Noyau d'un système d'exploitation
INF2610

Chapitre 1 : Concepts généraux

Département de génie informatique et génie logiciel

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Automne 2017

Chapitre 1 - Concepts généraux

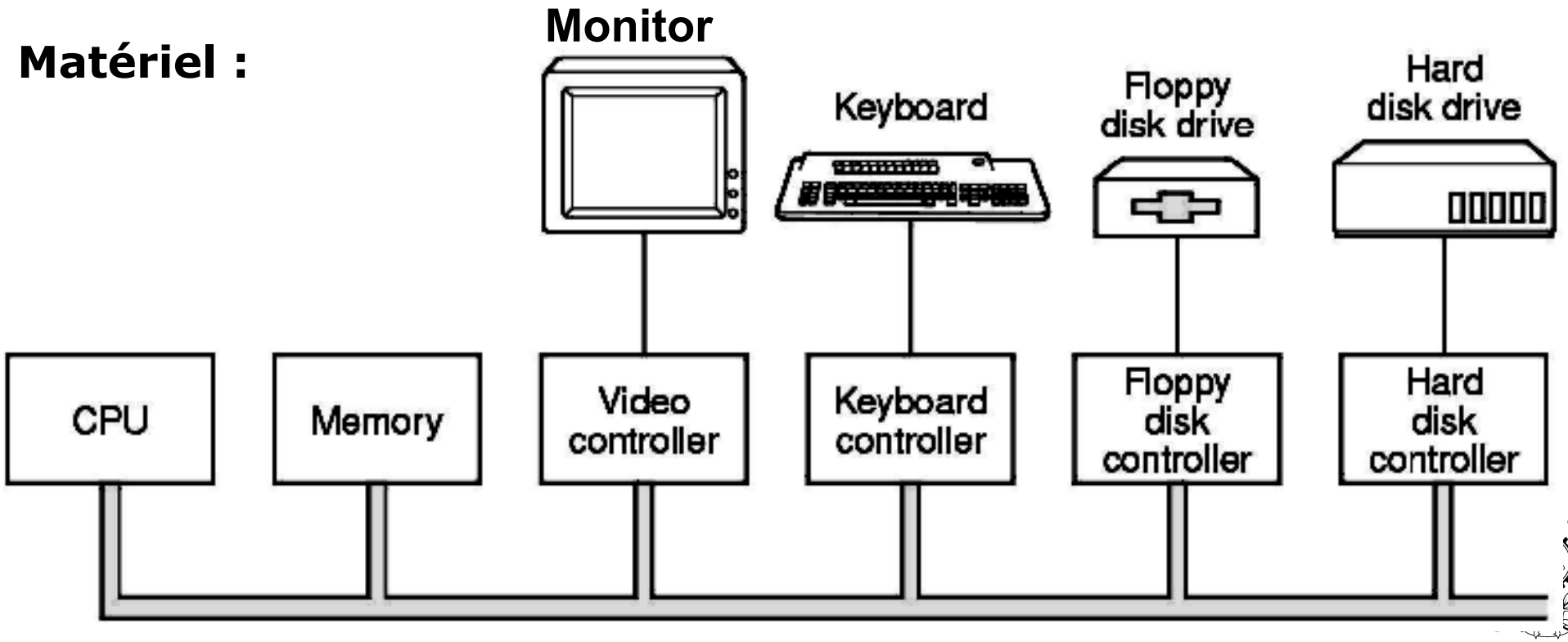
- **Qu'est ce qu'un système d'exploitation** ?
- **Interface avec le matériel**
- **Interactions utilisateur/système**
- **Principaux concepts :**
  processus, fichiers, mémoires virtuelles et E/S
- **Appels système**
- **Evolution du mode d'exploitation**
- **Structure des systèmes d'exploitation**

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

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

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

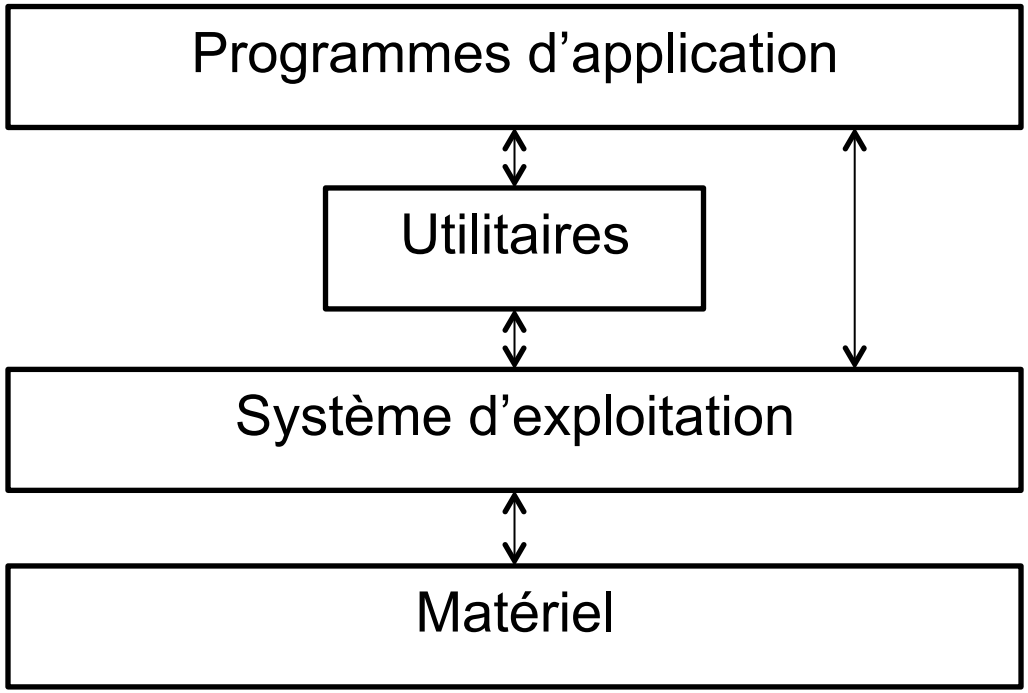
Matériel :



(à lire – architecture des ordinateurs)

Qu'est-ce qu'un système d'exploitation ? (2)

- **Les logiciels:**
  - les programmes d'application.
  - les programmes système:
    - les utilitaires (compilateurs, éditeurs, interpréteurs de commandes);
    - le système d'exploitation



Qu'est ce qu'un système d'exploitation ? (3)

Le système d'exploitation:

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

  services = {appels système}

But :

Développer des applications sans se soucier des détails de fonctionnement et de gestion du matériel, ou des interactions entre les applications.

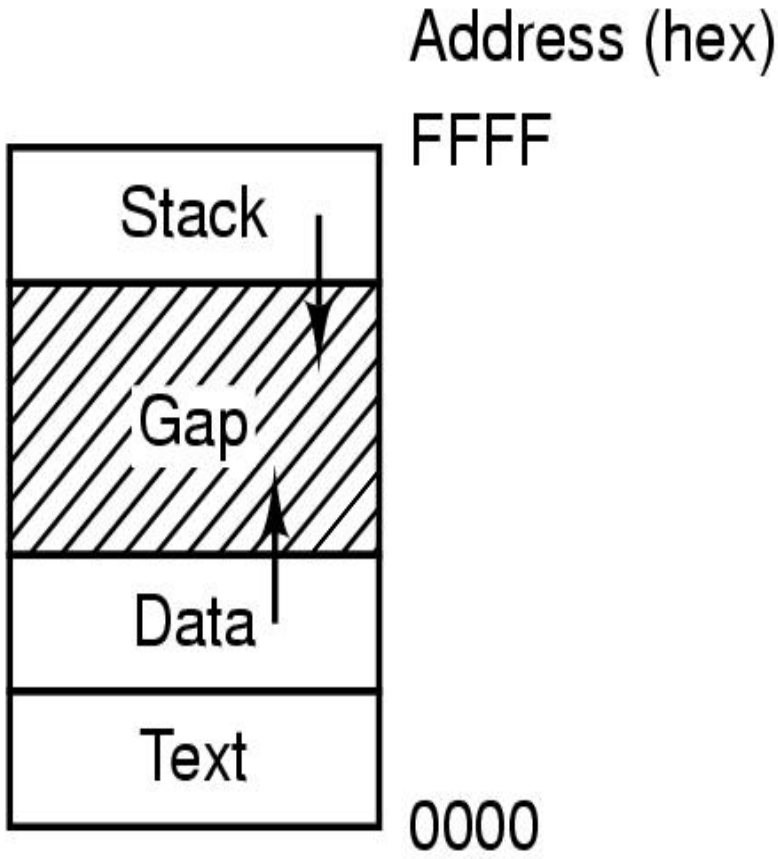
Qu’est ce qu’un système d’exploitation ? (4)

Fonctions principales d'un système d’exploitation :

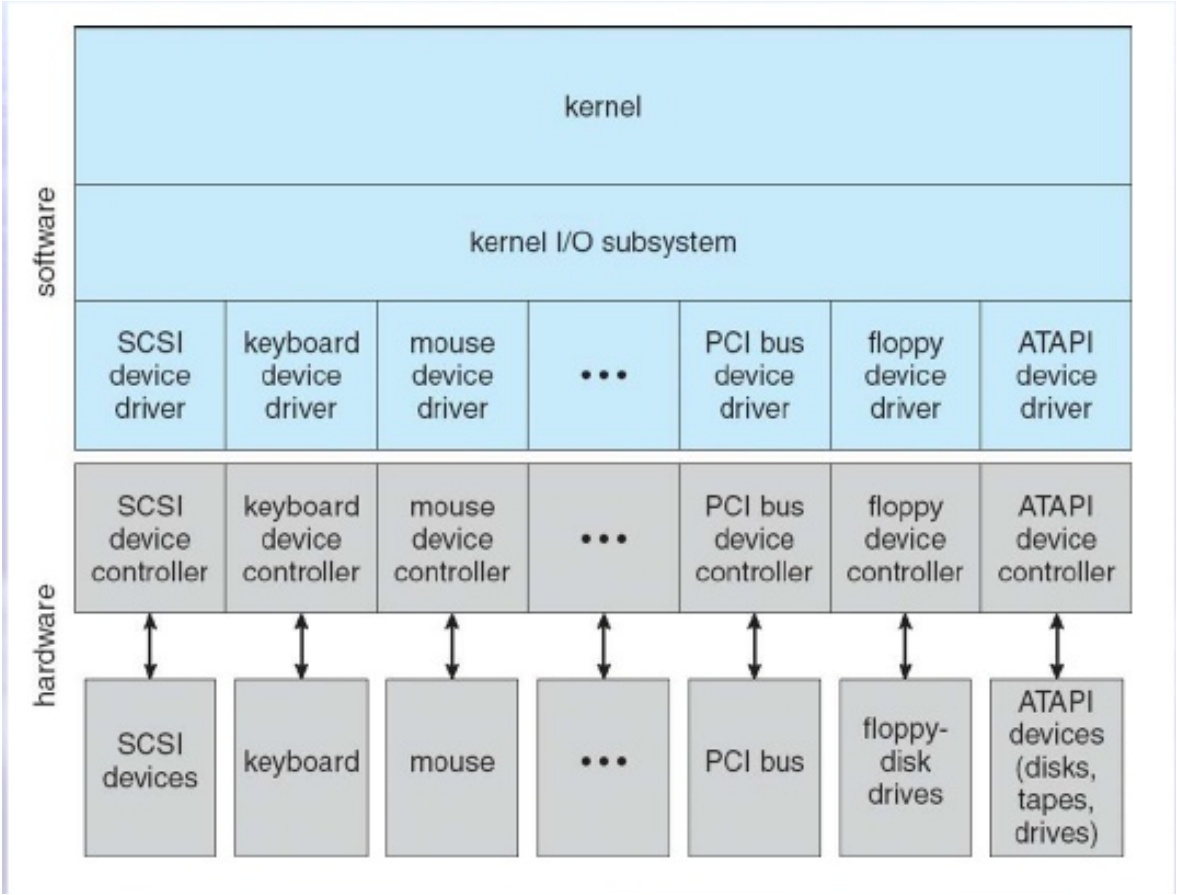
- Gestion de périphériques
- Gestion de la mémoire
- Gestion de processeurs
- Gestion de processus, fils (threads) ou tâches
- Gestion de fichiers
- Protection et détection d’erreurs

Concepts de base

- Processus : un programme en cours d'exécution, composé de: code + données + piles d'exécution, et de différents registres (e.g. compteur ordinal) caractérisant son état.
- Fichiers : données sur des unités de stockage (e.g. disque).
- Mémoires virtuelles : espaces d'adressage virtuels des processus (créés par les compilateurs) de taille pouvant excéder celle de la mémoire physique.
- Périphériques d'E/S : toute interaction avec le monde extérieur (e.g. clavier, souris, réseau).



Concepts de base



- Chaque périphérique d'E/S est généralement constitué de :
  - un composant mécanique (le périphérique lui-même) et
  - un composant électronique appelé contrôleur de périphérique qui a la charge de commander physiquement le périphérique, en réaction aux commandes envoyées par le SE via un pilote de périphérique (device driver).

Interface avec le matériel

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

- Le système d'exploitation gère et coordonne l'ensemble de ces composants, notamment, au moyen de lectures et d'écritures sur les bus et d'interruptions.

