E/S ou la réception d'un signal.

Fin: terminaison de l'exécution

Concepts du processus

Le bloc de contrôle et le contexte d'un processus

Pour suivre son évolution, le SE maintient pour chaque processus une structure de données particulière appelée bloc de contrôle de processus (PCB : Process Control Bloc) et dont le rôle est de reconstituer le contexte du processus.

Le contexte d'un processus est l'ensemble des informations dynamiques qui représente l'état d' exécution d'un processus

Le PCB contient aussi des informations sur l'ordonnancement du processus (priorité du processus, les pointeurs sur les files d'attente)

Numéro de processus
Etat du processus
Compteur d'instruction
Registres
Limites de la mémoire
Liste des fichiers ouverts
...

Ordonnancement des processus

Définition

L'ordonnanceur (scheduler en anglais) est un programme du SE qui s'occupe de choisir, selon une politique (ou algorithme) d'ordonnancement donnée, un processus parmi les processus prêts pour lui affecter le processeur.

Les critères d'évaluation entre les algorithmes d'ordonnancement sont:

Utilisation du processeur : Un bon algorithme d'ordonnancement sera celui qui maintiendra le processeur aussi occupe que possible.

Capacité de traitement : C'est le nombre de processus terminés par unité de temps.

Temps d'attente : C'est le temps passé à attendre dans la file d'attente des processus prêts.

Temps de réponse : C'est le temps passé depuis l'admission (état prêt) jusqu'à la terminaison (état fin).

Ordonnancement des processus

Les algorithmes d'ordonnancement

1. L'algorithme du Premier Arrive Premier Servi (FCFS)
2. L'algorithme du Plus Court d'abord (SJF)
3. Ordonnancement avec priorité
4. L'algorithme de Round Robin (Tourniquet)
5. Ordonnancement avec les d'attente multiniveaux

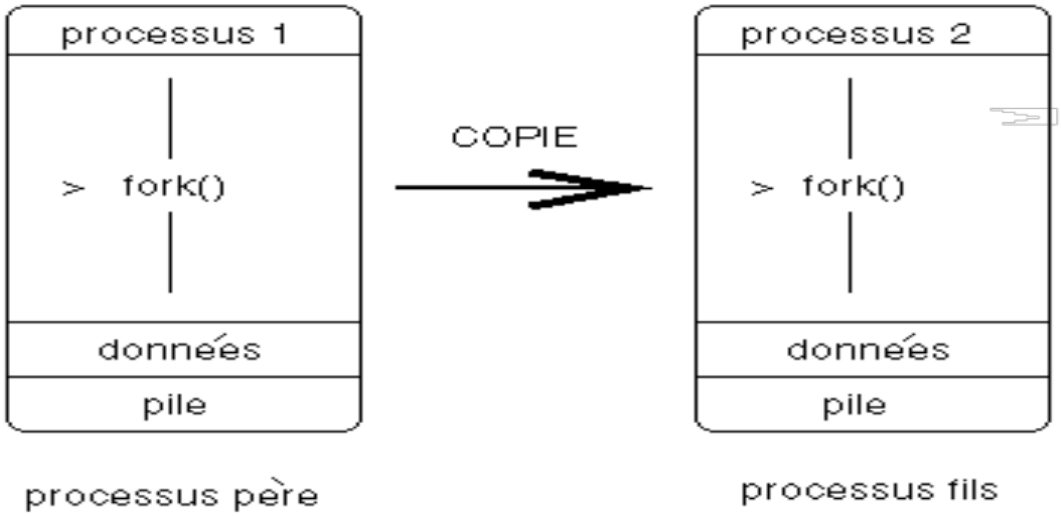
La création de processus

L'appel système ***fork()*** est une façon qui permet de créer des processus aussi appelés **processus lourds**.

Exemple 1 : Instruction de création :

```
#include <unistd.h>
int fork();
```

fork() est le moyen de créer des processus, par duplication (clonage) d'un processus existant. L'appel système fork() crée une copie exacte du processus original.



La création de processus

Pour distinguer le processus père du processus fils on regarde la valeur de retour de fork(), qui peut être:

- La valeur 0 dans le processus fils.
- Positive pour le processus père et qui correspond au PID du processus fils
- Négative si la création de processus a échoué ;

Lors du démarrage de Unix, deux processus sont crées :

1. le Swapper (pid = 0) qui gère la mémoire;
2. le Init (pid = 1) qui crée tous les autres processus.

La création de processus

Exemple 2 :L'exécution de Pgfork.c le père et le fils montrent leur PID.

```
//programme Pgfork.c : appel système fork()
#include <sys/types.h> / * typedef pid_t * /
#include <unistd.h> / * fork() * /
#include <stdio.h> / * pourperror, printf * /
int a = 20;
int main() {
pid_t x;
// création d'un fils
switch (x = fork()) {
case −1 : /* le fork a échoué */
printf("le fork a échoué !");
break;
case 0: /* seul le processus fils exécute ce ( case )*/
printf("ici processus fils, le PID %d.\n ", getpid());
a+ = 10;
break;
default: /* seul le processus père exécute cette instruction*/
printf("ici processus père, le PID %d.\n", getpid());
a += 100;
}
// les deux processus exécutent ce qui suit
printf("Fin du Process %d avec a = %d.\n", getpid(), a);
return 0;
}
```

Le système de fichiers

Le système de fichiers

Information indépendante du processus

Un processus à l'exécution peut stocker une quantité limitée d'information

- Limitation due à l'espace d'adressage (**La capacité de stockage est limitée a la mémoire vive)**

- Quand le processus meurt, cette information disparaît
- Plusieurs processus peuvent avoir besoin d'accéder à la même information
- Les données doivent donc être indépendantes d'un processus donné

Besoins

On doit pouvoir sauvegarder une grande quantité d'information

L'information doit survivre à la terminaison d'un processus

Plusieurs processus doivent avoir accès à l'information de manière concurrente

Le système de fichiers

La solution à ces problèmes est de stocker l'information dans des fichiers

Les processus peuvent lire/écrire/créer des fichiers

La gestion des fichiers est de la responsabilité du système d'exploitation

2 points de vue

Utilisateur :

- De quoi est fait un fichier
- Comment un fichier est nommé
- Quelles sont les opérations possibles…

OS designer

- Comment gérer l'espace disque
- Comment assurer de la fiabilité et de bonnes performances

Le système de fichiers

Nommage des fichiers

La plupart des FS supportent un nommage en 2 parties

- Nom (nombre de caractères variables)
- Extension
  - En général 1-3 caractères (DOS)
  - Plusieurs extensions possibles suivant les FS (.tar.gz…)

En général l'extension n'est qu'une indication

- Certains programmes insistent sur l'extension

Windows associe des opérations aux extensions

- Démarrage de certains programmes

Le système de fichiers

Types de fichiers

Les Os supportent différents types de fichiers

Les fichiers ordinaires (Regular Files) : contient les données utilisateurs

ASCII : facilement manipulables

binaires : format spécifique pour être exécuté

Les répertoires (Directories): sont des fichiers système qui conservent la structure du système de fichiers

Les fichiers spéciaux caractère (Character Special Files) : Liés aux I/O et utilisés pour les périphériques à accès séquentiel

Les fichiers spéciaux bloc(Block Special Files): Utilisés pour représenter les disques

Le système de fichiers

Structure des fichiers

- **Les fichiers peuvent être structures de différentes manières**
  - Suite d'octets
  - Suite enregistrement
  - Arbre

Le système de fichiers

Structure des fichiers

Fichier = suite d'octets

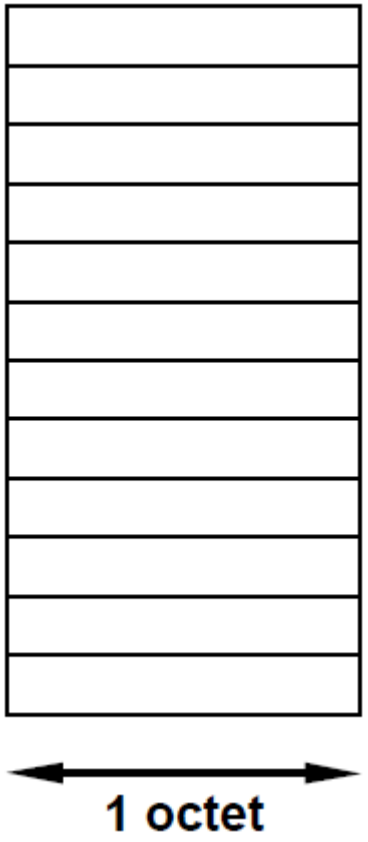
- **Le fichier est une suite d'octets sans structure**

– Le système d'exploitation ne connait pas et ne s'occupe pas du contenu de ce fichier

– Il ne voit que des octets

– Toute signification doit être apporte par le programme des utilisateurs

– UNIX et Windows suivent tout deux cette approche



Le système de fichiers

Structure des fichiers

Fichier= suite d'enregistrements

- **Le fichier est une suite d'enregistrements de longueur fixe**

– **Concept principal** : une opération de lecture renvoie un enregistrement/une opération d'écriture réécrit ou ajoute un enregistrement

  – 80 caractère → carte perforées de 80 colonnes

  – 132 caractères → imprimantes de 132 colonnes

– Les programmes lisaient les données par bloc de 80 caractères est écrivaient par bloc de 132 caractères

– Obsolète

Le système de fichiers

Structure des fichiers

Fichier= arbre

- **Le fichier est un arbre d'enregistrement**

– Les enregistrement n'ont pas la même longueur

– Chaque enregistrement contient une clé dont la position est fixe dans l'enregistrement

– L'arbre est trie en fonction des clés → permet de rechercher rapidement une clé donnée

– L'opération fondamentale ne consiste pas a obtenir le prochain enregistrement, mais obtenir un enregistrement avec une clé donnée

Le système de fichiers

L'accès au fichiers

Accès séquentiel

- **Les premiers systèmes d'exploitation proposaient un seul type d'accès au fichiers : l'accès séquentiel (*sequential access*).**

– Dans ce système, un processus pouvait lire tous les octets ou tous les enregistrements d'un fichier dans l'ordre en commençant au début

– Les fichiers séquentiels étaient pratiques quand le support de stockage était une bande magnétique .

Le système de fichiers

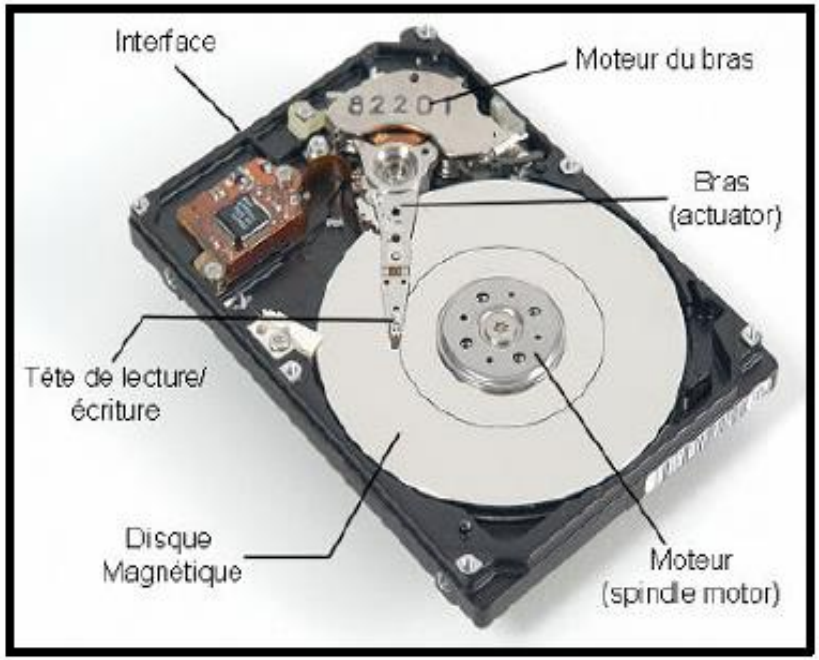
L'accès au fichiers

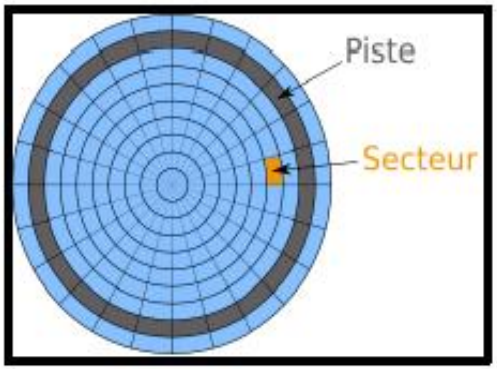
Accès aléatoire

- Lorsque les disques ont servi a l'enregistrement des fichiers, il est devenu possible de lire des octets ou des enregistrement d'un fichier dans n'importe quel ordre
- Les fichiers dont les octets ou les enregistrement peuvent être lus dans n'importe quel ordre sont appelés fichier a accès directe ou accès aléatoire (random access)

Le système de fichiers

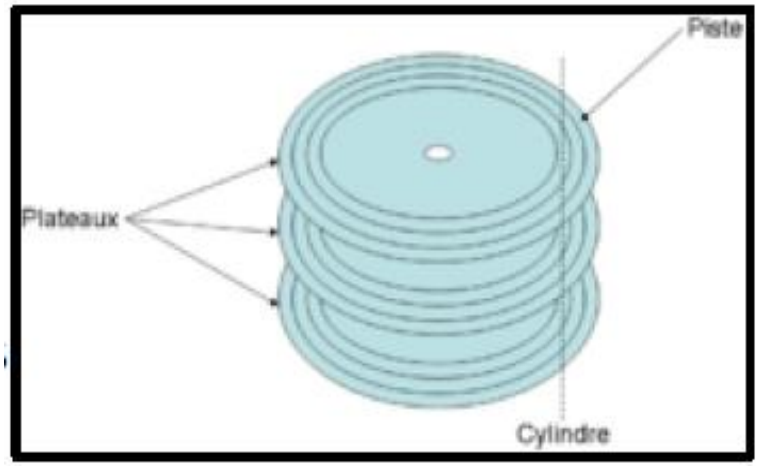
Structure des fichiers





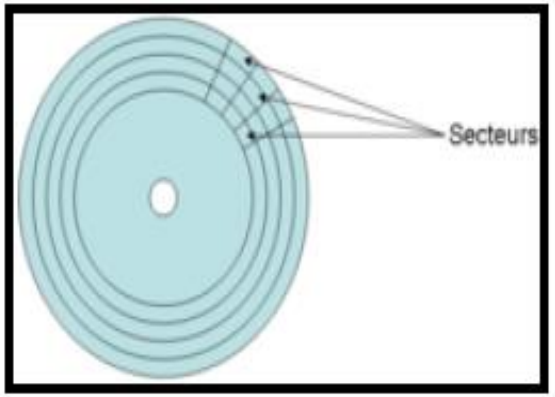
Le système de fichiers

Géométrie d'un disque dur
Adressage CHS : Cylinder/Head/Sector



Cylindre: superposition de plusieurs pistes
– Les pistes au dessus les unes des autres sont accessibles sans bouger les têtes de lecture

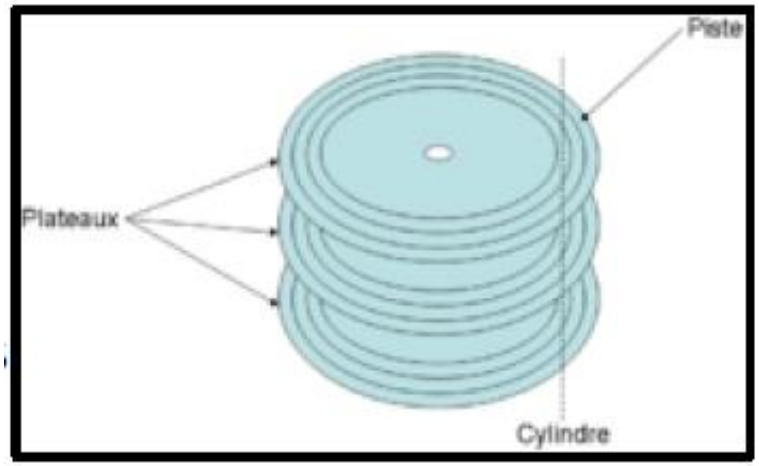
Head:
– Le plus souvent il y a une tête par surface soit deux par plateau
– Le chemin parcouru par une tête particulière sur un cylindre particulier s'appelle une piste



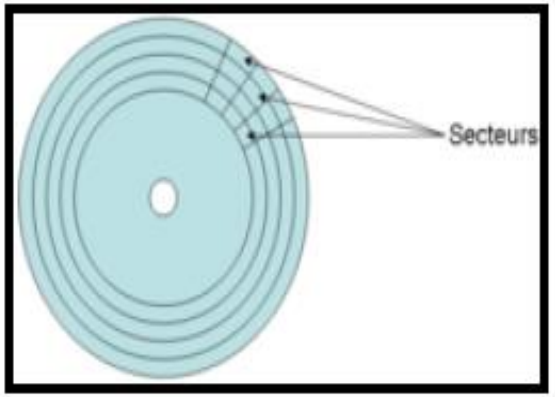
Secteur :
– La piste est divisée en secteurs contenant les données.
– **La taille d'une piste diminue en allant vers le centre du disque → le nombre de secteur par piste diminue aussi en allant vers le centre**
– Taille d'un secteur plus commune= 512 octet

Le système de fichiers

Géométrie d'un disque dur
Adressage CHS : Cylinder/Head/Sector



L'adresse CHS est simplement constituée par l'assemblage des trois composants décrits avant.
– Le tout premier secteur d'un disque est a l'adresse 0 / 0 / 1 : c'est le premier secteur accédé par la première tête positionnée sur le premier cylindre.



