Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master ISIL

Systèmes d'exploitation

A. ABBAS
abbasakli@gmail.com

Préambule

- Pré-requis: Cours (algorithmique, SE1, 2ème année Licence).
- Évaluation: continu + Examen
- Volume horaire hebdomadaire: cous 1,5h TD = 1,5 Coefficient =2

Crédit=2

Objectifs de l'enseignement :

Approfondir les différents concepts utiles pour la conception d'un système d'exploitation ou la programmation système.

CONTENU DU COURS

1. Introduction

Introduction aux Systèmes d'exploitation
Concepts du processus
Concepts du threads

2. SYSTEMES DE FICHIERS

+ Rappels sur l'interface des systèmes de fichiers
+ Structure d'un système de fichiers (organisation, montage)
+ Organisation physiques des fichiers (allocation contiguë, chaînée, indexée)
+ Gestion de l'espace libre (vecteur binaire, liste chaînée, groupement)
+ Implémentation des répertoires (linéaire, table de hachage)
+ Gestion des fichiers actifs: partages de fichiers
+ Protection
+ SGF sous Unix

3. SYNCHRONISATION DES PROCESSUS

+ Problème de l'exclusion mutuelle
+ Synchronisation
  . Sémaphores,
  . Moniteurs
  . Événements,
+ Exemples sous UNIX

CONTENU DU COURS

4. COMMUNICATION ENTRE PROCESSUS

+ Partage de variables (modèle de producteur/ consommateur, lecteurs/rédacteurs)

5. PROTECTION ET SECURITE

+ Protection
  - . Domaine de protection
  - . Matrices de droits
  - . Protection et langages évolués.
  - . Exemple de systèmes de protections
+ Sécurité
  - . Authentification
  - . Menaces
  - . Surveillance des menaces
  - . Cryptage

Chapitre 1:

Introduction

Introduction aux S.E

Le système d'exploitation et le système informatique :

➢Le **matériel** (hardware) d'un système informatique est composé :

de **processeurs** qui exécutent les instructions, de **mémoire centrale** qui contient les données et les instructions à exécuter (en binaire), de **mémoire secondaire** qui sauvegarde les informations et de **périphériques d'Entrées/Sorties** (clavier, souris, écran, modem, etc.) pour introduire ou récupérer des informations.

Les **logiciels** (software), d'un système informatique, sont à leur tour divisés en **programmes système** qui fait fonctionner l'ordinateur : **le système d'exploitation** et les **utilitaires** (compilateurs, éditeurs, interpréteurs de commandes, etc.) et en **programmes d'application** qui résolvent des problèmes spécifiques des utilisateurs.

Introduction aux S.E

Définition

- Un Système d'Exploitation (noté SE ou OS, abréviation du terme anglais Operating System) peut être défini comme un logiciel qui , dans un système informatique, pilote les dispositifs matériels et reçoit des instructions de l'utilisateur ou d'autres logiciels (ou applications).

- Le SE constitue donc une interface entre l'utilisateur, et la machine physique.

Introduction aux S.E

Les composants du S.E:

- **Le noyau** (en anglais kernel) : représentant les fonctions fondamentales du système d'exploitation telles que la gestion de la mémoire, des processus, des fichiers, des entrées-sorties principales, et des fonctionnalités de communication.

- **L'interpréteur de commande** (en anglais shell) permettant la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes.

- **Le système de fichiers** (en anglais «file system», noté FS), permettant d'enregistrer les fichiers dans le disque sous forme d'une arborescence.

- **Les pilotes :** Les pilotes sont fournis par l'auteur du système d'exploitation ou le fabricant du périphérique.

Introduction aux S.E

Rôle du système d'exploitation

- **Gestion des processeurs:** Gérer l'allocation des processeurs entre les différents programmes grâce à un algorithme d'ordonnancement.
- **Gestion des processus** (programmes en cours d'exécution) : Gérer l'exécution des processus en leur affectant les ressources nécessaires à leur bon fonctionnement.
- **Gestion des mémoires:** Gérer l'espace mémoire alloué à chaque processus.
- **Gestion des fichiers:** Gère l'organisation du disque dur et du système de fichiers
- Donne l'illusion du **multitâche.**

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.

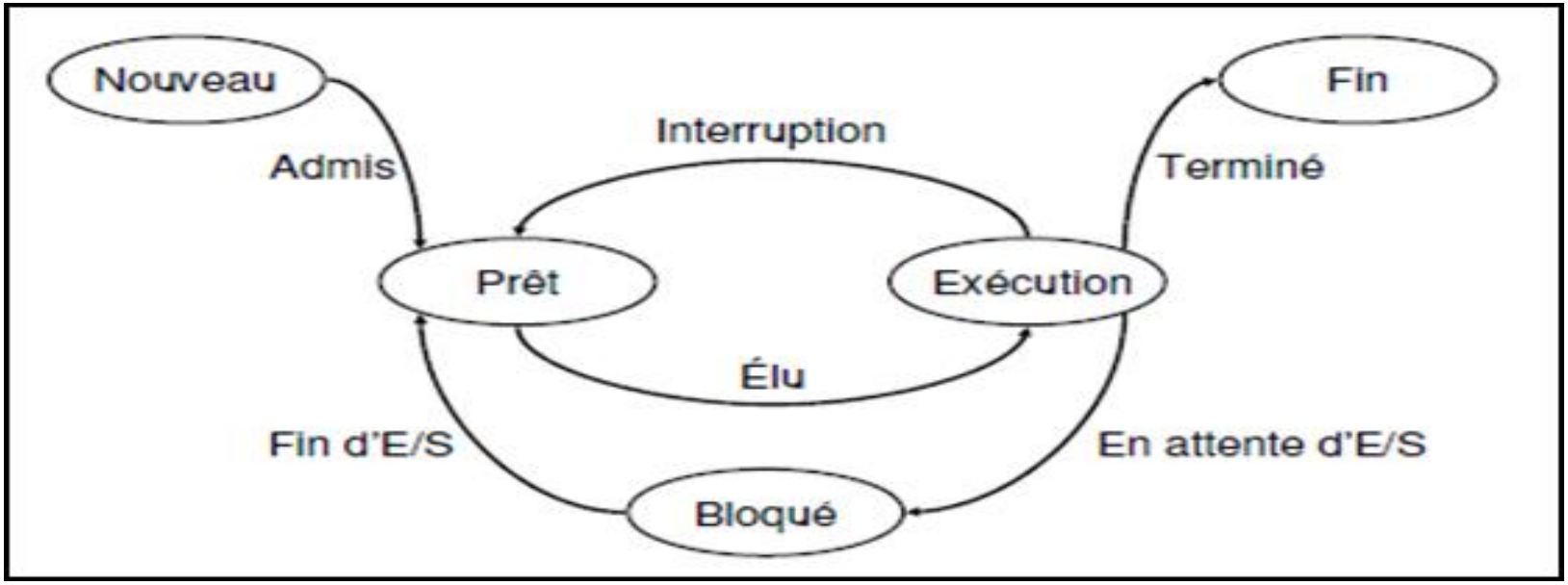
Un processus est caractérisé par:

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

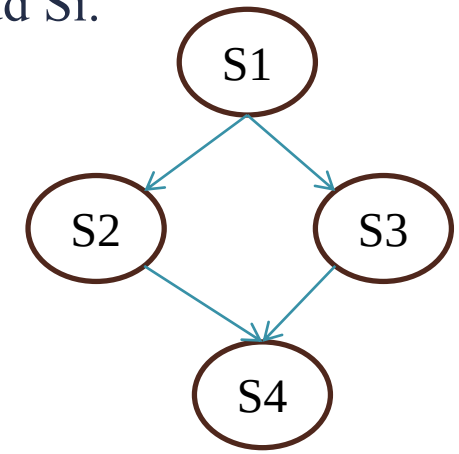
1. L'algorithme du Premier Arrive Premier Servi (FCFS)
2. L'algorithme du Plus Court d'abord (SJF)
3. Ordonnancement avec priorité
4. L'algorithme de Round Robin (Tourniquet)
5. Ordonnancement avec les d'attente multiniveaux

Activité Parallèle

Avec l'introduction du système à temps partagé (système d' interruption) et les machines multi-processeurs (Multi-core) le système d'exploitation permet l'exécution de plusieurs programmes en parallèle.

Graphe de précédence

C'est un graphe dirigé acyclique dont les nœuds représentent les instructions et les arcs les dépendances entre les nœuds incidents : un arc d'un nœud Si vers un nœud Sj signifie que le nœud Sj ne peut s'exécuter avant le fin du nœud Si.



S2 et S3 peuvent s'exécuter en parallèle après la fin S1

Comment détecter les instructions parallèles ?

Activité Parallèle

Condition de Bernstein:

Soit I une instruction d'un programme, on dénote par:

R(I): l'ensemble des variables de I qui ne change pas par l'exécution de I

W(I): l'ensemble des variables de I qui sont mises à jours par l'exécution de I.

On dira que deux instructions I1 et I2 peuvent s'exécuter en parallèle si les conditions suivantes sont satisfaites :

a) $R(I1) \cap W(I2) = \varnothing$

b) $W(I1) \cap R(I2) = \varnothing$

c) $W(I1) \cap W(I2) = \varnothing$

Exemple

```
main (int ac, char **av){
        int a, b;
        scanf ("%d", a);
        scanf ("%d", b);
        c=a+b;
        printf("c= %d\n", c);
}
```

R(scanf ("%d", a)) = ∅ ; W(scanf ("%d", a)) = {a}

R(scanf ("%d", b)) = ∅ ; W(scanf ("%d", b)) = {b}

R(c=a+b)= {a,b} ; W(c=a+b) = {c}

R(printf("c= %d\n", c))= {c} ;

W(printf("c= %d\n", c)) = {}

L'instruction scanf ("%d", b)) ne peut s'exécuter en // avec c=a+b; car W(scanf ("%d", b)) ∩ R(c=a+b)= {b}

Activité Parallèle

Condition de Bernstein:

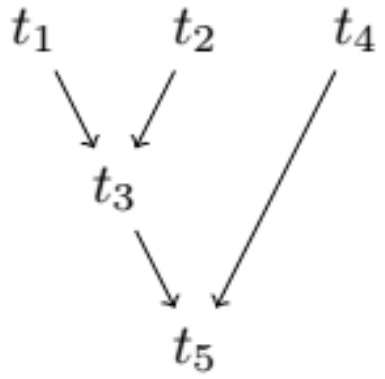
Exercice:

On cherche à évaluer $(a * b + c * d)/(e - f)$

On réalise un découpage en tâches $t1,...,tn$

$t1$: a := a*b ;
$t2$: c := c*d ;
$t3$: a := a+c ;
$t4$: e := e - f ;
$t5$: a := a / e ;

Graphe de précédence :



Donner le graphe de précédence :

La création de processus

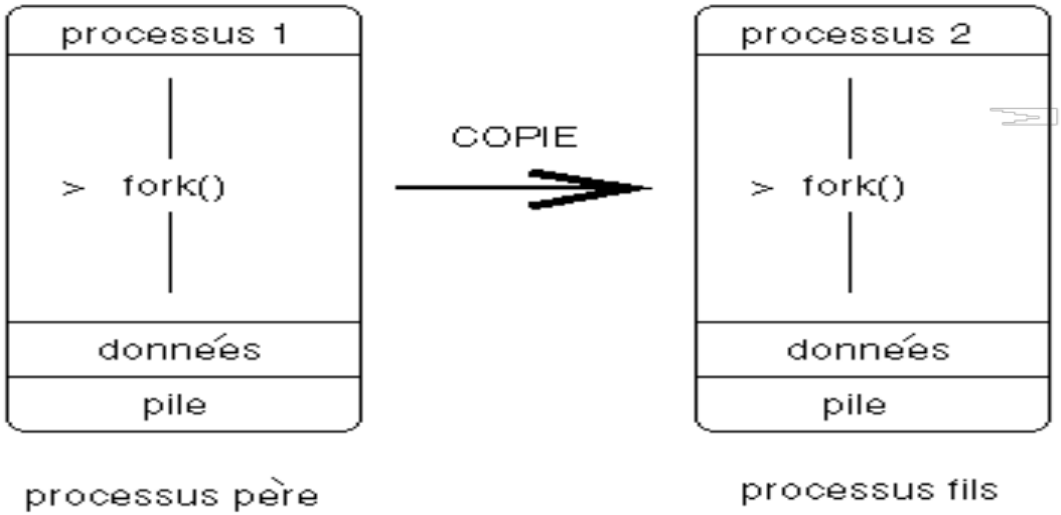
L'appel système ***fork()*** est une façon qui permet de créer des processus aussi appelés **processus lourds**.

