Noyau d'un système d'exploitation
INF2610

Chapitre 10 : Gestion des périphériques d'E/S & Systèmes de fichiers

Département de génie informatique et génie logiciel

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Hiver 2014

Gestion des périphériques

- Introduction
- Structure en couches du logiciel d'E/S
- Interface utilisateur
- Interface matériel
- Interface périphérique / SE → E/S mappées ou non en mémoire
- Qui fait quoi ?
  - Les E/S programmées
  - Les E/S pilotées par les interruptions
  - Les E/S avec DMA
  - E/S synchrones ou asynchrones
- Disques

Introduction

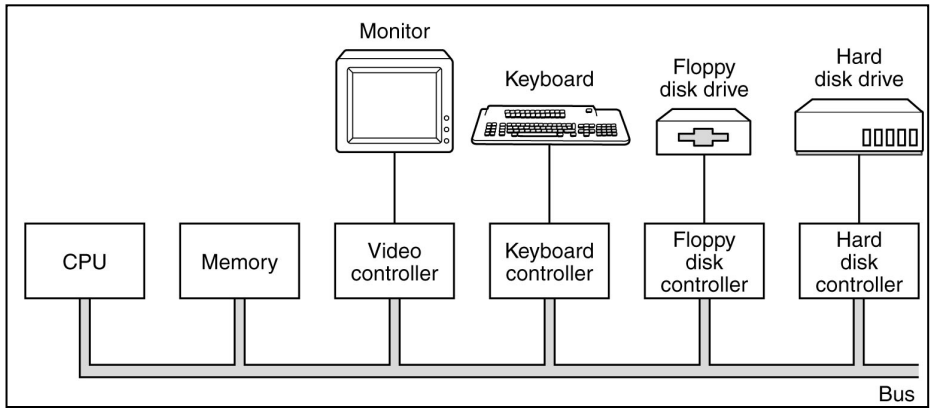
- Chaque ordinateur dispose de périphériques d'E/S tels que les terminaux, des disques, les imprimantes, les équipements de communication, etc.
- Les périphériques d'E/S permettent le stockage de données et les échanges d'information avec le monde extérieur.
- Ils ont des caractéristiques différentes.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

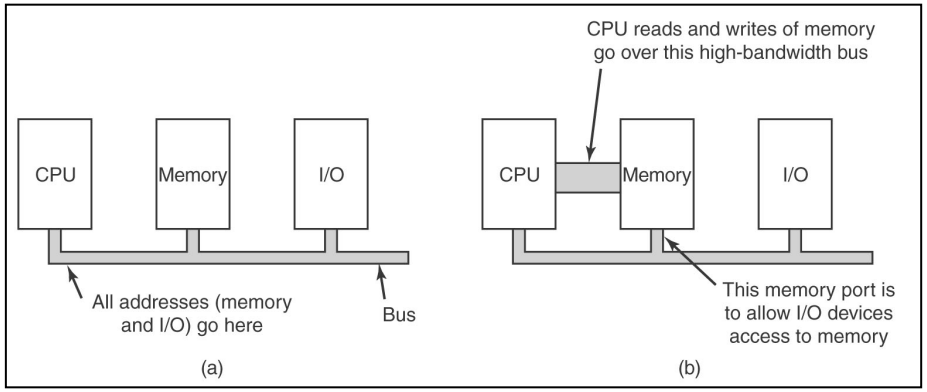
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec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

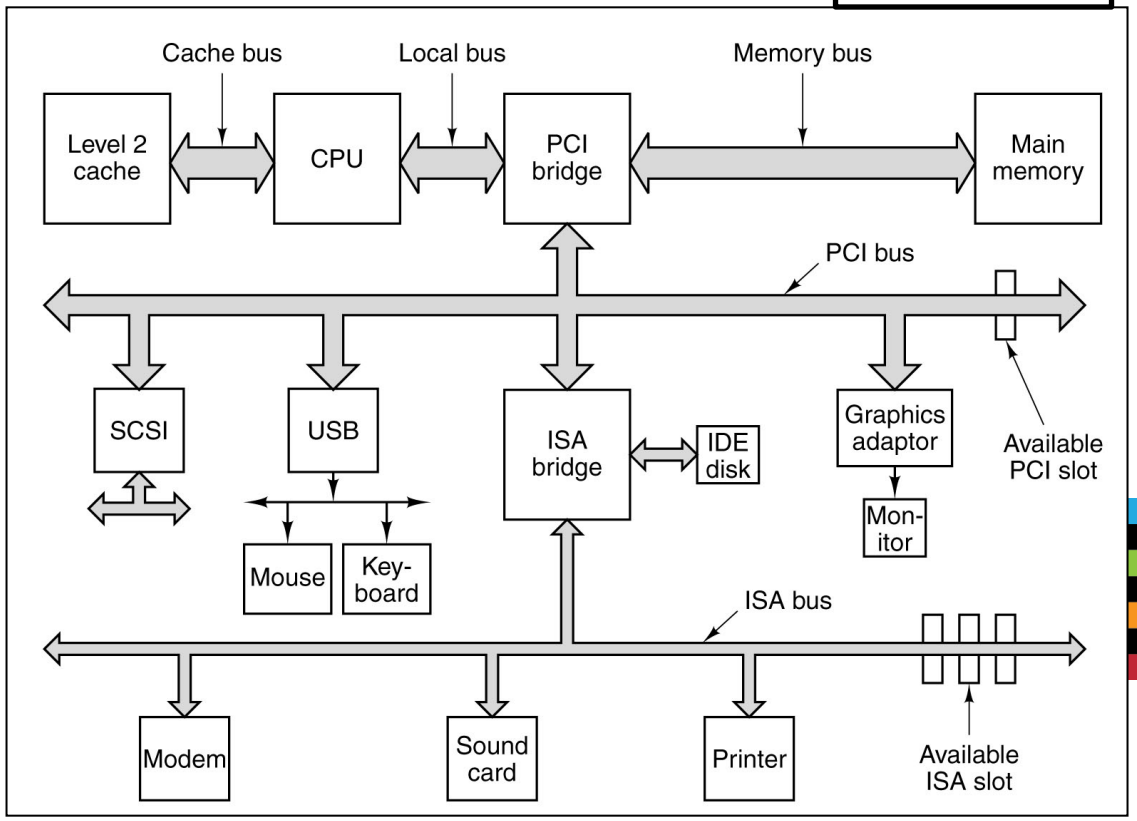
Introduction (2)

- Le système d'exploitation (SE) gère, contrôle et coordonne tous les composants de l'ordinateur (CPU, mémoire, bus, périphériques d'E/S).

- Ces composants sont connectés par un (ou plusieurs) bus et communiquent entre eux via ce(s) bus.



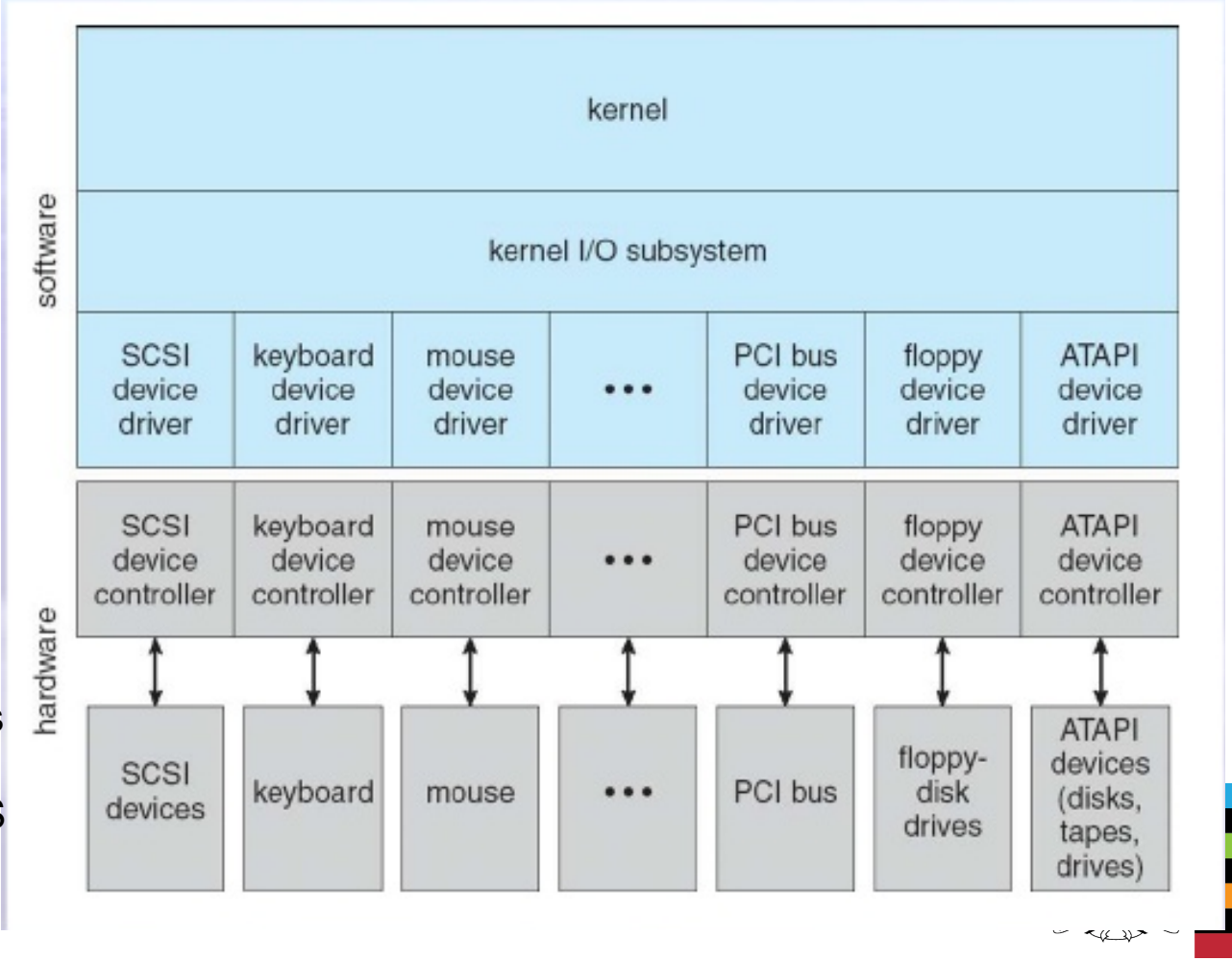




Introduction (3)

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

- Objectifs d'un gestionnaire de périphériques d'E/S :
  1. Offrir une interface complète, uniforme et simple d'emploi qui abstrait la complexité de fonctionnement des différents périphériques.
  2. Faciliter les références aux périphériques (désignation universelle qui passe à l'échelle).
  3. Être performant (maximiser les taux d'utilisation de tous les composants de l'ordinateur, optimiser les coûts des opérations d'E/S, …).
  4. Détecter et gérer les erreurs d'E/S (isoler l'impact des erreurs, recouvrir les erreurs, uniformiser les codes d'erreur, …).
  5. Permettre le partage des périphériques tout en assurant leur consistance et protection.