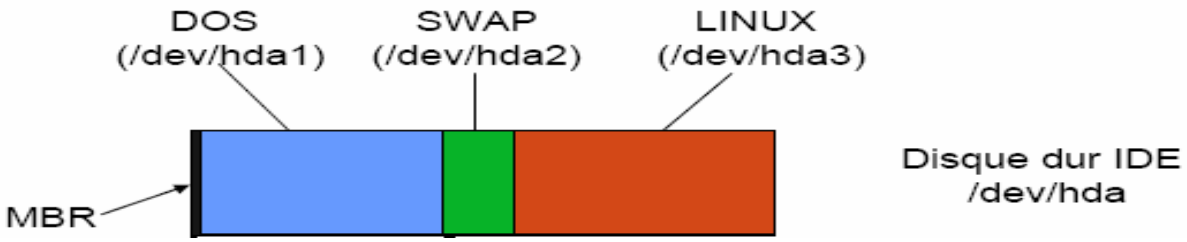
Le suivant sera 0 / 0 / 2 (ce secteur est naturellement atteint juste après par la même tête)

Taille d'un disque =
(Nombre de cylindre) x
(Nombre de piste par cylindre) x
(Nombre moyen de secteur par piste) x
(Taille du secteur)

Le système de fichiers

Partitionnement

Le partitionnement permet la cohabitation de plusieurs OS sur le même disque . L'information sur le partitionnement d'un disque est stockée dans son premier secteur (secteur 0) appelé **enregistrement d'amorçage maitre** MBR (Master Boot Record) sert a amorcer la machine**.**



Deux types de partitionnement :

- **Primaire** : On peut créer jusqu'à 4 partitions primaires sur un même disque.
- **Etendue** : est un moyen de diviser une partition primaire en sous-partitions (une ou plusieurs partitions logiques qui se comportent comme les partitions primaires, mais sont créées différemment (pas de secteurs de démarrage))

Dans un même disque, on peut avoir un ensemble de partitions (multi-partition), contenant chacune un système de fichier (par exemple DOS, Windows et Linux)

Le système de fichiers

Partitionnement

La fin du MBR comprend la table de partitions, laquelle indique l'adresse de début et de fin de chaque partition
– Une de ces partitions est marquée comme étant la partition active

Formatage

Le formatage logique d'un disque permet de créer un système de gestion de fichiers sur le disque, qui va permettre à un système d'exploitation (DOS, Windows, UNIX, ...) d'utiliser l'espace disque pour stocker et utiliser des fichiers.

- **Quand l'ordinateur est amorcé, le BIOS lit et exécute le MBR**

– Le programme MBR détermine la partition active, y lit le premier bloc appelé bloc d'amorçage ou secteur de boot (***boot block***) *et l'exécute.* Ce bloc de boot n'est rempli que si la partition est une partition de démarrage (partition qui contient une copie du noyau). Il contient un petit programme chargeant le noyau (SE) en mémoire et lui donnant le contrôle.

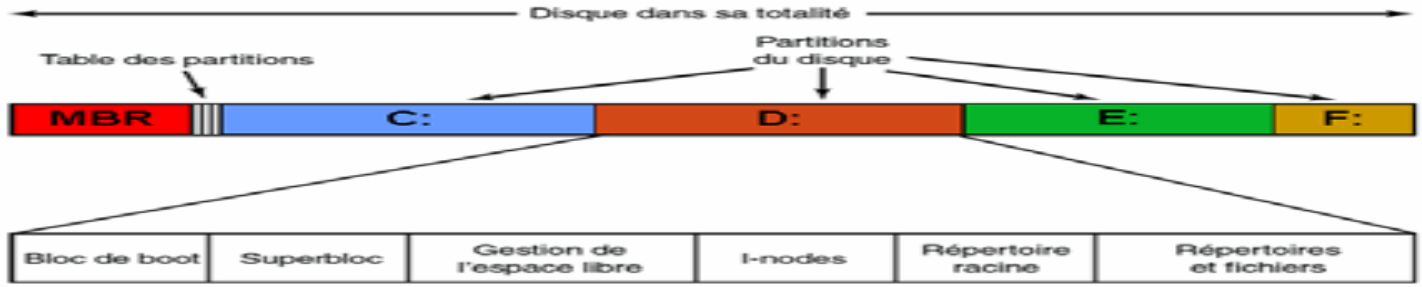
Le système de fichiers

Partitionnement

Apres le bloc de boot, vient le superbloc (*superblock*), *FAT, NTFS ou Ext2*…

– Contient tout les paramètres clé concernant le système de fichier

– L'identifiant du système de fichiers (C:, D : ..), Le nombre de blocs dans le système de fichiers, la liste des blocs libres, l'emplacement du répertoire racine,…



Le système de fichiers

Organisation de l'espace du disque

Taille des blocs

- Les fichiers de données sur les disques se répartissent dans des blocs de taille fixe correspondant à des unités d'entrées-sorties du contrôleur de ce périphérique. La lecture ou l'écriture d'un élément d'un fichier impliquera donc le transfert du bloc entier qui contient cet élément.
- Le compromis habituellement adopte consiste a prendre des blocs de 512 octets, 1 Ko ou 2 Ko. Le SGF manipule alors des blocs numérotés de 0 à N-1

  (N = taille du disque/taille d'un bloc).
- Si l'on prend des blocs de 1 Ko sur un disque dont les secteurs font 512 octets, le système de fichiers lit et écrit deux secteurs consécutifs en les considérant comme un ensemble unique et indivisible, appelé unité d'allocation (*cluster)*.

Le système de fichiers

Organisation logicielle de l'espace du disque
Stratégie de stockage des fichiers

- Il existe deux stratégies pour stocker un fichier de n blocs :

– **Allocation contiguë** : on alloue n blocs consécutifs sur le disque

– **Allocation non-contiguë** : on divise le fichier en plusieurs blocs (pas nécessairement contigus).

Le système de fichiers

Allocation contiguë

Un fichier est une suite de blocs contigus sur le disque

Localisation d'un fichier = adresse du premier bloc et nombre de blocs

Avantages

Lecture très performante, il suffit de se placer au premier bloc. Pas besoin de chercher les blocs

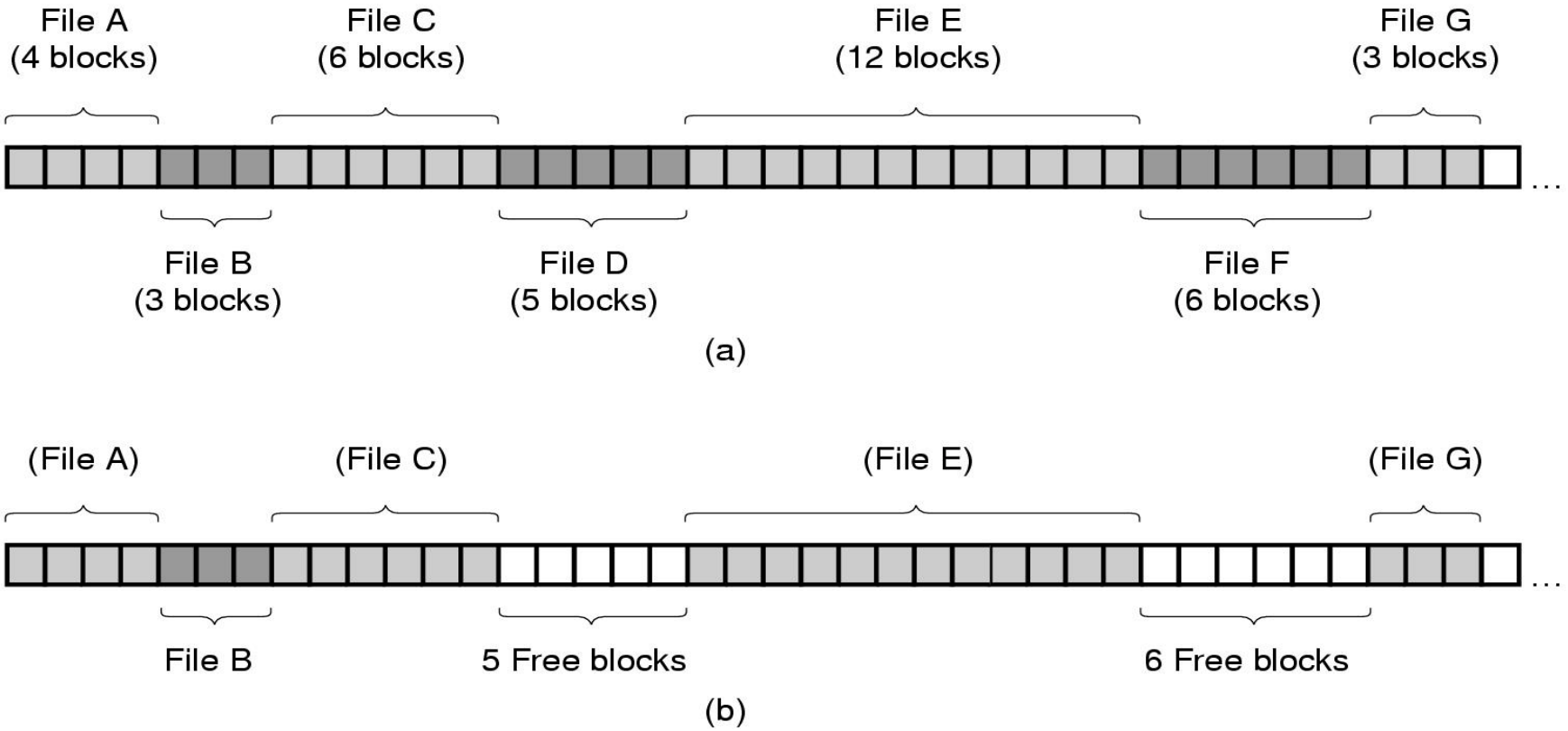
Problèmes

- Au cours du temps, des trous apparaissent (fichiers supprimés) : fragmentation
- On peut défragmenter le disque mais très coûteux

Utilisé dans les CDRoms

Le système de fichiers

Allocation contiguë- Exemple



(a) Allocation d'espace disque pour 7 fichiers

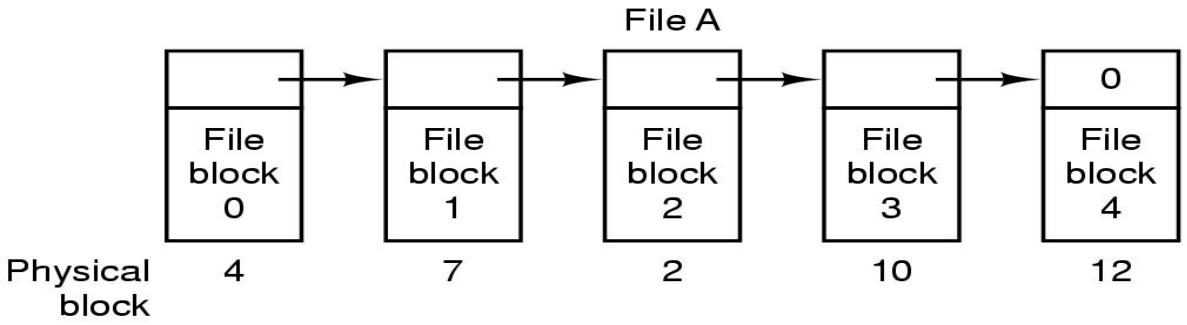
(b) Suppression des fichiers *D et F*

=> Fragmentation externe

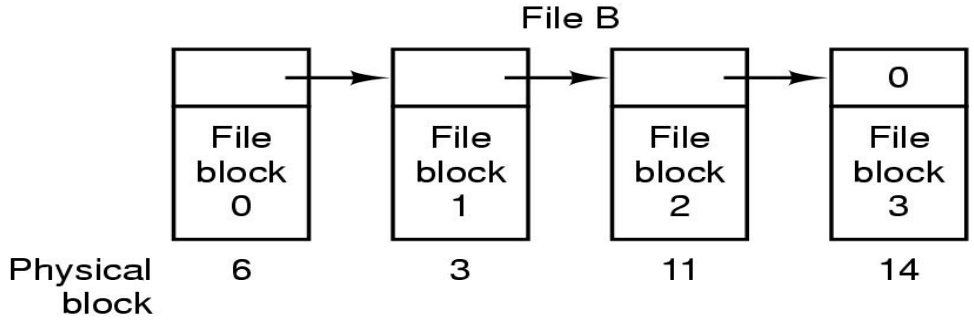
Le système de fichiers

Allocation non contiguë - Allocation par liste chaînée

- Un fichier est une série de blocs maintenus dans une liste chaînée
- Le début d'un bloc est utilisé comme pointeur vers le suivant
- Pas de fragmentation externe
- Fragmentation interne sur le dernier bloc
- Un fichier est trouvé par son adresse de premier bloc



- Pas pratique pour un accès direct à un bloc



Le système de fichiers

Allocation non contiguë - Table d'allocation de fichiers (FAT)

La FAT permet de localiser les blocs de chaque fichier du système de fichiers.

Une partition FAT dispose d'une table **FAT et cette dernière possède autant d'entrées qu'il y a de blocs sur** la partition.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	x	EOF	13	2	9	8	L	4	12	3	E	EOF	EOF	L	...

« XX » la taille du disque, « L » un bloc libre et « E » un bloc endommagé.

➢SGF conserve le premier bloc de chacun des fichiers dans son répertoire.

Nom du fichier	Extension	Attributs	Réservé	Heure	Date	Nº du 1er bloc	Taille

➢Le parcours de la FAT est nettement plus rapide que la chaîne des blocs.

➢Cependant si elle n'est pas constamment, tout entière en mémoire, elle ne permet pas d'éviter les entrées-sorties du disque.

Le système de fichiers

Allocation non contiguë - Table d'allocation de fichiers (FAT)

En FAT12 (FAT16, FAT32) les numéros de blocs sont écrits sur 12 (16, 32) bits.

Le FAT16 est utilisé par MS-DOS. En FAT16, les numéros de blocs sont écrits sur 16 bits. Si on suppose que la taille d'un bloc (cluster) est 32Ko, la taille maximale adressables est alors 2Go :

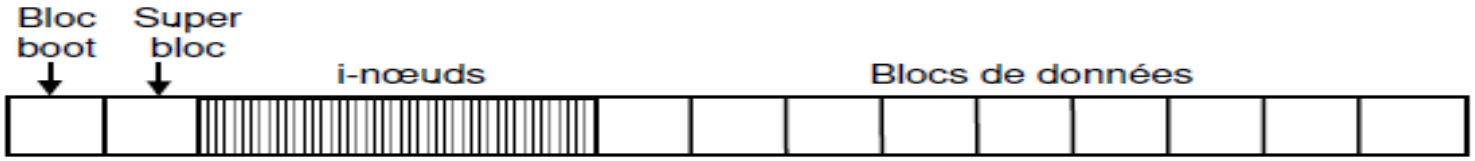
$$2^{16} \times 32 \text{ Ko} = 2097152 \text{ Ko} = 2\text{Go}$$

Sous **FAT32** la taille maximale adressable théoriquement est de 8 To (2^{28} x 32 Ko = 8 To). Toutefois, Microsoft la limite volontairement à 32 Go sur les systèmes Windows 9x afin de favoriser NTFS.

