

Systèmes d'exploitation

Plan

- I. Introduction**
- II. Gestion de Processus**
- III. Ordonnancement**
- IV. Gestion de la mémoire**
- V. Système de gestion de fichiers**

Introduction

- **Chapitre 1**
 - ❑ Définition
 - ❑ Fonctions
 - ❑ Interfaces d'un SE
 - ❑ Evolution des Ses
 - ❑ Types

Concepts importants du Chapitre 1

- Par lots
- Multiprogrammés – balance de travaux
- À partage de temps (time-sharing)
- Parallèles:
 - Fortement couplés
 - Symétriques,
 - Asymétriques: maître-esclave
 - Faiblement couplés:
 - Répartis
 - Réseaux
- **Caractéristiques de matériel et logiciel requises pour cette évolution**
- **Systèmes à temps réel: durs, souples**

Système d'exploitation (SE)

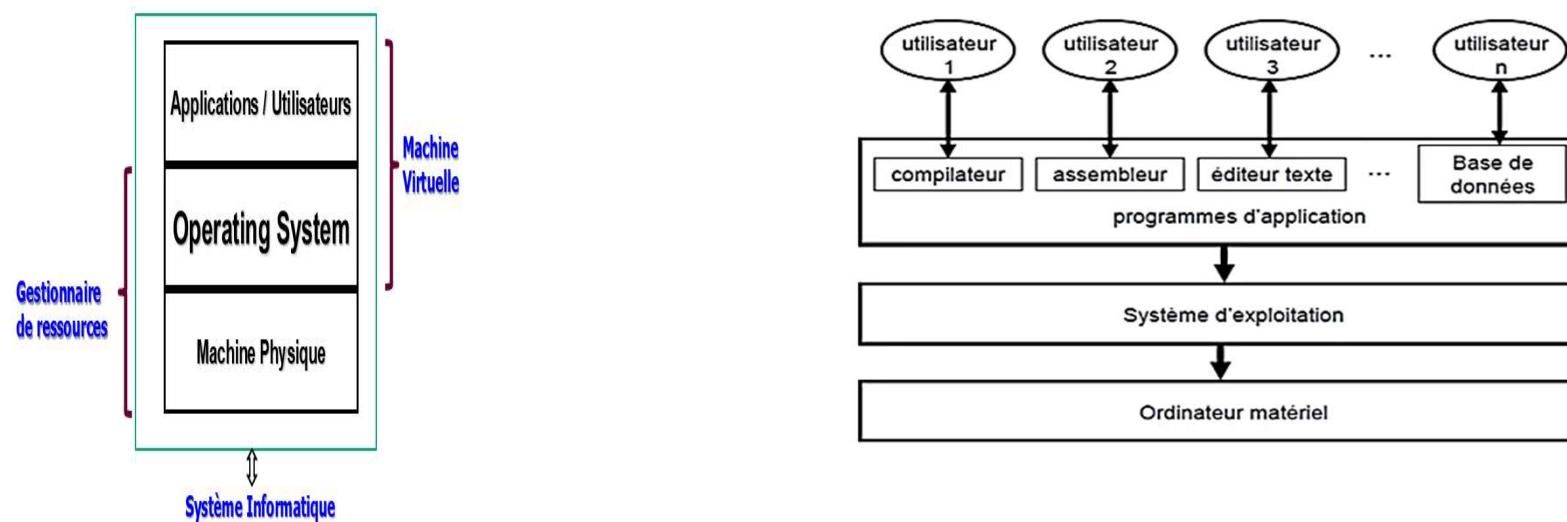
❖ Définition

Qu'est ce qu'un système d'exploitation?

- Un système peut être défini comme un ensemble de programmes qui contrôle et dirige l'utilisation d'un ordinateur à travers différentes applications.
- En autres, c'est un système qui exploite les ressources matérielles d'un ordinateur et assure la liaison entre les utilisateurs et les applications.
- C'est le premier programme exécuté lors du démarrage de l'ordinateur.
- Le système d'exploitation peut être noté SE ou OS (Operating System: traduction en anglais)
- Il optimise l'utilisation des ressources pour maximiser la performance du système

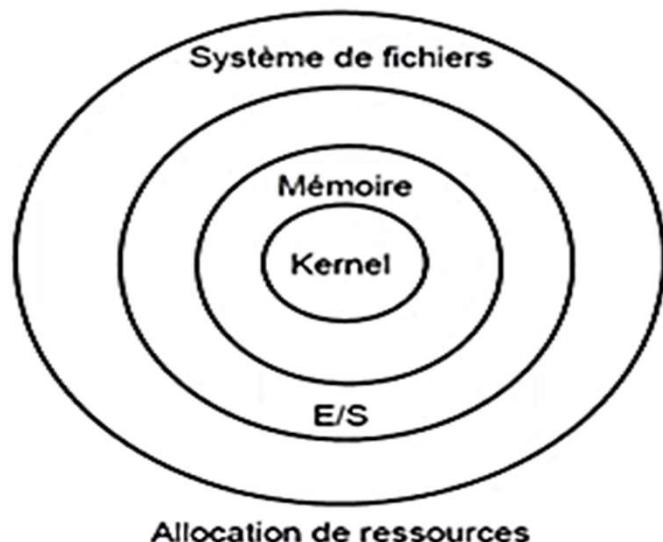
Vue abstraite d'un SE

- Le système d'exploitation est un **programme système** qui **gère et contrôle** les composants de l'ordinateur. Il fournit une base appelée **machine virtuelle**, sur laquelle seront construits les programmes d'application et les utilitaires au moyen des services ou **appels système**.



Les fonctions d'un système d'exploitation

- Chargement et lancement des programmes.
- Gestion des processeurs, de la mémoire, des périphériques.
- Gestion des processus (programmes en cours d'exécution) et des fichiers.
- Protection contre les erreurs et la détection des erreurs, etc.



Structure en couches d'un SE

Ressources et leur gestion

- **Ressources:**
 - physiques: mémoire, unités E/S, UCT...
 - Logiques = virtuelles: fichiers et bases de données partagés, canaux de communication logiques, virtuels...
 - les ressources logiques sont bâties par le logiciel sur les ressources physiques
- **Allocation de ressources: gestion de ressources, leur affectation aux usagers qui les demandent, suivant certains critères**

Pourquoi étudier les SE?

- **Logiciel très important...**
 - tout programme roule sur un SE
 - interface usager-ordinateur
- **Les SE utilisent beaucoup d'algorithmes et structures de données intéressants**
 - Les techniques utilisées dans les SE sont aussi utilisées dans nombreuses autres applications informatiques
 - il faut les connaître

Composants d'OS

Gestion de Processus

Créer, gérer, synchroniser, établir la communication entre eux

Gestion de Mémoire

Réserver et Tracker les données sur la mémoire centrale

Système de Fichiers

Gérer la façon de stocker les informations et de les organiser

Gestion Entrées Sorties

Gérer l'échange de données entre le périphériques et le processeur

Interfaces d'un système d'exploitation

Un système d'exploitation présente en général **deux** interfaces :

Interface "programmatique", ou API (Application Programming Interface) :

utilisable à partir des programmes s'exécutant sous le système.

composée d'un ensemble d'appels systèmes (appels de procédures, avec paramètres)

Interface de l'utilisateur, ou interface de commande :

utilisable par un usager humain, sous forme textuelle ou graphique.

composée d'un ensemble de commandes

textuelles (exemple en Unix : rm *.o)

graphiques (exemple : déplacer l'icône d'un fichier vers la corbeille)

```
...
while (bytesread = read(from_fd, buf, BLKSIZE
)) {
if ((bytesread == -1) && (errno != EINTR))
break;
else if (bytesread > 0) {
bp = buf;
while(byteswritten = write(to_fd, bp,
bytesread )) {
if ((byteswritten == -1) && (errno !=
EINTR))
break;
else if (byteswritten == bytesread)
break;
else if (byteswritten > 0) {
bp += byteswritten;
bytesread -= byteswritten;
}
}
if (byteswritten == -1)
break;
}
}
cp fich1 fich2
```

Développement de la théorie des SE

- La théorie des SE a été développée surtout dans les années 1960 (!!)
- A cette époque, il y avait des machines très peu puissantes avec lesquelles on cherchait à faire des applications comparables à celles d'aujourd'hui (mémoire typique)
 - machines devaient parfois desservir des dizaines d'usagers!
- Dont le besoin de développer des principes pour optimiser l'utilisation d'un ordinateur.
- Principes qui sont encore utilisés

Évolution historique des SE

1. Aperçu Historique:

- Les débuts (1946-1955): Premiers systèmes, Tubes à vide
- 1955-1965: Systèmes de traitement par lot, transistors
- 1965-1980: Systèmes multiprogrammés et à temps partagé, Circuits intégrés
- 1980-1990: Systèmes des ordinateurs personnels, LSI
- VLSI (1990...) Systèmes « micro-noyau » .

Premiers systèmes (1945–1955)

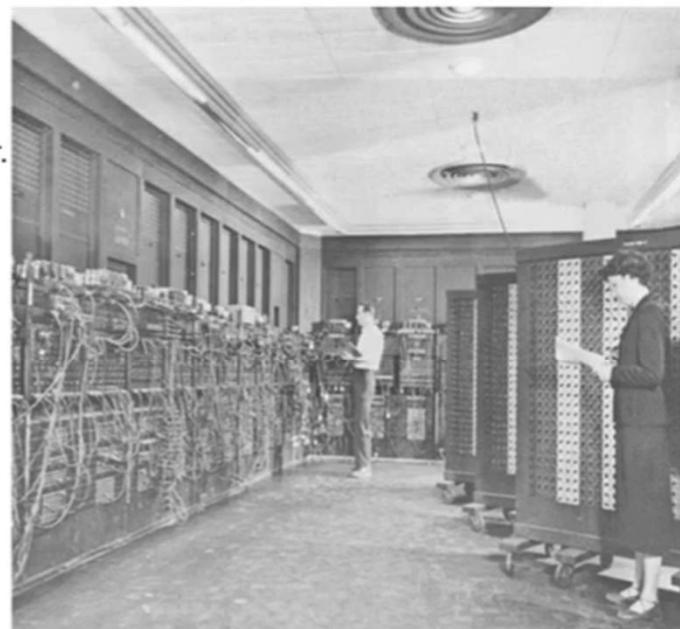
- Machines à tubes à vide volumineuses, très fragiles et très lentes
- Absence de système d'exploitation
- Calculs simples (tables des sinus et cosinus)
- Programmation en langage machine sur des cartes enfichables ou cartes perforées
- Construction, programmation et maintenance effectuées par un seul groupe
- Système mono-usager (réservation)
- Mauvaise utilisation de la CPU

Premiers systèmes (1945–1955) (1)

- porte ouverte : 1945-1955 :

- Technologie des tubes électroniques.
- Pas de système d'exploitation.
- Tranche de temps pour chaque programmeur.
- l'introduction de cartes perforées en 1950.

18000 tubes
1500 relais
70000 résistances
10000 condensateurs
170 M2 !!!!!



Machine ENIAC 1946

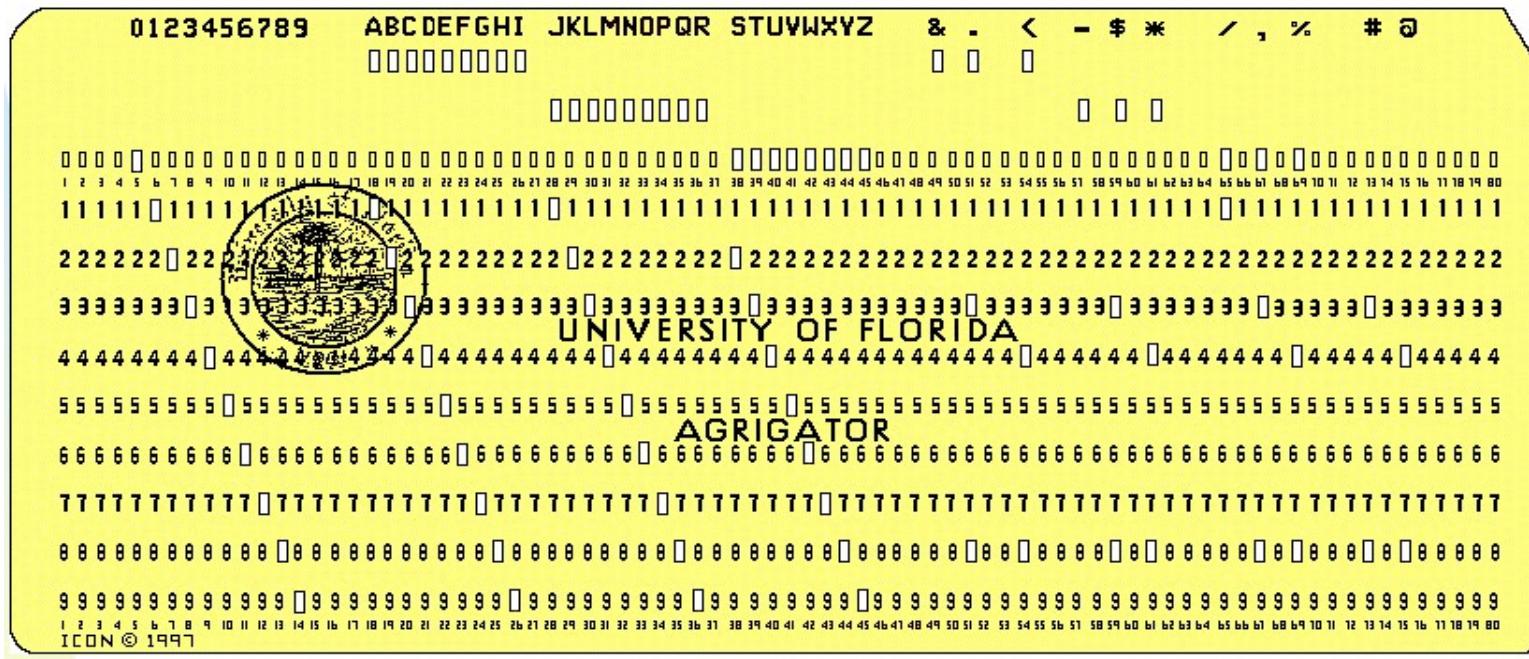
Systèmes de traitement par lots (1955–1965)

- Machines à transistors fiables mais coûteuses
- Programmation en Fortran et Assembleur
- Premiers logiciels de base : chargeur et compilateur Fortran
- Distinction entre opérateurs, constructeurs, programmeurs et utilisateurs
- Soumission des travaux
- Enchaînement des travaux par l'opérateur
- Inactivité de la CPU lors de la préparation
- Traitement par lots

Systèmes de traitement par lots

- Sont les premiers SE
- L'usager soumet une job à un opérateur
 - ❖ Programme suivi par données
- L'opérateur place un lot de plusieurs jobs sur le dispositif de lecture
- Un programme, le moniteur, gère l'exécution de chaque programme du lot
- Le moniteur est toujours en mémoire et prêt à être exécuté
- Un seul programme à la fois en mémoire, programmes sont exécutés en séquence
- La sortie est normalement sur un fichier, imprimante, ruban magnétique...

Les cartes perforées



Une ligne de données ou de programme était codée dans des trous qui pouvaient être lus par la machine

Opérateur lisant un paquet de cartes perforées



Le chargement successif

- Ralentir le temps d'exécution
- Diminuer et automatiser les chargements

Traitement par lots

Moniteur d'enchainement

Source:

http://www.tietokonemuseo.saunalahti.fi/eng/kuva_32_eng.htm

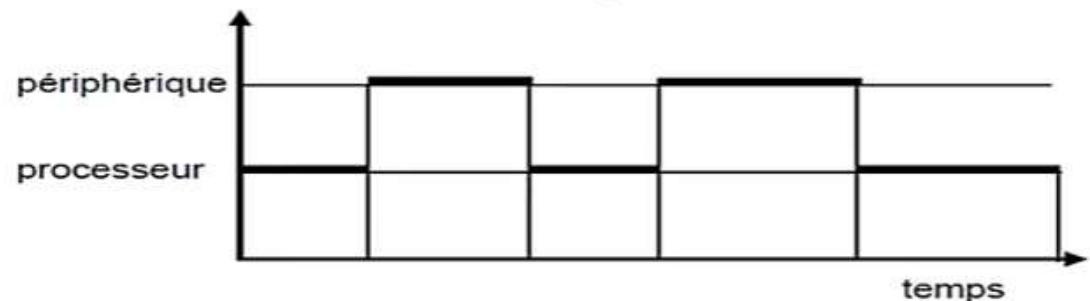
Finnish Data Processing Museum Association

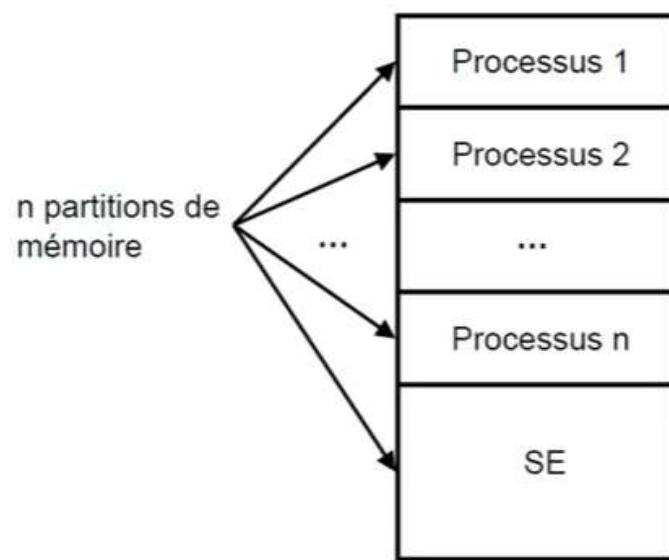
Traitement par lots : 1955-1965 :

- Introduction de la technologie des transistors.
- Utilisation des bandes magnétiques.
- Écriture des programme en Fortran ou assembleur.



Traitement par Lots



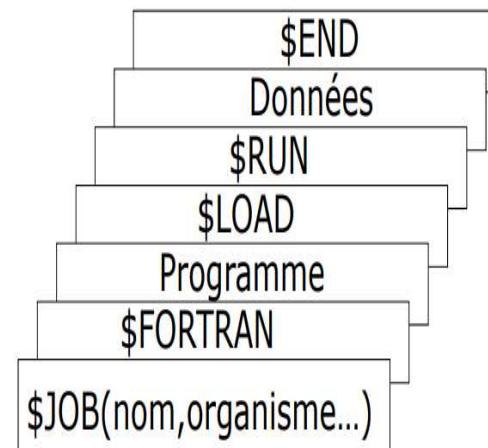


∴ *Partitions de mémoire.*

Langage de contrôle des travaux (JCL)

- Utilisé pour contrôler l'exec d'une job
 - le compilateur à utiliser
 - indiquer où sont les données
- Exemple d'une job:
 - paquet de cartes comme suit:
- **\$JOB début**
- **\$ FTN charge le compilateur FORTRAN et initie son exécution**
- **\$LOAD charge le programme objet (à la place du compilateur)**
- **\$RUN transfère le contrôle au programme usager**
- **les données sont lues par le moniteur et passées au progr. usager**

\$JOB
\$FTN
...
Programme
FORTRAN
...
\$LOAD
\$RUN
...
Données
...
\$END
\$JOB
...
(job suivant)



- L'E/S est déléguée au moniteur
- Chaque instruction d'E/S dans pgm usager invoque une routine d'E/S dans le moniteur:
- Quand le programme usager se termine, la prochaine ligne de JCL est lue et exécutée par le moniteur.

Le moniteur *par lots*

- Lecture de cartes perforées
- Interprétation de commandes JCL
- Lecture (load) d'une job (du lecteur de cartes)
- Chargement en mémoire
- Transfère le contrôle au programme usager
- Exécution du programme
- À ce point, le moniteur reprend le contrôle

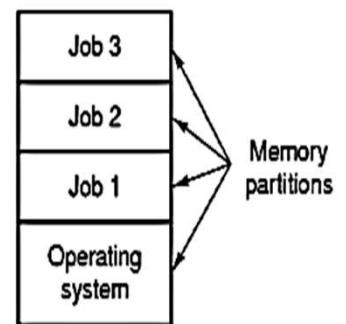
Traitement par lots : Inconvénients

- Mono-programmation
- CPU encore mal utilisée (attente des E/S)
- Machines incompatibles et SE différents :
 - IBM 7094 : calcul scientifique (equ. diff...)
 - IBM 1401 : traitement commercial (tri, impression...)
- ⇒ Coût de maintenance élevé

Systèmes multiprogrammés et à temps partagé (1965–1980) (1)

- Machines à circuits intégrés moins coûteuses et plus performantes
- Famille d'ordinateurs compatibles et SE unique (machines avec même architecture et jeu d'instructions : IBM 360, 370...)
- E/S spoolées : Utilisation de la MS au lieu des bandes magnétiques
- Apparition de la multiprogrammation
- Au lieu d'exécuter les travaux au fur et à mesure qu'ils sont lus, les stocker sur une mémoire secondaire (disque), Puis choisir quels programmes exécuter et quand.