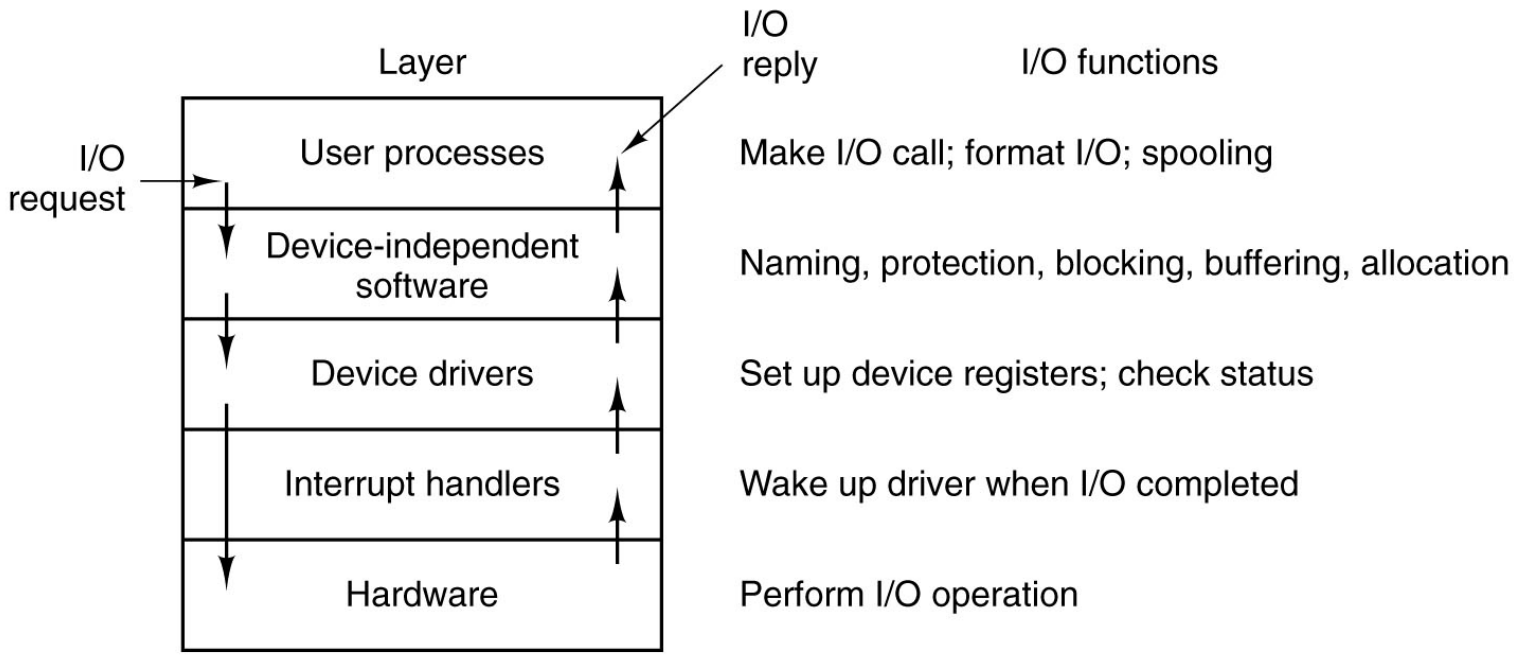


Structure en couches du logiciel d'E/S

- Le logiciel des E/S structuré en couches (ex. 4 couches) :

1. Le traitement des interruptions de fins des E/S.
2. Les pilotes de périphériques qui traitent les requêtes de plus haut niveau qui émanent du logiciel (indépendant du matériel).
3. Le logiciel du système d'exploitation indépendant des périphériques qui effectue les fonctions communes à tous les périphériques tout en fournissant une interface uniforme au logiciel des utilisateurs.
4. Le logiciel accessible au niveau de l'utilisateur (les appels système : mknod, open, read, write, aio_read, aio_write, etc.).

Chaque périphérique est identifié par un numéro majeur (qui set à localiser son pilote) et un numéro mineur (qui indique l'unité à lire ou dans laquelle écrire (ex. une partition)).



Interface utilisateur

- Les systèmes comme UNIX /Linux et Windows considèrent les périphériques d'E/S comme des fichiers spéciaux (objets) et intègrent la gestion des périphériques dans le système de fichiers.

- A chaque périphérique d'E/S est affecté un chemin d'accès, le plus souvent à partir du répertoire
  /dev.

- Ces fichiers spéciaux sont accessibles de la même façon que les autres fichiers (read, write..).

Exemple 1 :

/dev/lp
/dev/tty
/dev/net

```
echo   ABCDEF > /dev/tty
```

➔ L'interface utilisateur des périphériques est la même que celle des fichiers ordinaires.

Interface matériel : Composants d'un périphérique

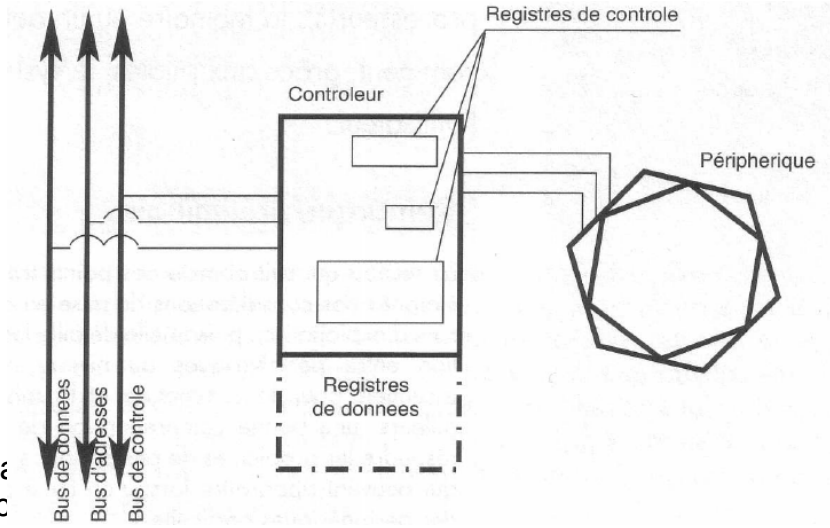
- Chaque périphérique d'E/S est généralement constitué de :
  - un composant mécanique (le périphérique lui-même) et
  - un composant électronique appelé contrôleur de périphérique ou adaptateur (ex. carte avec circuit imprimé).

- La carte du contrôleur est généralement équipée d'un connecteur sur lequel est branché un câble connecté au périphérique. Le contrôleur peut gérer plusieurs périphériques identiques.

- Le contrôleur a la charge de commander physiquement le périphérique, en réaction aux commandes envoyées par le SE (pilote de périphérique – device driver). Le fabriquant de contrôleur fournit un pilote pour chaque SE supporté.

- Chaque contrôleur possède un petit nombre de registres utilisables pour communiquer avec lui. En écrivant dans ces registres, le SE (via le pilote du périphérique) active le contrôleur pour réaliser une tâche. En lisant, ces registres, le SE peut connaître l'état du périphérique (prêt, occupé, erreur, etc.).

- En plus des registres de contrôle, de nombreux périphériques sont équipés d'un tampon de données que le SE peut lire et écrire.

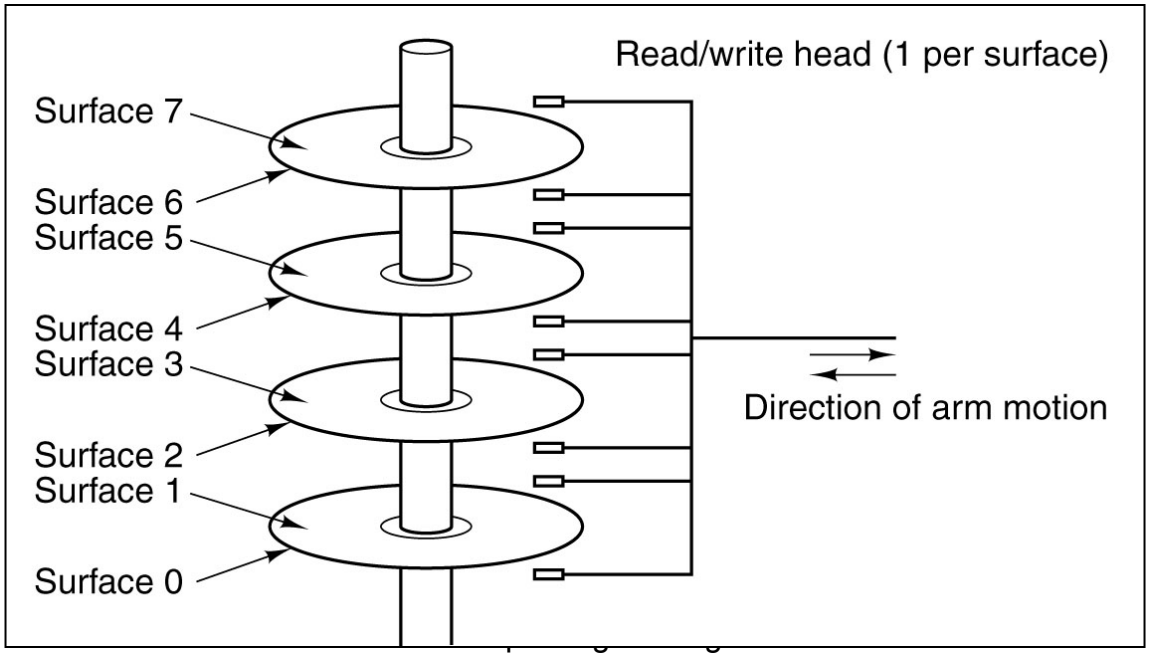


Interface matériel : Composants d'un périphérique (2)

Exemple 2 :

- Un contrôleur disque minimal comprendrait des registres pour renseigner : l'adresse disque, l'adresse mémoire, le numéro de secteur, le type d'opération (lecture ou écriture) et son état.

- Si un contrôleur disque reçoit une commande pour lire le secteur 11206 du disque 2, il doit alors :
  - convertir ce numéro linéaire de secteur en un triplet cylindre/secteur/tête selon la géométrie du disque,
  - déterminer sur quel cylindre se trouve le bras du disque pour lui indiquer de se déplacer pour se positionner sur le bon cylindre,
  - positionner la tête de lecture sur le bon secteur….



Interface système d'exploitation (SE) : Comment communiquer avec le SE ?