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

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

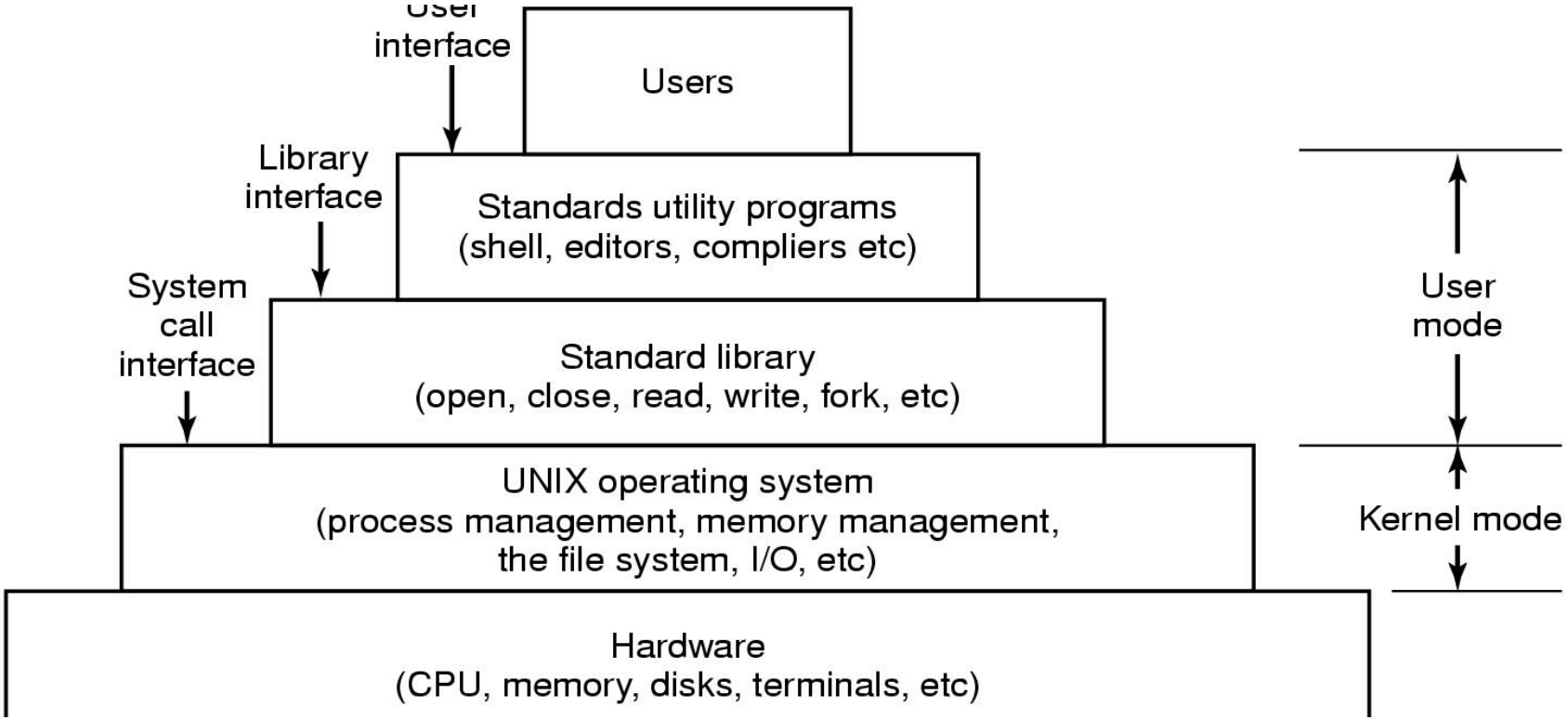
  - Interruptions logicielles :
    - Erreurs arithmétiques (division par zéro).
    - Données non disponibles en mémoire (défaut de page).
    - Appels système (invocation du système d'exploitation).

Interactions utilisateur/système

- Pour un utilisateur, le système d'exploitation apparaît comme un ensemble de procédures complexes (libc ou Win32) visant à abstraire les détails de fonctionnement et de gestion du matériel.

  Bibliothèque des appels système = {procédures}

- Les appels système peuvent être invoqués via un interpréteur de commandes, une interface graphique, des utilitaires ou encore des programmes applications.



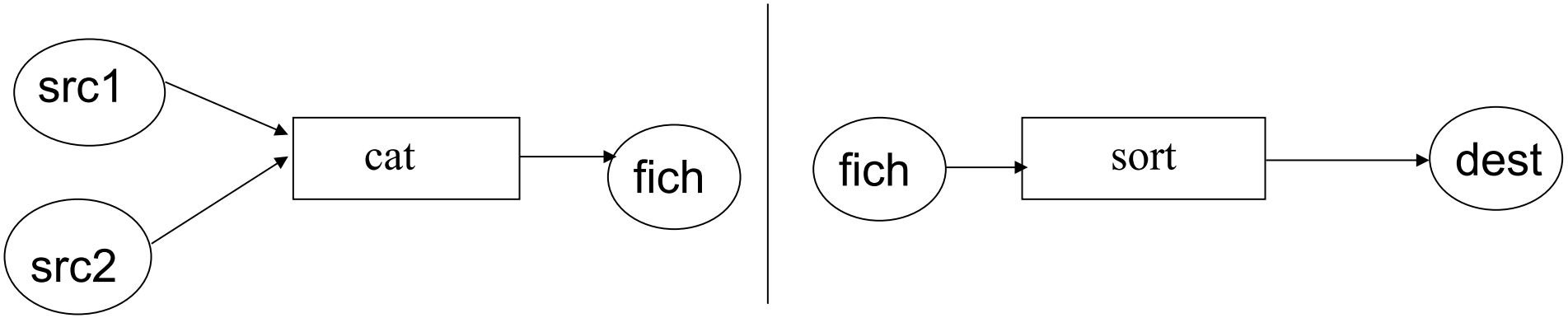
Interactions utilisateur/système (2) : UNIX/Linux

- L'interpréteur de commandes (Interface utilisateur/système) :
  - est lancé dès la connexion au système ;
  - invite l'utilisateur à introduire une commande ;
  - récupère puis exécute la commande par combinaison d'appels système et d'outils (compilateurs, éditeurs de lien,…).
  - affiche les résultats ou les erreurs puis se met en attente de la commande suivante.

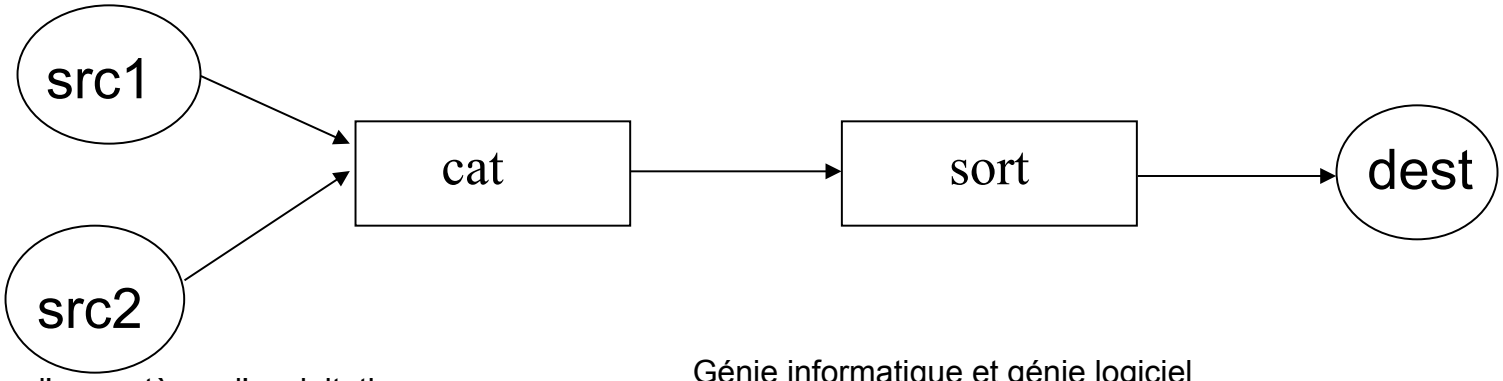
- Les interpréteurs de commandes Unix/Linux (shells) permettent une composition séquentielle ou parallèle de commandes avec redirection des entrées/sorties des commandes.

Interactions utilisateur/système (3) : appels système via un interpréteur de commandes

cat src1 src2 > fich; sort fich > dest



cat src1 src2 | sort >dest



Interactions utilisateur/système (4) : appels système via un interpréteur de commandes

- Les interpréteurs de commandes UNIX (shells) offrent des structures de contrôle semblables à celles des langages de programmation classiques (*shell script*).

```
jupiter% cat  script1
set  `ls`
for i in $* do
   if [ -d $i ]; then
       echo "$i est un répertoire"
    fi else if [ $i = "fich"  ]; then
       echo "fich trouvé. Affichage ? (o ou n) "
      read rep
      case $rep in
         o | O )      cat $i ;;
         n | N )      echo "pas de visualisation" ;;
         * )    echo  "réponse incorrecte"
      esac
    fi
done
```

Interactions utilisateur/système (5) : appels système via un programme

- Le programme C suivant utilise les appels système open, write et read. Il crée un fichier dans lequel il copie les données lues à partir du clavier.

```
#include <unistd.h>  // pour open, write et read
#include <fcntl.h>
#define taille 80
int main ( )
{ int fd , nbcar;
  char buf[taille] ;
  fd = open("fich", O_CREAT| O_WRONLY); // créer un fichier
  if(fd==-1)
  { write(2, "Erreur d'ouverture \n", 25) ;

    return -1 ;
  }
  write(1, "Ouverture avec succès \n" , 30) ;

  // copier les données à partir du clavier dans le fichier
  while ((nbcar = read(0, buf, taille)) > 0)
    if( write(fd, buf, nbcar) == -1) return -1 ;
  return 0 ;
}
```

Interactions utilisateur/système (6) : appels système via un programme

- Le programme suivant montre la différence entre write et printf

```
#include <unistd.h>                    /* pour write */
#include <stdio.h>                     /* pour printf */
int main( ) {

    printf(" ici 1er printf ");
    write(1," ici 1er write ",16);
    printf(" ici 2eme printf ");
    write(1," ici 2eme write ", 17);
    printf(”fin de ligne de printf \n");
    write(1, " ici 3eme write \n",18);
    return 0;

}
```

```
jupiter$ ./Write_Printf
  ici 1er write  ici 2eme write ici 1er printf  ici 2eme printf fin de
ligne de printf
 ici 3eme write
```

Que se passe-t-il lors d'un appel système ?

Appels système

- En général, les processeurs ont deux modes de fonctionnement :
  - le mode superviseur (noyau, privilégié ou maître): pour le système d'exploitation, où toutes les instructions sont autorisées.
  - le mode utilisateur (esclave): pour les programmes des utilisateurs et les utilitaires, où certaines instructions ne sont pas permises.

- Les processeurs sont dotés d'un bit de mode.

- Ces modes de fonctionnement assurent la protection du système d'exploitation contre les intrusions et les erreurs.

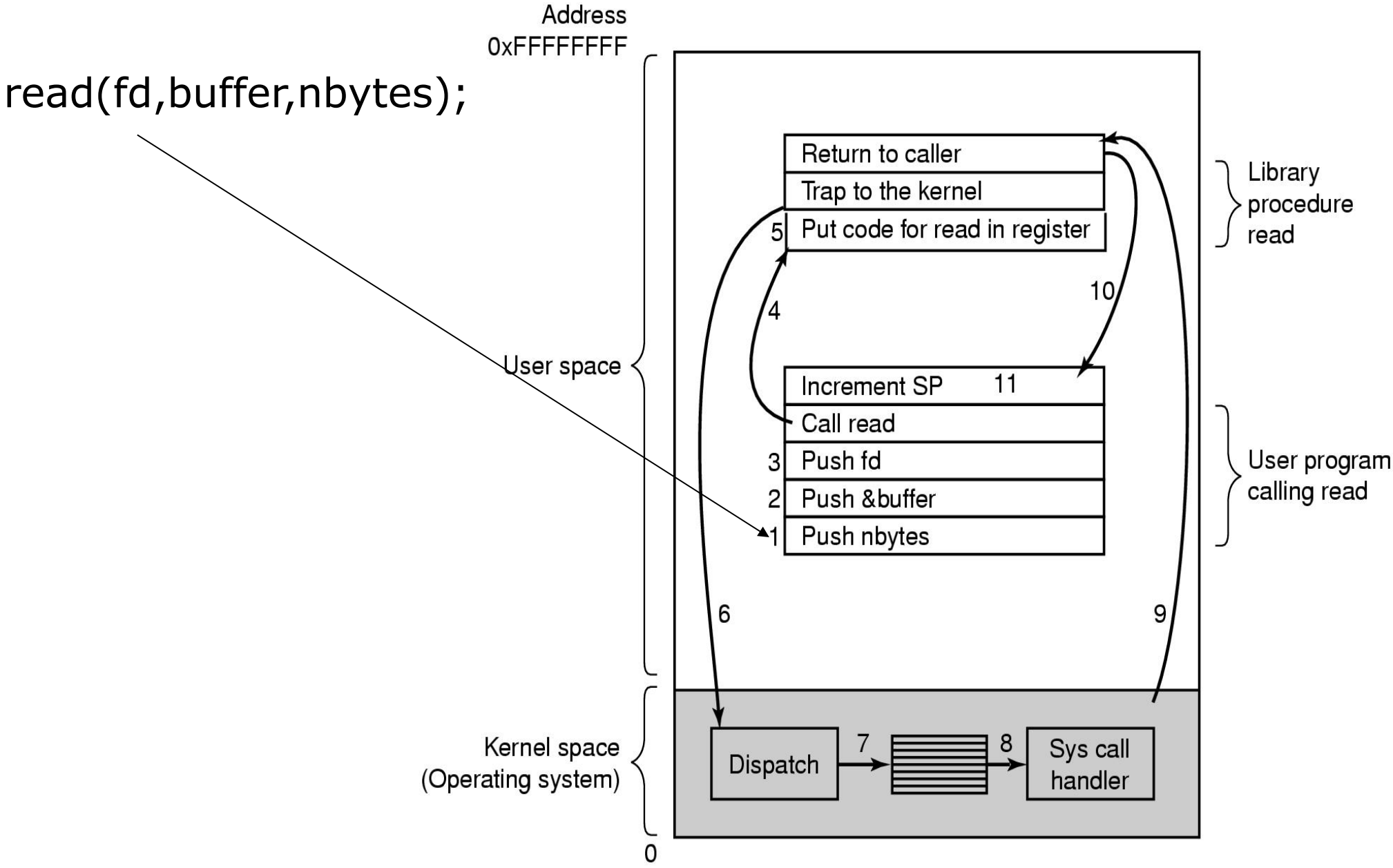
- Ce n'était pas le cas des systèmes mono-utilisateur (MS-DOS, MacOS) qui n'avaient qu'un seul mode de fonctionnement.

Appels système (2)

http://www.gladir.com/LEXIQUE/INTR/int80.htm

- Un appel système consiste en une interruption logicielle (instruction TRAP) qui a pour rôle d'activer le système d'exploitation:
  - changer le mode d'exécution pour passer du mode utilisateur au mode maître;
  - récupérer les paramètres et vérifier la validité de l'appel;
  - lancer l'exécution de la fonction demandée;
  - récupérer la (les) valeur(s) de retour;
  - retourner au programme appelant avec retour au mode utilisateur.