- MULTIPROGRAMMATION: Permet de conserver en mémoire plusieurs travaux et de gérer le partage du processeur et des périphériques entre-eux.
- La mémoire est organisée en un ensemble de partitions (1 travail par partition).
- L'introduction des unités de disque permettent des accès direct aux travaux.
- L'arrivée des contrôleurs DMA (Direct Memory Access) qui se chargent de gérer les E/S des périphériques, libère le processeur de cette tâche.



Systèmes multiprogrammés et à temps partagé (1965–1980) (2)

Multiprogrammation et traitement par lots : 1965-1980

- L'introduction des circuits intégrés.
 - création d'une seule gamme de produits.
 - un même programme peut tourner sur toutes ces machines.
 - pas de machine d'E/S et d'autres pour le calcul.
- L'arrivée sur le marché des unités de disques :
 - Disparition de la manipulation de bandes.
 - transfert des travaux des cartes vers le disque.
 - Application de la technique de spool (Simultaneous Peripheral Operation On Line).
- Multiprogrammation : charger plusieurs programmes dans la MC et de les exécuter en fonction des ressources qu'ils utilisent.

Systèmes multiprogrammés et à temps partagé (1965–1980) (4)

Concept de partage équitable de temps:

- Le processeur attribue à chaque travail un certain temps limité.
- Au bout de ce temps, si le travail n'est pas terminé, il retourne dans la file des travaux prêts
- Le CPU passe alors au traitement du travail suivant avec la même procédure.

Ce mode d'exploitation donne l'impression que les travaux s'exécutent en même temps (pseudo parallélisme).

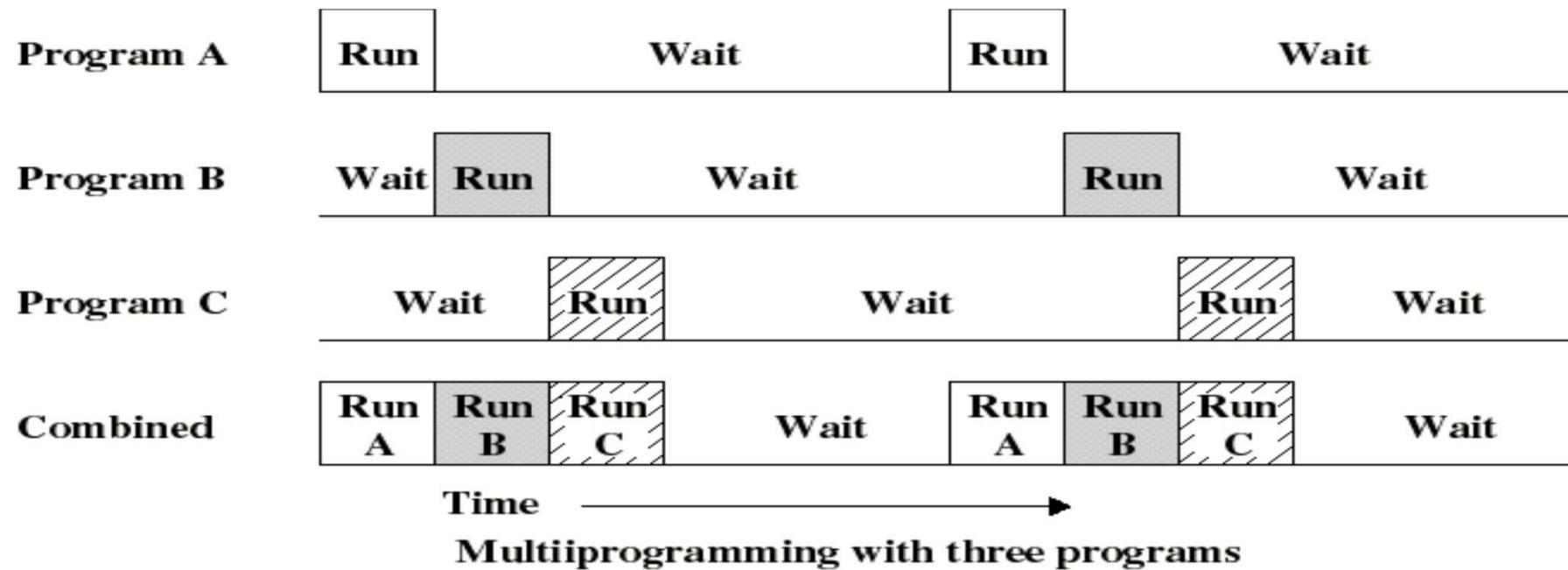
Traitement par lots multiprogrammé

- Les opérations E/S sont extrêmement lentes (comparé aux autres instructions)
 - Même avec peu d'E/S, un programme passe la majorité de son temps à attendre
- Donc: pauvre utilisation de l'UCT lorsqu'un seul pgm usager se trouve en mémoire



Traitement par lots multiprogrammé

- Si la mémoire peut contenir plusieurs programmes, l'UCT peut exécuter un autre programme lorsqu'un programme attend après E/S
- C'est la multiprogrammation



Exigences pour multiprogrammation

- **Interruptions**
 - afin de pouvoir exécuter d'autres jobs lorsqu'un job attend après E/S
- **Protection de la mémoire: isole les jobs**
- **Gestion du matériel**
 - plusieurs jobs prêts à être exécutées demandent des ressources:
 - ❖ UCT, mémoire, unités E/S
- **Langage pour gérer l'exécution des travaux: interface entre usager et OS**
 - jadis JCL, maintenant *shell*, *command prompt* ou semblables

Systèmes à temps partagé (time sharing)

- Partage de l'UC entre plusieurs tâches par quantum de temps \Rightarrow Traitement "simultané"
- Plusieurs utilisateurs connectés en ligne sont servis à la fois de façon interactive
- Systèmes d'exploitation complexes
 - CTSS *Compatible Time-Sharing System*
 - MULTICS (MULTplexed Information and Computing Service) ancêtre d'Unix
 - Unix (le plus porté)

Systèmes des ordinateurs personnels (1980-1990)

- Machines à circuits LSI (Large Scale Integration) : des transistors au cm²
- Machines moins coûteuses
- Ms-Dos, Windows, Mac OS, Linux...
- Apparition des systèmes centralisés en réseaux
- Apparition des systèmes distribués

SE en réseaux et SE répartis

- SE en réseaux (Windows, Linux...)
 - Chaque machine a son propre SE
 - Connexion à des machines distantes, transferts de fichiers
 - Machine utilisée par plusieurs utilisateurs
- SE répartis ou distribués
 - Système réparti sur un domaine
 - 1 Machine virtuelle (à plusieurs processeurs)
 - Transparence à la localisation des ressources
 - Fiabilité et tolérance aux pannes

Systèmes à temps réel

- Doivent réagir à ou contrôler des événements externes (p.ex. contrôler une usine). Les délais de réaction doivent être bornés
- **systèmes temps réel *souples*:**
 - les échéances sont importantes, mais ne sont pas critiques (p.ex. systèmes téléphoniques)
- **systèmes temps réel *rigides (hard)*:**
 - les échéances sont critiques, p.ex.
 - contrôle d'une chaîne d'assemblage
 - graphiques avec animation

Systèmes "micro-noyau" (1990...)

- Systèmes embarqués réalisés avec + ou - de modules (fonctions)
- Adaptés aux ordinateurs portables et de poche
 - PDA : Personal Digital Assistant
 - PIC : Personal Intelligent Communicator
- Palm OS, Windows CE...

Classes des systèmes

Selon les contraintes d'utilisation

- Mono-utilisateur/mono-tâche (MS-DOS)
 - Un seul utilisateur/une seule tâche à la fois
- Mono-utilisateur/multi-tâches(Windows XP)
 - Un seul utilisateur à la fois exécute plusieurs tâches simultanément
- Multi-utilisateurs/multi-tâches (Unix)
 - Plusieurs utilisateurs à la fois exécutent chacun plusieurs tâches simultanément et partagent les mêmes ressources matérielles

Classes des systèmes

Selon les services

- Systèmes temps réel
 - Utilisés dans des domaines spécifiques (procédés, robotique, centrales nucléaires...)
 - Temps de réponse des tâches critiques court
 - Fiables et tolérants aux pannes
- Systèmes transactionnels
 - Gestion des bases de données énormes (systèmes de réservation, systèmes bancaires...)
 - garantir des mises à jour sans incohérence

Classes des systèmes

Selon l'architecture matérielle

- Systèmes mono-processeur
 - Multiprogrammés et à temps partagé (pseudo-parallélisme)
- Systèmes parallèles : multiprocesseurs (SunOS 4, SunOS 5, Solaris 2 et Linux)
 - Traitement parallèle par plusieurs processeurs
 - Grande capacité de traitement, temps de réponse court et fiabilité

Modes d'exécution

- Mode utilisateur : non protégé
 - Exécution des programmes des utilisateurs
 - Permet à l'utilisateur de modifier des données de son programme
- Mode noyau : protégé et réservé au SE
 - Accès au code et données utilisées par le SE
 - Lecture et écriture sur les ports d'E/S
 - Permet de protéger les données sensibles

Calculs parallèles et Distribués

- Introduction
- Parallélisation sur mémoire distribuée.
- Parallélisation sur mémoire partagée

Introduction

- Architectures parallèles Pourquoi?
 - Il est très naturel de partager un gros travail parmi plusieurs personnes.
- En informatique, le parallélisme a rapidement considéré comme une option naturelle pour calculer.
- répondre à une forte demande
 - En puissance de calcul: simulation, modélisation
 - En puissance de traitement: base de données, serveurs multimédia
- **Problématique: Calculs trop gros, Calculs trop long**

Architectures parallèles Pourquoi?

Solutions :

- Approche classique: diminuer le temps de calcul
 - Matériels plus rapides: évolution processeurs et mémoire
- En calcul parallèle: Exécution simultané de plusieurs opérations (Tâches).
 - Meilleurs algorithmes
 - Machines parallèles

Architectures parallèles Pourquoi?

- Avantages :
 - Amélioration des performances de calcul
 - Accroissement de la taille des problèmes à résoudre
 - Résolution de nouveaux problèmes
- Problèmes
 - La remise en question des concepts d'algorithmique classique basés sur le principe de la machine séquentielle.
 - Diversité des modèles d'architectures parallèles
 - Difficulté de la programmation des machines parallèles

Algorithmique Parallèle Pourquoi?

- Approches de résolution de problèmes dans un contexte d'exécution parallèle.
- Modèles algorithmiques : contextes d'exécution parallèle simplifiés pour faciliter la conception.
- Analyse théorique de la performance

Définitions

- **Un programme séquentiel se caractérise par** l'exécution de plusieurs tâches l'un après l'autre avec un ordre prédéfini.
- **Un programme parallèle se caractérise par** l'exécution de plusieurs tâches distinctes ou non en même temps.
- **Un ordinateur parallèle : est une machine composée** de plusieurs processeurs qui coopèrent à la solution de même problème.
- **Un système distribué (ou réparti) est un système** de plusieurs processeurs impliqués dans la résolution d'un ou plusieurs problèmes.

Définitions

- Qu'est-ce que le parallélisme ?
 - Exécution d'un algorithme en utilisant plusieurs processeurs plutôt qu'un seul.
 - Division d'un algorithme en tâches pouvant être exécutées en même temps sur des processeurs différents.
- Le but : réduire le temps de résolution d'un problème en utilisant un ordinateur parallèle.
- 3 niveaux d'abstraction
 - **Architectures**
 - **Algorithmes**
 - **Programmation**

Définitions

- Le type de parallélisme: physique et logique
 - Le parallélisme **physique**
: Exécution de plusieurs tâches distincts ou égales.
 - Le parallélisme **logique** (pseudo-parallélisme): Exécution de plusieurs tâches par un seul processeur qui les traite alternativement

Systèmes parallèles (*tightly coupled*)

- Le petit coût des puces rend possible leur composition dans systèmes multiprocesseurs
- Les ordinateurs partagent mémoire, horloge, etc.

Avantages:

- travail plus fiable:

Systèmes parallèles

- **Symétriques**
 - Tous les UCTs exécutent le même SE
 - Elles sont fonctionnellement identiques
- **Asymétrique**
 - Les UCT sont des fonctionnalités différentes, par exemple il y a un *maître* et des *esclaves*.
- **Aujourd’hui, tout ordinateur puissant est un système parallèle.**

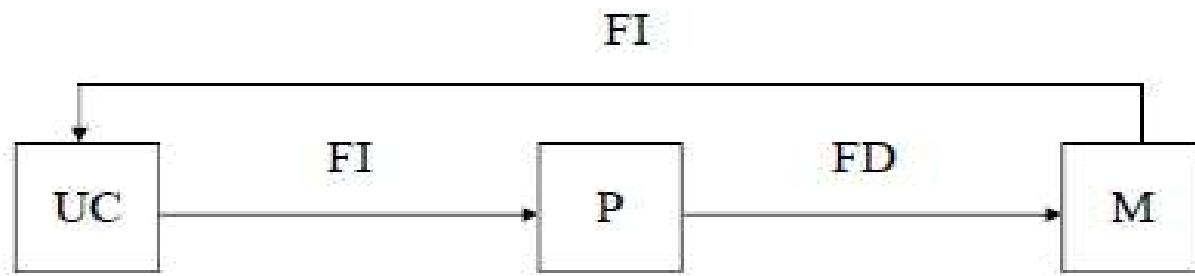
Architectures parallèles

- Plusieurs types d'architectures
 - Distribuées
 - Centralisées
- Modèles d'architecture : simplification du fonctionnement des ordinateurs parallèles
 - SIMD
 - MIMD
 - MIMD

Classification de Flynn

- En 1966 : basée sur la multiplicité du flot d'instructions et le flot de données
 - SISD : Single Instruction Single Data Stream
 - SIMD : Single Instruction Multiple Data Stream
 - MISD : Multiple Instruction Single Data Stream
 - MIMD : Multiple Instruction Multiple Data Stream.

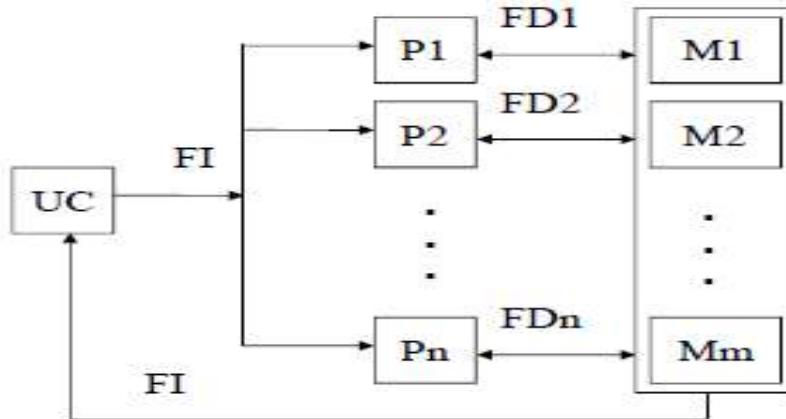
Classification de Flynn : SISD



UC : unité de contrôle
P : unité de processeur
M : module mémoire
FI : flot d'instruction
FD : flot de données

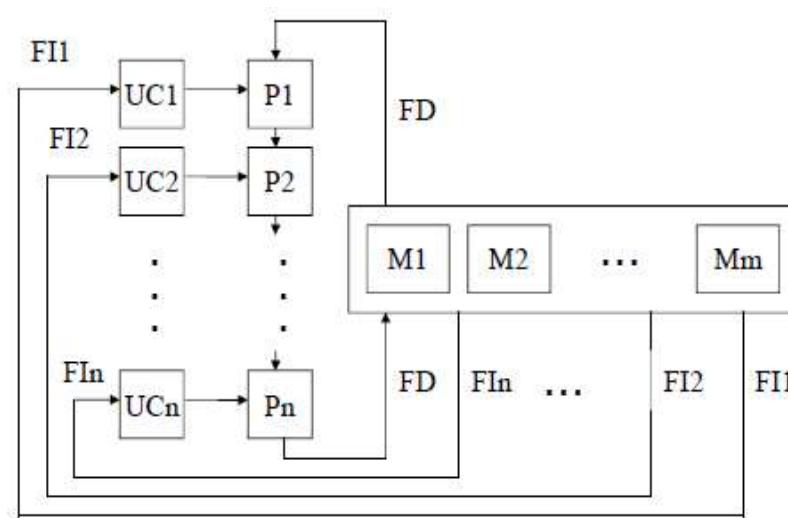
SISD ou encore **single instruction on single data** est un terme désignant une architecture matérielle dans laquelle un seul processeur exécute un seul flot d'instruction sur des données résidant dans une seule mémoire. De ce fait, il n'y aucune parallélisation. On retrouve là l'**architecture de von Neumann**.

Classification de Flynn : SIMD



Single Instruction on Multiple Data (signifiant en anglais : « instruction unique, données multiples »), ou **SIMD**, désigne un mode de fonctionnement des ordinateurs dotés de capacités de parallélisme. Dans ce mode, la même instruction est appliquée simultanément à plusieurs données pour produire plusieurs résultats

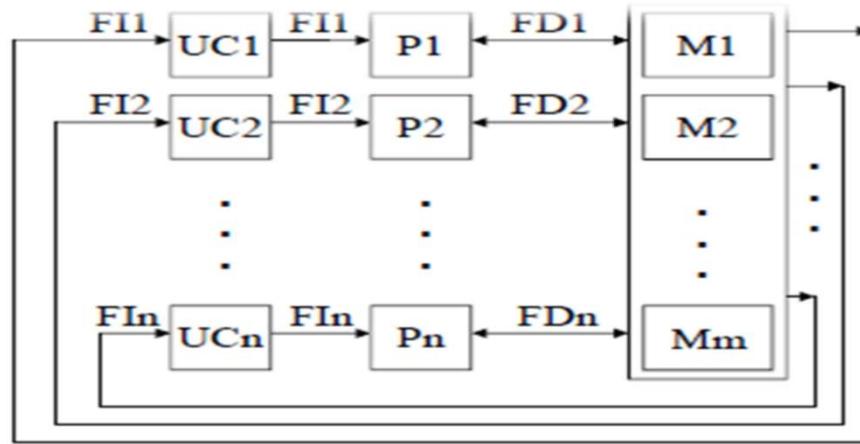
Classification de Flynn : MISD



Multiple instructions single data (ou **MISD**) désigne un mode de fonctionnement des ordinateurs dotés de plusieurs unités arithmétiques et logiques fonctionnant en parallèle.

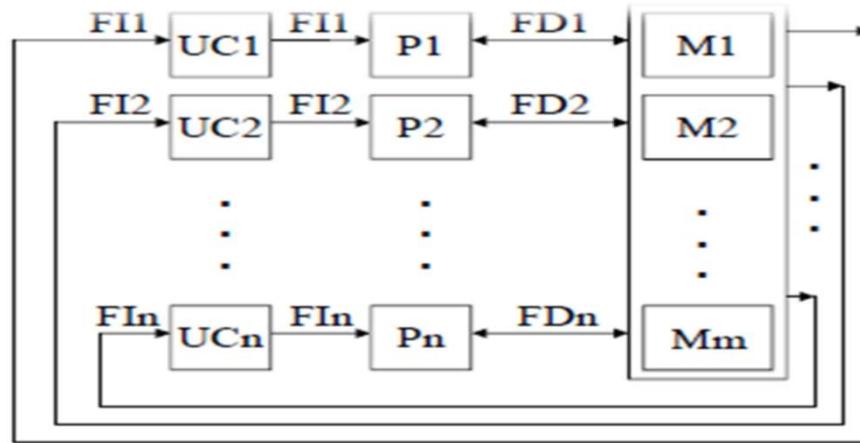
Dans ce mode, la même donnée est traitée par plusieurs processeurs en parallèle. Il existe peu d'implémentations en pratique. Cette catégorie peut être utilisée dans le filtrage numérique et la vérification de redondance dans les systèmes critiques. Cependant, un des exemples d'utilisation de l'architecture MISD sont les ordinateurs de contrôle de vol de la navette spatiale américaine.

Classification de Flynn : MIMD



Multiple Instructions multiple data ou **MIMD** désigne les machines multi-processeurs où chaque processeur exécute son code de manière asynchrone et indépendante. Pour assurer la cohérence des données, il est souvent nécessaire de synchroniser les processeurs entre eux, les techniques de synchronisation dépendent de l'organisation de la mémoire.

Classification de Flynn : MIMD

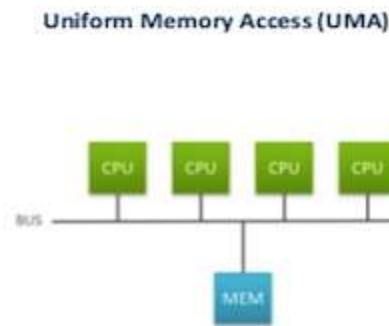


Multiple Instructions multiple data ou **MIMD** désigne les machines multi-processeurs où chaque processeur exécute son code de manière asynchrone et indépendante. Pour assurer la cohérence des données, il est souvent nécessaire de synchroniser les processeurs entre eux, les techniques de synchronisation dépendent de l'organisation de la mémoire.