Trois solutions possibles :

1. **Pas de mappage en mémoire :**

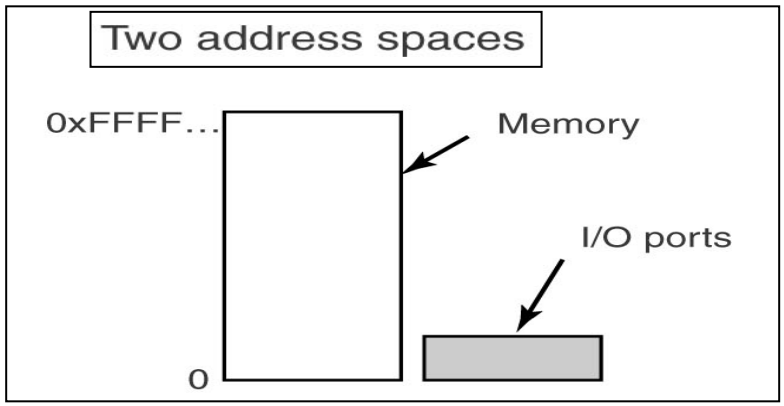
- Chaque registre de contrôle du périphérique se voit assigner un numéro de port d'E/S unique, correspondant à une adresse qui n'est pas dans l'espace d'adressage du SE.
- L'ensemble de ces registres constitue l'espace des ports d'E/S.
- Des instructions d'E/S spéciales sont utilisées pour lire/écrire dans un registre, comme :

  IN REG PORT // CPU récupère, dans son registre REG, le contenu du registre de contrôle PORT

  OUT PORT REG. // CPU copie le contenu de son registre REG dans le registre de contrôle PORT

- La majorité des ordinateurs récents utilisent cette méthode (économise la mémoire mais nécessite des instructions particulières à coder notamment en assembleur).
- Le test en boucle de l'état d'un périphérique (lecture et test d'un registre de contrôle) :

```
LOOP :    TEST PORT_4
          BEQ READY
          BRANCH LOOP
READY :
```



cat /proc/ioports

Interface système d'exploitation (SE) : Comment communiquer avec le SE ? (2)

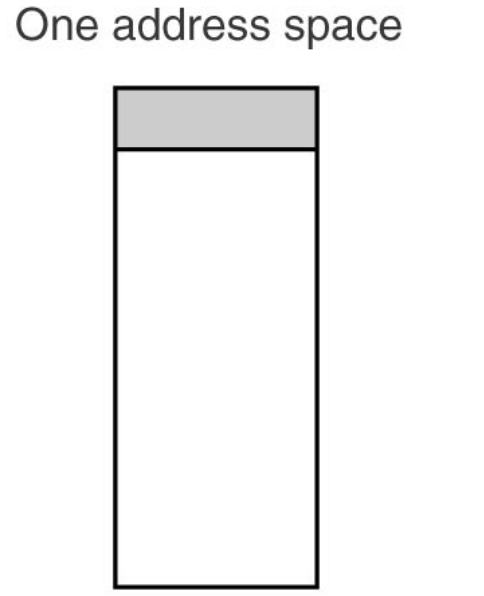
2. Mappage en mémoire :

cat /proc/iomem

•Les registres de contrôle et les tampons de données du périphérique sont mappées dans l'espace mémoire du noyau.

•Chaque registre de contrôle se voit attribuer une adresse mémoire unique (en général au sommet de l'espace d'adressage) → E/S mappées en mémoire.

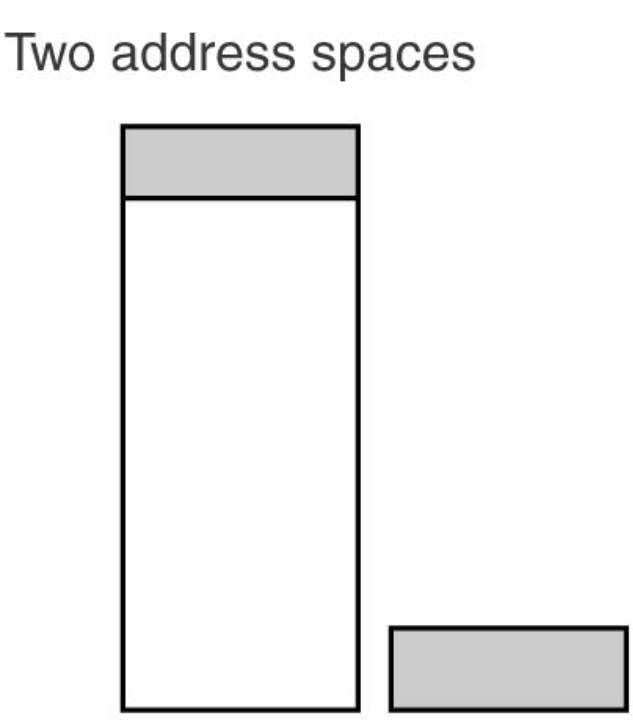
•On a pas besoin d'instructions spécifiques pour lire et écrire dans les registres de contrôle mais on a une consommation d'espace d'adressage.



3. Hybride :

•Les tampons de données sont mappées en mémoire mais pas les registres de contrôle.

•Pentium exploite cette architecture (dans IBM PC et compatibles, les adresses de 640 KO à 1 MO sont réservées aux tampons des données des périphériques).



Interface système f'exploitation (SE) : Comment communiquer avec le SE ? (4)

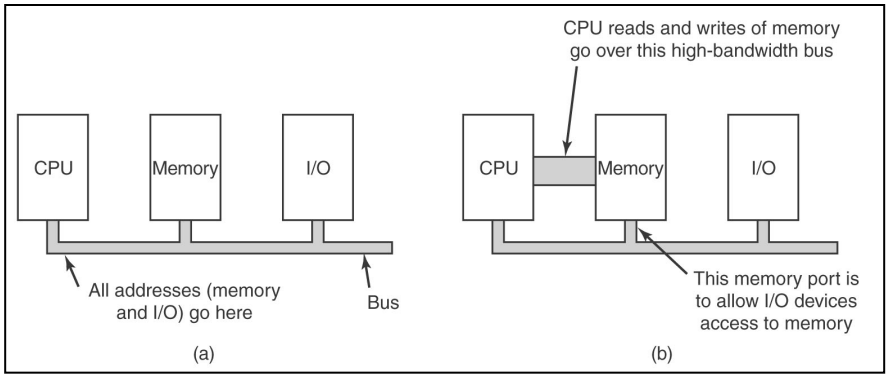
Mappage vs. non mappage en mémoire

•On a pas besoin d'instructions supplémentaires pour accéder aux ports d'E/S.

•SE peut permettre aux usagers d'accéder au périphérique (au besoin).

•La zone d'adresse réservée aux ports d'E/S ne doit pas être interceptée par la mémoire cache.

```
LOOP :    TEST PORT_4
          BEQ READY
          BRANCH LOOP
READY :
```



•Si l'ordinateur ne possède qu'un bus, la consultation de chaque adresse par tous est simple. Sinon, une solution serait de disposer de filtres d'adresses.

•De plus en plus, on retrouve des instructions dédiées et un circuit de gestion des E/S (IOMMU) afin de contrôler les accès aux ports d'E/S (conversion d'adresses virtuelles en adresses physiques).

➔Dans tous les cas (E/S mappées ou non), la CPU doit échanger des données (blocs ou flots de caractères) avec les périphériques. Comment procéder (qui fait quoi) ?

Qui fait quoi ? Les E/S programmées

- La CPU s'occupe de toute l'opération d'E/S (**attente active) :**

1. Faire un appel système que le noyau traduit en un appel à une procédure du pilote approprié.
2. Le pilote se charge de démarrer l'E/S quand le périphérique est prêt (transfert octet par octet ou bloc par bloc).
3. Quand celle-ci se termine, le pilote range le cas échéant les informations là où elles sont attendues puis termine en rendant le contrôle à l'appelant.

- **Exemple 3 : E/S mappées en mémoire**

Imprimante (100c/s) => CPU en attente active pendant 10 ms

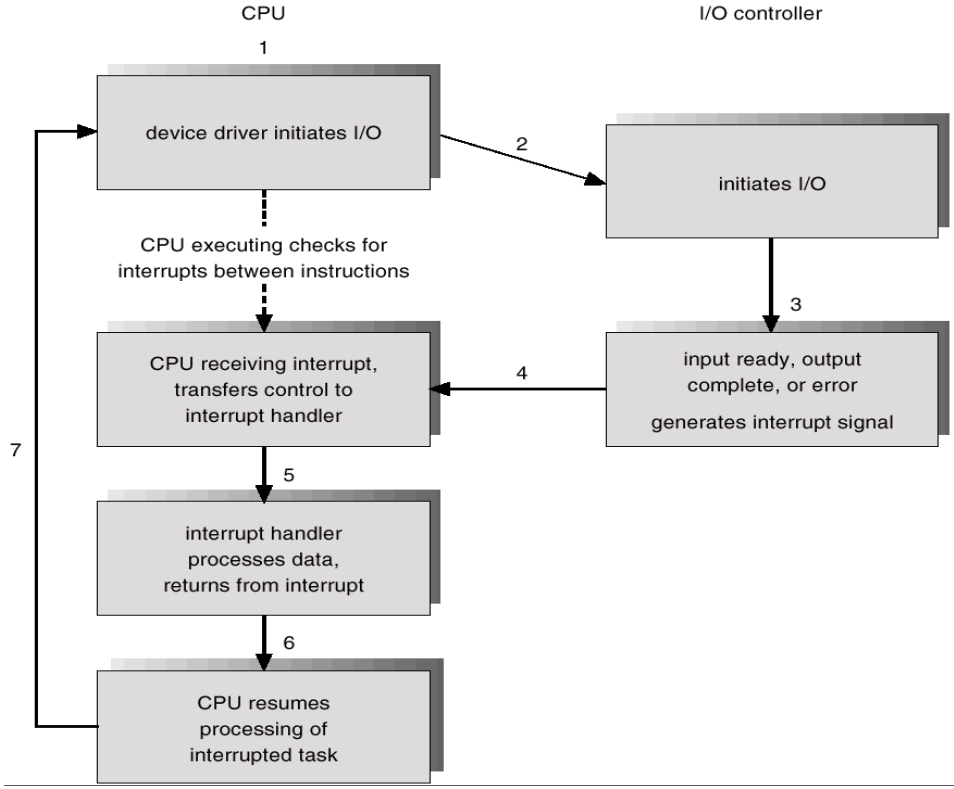
```
copy_from_user(buffer, p, count);                  /* p is the kernel bufer */
for (i = 0; i < count; i++) {                      /* loop on every character */
        while (*printer_status_reg != READY) ;     /* loop until ready */
        *printer_data_register = p[i];             /* output one character */
}
return_to_user( );
```

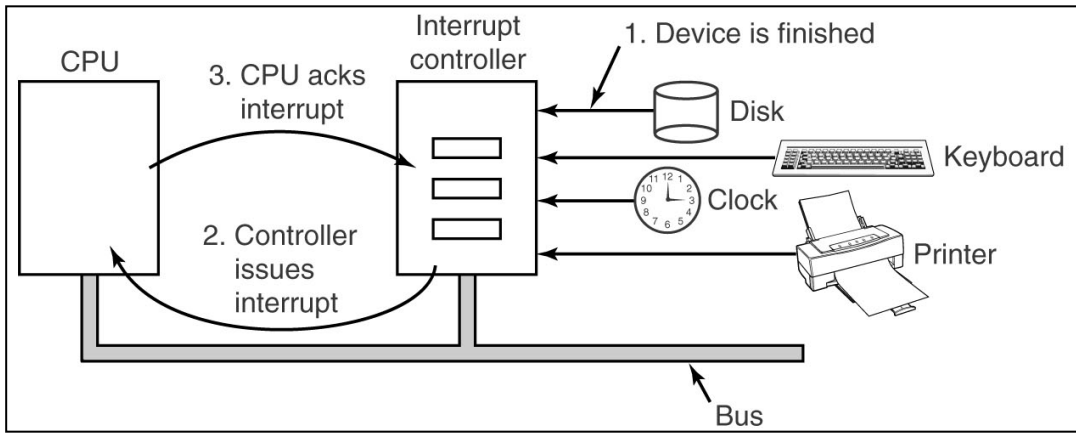
- Les E/S programmées sont simples mais accaparent la CPU.
- Elles pourraient être intéressantes si les attentes actives sont courtes ou la CPU n'a rien d'autre à faire.