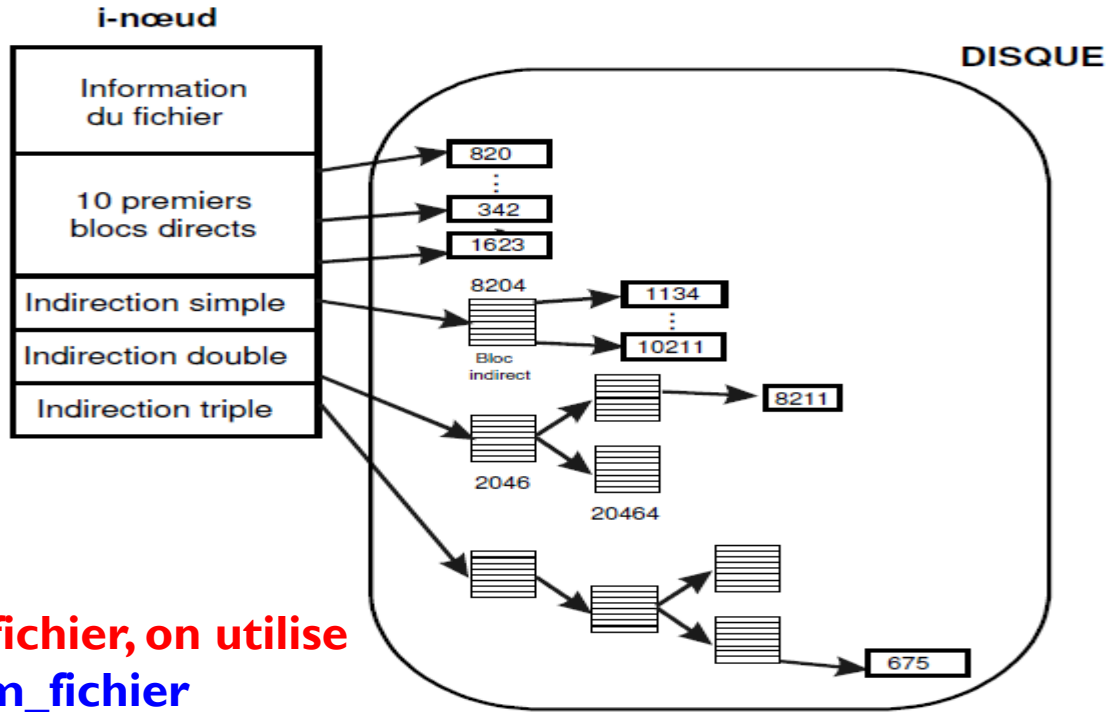
Le système de fichiers

Allocation non contiguë - Tables d'index des i-noeuds

- On associe à chaque fichier un Index-Node (i-node)
- Il contient les attributs et les adresses des blocs sur le disque



- La taille (32 bits)
- L'identité du propriétaire
- Les droits d'accès
- ….



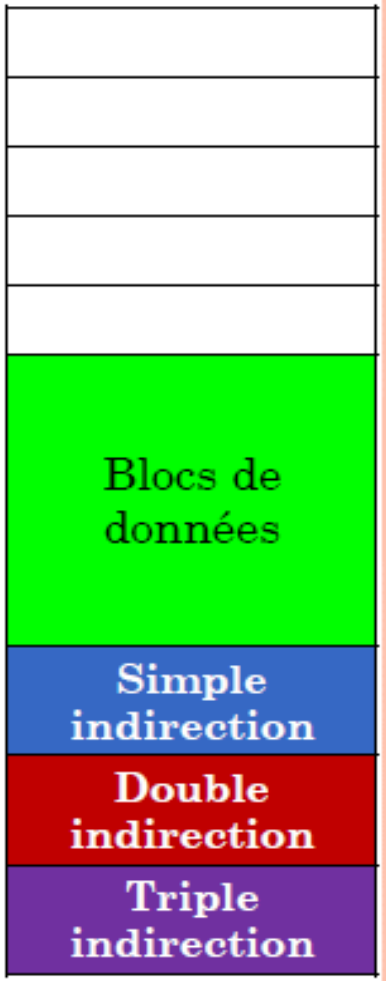
ext3fs d'Unix ou GNU/Linux (ext3fs pour *third extented file system)*

Pour voir le inode d'un fichier, on utilise la commande : ls -i nom_fichier

Le système de fichiers

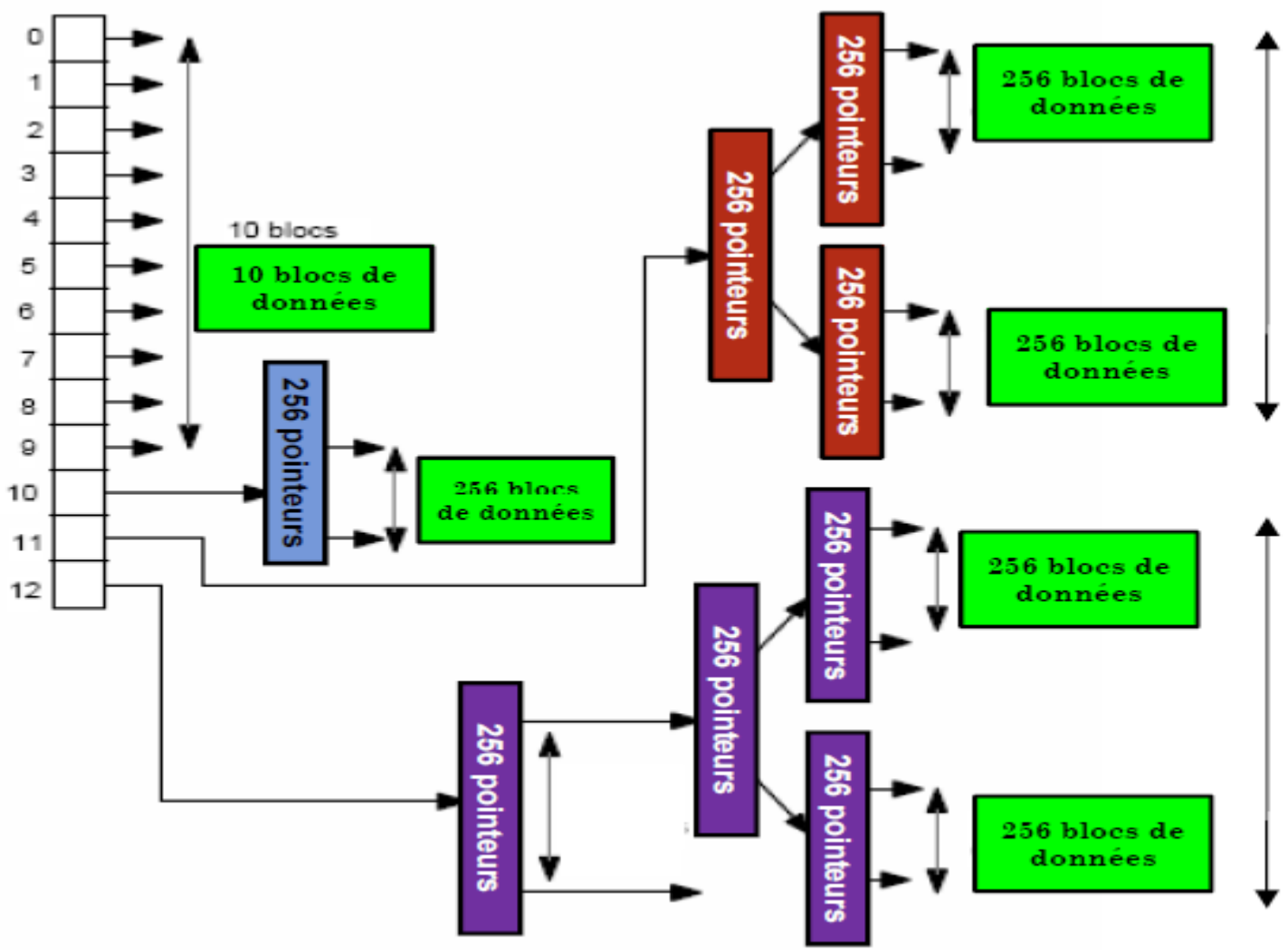
Allocation non contiguë - Tables d'index des i-noeuds

- Chaque fichier se voit affecter une table à 13 entrées :
  - les entrées 0 à 9 pointent sur un bloc de données
  - l'entrée 10 pointe sur un bloc d'index qui contient 256 pointeurs sur des blocs de données :
    - **simple indirection**
  - l'entrée 11 pointe sur un bloc d'index qui contient 256 pointeurs sur des blocs d'index dont chacun contient 256 pointeurs sur des blocs de données:
    - **double indirection**
  - l'entrée 12 pointe sur un bloc d'index qui contient 256 pointeurs sur des blocs d'index dont chacun contient 256 pointeurs sur des blocs d'index dont chacun contient 256 pointeurs sur des blocs de données:
    - **triple indirection**



Le système de fichiers

Allocation non contiguë - Tables d'index des i-noeuds



Le système de fichiers

Allocation non contiguë - Tables d'index des i-noeuds

Un bloc de 1024 octets pourra désigner jusqu'à 256 blocs de données, sachant que chacun d'eux est numéroté sur 4 octets

La taille maximale théorique d'un fichier décrit par un i-noeud est de :
$(10 \times 1k) + (256 \times 1k) + (256^2 \times 1k) + (256^3 \times 1k) > 16$ Go

Mais comme le champ taille du fichier dans un i-noeud est codé sur 32 bits, la taille maximale effective pour un fichier est de 4 Go (2^{32}).

Le système de fichiers

Allocation non contiguë - Tables d'index des i-noeuds

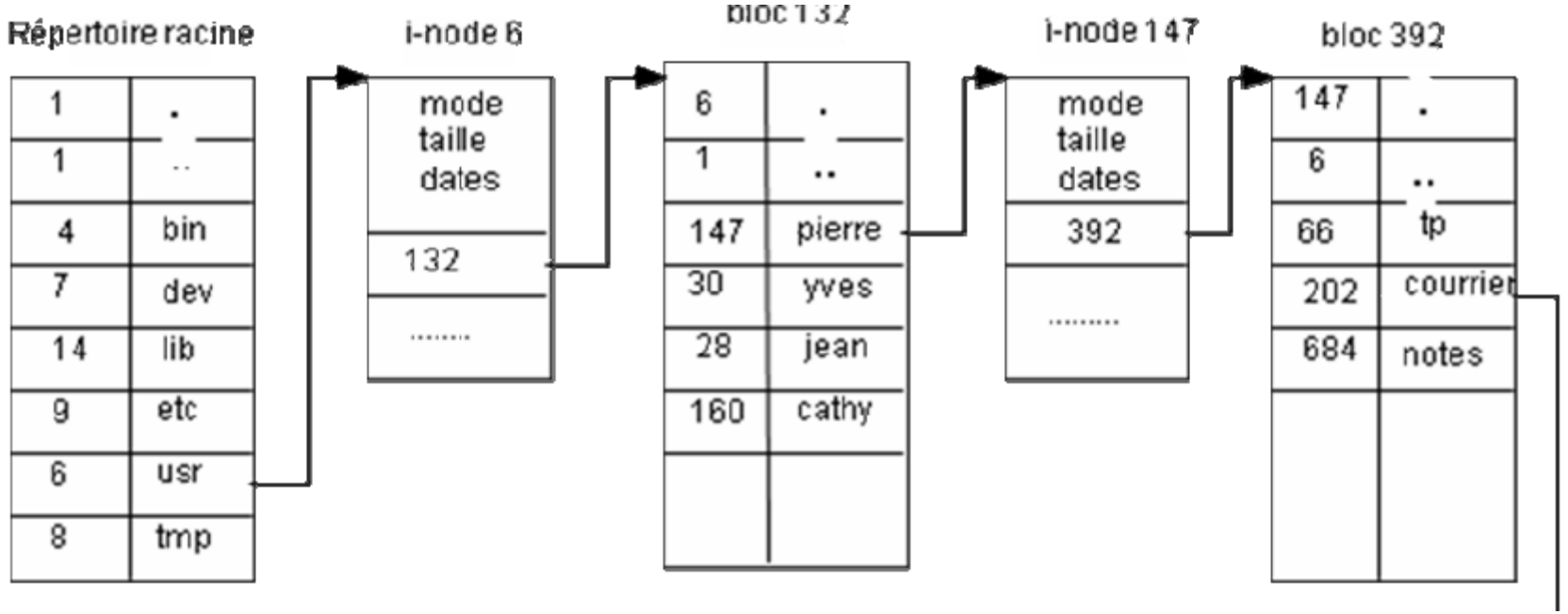
- L'avantage par rapport à une FAT est que seul l'inode d'un fichier ouvert a besoin d'être en mémoire
- Une FAT augmente linéairement avec la taille du disque
- La mémoire occupée par les inodes augmente avec le nombre de fichiers ouverts

Problème

- Si un inode a une taille max, il ne peut contenir qu'un nombre limité d'adresses
- Un fichier aura donc une taille maximale
- Utilisations de plusieurs indirections (UNIX)

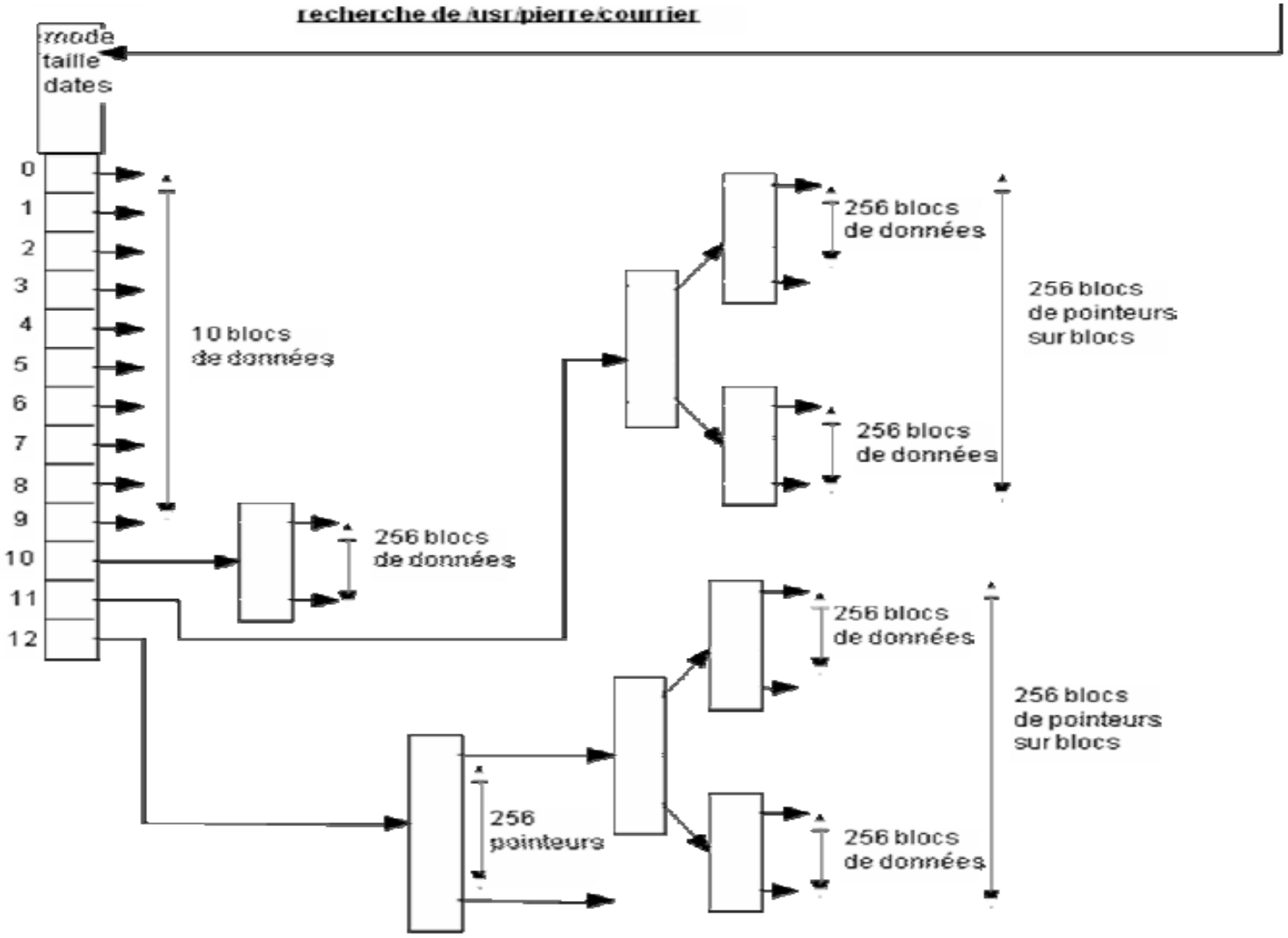
Le système de fichiers

Allocation non contiguë - Tables d'index des i-nœuds Exemple



Le système de fichiers

Allocation non contiguë - Tables d'index des i-nœuds Exemple



Le système de fichiers

Organisation de l'espace du disque
Repérage des blocs libres

- Des qu'on a choisi la taille des blocs, on doit mémoriser les blocs Libres
- Les deux méthodes les plus répandues sont