Parallélisme : Mémoire partagé vs Mémoire Distribuée

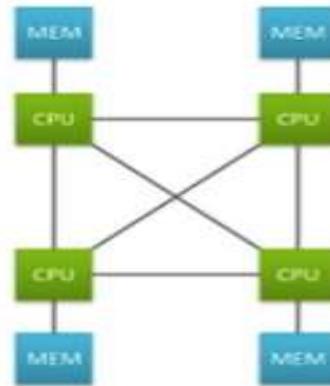
- Bas niveau de la programmation des abstractions qui sont utilisés avec certaines types de programmation parallèle.
- En informatique, le concept d'abstraction identifie et regroupe des caractéristiques et traitements communs applicables à des entités ou concepts variés ; une représentation abstraite commune de tels objets permet d'en simplifier et d'en unifier la manipulation.
- **Mémoire partagée:**

Les systèmes communiquent les uns avec les autres et avec partage de la mémoire principale sur un bus partagé (UMA: Uniform Memory Access)



Parallélisme : Mémoire partagé vs Mémoire Distribuée

Non-Uniform Memory Access (NUMA)

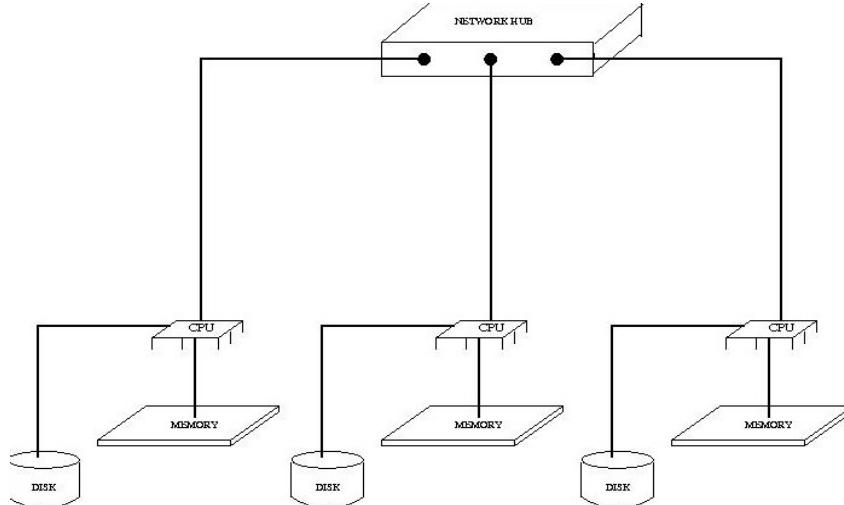


L'illustration montre les différences entre une architecture NUMA et UMA.
La configuration NUMA a une mémoire locale attribuée à chaque processeur,
tandis que dans la configuration UMA, plusieurs processeurs partagent la mémoire via un bus.

Parallélisme : Mémoire partagé vs Mémoire Distribuée

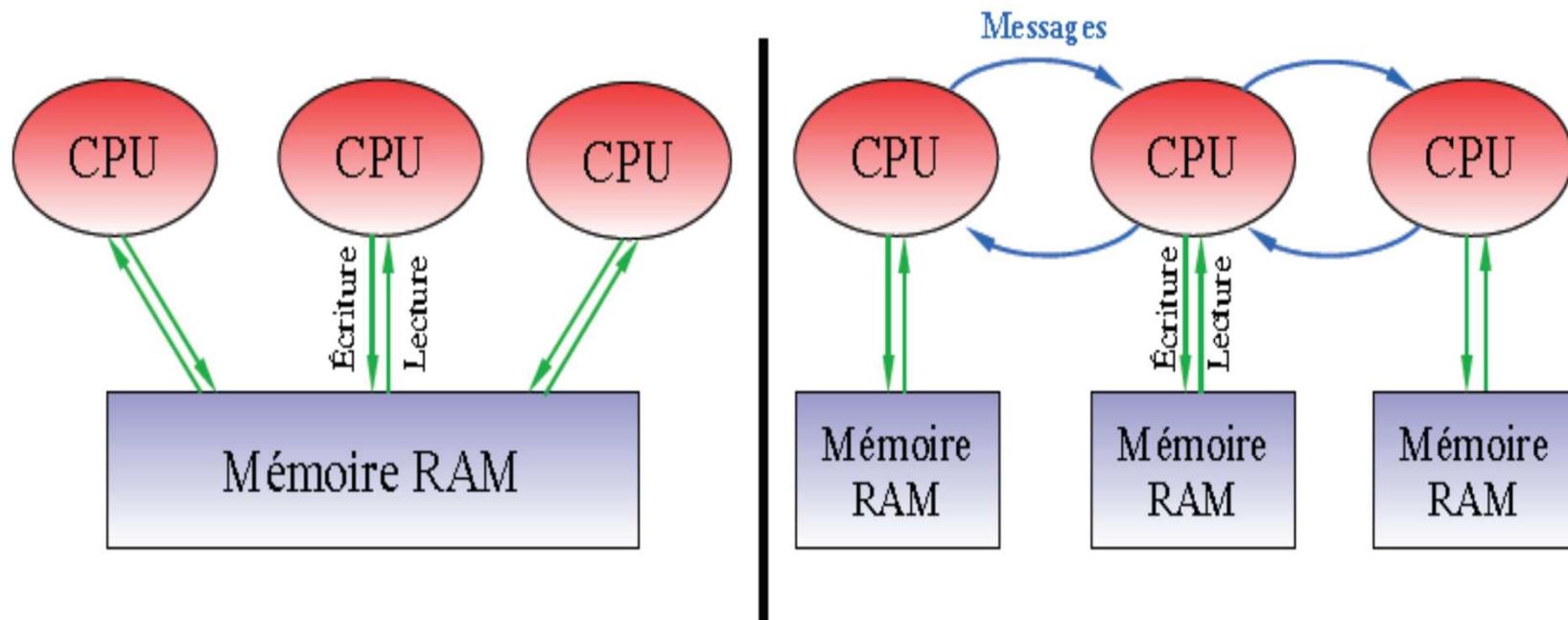
- Mémoire Distribuée

- La mémoire d'un système informatique multiprocesseur est dite **distribuée** lorsque la mémoire est répartie en plusieurs nœuds, chaque portion n'étant accessible qu'à certains processeurs. Un réseau de communication relie les différents nœuds, et l'échange de données doit se faire explicitement par « passage de messages »



Le modèle de passage de messages (message passing en anglais) est un modèle de communication entre ordinateurs ou entre processus à l'intérieur d'un même ordinateur. Il réalise l'envoi de messages simples. Il constitue la couche de base des Middleware Orientés Messages.

Parallélisme : Mémoire partagé vs Mémoire Distribuée



Les types de systèmes d'exploitation

- Les interfaces :

- La ligne de commande, le **mode texte** avec le clavier.

- L'interface graphique (GUI pour Graphical User Interface), le **mode graphique**, avec un pointeur comme une souris.

- Le nombre d'application qui tournent en simultané :

- Les systèmes d'exploitation **mono tâche**.

- Les systèmes d'exploitation **multi tâches** peuvent faire fonctionner plusieurs applications en même temps :

- Les systèmes d'exploitation **préemptifs** gèrent le temps processeur alloué à chaque application. Un commutateur de tâches intervient pour répartir l'allocation des ressources. Des degrés de priorité sont accordés à chaque application. Chaque application peut être interrompus sans interférer avec les autres applications.

- Les systèmes d'exploitation **coopératifs**. Une seule application peut monopoliser toutes les ressources de l'ordinateur, et ne rendre la main aux autre application uniquement quand elle aura terminer...

- Le nombre d'utilisateurs :

- Les systèmes d'exploitation **mono utilisateurs**.

- Les systèmes d'exploitation **multi utilisateurs** peuvent supporter plusieurs sessions en même temps.

- La connectivité réseau :

- Les systèmes d'exploitation **clients**.

- Les systèmes d'exploitation **serveurs**.

- Le nombre de bits des instructions des programmes qui sont développés pour fonctionner avec tel ou tel système :

- Les applications **16 bits**

- Les applications **32 bits**

- Les applications **64 bits** (pour bientôt)

- Le nombre de processeur :

- Les systèmes d'exploitation **mono processeur**

- Les systèmes d'exploitation **multi processeur** (Windows NT et UNIX)

Les types de systèmes d'exploitation

Les systèmes d'exploitation					
	Mono tâches	Multi tâches			
		Mono utilisateur		Multi utilisateurs	
		Coopératif	Préemptif	Coopératif	Préemptif
Le mode texte	MS-DOS				UNIX LINUX
Le mode graphique		Windows 3.x Windows 95 (16 bits)	Windows 95 (32 bits)		Windows NT OS/2 d'IBM NetWare de Novell

Chapitre 2

- Gestion de processus

Définition

1. Définition : Qu'est ce qu'un Processus ?

- Un processus est une **entité dynamique** qui matérialise un programme en cours d'exécution avec ses propres ressources.
 - Ressources d'un processus= **Ressources physiques** (mémoire, processeur, E/S..) + **Ressources logiques** (données, variables,...).
-
- Un programme (texte exécutable) a une **existence statique**.

Types de processus

- **Processus système** : processus lancé par le système (init processus père des tous les processus du système).
- **Processus utilisateur** : processus lancé par l'utilisateur (commande utilisateur).

Création/Terminaison d'un processus

- La création / terminaison de processus se fait par le biais des **appels système**.

- Création : **fork**
- Terminaison : **kill**

Sous Unix

- Un processus reçoit les paramètres suivants lors de sa création:

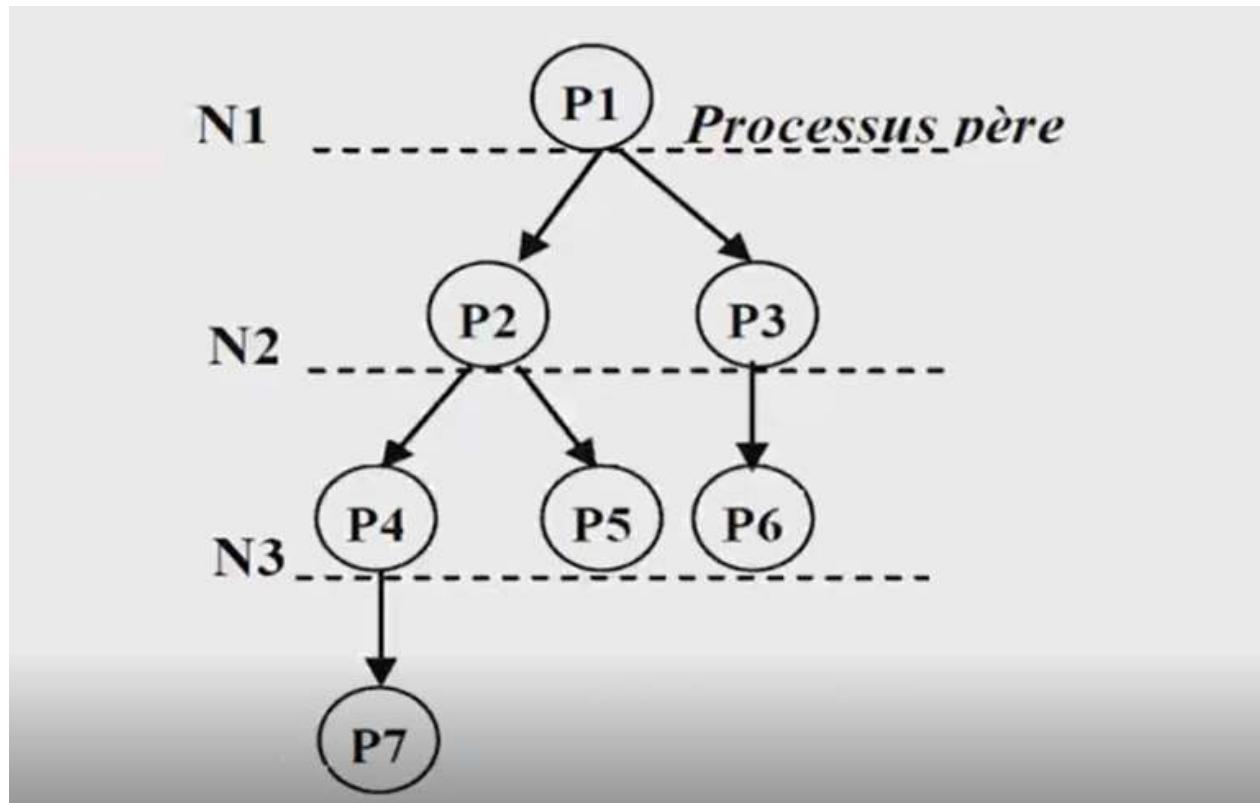
PID : identificateur du processus (numéro unique).

PPID : identificateur du processus père.

UID : identificateur de l'utilisateur qui a lancé le processus.

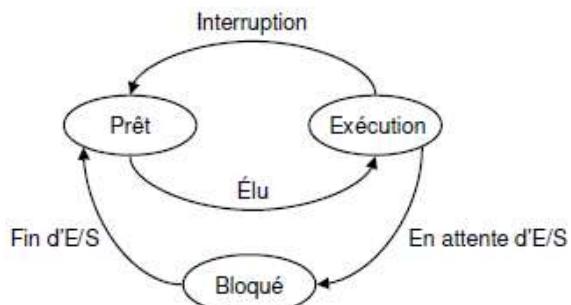
GID : identificateur du groupe de l'utilisateur qui a lancé le processus.

Création/Terminaison d'un processus



Etat d'un Processus

- Lorsqu'un processus s'exécute, il change d'état. Il peut se trouver alors dans l'un des trois états principaux :
 - états :
 - **Élu** : (en cours d'exécution) : si le processus est en cours d'exécution
 - **Bloqué** : attente qu'un événement se produit ou bien ressource pour pouvoir continuer
 - **Prêt** : si le processus dispose de toutes les ressources nécessaires à son exécution à l'exception du processeur.
- Un **seul** processus peut être en exécution sur n'importe quel processeur à tout moment.
- Toutefois, plusieurs processus peuvent être prêts et en attente



Processus

- Point de vue conceptuel: chaque processus possède son processeur virtuel.
- Réalité: le processeur bascule constamment d'un processus à l'autre.
- Ce basculement rapide est appelé multiprogrammation.
- Lorsque le processeur passe d'un processus à un autre, la vitesse de traitement d'un processus donné n'est pas uniforme et probablement non reproductible si le même processus s'exécute une nouvelle fois

Transitions des états d'un processus

- 1) en exécution → en attente** : a lieu quand le processus ne peut plus poursuivre son exécution car il a besoin d'une ressource non disponible.
- 2) en exécution → prêt** : a lieu quand le processus a terminé le temps imparti par le système d'exploitation pour son exécution.

Remarque:

Un processus ne s'exécute pas forcément jusqu'à la fin car le système d'exploitation a d'autres processus à exécuter. Cette transition a également lieu si un processus plus urgent doit prendre la main (par exemple processus du système d'exploitation)

Transitions des états d'un processus

- 3) **prêt** → **en exécution** : signale que le système d'exploitation a sélectionné un processus pour l'exécuter.
- 4) **en exécution** → **Terminé** : indique que le processus a fini son exécution.
- 5) **En attente** → **prêt**: a lieu sitôt que le processus n'a plus de raison d'être bloqué; par exemple les données deviennent disponibles. Le processus passe à l'état prêt.
- 6) **prêt** → **en exécution** a lieu quand l'événement attendu par le processus ne peut se réaliser. Il est donc inutile de faire patienter davantage ce processus, autant le terminer (interblocage).

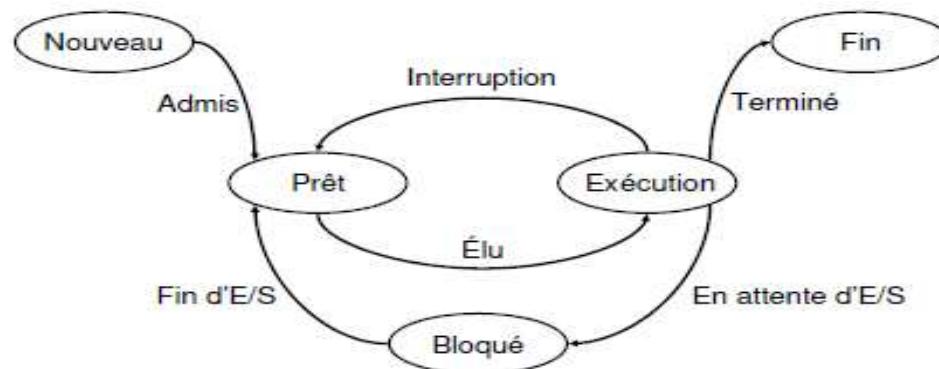
Etat d'un Processus

Initialement, un processus est à l'état prêt. Il passe à l'état exécution lorsque le processeur entame son exécution. Un processus passe de l'état exécution à l'état prêt ou lorsqu'il est suspendu provisoirement pour permettre l'exécution d'un autre processus. Il passe de l'état exécution à l'état bloqué s'il ne peut plus poursuivre son exécution (demande d'une E/S). Il se met alors en attente d'un événement (n de l'E/S). Lorsque l'événement survient, il redevient prêt.

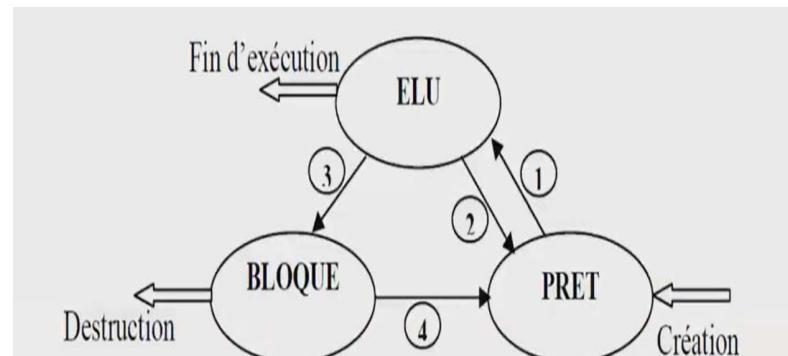
Etat d'un Processus

Le modèle à trois états est complété par deux états supplémentaires :

Nouveau et **Fin**, qui indiquent respectivement la création d'un processus dans le système et son terminaison



Etat d'un Processus (Résume)



Sémantique des Transitions :

- (1) : Allocation du processeur au processus sélectionné.
- (2) : Réquisition du processeur après expiration de la tranche du temps par exemple.
- (3) : Blocage du processus élu dans l'attente d'un événement (E/S ou autres).
- (4) : Réveil du processus bloqué après disponibilité de l'événement bloquant (Fin E/S, etc...)

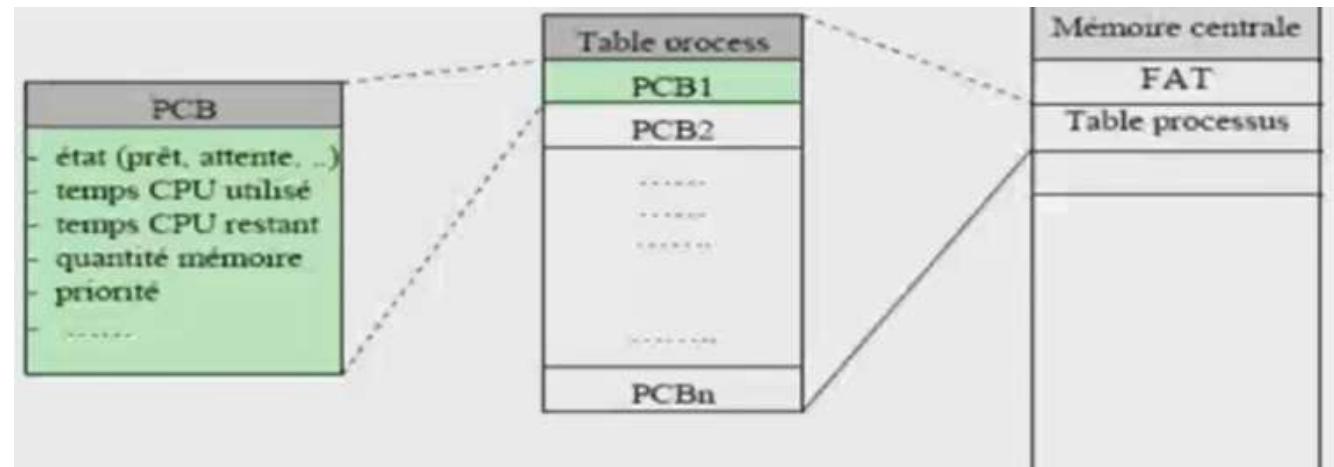
- Lorsqu' un processus est temporairement interrompu, comment peut il retrouver l'état où il se trouvait au moment de l'interruption ?

Introduction

Pour gérer les processus, le système d'exploitation sauvegarde plusieurs informations dans des structures de données. Il existe une table pour contenir les informations concernant tous les processus créés. Il y a une entrée par processus dans la table, appelée le **Bloc de Contrôle de Processus**

- Processus = programme + PCB (control du processus)
- Processus = programme en exécution
- Le processus est une notion qui englobe la notion du programme

On peut avoir plusieurs processus issus du même programme



Bloc de Contrôle de Processus

Chaque entrée de la table **PCB** comporte des informations sur :

- Le pid du processus.
- L'état du processus.
- Compteur d'instructions: indique l'adresse de l'instruction suivante devant être exécutée par ce processus.
- Informations sur la gestion de la mémoire: valeurs des registres base et limite, des tables de pages ou des tables de segments.
- Informations sur l'état des E/S: liste des périphériques E/S allouées à ce processus, une liste des fichiers ouverts, etc..
- Les valeurs contenues dans les registres du processeur.
- .Tout ce qui doit être sauvegardé lorsque l'exécution d'un processus est suspendue
- Le PCB change d'un système d'exploitation à un autre.

