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

Hiver 2019

Chapitre 1 - Concepts généraux

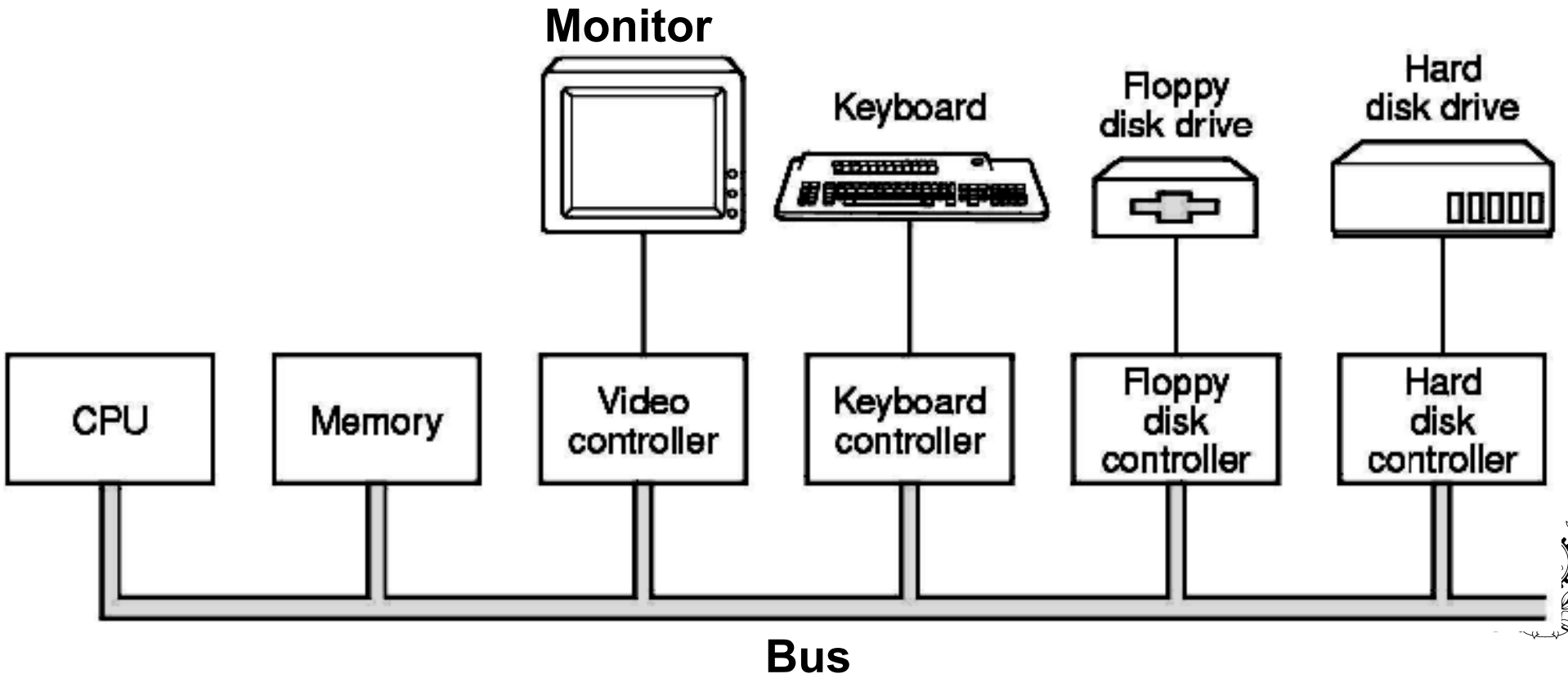
- **Qu'est ce qu'un système d'exploitation ?**
- **Concepts de base**
- **Interface avec le matériel**
- **Interactions utilisateur/système**
- **Appels système**
- **Evolution du mode d'exploitation**
- **Structure des systèmes d'exploitation**

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

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

- Les ordinateurs se composent de matériel et de logiciels.

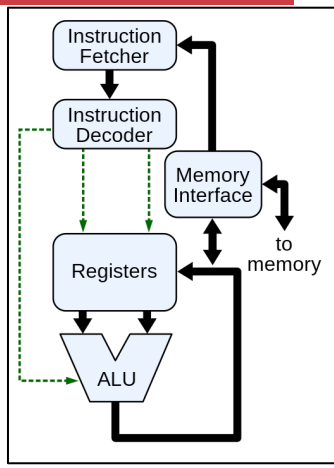
Matériel :



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

Matériel :

- Processeur (CPU): exécute les instructions (chargement, décodage, exécution). Sa puissance se mesure en Hz (nombre d'opérations par seconde).

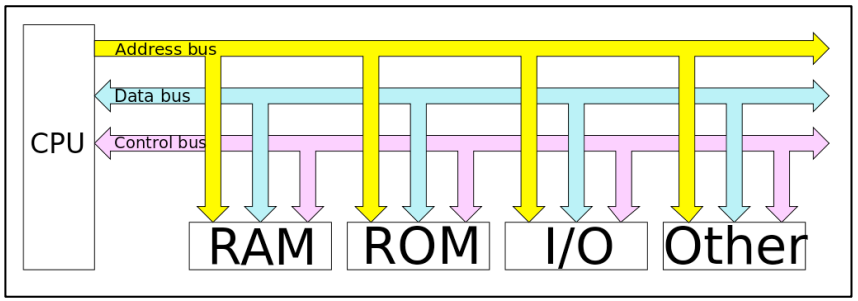


https://fr.wikipedia.org/wiki/Processeur

- Mémoire (espace de travail du CPU) : permet de stocker temporairement les instructions à exécuter et les données à traiter. Elle est organisée comme un tableau d'octets (ou cadres (frames)).

- Périphériques (clavier, souris, disques, carte réseau, etc.) : permettent au processeur de stocker et de récupérer des informations à travers le bus.

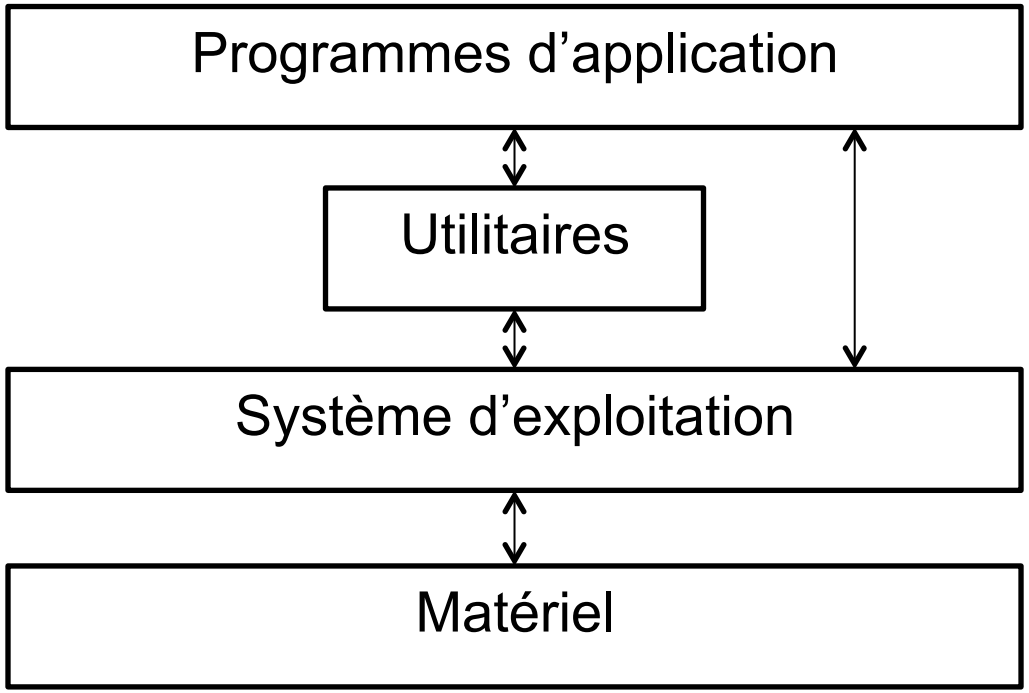
- Bus : connecte tous les périphériques d'E/S et la mémoire au processeur. Il permet le transfert de données par diffusion entre composants connectés (lignes de données, lignes d'adresse, lignes de contrôle).



http://www.wikiwand.com/fr/Bus_informatique

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

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



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

Le système d'exploitation :

- gère et contrôle les composants de l'ordinateur et
- fournit une base (une 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 ? (5)

Fonctions principales d'un système d’exploitation :