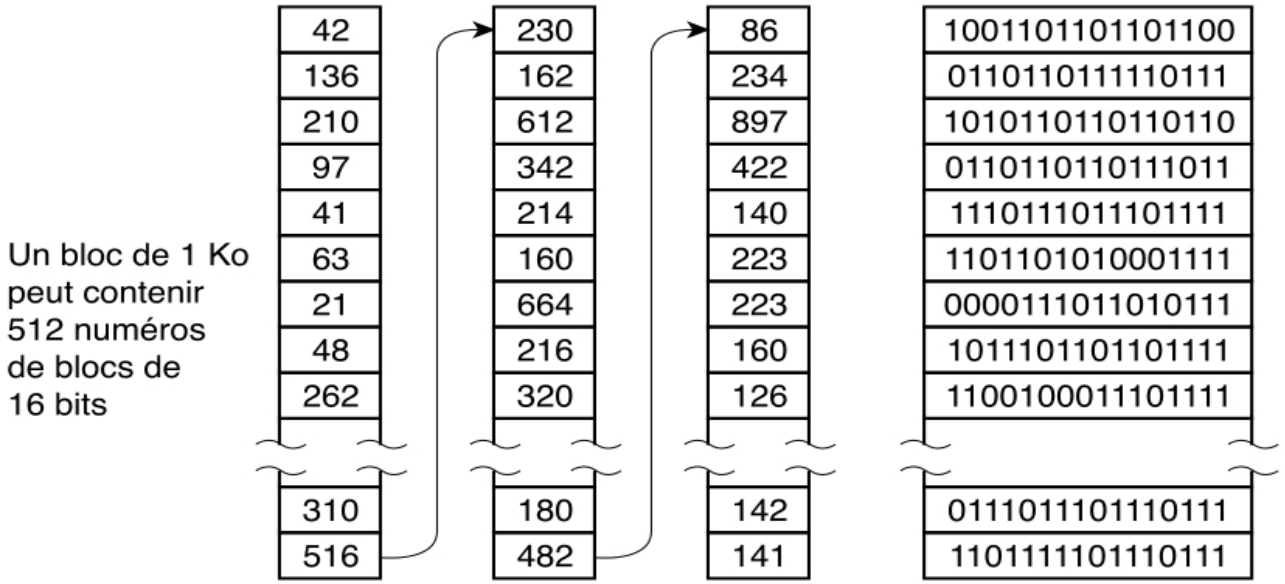
– Liste chainée
– Table de bits

- La première méthode (Liste chainée) consiste a utiliser une liste chainée des blocs du disque, chaque bloc contenant des numéros de blocs libres.
- La deuxième technique (Table de bits) de gestion des espaces libres a recours a une table de bits, chaque bit représentant un bloc et valant 1 si le bloc est occupe (ou libre suivant le système d'exploitation).

– Un disque de n blocs requiert une table de n bits.

Le système de fichiers

Organisation de l'espace du disque
Repérage des blocs libres



(a) Les blocs libres mémorisés dans une liste chaînée

(b) Une table de bits

Le système de fichiers

Organisation de l'espace du disque
Repérage des blocs libres - liste chaînée (Exemple)

- Considérons un disque dure de 500GB. Une taille d'un bloc de 1KB.

  → Le disque contient 524 millions de bloc (500*1024*1024= 524 288 000)

- Le numéro de bloc codé sur 32 bits. Taille d'un élément de la liste chainée = un bloc contiendra 256 numéros de bloc libres

  – Nombre de numéro de bloc libre que peut contenir un bloc de 1Ko = Taille d'un bloc en bit / taille du numero du bloc (1024 octet*8 bit /32 bits)

- Pour adresser tous les blocs du disque dur (524 millions de numero de blocs) on a besoin de (524 288 000 / 256 = 2 048 000 blocs)

→ 2 millions de blocs !!!!

Le système de fichiers

Organisation de l'espace du disque
Repérage des blocs libres -table de bit (Exemple)

- Considérons un disque dure de 500GB. Une taille d'un bloc de 1KB.

→ Le disque contient 524 millions de bloc (500*1024*1024)

- 1 bloc est représente par un bit → taille max de la table de bit = 524 millions de bits = (524288000/8) / 1024 = 64000 Ko = 64000 blocs
- Pour adresser tous les blocs du disque dur

→ 64000 bloc

Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master GSI
Semestre 1

Systèmes d'exploitation 2

Gestion de la mémoire

Contenu du cours

1. **Architecture des Systèmes d'Exploitation**
   a. **Présentation générale** (Rôle du système d'exploitation, Structure générale du système Linux, Organisation du noyau)
   b. **Processus** (Notions de base, Ordonnancement, Clonage)
   c. **Le système de fichiers** (Organisation des fichiers, I-noeuds, Verrouillage de fichiers, Système virtuel de fichiers)
   d. **Gestion de la mémoire** (Allocation mémoire, espace d'adressage des processus, la swap)
   e. **Signaux** (Emission d'un signal, détournement des signaux, implémentation des signaux :envoie / réception / blocage / détournement)
   f. **Gestion de périphériques** (Principes, mode bloc / mode caractère)
   g. **Modules chargeables** (Principes, Exemples de modules chargeables)
2. **Linux Temps Réel**
   a. **Unix et le temps réel**
   b. **RTLinux**

Gestion de la mémoire

1-Introduction

La mémoire est une ressource importante qui doit être gérée avec attention.

La mémoire physique est un ensemble d'emplacement. Pour qu'un programme puisse s'exécuter il doit être chargé dans la mémoire principale (MP).

Une MP est un ensemble de mots ou d'octets chacun possède sa propre adresse.

La gestion de la mémoire a deux objectifs principaux :

1) De partager le mémoire physique entre les programmes et les données des processus prêts (connaître l'état de la mémoire : libre ou occupée)
2) La mise en place des paramètres de calcul d'adresses, permettant de transformer une adresse virtuelle en adresse physique.

Gestion de la mémoire

2- Représentation des adresses d'un objet

Adresse symbolique: représente les noms des objets (données et instructions) contenus dans le code source. Exemple : int compteur ;

Adresse logique : correpond à la traduction des adresses symboliques lors de la phase de compilation ;

Adresse physique : représente l'emplacement physique des adresses logiques d'un programme lors de son exécution.

La CPU manipule des **adresses logiques (emplacement relatif)**

L'unité mémoire manipule des **adresses physiques (emplacement mé**moire).

MMU (*Memory Management Unit*): *Dispositif matériel qui fait la conversion des* adresses virtuelles à physiques

3- Allocation de la mémoire - Système Monoprogrammation

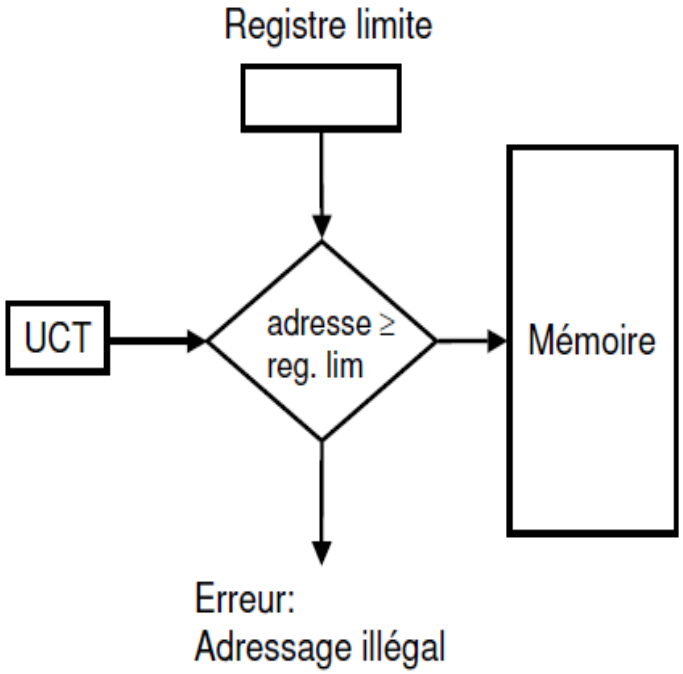
3-1 Partition unique :

La manière la plus simple de gérer la mémoire est d'**exécuter un seul programme à la fois.** Un seul processus peut se retrouver à l'instant **t** dans la mémoire principale.

La mémoire est habituellement subdivisée en deux partitions une pour le SE et l'autre pour les processus utilisateurs (mémoire haute).

Utilisé par certains (petits) MS-DOS..

Pour protéger le code du système d'exploitation, il y a un **registre limite** qui contient l'adresse à partir de laquelle commencent les instructions et données des programmes des utilisateurs.



Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

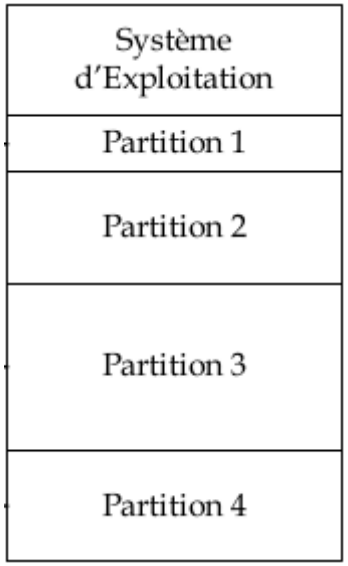
3 - 2 Partitions multiples: L'espace utilisateur est divisé en plusieurs partitions.

La plupart des SE modernes autorisent l'exécution de processus multiples en même temps: Lorsqu'un processus est bloqué en attente d'une E/S, un autre peut utiliser la CPU.

3 – 2- 1 Partitions multiples statiques

La manière la plus simple de faire de la multiprogrammation consiste à subdiviser la mémoire en **n** partitions de taille fixe.

Chaque partition peut contenir exactement 1 processus.



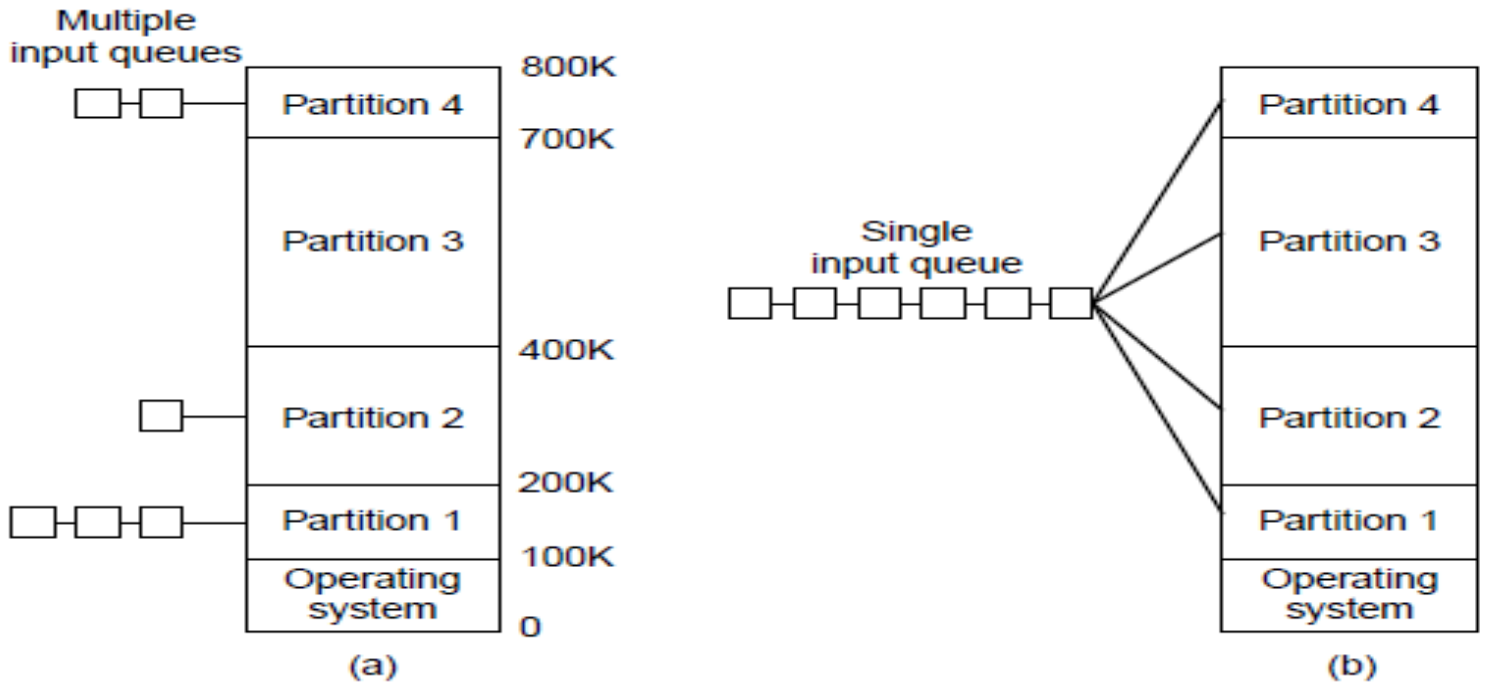
Degré de multi programmation = nombre de partitions.

SE maintient une table indiquant les parties de la mémoire disponibles et celles qui sont occupées.

Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 1 Partitions multiples statiques



(a) Partitions **statiques** avec des files d'attente différentes
(b) Partitions **statiques** avec une seule file d'attente différentes

Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 1 Partitions multiples statiques

(a) Partition statiques avec des files d'attente différentes

Problème : la file pour une grande partition est vide tandis que celle pour une petite partition est pleine.

(b) Partition statiques avec une seule file d'attente différentes

A chaque libération, choisir le 1er programme à exécuter.

Problème : il restera une partie libre non utilisée (**fragmentation interne**)

Parcourir la file d'attente pour chercher le plus gros travail qui peut être placé dans la partition.

Problème : Cet algorithme pénalise les petits travaux.