Mettre un processus en tâche de fond

- Dans un système informatique, un processus en **tâche de fond** (ou en arrière-plan) est un processus non attaché explicitement à un terminal, ou plus précisément sans interaction avec un utilisateur
- On peut aussi exécuter une commande en tache de fond. Le shell rend alors la main avant la fin de la commande. Pour le faire, on ajoute un & à la fin de la commande;
- Exemple:
- \$gcc –o grosprogramme grosfichier.c &

Objectif de la mise en tâche de fond

- L'objectif de la mise en tâche de fond est d'indiquer ou non à l'interface système, le shell Unix, s'il doit attendre ou pas la fin du processus dont il lancera l'exécution.
- En effet, si un utilisateur lance par exemple l'éditeur de texte « gedit » à l'aide de la commande gedit, celui-ci bloque l'invité de commande et par conséquent, l'utilisateur ne peut plus exécuter de commande car le système attend la fin de la tâche lancée. En lançant le processus en tâche de fond à l'aide du caractère & à la fin de la commande, l'interpréteur de commandes exécute le programme tout en redonnant la main, ce qui permettra d'exécuter une nouvelle ligne de commande.

Background et Foreground

- La commande `jobs` permet d'afficher les processus qui s'exécutent en tâche de fond.
- Certains shells permettent aussi de stopper une commande en premier plan à l'aide de "Control-Z" puis de la basculer en tâche de fond à l'aide de la commande "`bg`" (et son symétrique "`fg`" qui la bascule au premier plan). Cependant, avec certains shells, ce contrôle ne passe que le processus courant en tâche de fonds et des boucles de type "`while/do/done`" ou "`for/do/done`" se trouvent interrompues.
- À noter aussi, la commande "`nohup`" qui permet de lancer une tâche en la détachant du terminal ("`nohup command &`") ce qui permet de fermer le terminal sans que la tâche ne soit tuée.

Background et Foreground

- A partir de l'état suspendu, on peut faire passer un programme:
 - Au premier plan, en tapant fg;
 - En arrière plan, en tapant bg.

Background et Foreground

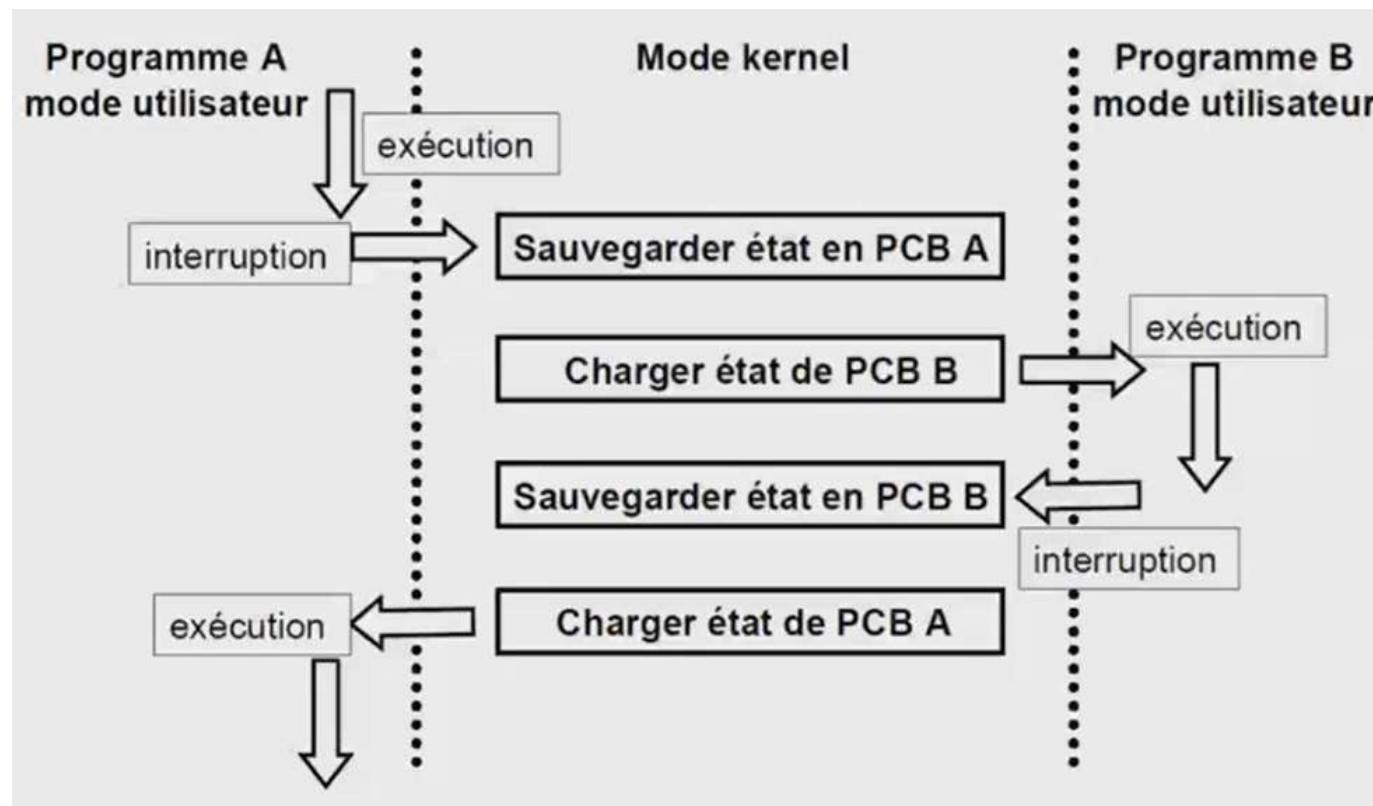
- Le programme tourne au premier plan:
 - ^Z le suspend;
 - ^C l'arrete
- Le programme est suspendu :
 - fg le passe au premier plan;
 - bg le passe en arrière plan
- Le programme tourne en arrière plan
 - Si c'est le seul dans ce cas, fg le passe au premier plan;

VOIR LES PROCESSUS

- A chaque processus est associé un certain nombre d'informations dans le PCB. On dénote:
- Numéro de processus PID (Process IDentifier)
- Numéro d'utilisateur UID (User IDentifier)
- Numéro de groupe GID (Group IDentifier)
- Durée de traitement utilisée(temps cpu) et priorité du processus.
- Référence au répertoire de travail courant (chaque processus a son répertoire courant)
- Table de référence des fichiers ouverts

Changement de contexte de processus

Le basculement d'un processus à un autre



Chapitre 3

Ordonnancement

Ordonnancement de processus

L'ordonnancement est assuré par deux modules :

- **Le Dispatcheur** : Il s'occupe de l'allocation du processeur à un processus sélectionné par l'Ordonnanceur du processeur.
- Commutation de contexte : sauvegarder le contexte du processus qui doit relâcher le processeur et charger le contexte de celui qui aura le prochain cycle processeur
- Commutation du mode d'exécution : basculer du mode Maître (mode d'exécution du dispatcheur) en mode utilisateur (mode d'exécution du processeur utilisateur)
- Branchement : se brancher au bon emplacement dans le processus utilisateur pour le faire démarrer.

Ordonnancement de processus

- **L'ordonnancement** : est la partie du système d'exploitation qui détermine dans quel ordre les processus prêts à s'exécuter (présents dans la file des prêts) seront élus.

Objectifs :

- Assurer le plein usage du CPU (agir en sorte qu'il soit le moins souvent possible inactifs);
- Réduire le temps d'attente des utilisateurs.
- Assurer l'équité entre les utilisateurs.

Ordonnancement de processus

Un algorithme d'ordonnancement permet d'optimiser une des grandeurs temporelles suivantes :

- **Le temps de réponse moyen :**

$$\text{TRM} = \sum_{i=1}^n TR_i / n, \text{ avec } TR_i = \text{date fin} - \text{date arrivée.}$$

- **Le temps d'attente moyen :**

$$\text{TAM} = \sum_{i=1}^n TA_i / n, \text{ avec } TA_i = TR_i - \text{temps d'exécution}$$

Algorithme d'Ordonnancement de processus

Il existe deux types d'algorithme pour l'ordonnancement :

Ordonnanceurs non préemptifs (sans réquisition) :

Allocation du processeur au processus jusqu'à ce qu'il se termine ou il se bloque (en attente d'un événement). Il n'y a pas de réquisition.

- Ordonnancement FCFS (first come first Served) :



- Ordonnanceur sans réquisition
- Ordonnanceur non adapté à un système temps partagé car dans un pareil, chaque utilisateur obtienne le processeur à des intervalles réguliers.

Algorithme d' Ordonnancement de processus

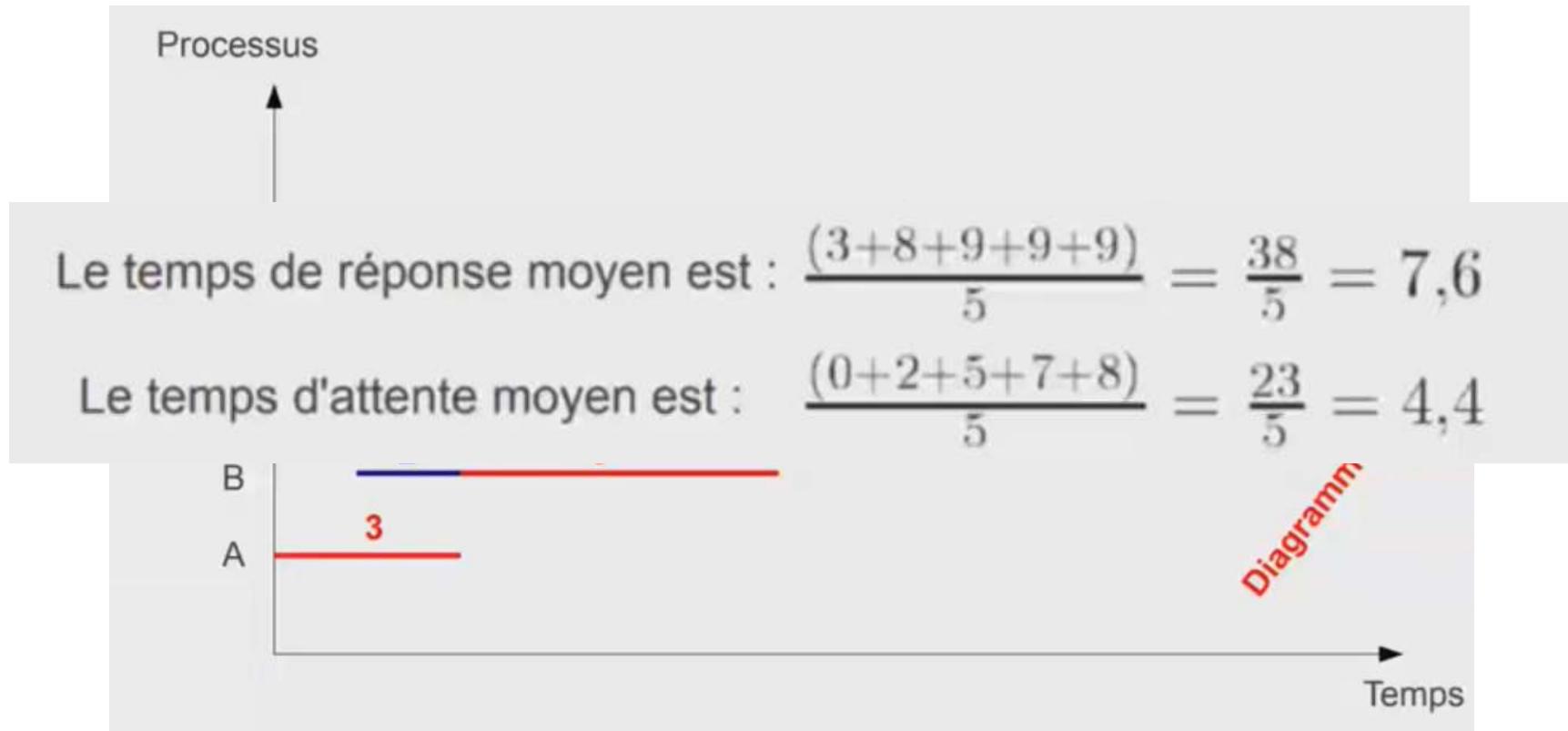
- Ordonnancement SJF (Shortest Job First):

- Allocation du processeur au processus ayant le plus court temps d'exécution.
- Si deux processus ont le même temps d'exécution alors on choisit le premier arrivé.

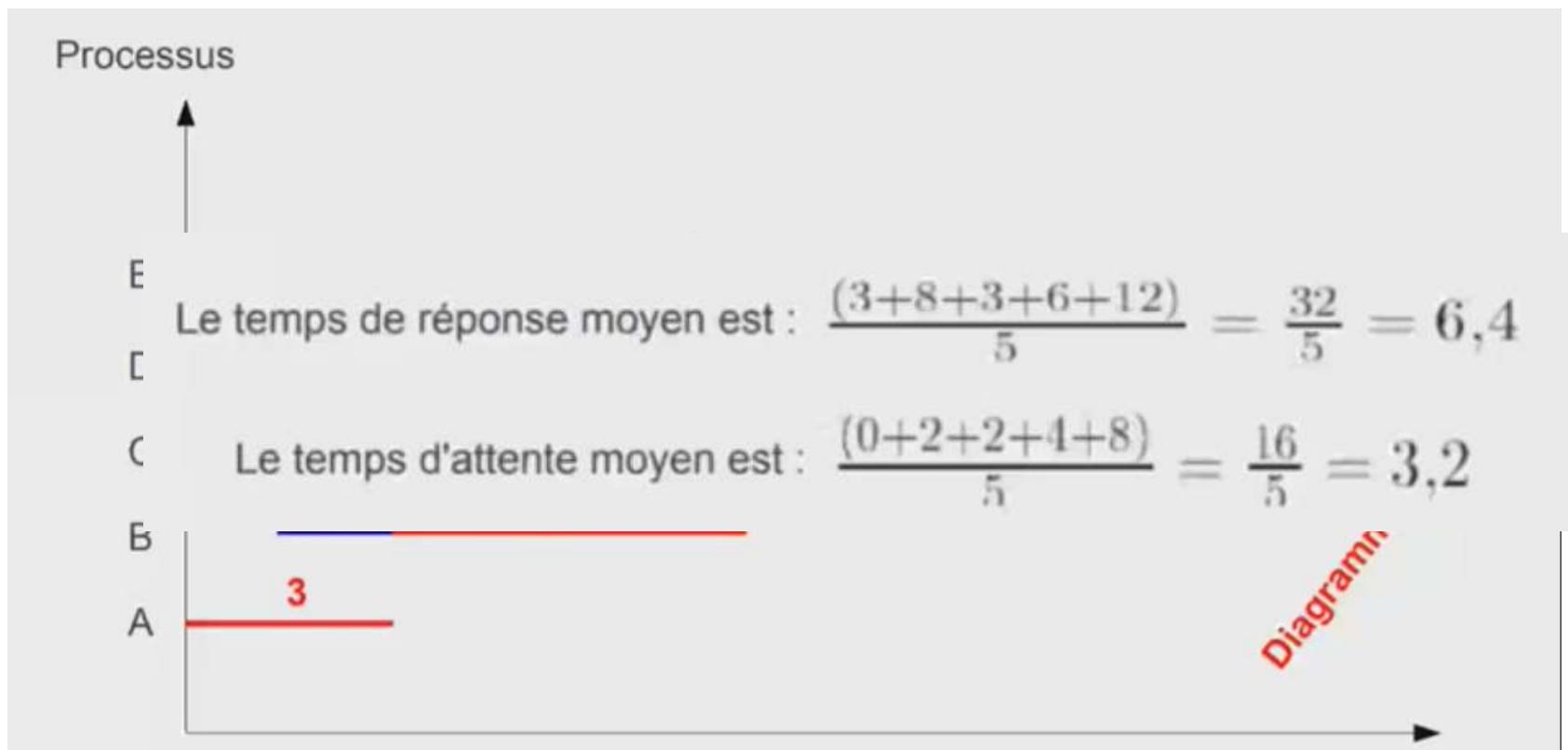
Exemple :

Processus	Temps d'exécution	Temps d'arrivée
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

Algorithme d' Ordonnancement de processus



Algorithme d' Ordonnancement de processus



Ordonnanceurs préemptifs (avec réquisition)

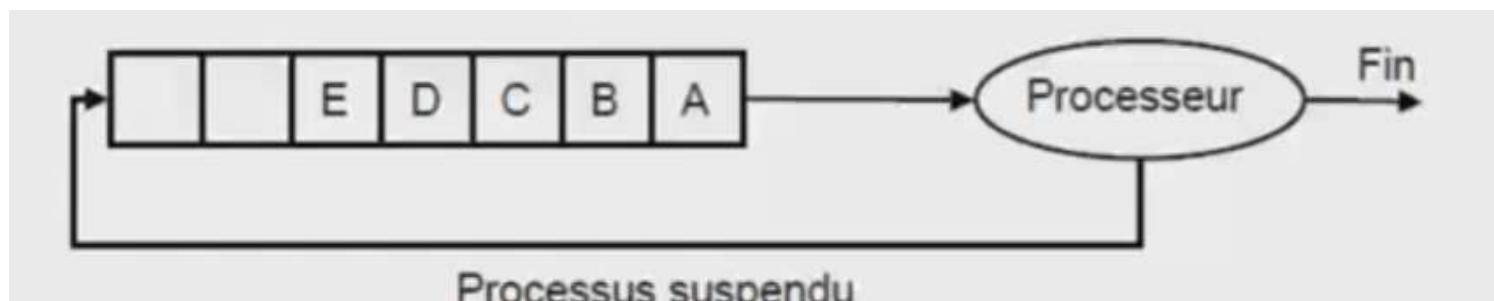
- Ordonnancement du plus court temps restant SRTN(Shortest Remaining Time Next) :

- La version préemptive de l'algorithme SJF.
- L'ordonnanceur compare le temps d'exécution du processus arrivant contre le temps d'exécution restant du processus en exécution.
- Nécessité de connaître les temps d'exécution restant de chaque processus .

Exemple : Soient deux processus A et B prêts tels que A est arrivé en premier suivi de B, 2 unités de temps après. Les temps de UCT nécessaires pour l'exécution des processus A et B sont respectivement 15 et 4 unités de temps.

- Ordonnancement circulaire : Le tourniquet (Round Robin)

- Chacun des processus prêts utilise le processeur pour une tranche de temps précise et fixe appelée **Quantum**.



- Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif.

Exemple 1

Round robin (quantum = 10) :

A	A	A	A	A	A	A	A	A	B	B	B	B	A	A	A	A	A
1				5				10					15				

Processus	Temps de séjour
A	$19-0 = 19$
B	$14-2 = 12$

$$\text{Temps moyen de séjour} = \frac{19+12}{2} = 15.5$$

Processus	Temps d'attente
A	$19-15 = 4$
B	$12-4 = 8$

$$\text{Le temps moyen d'attente} = \frac{4+8}{2} = 6$$

Il y a 3 changements de contexte.

Exemple 2

Round robin (quantum = 3) :

A	A	A	B	B	B	A	A	A	B	A	A	A	A	A	A	A	A	A
1				5					10				15					

Processus	Temps de séjour
A	$19-0 = 19$
B	$10-2 = 8$

$$\text{Temps moyen de séjour} = \frac{19+8}{2} = 13,5$$

Processus	Temps d'attente
A	$19-15 = 4$
B	$8-4 = 4$

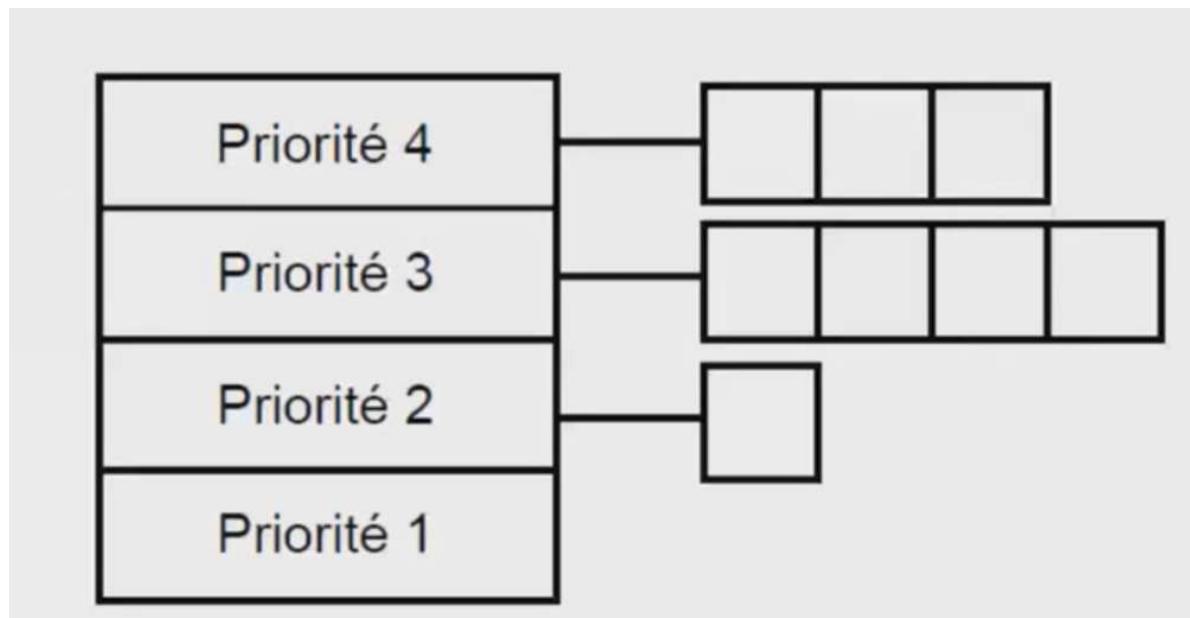
$$\text{Le temps moyen d'attente} = \frac{4+4}{2} = 4$$

Il y a 5 changements de contexte.