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

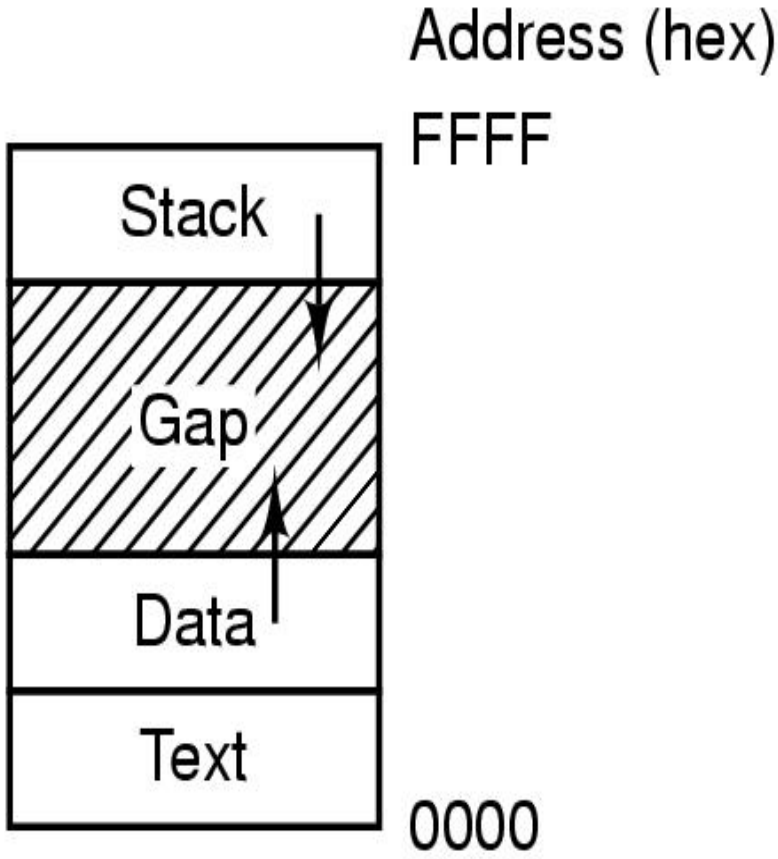
Concepts de base

- Processus : un programme en cours d'exécution (espace d'adressage, état, compteur ordinal, sommet de pile, etc.).
- Mémoires virtuelles : espaces d'adressage des processus pouvant être plus grands que la mémoire physique ➔ chargement à la demande.
- Fichiers :
  - Fichiers ordinaires : données stockées sur certains périphériques (e.g. disque).
  - Fichiers spéciaux : périphériques d'E/S (clavier, moniteur, disque, réseau, etc.). Ils permettent d'interagir avec le monde extérieur (récupérer, transmettre ou stocker des informations).

    Par exemple, la commande

    echo INF2610 > /dev/tty

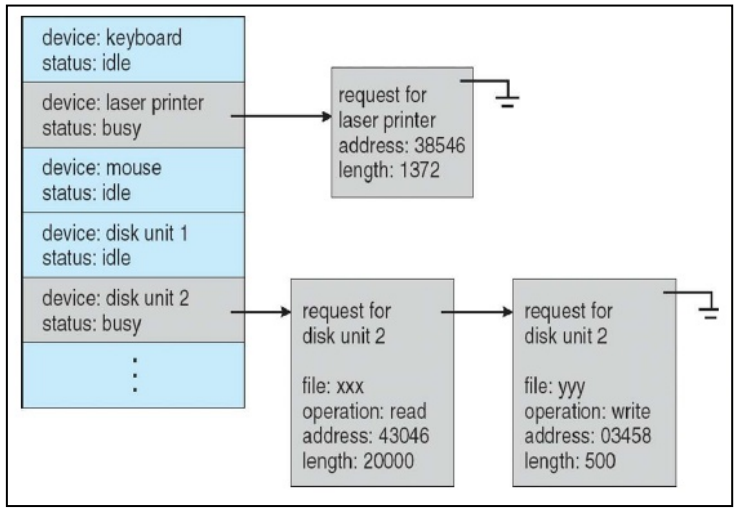
    va afficher à l'écran INF2610

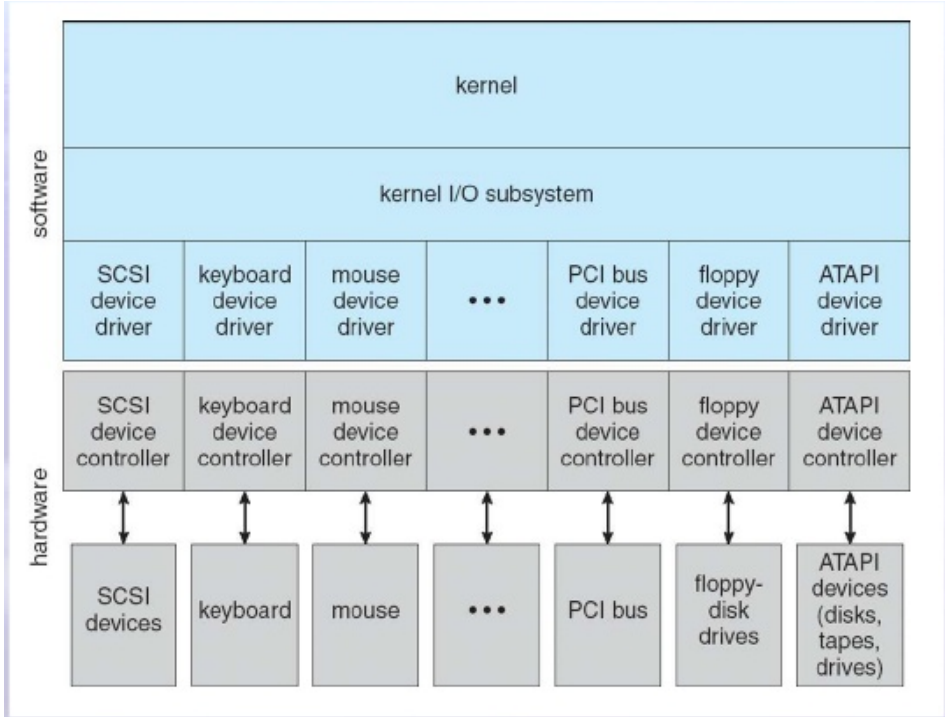


Interface avec le matériel

- Chaque composant (processeurs, mémoires, bus et périphériques d'E/S) de l'ordinateur a son propre code (câblé et/ou logiciel) qui assure son fonctionnement et les interactions avec les autres ➔ un contrôleur.

- Les pilotes (drivers) des périphériques offrent une interface entre le noyau du système d'exploitation et les périphériques. Ils permettent de commander des périphériques via des opérations simples (identiques à celles des fichiers ordinaires) : open, read, write, close, etc.





http://iips.icci.edu.iq/images/exam/Abraham-Silberschatz-Operating-System-Concepts---9th2012.12.pdf

Interface avec le matériel (2)

- Les temps d'accès à la mémoire et aux périphériques sont très lents par rapport au temps de calcul du processeur.

  ➔ Ajout de caches (CPU, E/S) pour améliorer les temps d'accès moyens.

  ➔ Exécution d'un autre processus, si le processus en cours doit attendre la fin d'une E/S.

- Pour détecter la fin de l'E/S, on distingue deux solutions :

  - **E/S par scrutation** (polling) : le système interroge régulièrement les périphériques sollicités.

  - **E/S dirigées par les interruptions** : Les périphériques utilisent le mécanisme **des interruptions pour prévenir le processeur de la fin de l'E/S** (via des lignes de bus dédiées). Le système vérifie s'il y a des interruptions en attente avant d'entamer l'exécution d'une instruction.