Qui fait quoi ? Les E/S pilotées par les interruptions

Il n'y a pas d'attente active. Le périphérique génère une interruption, quand il termine une opération d'E/S :

- On commence par faire un appel système que le noyau traduit en un appel à une procédure du pilote approprié.
- Le pilote se charge de démarrer l'E/S quand le périphérique est prêt (octet par octet ou bloc par bloc).
- Le pilote appelle ensuite l'ordonnanceur (pour bloquer l'appelant et chercher autre chose à faire).
- Quand le contrôleur détecte la fin de transfert d'un octet, un bloc d'octets ou une erreur, il génère une interruption de fin d'E/S.
- Quand la CPU décide de prendre en charge l'interruption, elle réalise le traitement associé à l'interruption puis rend le contrôle au programme qui était en cours (si pas de préemption).





Qui fait quoi ? Les E/S pilotées par les interruptions (2)

- **Exemple 4 : E/S mappées en mémoire**

Routine de traitement de l'interruption

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Une interruption par caractère à imprimer

- La mise en mémoire tampon permet de réduire le nombre d'interruptions et les activations de processus pour de courtes périodes (bloquer / débloquer un processus à chaque lecture / écriture d'un caractère entrainant beaucoup de changements de contexte).
- L'utilisation de tampons circulaires (modèle producteur / consommateur) pour chaque sens de transfert est très répandue.

Qui fait quoi ? Les E/S pilotées par les interruptions (4)

- De manière succincte, la routine de traitement de l'interruption consiste à :

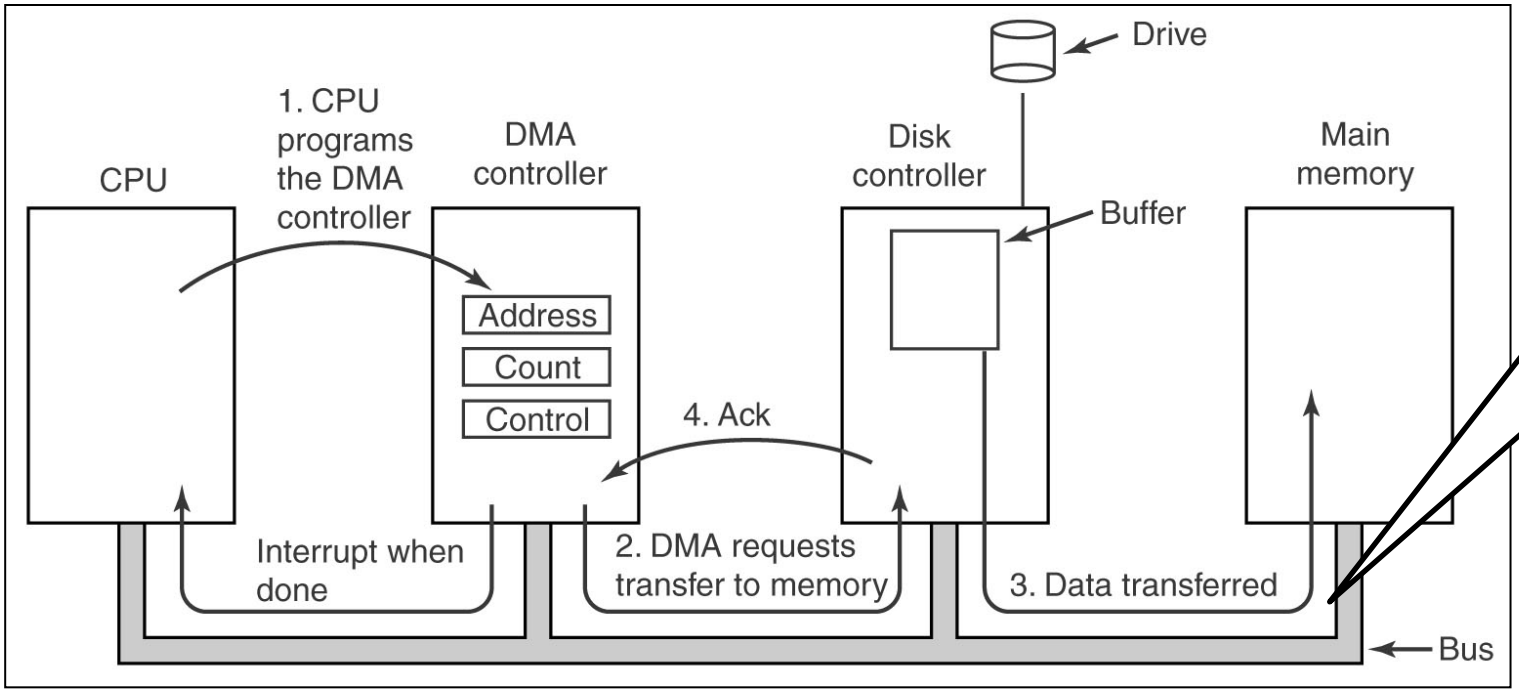
1. Sauvegarder le contexte courant (dans une pile).
2. Configurer un contexte et une plie pour la routine de service de traitement de l'interruption.
3. Acquitter l'interruption auprès du contrôleur d'interruptions (ou autoriser à nouveaux les interruptions s'il n'y a pas de contrôleur d'interruptions).
4. Copier les registres de l'emplacement où ils ont été sauvegardés vers la table des processus.
5. Exécuter la procédure de service de l'interruption. Cela extraira les informations des registres du contrôleur du périphérique qui a provoqué l'interruption.
6. Choisir le prochain processus à exécuter, si l'interruption a débloqué un processus.
7. Charger le contexte d'exécution du processus choisi (la CPU exécute le processus choisi).

Qui fait quoi ? Les E/S avec DMA

- On dispose d'un composant (puce spéciale) DMA qui contrôle le flux de bits entre la mémoire et un contrôleur de périphérique quelconque sans nécessiter l'intervention de la CPU.
- La CPU initialise la puce DMA en lui donnant le nombre d'octets à transférer, le périphérique, l'adresse mémoire concernée et le sens du transfert.
- Quand la puce a terminé le transfert, elle provoque une interruption.

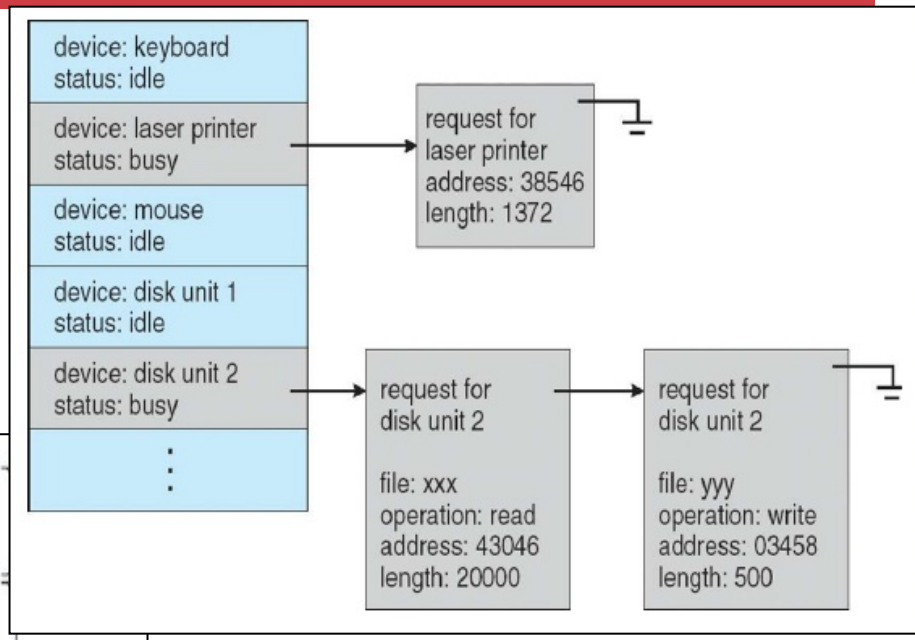
```
copy_from_user(buffer, p, count);          acknowledge_interrupt( );
set_up_DMA_controller( );                  unblock_user( );
scheduler( );                              return_from_interrupt( );

        (a)                                        (b)
```

- DMA permet de réduire le nombre d'interruptions.
- CPU beaucoup plus rapide que DMA

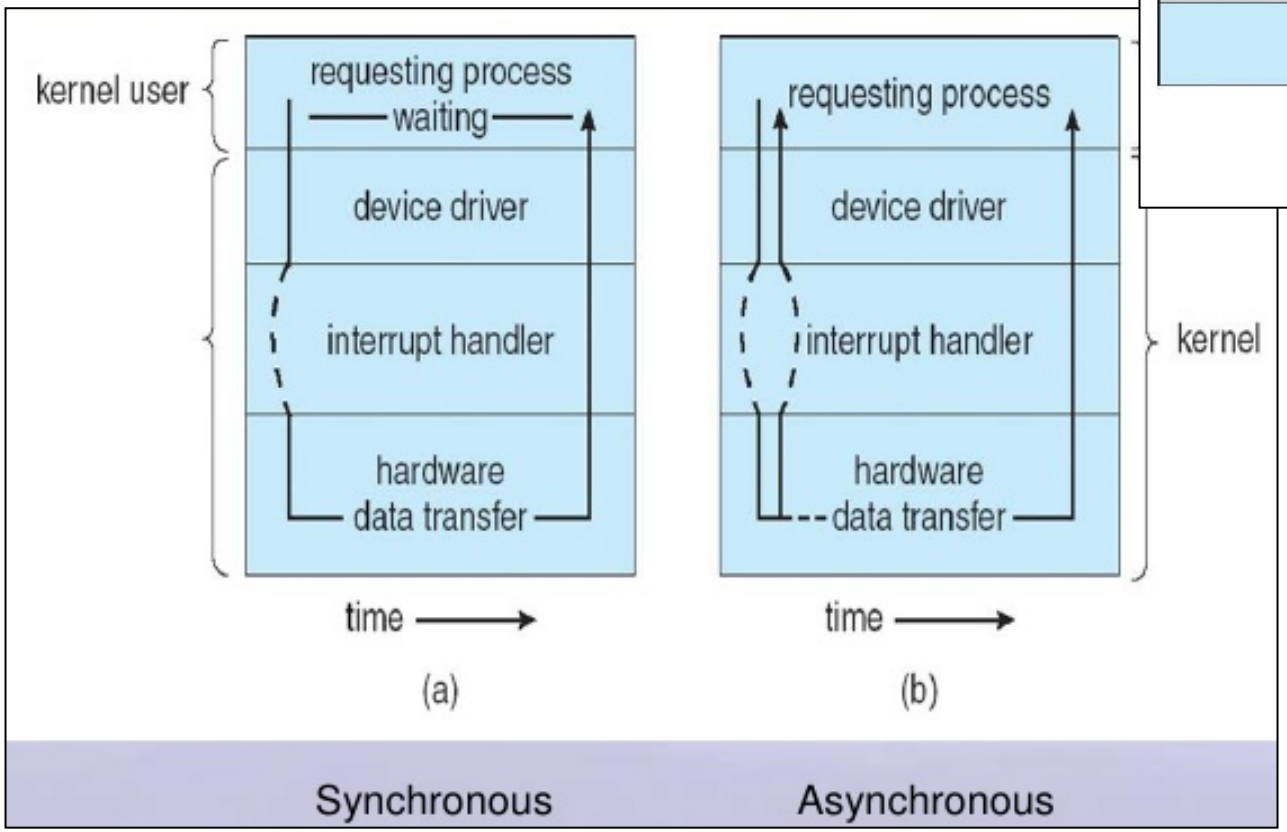


Deux modes de fonctionnement : mode avec un seul mot à la fois (vol de cycle) ou mode bloc (mode rafale)