- Ordonnancement circulaire à plusieurs niveaux :

- Gestion de n files d'attentes (F_0, \dots, F_{n-1}) pour les processus prêts.
- A chaque file est associée un quantum croissant selon le numéro de la file ($q_0 < q_1 < \dots < q_{n-1}$).
- Tout nouveau processus est mis dans la file d'attente F_0 .
- Tout processus de la file d'attente F_i ne sera servi que si toutes les files F_j tel que $j < i$ sont vides.
- Si un processus de la file F_i a consommé son quantum q_i sans être terminé, il passe dans la file F_{i+1} .
- Cet ordonnanceur favorise les processus court sans savoir leur temps d'exécution .

- Ordonnancement avec priorité :



- Algorithme d'ordonnancement qui peut être avec ou sans réquisition.
- Algorithme d'ordonnancement qui peut être avec des priorités fixes ou variables.

les systèmes d'interruption

Une **interruption** est un signal envoyé de façon **asynchrone** au processeur qui le force à suspendre l'activité en cours au profit d'une autre. La source peut être un autre processeur, un contrôleur d'entrées/sorties ou tout autre dispositif physique externe.

Le programme en cours d'exécution suspend son activité au **premier point interruptible**.

Le processeur exécute un programme prédéfini de traitement de l'interruption.

Les causes d'interruption sont multiples.

Organigramme générale d'interruption :

1. Sauvegarde du contexte dans une pile :

- adresse de la prochaine instruction à exécuter dans le programme interrompu,
- contenu des registres qui seront modifiés par le programme d'interruption,
- contenu du mot d'état rassemblant les indicateurs (Tout cela forme le contexte sauvegardé).

2. Chargement du contexte du programme d'interruption et passage en mode système (superviseur).

3. Exécution du programme d'interruption.

4. Retour au programme interrompu en restaurant le contexte et en repassant en mode utilisateur.

Type d'interruption

- **Interruption interne (déroulement)**: est un événement qui se produit lorsque certaines conditions particulières apparaissent dans l'exécution d'un programme. C'est le cas d'un débordement d'une opération arithmétique, d'une division par zéro, ou d'un accès sur une zone interdite de la mémoire.
- **Interruption logique (appel système)** : permet à un processus utilisateur de faire un appel au système d'exploitation. Il a pour but : changer de mode d'exécution pour passer du mode utilisateur au mode maître, récupérer les paramètres et vérifier la validité de l'appel, de lancer l'exécution de la fonction demandée, de récupérer la (les) valeur(s) de retour et de retourner au programme appelant avec retour au mode utilisateur.
- **Interruptions matérielles** : déclenchées par une unité électronique (lecteur, clavier, canal, contrôleur de périphérique, panne de courant,...) ou par un autre processeur.

Priorité d'interruption :

A chaque interruption, est associée une priorité (système d'interruptions hiérarchisées) qui permet de regrouper les interruptions en classes. Chaque classe est caractérisée par un degré d'urgence d'exécution de son programme d'interruption.
Règle : Une interruption de priorité j est plus prioritaire qu'une interruption de niveau i si $j > i$.

Masquage des interruptions :

Certaines interruptions présentent tellement d'importance qu'il ne doit pas être possible d'interrompre leur traitement. On masquera alors les autres interruptions pour empêcher leur prise en compte. Certaines interruptions sont non masquables : on les prend obligatoirement en compte.

Une interruption masquée n'est pas ignorée : elle est prise en compte dès qu'elle est démasquée.

Au contraire, une interruption peut-être désarmée : elle sera ignorée. Par défaut, les interruptions sont évidemment armées.

Les appels système `wait()`, `waitpid()` et `exit()`

- `wait ()` : Permet à un processus père d'attendre jusqu'à ce qu'un processus fils termine. Il retourne l'identifiant du processus fils et son état de terminaison dans `&status`.
- `waitpid ()` : Permet à un processus père d'attendre jusqu'à ce que le processus fils numéro `pid` termine. Il retourne l'identifiant du processus fils et son état de terminaison dans `&status`.
- `void exit ()` : Permet de finir volontairement l'exécution d'un processus et donne son état de terminaison. Il faut souligner qu'un processus peut se terminer aussi par un arrêt forcé provoqué par un autre processus avec l'envoi d'un **signal** du type `kill ()`.

La famille des appels système `exec()`

Un processus peut changer de code par un appel système à `exec`.

- code et données remplacés
- pointeur d'instruction réinitialisé

L'appel système `exec` permet de remplacer le programme en cours par un autre programme **sans changer** de numéro de processus (PID).

Autrement dit, un programme peut se faire remplacer par un autre code source ou un script shell en faisant appel à `exec`.

La famille des appels système **exec()**

- `execl()` : permet de passer un nombre fixé de paramètres au nouveau programme.
- `execv()` : permet de passer un nombre libre de paramètres au nouveau programme.
- `execle()` : même fonctionnement qu'`execl()` avec en plus, un argument `envp` qui représente un pointeur sur un tableau de pointeurs sur des chaînes de caractères définissant l'environnement.
- `exelp()` : Interface et action identiques à celles d'`execl()`, mais la différence vient du fait que si le nom du fichier n'est pas un nom complet — par rapport à la racine — le système utilisera le chemin de recherche des commandes — les chemins indiqués par la variable `PATH` — pour trouver dans quel répertoire se trouve le programme.
- `exevp()` : Interface et action identiques à celles d'`execv()`, mais la différence vient du fait que si le nom de fichier n'est pas un nom complet, la commande utilise les répertoires spécifiés dans `PATH`.

Qu'est ce qu'un Processus Léger ?

Un **thread** (fil) est un programme en cours d'exécution qui partage son code et ses **données**.

- Chaque thread a une pile d'exécution autonome.
- Un processus est composé de *threads*

Implémentation des Threads

Implantation au niveau du S.E. :

- + le S.E. connaît et ordonne les threads,
- + la répartition de la CPU est bonne,
- les structures du S.E. sont alourdies.

Implantation au niveau utilisateur (java) :

- + une librairie se charge de la gestion des threads (création, destruction, etc.),
- + la commutation entre threads d'un même processus est plus rapide,
- la répartition de la CPU n'est pas équitable,
- la mise « en attente » d'un processus entraîne le blocage de tous ses threads.

Sous Linux ?

Linux ne fait pas de distinction entre les processus et les threads. Un thread est un processus qui partage un certain nombre de ressources avec le processus créateur : l'espace d'adressage, les fichiers ouverts ou autres. Pour la gestion Posix de threads, Linux utilise la bibliothèque *pthread*.

Création de threads

```
int pthread_create (pthread_t *thread ,  
                    pthread_attr_t *attr,  
                    void *nomfonction,  
                    void *arg );
```

Le service `pthread_create()` crée un processus léger qui exécute la fonction `nomfonction` avec l'argument `arg` et les attributs `attr`. Les attributs permettent de spécifier la taille de la pile, la priorité, la politique de planification, etc. Il y a plusieurs formes de modification des attributs.

Sous Linux ?

Suspension de threads

```
int pthread_join(pthread_t *thid, void **valeur_de_retour);
```

`pthread_join()` suspend l'exécution d'un processus léger jusqu'à ce que termine le processus léger avec l'identificateur `thid`. Il retourne l'état de terminaison du processus léger.

Terminaison de threads

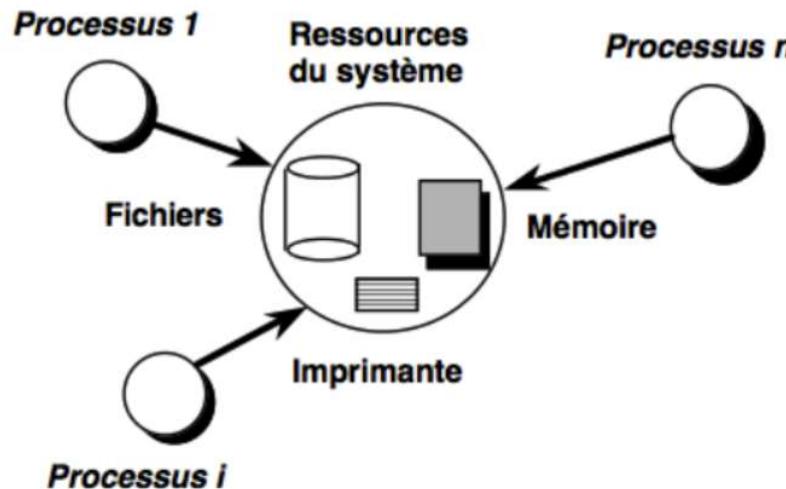
```
void pthread_exit(void *valeur_de_retour);
```

`pthread_exit()` permet à un processus léger de terminer son exécution, en retournant l'état de terminaison.

```
int pthread_attr_setdetachstate(pthread_attr_t *attr,  
                                int detachstate);
```

Synchronisation des processus

- Un système d'exploitation dispose de ressources (imprimantes, disques, mémoire, fichiers, base de données, ...), que les processus peuvent vouloir partager.



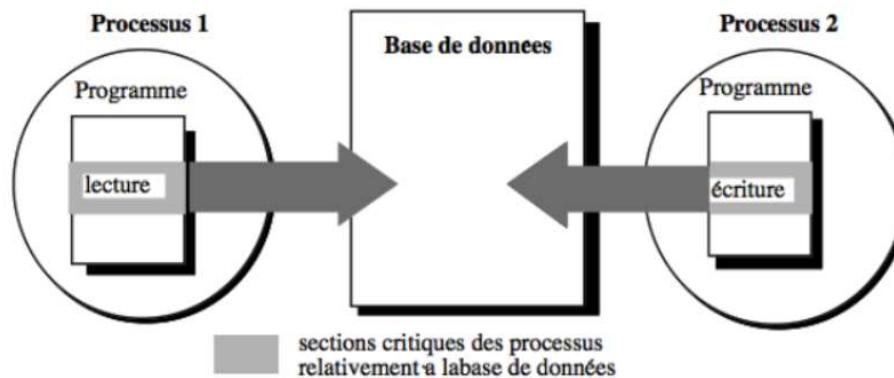
- Ils sont alors en situation de **concurrence** (race) vis-à-vis des ressources. Il faut synchroniser leurs actions sur les ressources partagées.

Section critique

Synchronisation des processus

- La partie de programme dans laquelle se font des accès à une ressource partagée s'appelle une **section critique**.
- Dans la plupart des cas l'accès à cette section critique devra se faire en **exclusion mutuelle**, ce qui implique de rendre indivisible ou atomique la séquence d'instructions composant la section critique.
- Exemple : accès à une base de données.

Plusieurs processus peuvent la lire simultanément, mais quand un processus la met à jour, tous les autres accès doivent être interdits.



Synchronisation des processus

Comment les processus vont-ils synchroniser leurs accès respectifs aux sections critiques ?

Les conditions d'accès à une section critique sont les suivantes :

- deux processus ne peuvent être ensemble en section critique,
- un processus bloqué en dehors de sa section critique ne doit pas empêcher un autre d'entrer dans la sienne,
- si deux processus attendent l'entrée dans leurs sections critiques respectives, le choix doit se faire en un temps fini,
- tous les processus doivent avoir une chance d'entrer en section critique,
- pas d'hypothèses sur les vitesses relatives des processus (indépendance vis à vis du matériel).

Outils de synchronisation

- Matériels :

- Emploi d'une instruction de type **Test and Set** qui teste l'état d'une case mémoire commune aux deux processus et change sa valeur,

- Logiciels :

- **Sémaphores** (outils logiciels mis à la disposition des utilisateurs par le système d'exploitation),
- Algorithmes de synchronisation.

- Gestion de mémoire

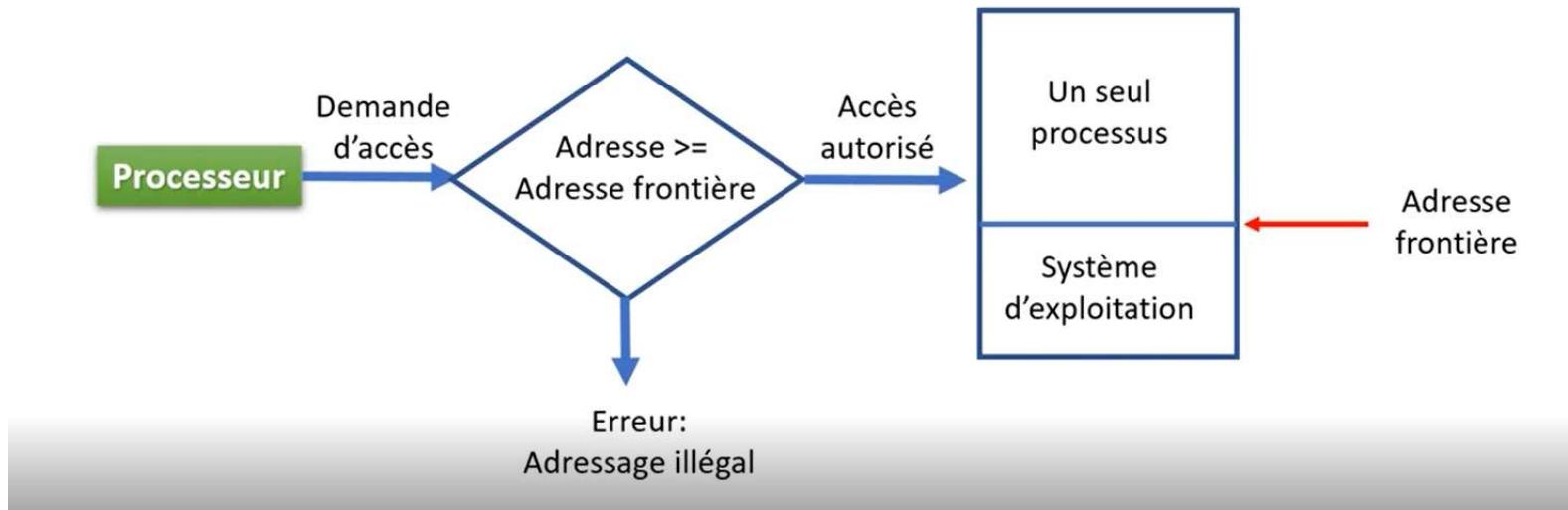
Gestionnaire de mémoire

- La **mémoire** est le lieu où se trouvent les programmes en cours d'exécution (**Processus**).
- La taille continuellement **grandissante** des programmes ==> **insuffisance** de mémoire
- Le **gestionnaire de la mémoire** est la partie du S.E qui se charge de:
 - Connaître les parties libres de la mémoire.
 - Allouer de la mémoire au processus en évitant le gaspillage le plus possible.
 - Récupérer la mémoire libérée.
 - Offrir au processus des services de mémoire virtuelle.

Stratégies d'allocation de la mémoire

Mono-programmation :

- Monopolisation de la mémoire par un seul processus.
- Gestion simple de la mémoire (seulement deux zones).



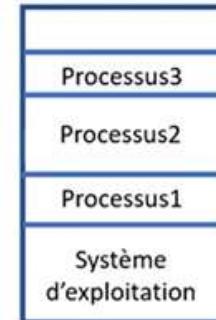
Stratégies d'allocation de la mémoire

Multi-programmation:

- ❑ Présence de plusieurs processus en mémoire.

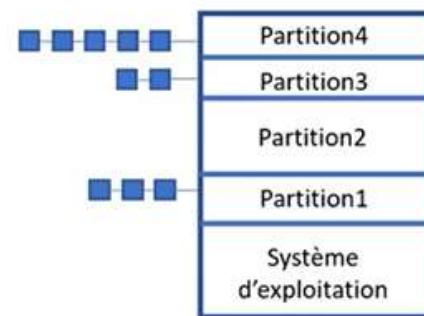
Partitionnement statique :

- ❑ Division de l'espace mémoire en plusieurs partitions de tailles fixes.
- ❑ Nombre et tailles des partitions fixés à la génération du système.
- ❑ Problème de fragmentation interne.



✓ Programmes absolus :

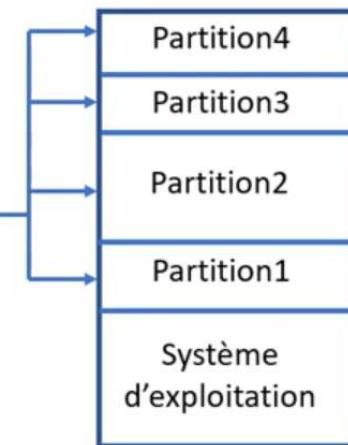
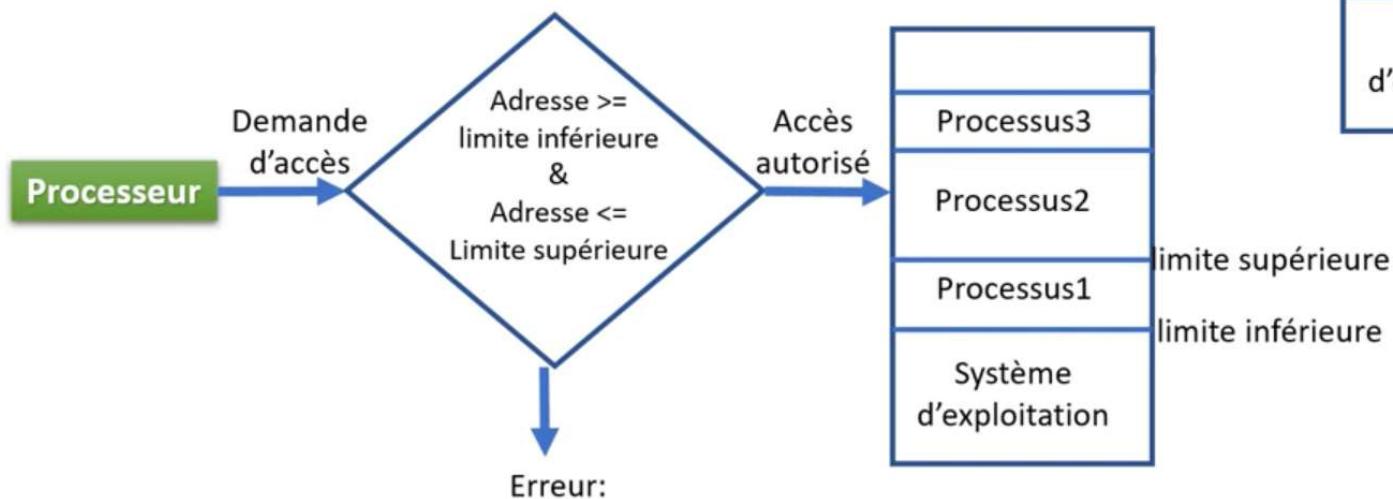
- Liaison statique entre les objets du programme et les adresses mémoire.
- Gestion de plusieurs files d'attente.