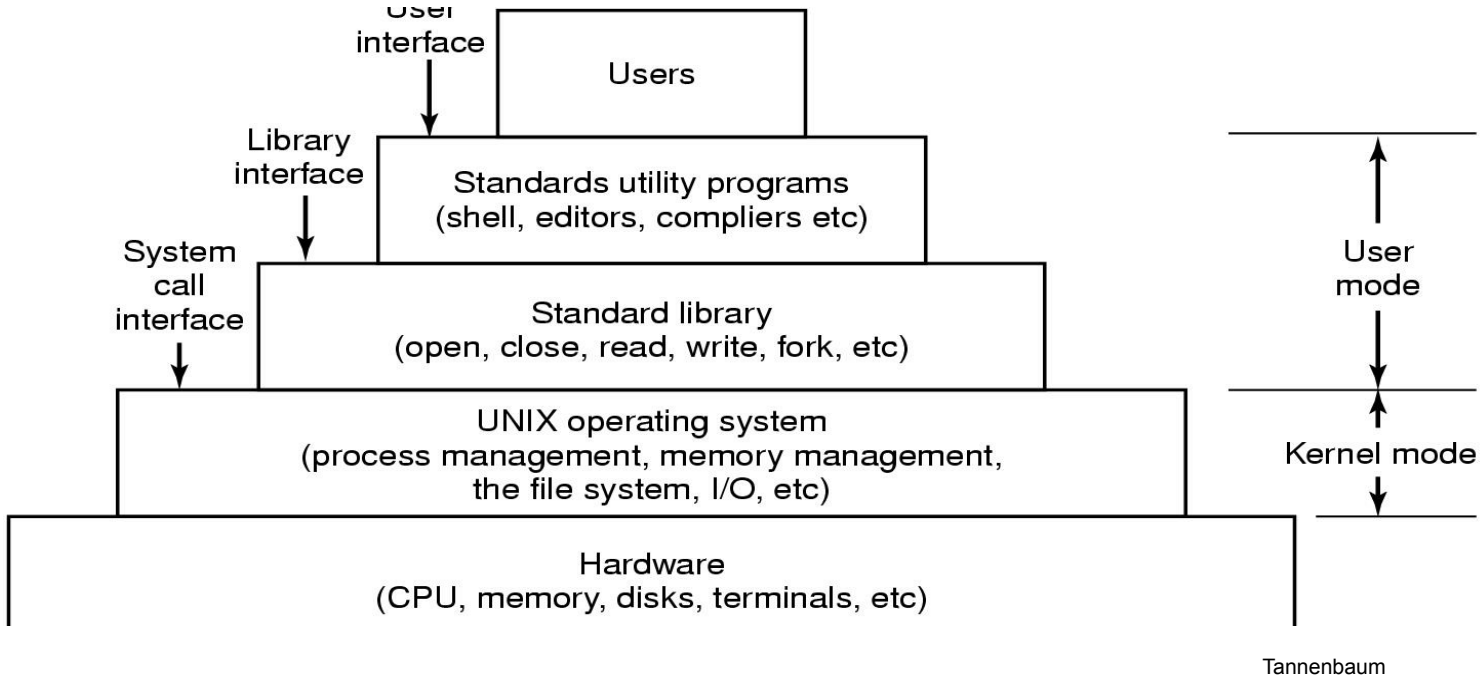
Interface avec le matériel (3)

- Les interruptions permettent au système d'exploitation de reprendre le contrôle et de réagir conséquemment à des événements :
  - Interruptions matérielles :
    - Horloges (pour limiter le temps d'allocation du processeur).
    - Périphériques (pour signaler la fin d'une E/S).
  - Interruptions logicielles (déroutements ou traps) :
    - 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 est un logiciel qui offre un ensemble de services (appels système) permettant de développer des applications sans se soucier des détails de fonctionnement et de gestion du matériel.

- **POSIX** (*Portable Operating System Interface*) : normes IEEE de standardisation des interfaces de programmation système basées sur le système d'exploitation Unix.



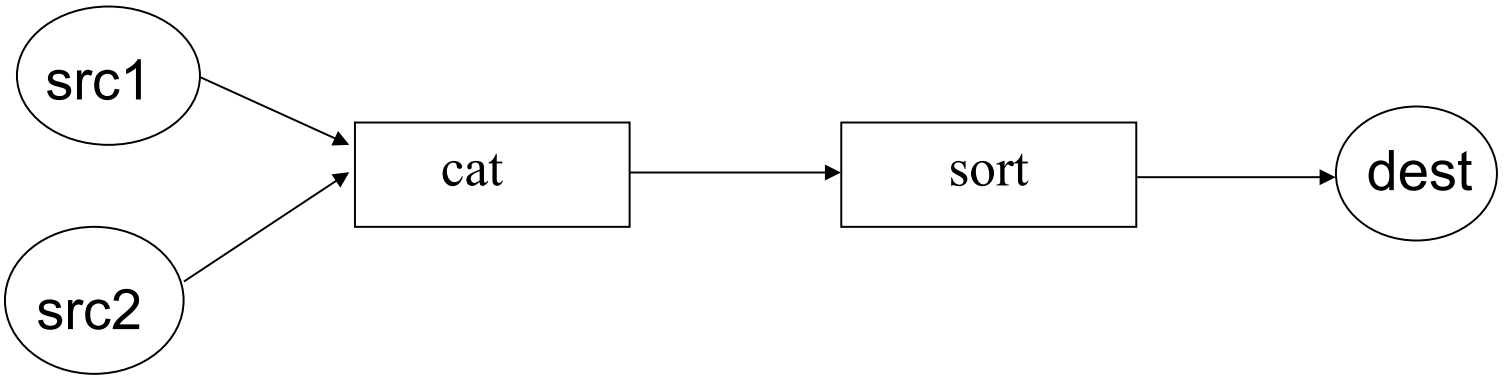
- Les appels système peuvent être invoqués via un interpréteur de commandes, une interface graphique, des utilitaires ou encore des programmes d'application.

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'utilitaires (compilateurs, éditeurs de lien, etc.),
  - affiche les résultats ou les erreurs puis se met en attente de la commande suivante, et ainsi de suite.