E/S synchrones ou asynchrones

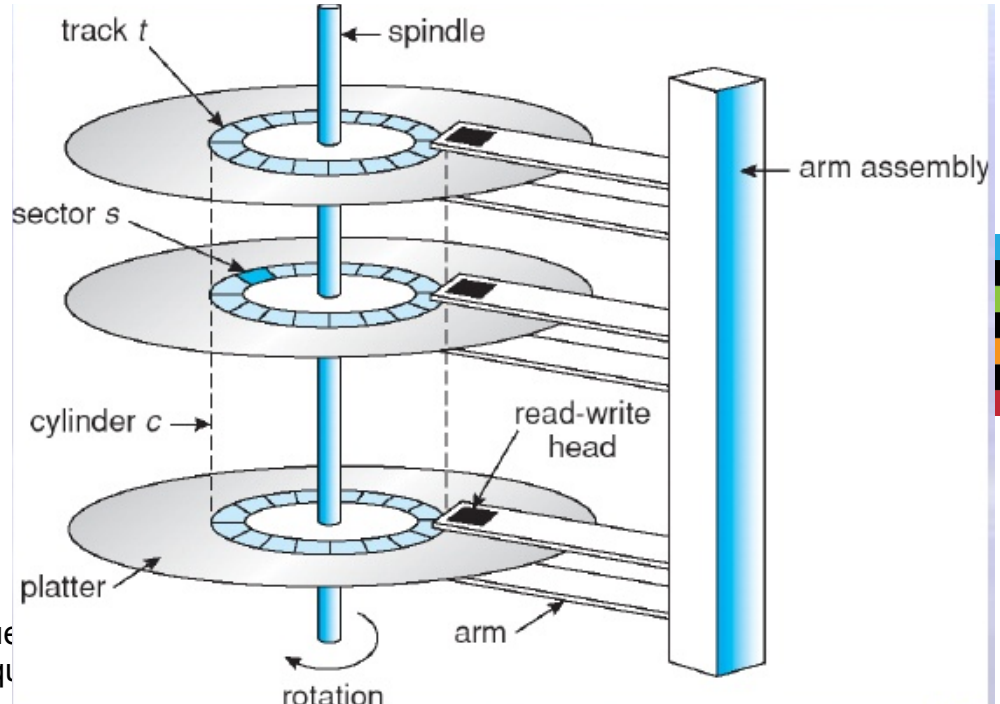


Deux types d'E/S :



Disques

- Tous les disques sont composés de cylindres qui contiennent autant de pistes qu'il y a de têtes placées verticalement.
- Les pistes se divisent en secteurs, le nombre de secteurs est compris entre 8 et 32.
- Tous les secteurs contiennent le même nombre d'octets.
- Chaque disque a un pilote qui reconnaît et exécute les demandes de lecture/écriture.
- Chaque demande de lecture/écriture nécessite d'abord de positionner le bras pour ramener la tête de lecture sur le secteur concerné.



Disques (2)

- S'il y a plusieurs demandes de lecture/écriture, une première solution consiste à traiter les requêtes dans l'ordre d'arrivée (PAPS). Cependant, pour traiter un maximum de demandes en un temps plus court, il serait plus intéressant de minimiser les déplacements du bras.

- De nombreux pilotes utilisent une table indexée par les numéros des cylindres, où chaque entrée pointe sur le premier élément d'une liste chaînée des différentes requêtes qui concernent un cylindre.

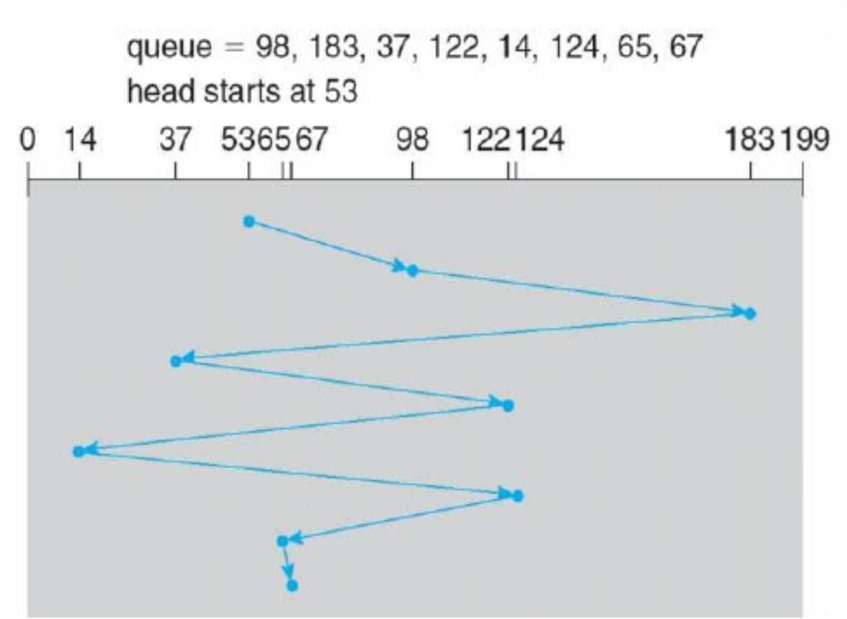
- A la fin d'une requête, le pilote doit choisir la prochaine requête à traiter par exemple celle qui concerne le cylindre le plus proche de la position actuelle du bras (plus court déplacement d'abord Shortest Seek First SSF) .

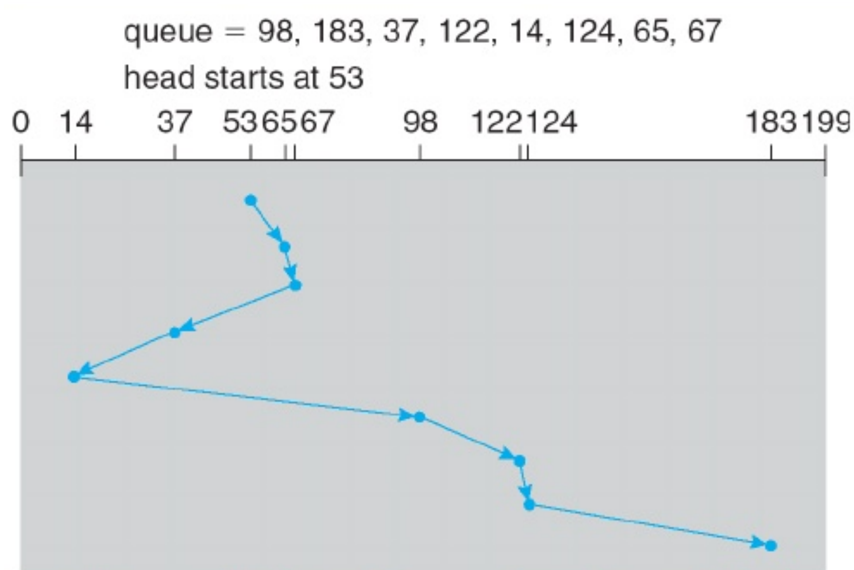
- Cet algorithme présente un inconvénient. Si de nouvelles requêtes arrivent continuellement concernant des cylindres proches de la position actuelle, le traitement des requêtes qui concernent les cylindres éloignés risquerait d'être retardé indéfiniment.

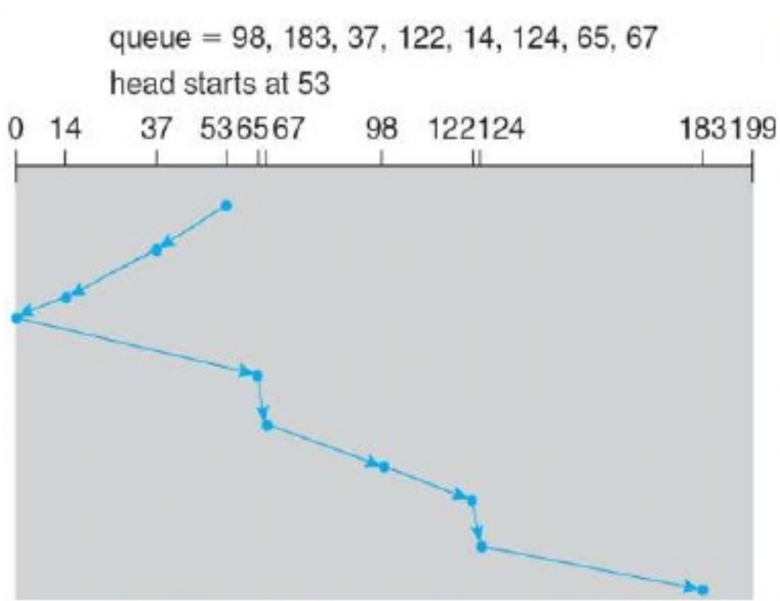
- Une autre solution consiste à choisir un sens de parcours (montant ou descendant) puis à servir toutes les requêtes dans cette direction en commençant par les plus proches, jusqu'à épuisement. On inverse alors le sens de parcours et on recommence (l'algorithme de l'ascenseur SCAN). Le déplacement maximum du bras est égal à deux fois le nombre de cylindres.