Stratégies d'allocation de la mémoire

✓ Programmes relogeables :

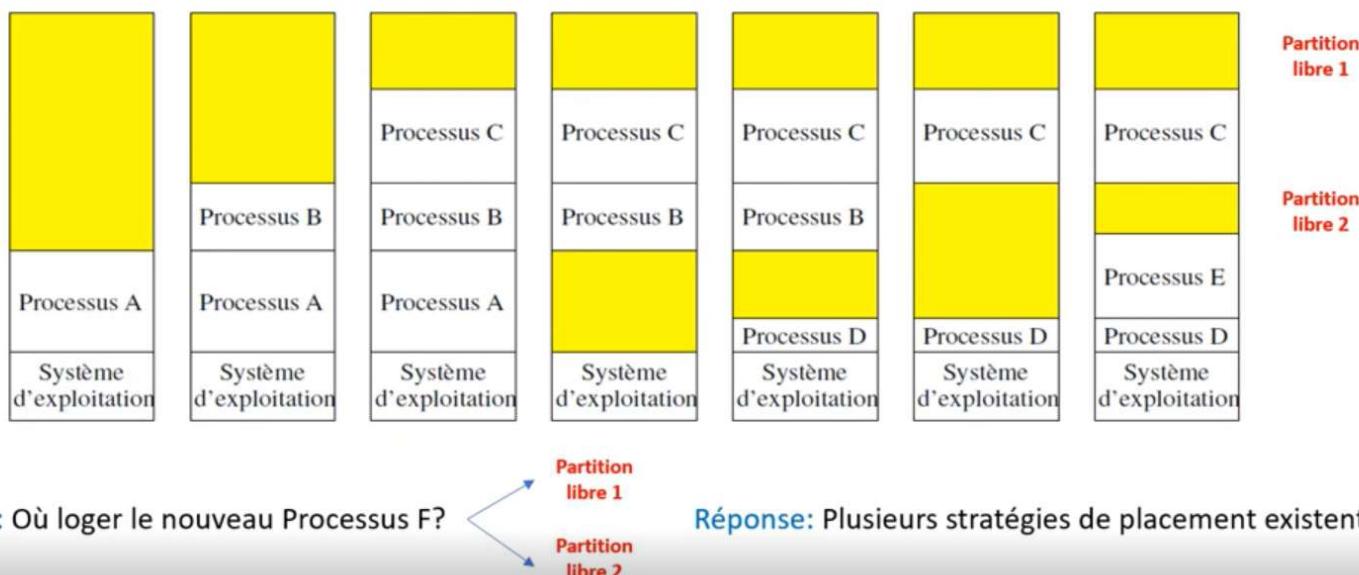
- Liaison dynamique entre les objets du programme et les adresses mémoire.



Stratégies d'allocation de la mémoire

Partitionnement dynamique :

- Partitionnement dynamique selon la demande==> Taille partition=Taille processus.
- Elimination de la fragmentation interne.



Stratégies d'allocation de la mémoire

Stratégies de placement :

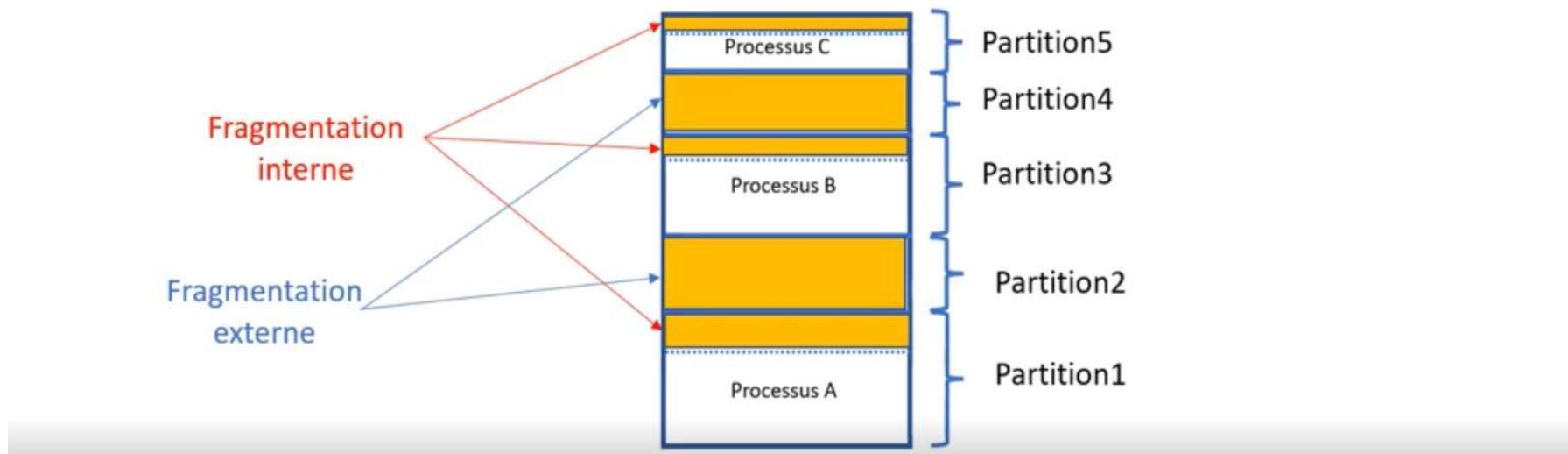
- **First Fit (premier qui convient) :** Placer le processus dans la première partition libre de taille suffisante.
 - ✓ **Avantage:** rapidité.
 - ✓ **Inconvénient:** ne prend pas en considération les autres partitions libres.

- **Best Fit (meilleur qui convient) :** Placer le processus dans la plus petite partition libre de taille suffisante.
 - ✓ **Avantage:** éviter le partitionnement des partitions libres de tailles importantes .
 - ✓ **Inconvénient:** parcours de toute la liste des partitions libres (couteuse en temps).

- **Worst Fit (pire qui convient) :** Placer le processus dans la plus grande partition libre de taille suffisante.
 - ✓ **Avantage:** Eviter la génération de minuscules trous mémoire .
 - ✓ **Inconvénient:** parcours de toute la liste des partitions libres (couteuse en temps).

Types de fragmentation

- ❑ **Fragmentation interne:** Espace de partition inutilisable par le processus et ne pouvant pas être alloué à d'autres processus.
- ❑ **Fragmentation externe:** Espace libre suffisant découpé en plusieurs partitions libres.

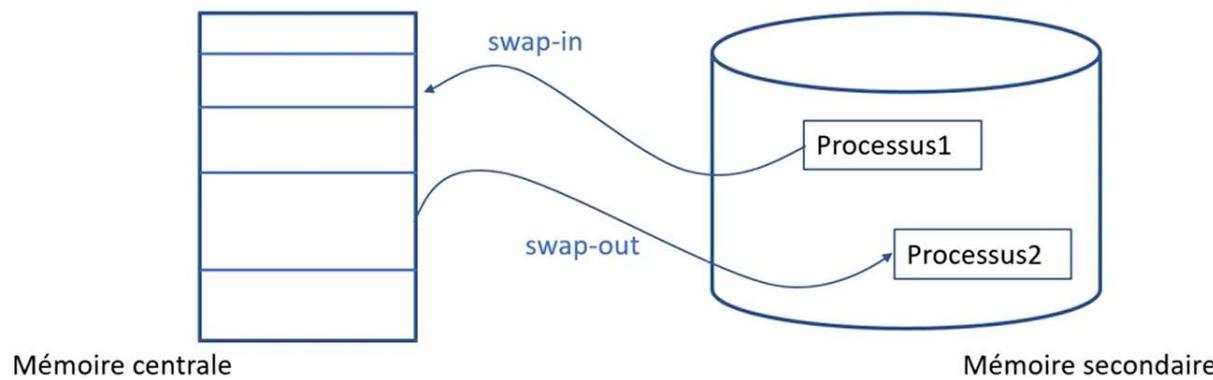


Mémoire virtuelle

Mémoire virtuelle :

Problème: Que faire si la taille du processus est supérieure à la taille de la mémoire ?

Solution: Découper le processus en plusieurs parties et utiliser le swap-in swap-out.

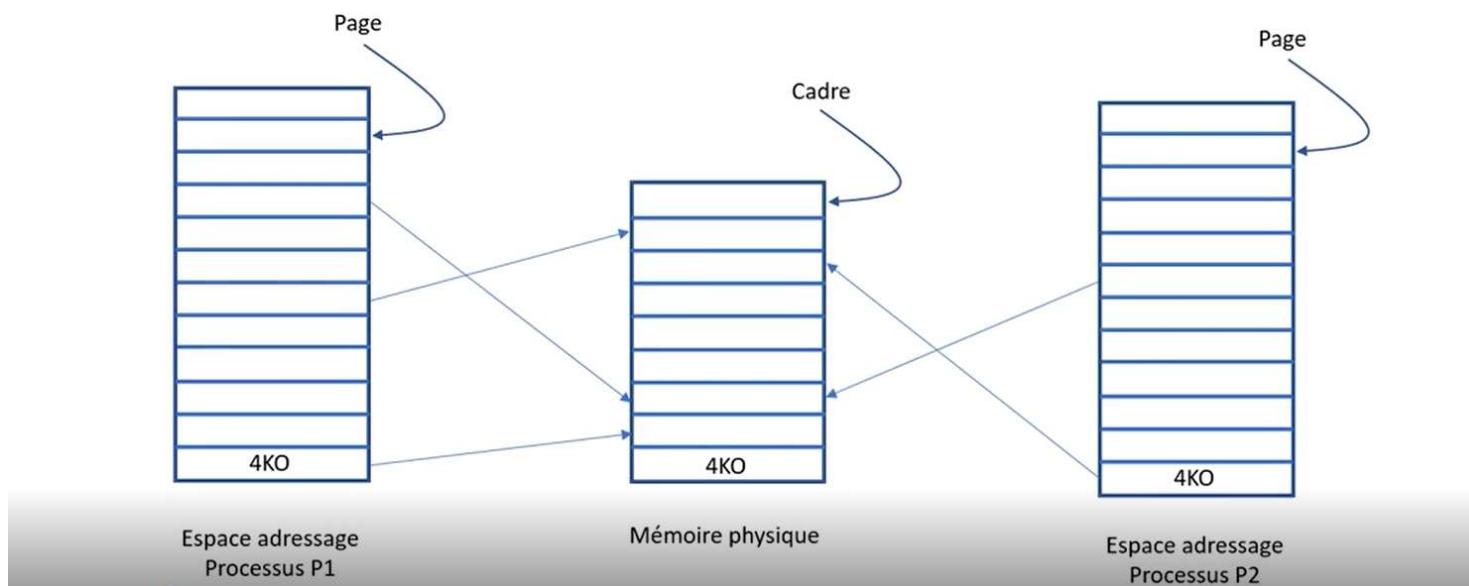


Question: Comment effectuer le découpage?

Réponse: techniques de la mémoire virtuelle

La pagination

- Principes:
 - ✓ Découpage de l'espace mémoire physique ainsi que l'espace d'adressage d'un processus en partitions égales appelées pages (physique- logique).
 - ✓ Traduction d'adresses virtuelles (relative au processus) à des adresses physiques (relative à la mémoire).



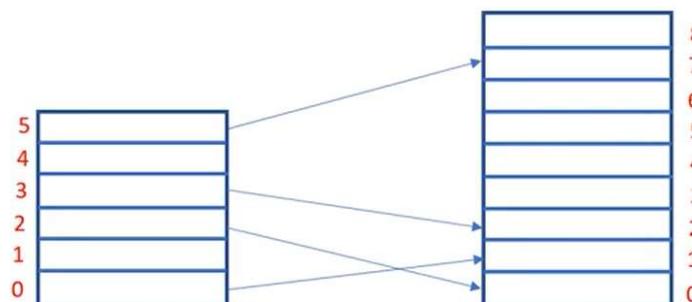
Le cadre qui reçoit la page

La pagination

- Structures utilisées:
 - ✓ Utilisation de **tables de pages** pour sauvegarder les correspondances entre les pages des processus et les cadres de la mémoire ainsi que les informations relatives aux pages.
 - ✓ Chaque processus à sa propre table de pages.

N° Page	N° Cadre	Infos
0	1	
1	X	
2	0	
3	2	
4	X	
5	7	

Table de page du processus P1



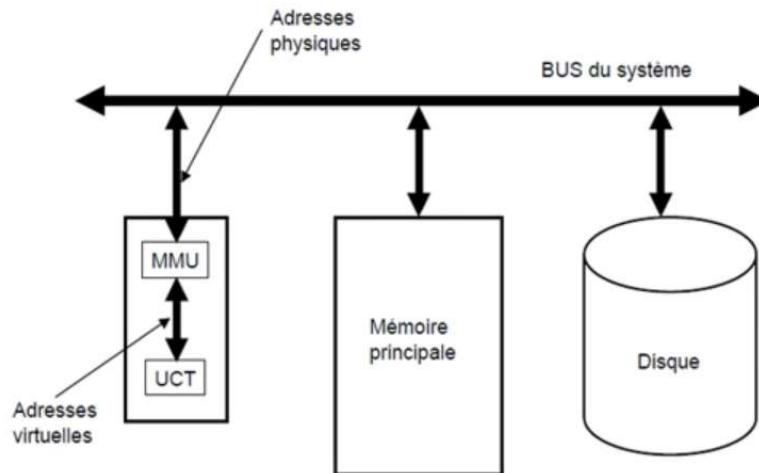
Espace adressage
Processus P1

Mémoire physique

- Présente/absente
- Protection: lecture/ écriture
- Modification
- référencement

La pagination

- Translation d'adresses:
 - ✓ La MMU traduit les adresses virtuelles en adresses physiques.
- Etapes:
 1. Décomposer l'adresse virtuelle en un couple (Numéro de page, déplacement).
 2. Accéder à la table de pages pour récupérer les informations de la page.
 3. Si la page est présente en mémoire, remplacer le numéro de page par le numéro de cadre associé. Autrement, générer un défaut de pages.
 4. Former l'adresse physique sous forme d'un couple (numéro de cadre, déplacement).



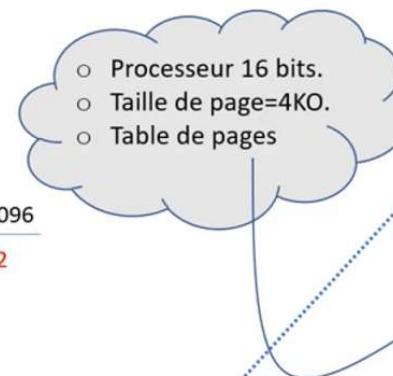
La pagination

Pagination :

- Exemple:

@virtuelle= 8196 @physique= ????

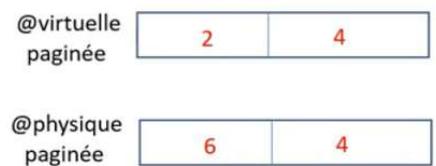
1. @virtuelle paginée (N°page, déplacement):



N° Page	N° Cadre	Infos
0	2
1	1
2	6
3	0
4	4
5	3
6	X
7	X
8	X
9	5
10	X
11	X
12	X
13	X
14	X
15	X

Table de pages du processus P1

2. @physique paginée (N°cadre, déplacement):



3. @physique:

$$\begin{aligned} @physique &= (\text{N°cadre} * \text{Taille de page}) + \text{déplacement} \\ &= (6 * 4096) + 4 \end{aligned}$$

@physique=24580

La pagination

Problème1: Que faire si la page demandée ne se trouve pas en mémoire? (Défaut de page)

Solution:

1. Suspendre l'exécution de processus et le mettre à l'état « bloqué ».
2. La MMU génère une E/S pour charger la page à partir du disque dur.
3. Placer la page chargée dans un cadre vide de la mémoire.
4. Mettre à jour la table de page du processus (modifier le bit de présence ainsi que l'adresse de la page) .
5. Modifier l'état du processus de « bloqué » vers « prêt ».

Sous Problème: Que faire s'il n'existe aucun cadre vide dans la mémoire?

Choisir une page (**page victime**) de la mémoire dont le but de la remplacer par la page demandée:

- ✓ Choix de la page victime: **Localement** au processus / **Globalement** sur l'ensemble des processus?
- ✓ Choix selon quel critère: **Algorithme de remplacement de pages (FIFO, LRU, LFU, MFU...etc)**

La pagination

- Pagination à la demande & Algorithmes de remplacement de pages:
 - ✓ Pagination à la demande: retarder le chargement de la page jusqu'au dernier moment (premier référencement).
 - ✓ Les performances dépendent du nombre de défauts de pages (accès mémoire, disque).

Problème: En cas d'absence de cadre libre, comment choisir la page victime tout en minimisant le nombre de futurs défauts de pages?

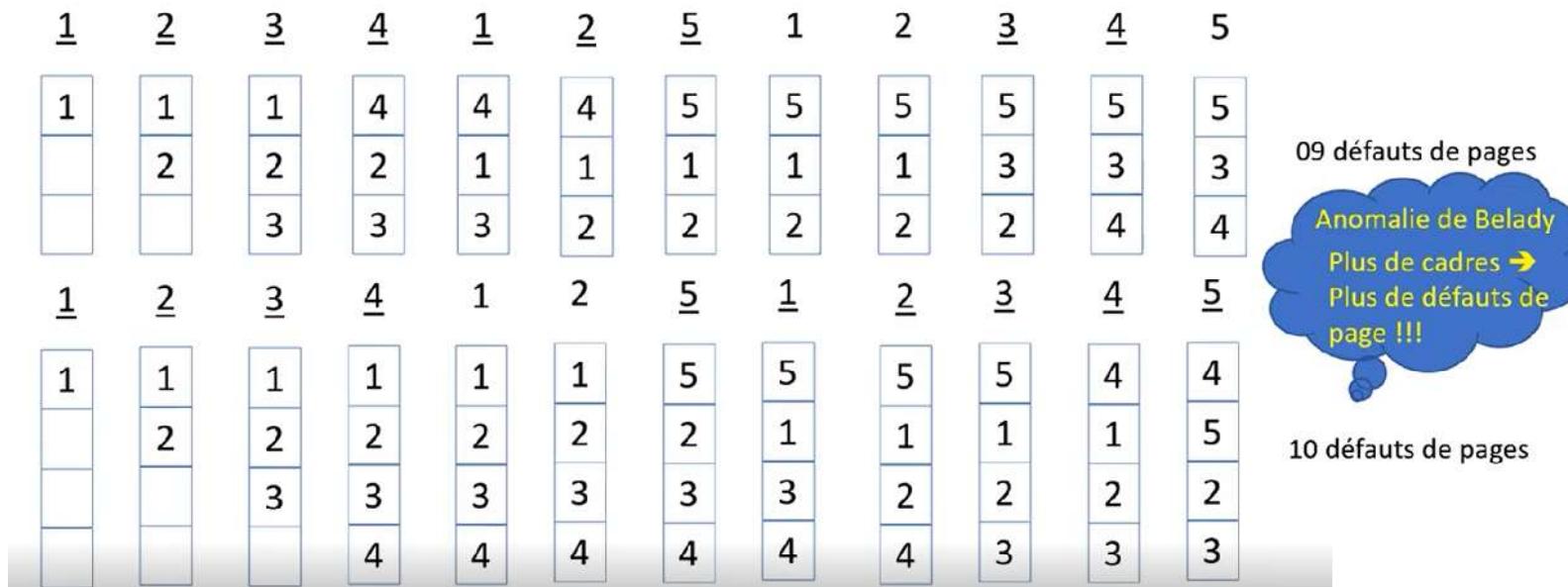
Solution: Plusieurs algorithmes de remplacement de pages existent (FIFO, LRU, LFU,...etc)

La pagination

- Pagination à la demande & Algorithmes de remplacement de pages:

- **Algorithme FIFO (First In First Out):** Retirer la page la plus ancienne (chargée depuis le plus longtemps).

Suite de références ={1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}

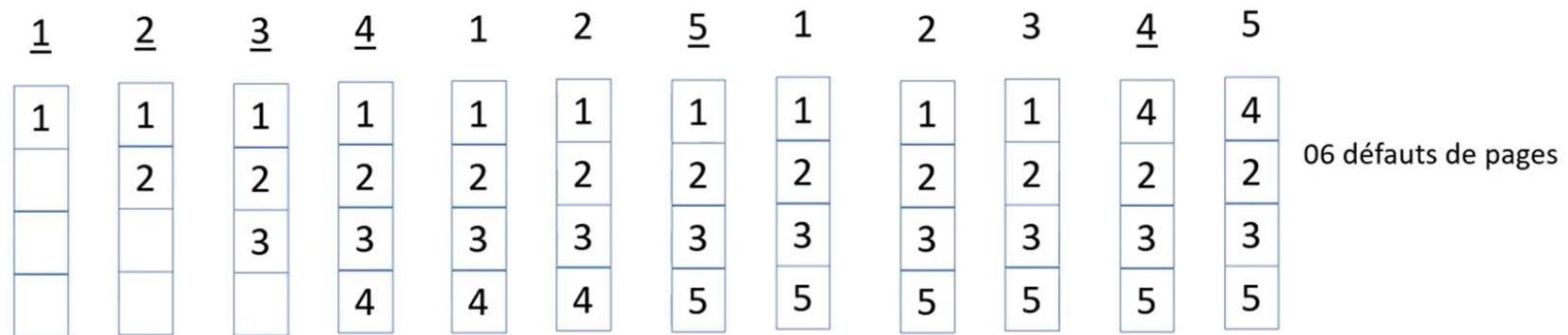


La pagination

- Pagination à la demande & Algorithmes de remplacement de pages:

Algorithme Optimal: Retirer la page qui sera référencée le plus tard possible dans le futur.

Suite de références ={1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}



❖ Impossibilité de mise en œuvre (les références futures!!!???).

❖ utilisé comme base de référence pour les autres stratégies.