Appels système (3) : exemple de lecture

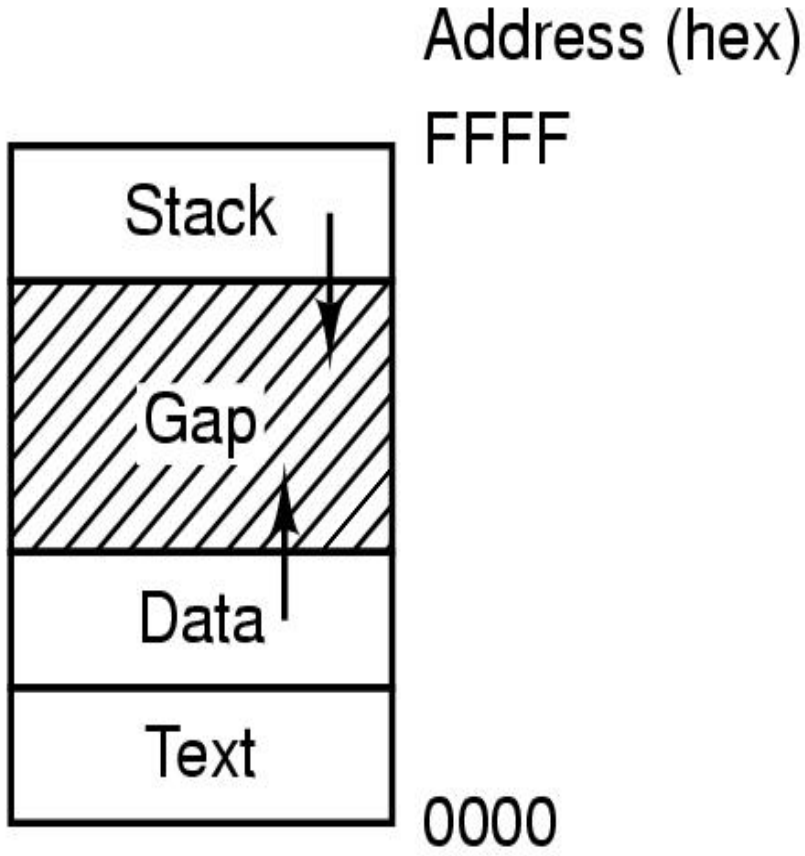


Appels système (4) POSIX et Win32

POSIX	Win32	Description
fork	CreateProcess	Créer un nouveau processus / fork + execve
waitpid	WaitForSingleObject	Attendre la fin d'un processus
execve		Remplacer l'exécutable du processus
exit	ExitProcess	Terminer le processus
open	CreateFile	Créer ou ouvrir un fichier
close	CloseHandle	Fermer un fichier
read	ReadFile	Lire d'un fichier
write	WriteFile	Ecrire dans un fichier
lseek	SetFilePointer	Changer la position courante dans le fichier
stat	GetFileAttributesEx	Lire les attributs d'un fichier
mkdir	CreateDirectory	Créer un répertoire
rmdir	RemoveDirectory	Enlever un répertoire vide
link		Ajouter un lien vers un fichier
unlink	DeleteFile	Enlever un fichier
mount		Ajouter un système de fichiers
umount		Retirer un système de fichiers
chdir	SetCurrentDirectory	Changer le répertoire courant
chmod		Changer les permissions d'un fichier
kill		Envoyer un signal à un processus
time	GetLocalTime	Obtenir le temps courant

Appels système (5) : gestion de processus

- Les appels système permettent notamment:
  - la création,
  - la communication interprocessus,
  - la synchronisation et
  - l'arrêt des processus.

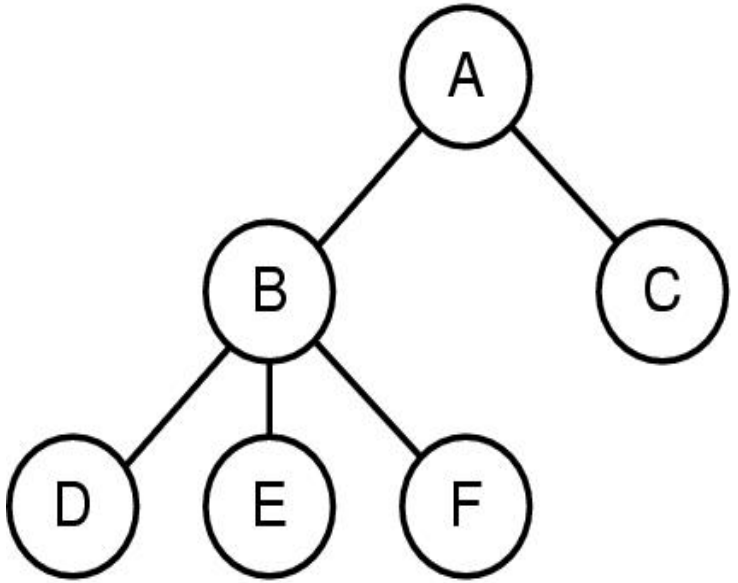


Appels système (6) : gestion de processus

Appel système	Description
pid = fork()	Créer un processus enfant identique
pid = waitpid(pid, &statloc, options)	Attendre la fin d'un processus
s = execve(name, argv, environp)	Remplacer le fichier exécutable
exit(status)	Terminer le processus

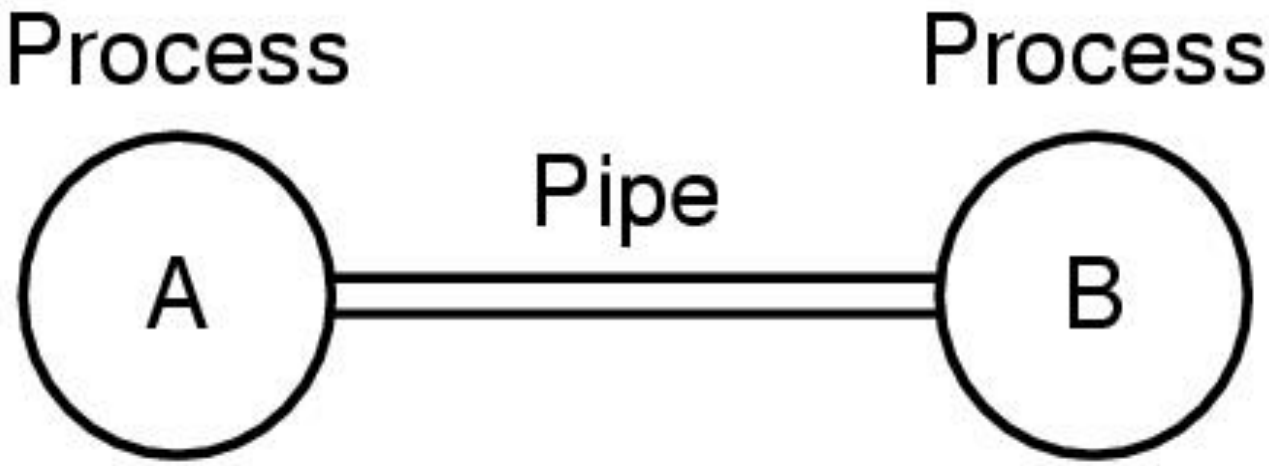
- Un processus peut créer un ou plusieurs processus enfants qui, à leur tour, peuvent créer des processus enfants (structure arborescente).

- Un processus peut être partitionné en plusieurs fils d'exécution (threads) concurrents partageant un même environnement d'exécution. Les fils d'exécution sont un moyen de raffiner et de diviser le travail normalement associé à un processus.

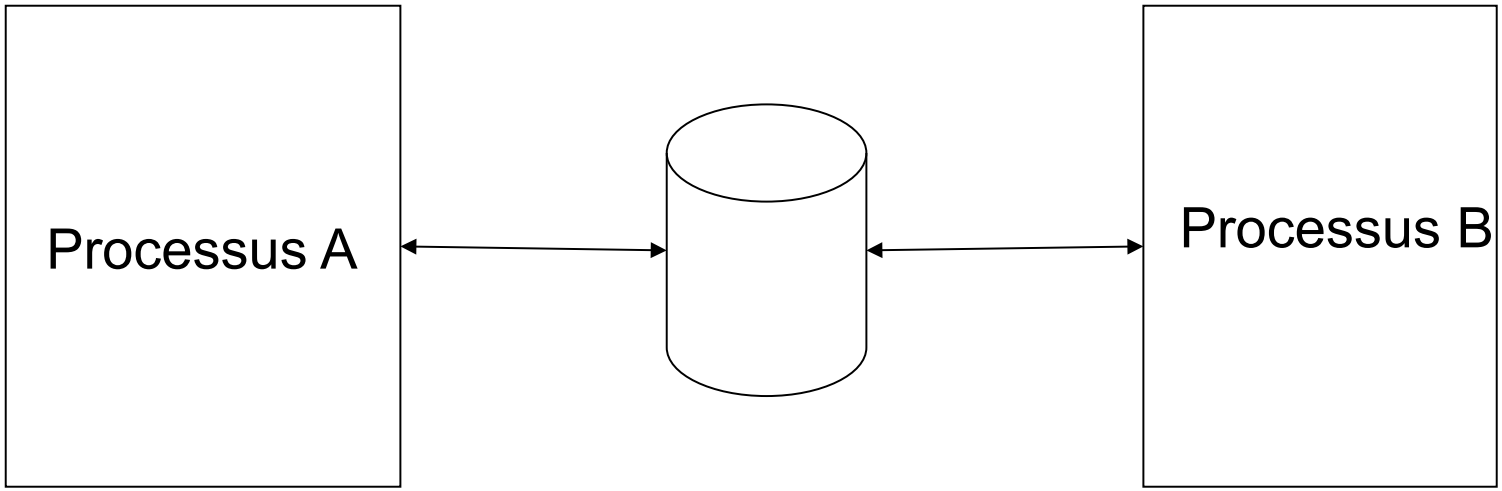


Appels système (7) : communication interprocessus

- Segments de mémoire partagés;
- Fichiers;
- Signaux;
- Messages -> Tubes de communication (pipe)…



Appels système (8) : synchronisation de processus

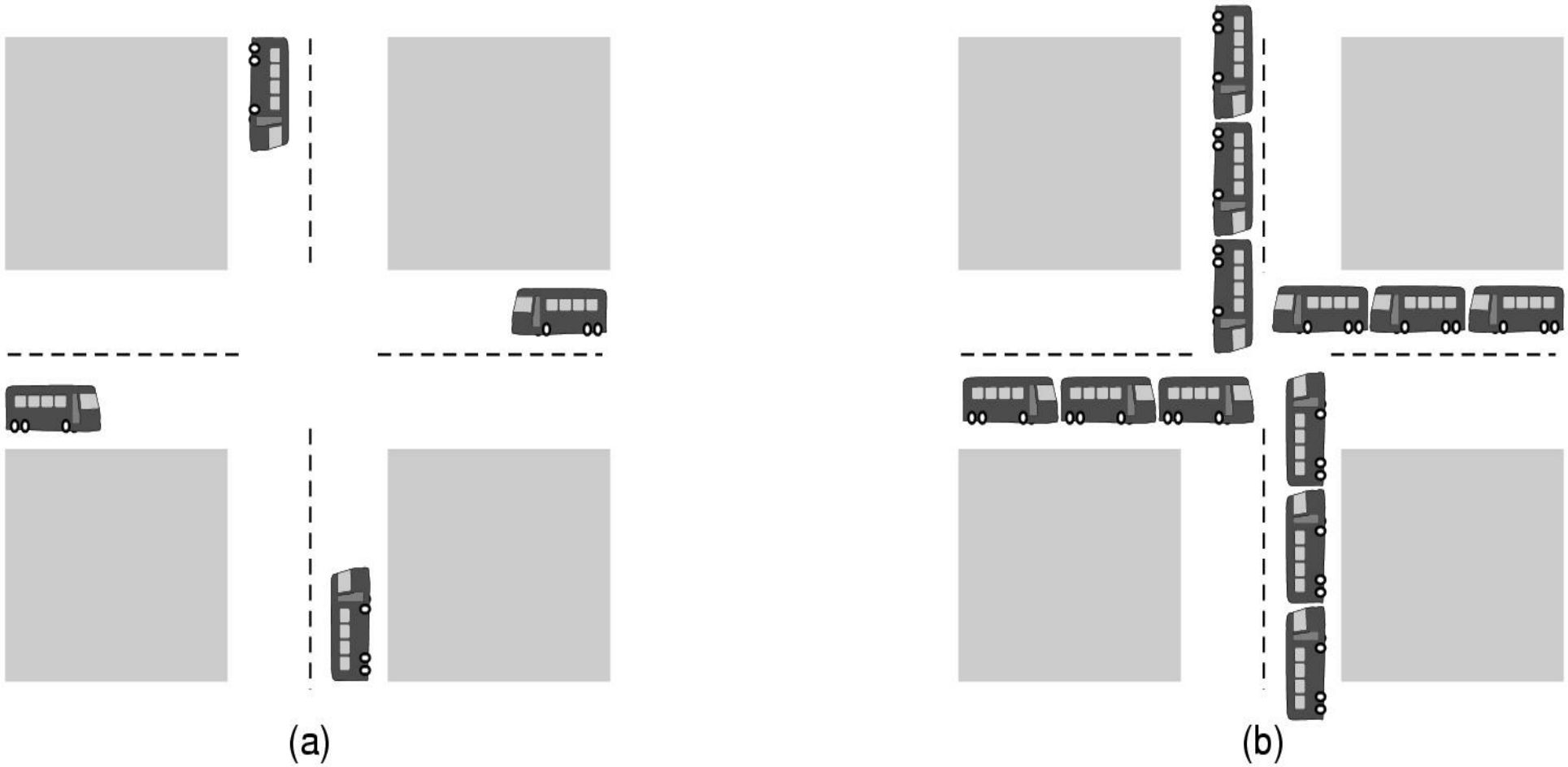


Éviter l'accès simultané lecture/écriture ou écriture/écriture à une même donnée.

Appels système (9) : problème d'interblocage

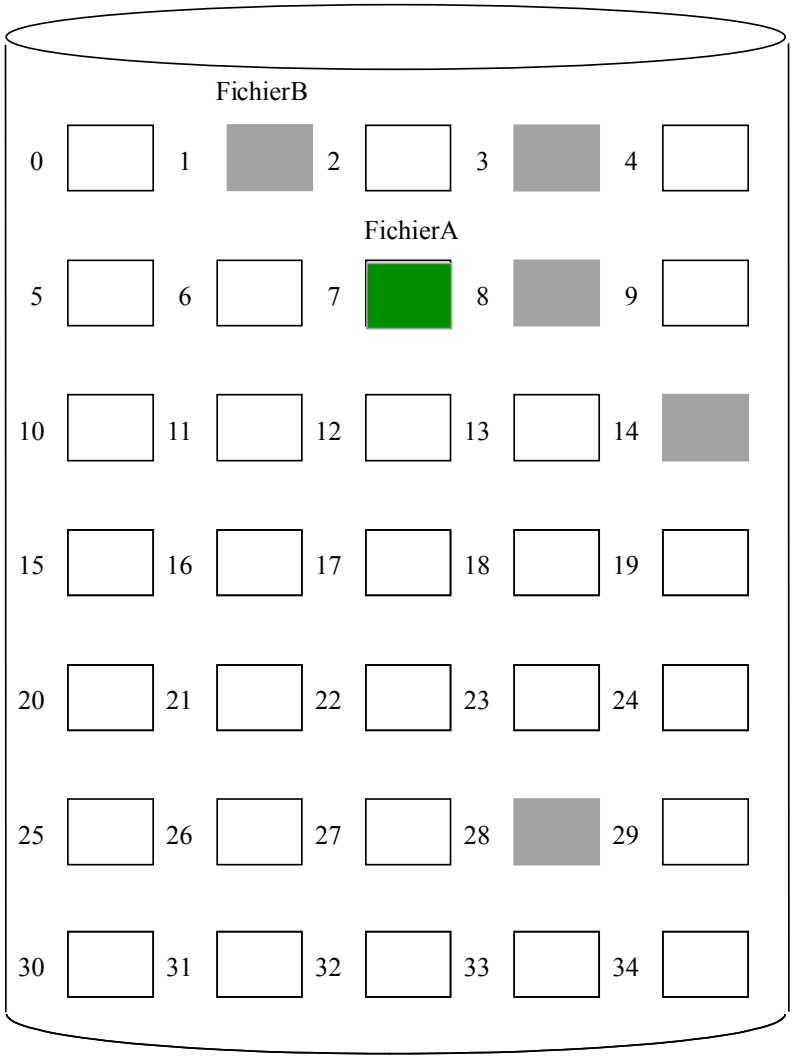
Attention :