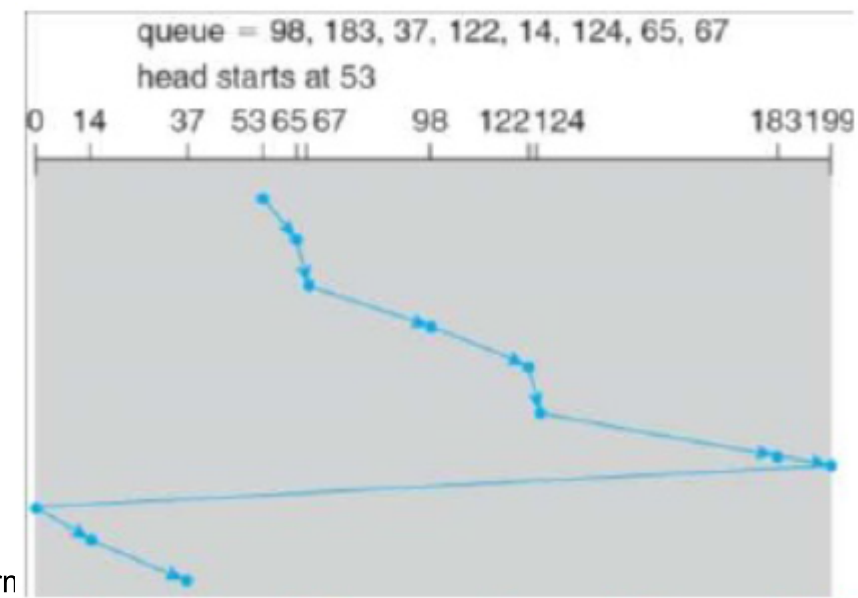
Disques (3)
PAPS, SSF, SCAN et C-SCAN (Circular-SCAN)

Exemples 5 :









Systèmes de fichiers

- **Qu'est-ce qu'un système de fichiers ?**
- **Stockage des fichiers**
- **Systèmes de fichiers UNIX (FFS), Linux Ext2, MS-DOS (FAT) et NTFS**

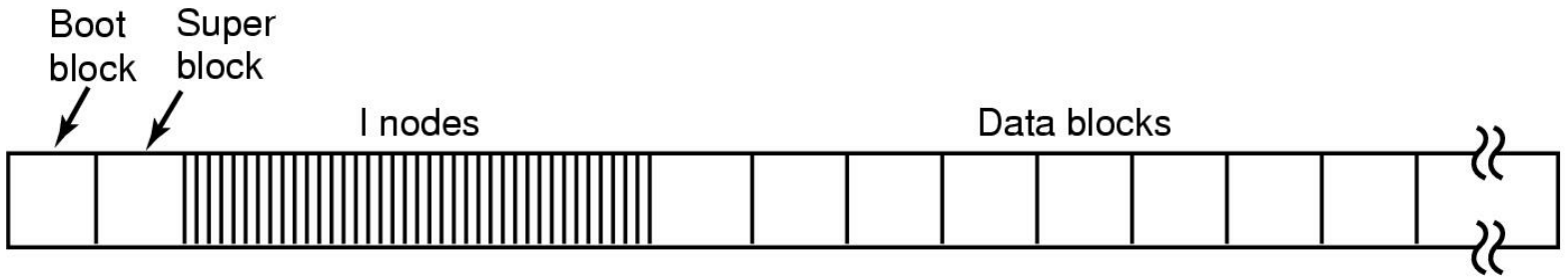
Qu'est ce qu'un système de fichiers ?

- Un système de fichiers est la partie du système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers (sur une unité de stockage : partition, disque, CD, disquette).

- Avant qu'un système de fichiers puisse créer et gérer des fichiers sur une unité de stockage, son unité doit être formatée selon les spécificités du système de fichiers.

- Le formatage inspecte les secteurs, efface les données et crée le répertoire racine du système de fichiers.

- Il crée également un superbloc pour stocker les informations nécessaires à assurer l'intégrité du système de fichiers.



Qu'est ce qu'un système de fichiers ? (2)

- Un fichier désigne un ensemble de données manipulées comme une seule unité ou individuellement :
  - create, open, close, unlink, copy, rename, list….
  - read, write…

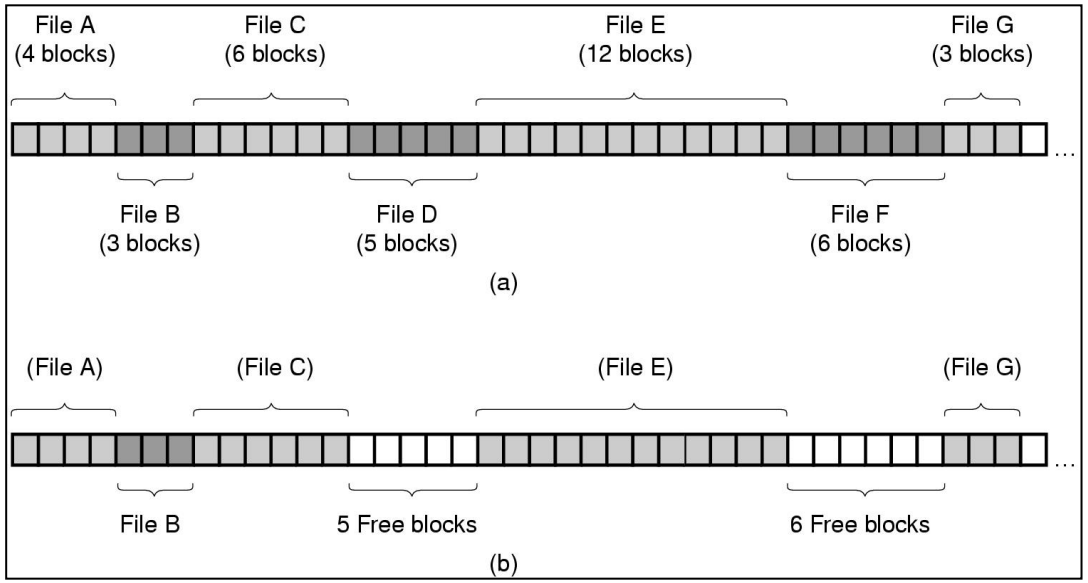
System call	Description
fd = creat(name, mode)	One way to create a new file
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

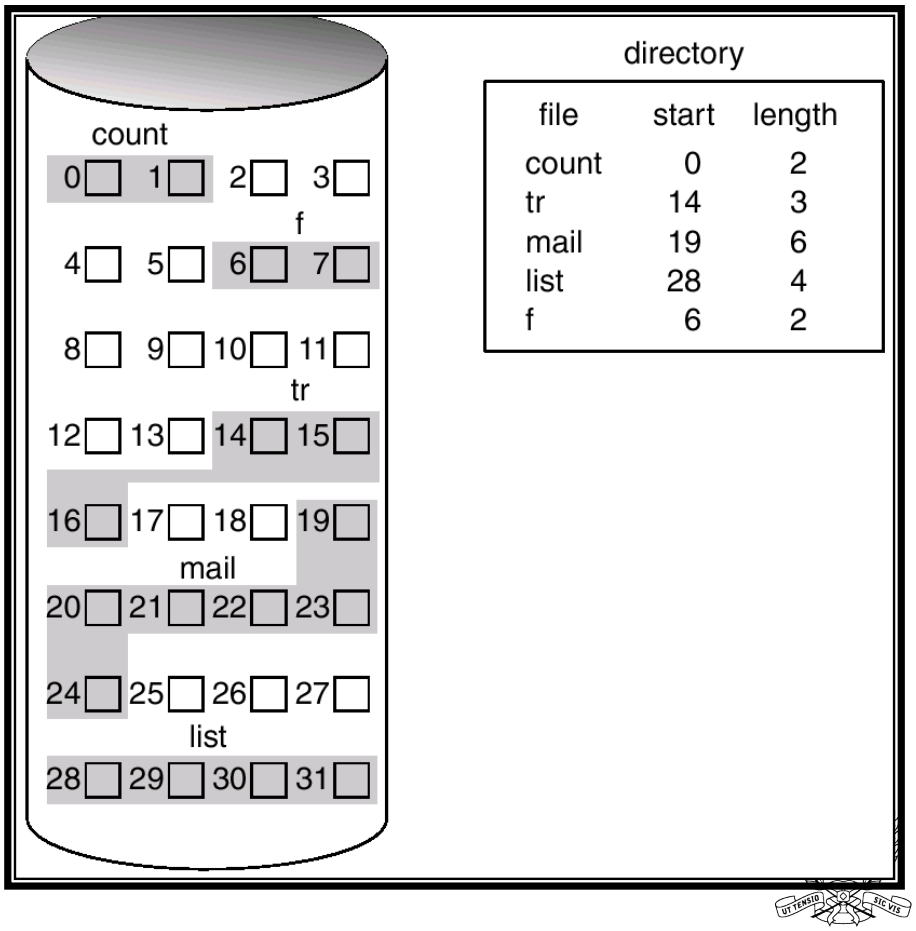
- Il a un ensemble d'attributs qui peuvent être regroupés en deux catégories :
  - Les attributs qui servent à contrôler les accès (code de protection, mot de passe, propriétaire …).
  - Les attributs qui définissent le type et l'état courant du fichier (indicateur du type ASCII/binaire, taille courante, location, date de création, date de la dernière modification, …).
- Pour le système d'exploitation, un fichier est une suite d'octets. Par contre, les utilisateurs peuvent donner des significations différentes au contenu d'un fichier (suites d'octets, suite d'enregistrements, arbre…)

Stockage des fichiers

- Chaque fichier (ordinaire ou répertoire) d'un système de fichiers est stocké sur l'unité de stockage du système de fichiers. Ses données sont dans des blocs de taille fixe (512, 1024, ou 2048 octets, …).
- À chaque fichier est alloué un nombre de blocs. Le bloc est l'unité d'allocation => fragmentation interne.
- La lecture ou l'écriture d'un élément d'un fichier impliquera le transfert vers la mémoire du bloc entier qui contient cet élément.
- Le système de fichiers conserve, dans un ou plusieurs blocs spécifiques (superblocs), un certain nombre d'informations telles que le nombre de ses blocs, leur taille, la liste des blocs libres….
- Pour pouvoir récupérer l'état des blocs (libre ou alloué), les systèmes de fichiers maintiennent :
  - une liste chaînée des blocs libres ou
  - une table de bits contenant un bit pour chaque bloc (0 pour libre).
- Les blocs d'un même fichier sont contiguës ou non contiguës