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

cat src1 src2 | sort >dest



Interactions utilisateur/système (3) : 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"
    elif [ $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
jupiter% ./script1
...
```

Interactions utilisateur/système (4) : 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, read
#include <fcntl.h>  // O_CREAT, O_WRONLY, O_TRUNC
#define taille 80
int main ( )
{ int fd , nbcar;
  char buf[taille] ;
  //créer/ouvrir un fichier
  fd = open("fich", O_CREAT| O_WRONLY | O_TRUNC);
  if(fd==-1)
  { write(2, "Erreur d'ouverture \n", 20);

    return -1 ;
  }
  write(1, "Ouverture avec succès \n" , 23);

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

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

Appels système

https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux#Making_a_syscall

- En général, les processeurs ont deux modes de fonctionnement :
  - le mode noyau (superviseur 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.

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

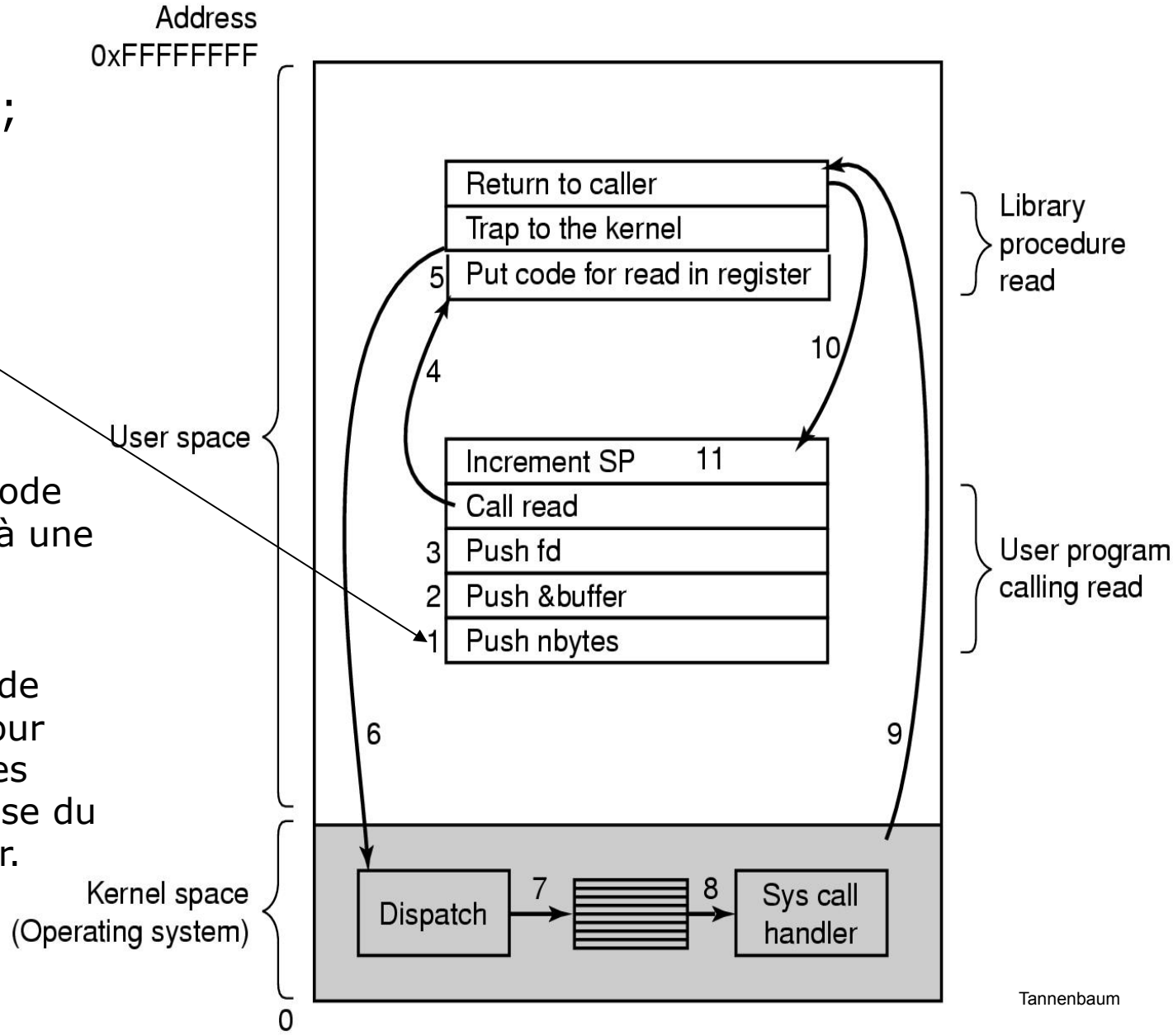
- Un appel système consiste en une interruption logicielle qui a pour rôle d'activer le système d'exploitation :
  - changer le mode d'exécution pour passer du mode utilisateur au mode noyau (exemple, sous Linux: int ox80 ou syscall);
  - récupérer les paramètres et vérifier la validité de l'appel;
  - exécuter la fonction demandée;
  - récupérer la (les) valeur(s) de retour;
  - retourner au programme appelant avec retour au mode utilisateur.

Appels système (2) : exemple de lecture d'un fichier

read(fd,buffer,nbytes);

Étape 6 : bascule vers le mode noyau puis se branche à une adresse fixée du noyau

Étape 7 : utilise le numéro de l'appel système read pour récupérer de la table des appels système, l'adresse du code de read à executer.



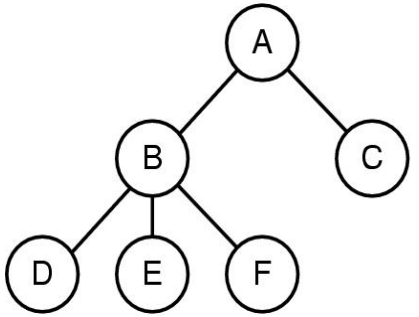
Appels système (3) : POSIX et Win32 (Windows)

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	Récupérer les attributs d'un fichier
mkdir	CreateDirectory	Créer un répertoire
rmdir	RemoveDirectory	Supprimer un répertoire vide
link		Ajouter un lien physique vers un fichier
unlink	DeleteFile	Supprimer un lien physique vers un fichier
mount		Ajouter un système de fichiers à l'arborescence de fichiers
umount		Retirer un système de fichiers
chdir	SetCurrentDirectory	Changer de répertoire de travail (courant)
chmod		Changer les permissions d'accès d'un fichier
kill		Envoyer un signal à un processus
time	GetLocalTime	Obtenir le temps courant

man syscalls

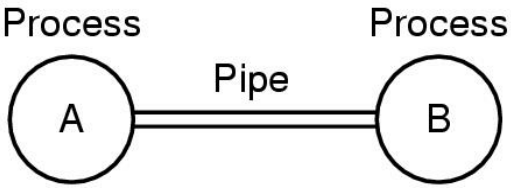
Appels système (4) : gestion de processus

- Les appels système permettent, notamment, :

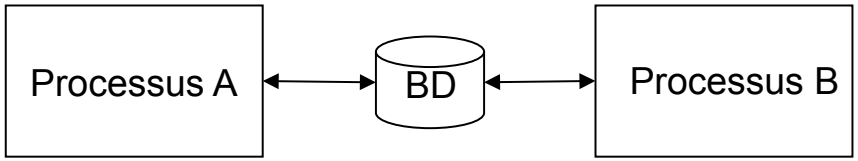


– la création, la terminaison de processus,

– la communication interprocessus (segments de données partagés, fichiers, tubes de communication (pipes), etc.), et

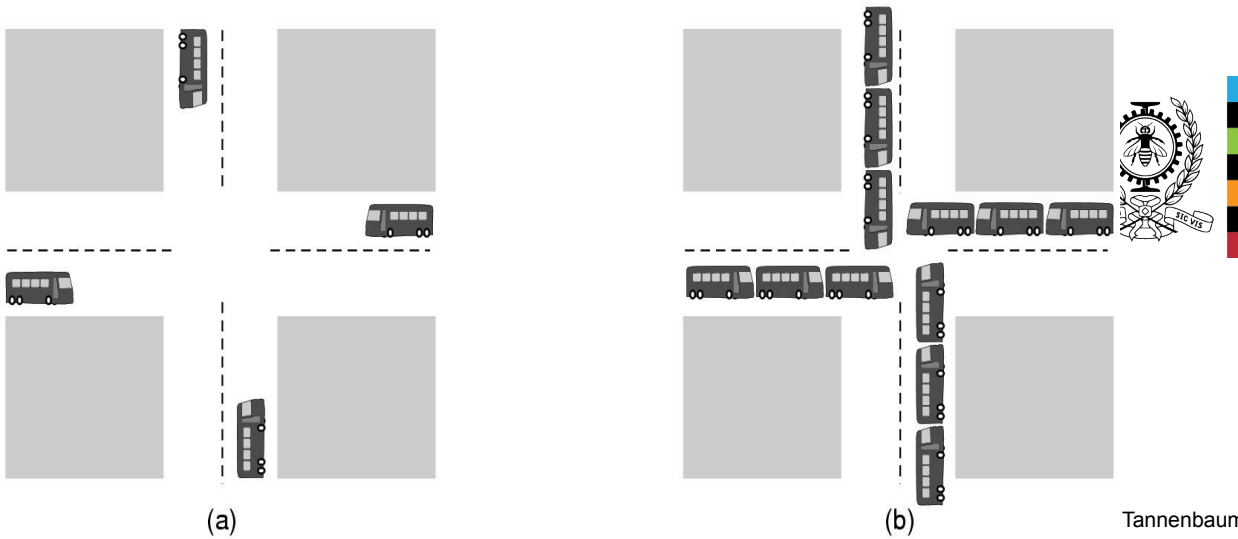


– la synchronisation (éviter les accès simultanés lecture/ écriture ou écriture/écriture à une même donnée) et



Attention : Partage de ressources
➔ interblocage

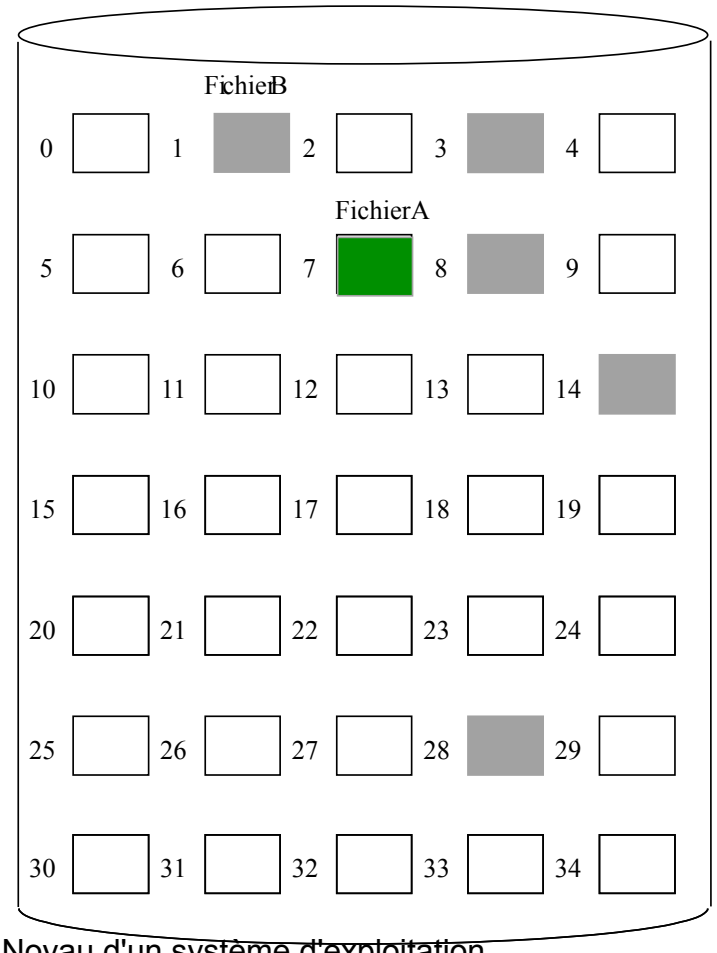
Comment éviter de telles situations ?



Tannenbaum

Appels système (5) : gestion de fichiers

- Les appels système permettent de créer des fichiers et des répertoires, de les supprimer, de les ouvrir, de les lire, de les modifier, de récupérer leurs attributs, etc.
- Un fichier ordinaire est un ensemble de blocs de données. Un bloc est composé d'un nombre fixe d'octets (unité d'allocation).

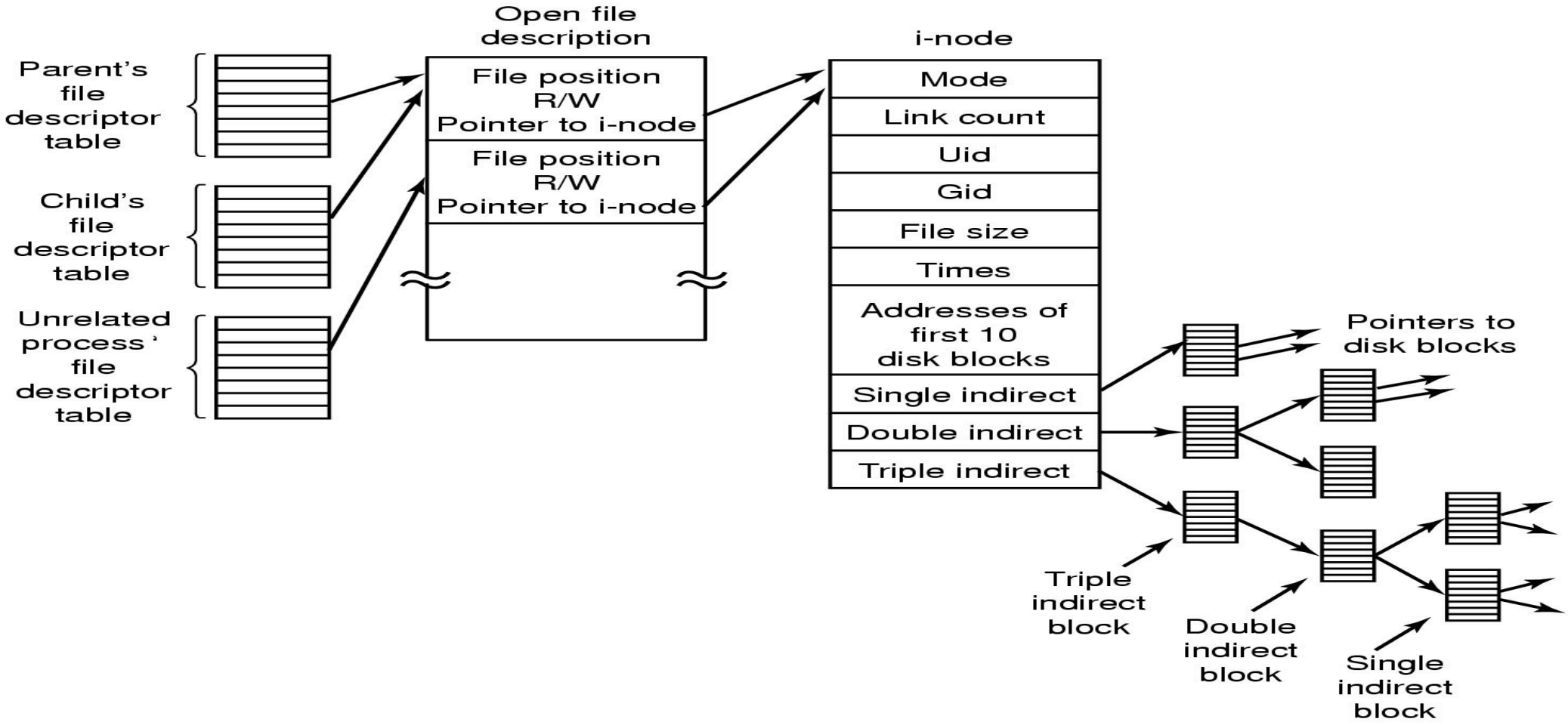


Nom du fichier	Blocs
Fichier A	7
...	...
Fichier B	1, 8, 3, 14, 28
...	...

- Les fichiers sont regroupés dans des répertoires.
- Un répertoire peut contenir aussi des répertoires (structure arborescente).
- L'accès à un fichier se fait en spécifiant le chemin d'accès (la liste des répertoires à traverser pour accéder au fichier).
  ➔ Le chemin d'accès est relatif ou absolu.

Appels système (6) : gestion de fichiers

- Sous Linux/Unix, un fichier est représenté par un i-nœud (i-node).



Tannenbaum

Appels système (7) : write vs. printf

- 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",17);
    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
```