Partage de ressources → interblocage



Appels système (10) : gestion de fichiers

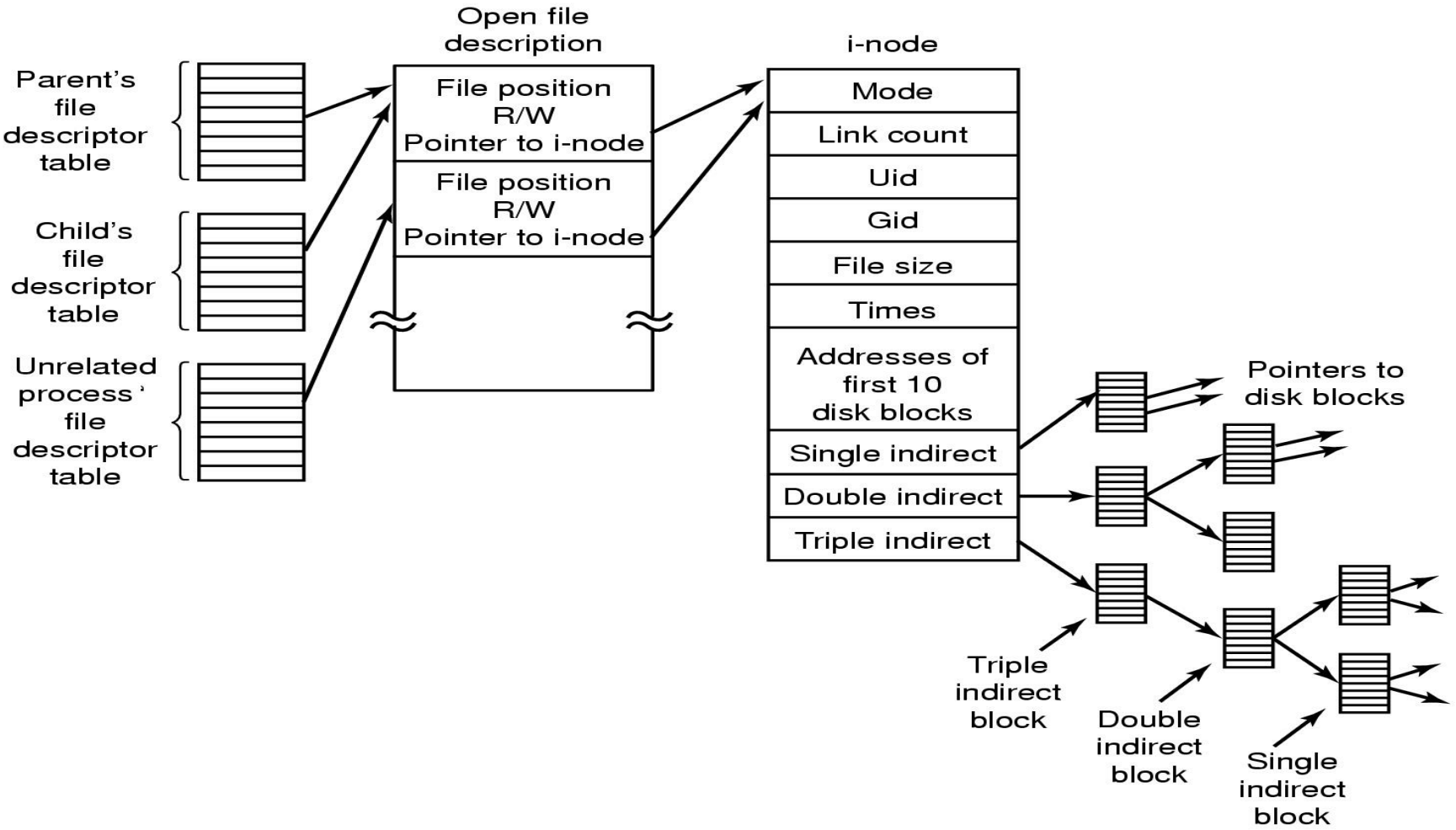
- Un fichier est un ensemble de blocs sur le disque.
- Un bloc est composé d'un nombre fixe d'octets



Nom du fichier	Blocs
Fichier A	7
...	...
Fichier B	1, 8, 3, 14, 28
...	...

Appels système (11) : système de fichier

- 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 (disque, CD, disquette, partition,…).



Appels système (12) : gestion de fichiers

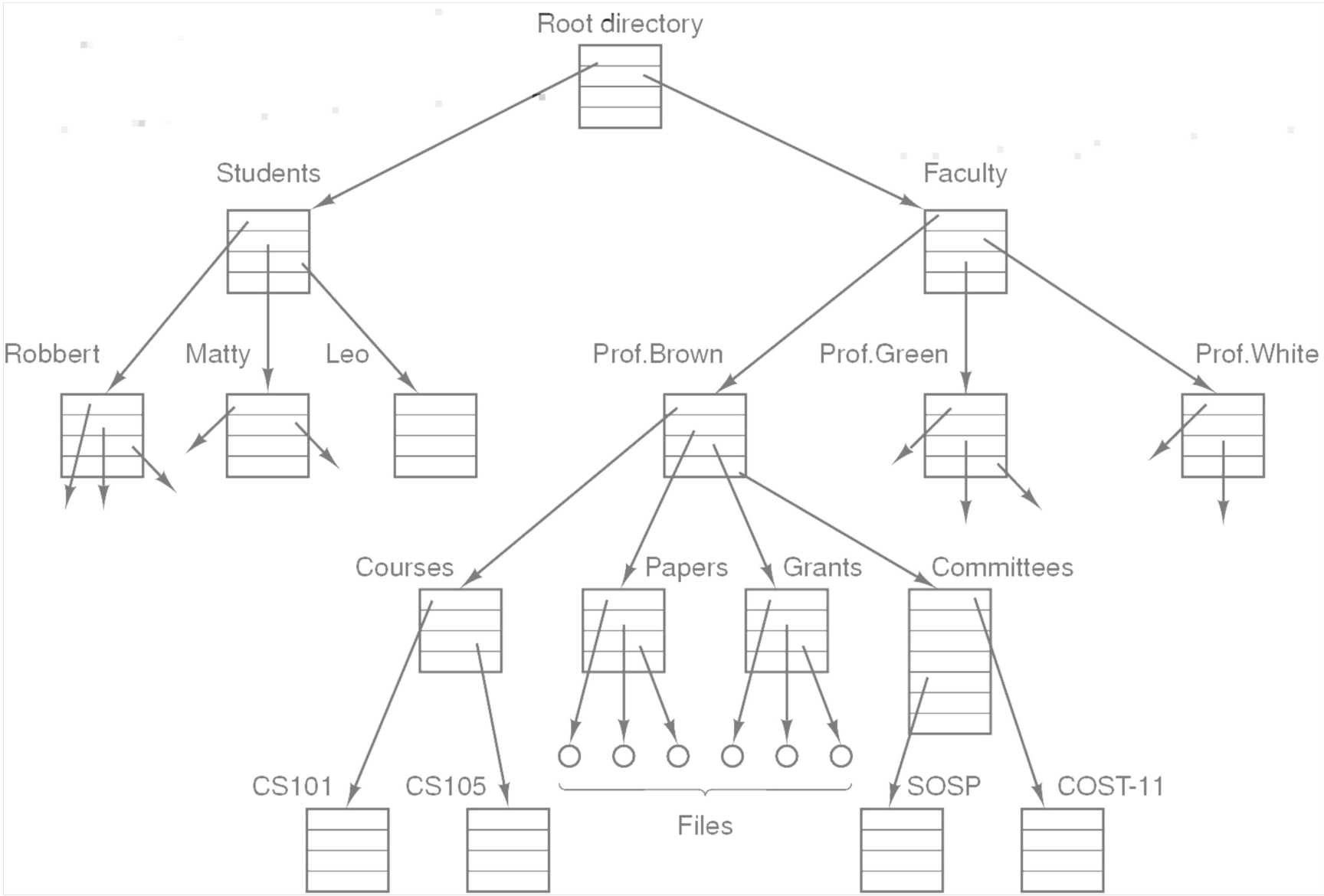
- Les appels systèmes permettent de créer des fichiers, de les supprimer, de les ouvrir, de les lire, de les modifier, de récupérer leurs attributs…

Appel système	**Description**
fd = open(path, flags, mode)	Ouvrir un fichier pour lecture ou écriture
s = close(fd)	Fermer un fichier
n = read(fd, buffer, nbytes)	Lire d'un fichier
n = write(fd, buffer, nbytes)	Ecrire dans un fichier
position = lseek(fd, offset, how)	Changer la position courante dans le fichier (le pointeur de fichier)
s = stat(name, &buf)	Lire le statut d'un fichier

Appels système (13) : répertoires

- Les fichiers sont regroupés dans des répertoires. Un répertoire peut contenir soit des fichiers, soit d'autres répertoires (structure arborescente).
- L'accès au fichier se fait en spécifiant le chemin d'accès (la liste des répertoires à traverser pour accéder au fichier).
- Un chemin d'accès est absolu si le point de départ est le répertoire racine.
- Un chemin d'accès est relatif si le point de départ est le répertoire courant.

Appels système (14) : exemple de répertoires



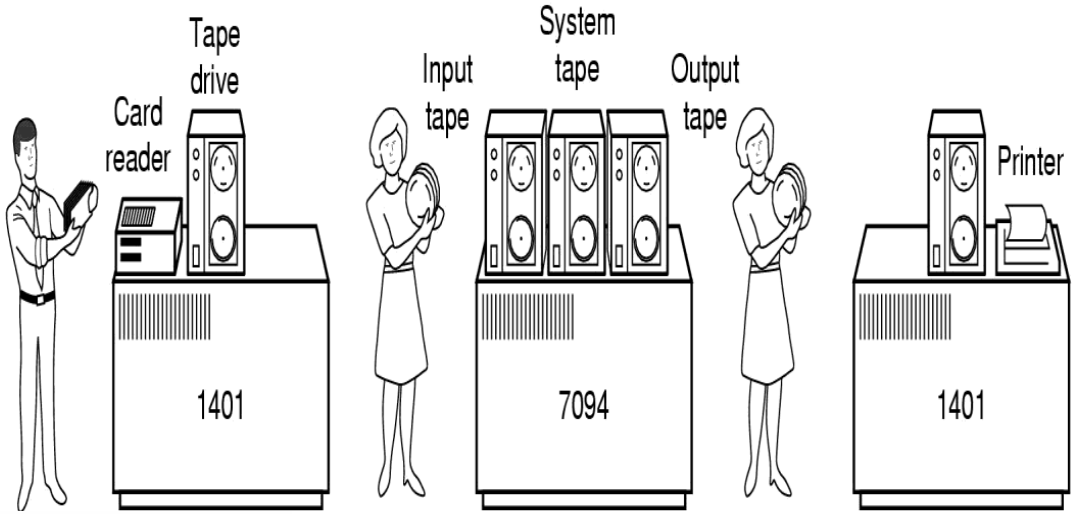
Appels système (15) : gestion de répertoires

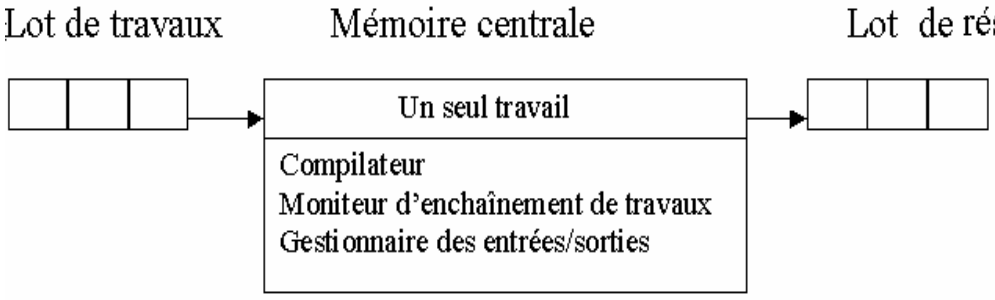
Appel système	Description
s = mkdir(name, mode)	Créer un nouveau répertoire
s = rmdir(name)	Enlever un répertoire vide
s = link(name1, name2)	Créer un nouveau lien vers un fichier
s = unlink(name)	Enlever un lien vers un fichier
s = mount(special, name, flag)	Ajouter un système de fichier à l'arbre
s = umount(special)	Retirer un système de fichier de l'arbre

Évolution du mode d’exploitation

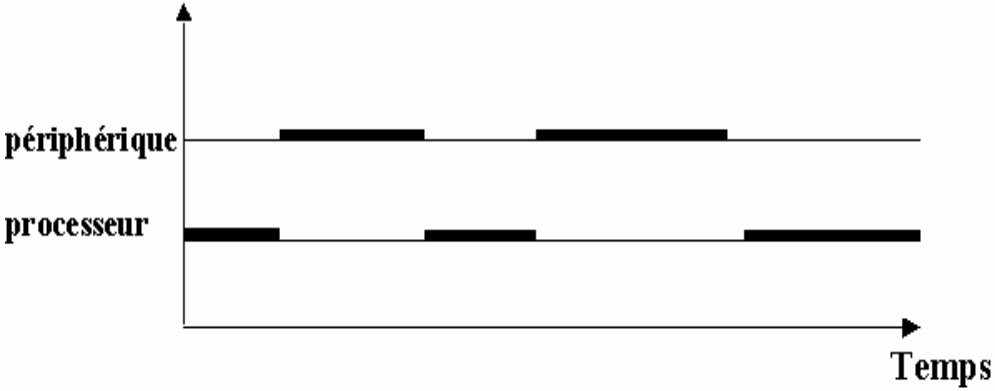
Traitement par lots (transistors, 1955-1965)

- Les programmes étaient écrits **en Fortran ou en assembleur** sur des cartes perforées.
- Ce mode d'exploitation nécessitait deux types de machines dont la plus puissante était réservée aux calculs et l'autre, moins chère, s'occupaient des périphériques





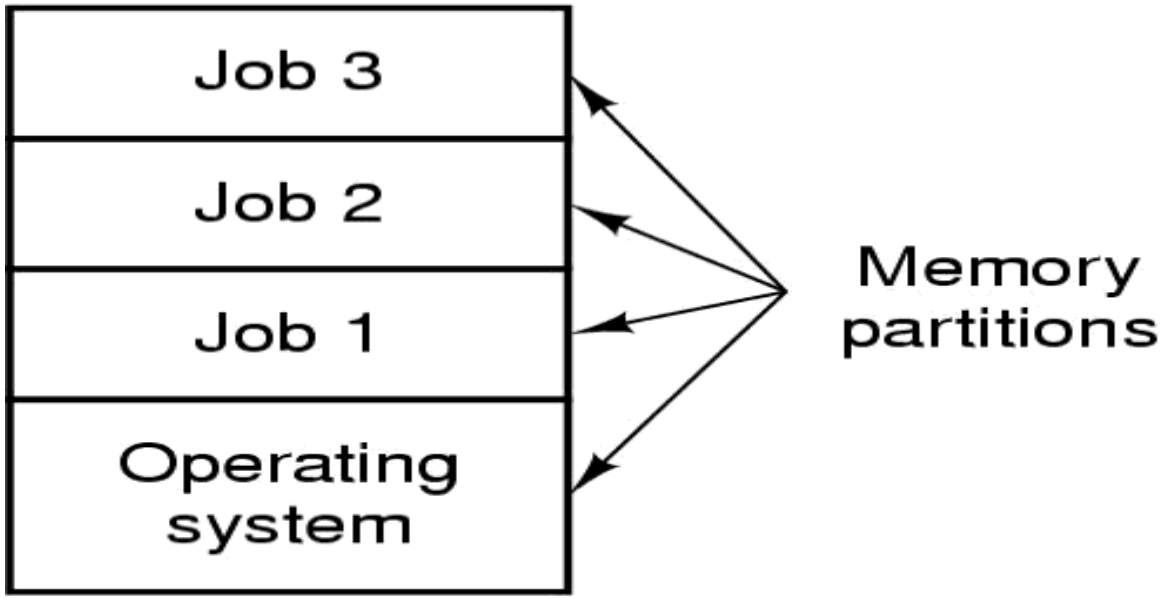
Organisation de la mémoire



Comment maximiser le taux d'utilisation du processeur ?