Exemple 1 : Instruction de création :

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

fork() est le moyen de créer des processus, par duplication 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.

```
1.  #include <sys/types.h>//pour le type pid_t
2.  #include <unistd.h>//pour fork
3.  #include <stdio.h>//pour perror, printf
4.  int main(void){
5.  pid_t p;
6.  int a=20;
7.  switch(p=fork()){ //création d'un fils
8.  case 1: //le fork a échoué
9.          perror("le fork a échoué!");
10.         break;
11. case 0: //Il s'agit du processus fils
12.         printf("ici processus fils, le PID=%d\n",       getpid());
13.         a+=10;
14.         break;
15. default: //Il s'agit du processus père
16.          printf("ici processus pere ,le PID=%d \n",getpid()); a+=100;
17. }//les deux processus exécutent cette instruction
18.     printf("Fin du processus%d avec a=%d \n",getpid(),a);
19. return 0 ;
20.  }
```

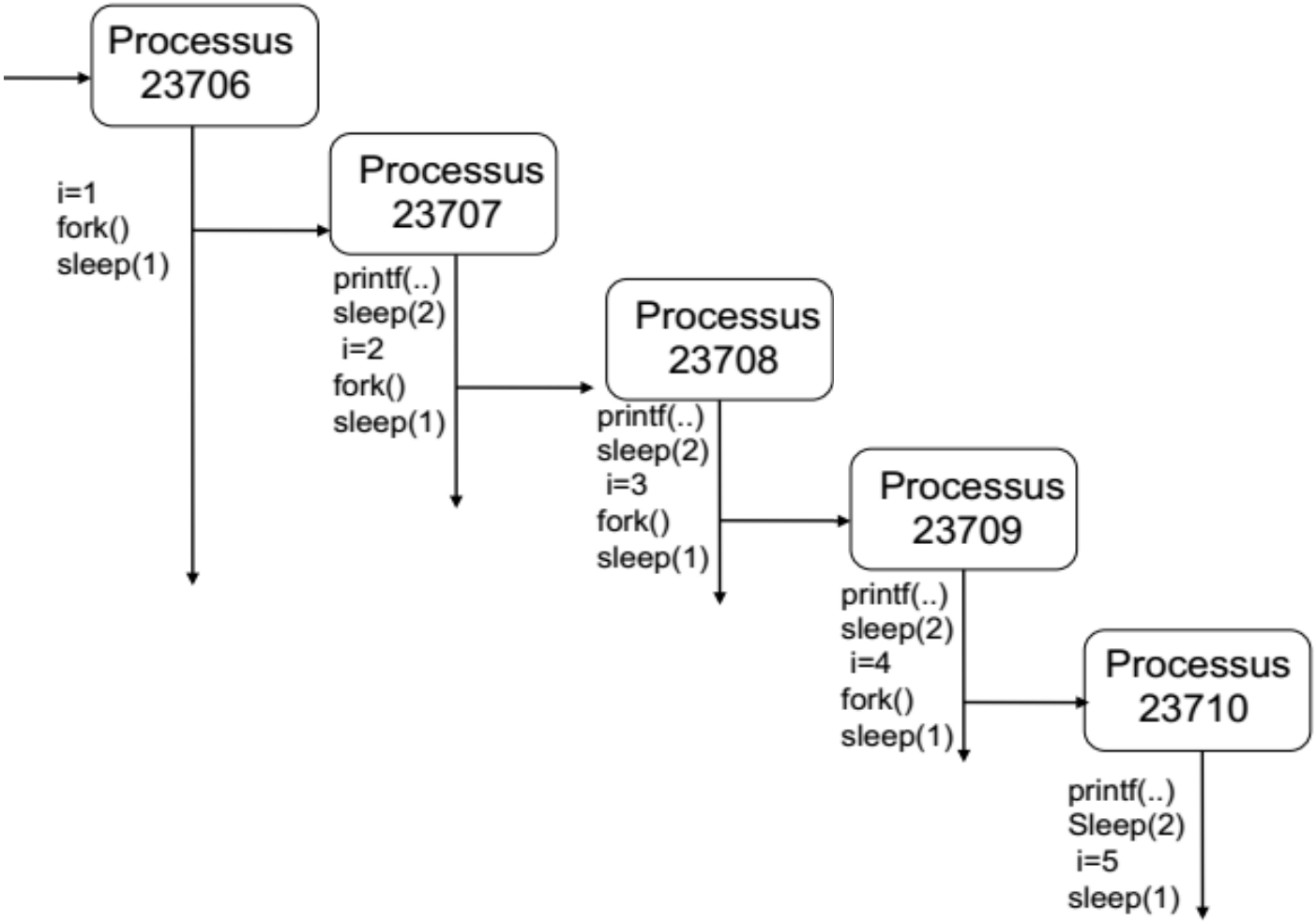
La création de processus

Exemple 3 : programme tree.c : appel système fork()

```
1.  #include <sys/types.h>
2.  #include <unistd.h>
3.  #include <stdio.h>
4.  int main(int argc, char **argv) {
5.          pid_t p;
6.          int i, n=5;
7.  for (i=1; i<n; i++) {
8.          p = fork();
9.          if (p > 0)
10.         break ;
11.         printf(" Processus %d de père %d. \n", getpid(), getppid());
12.         sleep(2);
13.         }
14. sleep(1);
15. return 0;
16. }
```

La création de processus

Exemple 3 : programme tree.c : appel système fork()



La hiérarchie des processus

Le système d'exploitation fournit un ensemble d'appels système qui permettent la création, la destruction, la communication et la synchronisation des processus.

Un processus fils peut partager certaines ressources (mémoire, fichiers) avec son processus père ou avoir ses propres ressources. Le processus père peut contrôler l'usage des ressources partagées.

Le processus père peut avoir une certaine autorité sur ses processus fils. Il peut les suspendre, les détruire (appel système **kill**), attendre leur terminaison (appel système **wait**) mais ne peut pas les renier.

Attente de la fin d'un processus: wait, waitpid

Afin de synchroniser les processus entre eux, il est nécessaire qu'un parent puisse attendre qu'un de ses enfants se termine. Il existe deux fonctions qui permettent d'accomplir cette tâche :

```
pid_t   wait(int *status)
pid_t   waitpid(pid_t pid, int *status, int options)
```

Ces deux fonctions attendent qu'un enfant se termine : dans le cas de wait, on attend le premier enfant qui se termine, alors que dans le cas de waitpid, on attend que l'enfant d'identificateur pid se termine.

Un appel à la fonction wait ou waitpid bloque le processus appelant.

Pour les deux fonctions, la valeur de retour est l'identificateur de l'enfant qui vient de se terminer.

```
while (wait(NULL)>=0) ; /* Attendre la fin de chaque enfant */
```

L'entier **status** nous indique la manière dont le processus fils s'est terminé. Pour l'interpréter, on peut utiliser des macros prédéfinies :

WIFEXITED (status) : le code de retour de la macro est non nul si le fils s'est terminé normalement
WEXITSTATUS (status) : donne le code de retour de l'enfant, provenant soit de l'appel exit() ou du return de la routine main.
…

Concept de threads

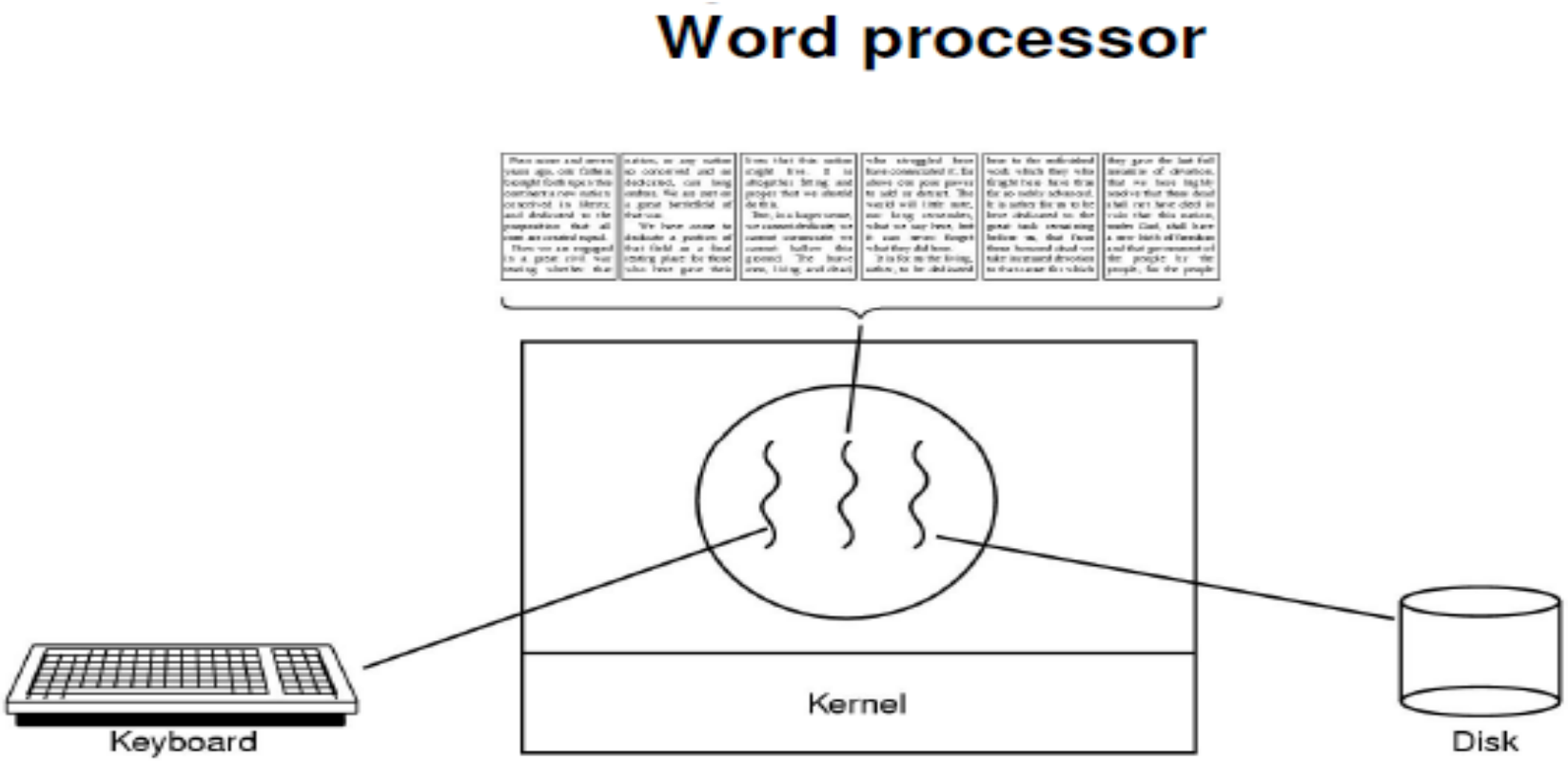
- Les threads appelés aussi processus légers, comme les processus, est un mécanisme permettant à un programme de faire plus d'une chose à la fois (c.-a-d. exécution simultanée des parties d'un programme).

- Un thread est une unité d'exécution rattachée à un processus permettant son exécution.

- Comme les processus, les threads semblent s'exécuter en parallèle; le noyau du SE les ordonnances, interrompant chaque thread pour donner aux autres une chance de s'exécuter.

- Lorsqu'un programme crée un nouveau thread, dans ce cas, rien n'est copie. Le thread créateur et le thread crée partagent tous deux le même espace mémoire, les mêmes descripteurs de fichiers et autres ressources.

Cas d'utilisation de threads

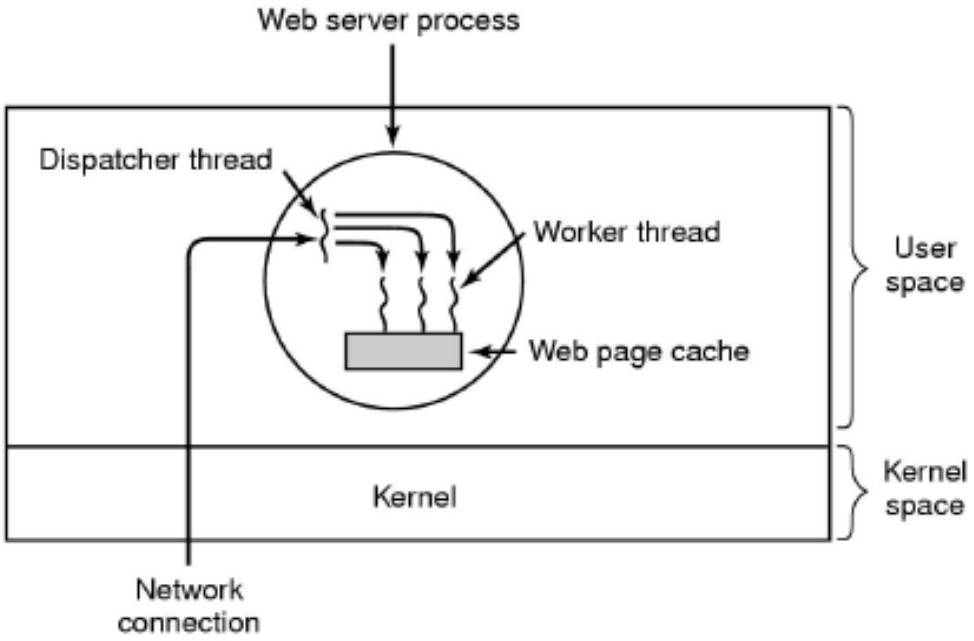


- Un thread pour interagir avec l'utilisateur,
- Un thread pour reformater en arrière plan,
- Un thread pour sauvegarder périodiquement le document

Cas d'utilisation de threads

Serveur Web

- Le Dispatcher thread se charge de réceptionner les requêtes.
- Il choisit, pour chaque requête reçue, un thread libre (en attente d'une requête). Ce thread va se charger d'interpréter la requête.




```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Création de Threads

La fonction **pthread_create** crée un nouveau thread.

Syntaxe:

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

Paramètres du thread :

- Un pointeur (*thread) vers une variable *pthread_t*, dans laquelle l'identifiant (TID) du thread sera stocke;
- Un pointeur (*attr) vers un objet d'attribut de thread `(taille de la pile, priorité….). Par défaut, il prend NULL
- Un pointeur (* nomFonct) vers la fonction de thread. Il s'agit d'un pointeur de fonction ordinaire de type: void* (*funct) (void*);
- Une valeur d'argument de thread de type void*.

L'appel renvoie 0 s'il réussit, sinon il renvoie une valeur non nulle identifiant l'erreur qui s'est produite

Création de Threads

Fonctions pthread_join(), pthread_self() et pthread_exit()

void pthread_join(pthread_t tid, void * *status);

• Attend la fin d'un thread. L'équivalent de waitpid des processus sauf qu'on doit spécifier le tid du thread à attendre.

• status sert à récupérer la valeur de retour et l'état de terminaison.

pthread_t pthread_self(void);

• Retourne le TID du thread.

void pthread_exit(void * status);

• Termine l'exécution du thread

Création de Threads

```
#include <pthread.h>
#include <stdio.h>
void* A (void *data){     /* Affiche Hello world */
        printf ("Hello world \n");
        return 0
}
int main (){              /* Le programme principal. Processus*/
        pthread_t thread_id;
/* Cree un nouveau thread. Le nouveau thread exécutera la fonction A */
        Pthread_create (&thread_id, NULL, &A, NULL);
/* Attend la fin du thread.
        pthread_join (thread_id, NULL);
        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 sur le disque.