Appels système (8) : write vs. printf

- 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",17);
    return 0;
}
```

```
jupiter$  strace  -e  trace=write  -o fich  ./Write_Printf
 ici 1er write  ici 2eme write  ici 1er printf  ici 2eme printf fin de ligne de printf
 ici 3eme write
jupiter$ cat fich
write(1, " ici 1er write \0", 16)      = 16
write(1, " ici 2eme write \0", 17)     = 17
write(1, " ici 1er printf  ici 2eme printf"..., 57) = 57
write(1, " ici 3eme write \n", 17)     = 17
+++ exited with 0 +++
jupiter$
```

Appels système (9) : write vs. printf

- 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",17);
    return 0;
}
```

```
jupiter$  strace  -e  trace=write  -o fich –s 57 ./Write_Printf
 ici 1er write  ici 2eme write  ici 1er printf  ici 2eme printf fin de ligne de printf
 ici 3eme write
jupiter$ cat fich
write(1, " ici 1er write \0", 16)      = 16
write(1, " ici 2eme write \0", 17)     = 17
write(1, " ici 1er printf  ici 2eme printf fin de ligne de printf \n", 57) = 57
write(1, " ici 3eme write \n", 17)     = 17
+++ exited with 0 +++
jupiter$
```

Appels système (10) : write vs. printf

- Le programme suivant montre la différence entre **write** et **printf**

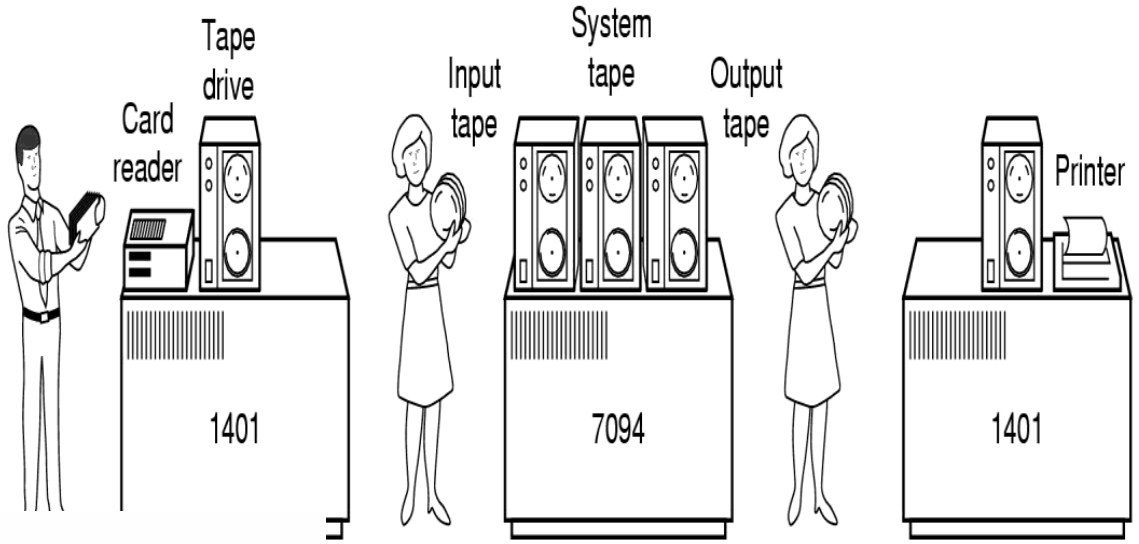
```
#include <unistd.h>                /* pour write */
#include <stdio.h>                 /* pour printf */
int main( ) {
    printf(" ici 1er printf ");
    write(1," ici 1er write ",16);
    printf(" ici 2eme printf ");
    write(1," ici 2eme write ", 17);
    printf(”fin de ligne de printf \n");
    write(1, " ici 3eme write \n",17);
    return 0;  }
```

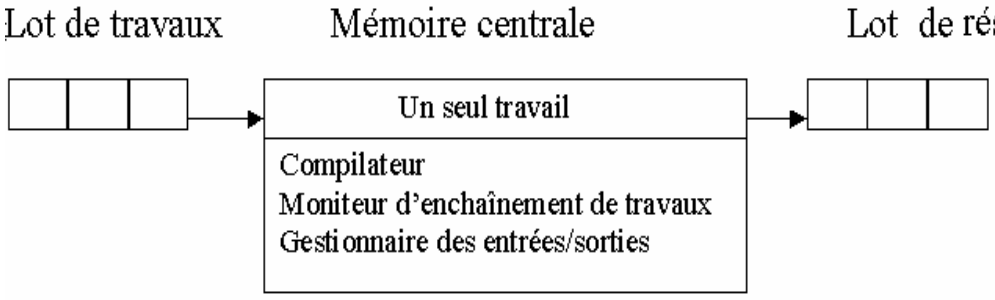
```
jupiter$  ltrace  -o fich  ./Write_Printf
 ici 1er write  ici 2eme write  ici 1er printf  ici 2eme printf fin de ligne de printf
 ici 3eme write
jupiter$  cat fich
printf(" ici 1er printf ")                                          = 16
write(1, " ici 1er write ", 16)                                      = 16
printf(" ici 2eme printf ")                                         = 17
write(1, " ici 2eme write ", 17)                                     = 17
puts("fin de ligne de printf ")                                     = 24
write(1, " ici 3eme write \n", 17)                                   = 17
+++ exited (status 0) +++
jupiter$
```