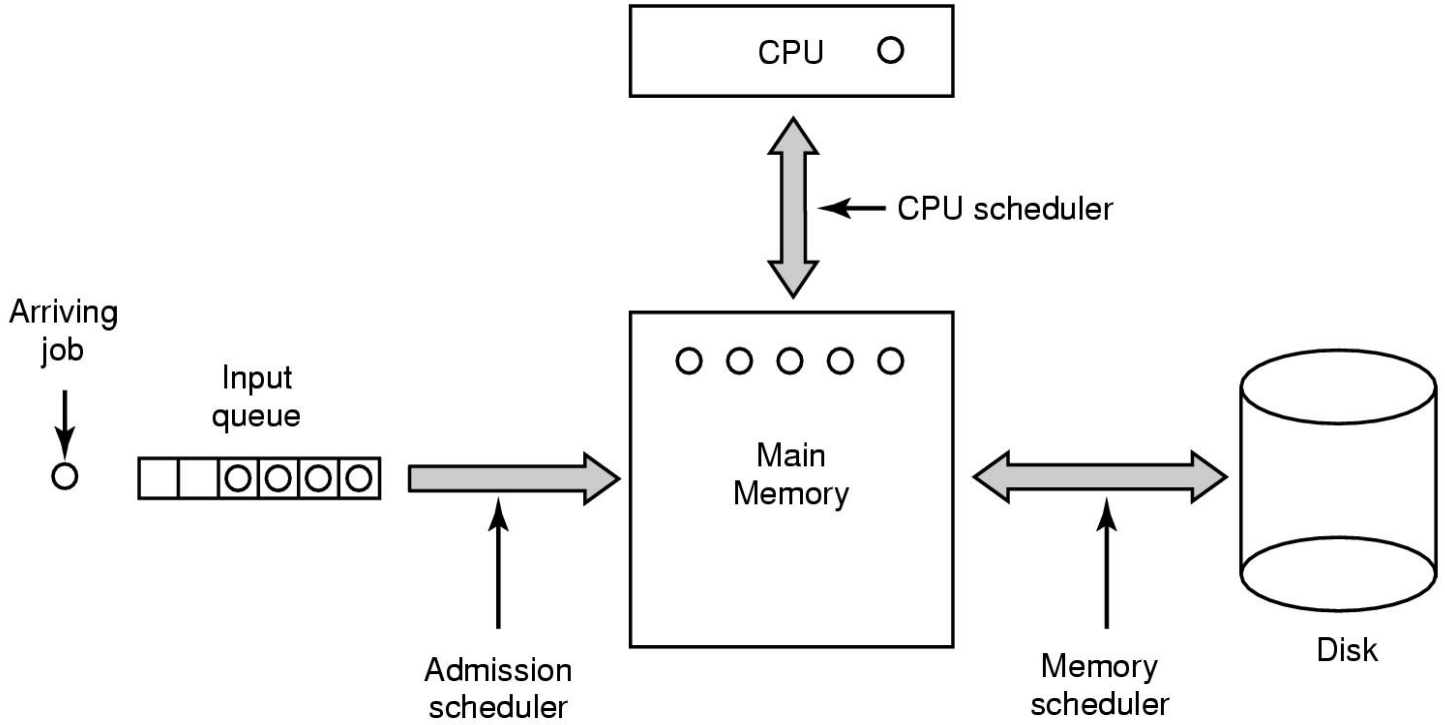
Multiprogrammation (circuits intégrés, 1965-1980)

- Introduction des unités de disque permettant un accès direct.
- Transfert des travaux vers le disque dès leur arrivée dans la salle machine (spool).
- La mémoire est organisée en un ensemble de partitions (1 travail/partition).
- Le système d'exploitation conserve en mémoire plusieurs travaux et gère le partage du processeur central et des périphériques entre les différents travaux chargés en mémoire (la **multiprogrammation**):



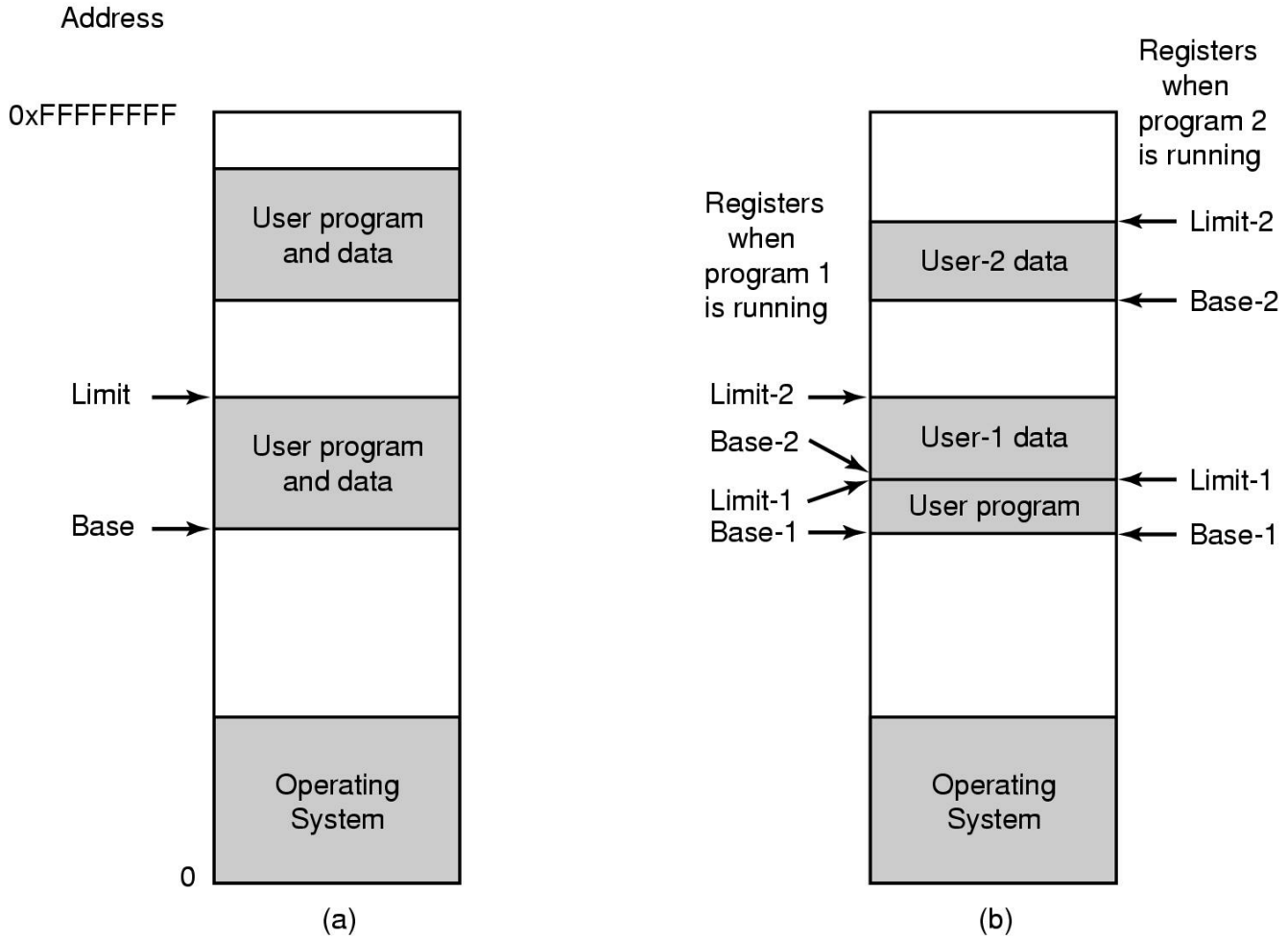
Multiprogrammation (2)

- Quelques travaux (i.e. 3) de la queue de travaux sont chargés en mémoire.
- Allocation du processeur à un travail (premier arrivé, premier servi).
- Lorsque le travail demande une E/S, le processeur est alloué à un autre travail en mémoire.
- A la fin de l'E/S, une interruption se produit et le système d'exploitation reprend le contrôle pour traiter l'interruption et lancer ou reprendre l'exécution d'un travail.
- Dès qu'un travail se termine, le système d'exploitation peut lancer le chargement, à partir du disque, d'un nouveau travail dans la partition libérée.

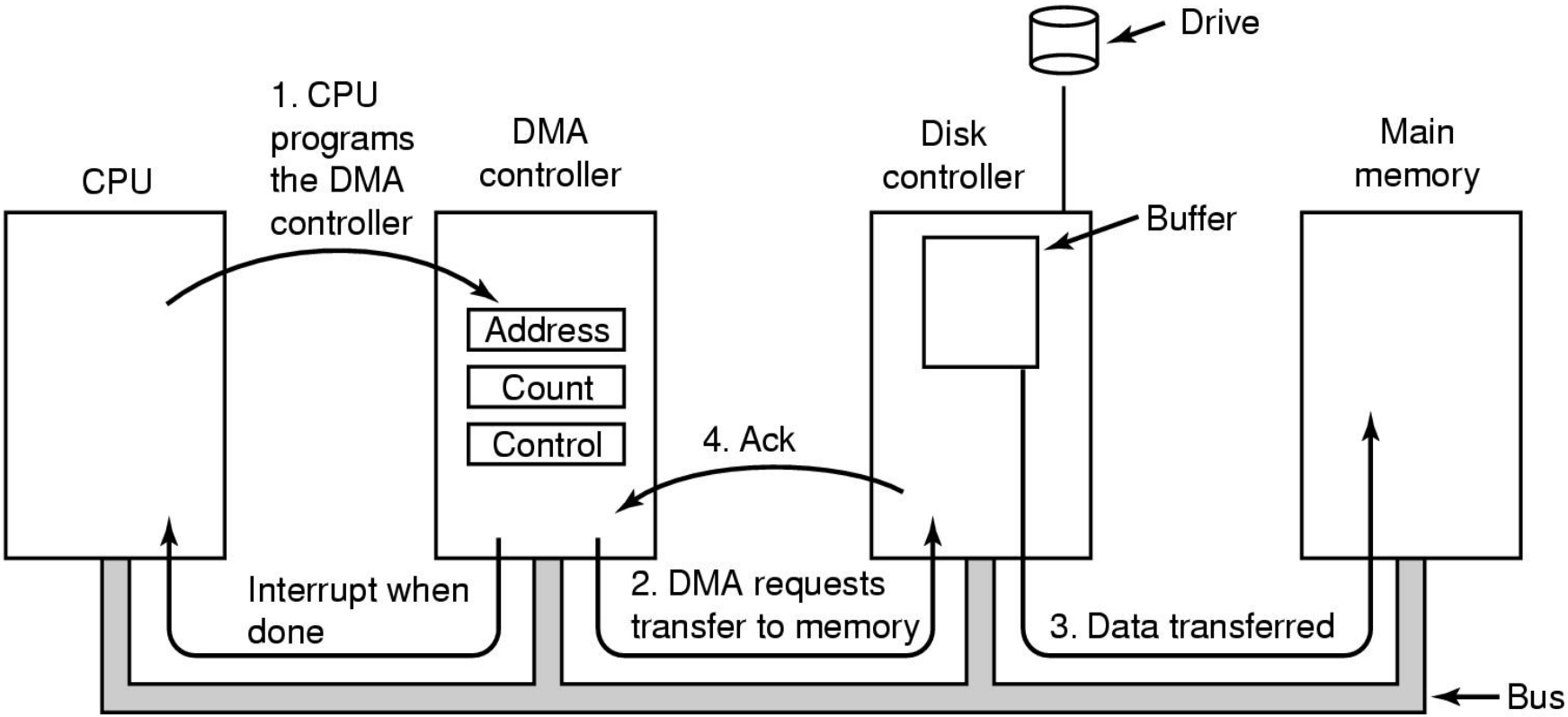


Multiprogrammation (3)

- La multiprogrammation nécessite des circuits de contrôle pour protéger chaque travail contre les intrusions et les erreurs des autres (avec possibilité de partage de programmes).