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**

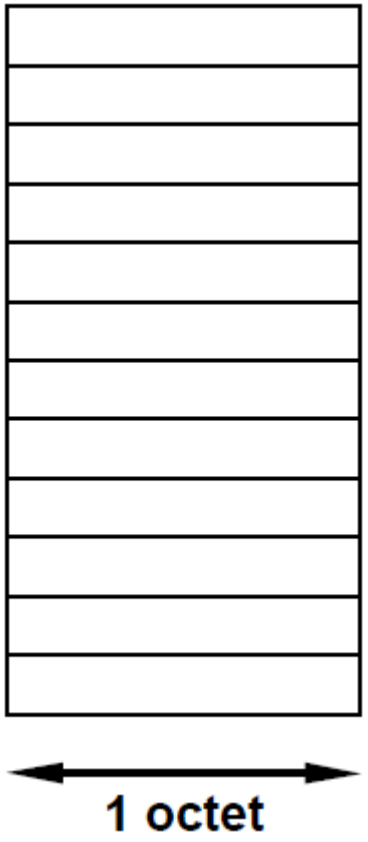
sans structure

– 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

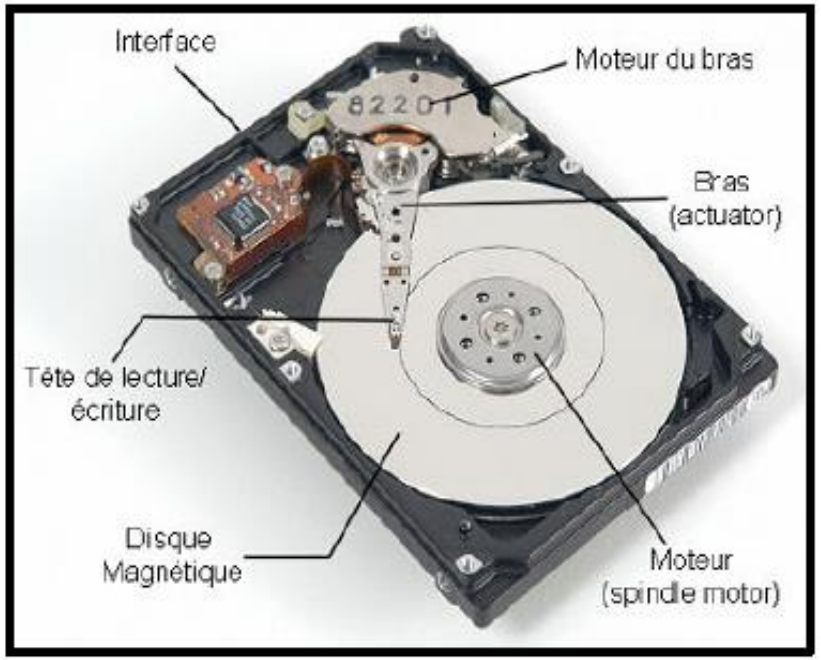
– Les fichiers séquentiels étaient pratiques quand le support de stockage était une bande magnétique .

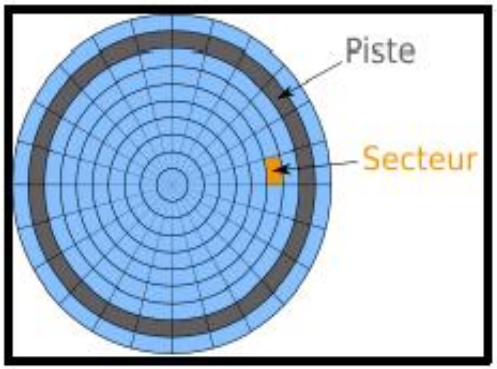
L'accès au fichiers

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)

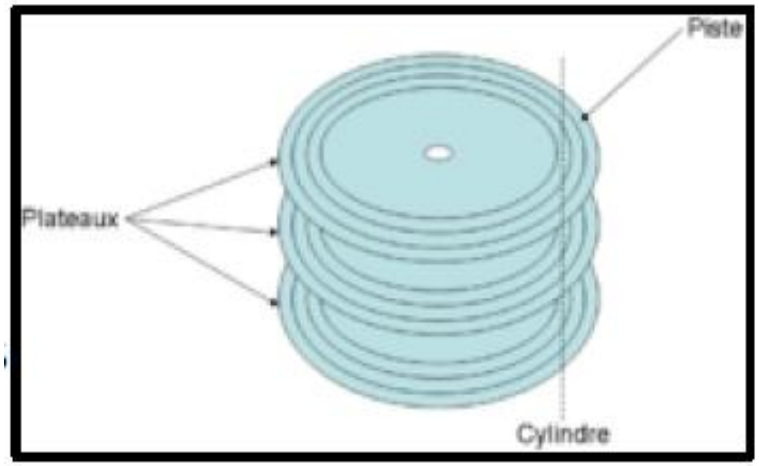
Le système de fichiers





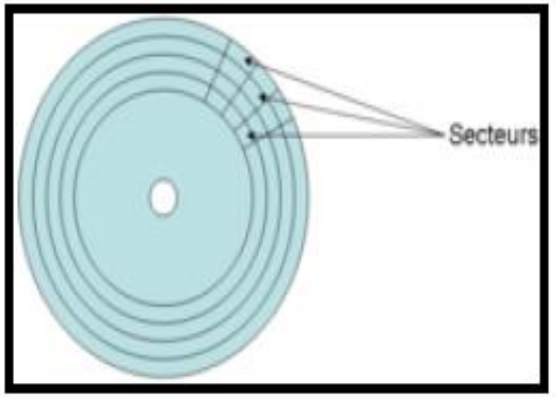
Le système de fichiers

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

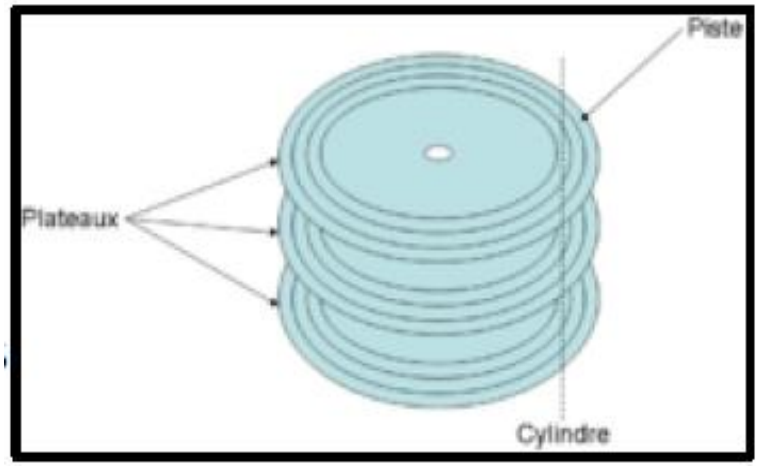
Head:
– Le plus souvent il y a une tête par surface soit deux par plateau
– Le chemin parcouru par une tête particulière sur un cylindre particulier s'appelle une piste



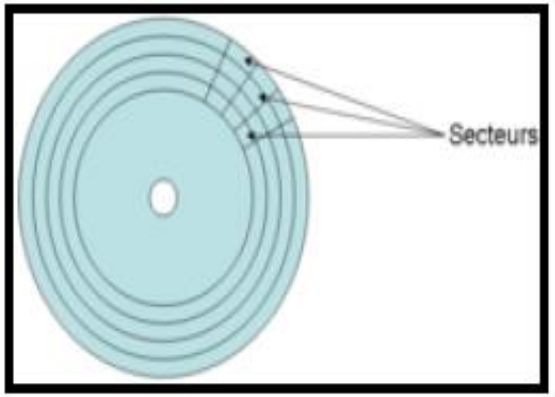
Secteur :
– La piste est divisée en secteurs contenant les données.
– **La taille d'une piste diminue en allant vers le centre du disque → le nombre de secteur par piste diminue aussi en allant vers le centre**
– Taille d'un secteur plus commune= 512 octet

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.



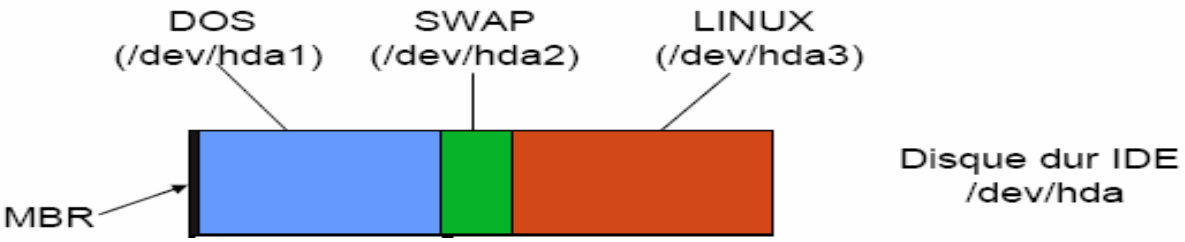
Le suivant sera 0 / 0 / 2 (ce secteur est naturellement atteint juste après par la même tête)

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.