Stockage des fichiers (2)
Allocation contiguë





(a) Allocation d'espace disque pour 7 fichiers

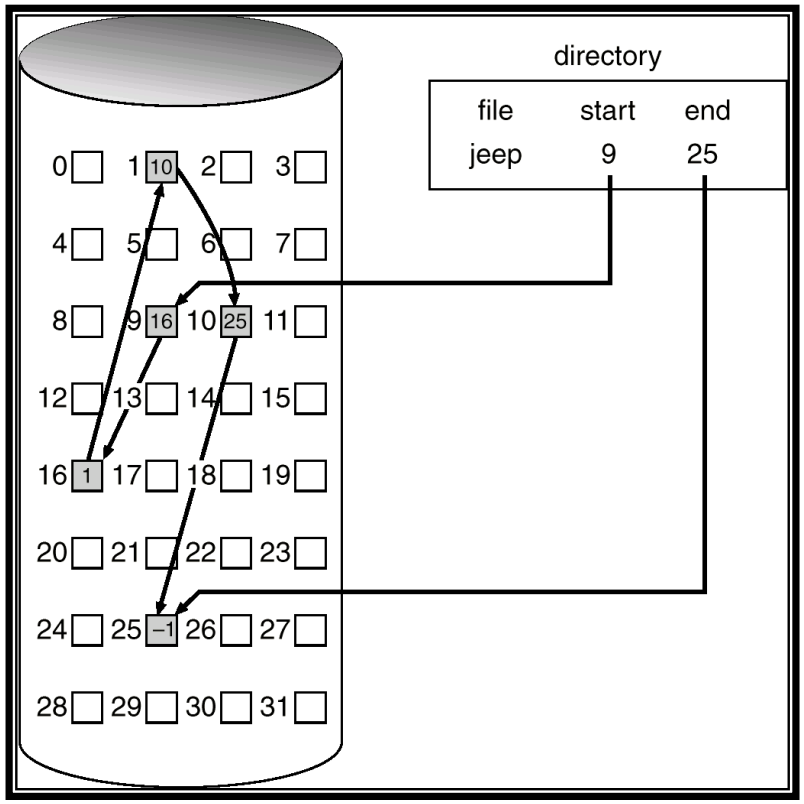
(b) Suppression des fichiers *D* et *F*

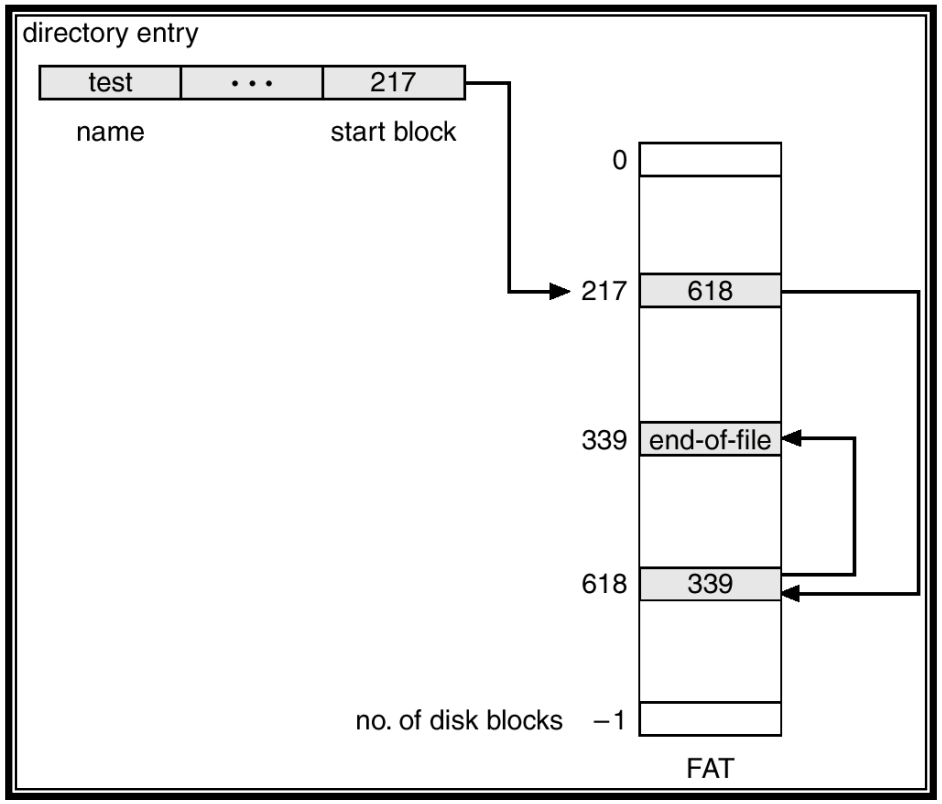
⇒ Fragmentation externe

⇒ Facile à implémenter

⇒ Bonne performance de lecture

Stockage des fichiers (3)
Allocation non-contiguë : Chaînage des blocs & table d'index





⇒ Élimine la fragmentation externe

⇒ Temps d'accès à un bloc est important (accès à tous les blocs qui le précèdent)

Table d'index (File Allocation Table (FAT))

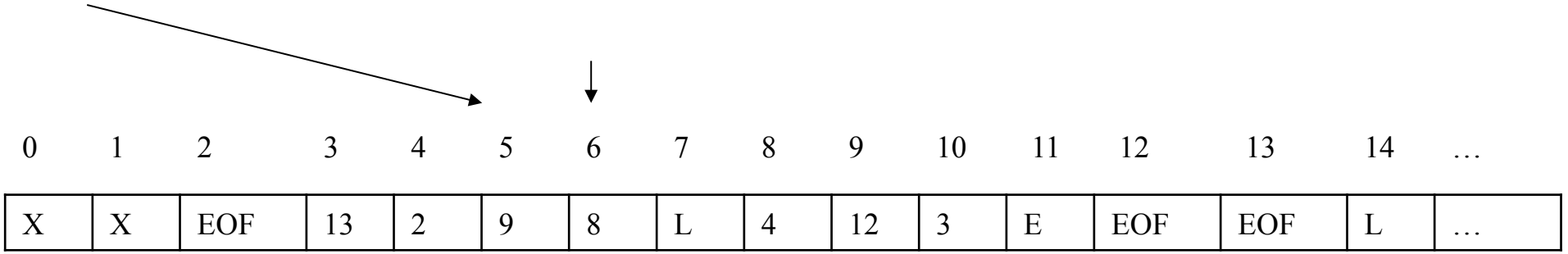
- Table trop grande (ex. un disque de 200 GO avec des blocs de 1 KB, cette table nécessite 600 MO).
- **La taille de la table croit linéairement avec la taille du disque.**

Système de fichiers FAT (MS-DOS, clés USB)

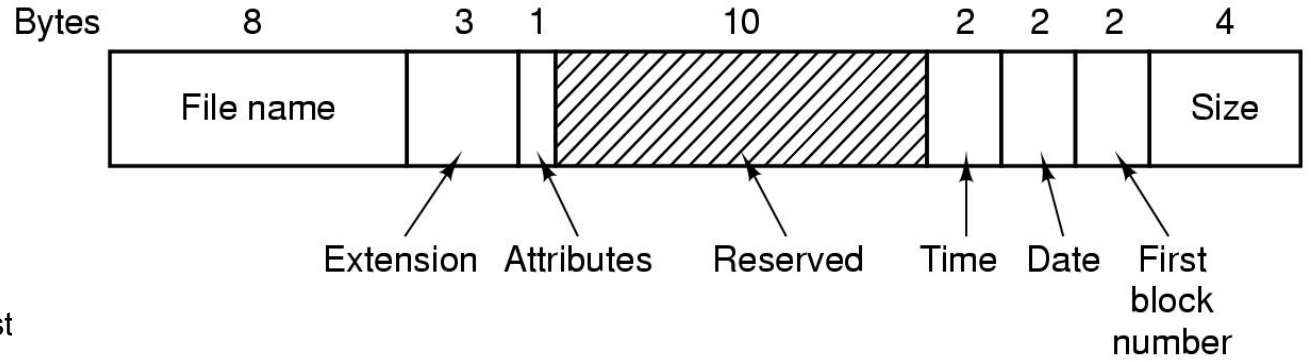
Disque

Boot	Deux copies de la FAT	Répertoire Racine			Données et Répertoires		

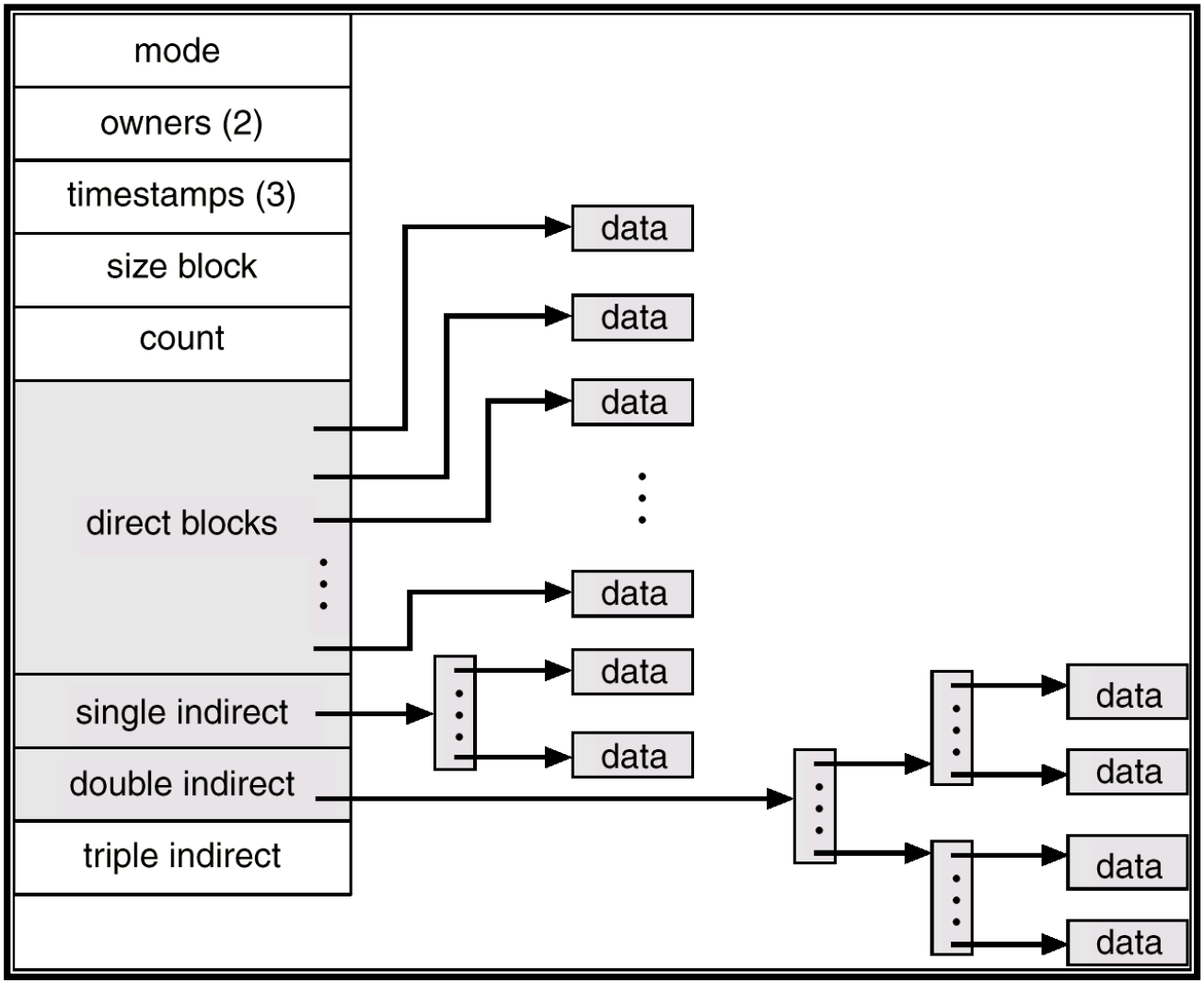
- La FAT permet de localiser les blocs de chaque fichier du système de fichiers.
- Elle comporte une entrée pour chaque bloc de l'unité de stockage.
- Le numéro du premier bloc d'un fichier est un attribut du fichier.