La pagination

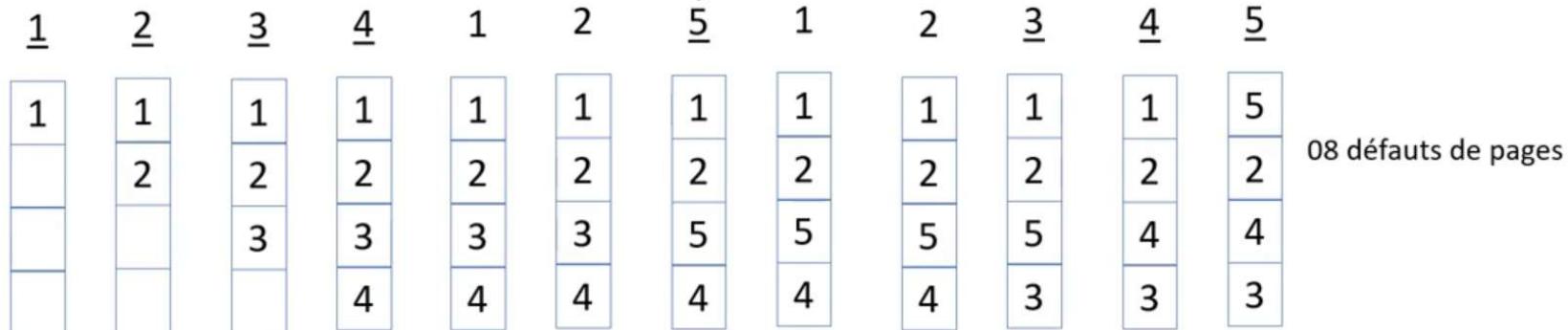
- Pagination à la demande & Algorithmes de remplacement de pages:

· **Algorithm LRU (Least Recently Used):** Retirer la page la moins récemment utilisée.

- ✓ Implémentation avec compteurs: (compteur pour chaque page mis à jour avec le temps de la dernière utilisation) .
- ✓ Implémentation avec pile: (empiler les référencements dans une pile tout en remettant au sommet de la pile les références anciennes nouvellement référencées.

Suite de références ={1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}

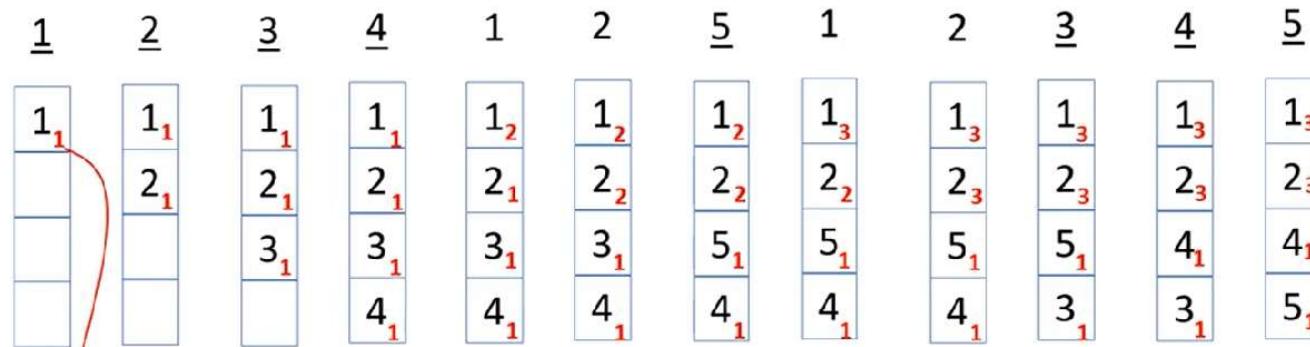
La page 3 est la page non référencée depuis le plus longtemps. C'est la page la plus «oubliée».



La pagination

Algorithme LFU (Least Frequently Used): Retirer la page la moins fréquemment utilisée.

Suite de références ={1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}



08 défauts de page:

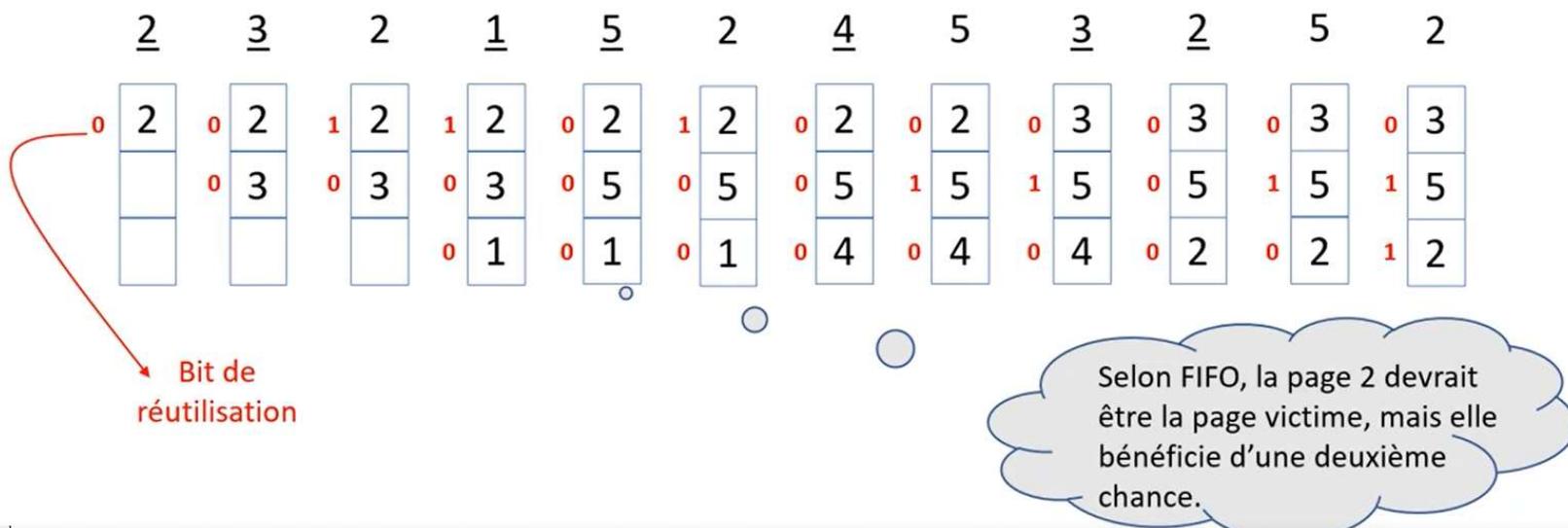
Nombre de
référencements

La pagination

Algorithme de la seconde chance: Donner une deuxième chance à la page la plus ancienne si elle a été réutilisée .

- Amélioration de l'algorithme FIFO.

Suite de références ={2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2}

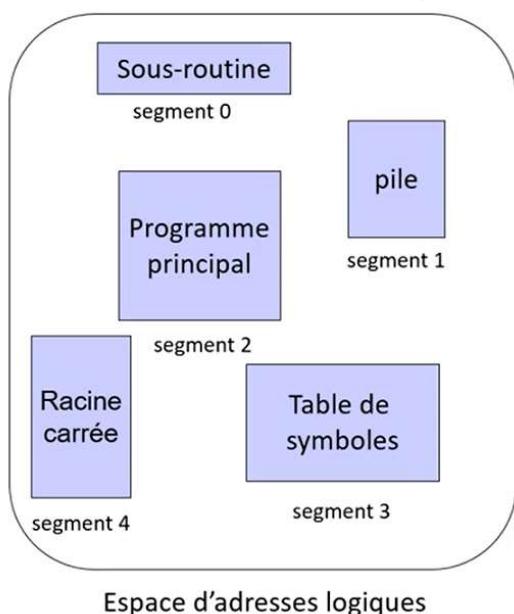


Algorithme de remplacement aléatoire: choisir aléatoirement la page victime.

Segmentation

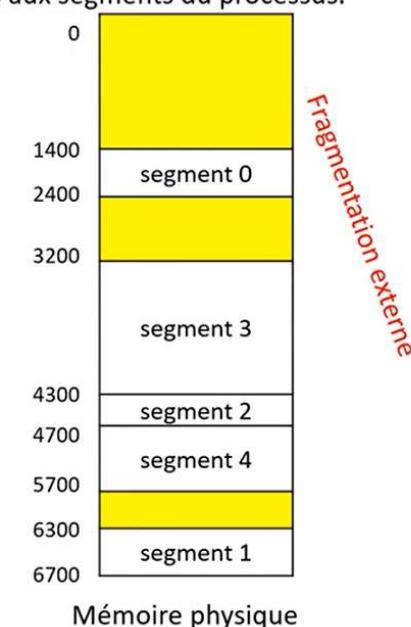
- **Principes:**

- ✓ Découpage de l'espace d'adressage d'un processus en partitions de tailles variables appelées **segments**.
- ✓ Découpage prenant en considération la vue (logique) utilisateur de la mémoire.
- ✓ Utilisation d'une table de segments pour sauvegarder les informations relatives aux segments du processus.



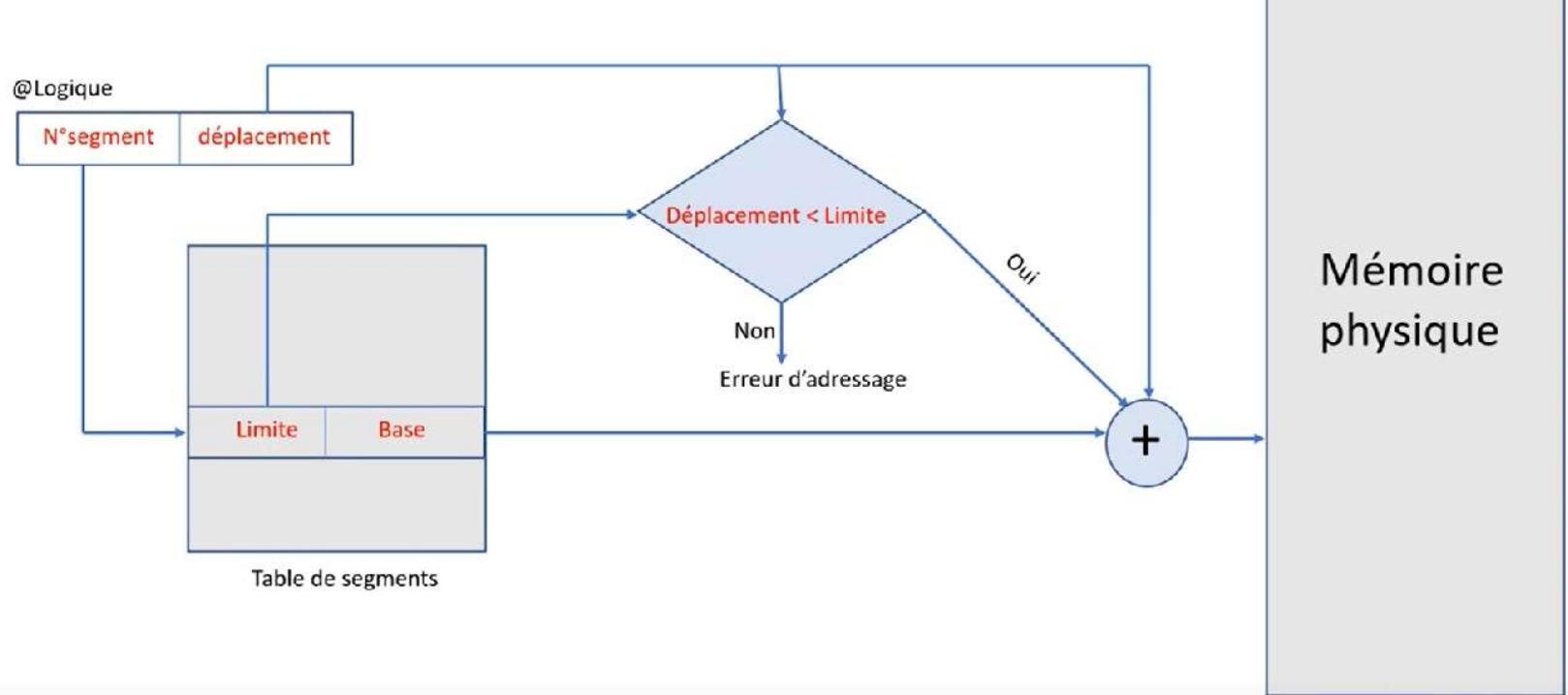
N° segment	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Table de segments



Segmentation

- Translation d'adresses:



Segmentation

Segmentation :

1- calcul d'@ physiques:

$@physique = @Base(segment) + déplacement$

➤ (0:128) Condition vérifiée: $128 < 234$

$$@physique = @Base(segment) + déplacement \\ = 540 + 128 = 668$$

➤ (1:100) Condition vérifiée: $100 < 128$

$$@physique = 1254 + 100 = 1354$$

➤ (2:465) Condition non vérifiée: $465 > 328$

➤ (3:888) Condition vérifiée: $388 < 1024$

$$@physique = 2048 + 888 = 2936$$

➤ (4:100) Condition vérifiée: $100 < 200$

$$@physique = 976 + 100 = 1076$$

➤ (4:344) Condition non vérifiée: $344 > 200$



Segment	Base	Limite
0	540	234
1	1254	128
2	54	328
3	2048	1024
4	976	200

Table de segment du processus P1

2- Validité d'@ virtuelle (4:200):

Condition non vérifiée: déplacement non valide.
Intervalle des déplacements valides= [0-199].

Segmentation

Avantages:

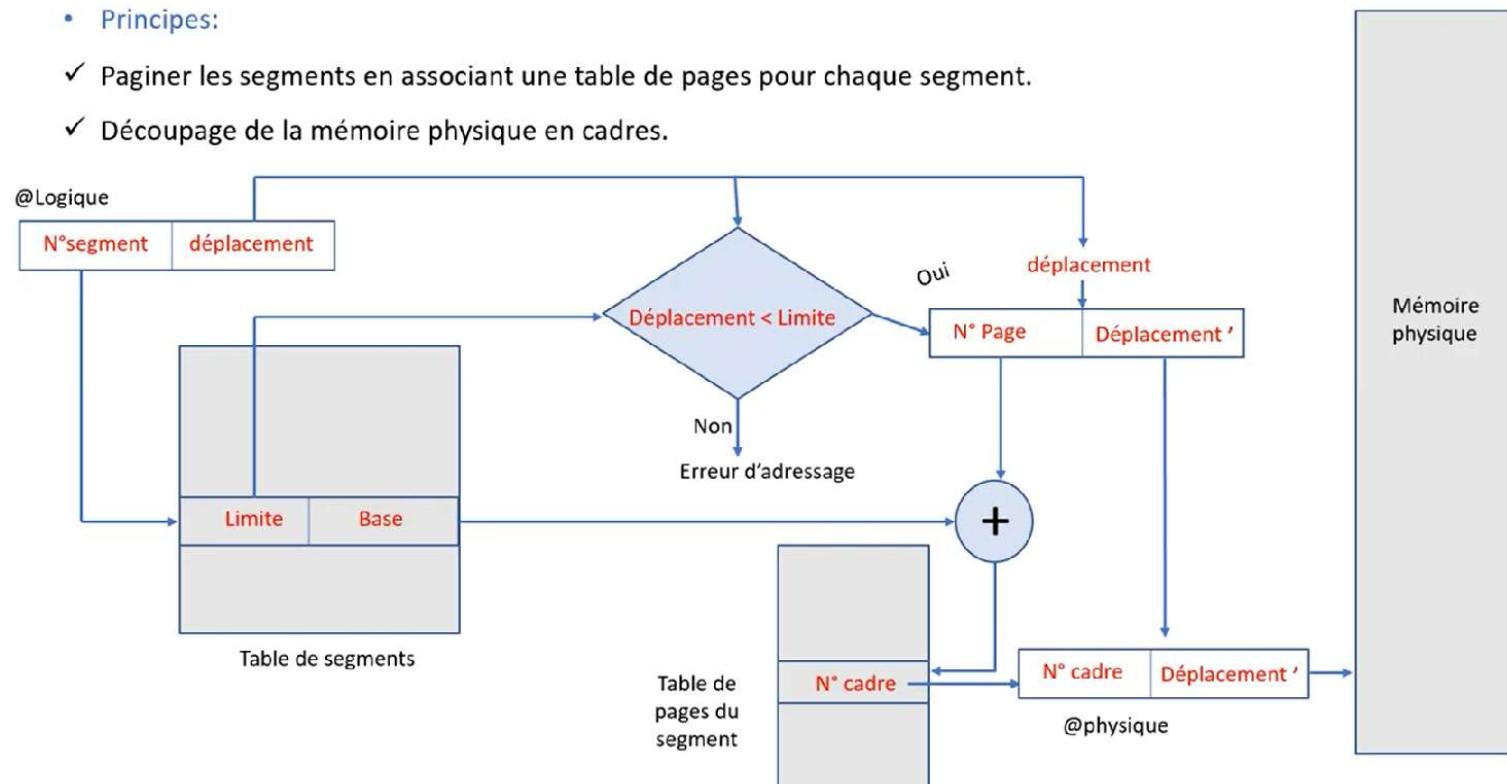
- ✓ Elimination de la fragmentation interne.
- ✓ Possibilité d'agrandir ou de rapetisser les segments.
- ✓ Possibilité de donner à chaque segment ses propres informations de protection (plus facilement que la pagination) .
- ✓ Facilité de partage de code entre processus.

Inconvénients:

- ✓ Fragmentation externe.
- ✓ Nécessité de connaître le modèle de mémoire utilisée .
- ✓ Contiguïté de l'espace alloué pour chaque segment.
- ✓ Les segments peuvent être trop larges pour entrer dans la mémoire physique????

Segmentation paginée :

- Principes:
- ✓ Paginer les segments en associant une table de pages pour chaque segment.
- ✓ Découpage de la mémoire physique en cadres.



Segmentation paginée :

Segmentation paginée : Exercice

- Technique: Segmentation paginée.
- Taille de page = 4KO.
- Taille mémoire physique = 64 KO
- Taille S0= 16 KO. Taille S1= 8 KO. Taille S2= 4 KO.

@virtuelle= 8212 @physique= ???

$$\begin{array}{r} 8212 \\ \hline 4096 & \\ 20 & 2 \end{array} \quad 8212 = 2 * 4096 + 20 \quad \xrightarrow{\text{ }} \text{Page N}^{\circ}2, \text{déplacement } 20$$

Segment	N° page	N° cadre
S0	1	2
	2	0
S1	1	9
S2	0	12

1- Numéro de segment= 0 (Page N°2 ∈ Segment S0 (4 pages))

2- Numéro de page= 2

3- Déplacement= 20

4- Numéro de cadre= 0

5- Déplacement (cadre) = 20

6- @physique:

$$@\text{physique (décimale)} = \text{N}^{\circ} \text{ cadre} * \text{taille de cadre} + \text{déplacement} = 0 * 4096 + 20 = 20$$

• Taille mémoire physique = 64 KO = 2^{16} octets $\xrightarrow{\text{ }} @\text{physique sur 16 bits}$

• Taille de page = 4KO = 2^{12} octets. $\xrightarrow{\text{ }} 12$ bits pour le déplacement

$$@\text{physique (binaire)} = \boxed{0000 | 00\ 00\ 0001\ 0100}$$

Gestion de mémoire par subdivision (frères siamois):

- Manipulation de partitions dont la taille est une puissance de deux (1, 2, 4, 8 octets,, la taille maximale de la mémoire).
- **Allocation:**
 - Sélection de partition **libre** dont la taille correspond à la **plus petite puissance de deux supérieure ou égale** à la taille demandée (si elle existe).
 - Autrement,
 - ❖ Sélection de partition libre ayant une taille de puissance immédiatement supérieure.
 - ❖ Division de la partition sélectionnée en deux moitiés (frères siamois)
 - ❖ Allouer une moitié et l'autre moitié forme une partition libre.
- **Désallocation:**
 - Libérer la partition.
 - Si la partition voisine est libre, les deux partitions sont fusionnés pour redonner une unique partition de taille supérieure.
 - processus se poursuit récursivement jusqu'à ce que plus aucune fusion ne soit possible

Gestion de mémoire par subdivision (frères siamois):

Etat initial	1024				
Allouer A=70	A	128	256	512	
Allouer B=35	A	B	64	256	512
Allouer C=200	A	B	64	C	512
Libérer A	128	B	64	C	512
Allouer D=60	128	B	D	C	512
Libérer B	128	64	D	C	512
Libérer D	256		C		512
Libérer C	1024				

Pagination :

Problème2: Que faire si les tables de pages sont de tailles très importantes ?

- Exemple:
 - Espace d'adressage sur 32 bits \Rightarrow Taille maximale d'un processus= 2^{32} = 4 Go
 - Taille de page= 4Ko \Rightarrow Nombre de page= Taille maximale d'un processus/ Taille de page = $2^{32} / 2^{12} = 2^{20}$ pages
 - Nombre de bits pour chaque entrées dans la table de pages= 26 bits \Rightarrow
 - Taille de la table= nombre de bits pour chaque entrées X nombre de pages= $26 \times 2^{20} = 3,25$ Mo

Machine sur 64 bits \Rightarrow Nombre de page= 2^{52} pages!!