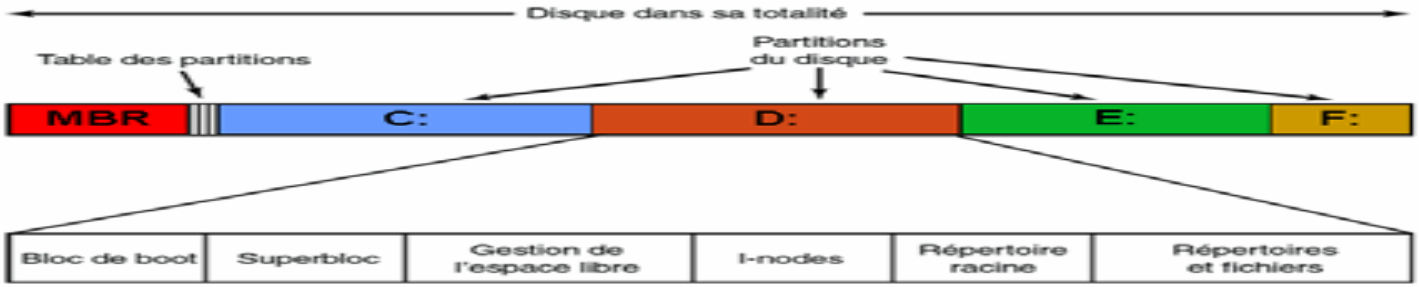
Le système de fichiers

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

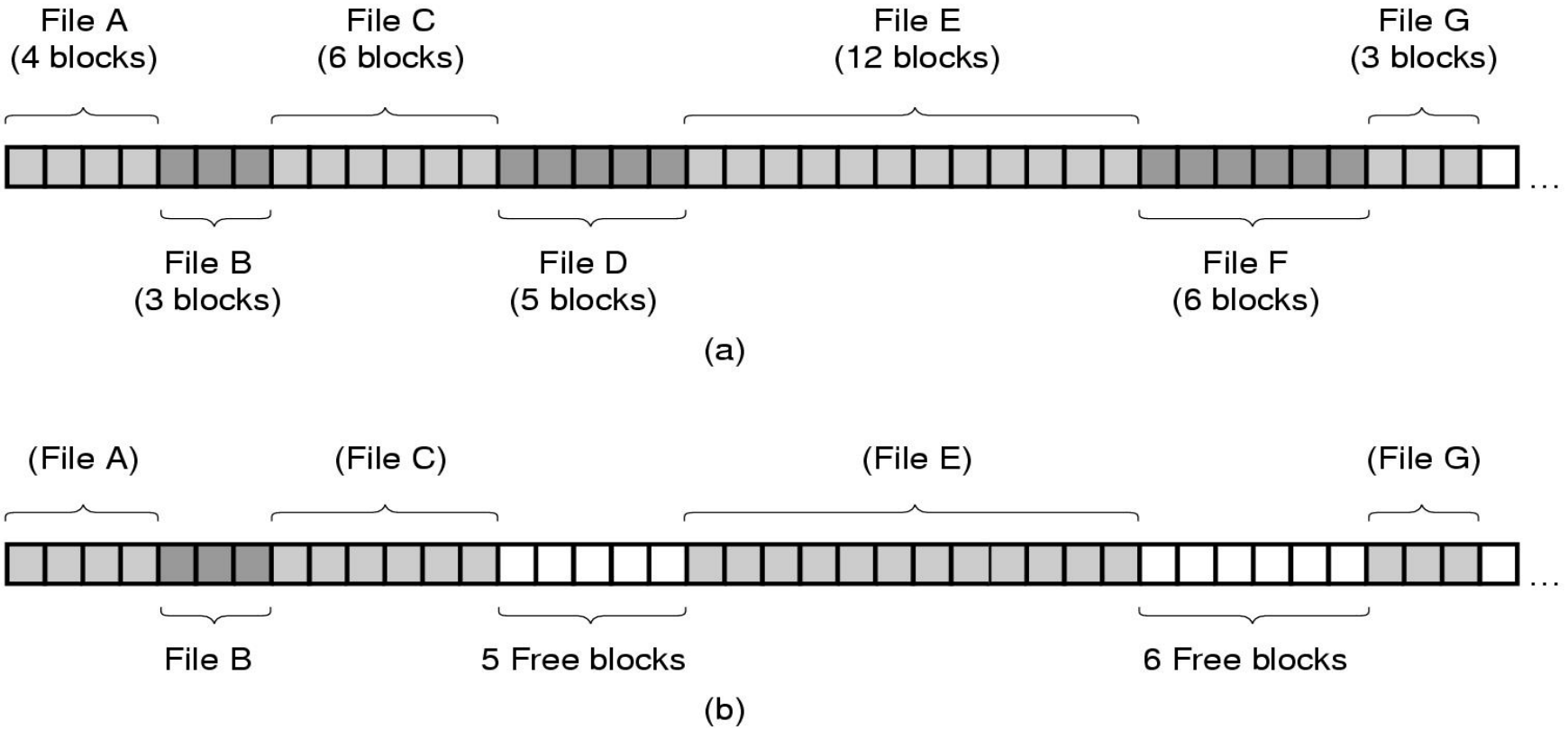
Problèmes

- 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

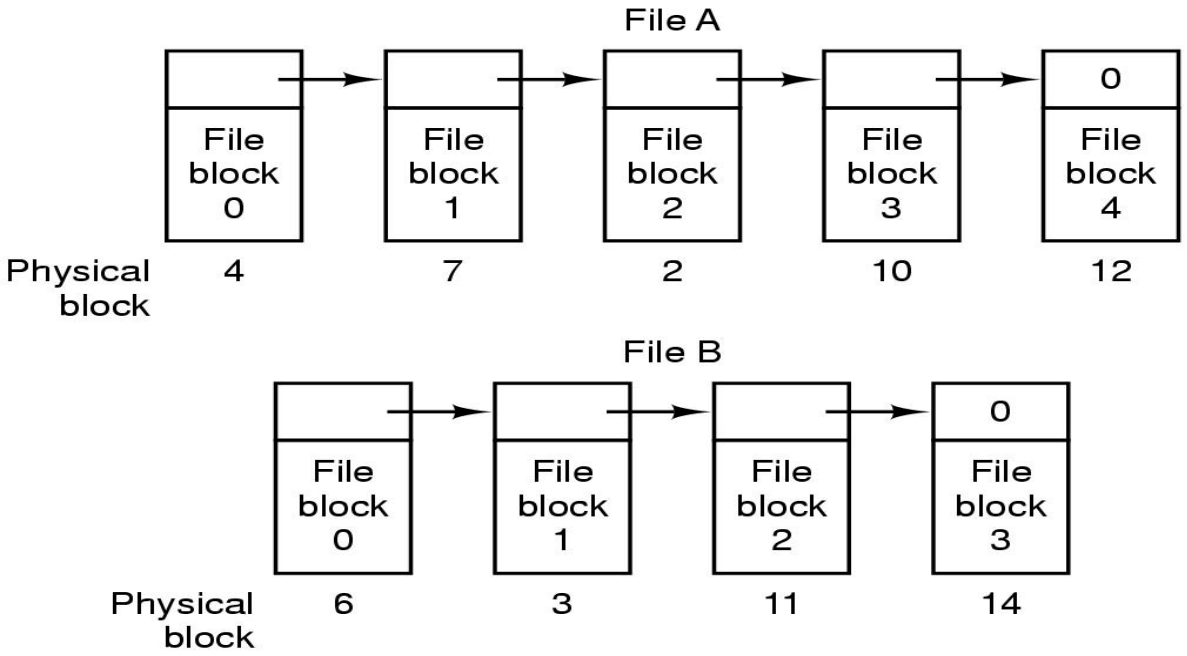
(b) Suppression des fichiers *D et F*

=> 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 :

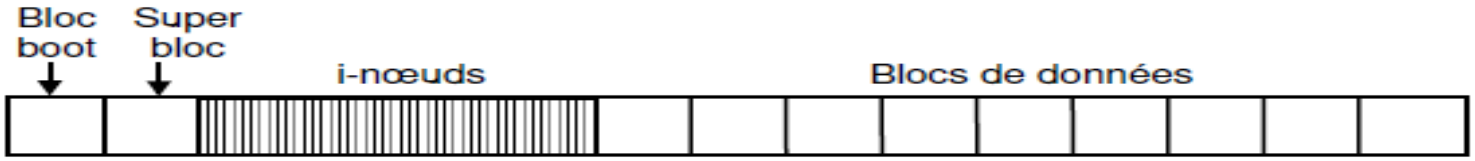
$$2^{16} \text{ x } 32 \text{ Ko} = 2097152 \text{ Ko} = 2\text{Go}$$

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.

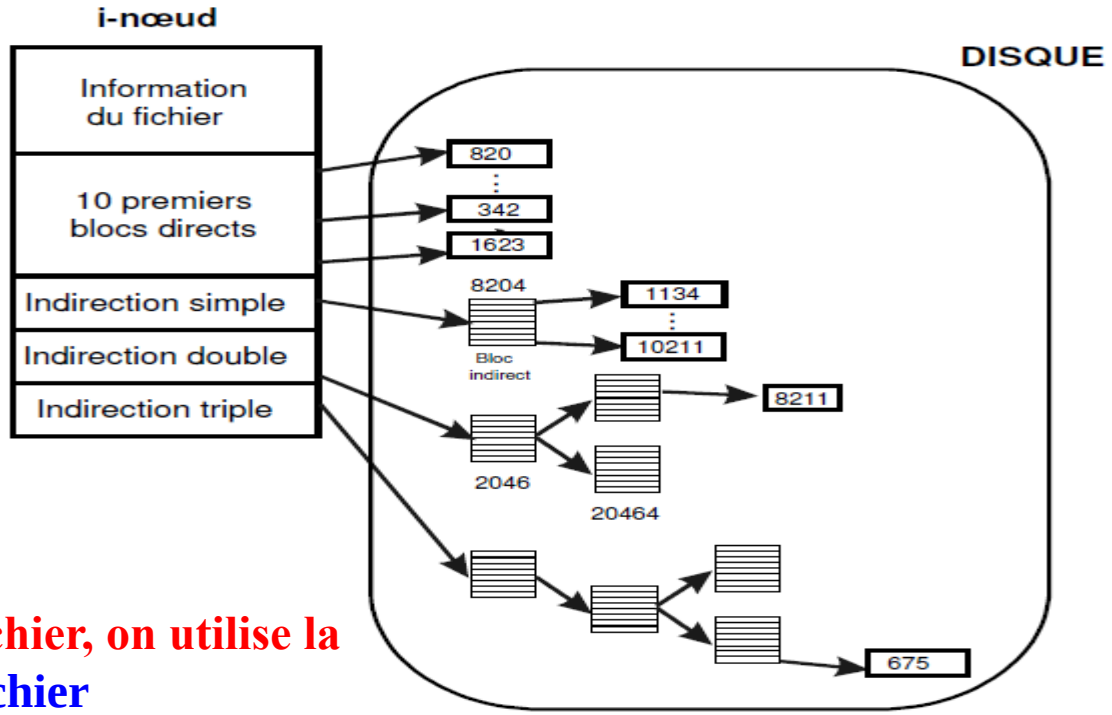
Le système de fichiers

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
- ….



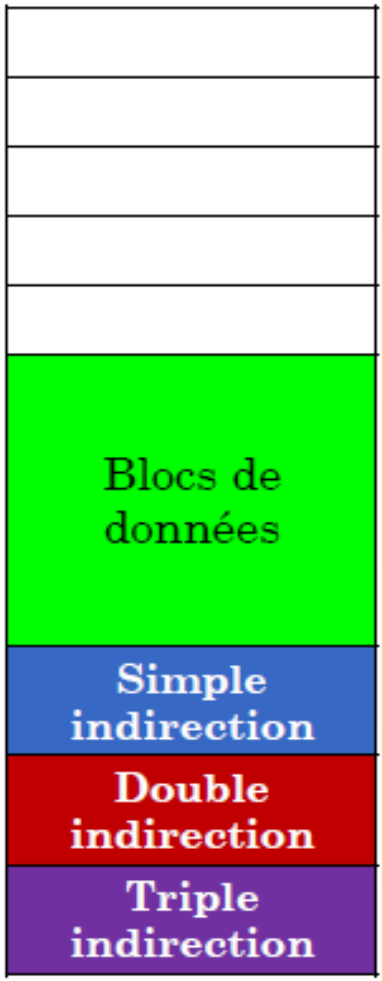
ext3fs d'Unix ou GNU/Linux (ext3fs pour *third extented file system)*

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

Le système de fichiers

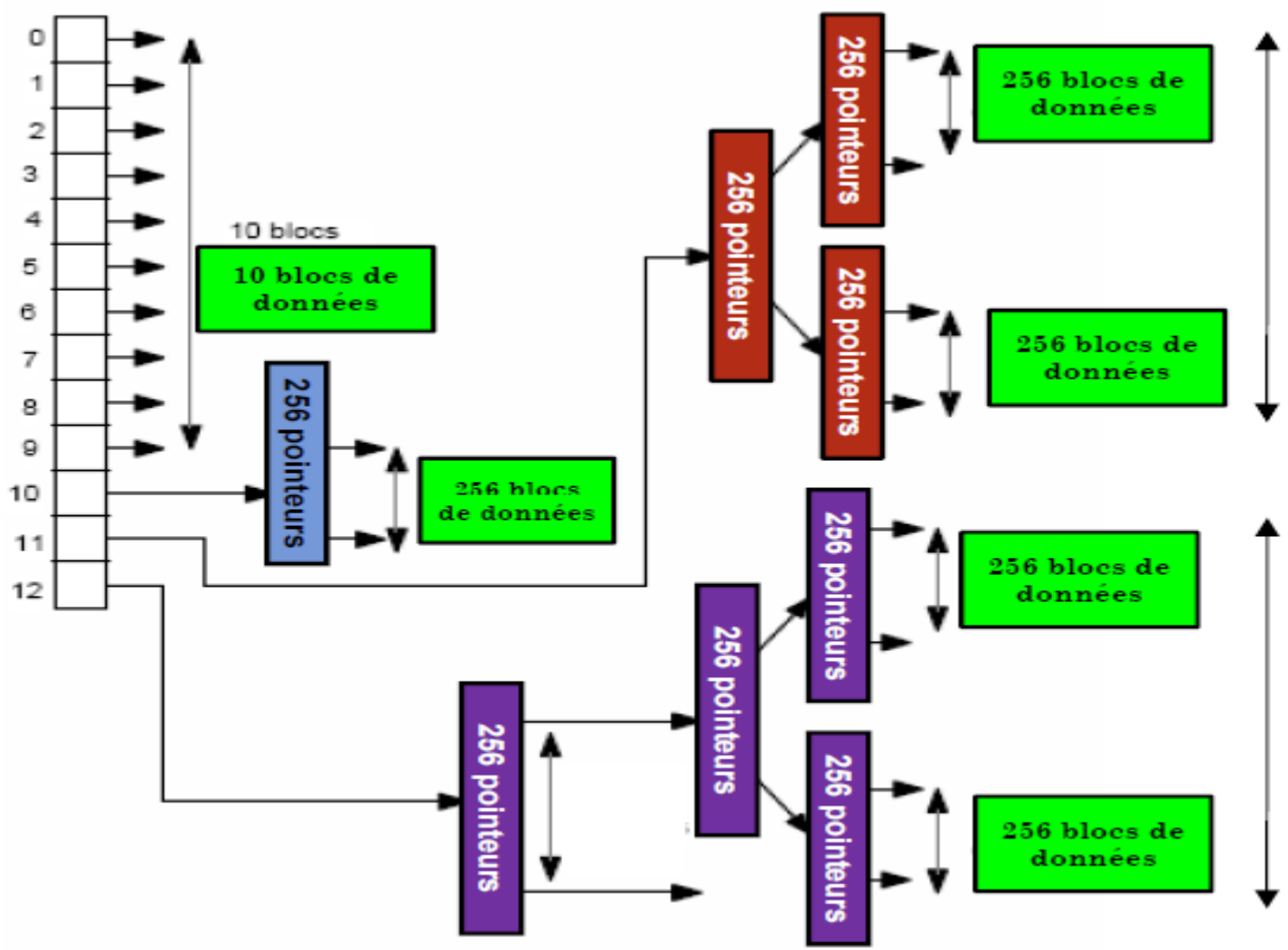
Allocation non contiguë - Tables d'index des i-noeuds

- 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 :
$(10x1k) + (256x1k) + (256^2x1k) + (256^3x1k) > 16$ 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

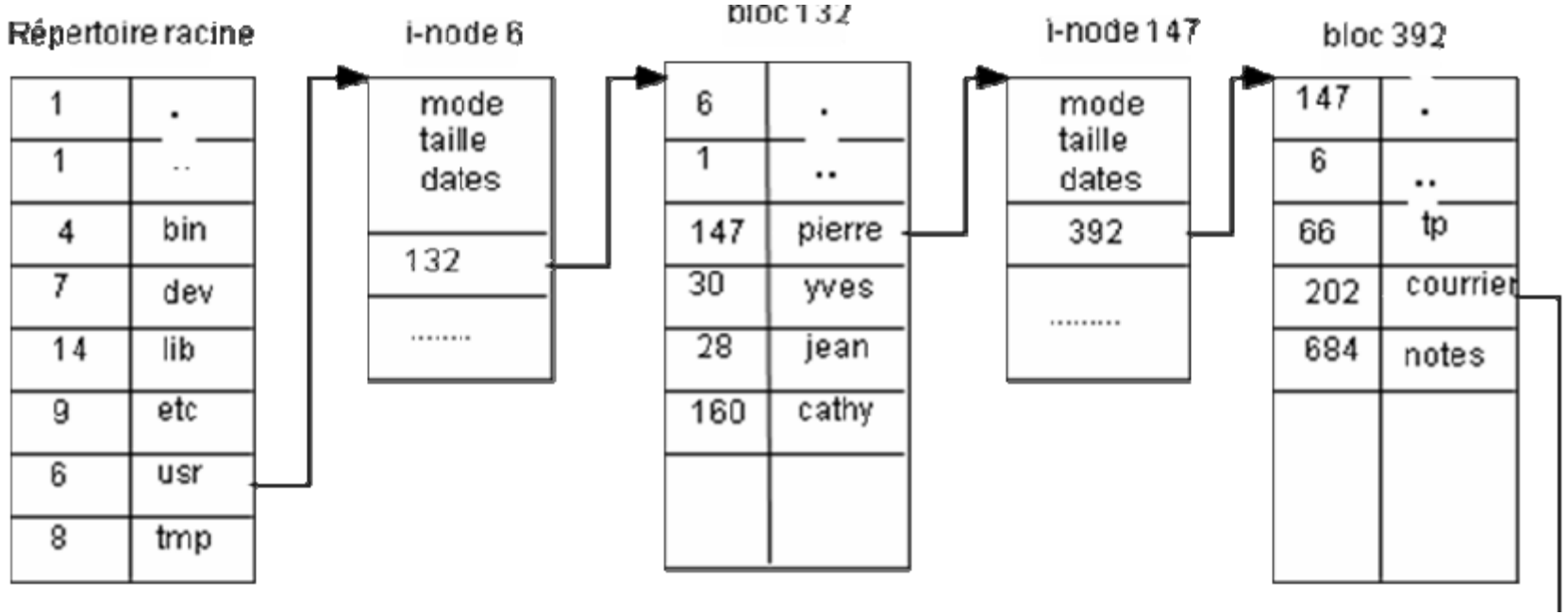
- L'avantage par rapport à une FAT est que seul l'inode d'un fichier ouvert a besoin d'être en mémoire
- Une FAT augmente linéairement avec la taille du disque
- La mémoire occupée par les inodes augmente avec le nombre de fichiers ouverts