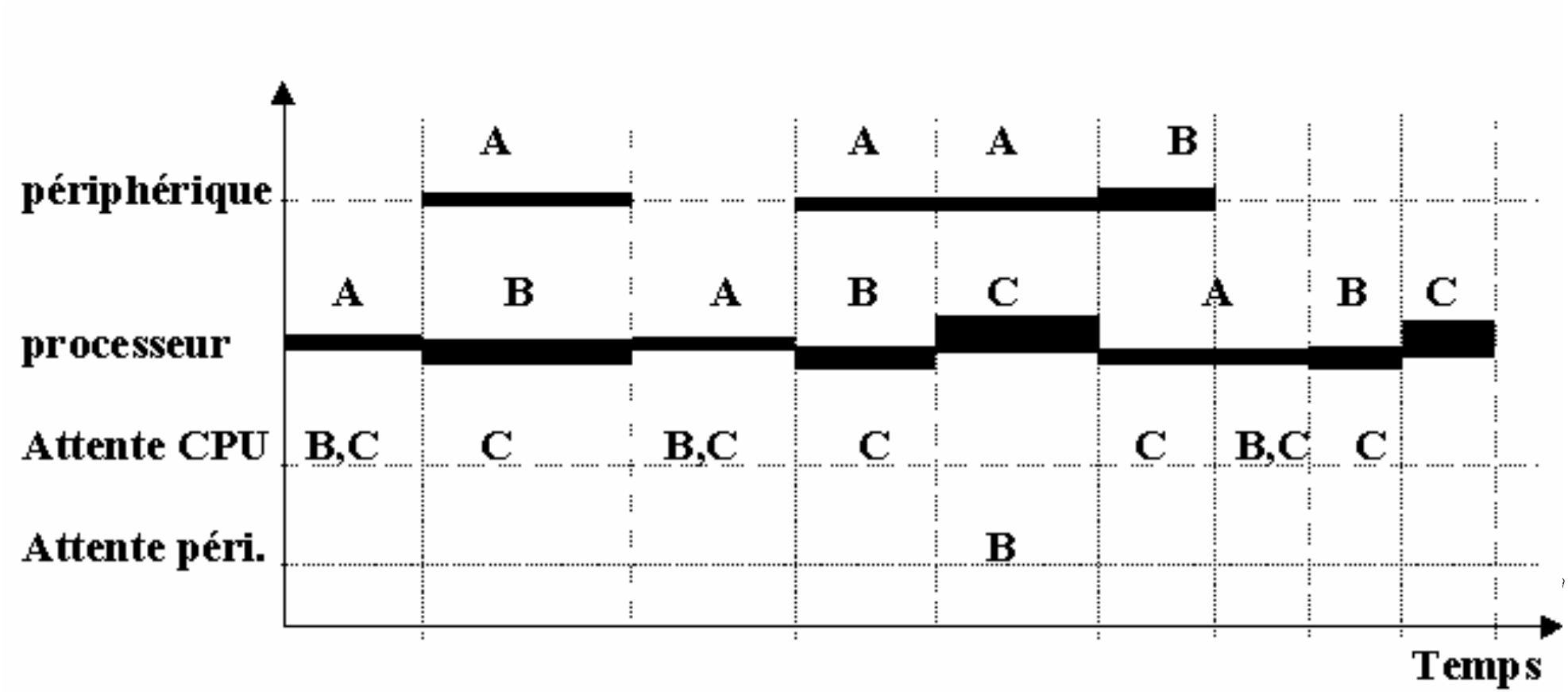


Multiprogrammation et DMA



Multiprogrammation et DMA (2)

Exemple 1 :

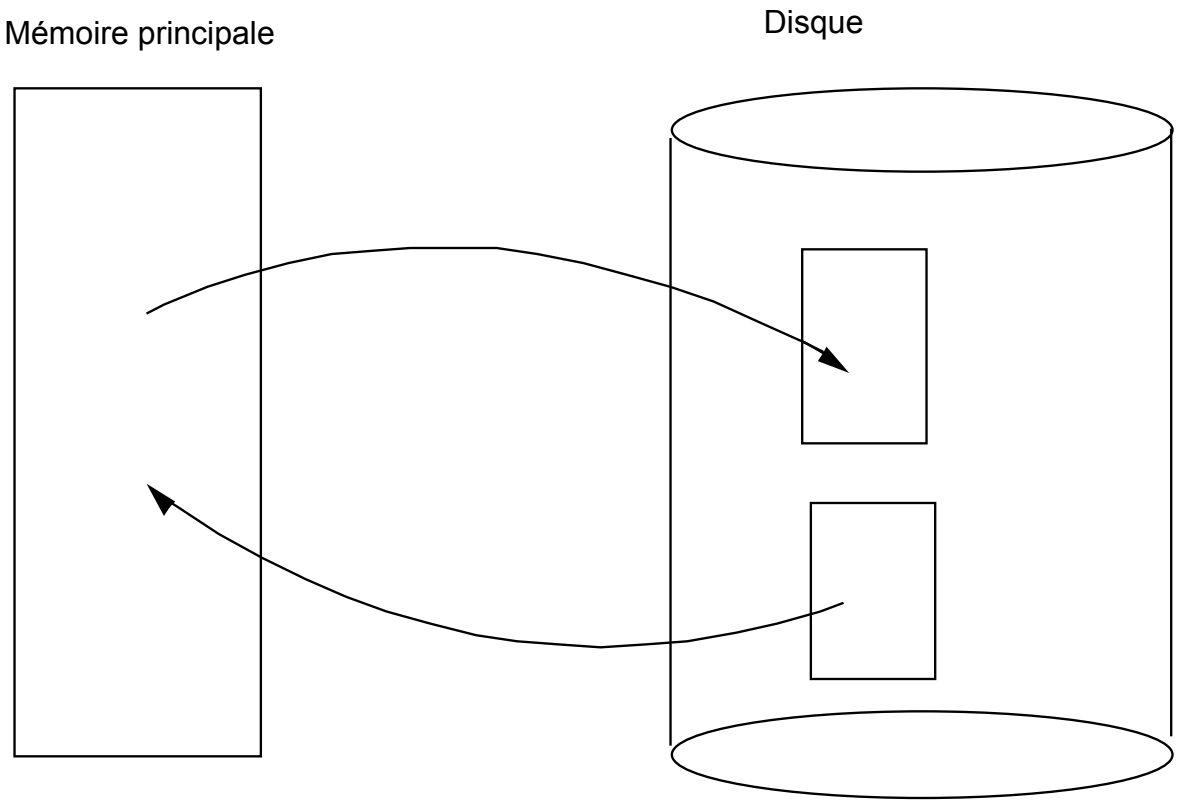


Problème:

si tous les travaux en mémoire sont en attente d'E/S, le processeur est inactif.

Va-et-vient (swapping)

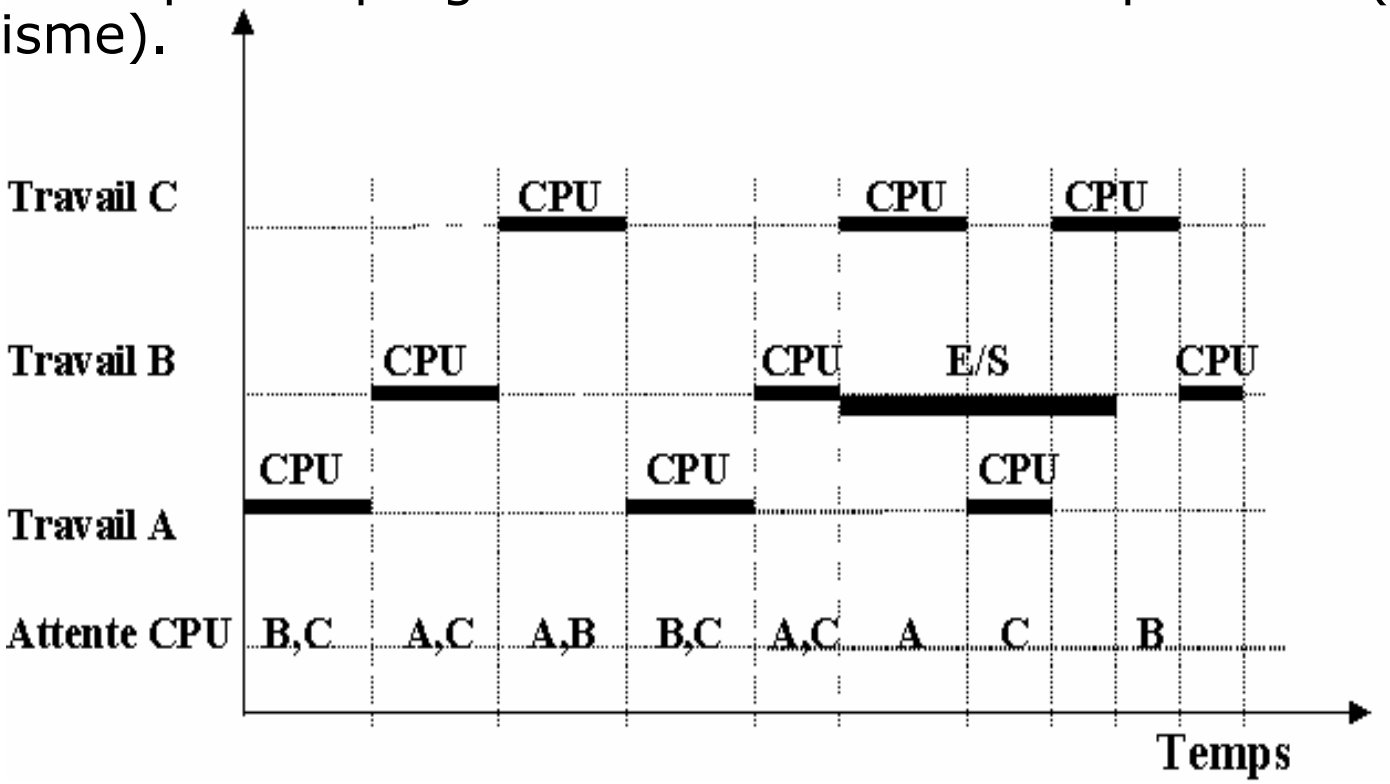
- Les travaux en mémoire qui sont en attente (d'une E/S ou d'un événement) peuvent être retirés de la mémoire pour y charger d'autres travaux prêts (en attente d'exécution).
- Ainsi durant l'exécution d'un travail, il peut subir plusieurs va-et-vient entre la mémoire et le disque (zone de swapping).



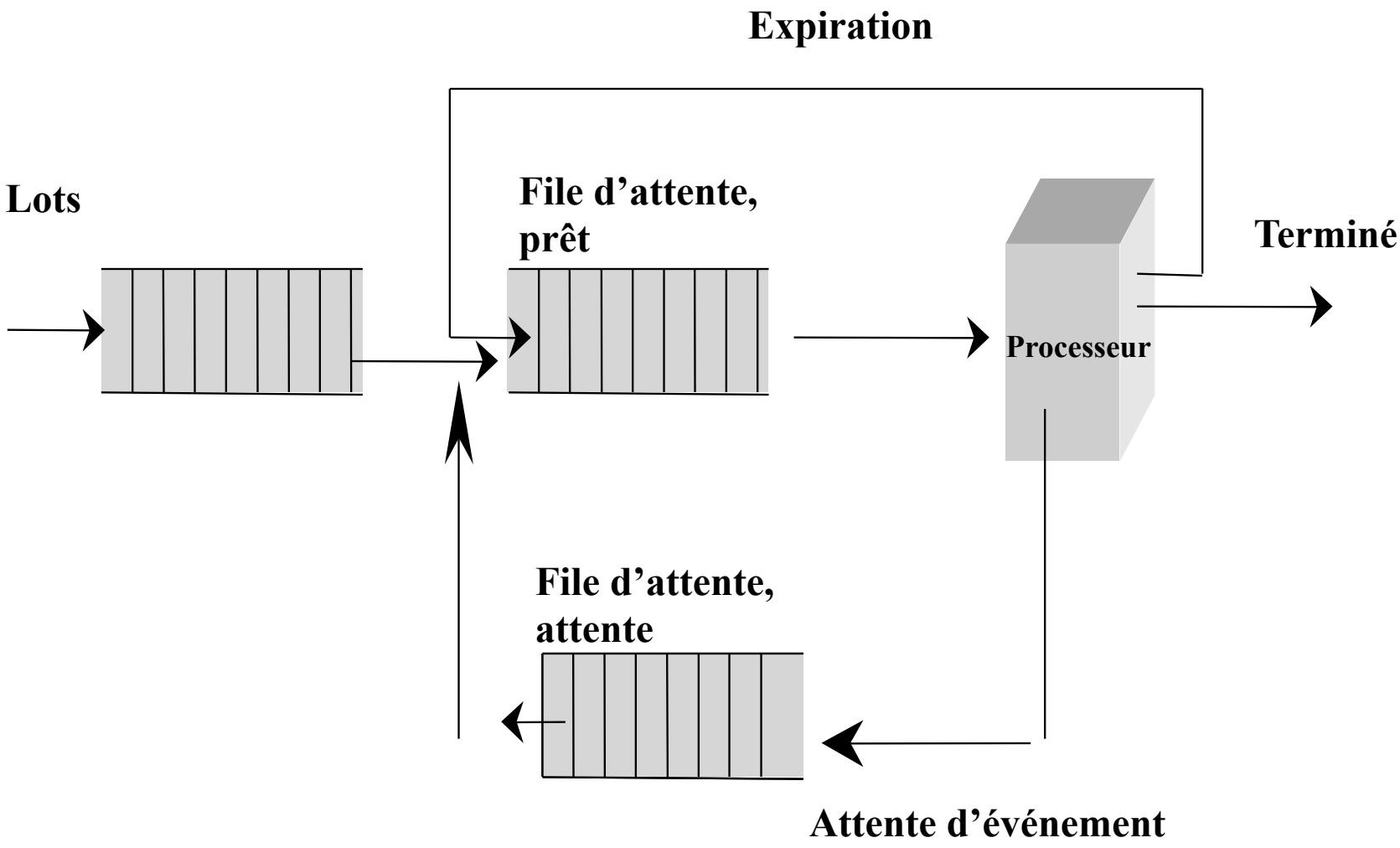
Garantir un temps de réponse acceptable à chaque utilisateur

Exécution en temps partagé

- Le processeur est alloué, à tour de rôle, pendant un certain temps à chacun des travaux en attente d'exécution. Au bout ce temps, l'exécution du travail en cours est suspendue. Le processeur est alors alloué à un autre travail.
- Si plusieurs utilisateurs lancent à partir de leurs terminaux leurs programmes simultanément, ce mode d'exploitation donne l'impression que les programmes s'exécutent en parallèle (pseudo parallélisme).