Évolution du mode d’exploitation

Traitement par lots (transistors, 1955-1965)

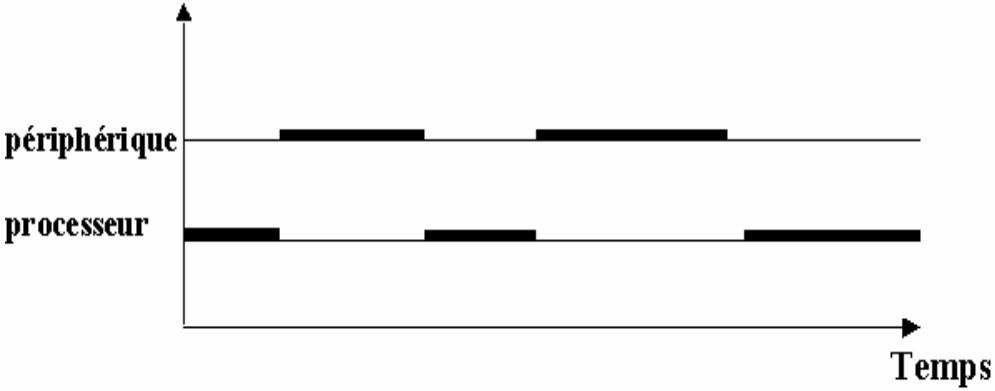
- Les programmes étaient écrits **en Fortran ou en assembleur** sur des cartes perforées.
- Ce mode d'exploitation nécessitait deux types de machines dont la plus puissante était réservée aux calculs et l'autre, moins chère, s'occupaient des périphériques lents.





Organisation de la mémoire

Tannenbaum

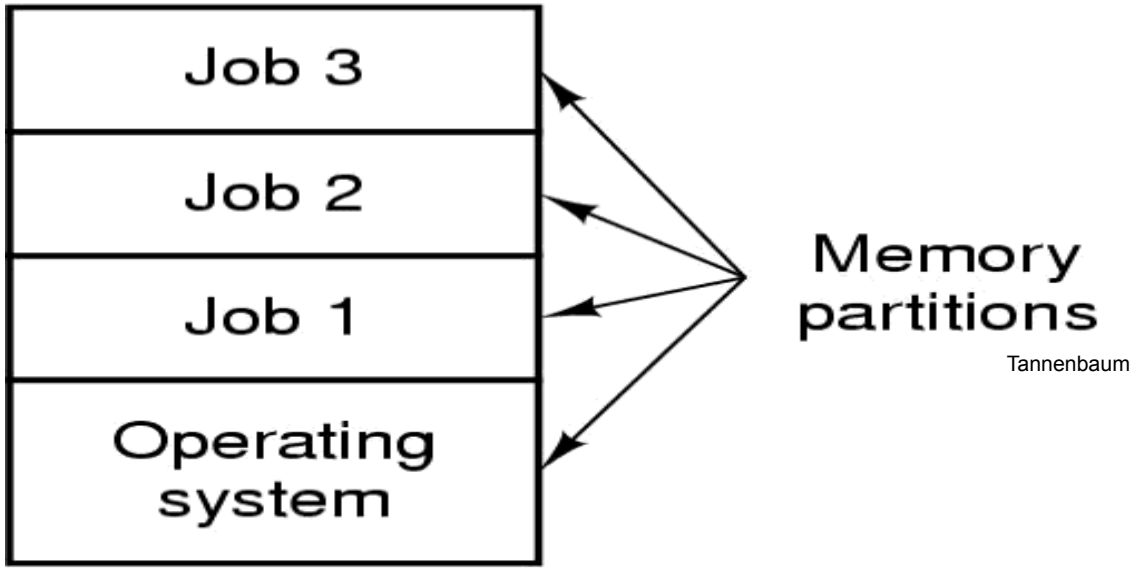


Comment maximiser le taux d'utilisation du processeur ?

Multiprogrammation et traitement par lots (circuits intégrés, 1965-1980)

- Le système d'exploitation conserve en mémoire plusieurs travaux et gère le partage du processeur et des périphériques entre les différents travaux en mémoire (la **multiprogrammation**).

- La mémoire est organisée en un ensemble de partitions (1 travail/partition).

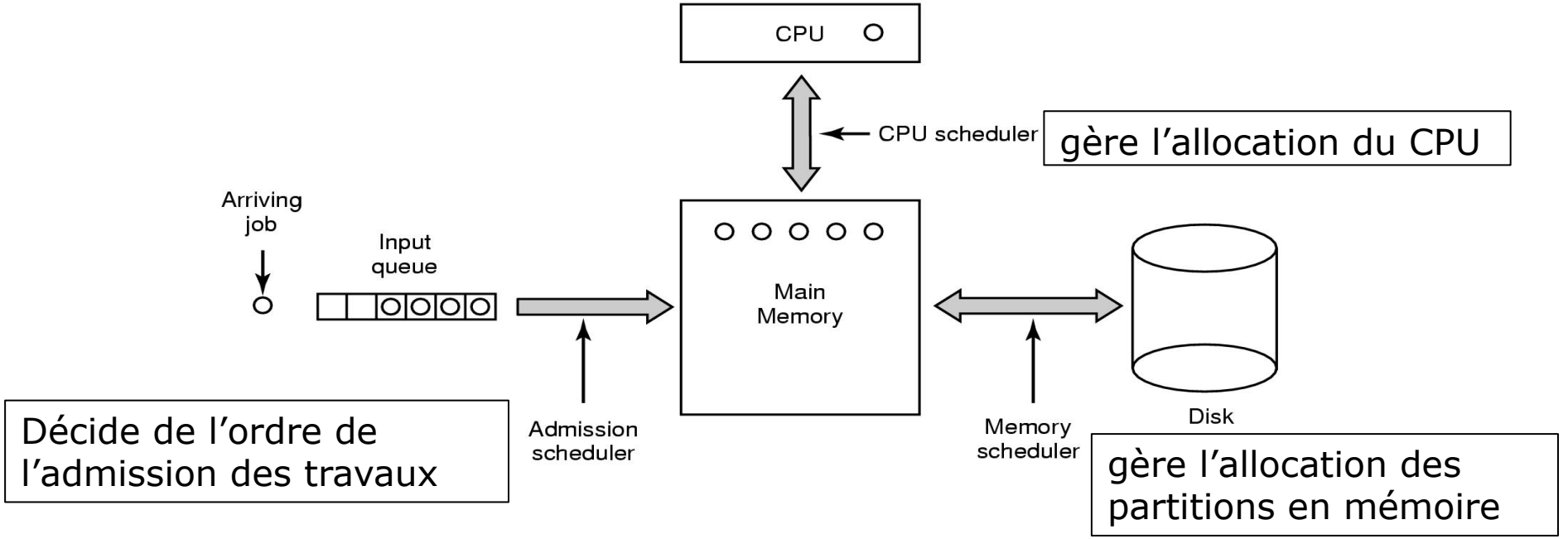


Tannenbaum

- L'introduction des unités de disque (qui permettent des accès direct aux travaux) a favorisé la multiprogrammation.

- Les travaux sont transférés vers le disque dès leurs arrivés dans la salle machine avec possibilité d'accéder directement à ces travaux.