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)

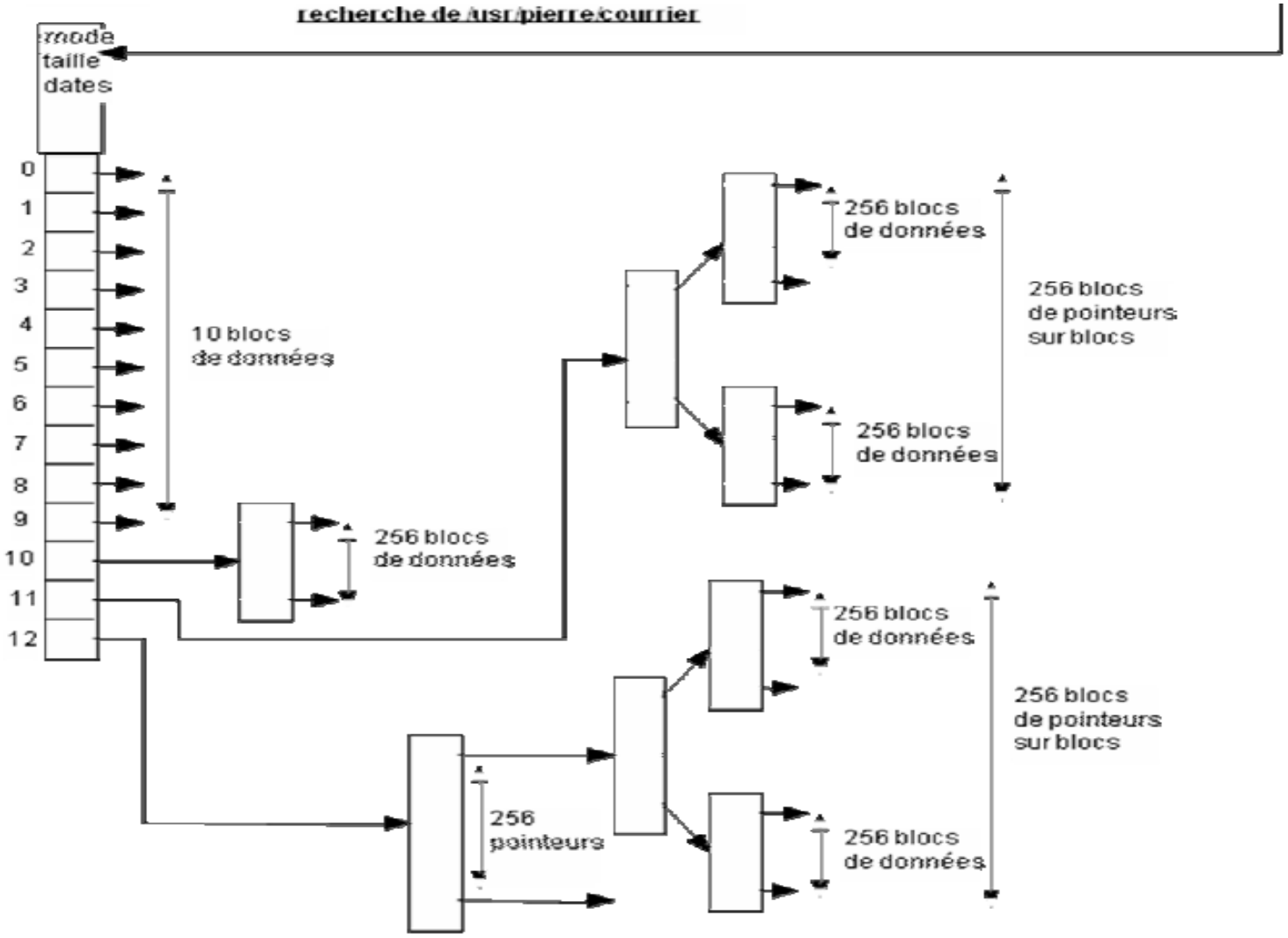
Le système de fichiers

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

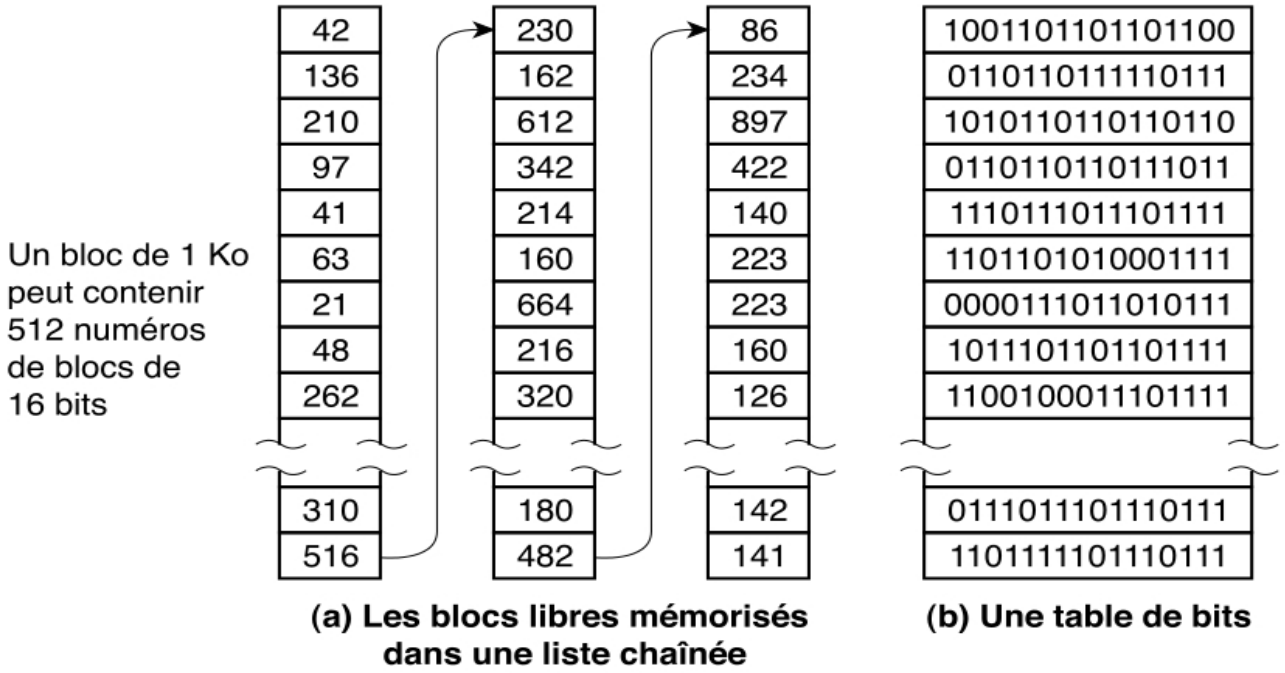
Organisation de l'espace du disque
Repérage des blocs libres

- 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

Le système de fichiers

Les verrous d'accès a un fichier

Unix/linux étant un système multiutilisateur, il se peut que deux utilisateurs veuillent accéder en même temps au même fichier. Si les fichiers sont consultés en lecture, il n'y a pas de conflit. Les problèmes surgissent lorsqu'un des processus modifie le fichier pendant que d'autres le lisent. Sans précaution, des informations erronées peuvent être enregistrées. Un processus lit un enregistrement dans son espace d'adressage, le modifie et le réécrit sur disque. Si deux processus lisent une même zone du fichier, la modifient et la réécrivent, les résultats dépendront de l'ordre de réécriture, mais les modifications d'un des processus seront perdues. On a un problème d'accès concurrent, et il faut édicter des règles pour que tout se passe bien.

Le système de fichiers

Les verrous d'accès a un fichier

Le SE maintient alotrs une table des verrous. Les verrous sont des mécanismes de contrôle d'accès aux fichiers. Les verrous disposent de 2 caractéristiques :

- sa porte : les numéros logiques de début et de fin auxquels le verrou s'applique. On peut ainsi verrouiller qu'une partie de fichier ;
- son type : Deux sortes de verrouillage de fichiers sont disponibles : **partagé et exclusif**. Le verrouillage **partagé** autorise la cohabitation avec un autre verrou (partagé) bloquant le même fichier. Si une partie de fichier contient un **verrou exclusif,** toute tentative d'en verrouiller une quelconque portion sera rejetée tant que le verrou n'est pas relâché.

Ainsi, le SE, doit tester la présence d'un verrou des qu'un processus demande un accès a un fichier et suivant l'état du verrou autoriser ou non la requête.

Le système de fichiers

Les verrous d'accès a un fichier (linux)

flock - Placer ou enlever un verrou coopératif sur un fichier ouvert

#include <sys/file.h>

int flock(int fd**, int** operation**);**

Place ou enlève un verrou consultatif sur un fichier ouvert dont le descripteur est fd.

Le paramètre operation est l'un des suivants :

LOCK_SH : Verrouillage partagé.

LOCK_EX :Verrouillage exclusif.

LOCK_UN : Déverrouillage d'un verrou tenu par le processus.

LOCK_NB : Non bloquer le verrouillage.

et le retour vaut 0 si OK, -1 si erreur.

Le système de fichiers

Les systèmes de fichiers virtuels

Plusieurs systèmes de fichiers sont utilises, souvent sur le même ordinateur, parfois avec un même système d'exploitation

– Un système Windows peut gérer NTFS comme système de fichiers principal mais aussi un système FAT-32 ou FAT-16 par héritage

– Un système Linux peut avoir un système ext2 comme racine du système de fichiers, une partition ext3 montée sur /usr et un second disque dur avec RiserFS.

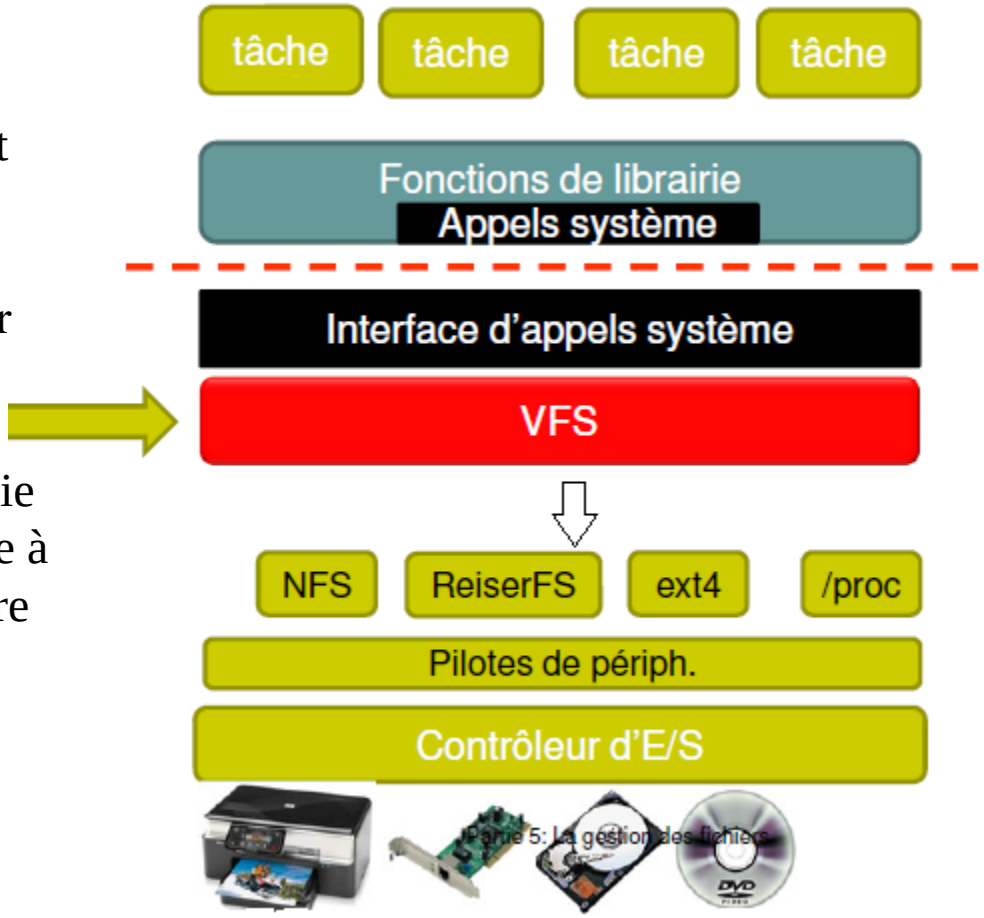
→ Du point de vue de l'utilisateur, c'est un système de fichiers unique et hiérarchique. Ce qui se passe pour intégrer ces divers et incompatibles systèmes de fichiers est invisible aux utilisateurs et aux processus.

Le système de fichiers

Les systèmes de fichiers virtuels

Les systèmes UNIX utilisent le concept de système de fichiers virtuels ou VFS (*Virtual File System)* pour tenter d'intégrer différents systèmes de fichier dans une structure ordonnée.

● L'idée principale est d'extraire la partie du système de fichiers qui est commune à tous les systèmes de fichiers et de mettre le code dans une couche séparée qui appelle les systèmes de fichiers réels sous-jacent pour gérer les données.



On effectue le montage d'un système de fichiers FAT avec la commande shell suivante :
#mount -t vfat /dev/hda1 /mnt/fatpartition

Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Master ISIL

Systèmes d'exploitation

A. ABBAS
abbasakli@gmail.com

Chapitre 2

PROTECTION ET SECURITE

+ Protection
    . Domaine de protection
    . Matrices de droits
    . Protection et langages évolués.
    . Exemple de systèmes de protections
+ Sécurité
    . Authentification
    . Menaces
    . Surveillance des menaces
    . Cryptage

PROTECTION ET SECURITE

Protection et sécurité

Introduction

Le système de fichiers d'un système d'exploitation est devenu la partie la plus exposée aux **dysfonctionnements** où de plus en plus d'informations sont confiées à des systèmes régissant des ordinateurs connectés en réseau, d'où de plus en plus d'attaques possibles (hackers, **virus**), d'où une mise en œuvre de la protection des données.