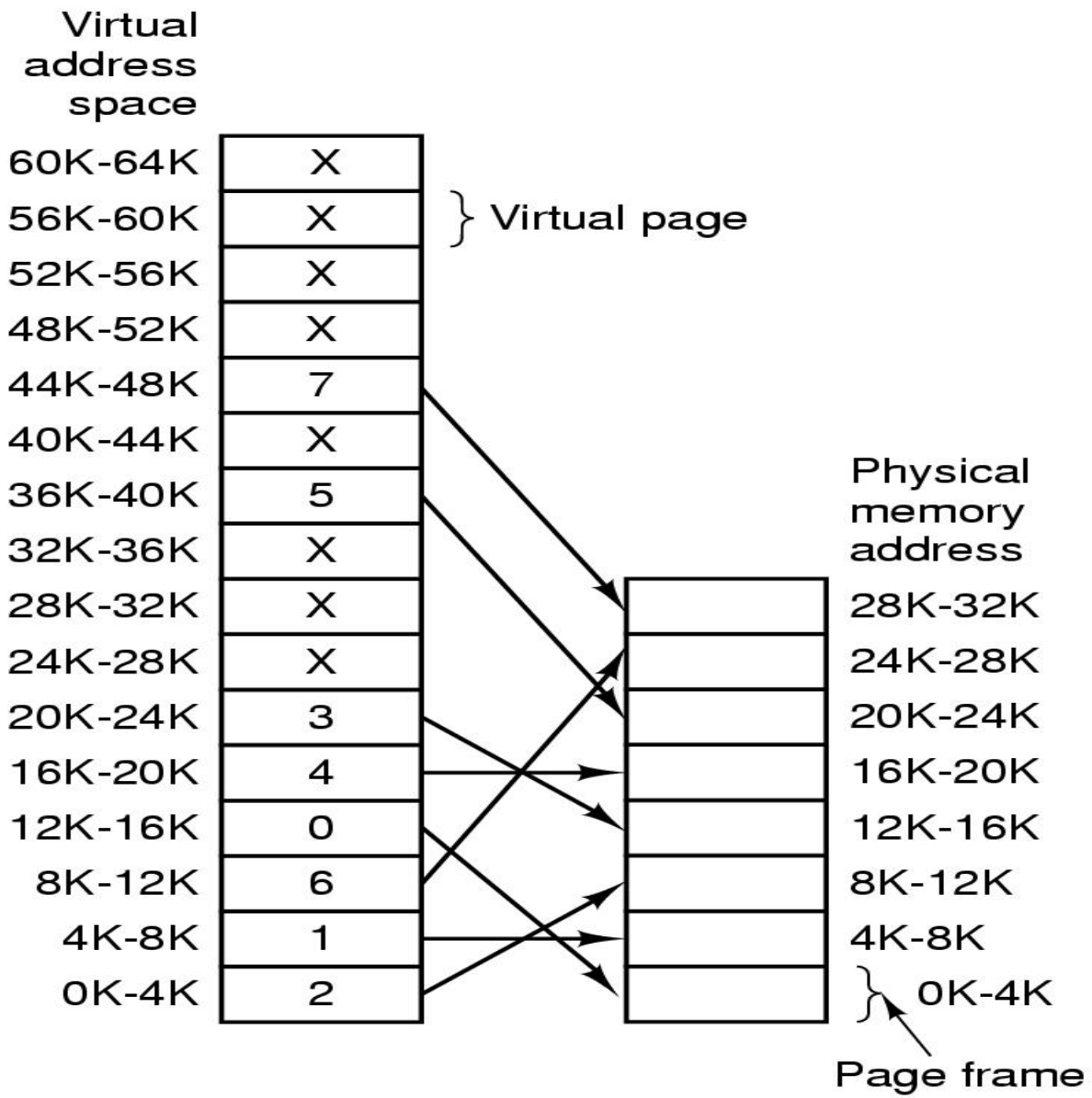
Exécution en temps partagé (2)



Exécuter des programmes dont la taille peut dépasser celle de la mémoire physique

Mémoire virtuelle

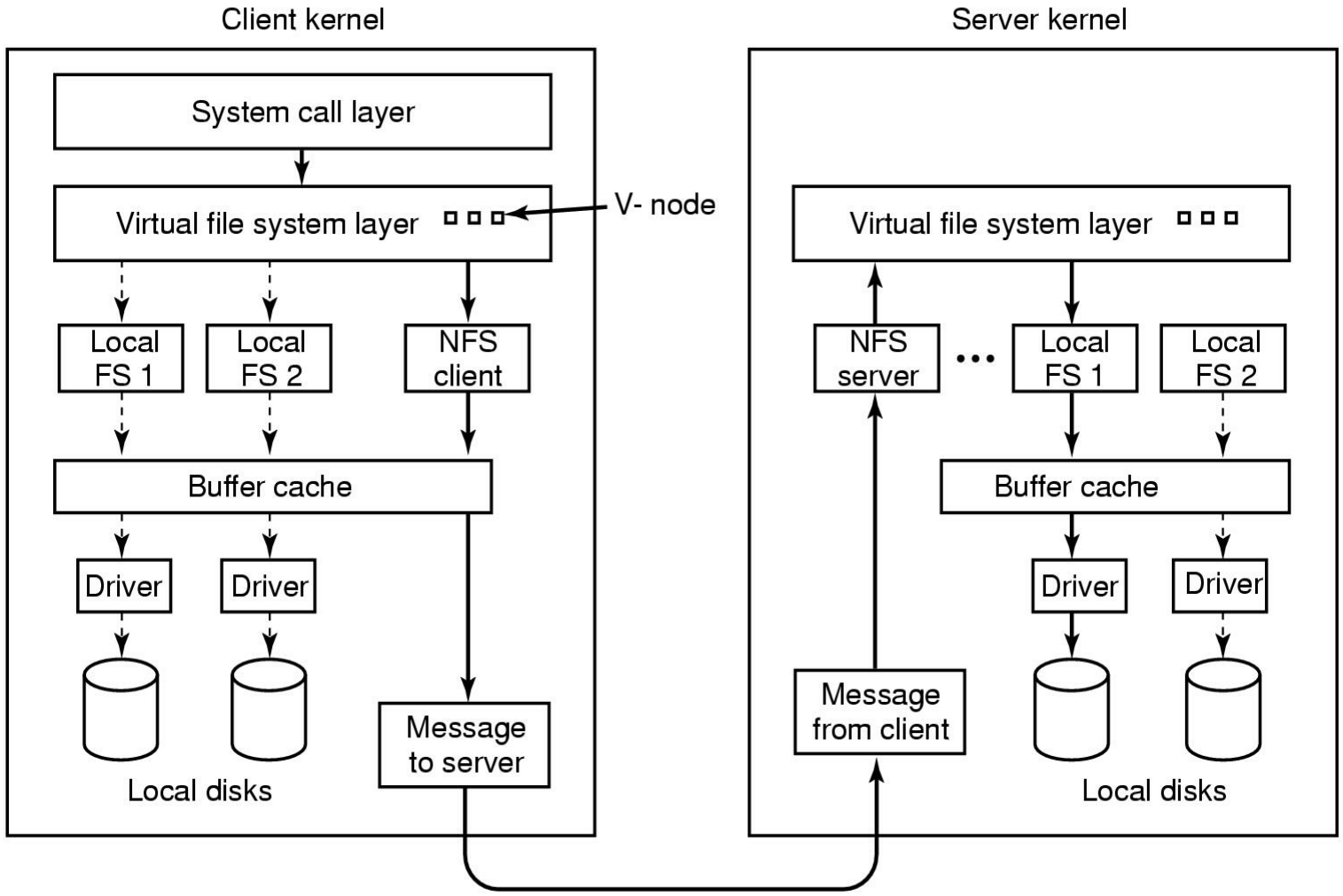
- Le compilateur génère pour chaque programme un espace d'adressage virtuel dont la taille peut surpasser celle de la mémoire physique.
- À l'exécution, une partie de cet espace virtuel est en mémoire.
- Chargement au besoin



Communiquer avec d'autres ordinateurs

Exploitation en réseau (client/serveur)

- Chaque système est doté d'une interface réseau qui lui permet de communiquer avec d'autres machines.



Exploitation distribuée & temps réel

INF8601
Systèmes informatiques parallèles
INF4404
Systèmes répartis et infonuagique
INF3610
Systèmes embarqués

Exploitation distribuée

- Les réseaux d'ordinateurs qui fonctionnent sous des systèmes d'exploitation répartis apparaissent aux yeux des utilisateurs comme une machine monoprocesseur.

- Le système d'exploitation réparti gère et contrôle l'ensemble des composants de tous les ordinateurs connectés (les processeurs, les mémoires, les disques, …).

INF3610

Systèmes d'exploitation temps réel

- Ce sont des systèmes spécialisés dans la conduite d'appareillages industriels ou dans la commande de processus où le temps joue un rôle critique (des contraintes temporelles strictes à respecter).

- L'exploitation met l'accent sur le temps de réponse (respect des contraintes temporelles imposées par l'environnement).

➔ RT-Linux, **VxWorks**, QNX et **MicroC**.

Quelques types de systèmes d'exploitation

- Mono / multi-processeur,
- Mono / multi-utilisateur,
- Mono / multi-tâche (multiprogrammation, temps partagé)
- Entrées-sorties autonomes (DMA),
- Mémoire virtuelle,
- Préemptif / non-préemptif, etc.

Quelques systèmes d'exploitation (célèbres)

- 1964 : OS/360 d'IBM 360 (1er système d'exploitation).
- 1969, : 1ère version d'UNIX (Bell Labs de la société américaine AT&T).
- 1974, CP/M (1er système d'exploitation pour micro-ordinateur ➔ MS-DOS).
- 1980, 1ère version de XENIX (UNIX pour PC) par Microsoft.
- 1981, 1er ordinateur personnel équipé de MS-DOS (IBM-Microsoft).

- **UNIX d'AT&T** dérivé de MULTICS (MIT, Bell AT&T, 1969, temps partagé, interactif et multi-utilisateur), **BSD** (1977), **MINIX, Linux** (1991), **MacOS X** (1999).

- **MS-DOS** (1981, mono-tâche, mono-utilisateur, en mode console, ordinateurs personnels) ; **Windows 3.1** (1985, multi-tâche, mono-utilisateur, interface graphique), **Windows NT** (1993, +multi-utilisateur mais un seul utilisateur simultanément), Windows 7 (2009, +multi-utilisateur), etc.

⇒ Systèmes d'exploitation écrits majoritairement en langage de haut niveau.

⇒ **POSIX** (*Portable Operating System Interface*) : normes IEEE de standardisation des interfaces de programmation des logiciels (fonctionner sur divers systèmes d'exploitation).

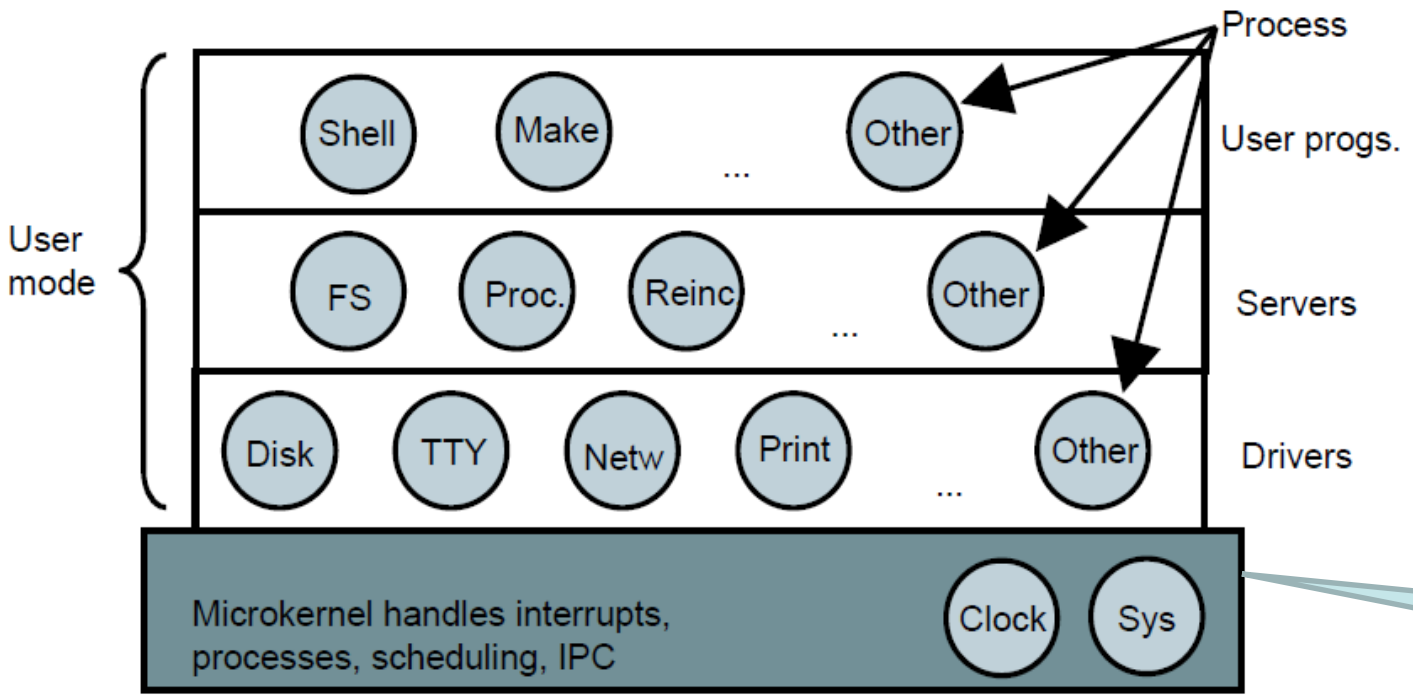
http://fr.wikipedia.org/wiki/Chronologie_informatique
https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation#Quelques_exemples

Structure des systèmes d’exploitation

Structure des systèmes d'exploitation

- **Noyau monolithique modulaire**: est un ensemble de modules chargeables dynamiquement (en mode noyau, **Linux**, **BSD**, **Solaris**).

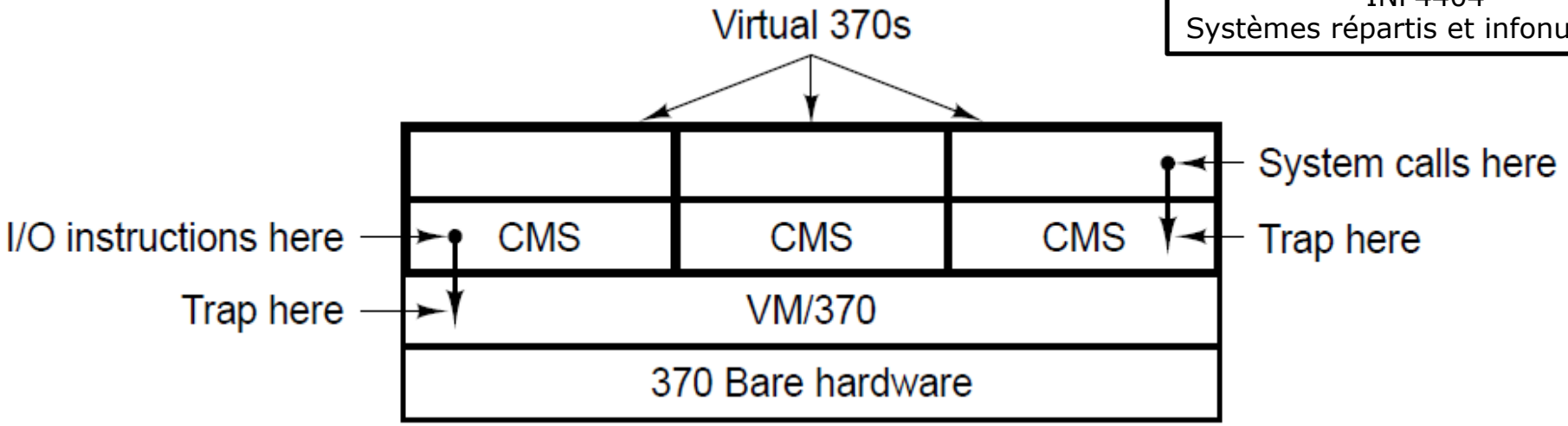
- **Micronoyau :** contient le strict minimum qui fonctionne en mode noyau **(MINIX, Mach)**.