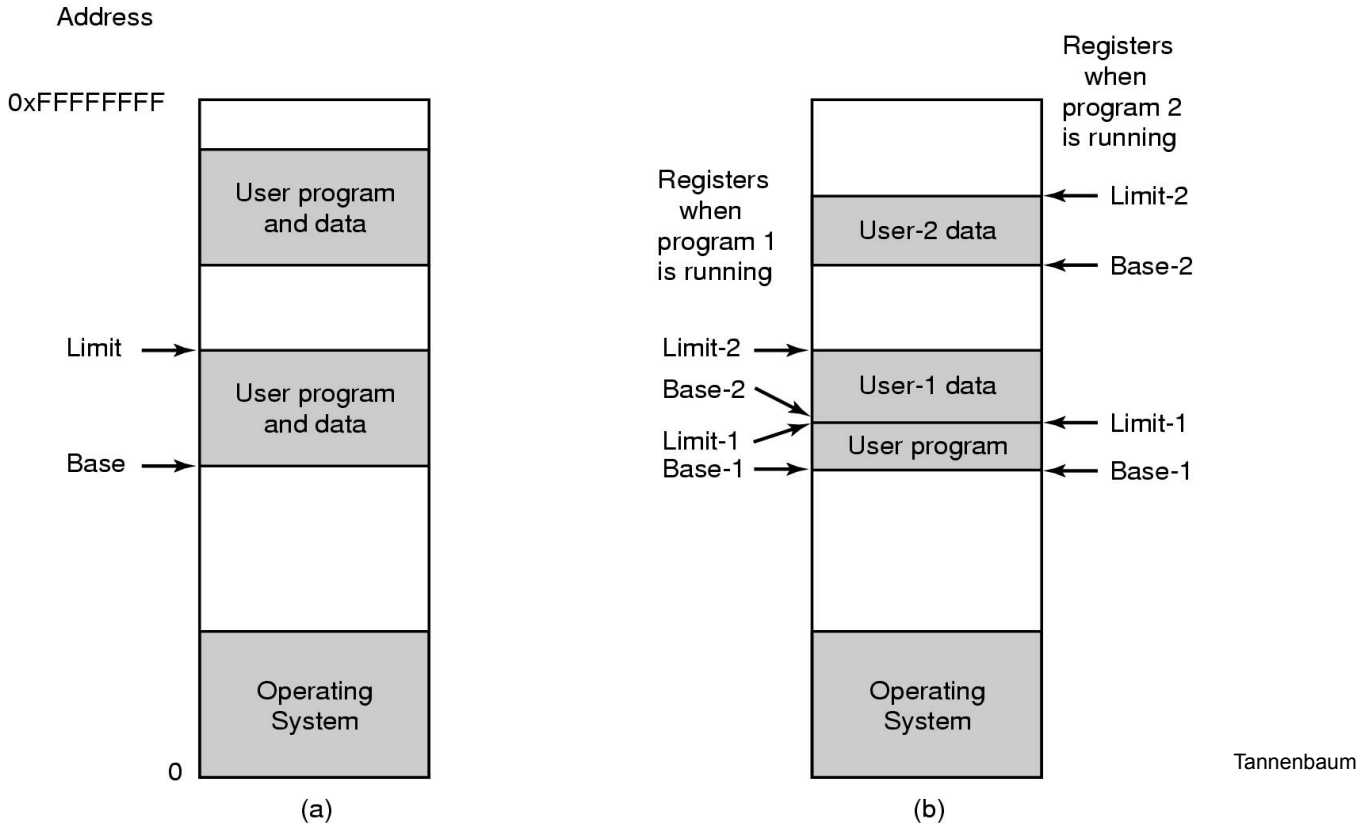
Multiprogrammation et traitement par lots (circuits intégrés, 1965-1980) (2)



- Le système alloue le processeur à un travail chargé en mémoire (premier arrivé, premier servi).
- Lorsque le travail demande une E/S, le processeur est alloué à un autre travail présent en mémoire.
- À la fin de l'E/S, une interruption se produit et le système d'exploitation reprend le contrôle pour traiter l'interruption et lancer/reprendre l'exécution d'un travail.
- Dès qu'un travail se termine, le système d'exploitation lance le chargement, à partir du disque, d'un nouveau travail dans la partition libérée.

Multiprogrammation et traitement par lots (circuits intégrés, 1965-1980) (3)

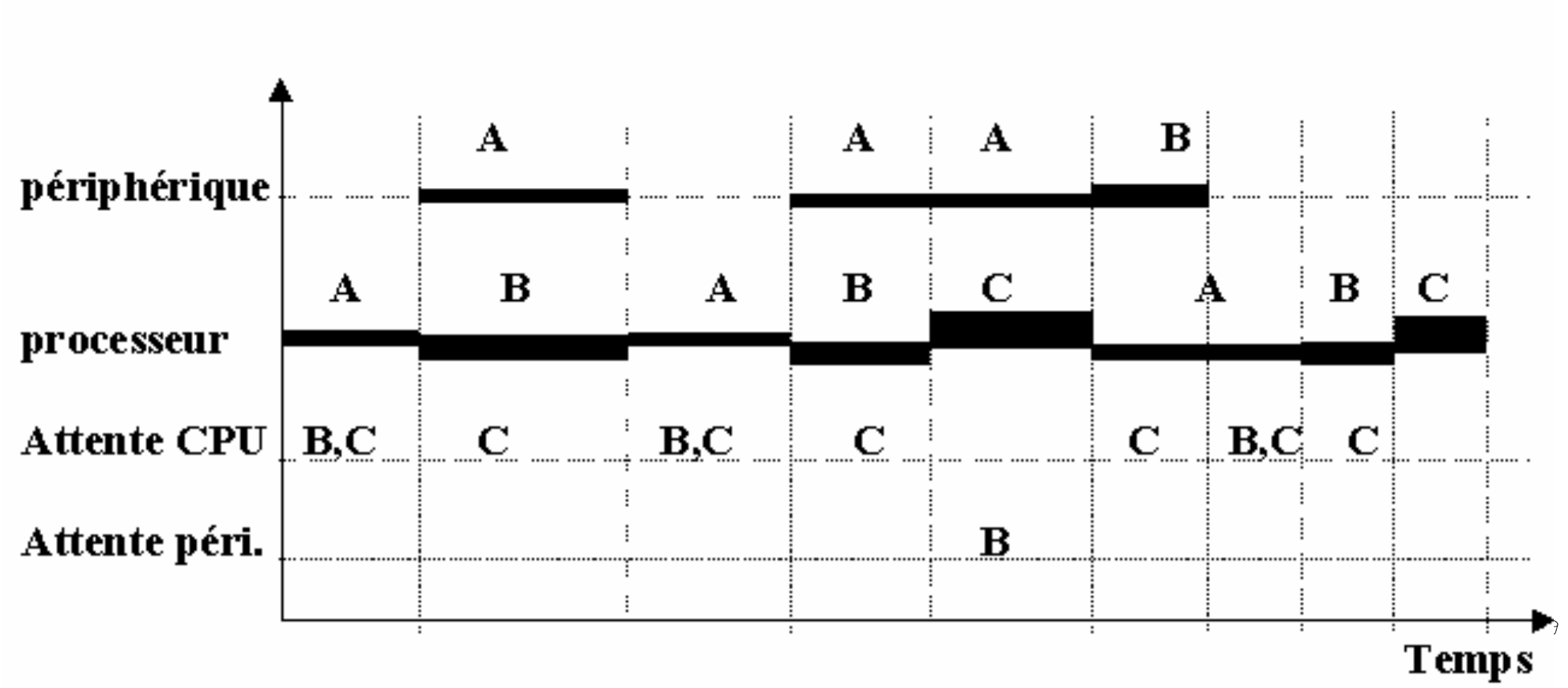
- La multiprogrammation nécessite des circuits de contrôle pour protéger chaque travail contre les intrusions et les erreurs des autres (avec possibilité de partage de codes).



- Pour décharger le processeur des opérations d'E/S, des **contrôleurs DMA (Direct Memory Access) sont utilisés** pour un transfert direct de données entre un périphérique et la mémoire ➔ E/S autonomes.

Multiprogrammation et traitement par lots (circuits intégrés, 1965-1980) (4)

Multiprogrammation et DMA :



Supposition : le système privilégie les plus anciens travaux.

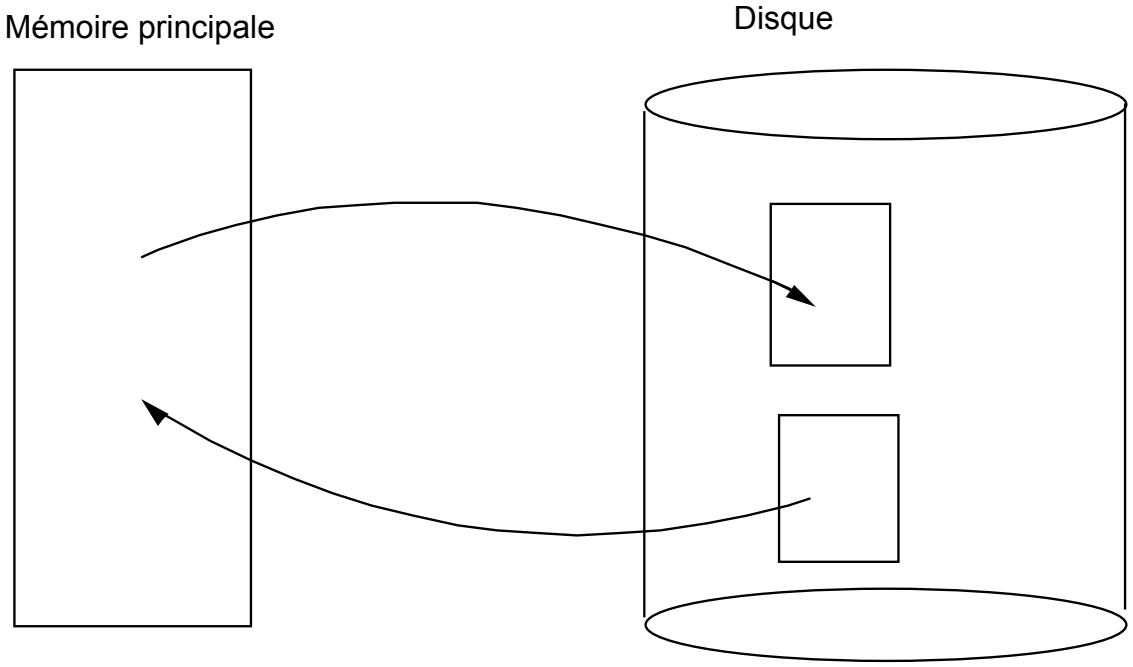
Problème :

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

Va-et-vient (swapping)

- Les travaux en mémoire 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 swap).