Causes de dysfonctionnement : 2 types de causes de dysfonctionnement du SE:

- volontaires : erreurs de conception du S.E. lui-même, erreurs matérielles
- involontaires : opérations malveillantes

Il est donc nécessaire d'avoir un système de fichiers stable.

Exemple : une mise en œuvre de sauvegarde permettant l'éventuelle reconstruction de fichiers perdus.

Protection et sécurité

La protection dans les SE

La protection consiste à empêcher qu'un utilisateur puisse altérer un fichier qui ne lui appartient pas et dont le propriétaire ne lui en a pas donné l'autorisation, ou encore à empêcher qu'un processus en cours d'exécution ne modifie une zone mémoire attribuée à un autre processus sans l'autorisation de celui-ci.

Le SE comporte :

- des entités actives ou **sujets** : les processus
- des entités accessibles ou **objets** qui sont :
    - les ressources matérielles : UC, mémoire centrale, entrées/sorties, etc...
    - les ressources logicielles : fichiers, programmes, etc...

Protection et sécurité

Modèle de protection

Le ***modèle de protection*** définit quels sujets ont accès à quels objets suivant quelles modalités d'accès.

Un modèle de protection doit répondre à la question : Quels sujets ont accès à quels objets ? De quelle façon (modalités d'accès) ?

Les *droits d'accès* seront définis par un couple :

(nom d'un objet, modalités d'accès)

Exemple : (fichier, R) ; (imprimante, afficher)

Le modèle de protection doit fixer, à chaque instant, les droits d'accès dont dispose chaque sujet (processus). L'ensemble de ces droits pour un processus est appelé **domaine de protection du** processus (sujet).

Exemple : D1 = { (fichier f1 , ouvrir) , (fichier f1 , fermer) , (fichier f1 , lire) , (fichier f1 , écrire) , (éditeur , appeler) , (console 5 , lire) , (imprimante , afficher) }

Protection et sécurité

Domaine de protection

Un domaine de protection peut se représenter par une matrice d'accès.

Objets

Fichier 1 segment 1 segment 2 imprimante

Sujets P1

	O1	O2	O3	O5
D1	Lecture		lecture	
D2				Impression
D3		lecture	Exécution	
D4	Lecture écriture		Lecture écriture	

Exemple : chacl - change the access control list of a file or directory

Protection et sécurité

Domaine de protection

Implémentation de la matrice de droits

- **Complète :** table globale en mémoire. Grande taille, matrice creuse.
- **Par colonne :** « ACL » (Access Control List). Pour chaque objet : liste de paires [domaines , droits d'accès (non vides)]. Exemple des fichiers UNIX
- **Par ligne :** « Capacités de domaines ». Pour chaque domaine (processus) : liste de paires objets / droits non vides.

Liaison processus / domaine

1. **Statique :** ensemble de ressources disponibles fixe. Le principe de nécessité d'accès requiert un mécanisme de modification des contenus de domaines.
2. **Dynamique :** requiert un mécanisme de « commutation de domaine » (pas nécessairement de modification).

Protection et sécurité

Domaine de protection

Domaine = Utilisateur : les objets auxquels on peut accéder dépendent de l'utilisateur qui y accède. Commutation de domaine au changement d'utilisateur.

Unix : Bit « setuid » (root) : indique un éventuel changement d'identité pour les accès (privilégiés).

Domaine = Processus : les objets auxquels on peut accéder dépendent du processus qui y accède. Commutation de domaine à la commutation de contexte.

Domaine = Procédure : les objets auxquels on peut accéder correspondent aux variables utilisées par la procédure. Commutation de domaine à chaque appel de procédure.

Protection et langages évolués.

Mécanismes de protection Java

- Le contrôle statique réalisé par le JDK est destiné surtout à détecter les erreurs de programmation.
- La protection mémoire repose sur une architecture logicielle à domaines. Lorsqu'une applet fait appel à un classe de l'API java, ceci est détecté par le chargeur de classes qui appelle le moniteur de sécurité de la JVM. En fonction des capacités du règlement de l'applet l'appel est autorisé ou interdit.
- Java peut également utiliser un mécanisme permettant de signer une applet lors de sa création ou se son installation sur le serveur. La signature, basée sur un certificat, est contrôlée lors du téléchargement.

Protection et langages évolués.

Limites de la protection Java

- Erreur de programmation JVM et navigateurs
- Pas de protection matérielle
- Règlements de sécurités adaptés en environnement ouvert

Protection et sécurité

Exemple de systèmes de protections

Windows NT/2000/XP (1)

Utilisateurs: Un utilisateur est associé à compte dont les principaux attributs sont un non unique, un identifiant (SID: Security Identificator), un authentifiant qui est un mot de passe avec une stratégie de gestion de cet authentifiant, les groupes auquel appartient l'utilisateur, les droits et permissions qui lui sont propre.

Droits: Un droit est l'autorisation pour un utilisateur ou un groupe d'utilisateur d'exécuter une opération donnée sur une classe de ressources particulières du système. Les droits existant dans NT sont prédéfinis.

Permissions: Une permission est une règle associée à des ressources particulières (fichiers, répertoires, imprimantes) définissant quels ou quels groupes d'utilisateurs ont accès aux ressources et de quelle manière. Les droits passent toujours les permissions.

Protection et sécurité

Exemple de systèmes de protections

Windows NT/2000/XP (1)

Listes de contrôle d'accès : Dans NT la plus grande partie des droits et permissions sont stockés dans l'environnement des ressources protégées La liste des droits associés à une ressource est une ACL.

Groupes : NT permet de créer facilement des groupes d'utilisateurs ayant certains droits, les groupes peuvent être hiérarchisés et un compte utilisateur peut appartenir à plusieurs groupes.

Trois types de groupes: les groupes locaux , les groupes globaux, les groupes spéciaux. Un groupe local est constitué à partir de comptes utilisateurs et de groupes globaux. Un groupe global n'est constitué qu'à partir de comptes utilisateurs membre du domaine de définition.

Groupes locaux prédéfinis :

- Le groupe *opérateur de compte*.
- Le groupe des *administrateurs*.
- Le groupe des *opérateurs de sauvegarde*
- Le groupe *invité*.
- Le groupe utilisateur.

Protection et sécurité

Notion de sécurité

Protection , Sécurité

• Protection : problème strictement interne. Fournir un accès contrôlé aux ressources informatiques.

• Sécurité : problème externe. Requiert la prise en compte de l'environnement du système.

=⇒ S'assurer que le système est utilisé comme il a été prévu qu'il le soit.

Protection et sécurité

Notion de sécurité

Authentification (Nom d'utilisation et mot de passe)

L'utilisateur est-il celui qu'il prétend être ?

- **Possession utilisateur** : clé, carte d'accès. . .
- **Connaissance utilisateur** : identificateur, mot de passe. . .
- **Attribut utilisateur** : empreinte digitale, rétinienne, signature. . .

Protection et sécurité

Notion de sécurité

- **Faiblesses du mot de passe**
- **Exposition intentionnelle :** Transfert à un tiers, *etc.*
- **Exposition accidentelle :** Surveillance visuelle, vidéo, informatique
- **Découverte par test :** connaissance de l'utilisateur (mots de passe trop évidents : 80%), force brute (dictionnaires)
- **Mesures préventives**
  - Interdire les mots de passe trop simples
  - Changer les mots de passe à intervalles réguliers
  - Encrypter les transmissions
  - Stocker et cacher les formes encryptées
  -

Protection et sécurité

Menaces contre les programmes

1. Fonctionnement des virus : reproduction + modification

Le virus est un programme (ou une suite d'instructions cachées) qui se reproduit à partir de son code. Il entraîne souvent des troubles de fonctionnement, des pannes majeures, et contaminent d'autres systèmes informatiques.

Un virus est dangereux uniquement lorsqu'il réussit à s'exécuter sur un disque dur.

2. Familles de virus

Virus de boot (sous DOS) *:* utilisent les zones d'exécution automatiques des disques durs (ex : secteur de démarrage) pour se propager.

Virus exécutables (sous Windows) : ils insèrent leur propre code à l'intérieur des fichiers programmes. Leur reproduction consiste à rechercher d'autres programmes à infecter lorsque le fichier infecté est exécuté.

Protection et sécurité

Menaces contre les programmes

2. Familles de virus

Macrovirus : développés avec le langage VBA (*Visual Basic for Applications)*.

Utilisent le langage de programmation d'un logiciel pour en altérer le fonctionnement.

Vers (=worms) : ce sont des programmes qui se propagent en utilisant des réseaux (P2P, mail, ...). Ils n'ont pas besoin de programme hôte pour se reproduire.

Ils se servent de différentes sources d'information mises à leur disposition pour « trouver » et « infiltrer » comme les failles humaines (ouverture de fichiers attachés), les failles logicielles (carnet d'adresses , ...).

Leur action est surtout de se reproduire pour saturer, détruire, offrir un point d'accès caché (ex : envoi d'un script par mail). Souvent il n'y a qu'une seule copie sur le disque de l'hôte, donc il suffit de le détruire (si on le trouve) pour que le système ne soit plus infecté.

Protection et sécurité

Menaces contre les programmes

2. Familles de virus

Cheval de troie : ce n'est pas vraiment un virus. Il est introduit intentionnellement par une personne (pirate) dans un système en vue de détruire ou récupérer des données confidentielles (entreprises, personnelles, ...), c'est un logiciel d'apparence légitime. Il ne se reproduit pas et cible la machine dont il prend le contrôle.

Logiciel espion (=spyware) : c'est un logiciel malveillant qui s'installe dans un ordinateur pour collecter des données.

Composeur d'attaque : c'est un logiciel permettant de brancher un ordinateur à un autre, au réseau (souvent il faut composer un numéro de téléphone pour brancher l'ordinateur au réseau). Ce sont des logiciels qui sont souvent légaux (souvent inclus dans le système d'exploitation), mais parfois on peut en trouver des malveillants.

Protection et sécurité

Protection des données : cryptographie ou chiffrement

Le chiffrement, ou cryptage est le procédé grâce auquel on souhaite rendre la compréhension d'un document impossible à toute personne qui n'a pas la clé de déchiffrement.

La sécurité d'un système de chiffrement repose plus sur le secret de la clé de chiffrement que sur l'algorithme lui-même.

Un système de chiffrement est dit :

- **symétrique :** quand il utilise la même clé pour chiffrer et déchiffrer.

- **asymétrique** *:* quand il utilise des clés différentes : une paire composée d'une clé publique, servant au chiffrement, et d'une clé privée, servant à déchiffrer. Le point fondamental soutenant cette décomposition publique/privée est l'impossibilité calculatoire de déduire la clé privée de la clé publique.

Protection et sécurité

Protection des données : cryptographie ou chiffrement

Les méthodes les plus connues sont le DES, le Triple DES et l'AES pour la cryptographie symétrique, et le RSA pour la cryptographie asymétrique, aussi appelée cryptographie à clé publique.

L'utilisation d'un système symétrique ou asymétrique dépend des tâches à accomplir. La cryptographie asymétrique présente deux intérêts majeurs : elle supprime le problème de transmission sécurisée de la clé, et elle permet la signature électronique. Elle ne remplace cependant pas les systèmes symétriques car ses temps de calcul sont nettement plus longs.

Université de Bouira
Faculté des Sciences et des Sciences appliquées
Département d'Informatique
Licence SI (Systèmes Informatiques)
Semestre 3

Systèmes d'exploitation 2

A. ABBAS
abbasakli@gmail.com

Chapitre 2:

Synchronisation

CONTENU DU COURS

Synchronisation

- Problème de l'exclusion mutuelle
- Synchronisation
  - Evénements, Verrous
  - Sémaphores
  - Moniteurs
  - Régions critiques.
  - Expressions de chemins

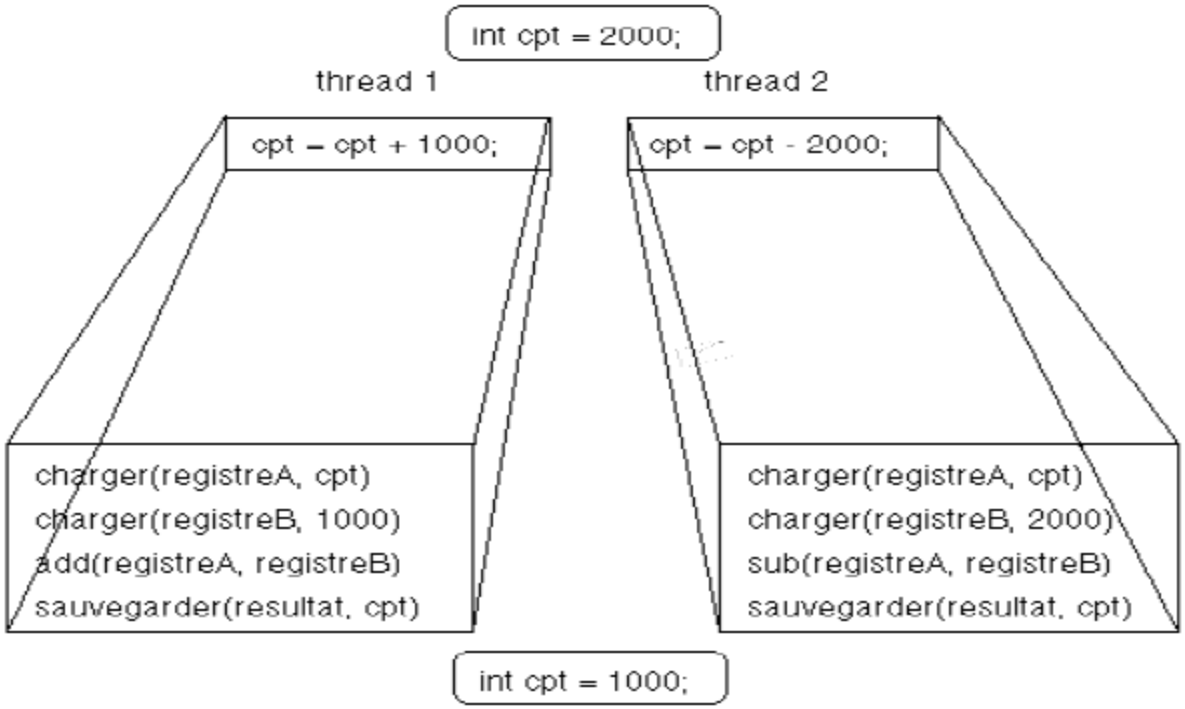
Problématique

Lorsque plusieurs processus (ou threads) qui s'exécutent sur processeur sont amenés a partager des ressources comme : les périphériques d'entrées-sorties (écrans, imprimantes,..), les moyens de mémorisation (mémoire centrale,...) ou des moyens logiciels (fichiers, base de données, ...) soit **volontairement** s'ils coopèrent pour traiter un même problème, soit **involontairement** parce qu'ils sont obliges de se partager ces ressources vue leurs nombre limite.