Solution: Pagination multi-niveaux

- Gestion des Fichiers

Introduction

- *La mémoire centrale est une mémoire volatile.*
- *Il faut stocker les données devant être conservées au delà de l'arrêt de la machine sur un support de masse permanent.*
- *L'unité de conservation sur le support de masse est le **fichier**.*
- *Le système d'exploitation gère les fichiers via le Système de gestion de fichiers (SGF).*
- *Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers.*

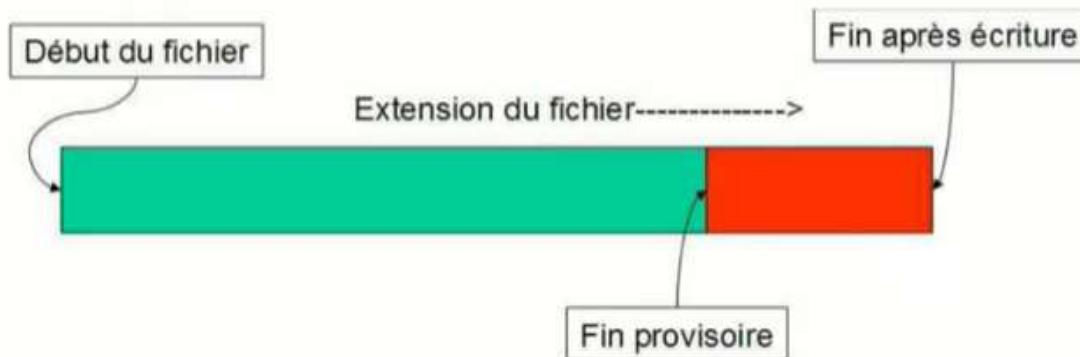
Fichier

- Une unité de stockage logique.*
- Ensemble d'informations en relation entre elles, qui est enregistré sur la mémoire auxiliaire.*
- Le SE établit une correspondance entre les fichiers et les dispositifs physiques.*
- Deux visions d'un système de fichiers:*
 - **Point de vue de l'utilisateur:** nommage des fichiers, protection et droit d'accès, opération autorisées, etc.
 - **Point de vue de l'implantation:** organisation physique d'un fichier sur un disque, gestion des blocs et manipulation des blocs physiques attribués à un fichier, gestion de l'espace libre du disque.

Modes d'accès du SGF

Accès séquentiel:

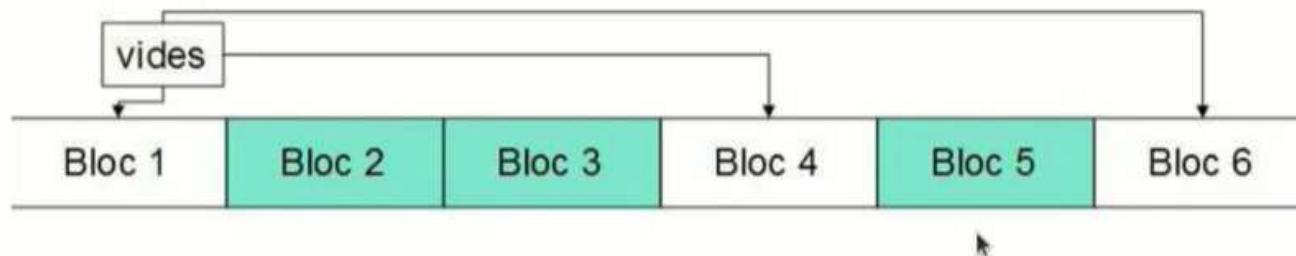
- ❑ L'information dans le fichier est traitée en ordre, un enregistrement après l'autre.
- ❑ Éditeurs et compilateurs utilisent cette méthode. Pratique quand le support de stockage était une bande magnétique.



Modes d'accès du SGF(Suite)

Accès direct (aussi dit accès aléatoire):

- Constitué d'enregistrements logiques de longueur fixe
- Permet l'accès immédiat à un enregistrement
- La taille du fichier est connue et peut être réservée à sa création
(taille d'un bloc * nombre de blocs)

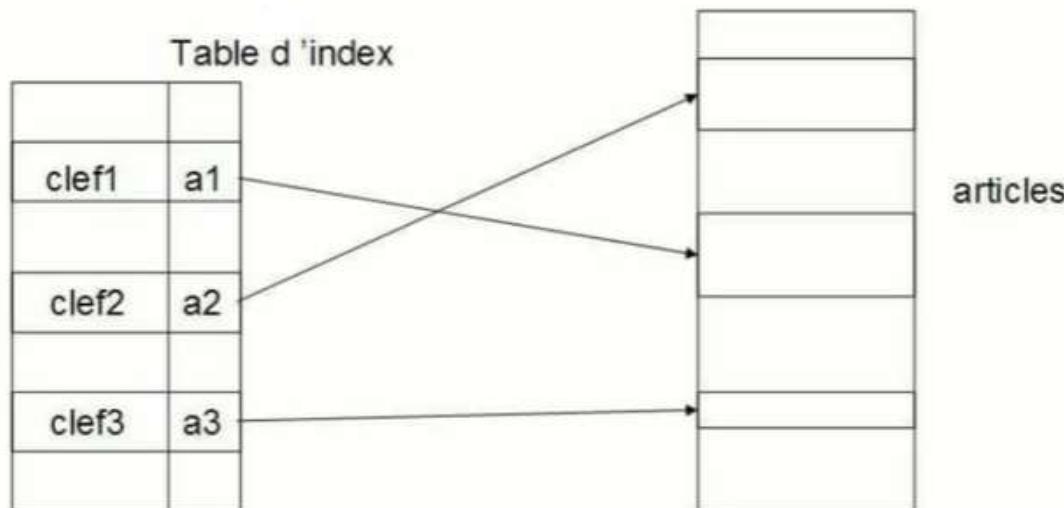


- On peut accéder directement au bloc 5 sans que les précédents ont été remplis au préalable

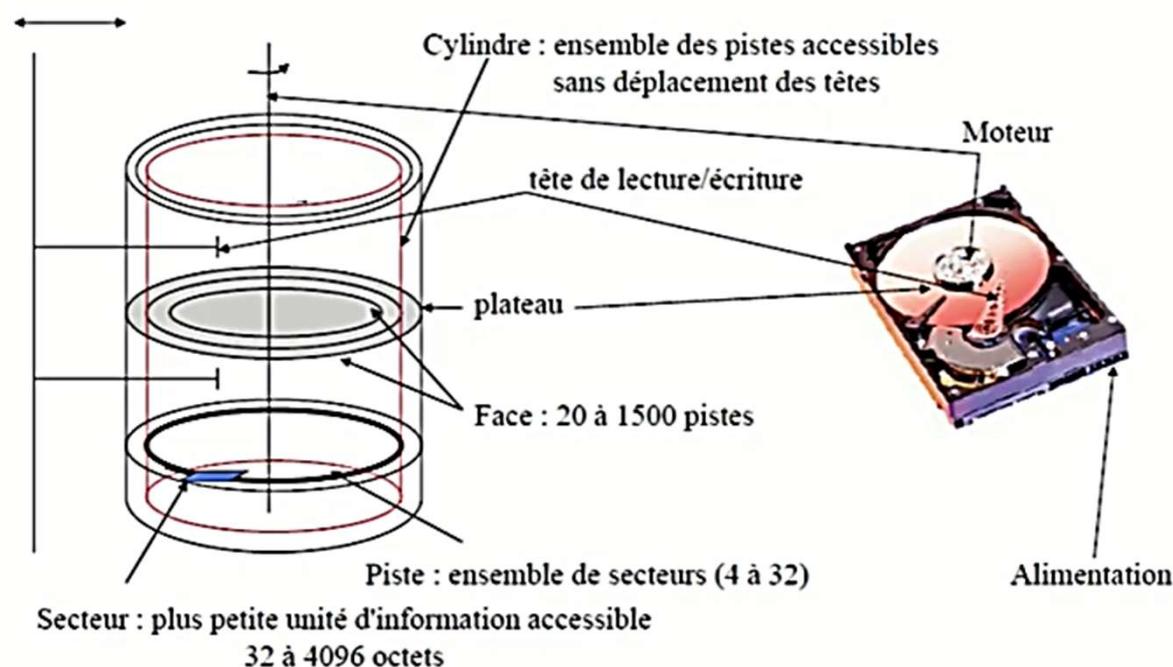
Modes d'accès du SGF(Suite)

Accès indexé :

- Nécessite d'avoir un ensemble de clés ordonnées
- Relation entre clés et articles établie au moyen d'une table d'index.



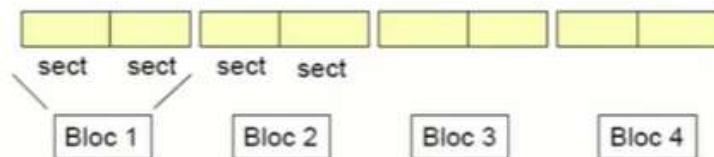
Structure du disque dur



Gestion de l'espace disque

- ❑ *Le fichier physique correspond à l'implémentation sur le support de masse de l'unité de conservation fichier.*
- ❑ *Un fichier physique est constitué d'un ensemble de blocs physique. Il existe plusieurs méthodes d'allocation des blocs physiques :*
 1. *allocation contiguë (séquentielle simple)*
 2. *allocation par blocs chainés*
 3. *allocation indexée*

*Les opérations de lecture et d'écriture du SGF se font bloc par bloc
1 bloc = 2 secteurs de 512 octets soit 1KO*

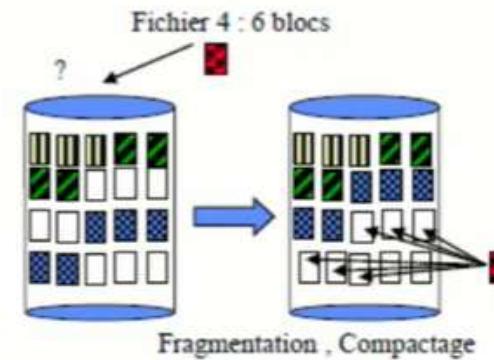
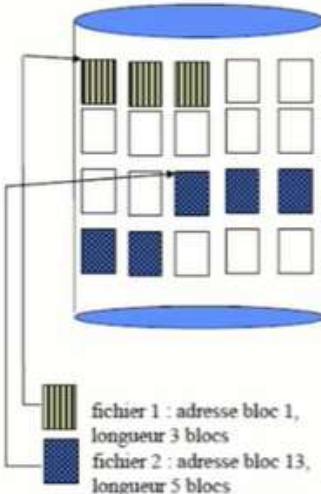


Allocation contiguë

- Un fichier occupe un ensemble de blocs contigus sur le disque
- Difficultés:

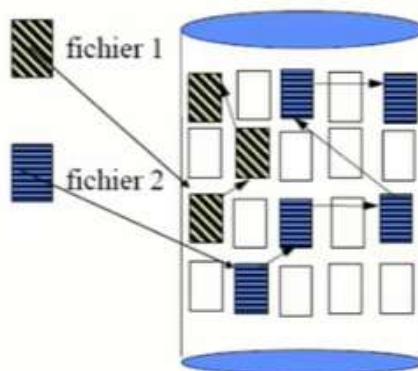
Fragmentation: trouver des espaces suffisants à une nouvelle allocation.

Compactage: regrouper les espaces libres dispersés en un seul espace libre exploitable.



Allocation par bloc chainée

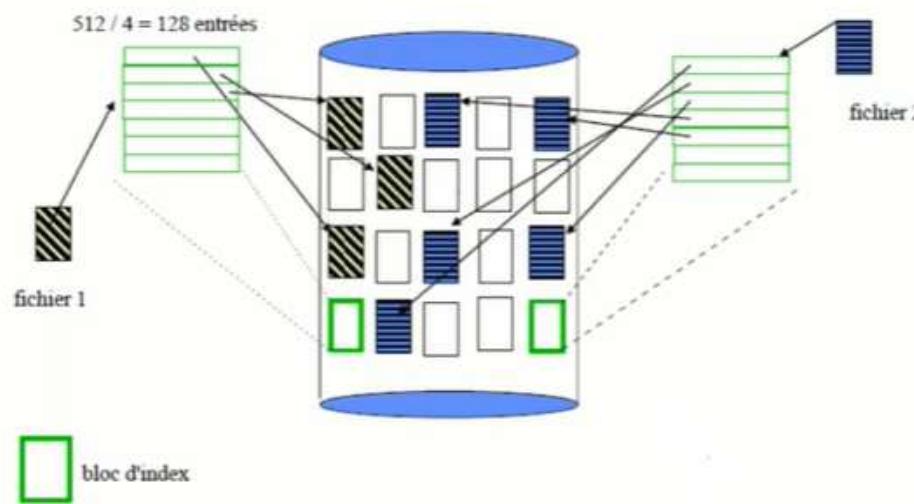
- ❑ Un fichier est constitué comme une liste chaînée de blocs physiques, qui peuvent être dispersés n'importe où sur le support de masse.



- ❑ Extension simple du fichier : allouer un nouveau bloc et le chainer au dernier
- ❑ Pas de fragmentation
- ❑ Difficultés :
 - ✓ Le seul mode d'accès utilisable est le mode d'accès séquentiel.
 - ✓ La perte d'un chaînage entraîne la perte de tout le reste du fichier.

Allocation indexée

Les adresses des blocs physiques constituant un fichier sont rangées dans une table appelée **index**, elle-même contenue dans un ou plusieurs blocs disque.

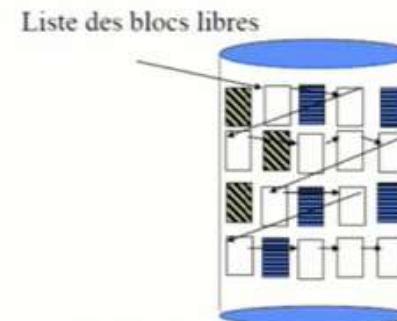


- Supporte bien l'accès direct
- «Perte de place » dans le bloc d'index

Gestion de l'espace libre

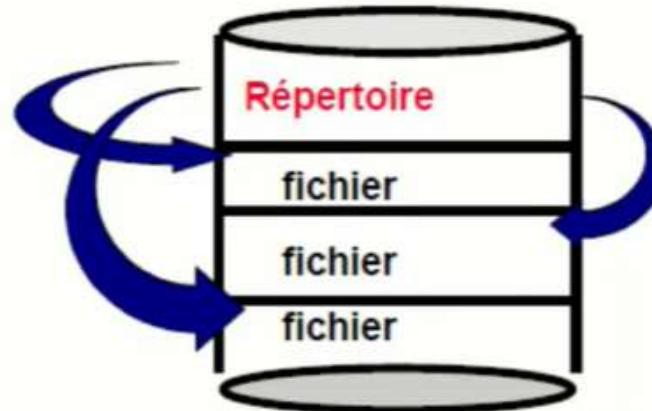
Les systèmes d'exploitation utilisent deux approches pour mémoriser l'espace libre : une statique et une dynamique.

- *Approche statique utilise une table de bits. A chaque bloc du disque, correspond un bit dans la table, positionné à 1 si le bloc est occupé, à 0 si le bloc est libre.*
- *Liste chaînée: Approche dynamique utilise une liste chaînée constituée d'éléments, chacun mémorisant des numéros de blocs libres.*
 - *Nécessite le parcours d'une grande partie de la liste chaînée*
 - *Difficile de trouver un groupe de blocs libres*



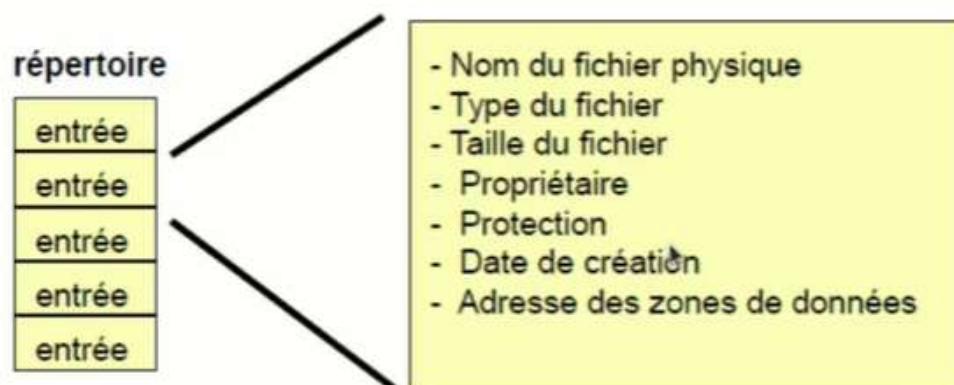
Le répertoire

- ❑ Le répertoire est une table sur le support permettant de référencer tous les fichiers existants du SGF avec leur nom et leurs caractéristiques principales.
- ❑ Le répertoire stocke pour chaque fichier l'adresse des zones de données allouées au fichier.



Le répertoire (suite)

- Un répertoire est une zone disque réservée par le SGF.
- Le répertoire comprend un certain nombre d'entrées.
- Une entrée de répertoire concernant un fichier donné, contient les informations suivantes :

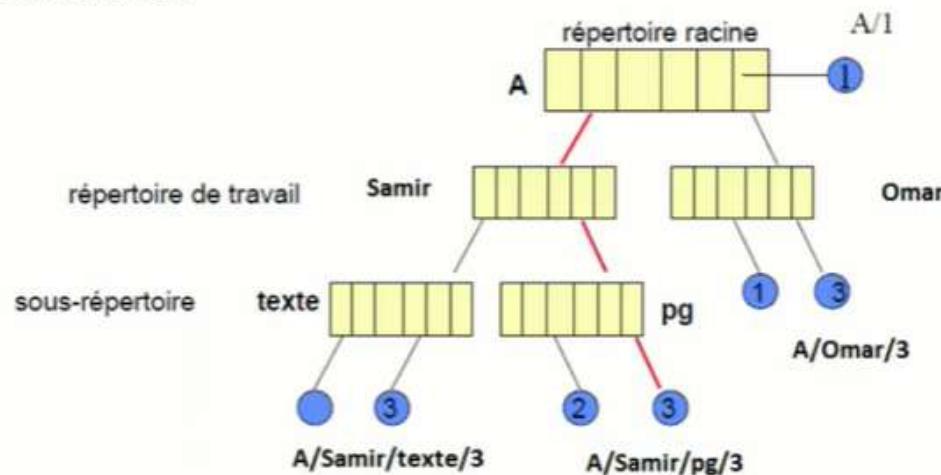


1 entrée : attributs du fichier physique

Organisation des répertoires

Répertoire à structure arborescente :

- ❑ Chaque utilisateur dispose d'un sous-répertoire propre (Répertoire de travail).
- ❑ L'utilisateur peut créer des sous-répertoires à l'intérieur de son répertoire de travail.



- *Identifié par un nom, sans structure logique (suite d'octets) .*
- *La méthode d'allocation mise en œuvre est de type allocation indexée.*
- *Un fichier Linux est composé d'un descripteur appelé «**inode**» et de blocs physiques, qui sont soit des blocs d'index, soit des blocs de données.*
- *Un bloc est identifié par un numéro codé sur 4 octets. La taille d'un bloc est un multiple de la taille d'un secteur (512 octets)*

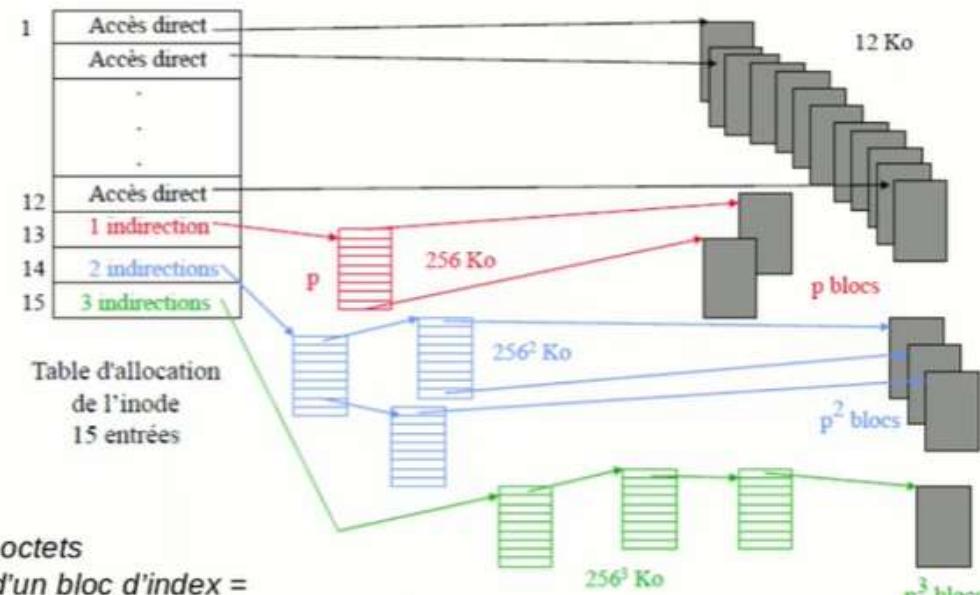
Fichier Linux: inode

L'inode du fichier est une structure stockée sur le disque, allouée à la création du fichier et repérée par un numéro. Contient les attributs du fichier :

- ❑ Nom
- ❑ Type : fichiers normaux, répertoires, périphériques, tubes nommés, sockets
- ❑ Droits d'accès
- ❑ Heures diverses
- ❑ Taille du fichier en octets
- ❑ Table des adresses des blocs de données

Fichier Linux: structure

L'inode du fichier contient un tableau de `EXT2_N_BLOCKS` entrées qui égale par défaut à 15. L'organisation de cette table suit l'allocation indexée.



*JE VOUS SOUHAITE BONNE CHANCE
MERCI
PR. DR. ING. ABDELLAH AMINE*