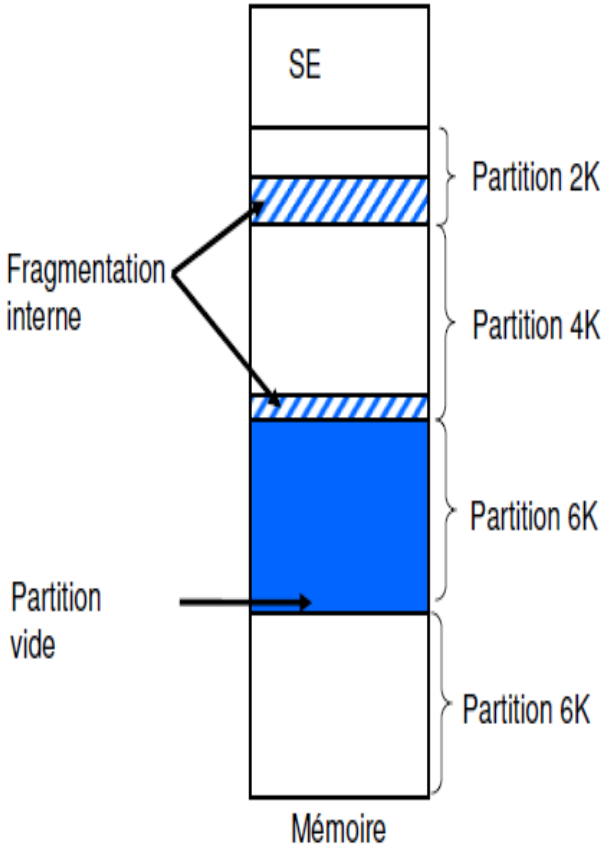
Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 1 Partitions multiples statiques

Fragmentation interne: la mémoire allouée peut être plus grande que le mémoire requise. Cette différence est interne à une partition mais n'est pas utilisée.

Fragmentation externe: lorsque aucune partition n'est suffisante pour accueillir un Pg, alors que la somme des partitions libers permet le chargement du Pg.



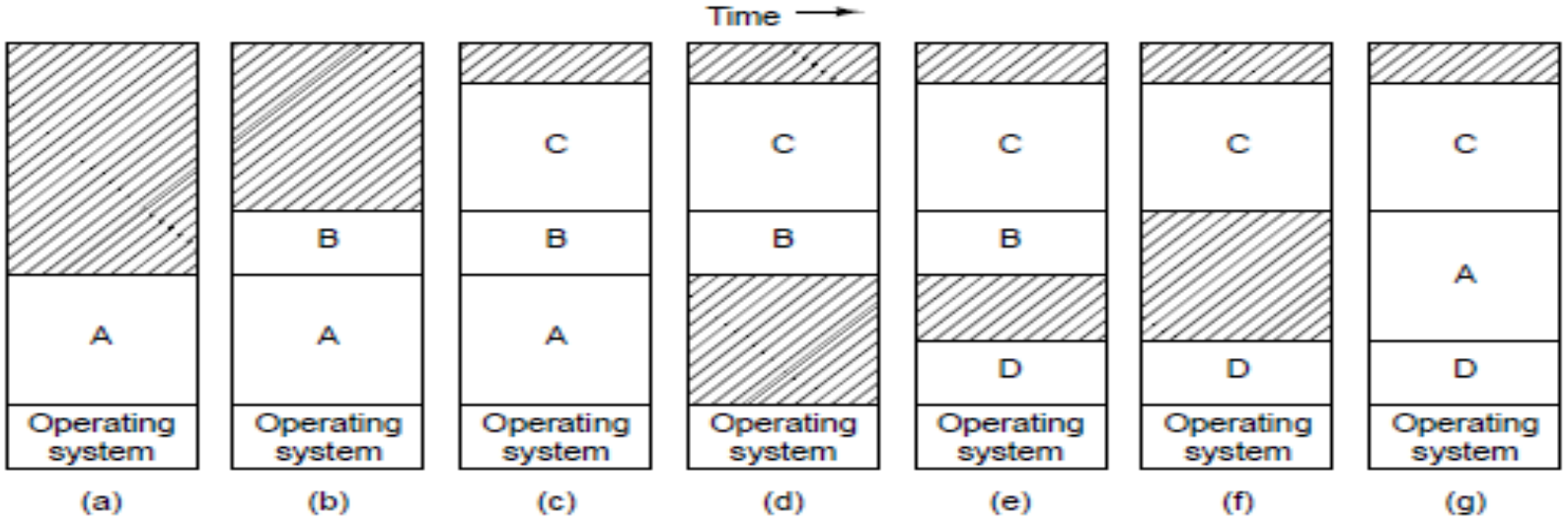
Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

Swapping : Conserver les processus supplémentaires sur un disque.

Transfert temporaire d'un processus de la mémoire principale à une mémoire auxiliaire - **il sera ensuite ramené en mémoire pour continuer son exécution.**

Contraintes: Processus à transférer doit être inactif.



3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

On partitionne la mémoire principale dynamiquement selon la demande. À chaque processus on alloue une partition exactement égale à sa taille.

Stratégies de placement

Dans un partitionnement multiple dynamique, le placement des programmes dans les partitions se fait selon un certain nombre de stratégies possibles.

On distingue les stratégies suivantes :

First-Fit : premier trou suffisant**. Rapide,** provoque la fragmentation de la mémoire
Next-Fit : recherche à partir de l'emplacement précédent. **Un peu moins bon.**
Best-Fit : trou le plus petit possible. **Moins performant.**
Worst-Fit : trou le plus grand**. Bof.**

Remarques

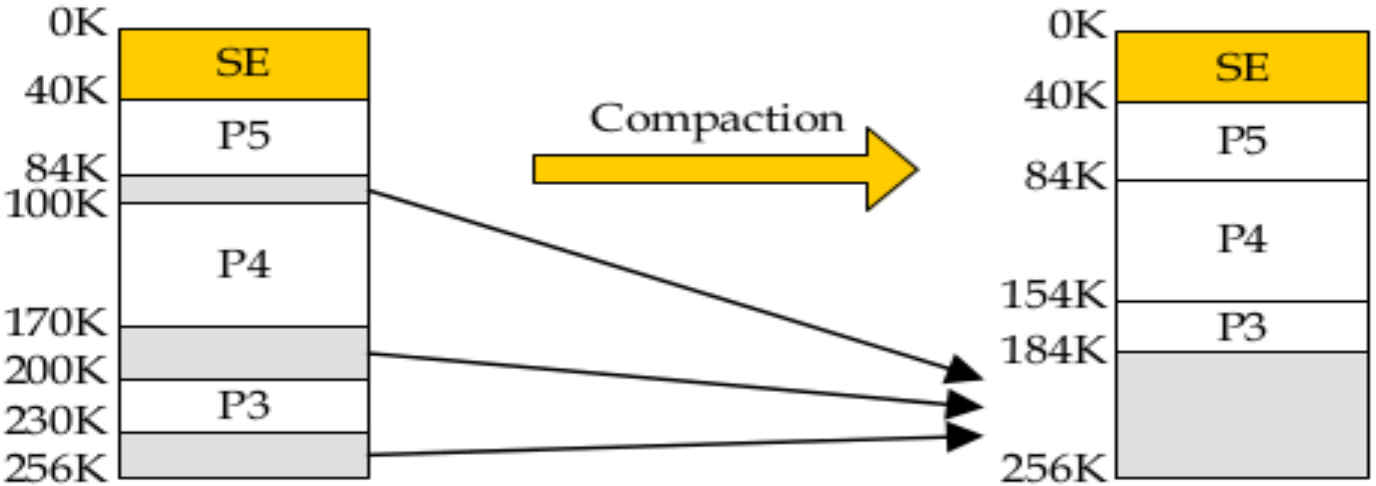
Les deux autres stratégies sont lourdes à mettre en oeuvre ; Un problème de fragmentation externe se pose et la solution c'est le compactage.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Compactage

Le compactage est une solution pour la fragmentation externe qui permet de regrouper les espaces inutilisés. L'opération de compactage est effectuée quand un programme qui demande d'être exécuté ne trouve pas une partition assez grande, mais sa taille est plus petite que la fragmentation externe existante



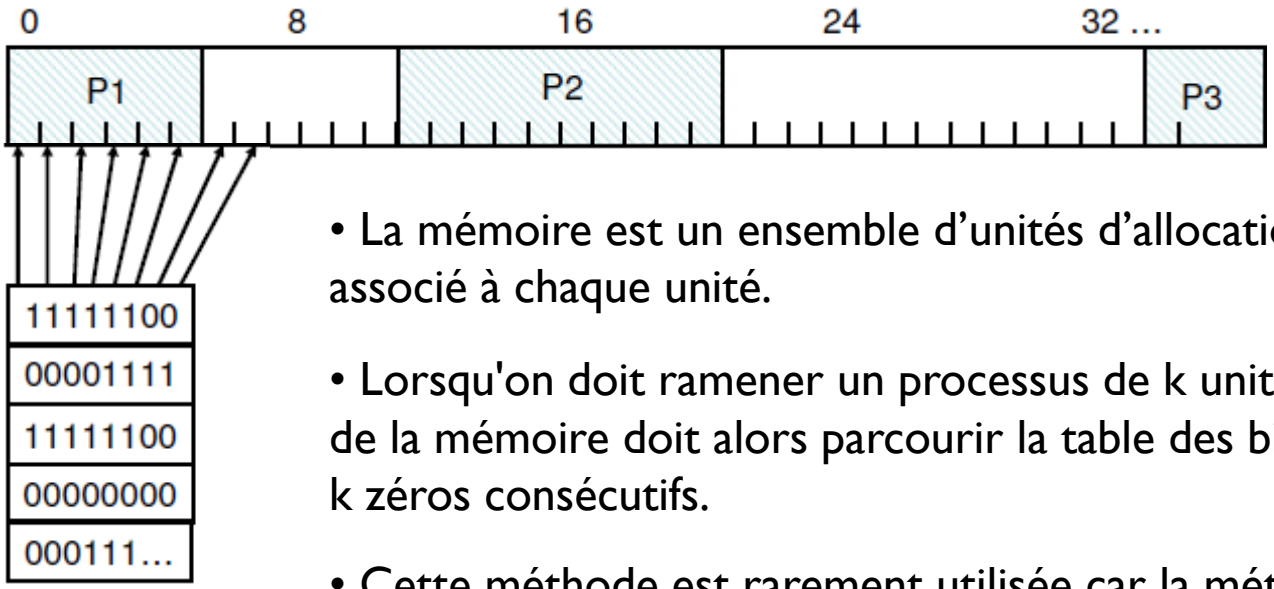
3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pour gérer l'allocation et la libération de l'espace mémoire, le gestionnaire doit connaître l'état de la mémoire :

- Tables de bits,
- Listes chaînées.

Tables de bits



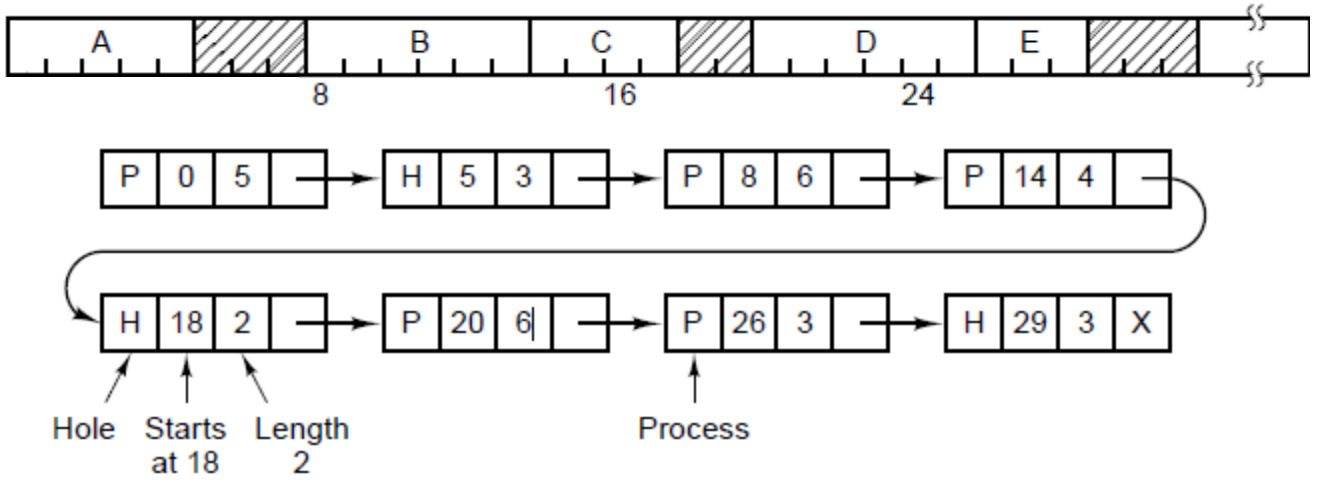
- La mémoire est un ensemble d'unités d'allocation. Un bit est associé à chaque unité.
- Lorsqu'on doit ramener un processus de k unités, le gestionnaire de la mémoire doit alors parcourir la table des bits à la recherche de k zéros consécutifs.
- Cette méthode est rarement utilisée car la méthode de recherche est lente (k zéros consécutifs).

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Listes chaînées.

- Maintenir une liste chainée des partitions mémoire allouées et libres
- Dans cette liste chainée, un élément est soit un processus, soit un espace libre entre deux processus



3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination

C'est une autre solution au problème de la fragmentation externe. Elle permet aussi l'allocation d'espace non nécessairement contigu pour l'exécution d'un processus. Dans ce cas, l'espace d'adressage d'un processus est divisé en partitions égales appelées pages.

Cadres de page : mémoire physique découpée en zones de taille fixe ***(frames)***

Taille: puissance de 2 ⇒ entre 512 octets et 8192 octets.

Table de cadre (*Frame table) indique les cadres libres et occupés*

Adresse logique : numéro de page + déplacement dans la page

Table de pages : liaison entre numéro de page et cadre de page (une table par processus) : **Taille pages = Taille cadres**

Taille des pages : puissance de 2

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination - Exemple

Adresses virtuelles: Un ordinateur peut produire des adresses sur 16 bits avec des valeurs entre 0 et 64 K octets.

L'ordinateur a seulement 32 K octet de mémoire physique.

La taille de page et de cadre est de 4K octet .

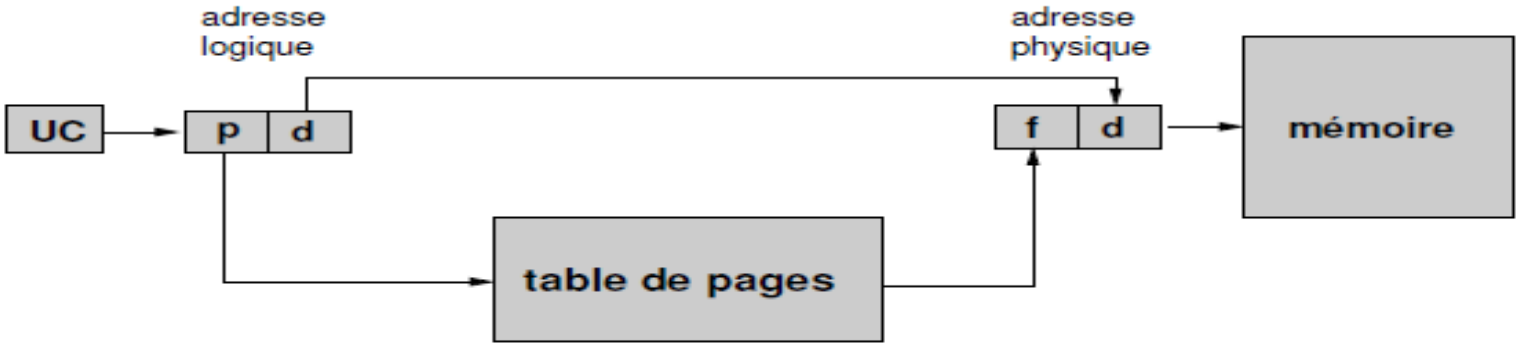
Nous avons donc …. pages virtuelles et …..cadres de pages.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination

Chaque adresse générée par le processeur central est divisée en deux parties : un numéro de page p et un déplacement page d. La table des pages contient l'adresse de base de chaque page dans la mémoire physique.



Si un processus nécessite n pages et 1 octet, on doit lui allouer (n + 1) pages.