Dans ce cas, les processus vont se trouver en situation de concurrence d'accès vis-à-vis de ces ressources offertes par le système d'exploitation.

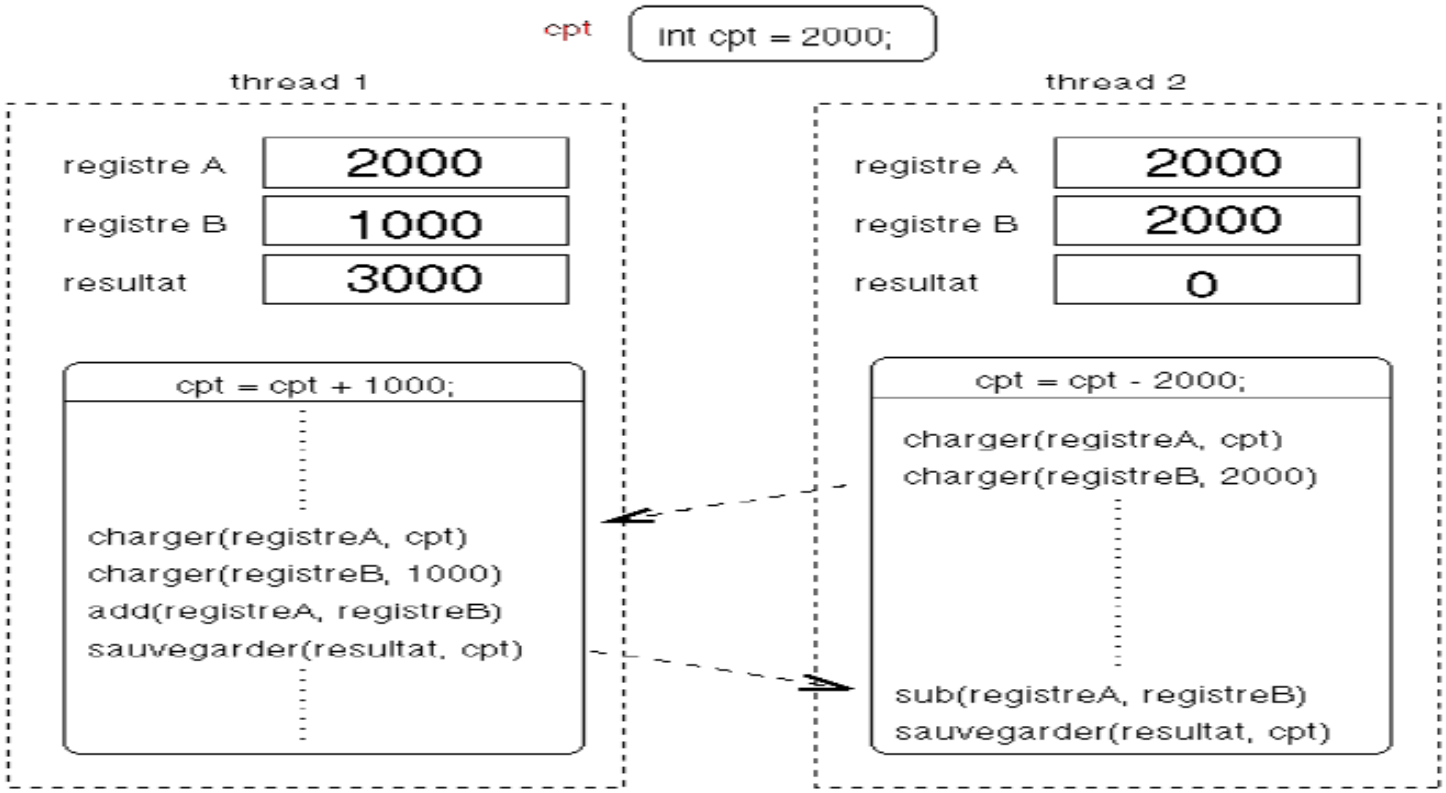
Malheureusement, le partage des ressources sans précaution particulière peut conduire à des résultats imprévisibles. L'état final des données dépend de l'ordonnancement des processus.

Problématique



La mise a jour concurrente des données peut se dérouler sans problème, où la variable cpt, initialisée à 2000, aura comme valeur finale 1000 (2000 + 1000 - 2000).

Problématique



Malheureusement, la mise a jour concurrente peut générer des incohérences, où la variable cpt, aura comme valeur finale 0 après l'exécution du même code. On voit que l'exécution de plusieurs threads peut conduire a des résultats incohérents.

La cause de cette incohérences est due a l'utilisation simultanée de la ressource partagée (la zones mémoire de la variable cpt) par plusieurs programmes.

Synchronisation

1- Introduction

Lorsque plusieurs processus s'exécutent dans un système d'exploitation multitâches, en pseudo-parallèle ou en parallèle et en temps partage, ils partagent des ressources (mémoires, imprimantes, etc.). Cependant, le partage d'objets sans précaution particulière peut conduire a des résultats incohérents. La solution a ce problème s'appelle synchronisation des processus.

Synchronisation

2- Définitions

1. **Ressources critiques:**

a. Les ressources partagée, comme des zones mémoire, cartes d'entrées/sorties, etc., sont dites critiques si elles ne peuvent être utilisées **simultanément** que par un seul processus.

b. On appelle **ressource critique** tout objet : variable, table, chier, périphérique, ... qui peut faire l'objet d'un accès concurrent (ou simultané) par plusieurs processus.

2. **Section critique**

a. Une section critique (SC) est un ensemble d'instruction d'un programme qui peuvent engendrer des résultats imprévisibles (ou incohérents) lorsqu'elles sont exécutées simultanément par des processus différents.

b. Une SC est une suite d'instructions qui opèrent sur un ou plusieurs ressources partagées (critiques) et qui nécessitent une utilisation **exclusive** de ces ressources.

2- Définitions

3. Exclusion mutuelle

Les problème d'incohérences des résultats, posé par les accès concurrents, montrent que la solution consiste à exécuter les sections critiques en exclusion mutuelle. C'est à dire qu'une section critique (SC) ne peut être entamée (ou exécutée) que si aucune autre SC du même ensemble n'est en exécution.

Le principe général d'une solution garantissant que l'exécution simultanée de plusieurs processus ne conduirait pas à des résultats imprévisibles est : avant d'exécuter une SC, un processus doit s'assurer qu'aucun autre processus n'est en train d'exécuter une SC du même ensemble. Dans le cas contraire, il ne devra pas avancer tant que l'autre processus n'aura pas termine sa SC.

2- Définitions

3. Exclusion mutuelle

Par exemple, avant d'entrer en SC, le processus doit exécuter un protocole d'entrée. Le but de ce protocole est de vérifier si la SC n'est occupée par aucun autre processus. A la sortie de la SC, le processus doit exécuter un protocole de sortie de la SC. Le but de ce protocole est d'avertir les autres processus en attente que la SC est devenue libre.

Le pseudo-code suivante résume ce fonctionnement :

```
Processus Pi
Début
...<Instructions hors de la section critique>
Protocole d'entrée en SC
|| SC
Protocole de sortie de SC
...<Instructions hors de la section critique>
Fin.
```

Synchronisation

2- Conditions nécessaires pour réaliser une exclusion mutuelle

Pour réaliser une exclusion mutuelle utile on admet que certaines conditions doivent être respectées :

1. **Le déroulement :** Le fait qu'un processus qui ne demande pas à entrer en section critique ne doit pas empêcher un autre processus d'y entrer. En plus, aucun processus suspendu en dehors de sa section critique ne doit bloquer les autres processus.

2. **L'attente infinie :** Si plusieurs processus sont en compétition pour entrer en SC, le choix de l'un d'eux ne doit pas être repoussé indéfiniment. Autrement dit, la solution proposée doit garantir que tout processus n'attend pas indéfiniment.

3. Deux processus ne peuvent être en même temps dans leurs sections critiques.

4. Tous les processus doivent être égaux vis a vis de l'entrée en SC.

3- Solutions pour réaliser une exclusion mutuelle

1- Masquage des interruptions

Avant d'entrer dans une section critique, le processus masque les interruptions.

Il les restaure (non masque) a la fin de la section critique.

Il ne peut être suspendu alors durant l'exécution de la section critique.

Exemple : Masque des interruptions

```
Local_irq_disable(); /* Les interruptions sont masquées .. */
Section critique ();
Local_irq_enable();
```

Problèmes:

1- Si le processus ne restaure pas les interruptions a la sortie de la section critique, ce serait le bug du système.

2- Elle assure l'exclusion mutuelle, si le système est monoprocesseur puisque le masquage des interruptions concerne le processeur qui a demandé ce masquage. Les autres processus exécutés par un autre processeur pourront donc accéder aux ressources partagées.

3- Solutions pour réaliser une exclusion mutuelle

2- Attente active

Utiliser une variable de verrouillage partagée ***verrou***, unique, initialisée a 0. Pour rentrer en section critique, un processus doit tester la valeur de ***verrou***

- Si elle est égale à 0, le processus modifie la valeur du verrou ←1 et exécute sa section critique. A la fin de la section critique, il remet le verrou a 0.
- Sinon, il attend (par une attente active) que le verrou devienne egal à 0, c.-a-d. : while(verrou ≠0);

```
while (verrou ≠ 0) do
; // Attente active
end while
verrou ⟵ 1;
Section_critique() ;
verrou ⟵ 0;
```

3- Solutions pour réaliser une exclusion mutuelle

2- Attente active

Problèmes: Cette méthode n'assure pas l'exclusion mutuelle

- Si un processus P1 est suspendu juste après avoir lu la valeur du verrou qui est égal à 0 (par exemple: mov verrou).
- Ensuite, si un autre processus P2 est élu et il teste le verrou qui est toujours égal a 0, met verrou ←1 et entre dans sa section critique.
- Si P2 est suspendu avant de quitter la section critique et que P1 est réactivé et entre dans sa section critique et met le verrou←1

Résultats: Les deux processus P1 et P2 sont en même temps en section critique.

Susceptible de consommer du temps en bouclant inutilement.

3- Solutions pour réaliser une exclusion mutuelle

3 - Solution de Peterson

- Cette solution se base sur deux fonctions **entrer_region** et **quitter_region**.

- Chaque processus doit, avant d'entrer dans sa section critique, appeler la fonction **entrer_region** en lui fournissant en paramètre son numéro.

- Cet appel le fera attendre si nécessaire jusqu'a ce qu'il n'y ait plus de risque.

- A la fin de la section critique, il doit appeler **quitter_region** pour indiquer qu'il quitte sa section critique et pour autoriser l'accès aux autres processus.

Processus P_i

```
while (1){
/*attente active*/
entrer_region(i) ;
section_critique_Pi() ;
quitter_region(i);
...}
```

3- Solutions pour réaliser une exclusion mutuelle

3 - Solution de Peterson

Le pseudo code des deux fonction est donné ci-dessous:

```
void entrer_region(int process)
{
int autre;
if(process==1)
autre = 2;
else
autre=1; //l'autre processus
interesse[process] = TRUE; //indiquer qu'on est intéressé
tour = process; //la course pour entrer se gagne ici
while (tour == process and interesse [autre] == TRUE);
}
```

```
void quitter_region (int process )
{// Processus quitte la région critique
interesse[process] = FALSE;
}
```

Note: Les variables *tour* et *interesse* sont globales.

- La généralisation de cette solutions aux cas de plusieurs processus est bien complexe.
- Susceptible de consommer du temps en bouclant inutilement.

3- Solutions pour réaliser une exclusion mutuelle

4 - Sémaphore

Pour contrôler les accès a un objet partage, Dijkstra (en 1965) a suggéré l'utilisation d'un nouveau type de variables appelées les sémaphores.

Définition : Un sémaphore S est un ensemble de deux variables :

1. Une valeur (ou compteur) entier notée value désigne le nombre d'autorisations d'accès à une section critique. Cette valeur est manipulable au moyen des opérations P (ou wait) et V (ou signal);
2. Une file F des processus en attente.

3- Solutions pour réaliser une exclusion mutuelle

4 - Sémaphore

L'opération P(S) décrémente la valeur du sémaphore S. Puis, si cette dernière est inferieure à 0 alors le processus appelant est mis en attente. Sinon le processus appelant accède a la section critique.

```
P(S)
{
        S.value = S.value - 1;
        if (S.value < 0)
          {
            état(Pro i) = bloqué   ; // Pro i = processus appelant
            mettre Pro i dans S.F   ;
          }
}
```

3- Solutions pour réaliser une exclusion mutuelle

4 - Sémaphore

L'opération V(S) incrémente la valeur du sémaphore S. Puis si cette valeur est supérieure ou égale à 0 alors l'un des processus bloqués par l'opération P(S) sera choisi et redeviendra prêt.

```
V(S)
{
        S.value = S.value + 1;
        if (S.value ≥ 0)
        {
            < sortir un processus Pro j de S.F >;
            état(Pro j) = prêt
        }
}
```

3- Solutions pour réaliser une exclusion mutuelle

4 - Sémaphore

L'existence d'un mécanisme de file d'attente (FIFO ou LIFO), permettant de mémoriser le nombre et les demandes d'opération P(S) non satisfaites et de réveiller les processus en attente.