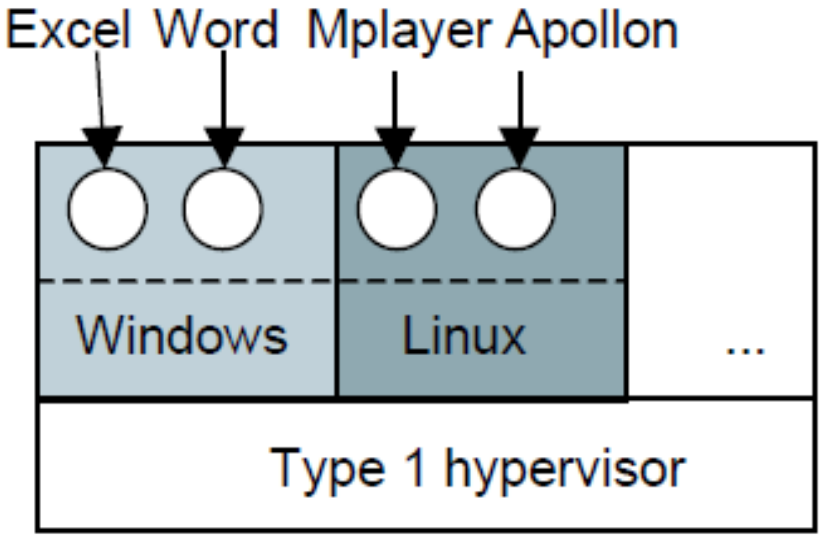


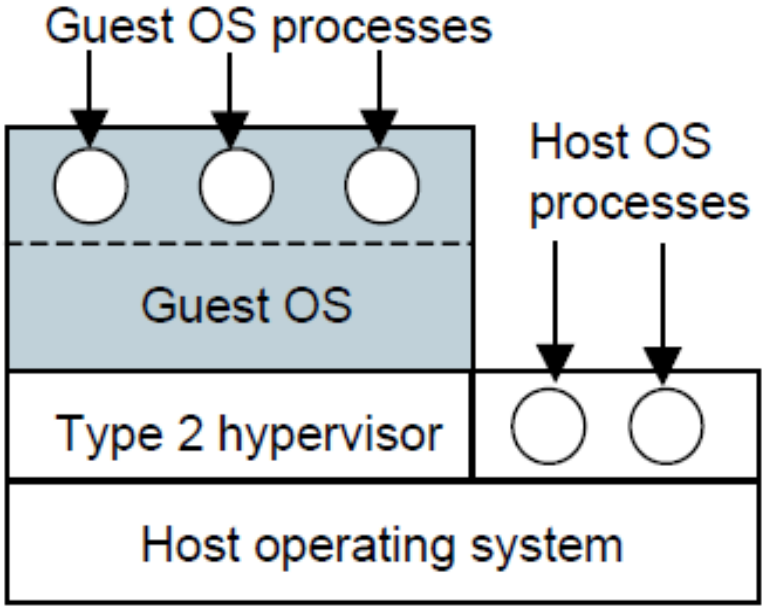
- **Noyau hybride** : est entre les deux (**Windows NT, Mac OS X** (Mach + une partie du BSD)).

Structure des systèmes d'exploitation (2)

- **Machines virtuelles**



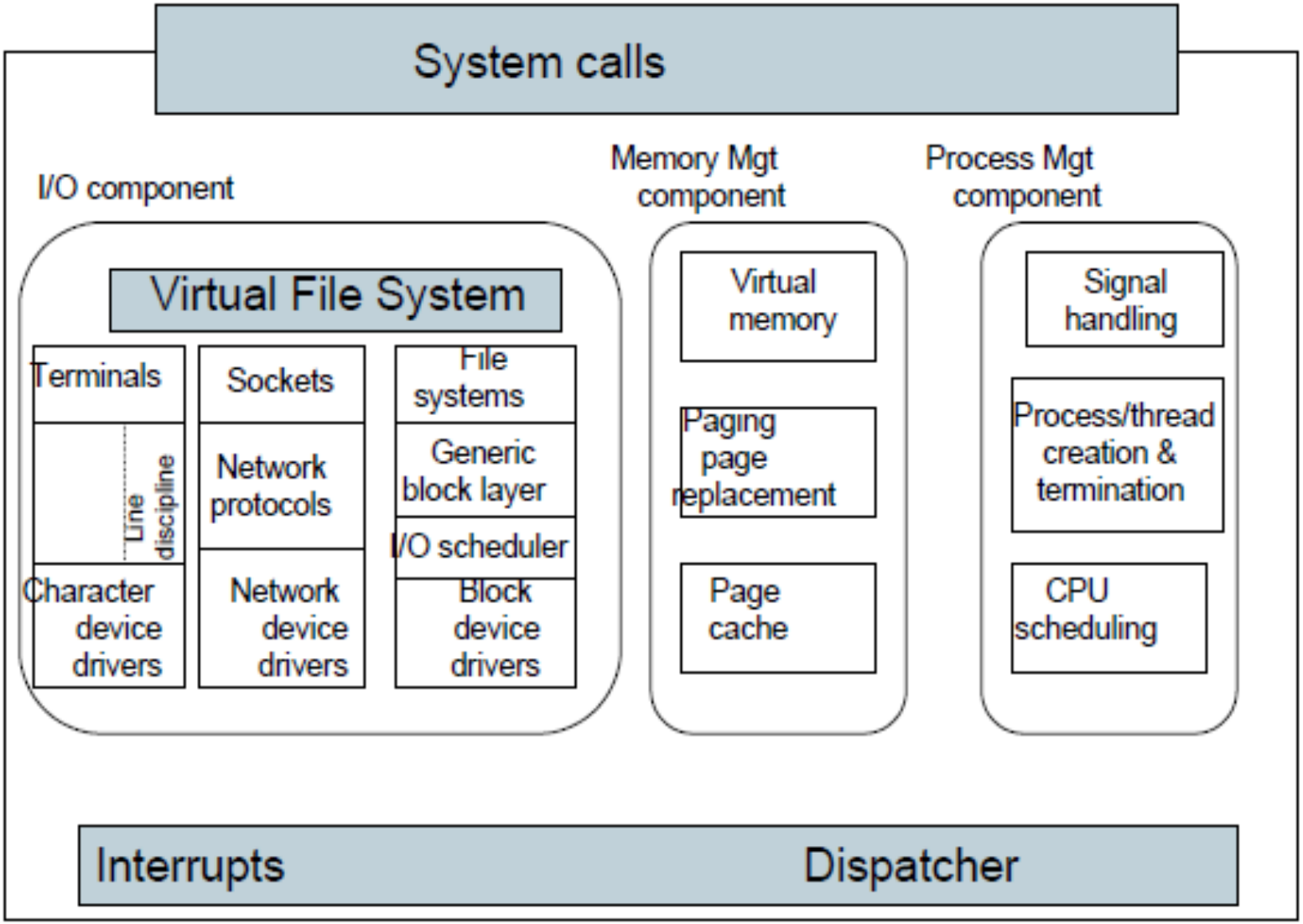




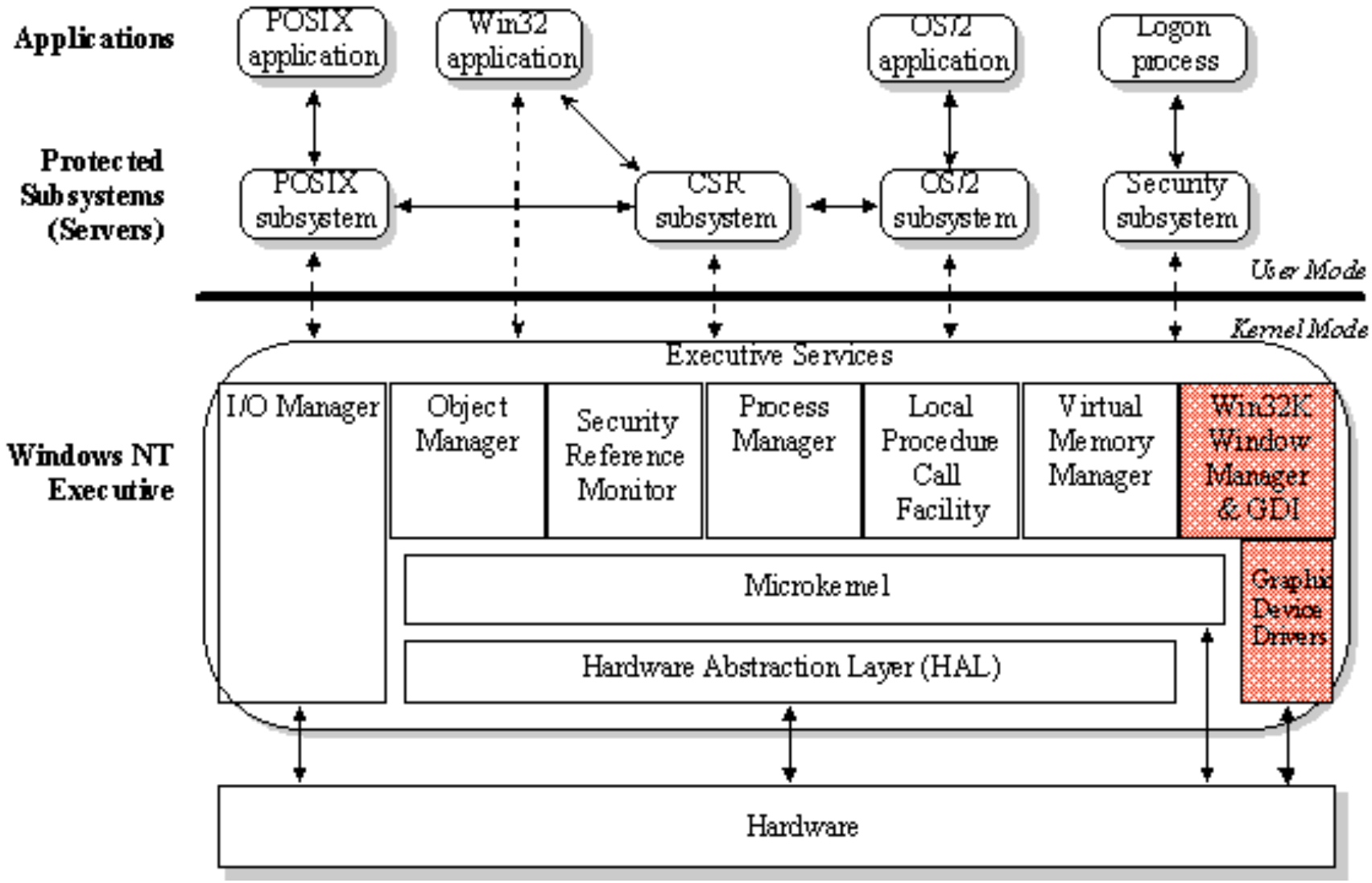
UNIX (4.4BSD kernel)

<table>
<tr><td colspan="5">System calls</td><td colspan="3">Interrupts and traps</td></tr>
<tr><td colspan="2">Terminal handing</td><td>Sockets</td><td>File naming</td><td>Map-ping</td><td>Page faults</td><td rowspan="2">Signal handling</td><td rowspan="2">Process creation and termination</td></tr>
<tr><td rowspan="2">Raw tty</td><td>Cooked tty</td><td>Network protocols</td><td>File systems</td><td colspan="2">Virtual memory</td></tr>
<tr><td>Line disciplines</td><td>Routing</td><td>Buffer cache</td><td colspan="2">Page cache</td><td colspan="2">Process scheduling</td></tr>
<tr><td colspan="2">Character devices</td><td>Network device drivers</td><td colspan="2">Disk device drivers</td><td colspan="2">Process dispatching</td></tr>
<tr><td colspan="8">Hardware</td></tr>
</table>

Linux


System calls
I/O component
Memory Mgt component
Process Mgt component
Virtual File System
Terminals
Line discipline
Character device drivers
Sockets
Network protocols
Network device drivers
File systems
Generic block layer
I/O scheduler
Block device drivers
Virtual memory
Paging page replacement
Page cache
Signal handling
Process/thread creation & termination
CPU scheduling
Interrupts
Dispatcher

Windows (NT 4.0)



https://technet.microsoft.com/en-us/library/Cc750820.f3af_big%28l=en-us%29.gif

Quelques unités de mesure

Spatiales

1 O (octet)	8 bits	1 KiO (Kilo)	2^{10} octets
1 MiO (Mega)	2^{20} octets	1 GiO (Giga)	2^{30} octets
1 TiO (Tera)	2^{40} octets	1 PiO (Peta)	2^{50} octets

Temporelles

1 s	1 seconde	1 ns (nano)	10^{-9} s
1 ms (milli)	10^{-3} s	1 ps (pico)	10^{-12} s
1 µs (micro)	10^{-6} s	1 fs (femto)	10^{-15} s

Lectures suggérées

- Notes de cours: Chapitres 1 et 2 (http://www.groupes.polymtl.ca/inf2610/documentation/notes/index.php).

- Architecture des ordinateurs (http://rmdiscala.developpez.com/cours/LesChapitres.html/Cours1/Chap1.5.htm#1.5)

Annexe : Appels système sous Linux (pp 165-187)

Patrick Cegielski "Conception de systèmes d'exploitation - Le cas Linux", 2nd edition Eyrolles, 2003.

- Un thread s'exécute en mode utilisateur et doit passer en mode noyau pour exécuter les appels système.
- Un appel système est caractérisé par un numéro, un nom et une fonction de code. Les appels système sont regroupés dans une table, appelée sys_call_table[].
- Tout appel système peut avoir jusqu'à 6 paramètres (noyau 2.6.0) et renvoie une valeur entière (-1 en cas d'erreur).
- Chaque processus (thread) a une variable globale appelée errno qui sert à récupérer les erreurs qui se produisent lors de ses appels système. Elle est mise à jour après chaque appel système ayant causé une erreur. Elle contient un code (un entier) indiquant la cause de l'erreur.

Annexe : Appels système sous Linux (2)

- L'exécution d'un appel système se fait grâce à une interruption logicielle (80h). Sa routine de service est définie en langage d'assemblage.

- Avant l'appel de l'interruption logicielle, le numéro de l'appel système doit être placé dans le registre eax et les paramètres dans d'autres registres. Le code d'erreur de retour sera placé dans le registre eax.

- Les actions exécutées par cette routine de service sont les suivantes :
  - si le numéro de la fonction est supérieur au nombre d'appels système, retourner le code d'erreur -1;
  - sauvegarder certains registres sur la pile (de telle façon que le premier paramètre de l'appel système se trouve au sommet de la pile et les autres en-dessous) ;
  - pointer les registres réservés sur le segment de données du mode noyau et le segment de données du mode utilisateur ;

Annexe : Appels système sous Linux (3)

- utiliser le numéro de l'appel système (transmis dans le registre eax) comme index dans la table sys_call_table[] , qui contient les adresses des fonctions de code des appels système, pour appeler la fonction du noyau correspondant à l'appel système ;

- au retour de cette fonction, placer le code d'erreur de cette fonction (contenu dans le registre eax) au sommet de la pile et placer le numéro de processus en cours (connu grâce à la variable globale current) dans eax ;

- traiter alors les signaux;

- retourner à l'appelant ; ce retour fait repasser le processus en mode utilisateur.

Quelques liens

- http://www.unixgarden.com/index.php/programmation/programmation-systeme-sous-unix-preliminaires
- http://msdn.microsoft.com/en-us/library/ms682658(VS.85).aspx
- http://www.dina.dk/~abraham/Linus_vs_Tanenbaum.html