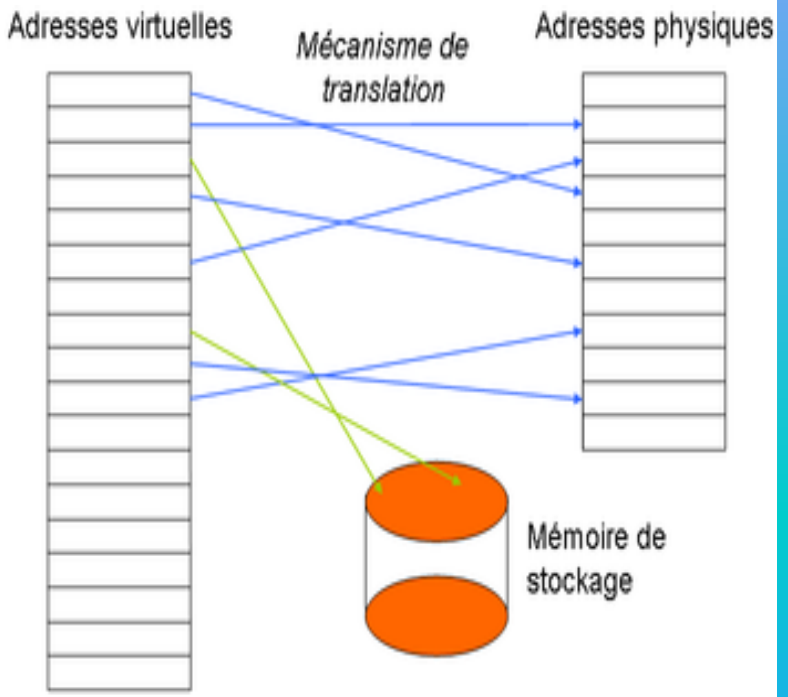
Dans la plupart des OS modernes, la mémoire physique est divisée en pages de quelques kilo-octets (1, 2 ou 4 ko généralement). Dans la plupart des ordinateurs, la dimension de la page est fixée à 4 k.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination - Défauts de page et swap file

Si le processus illustré plus haut tente d'accéder à une page, qui n'a pas de correspondance dans la mémoire physique. Il se produit alors un *défaut de page* : le système d'exploitation doit sélectionner un cadre de page (physique), le libérer en le copiant sur le disque (mémoire secondaire: swap file) et le remplacer par la mémoire désirée. On met la table des pages à jour ensuite.



3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination - Remplacement de pages

Quand une page n'est pas en mémoire...

1. Trouver l'emplacement de la page désirée sur disque.
2. Trouver un **cadre de page libre.**
   - (a) S'il existe un cadre libre, l'utiliser.
   - (b) Sinon, utiliser un **algorithme de remplacement** de pages pour sélectionner un cadre (victime).
   - (c) Enregistrer la page victime dans le disque, modifier les tables de pages et de cadres.
3. Ecrire la page désirée dans le cadre libéré, modifier les tables de pages et de cadres.
4. Redémarrer le processus utilisateur.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination -Algorithmes de remplacement des pages

Critère: choisir l'algorithme avec le taux de défaut de page le plus bas.

1- Algorithme optimal : Remplacer la page qui mettra le plus de temps à être de nouveau utilisée. **Impossible à implanter:** il requiert une connaissance future de la chaîne de références. Utilisé pour effectuer des études comparatives.

2- Algorithme FIFO : Associe à chaque page le moment où l'on a ramené en mémoire. Si il faut remplacer une page, c'est la plus ancienne que l'on sélectionne. Sa performance n'est pas toujours bonne. Anomalie de Belady (1969) : augmenter le nombre de cadres de page n'améliore pas forcément les performances du système

3- Algorithme LRU (*least-recently used)* Remplacer la page que n'a pas été utilisée pendant la plus longue période de temps. Très utilisé.

Implantation par pile: Chaque fois que l'on référence une page, on la supprime et on la met au somment de la pile (sommet ⇒ la page récemment utilisée; base ⇒ page LRU)

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination -Allocation de cadres de pages

Nombre de cadres de pages (*frames): défini par l'architecture.*

1- Allocation équitable

m cadres, n processus m/n cadres pour chaque processus

2- Allocation proportionnelle : Allouer la mémoire disponible à chaque processus pi selon sa taille.

si : taille de la mémoire logique pour pi. **S = ∑ si.**

m : nombre total de cadres.

ai : cadre alloués à pi. **ai =(si/S)∗ m.**

Arrondir chaque ai pour que ce soit un entier supérieur au nombre minimal de cadres
La somme des ai ne doit pas dépasser m.

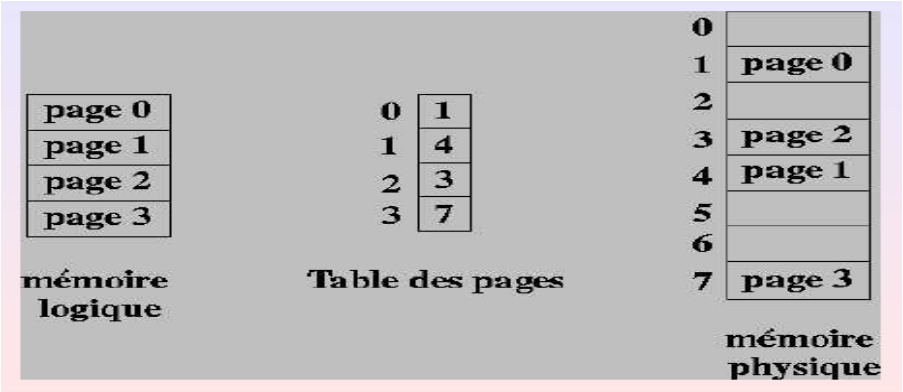
3- Allocation prioritaire Utiliser un schéma d'allocation proportionnelle où la quantité de cadres dépend non de la taille des processus mais de leurs priorités.

Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination- Exemple



L'adresse virtuelle de 16 bits est divisée en deux parties: un numéro de page sur 4 bits (donc nous pouvons avoir 16 pages) un décalage (*offset) sur 12 bits (*La taille de page et de cadre est de 4K octet)

Problème :

1- Le temps d'accès en mémoire est doublé.

2-Dans les archi à 32 bits, par exemple, l'adresse logique $\in \{0; \ldots; 2^{32}\}$ la taille d'une page = 4K = 2^{12} ; alors taille de la table de pages = 2^{20}

Solutions :

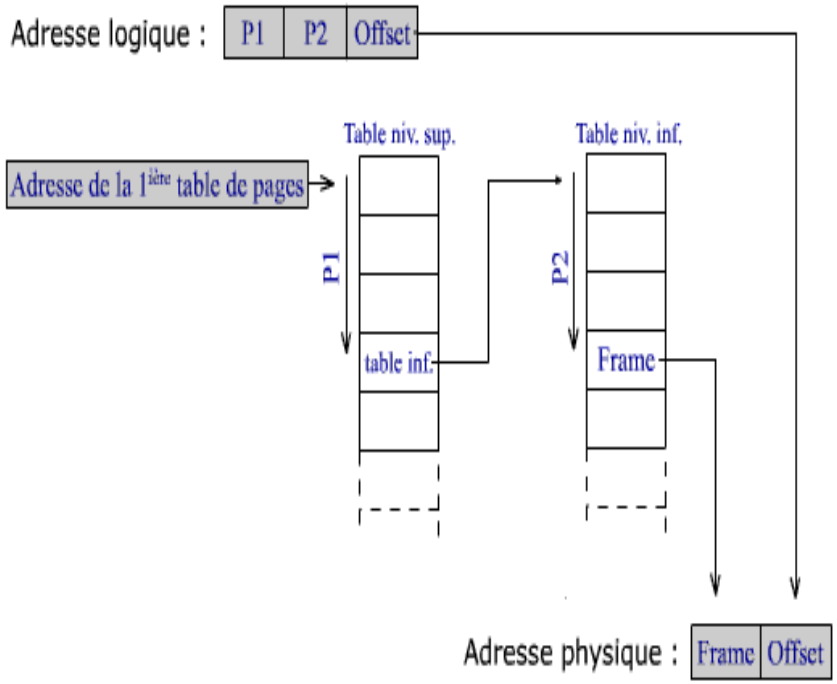
plusieurs niveaux d'indirection, tables de pages inversées.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Pagination multiniveaux

Une table de page de niveau supérieur dont chaque élément pointe vers une table de niveau inférieur. L'adresse logique contient dès lors deux nombres pour aboutir au numéro de page. Le premier sert d'index dans la table de niveau supérieur, le second sert d'index dans la table du niveau suivant.



3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation

Chaque processus est constitué d'un ensemble de segments. Chaque segment est un espace linéaire (continu) . Les segments sont des espaces d'adressages indépendants de différentes longueurs et qui peuvent même varier en cours d'utilisation. Ils correspondent à des subdivisions logiques déterminées par le programmeur ou par le compilateur.

Organisation de la mémoire en unités logiques :

code (TEXT), données statiques initialisées (DATA), données statiques non initialisées (BSS), données dynamiques (TAS), pile d'exécution.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation

Dans une mémoire segmentée, chaque unité logique d'un programme est stockée dans un bloc mémoire, appelé **segment** à l'intérieur duquel les adresses sont relatives au début du segment. Ces segments sont de tailles différentes. Un programme sera donc constitué d'un ensemble de segments de code et de données, pouvant être dispersés en mémoire principale.

La famille x86 qui possède 4 segments : le code (CS Code Segment), la pile (SS Stack Segment), les données (DS Data Segment) et un supplémentaire, à la discrétion du programmeur (ES Extra Segment)

La segmentation facilite l'édition de liens, ainsi que le partage entre processus de segments de données ou de codes.

3 - Allocation de la mémoire - Système Multiprogrammation

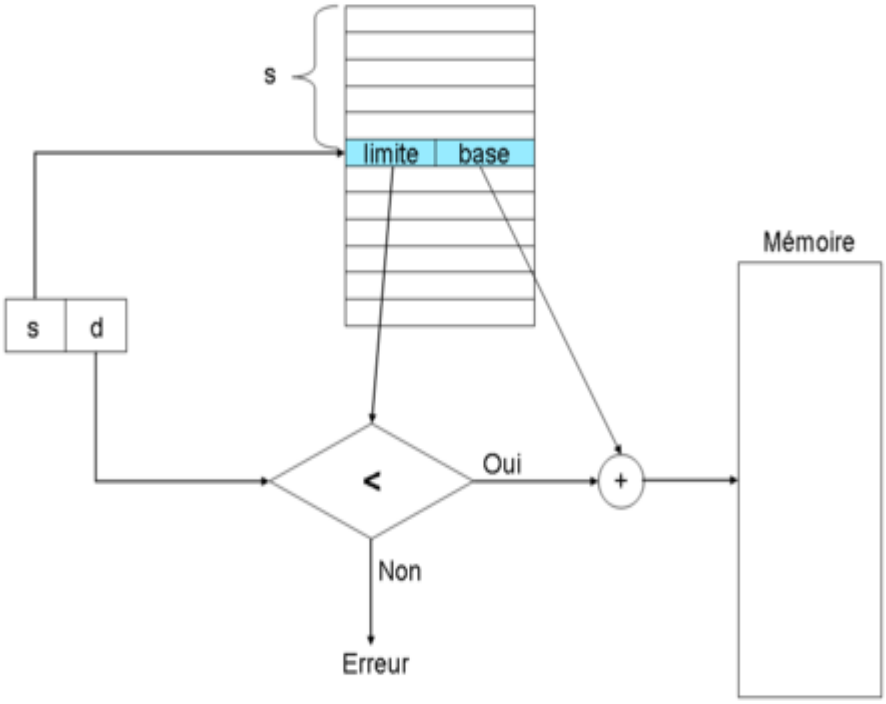
3 – 2- 2 Partitions multiples dynamiques

Segmentation - Schéma de translation d'adresses

Chaque segment est repéré par son numéro S et sa longueur variable L.

Une adresse logique est donnée par un couple (S; d), où S est le numéro du segment et d le déplacement dans le segment.

L'association d'une adresse logique à une adresse physique est décrite dans une table appelée table de segments.



La **base** est l'adresse de début du segment, et **limite** la dernière adresse du même segment.

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation

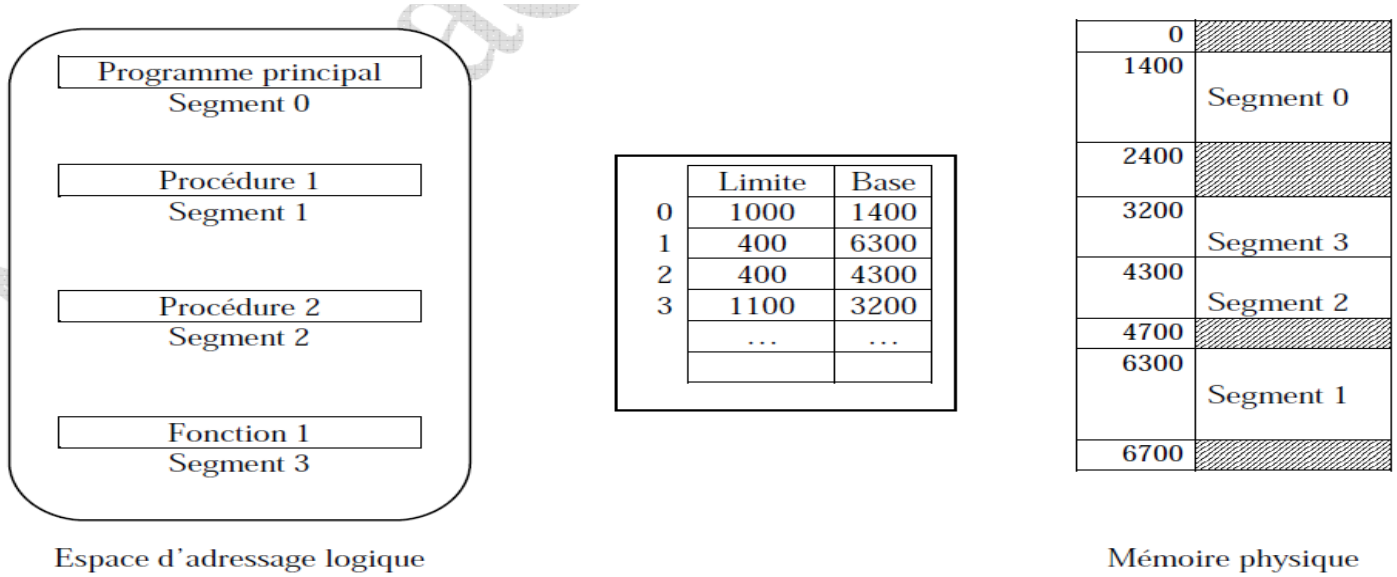
Contrairement à la pagination, la segmentation permet d'éliminer la fragmentation interne car l'espace mémoire alloué à un segment est de taille égale exactement à la taille du segment. Cependant, une fragmentation externe peut se produire due au fait qu'on fait une allocation dynamique de l'espace mémoire.

Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation- Exemple



Adresse logique		Adresse physique
N° segment	Déplacement	
2	53	
3	852	
0	1222	

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation - pagination

Microprocesseurs: 1. Ligne Motorola: 68000 2. Intel 80X86
Modèle de mémoire fusionné (mélange de pagination et segmentation)

L'adressage est composé de deux niveaux :

- Le niveau segment et
- Le niveau page

Chaque segment est un espace linéaire d'adressage logiques. Il est découpé en pages. Une adresse est formée de trois parties : (s, p, d')
s : numéro du segment p : numéro de page d' : déplacement dans la page

Méthode : Table de page

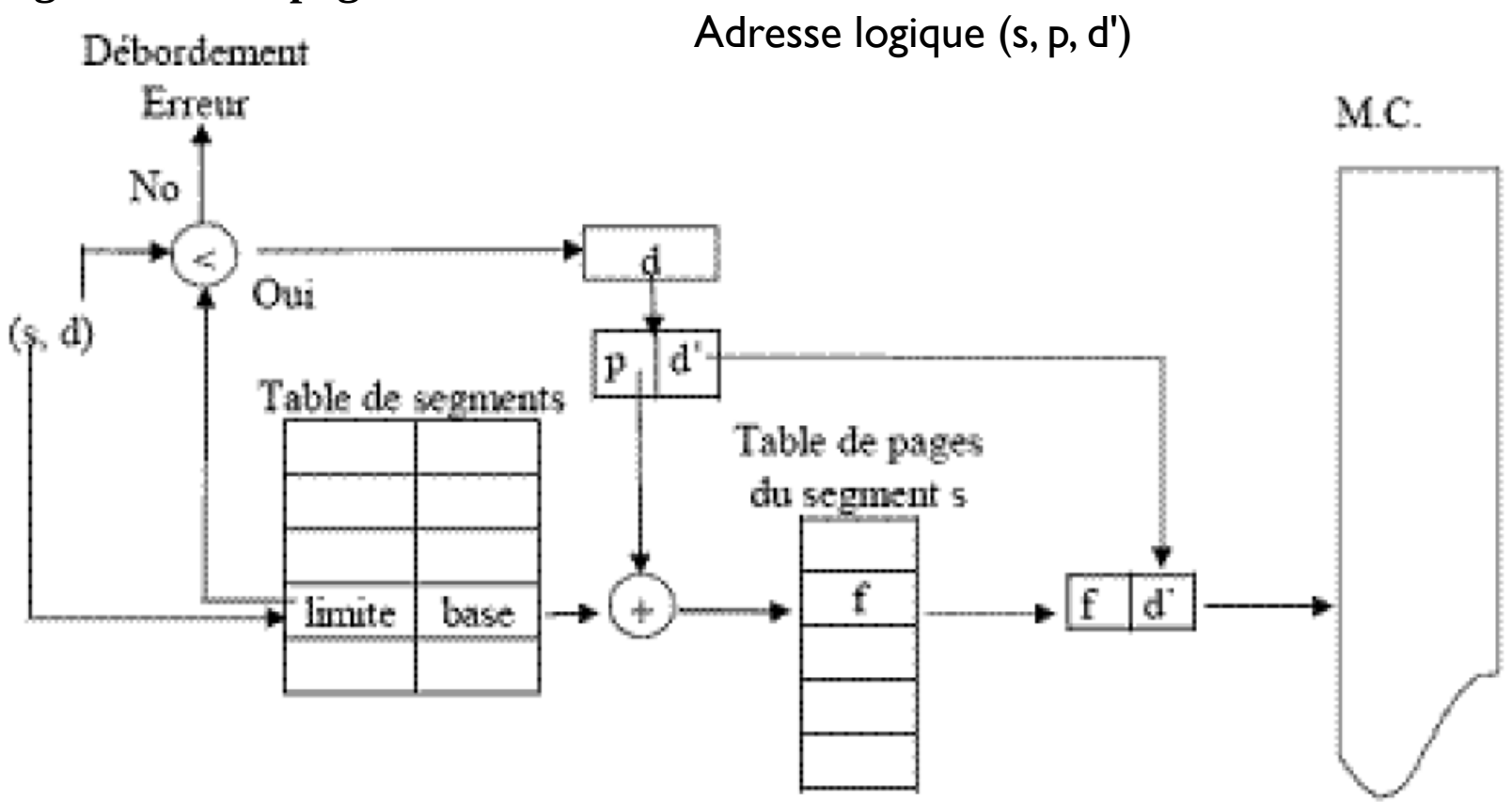
La table de page est divisée en portions . Chaque portion concerne un segment bien précis. La base permet de retrouver la portion concernant le segment donné, et p donne le numéro de page dans cette portion.

Gestion de la mémoire

3 - Allocation de la mémoire - Système Multiprogrammation

3 – 2- 2 Partitions multiples dynamiques

Segmentation - pagination



Gestion de la mémoire

Exercice 2 **Algorithmes de remplacement de pages:**

Un processus possède un espace d'adressage de 600 mots. Soit la suite des adresses logiques suivants :

34; 123; 145; 510; 456; 345; 412; 10; 14; 12; 234; 336; 412.

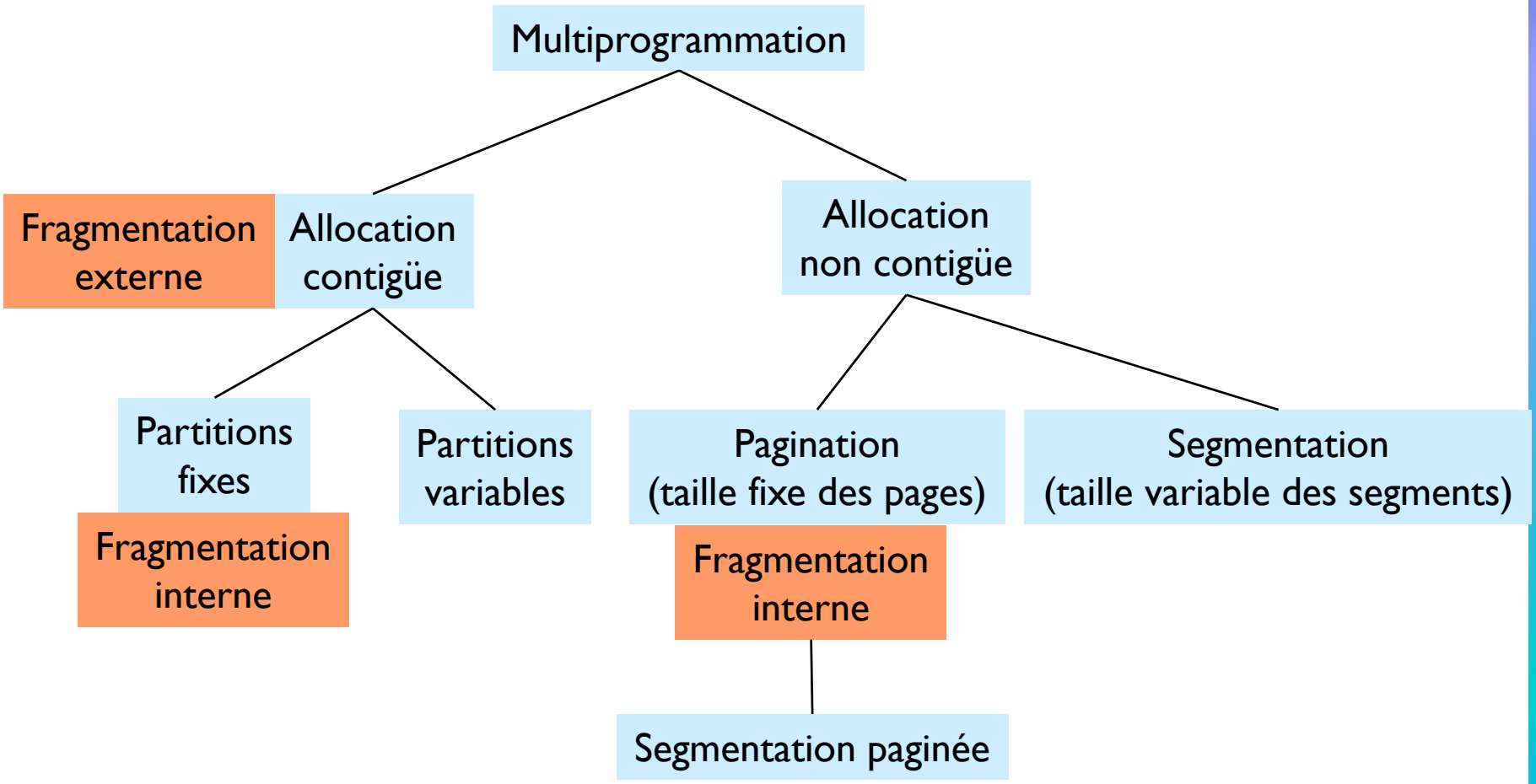
1. Donner la suite des numéros de pages références, sachant que chaque page comporte 100 mots.

Ce processus dispose de 300 mots en mémoire centrale.

2. Calculer le taux de défauts de page (en supposant la mémoire initialement vide) pour les algorithmes :

a) OPT b) FIFO c) LRU

Gestion de la mémoire



Gestion de la mémoire

Exercice 2: Solution

OPT :

Req	**0**	**1**	**1**	**5**	**4**	**3**	**4**	**0**	**0**	**0**	**2**	**3**	**4**
C_1	0	0	0	0	0	0	0	0	0	0	2	2	2
C_2	-	1	1	1	4	4	4	4	4	4	4	4	4
C_3	-	-	-	5	5	3	3	3	3	3	3	3	3
Def	X	X		X	X	X					X		

Gestion de la mémoire

Exercice 2: Solution

FIFO :

Req	**0**	**1**	**1**	**5**	**4**	**3**	**4**	**0**	**0**	**0**	**2**	**3**	**4**
C_1	0	0	0	0	4	4	4	4	4	4	2	2	2
C_2	-	1	1	1	1	3	3	3	3	3	3	3	4
C_3	-	-	-	5	5	5	5	0	0	0	0	0	0
Def	X	X		X	X	X		X			X		X

Gestion de la mémoire

Exercice 2: Solution

LRU :

Req	0	1	1	5	4	3	4	0	0	0	2	3	4
C_1	0	0	0	0	4	4	4	4	4	4	4	3	3
C_2	-	1	1	1	1	3	3	3	3	3	2	2	2
C_3	-	-	-	5	5	5	5	0	0	0	0	0	4
Def	X	X		X	X	X		X			X	X	X

Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master GSI
Semestre 1

Systèmes d'exploitation 2

Gestion de périphériques

A. ABBAS
abbasakli@gmail.com

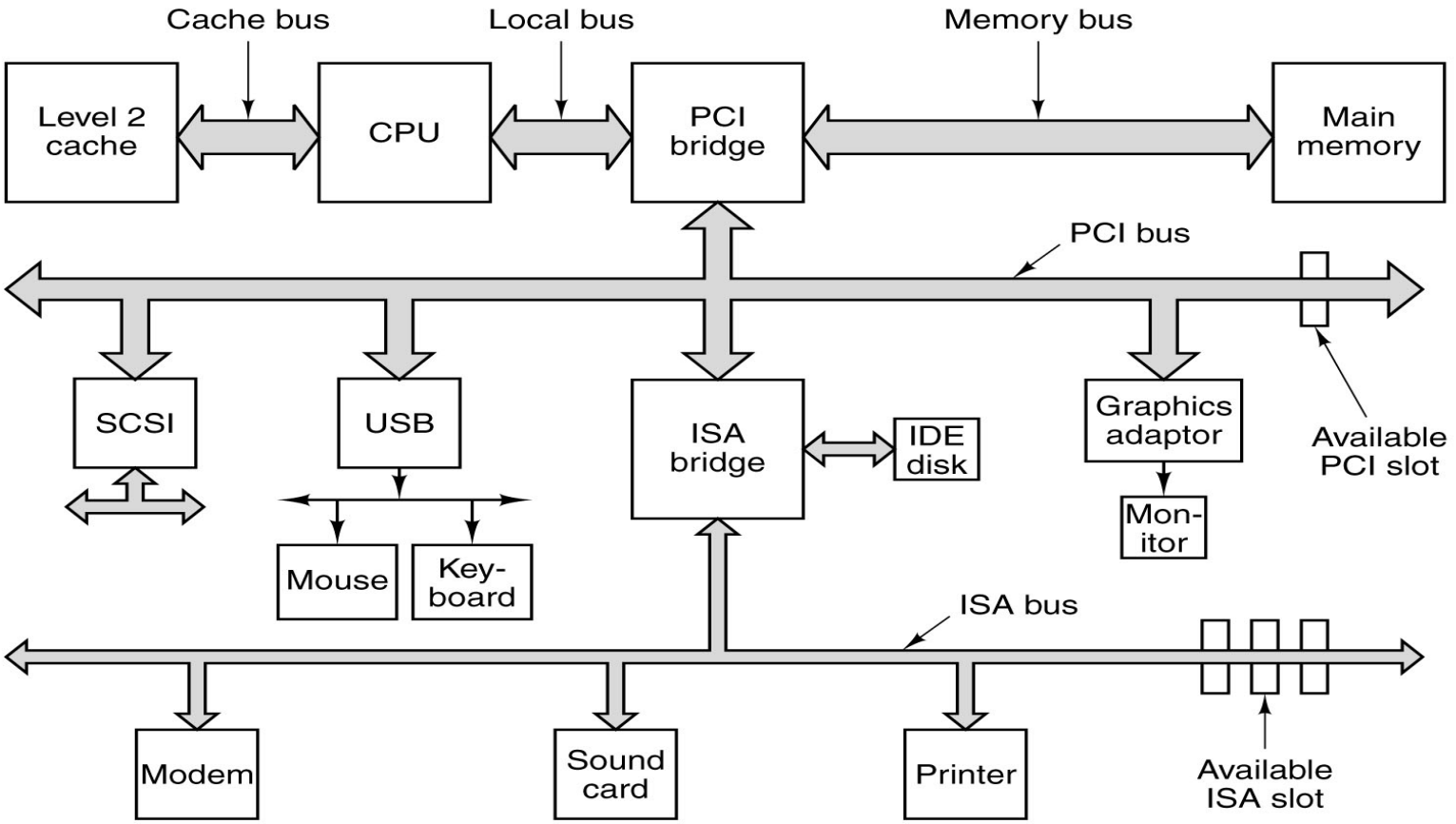
Gestion de périphériques

• Chaque ordinateur dispose de périphériques d'E/S tels que les écrans, des disques, les imprimantes, les équipements de communication, etc.
• Les périphériques d'E/S permettent le stockage de données et les échanges d'information avec le monde extérieur.
• Ils ont des caractéristiques différentes.
• On distingue plusieurs catégories :
- Périphériques par blocs (disques, stockent des informations dans des blocs de taille fixe).
- Périphériques par caractères ou alphanumériques (acceptent ou fournissent un flot de caractères).
- Périphériques réseaux

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Gestion de périphériques

• Le système d'exploitation (SE) gère, contrôle et coordonne tous les composants de l'ordinateur (CPU, mémoire, bus, périphériques d'E/S).
• Ces composants sont connectés par un (ou plusieurs) bus et communiquent entre eux via ce(s) bus.



Gestion de périphériques

• Le gestionnaire de périphériques d'E/S gère et contrôle tous les périphériques d'E/S.

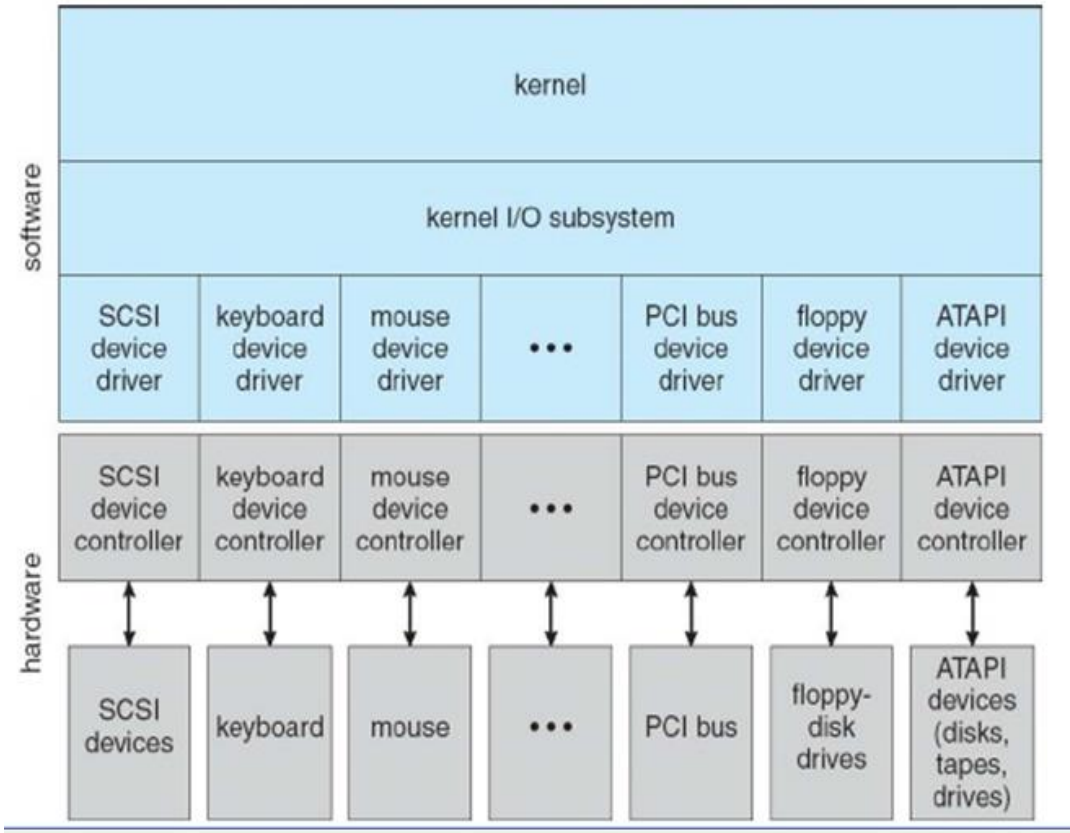
• Objectifs d'un gestionnaire de périphériques d'E/S :

1. Offrir une interface complète, uniforme et simple d'emploi qui abstrait la complexité de fonctionnement des différents périphériques.