Remarque 1 : Le test du sémaphore, le changement de sa valeur et la mise en attente éventuelle sont effectués en une seule opération atomique indivisible.

Remarque 2 : La valeur initiale du champ value d'un sémaphore doit être un nombre non négatif. La valeur initiale d'un sémaphore est le nombre d'unités de ressource.

3- Solutions pour réaliser une exclusion mutuelle

4 - Sémaphore

Ainsi, on peut proposer un schéma de synchronisation de n processus voulant entrer simultanément en SC, en utilisant les deux opérations P(S) et V(S).

En effet, il suffit de faire partager les ***n*** processus un sémaphore S , initialise a 1, appelé sémaphore d'exclusion mutuelle.

Chaque processus Pi a la structure suivante :

```
Processus Pi
{
        ...
      P(S) ;

        || Sc();

      V(S);
        ...
}
```

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 1

Considérons deux processus P1 et P2 exécutent respectivement deux instructions

S1 = A+2 B; et S2 = S1+10.

Si nous souhaitons que S2 ne doit s'exécuter qu'après l'exécution de S1, nous pouvons implémenter ce schéma en faisant partager P1 et P2 un sémaphore commun S, initialise a 0 et en insérant les primitives P(S) et V(S) de la façon suivante:

Processus P1	Processus P2
S1; V(S);	P(S); S2;

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 2: Lecteurs / Rédacteurs

Considérons un objet (une base de donnée par exemple) qui n'est accessible que par deux catégories d'opérations : les lectures et les écritures. Plusieurs lectures (consultations) peuvent avoir lieu simultanément ; par contre les écritures (mises a jour) doivent se faire en exclusion mutuelle.

On appellera *lecteur* un processus faisant des lectures et *rédacteur* un processus faisant des écritures.

Il s'agit donc de réaliser la synchronisation entre lecteurs et rédacteurs en respectant les contraintes suivantes :

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 2: Lecteurs / Rédacteurs

Exclusion mutuelle entre lecteurs et rédacteurs : si un lecteur demande a lire et qu'il y a une écriture en cours , la demande est mise en attente. De même que si un rédacteur demande à écrire et qu'il y a au moins une lecture en cours , la demande est mise en attente.

Exclusion mutuelle entre rédacteurs : si un rédacteur demande a écrire et qu'il y a une écriture en cours , la demande est mise en attente.

Pour satisfaire ces contraintes ci-dessus , on peut procéder comme suit :

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 2: Lecteurs / Rédacteurs

```
var S1,S2 : sémaphore init 1,1 ;
nl : entier init 0 ;
```

Processus Lecteur	Processus Rédacteur
Début Début	Début
.	
.	
.	
P(S2)	P(S1)
nl := nl + 1	Ecriture
si nl=1 alors P(S1) finsi	V(S1)
V(S2)	.
Lecture	.
P(S2)	.
nl := nl – 1	Fin
si nl=0 alors V(S1) finsi	
V(S2)	
.	
.	
.	
Fin	

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Services Posix sur les semaphores

Le système d'exploitation Linux permet de créer et d'utiliser les sémaphores définis par le standard Posix.

Les services Posix de manipulation des sémaphores se trouvent dans la librairie <semaphore.h>.

Le type sémaphore est désigné par le type sem_t.

Initialisation d'un sémaphore

int sem_init(sem_t *sem, int pshared, unsigned int valeur)

- ***sem* :** est un pointeur sur le sémaphore à initialiser;
- **value :** est la valeur initiale du sémaphore ;
- **pshared :** indique si le sémaphore est local au processus (pshared=0) ou partage entre le père et le fils pshared ≠0.

Synchronisation

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Services Posix sur les sémaphores

int sem_wait(sem_t *sem) : est équivalente à l'opération P (S) et retourne toujours 0.

int sem_post(sem t *sem): est équivalente a l'opération V(S) : retourne 0 en cas de succès ou -1 autrement.

int sem trywait(sem_t *sem) : décrémente la valeur du sémaphore sem si sa valeur est supérieure à 0 ; sinon elle retourne une erreur. C'est une opération atomique.

int sem_getvalue (sem_t * sem, int * sval): récupérer la valeur d'un sémaphore : il retourne toujours 0.

int sem destroy (sem t* sem): détruire un sémaphore. Retourne -1 s'il y a encore des

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 3: le problème du producteur-consommateur

Le problème de Producteur-Consommateur est un problème de synchronisation classique qui permet de représenter une classe de situations où un processus, appelé **Producteur**, délivre des messages (informations) à un processus **Consommateur** dans un tampon (par exemple, un programme qui envoie des données sur le spool de l'imprimante).

Le **Producteur** produit un message dans la *ZoneP*, puis le dépose dans le buffer. Le **Consommateur** prélève un message du Buffer et le place dans la ZoneC où il peut le consommer.

On considérera que le buffer est de N cases, de 0 à N-1, et organisé de façon circulaire. Le Producteur dépose les messages par un bout du buffer alors que le consommateur les consomme au fur et à mesure par l'autre bout.

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 3: le problème du producteur-consommateur

n producteurs produisent des messages et les déposent dans un tampon de taille illimitée. n consommateurs peuvent récupérer les messages déposés dans le tampon.

```
Semaphore Mutex = 1 ;
Message tampon[ ];
Producteur ( ){
Message m ;
Tantque Vrai faire
        m = creermessage() ;
        Mutex.P() ;
                EcritureTampon(m);
        Mutex.V() ;
FinTantque
}
```

```
Consommateur( ){
Message m ;
Tantque Vrai faire
        Mutex.P() ;
                m = LectureTampon();
        Mutex.V() ;
Fin Tantque
}
```

Adaptez cette solution pour que l'échange des message soit synchronisé (consommateurs bloqués si le tampon est vide).

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore - Exemple 3: le problème du producteur-consommateur

```
Semaphore Mutex = 1, Plein = 0 ;
Message tampon[];
Producteur ( ){
Message m ;
Tantque Vrai faire
        m = creermessage() ;
        Mutex.P() ;
                EcritureTampon(m);
        Mutex.V() ;
        Plein.V();
FinTantque
}
```

```
Consommateur( )
{
Message m ;
Tantque Vrai faire
        Plein.P();
        Mutex.P() ;
                m = LectureTampon();
        Mutex.V() ;
Fin Tantque
}
```

3- Solutions pour réaliser une exclusion mutuelle

4 – Sémaphore -Critiques

Les programmes utilisant les sémaphores sont généralement difficile à réaliser et à "débuguer" car les erreurs, (oublie de V ou utiliser P à la place de V), sont dues à des conditions de concurrence, des inter-blocages ou d'autres formes de comportement imprévisibles et/ou difficiles à reproduire.

D'autre part, les sémaphores sont des objets globaux et doivent être connus de tous les participants. De plus, la synchronisation par les sémaphores nécessite l'étude de tout un programme concurrent pour comprendre l'aspect synchronisation qu'il renferme.

- Pour palier à ces points faibles, le concept de ***moniteur*** a vu le jour
- Ce concept a été implémenté dans des langages de programmation concurrente : C# ainsi que Java.

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs -Définition

L'idée des moniteurs est de regrouper dans un module spécial, appelé moniteur, toutes les sections critiques d'un même problème.

Un moniteur est un ensemble de **procédures**, de **variables** et de **structures de données**.

Les processus peuvent appeler (ou utiliser) les procédures du moniteur mais ils ne peuvent pas accéder aux variables et aux structures de données interne du moniteur à partir des procédures externes.

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs -Exclusion mutuelle par des moniteurs

Pour assurer l'accès aux ressources critiques en exclusion mutuelle, il suffit qu'il y ait à tout instant pas plus d'un processus actif dans le moniteur, c'est-a-dire que les procédures du moniteur ne peuvent être exécutées que par un seul processus a la fois. C'est le compilateur qui effectue de cette tâche (assurer l'accès aux ressources critiques).

Pour cela, le compilateur rajoute, au début de chaque procédure du moniteur un code qui réalise ce qui suit:

- S'il y a un processus P1 actif dans le moniteur, alors le processus P2 appelant est suspendu jusqu'a ce que le processus actif dans le moniteur se bloque en exécutant un **wait** sur une variable conditionnelle (wait(c)) ou quitte le moniteur.

- Le processus bloqué est réveillé par un autre processus en lui envoyant un signal sur la variable conditionnelle (signal(c)).

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs

Une variable de condition (des moniteurs) est une condition manipulée au moyen de deux opérations wait et signal :

wait(x) :

- suspend l'exécution du processus (thread) appelant (le met en attente de x);
- autorise un autre processus en attente du moniteur a y entrer.

signal(x) : débloqué un processus en attente de la condition x.

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs -Implémentation des moniteurs

L'utilisation de moniteurs est plus simple que les sémaphores puisque le programmeur n'a pas a se préoccuper de contrôler les accès aux sections critiques.

Mais, malheureusement, a l'exception de JAVA (avec le mot clé synchronized) et C#, la majorité des compilateurs utilisés actuellement ne supportent pas les moniteurs. Par contre, on peut simuler les moniteur avec d'autres langages à partir des mutex comme suit:

- Le moniteur contient un sémaphore binaire (par exemple: mutex)
- Toutes les procédures commencent par l'acquisition du mutex et finissent par sa libération.

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs –Exemple 1 : Producteur/consommateur

- Les sections critiques du problème du producteur et du consommateur sont les opérations de dépôt et de retrait dans le tampon partagé.
- Le consommateur est actif dans le moniteur et le tampon est vide => il devrait se mettre en attente et laisser place au producteur.
- Le producteur est actif dans le moniteur et le tampon est plein => il devrait se mettre en attente et laisser place au consommateur.

Processus producteur	Processus consommateur

{
int p ;
while (1){
Produire (p) ;
ProducteurConsommateur.mettre (p);
}
}
	{
	int c ;
	while (1){
	ProducteurConsommateur.retirer(c);
	Consommer (c);
	}
	}

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs –Exemple 1 : Producteur/consommateur

```
Moniteur ProducteurConsommateur (){
bool nplein, nvide ; //variable de condition pour non
plein et non vide
int compteur =0, ic=0, ip=0, N ;

void mettre (int objet) // section critique pour le dépôt
{
  if (compteur==N) wait(nplein) ; //attendre
                jusqu'a ce que le tampon soit non plein

                tampon[ip] = objet ;
                ip = (ip+1)%N ;
                compteur++ ;

                // si le tampon était vide, envoyer un
                signal pour réveiller le consommateur.

        if (compteur==1) signal(nvide) ;
}

void retirer (int* objet) //section critique
pour le retrait
{               if (compteur ==0) wait(nvide) ;
                objet = tampon[ic] ;
                ic = (ic+1)%N ;
                compteur - ;

// si le tampon était plein, envoyer un
signal pour réveiller le producteur.

if(compteur==N-1) signal(nplein) ;
}
}
```

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs - Règles d'utilisation des moniteurs

An d'éviter que tous les processus réveillés se retrouvent au même temps dans le moniteur, différentes règles ont été établies pour définir ce qui se passe a l'issue d'un signal.

1. **Règle de Hoare** : ne laisser entrer dans le moniteur que le processus qui a été suspendu le moins longtemps;

2. Regle de Brinch Hansen : exiger du processus qui fait **Signal** de sortir immédiatement du moniteur. Il laisse ainsi la place a tous ceux qui étaient en attente. L'ordonnanceur choisira un parmi ceux ci.

3. 2ème règle de Brinch Hansen : si un Signal est réalise sur une variable conditionnelle et qu'aucun processus ne l'attend, alors ce signal est perdu.

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs - Réalisation sous C/POSIX

Pour garantir qu'une seule procédure du moniteur soit activée à instant donne, il suffit de protéger l'exécution de toutes les procédures par le même sémaphore d'exclusion mutuelle.

Exemple Moniteur: probleme Producteur/consommateur

```
pthread_mutex_t mutex ;
pthread_cond_t est_vide, est_plein;
char *buffer;
void *mettre (char msg) {// section critique pour le dépôt
pthread_mutex_lock ( & mutex);
        while (buffer ! = NULL)
         pthread_cond _wait ( & est_vide, & mutex);
        // buer = NULL
        buffer = strdup(msg);
        Pthread_cond_signal ( & est_plein);
pthread_mutex _unlock ( & mutex);
}
```

3- Solutions pour réaliser une exclusion mutuelle

5 – **Les Moniteurs -** Réalisation sous C/POSIX

```
char *retirer () {//section critique pour le retrait
char *result;
    pthread_mutex_lock ( & mutex);
while (buffer == NULL)
            pthread_cond _wait ( & est_plein, & mutex);
    result = buer;
    buffer = NULL;
    pthread_cond_signal ( & est_vide);
    pthread_mutex_unlock ( & mutex);
    return result;
}
```

```
void main (void){
        pthread_t ta; pthread_t tb;
        pthread_mutex_init (&mutex, NULL);
        pthread_cond_init (& est_vide, NULL);
        Pthread_cond_init (& est_plein, NULL);
        buffer = NULL;
}
```

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs - Résumé des types et des fonctions utilisés

Types et fonctions	Description
pthread_t	Contexte de tâche
pthread_create()	Création d'une tâche
pthread_join()	Attente de fin de tâche
pthread_mutex_t	Sémaphore d'exclusion mutuelle
pthread_mutex_lock()	Début de zone critique
pthread_mutex_unlock()	Fin de zone critique
pthread_cond_t	Condition
pthread_cond_wait()	Mise en attente
pthread_cond_signal()	Déclenchement

3- Solutions pour réaliser une exclusion mutuelle

5 – Les Moniteurs - Réalisation avec JAVA

Le concept de moniteur est implémente dans la MVJ de la manière suivante :

- Les données du moniteur doivent être déclarées avec le mot clé **privates** pour que seules les méthodes du moniteur accèdent à ces données,

- Les méthodes (ou procédures d'entrée) du moniteur doivent être déclarées avec le mot clé **synchronized** pour qu'elles puissent s'exécuter en exclusion mutuelle,

- La classe **Object** fournit les méthodes *wait()* et *notify()* pour la synchronisation des threads.