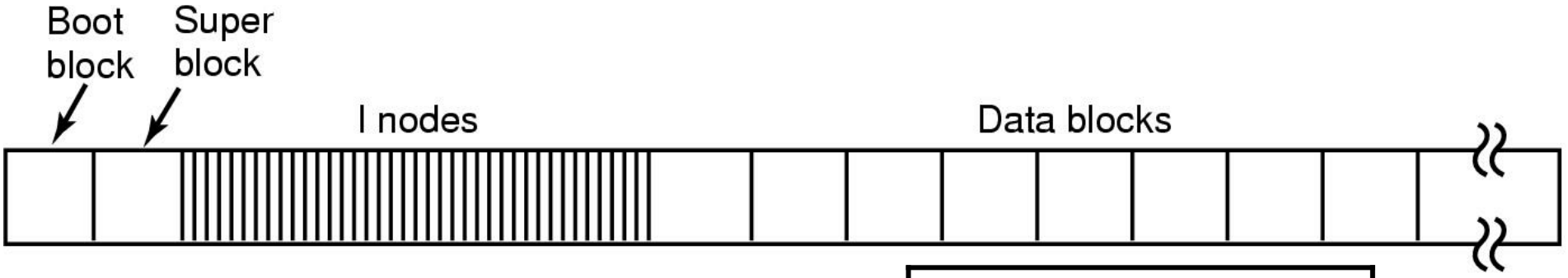
- Structure d'un répertoire (cas de MS-DOS)

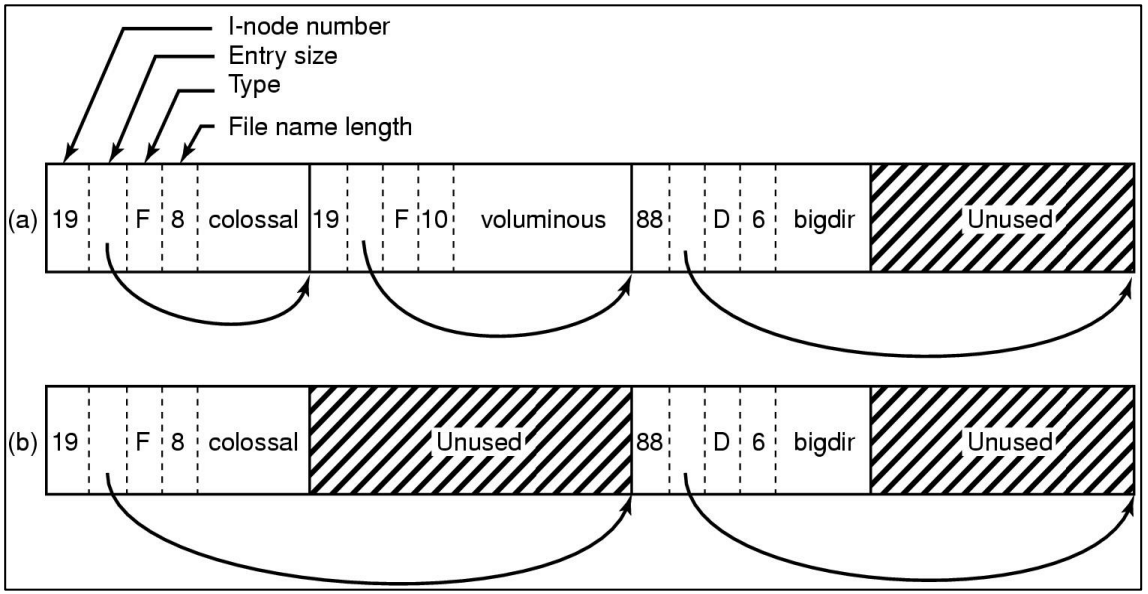


Stockage des fichiers (4)
Allocation non-contiguë : Tables d'index des i-noeuds

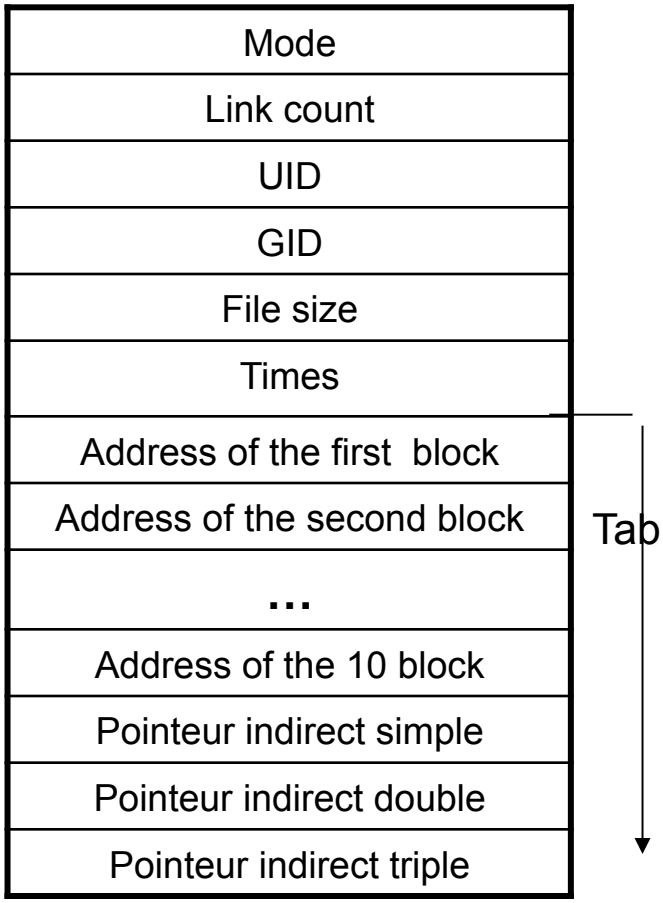


Système de fichiers d'UNIX (FFS)



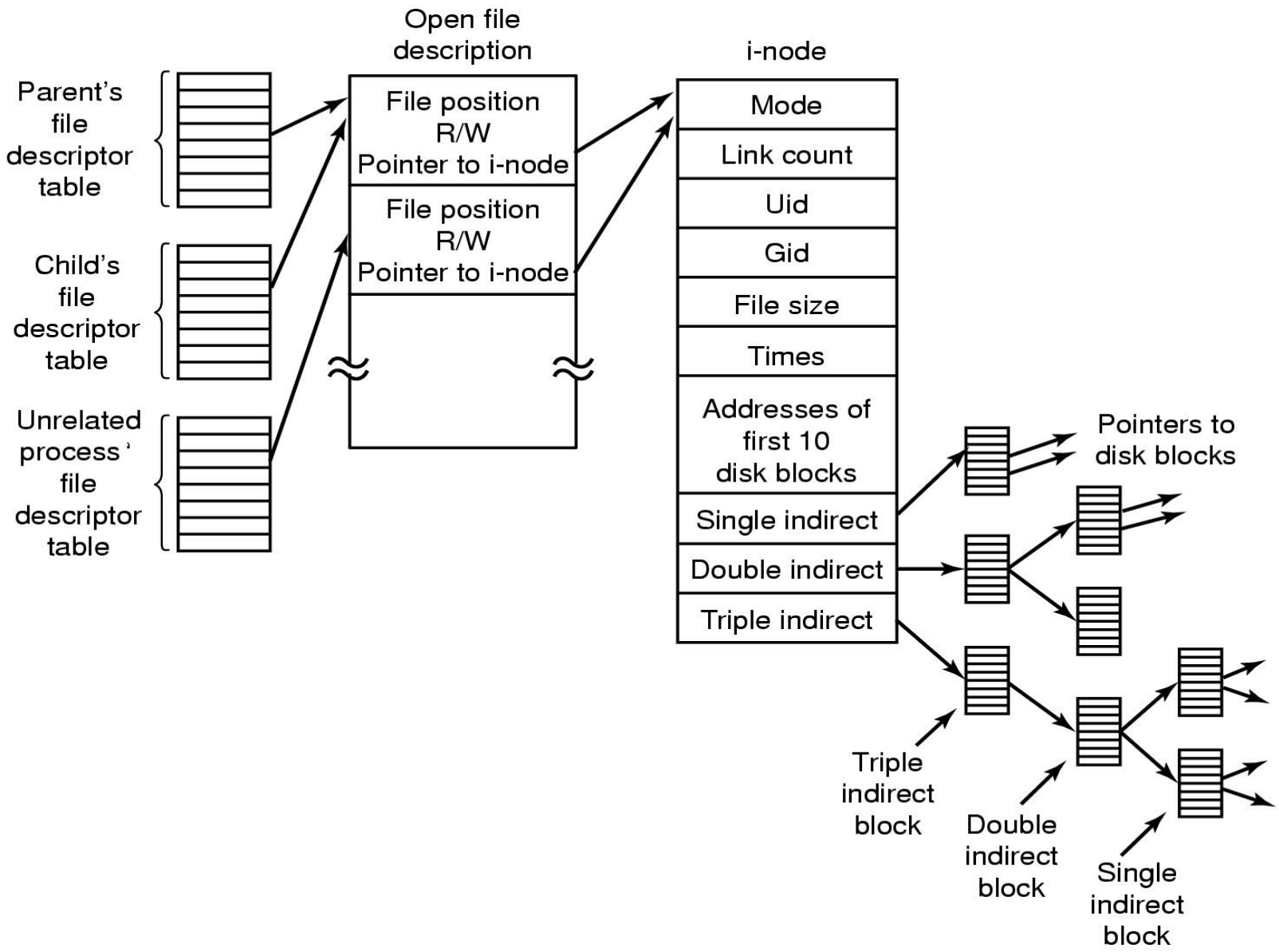


Structure d'un répertoire



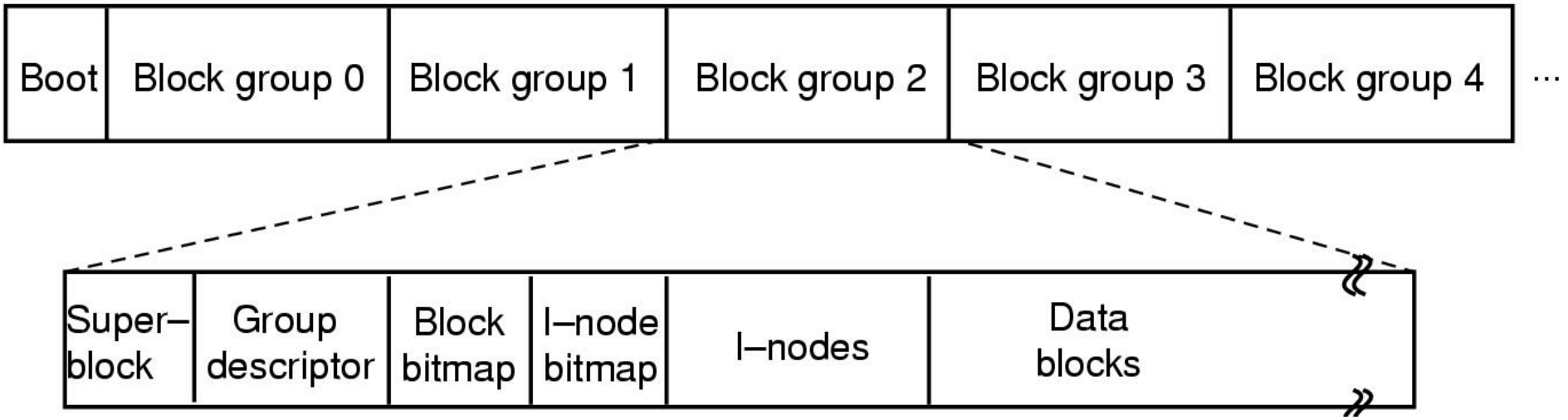
Structure d'un i-noeud

Système de fichiers d'UNIX (2)



Relation entre la table des descripteurs de fichiers et la table des fichiers ouverts

Système de fichiers de Linux Ext2



Exercice

- On considère un système disposant d'un système de fichiers similaire à celui d'UNIX avec une taille de blocs de données de 4KO (4096 octets) et des pointeurs (numéros de blocs) définies sur 4 octets.

- On suppose que le i-noeud de chaque fichier compte 12 pointeurs directs, 1 pointeur indirect simple, 1 pointeur indirect double et 1 pointeur indirect triple.

- On désire créer un fichier contenant un total de 20.000.000 (vingt millions) de caractères (caractères de fin de ligne et de fin de fichier compris).
- Quelle est la fragmentation interne **totale** sur le disque résultant de la création de ce fichier ?
  - 20 000 000 octets = 4882 * 4096 + 3328 octets
  - 4883 = 12 + 4096 + 775
  - 4096 – 3328 = 768