2. Faciliter les références aux périphériques .

3. Être performant (maximiser les taux d'utilisation de tous les composants de l'ordinateur, optimiser les coûts des opérations d'E/S, …).

4. Détecter et gérer les erreurs d'E/S (isoler l'impact des erreurs, recouvrir les erreurs, uniformiser les codes d'erreur, …).



5. Permettre le partage des périphériques tout en assurant leur consistance et protection.

Gestion de périphériques

• Chaque périphérique a son propre pilote (un pilote par type de périphérique) qui dialogue avec lui.

• Trois façons d'intégrer un pilote à un SE :

1. Ajouter le code au noyau, refaire l'édition de liens et redémarrer le système (UNIX).

2. Placer une nouvelle entrée dans un fichier spécial indiquant que le système a besoin de ce pilote et redémarrer le système (Windows)

3. Intégrer le pilote à la volée sans nécessiter le redémarrage (ceux de USB).

Gestion de périphériques

Qu'est-ce qu'un pilote de périphérique ?

Un pilote de périphérique (device driver) permet à un programme utilisateur d'accèder à des ressources externes. Ce ressources peuvent être de type:

matérielles: carte d'extension, périphérique sur la carte mère

logicielles: mémoire physique

Il existe différents types de pilotes suivant le périphérique à controler:

- les pilotes en mode caractère (char drivers):
  - ce type de pilote est le plus simple à mettre en œuvre.
  - Il permet de dialoguer avec le périphérique en échangeant un nombre variable d'informations binaires. Les ports séries et parallèles sont des exemples de périphériques controlés en mode caractère.
- les pilotes en mode bloc (block drivers):
  - ces pilotes échangent des informations avec les périphériques uniquement par blocs de données. Un exemple d'un tel périphérique est le disque dur.
- les pilotes réseau (network drivers): ces pilotes sont destinés à contrôler des ressources réseau.

Gestion de périphériques

Les pilotes – cas linux

- Sous Linux, les pilotes de périphériques font partie du noyau et peuvent être intégrés de façon statique à celui-ci ou chargés à la demande sous forme de **modules**.

- Les pilotes de périphériques s'exécutent comme s'ils faisaient partie du noyau et ne sont pas accessibles directement aux processus utilisateur.

- Linux propose un mécanisme à ces processus pour communiquer avec un pilote (donc avec le matériel) via des objets semblables aux fichiers.

- Les programmes peuvent communiquer avec des dispositifs matériels via des objets semblables aux fichiers soit en utilisant les opérations d'E/S de bas niveau de Linux , soit les opérations de la bibliothèque d'E/S standard du C.

Gestion de périphériques

Les pilotes – cas linux

➢ Pour accéder à un pilote on crée donc un fichier spécial appelé également nœud (node), localisé dans le répertoire /dev, sur lequel on appliquera les appels système open, read, write, close …

Exemple : Liste tous les fichiers spéciaux

root@a:/dev# ls -l /dev

L'exemple ci-dessous donne la liste des fichiers spéciaux associés aux pilotes des ports séries:

a@a:/dev$ ls -l /dev/tty*

```
crw--w---- 1 root tty     4,  0 janv.  7  2017 /dev/tty0
crw--w---- 1 root tty     4,  1 janv.  7  2017 /dev/tty1
crw--w---- 1 root tty     4, 10 janv.  7  2017 /dev/tty10
crw--w---- 1 root tty     4, 11 janv.  7  2017 /dev/tty11
```

Gestion de périphériques

Les pilotes – cas linux

Ces pilotes sont caractérisés par deux valeurs numériques:

- **Le *MAJEUR* (MAJOR)** : qui identifie le pilote correspondant au périphérique ou représente le pilote qui gère le fichier spécial

le numéro majeur 3 pour les disques IDE, 6 pour les imprimantes,

la commande cat /proc/devices donne, pour les modes caractère et bloc, les numéros majeurs actuellement reconnus par les drivers chargés en mémoire.

- **Le *MINEUR* (MINOR)** qui représente une sous-adresse en cas de présence de plusieurs périphériques identiques, contrôlés par un même pilote.

L'exemple ci-dessous donne la liste des fichiers spéciaux associés aux pilotes des ports séries:

```
a@a:/dev$ ls -l /dev/tty*
crw--w---- 1 root tty     4,  0 janv.  7  2017 /dev/tty0
crw--w---- 1 root tty     4,  1 janv.  7  2017 /dev/tty1
crw--w---- 1 root tty     4, 10 janv.  7  2017 /dev/tty10
crw--w---- 1 root tty     4, 11 janv.  7  2017 /dev/tty11
```

Dans ce cas la, le majeur vaut 4 (les ports séries) et les mineurs vont 0, 1, 10, et 11.

Gestion de périphériques

Les pilotes – cas linux

Pour Quoi sert le numéro majeur ?

- Lorsque le driver des disques durs IDE démarre par exemple, il informe le noyau qu'il va s'occuper de tous les fichiers spéciaux en mode bloc et de numéro majeur égal à 3.

- Le noyau tient à jour une table des drivers et ainsi, il sait quel driver appeler lorsqu'un appel système tel que open(), read() ou write() est lancé sur un fichier spécial.

Gestion de périphériques

Les pilotes en mode caractère – cas linux

Création d'un fichier périphérique

Pour créer une nouvelle entrée dans le répertoire */dev*, on utilisera la commande *mknod*:

mknod /dev/monpilote mode majeur mineur

Le *mode* (valeur =c) indique que l'on crée un fichier pour un pilote en mode caractère. Si l'on désirait créer une entrée pour un pilote en mode bloc, on utiliserait la même commande avec un *b* à la place du *c*.

La liste des majeurs utilisés par le noyau LINUX est disponible dans le fichier /proc/*devices.txt* de l'arborescence des sources du noyau.

Exemple :

root@a:/dev# mknod pilot c 5 0

Gestion de périphériques

Les pilotes – cas linux

A partir du moment ou le noeud est créé et le pilote installé, on peut accèder au périphérique en utilisant les commandes standards de LINUX comme:

pour écrire sur le périphérique.

```
echo "une commande" > /dev/pilote
```

Exemple :

```
root@a:/dev# echo ABCDEF > /dev/pilot
ABCDEF
```

pour lire 10 caractères du péripéhrique.

```
dd bs=1 count=10 < /dev/pilote
```

Gestion de périphériques

Les pilotes – cas linux

On peut bien entendu accéder au périphérique en utilisant des langages évolués comme le C, exemple:

pour ouvrir un descripteur de fichier associé au périphérique.

```
int fd = open ("/dev/pilote", O_RDWR);
```

pour écrire une chaîne sur le périphérique.

```
char *s = "une commande";        write (fd, s, strlen(s));
```

pour lire n caractères du périphérique.

```
char buf[256];                   read (fd, buf, n);
```

pour terminer l'accès au périphérique.

```
close (fd);
```

Gestion de périphériques

Les pilotes – cas linux

Les sources des pilotes se trouvent généralement dans
/usr/src/kernels/linuxxxx/drivers

Accès à un pilote

Un pilote en mode caractère contiendra un certain nombre de point d'entrées associés aux requêtes des programmes utilisateurs. Ces requêtes sont les primitives qui sont utilisées pour accéder aux fichiers :

une méthode *open* : associée à l'ouverture du périphérique (détection, initialisation du périphérique…)

une méthode *read* : associée à la lecture qui permet de lire des données sur le périphérique (dans l'espace noyau) puis de faire passer ces données dans l'espace utilisateur appelant.

Gestion de périphériques

Les pilotes – cas linux

Accès à un pilote

une méthode *write* : associée à l'écriture qui permet de passer des données de l'espace utilisateur à l'espace noyau puis d'envoyer les données au périphérique.

une méthode *close* : associée à la fermeture d'accès (action matérielle nécessaire à la fermeture du périphérique…)

Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master GSI
Semestre 1

Systèmes d'exploitation 2

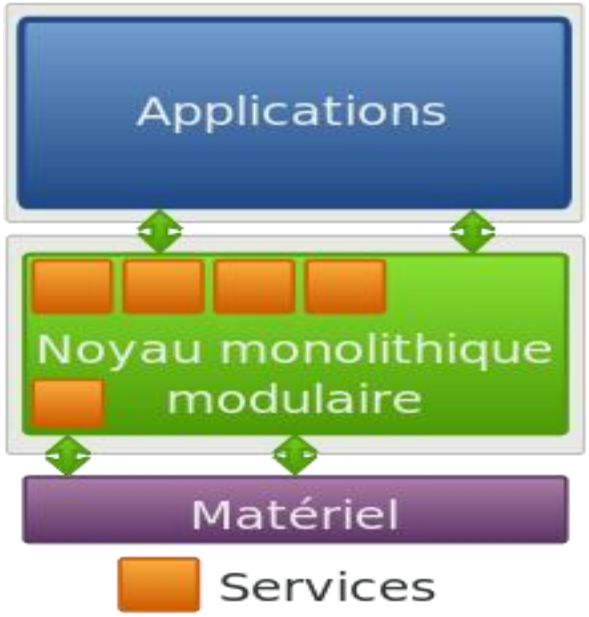
Modules chargeables

Présentation générale des Systèmes d'Exploitation

1. Les systèmes monolithiques modulaires

Dans ce type de noyau, seules les parties fondamentales du système sont regroupées dans un bloc de code unique (monolithique).

Les autres fonctions, comme les pilotes matériels, sont regroupées en différents modules qui peuvent être séparés tant du point de vue du code que du point de vue binaire.



Par exemple avec le noyau Linux, certaines parties peuvent être non compilées ou compilées en tant que modules chargeables directement dans le noyau.

Une erreur dans un module met en danger la stabilité de tout le système.

Exemple :

Linux > 1.2, OS/2, Unix SysVr4 / SunOS 5

Modules chargeables

Qu'est-ce qu'un Module ?

- Morceau de code permettant d'ajouter des fonctions au noyau
  - pilotes de périphériques
  - appels systèmes
  - protocole réseau
- Un module est chargé dynamiquement dans le noyau sans recompilation et sans redémarrage
- Un module peut aussi être déchargé
- Un module s'exécute avec les droits du noyau

Les modules chargeables permettent d'avoir un noyau le plus petit que possible, chargeant à la demande ce dont il a besoin :

- soit d'une manière manuelle par le super utilisateur de la machine,
- soit d'une manière automatique. De cette manière, le gain de ressources est non négligeable.

Modules chargeables

Première méthode : à la main

Le chargement manuel est basé sur trois commandes:

insmod : insère un module dans le noyau;

rmmod : décharge un module, si aucun processus ne l'utilise;

lsmod : affiche la liste des modules chargés

Exemple :

```
a# insmod pilot.o
a# lsmod
            Module:          #pages:          Used by:
             pilot              12               4
```

Modules chargeables

Automatisation : kerneld

Le système de chargement automatique de modules permet de réduire au minimum la taille de son noyau. Le principe de fonctionnement est particulièrement simple :

Un démon en mode utilisateur est à l'écoute des ordres du noyau (via une file de message de type IPC Système V).

Lorsque un processus essaye d'accéder à une ressource système (via un appel système open, etc...), le noyau envoie l'ordre de chargement du module à **kerneld**

Une fois le message reçu, **kerneld** exécute un modprobe pour charger les modules nécesaires

Modules chargeables

Première méthode : à la main Premier module

Soit fichier “pilote.c”:

```
#include <linux/module.h>
#include <linux/string.h>
int init_module (void)
{
     printk ("module pilote charge en memoire\n");
     return 0;
}
int cleanup_module (void)
{
   printk ("Goodbye World\n");
            return 0;
}
```

Contient les macros définit dans les sources du noyaux

fonction appelée lors du chargement du module

fonction appelée lors du déchargement du module

Modules chargeables

Première méthode : à la main Premier module

```
// Fichier Makefile
obj-m +=pilote.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(pwd) modules
clean :
        make -C /lib/modules/$(shell uname -r)/build M=$(pwd)  clean
```

Compiler le Fichier pilote1.c

```
root@a:#  -C /lib/modules/$(uname -r)/build M=$(pwd) modules
```

```
root@a:# insmod pilote.ko
root@a:# lsmod
```

Modules chargeables

Enregistrement du nombre majeur

- Les nombres majeurs doivent être enregistrés auprès du noyau pour que celui-ci fasse la liaison entre les fichiers spéciaux et le pilote.

- Nous ne pouvons pas choisir celui qui nous plaît, nous risquerions d'avoir des problèmes de conflit.

- Il y a donc des plages réservées qui sont définies dans le fichier :
  « /usr/src/linux/Documentation/devices.txt».

- Il existe une méthode pour obtenir le nombre majeur dynamiquement en fonction de ceux déjà utilisés.

- Une fois l'enregistrement terminé, le module possède une entrée dans le fichier « /proc/devices » et peut utiliser les fichiers spéciaux y faisant référence.

- Une fois que nous avons terminé l'utilisation de notre module, il faut libérer le « nombre majeur » obtenu qui pourra alors être utilisé à son tour par un autre pilote.

Modules chargeables

Enregistrement du module dans le noyau

```
#include <linux/version.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/pci.h>
#define CHRDEV 242
static /*const*/ struct file_operations ops = {
        read:       NULL,
        write:      NULL,
        open:       NULL,
        release:    NULL,};
static int init_drv(void){
printk("module pilote2 charge en memoire\n");
        register_chrdev(CHRDEV,"pilote2",&ops);
        return 0;}
static void exit_drv(void){
        printk("module pilote2 decharge de la memoire\n");
        unregister_chrdev(CHRDEV,"pilote2");}
module_init(init_drv);
module_exit(exit_drv);
```

La fonction *register_chrdev* enregistre le module dans le noyau sous le numéro CHRDEV (242),

Création du fichier d'accès :
mknod /dev/pilote2 c 242 0

Modules chargeables

Accès au pilote par les primitives de gestion de fichiers

```
#include <linux/version.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/pci.h>
#define CHRDEV 243

static int ouverture(struct inode *inode, struct file *file){
    printk("ouverture du module pilote3 dans le mode : %d\n",file->f_mode);
    MOD_INC_USE_COUNT;    // macro pour incrémenter le nombre d'ouvertures
return 0;}

static int fermeture(struct inode *inode, struct file *file){
printk("fermeture du module pilote3\n");
MOD_DEC_USE_COUNT;  // macro pour décrémenter le nombre d'ouvertures
return 0;
}
```

Modules chargeables

Accès au pilote par les primitives de gestion de fichiers

```
static ssize_t ecriture(struct file *file, const char *buffer, size_t count, loff_t *ppos)
{
        printk("ecriture dans pilote3\n");
        return 0;
}

static ssize_t lecture(struct file *file, char *buffer, size_t count, loff_t *ppos)
{
        printk("lecture dans pilote3\n");
        return 0;
}
```

Modules chargeables

Accès au pilote par les primitives de gestion de fichiers

```
static /*const*/ struct file_operations ops =        {
                                    read:            lecture,
                                    write:           ecriture,
                                    open:            ouverture,
                                    release:         fermeture,
};
static int init_drv(void){
        printk("module pilote3 charge en memoire\n");
        register_chrdev(CHRDEV,"pilote3",&ops);
        return 0;
}
static void exit_drv(void){
        printk("module pilote3 decharge de la memoire\n");
        unregister_chrdev(CHRDEV,"pilote3");
}
module_init(init_drv);
module_exit(exit_drv);
```

Modules chargeables

Accès au pilote par les primitives de gestion de fichiers

```
#include <fcntl.h>

main(){
int id = open ("/dev/pilote3", O_WRONLY);

printf("ouverture de /dev/pilote3 (descripteur : %d)\n",id);

write (id, "bonjour",7);

close (id);
}
```