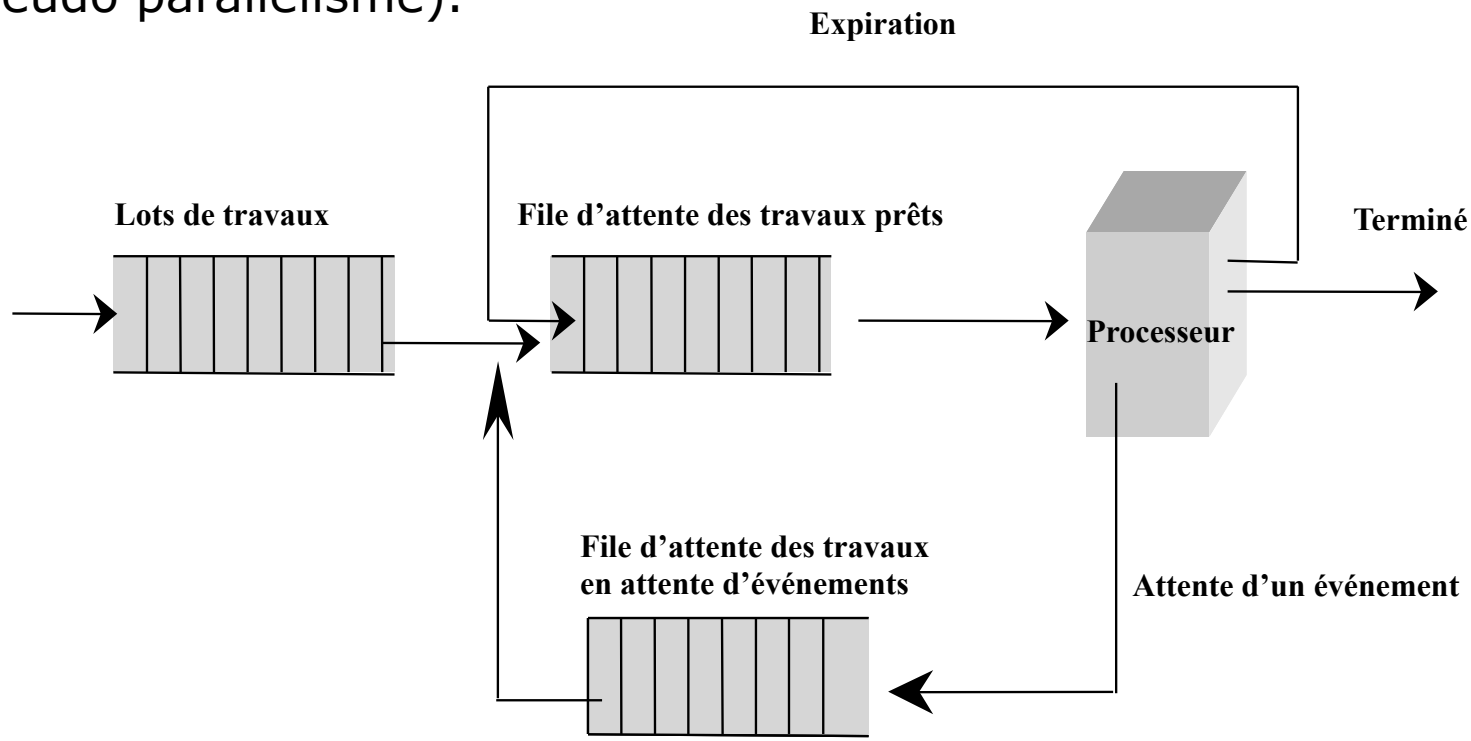
Garantir un temps de réponse acceptable à chaque utilisateur

Multiprogrammation et partage de temps : 1965-1980

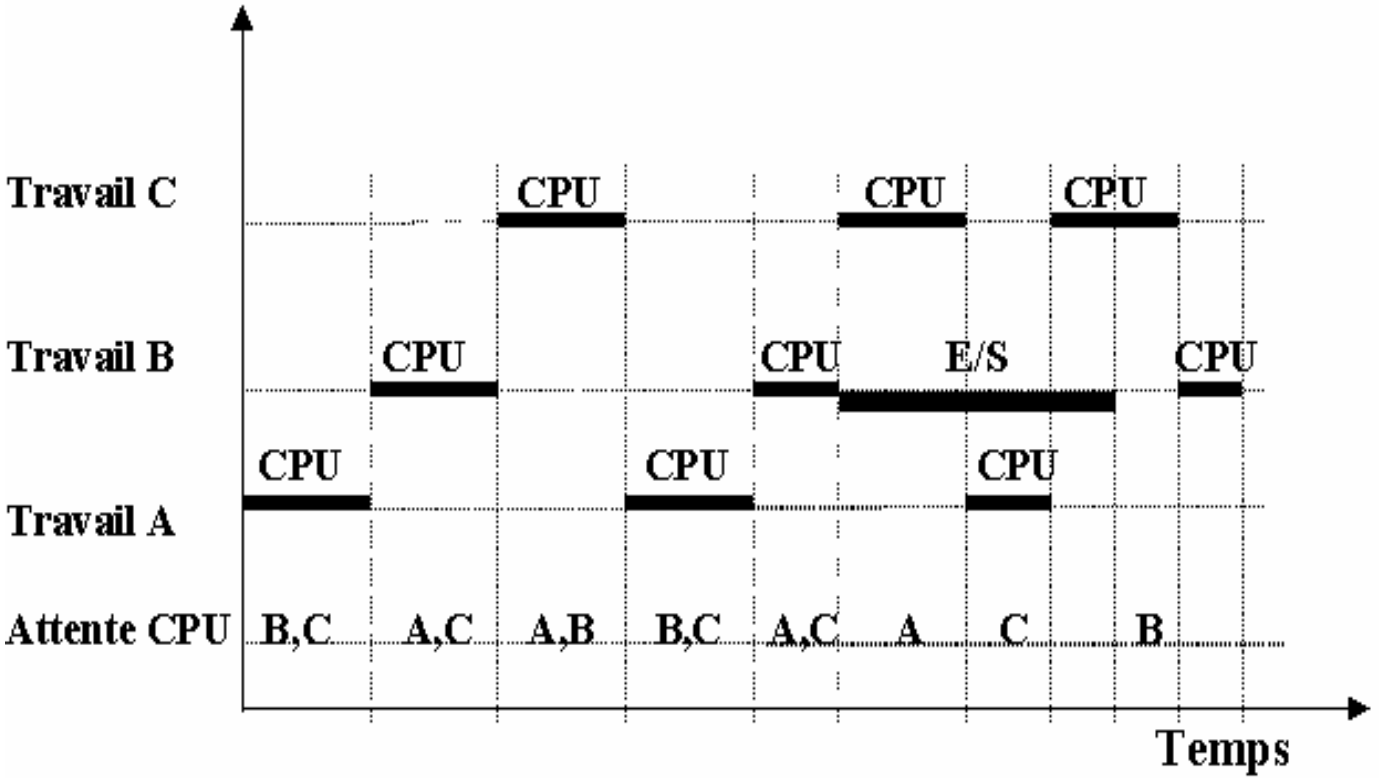
- Le processeur est alloué, à tour de rôle, pendant un certain temps à chacun des travaux en attente d'exécution. Au bout de ce temps, l'exécution du travail en cours est suspendue.

- Le processus suspendu est inséré à la fin de la file d'attente des travaux prêts et le processeur est alloué à un autre travail.

- Ce mode d'exploitation donne l'impression que les programmes s'exécutent en parallèle (pseudo parallélisme).



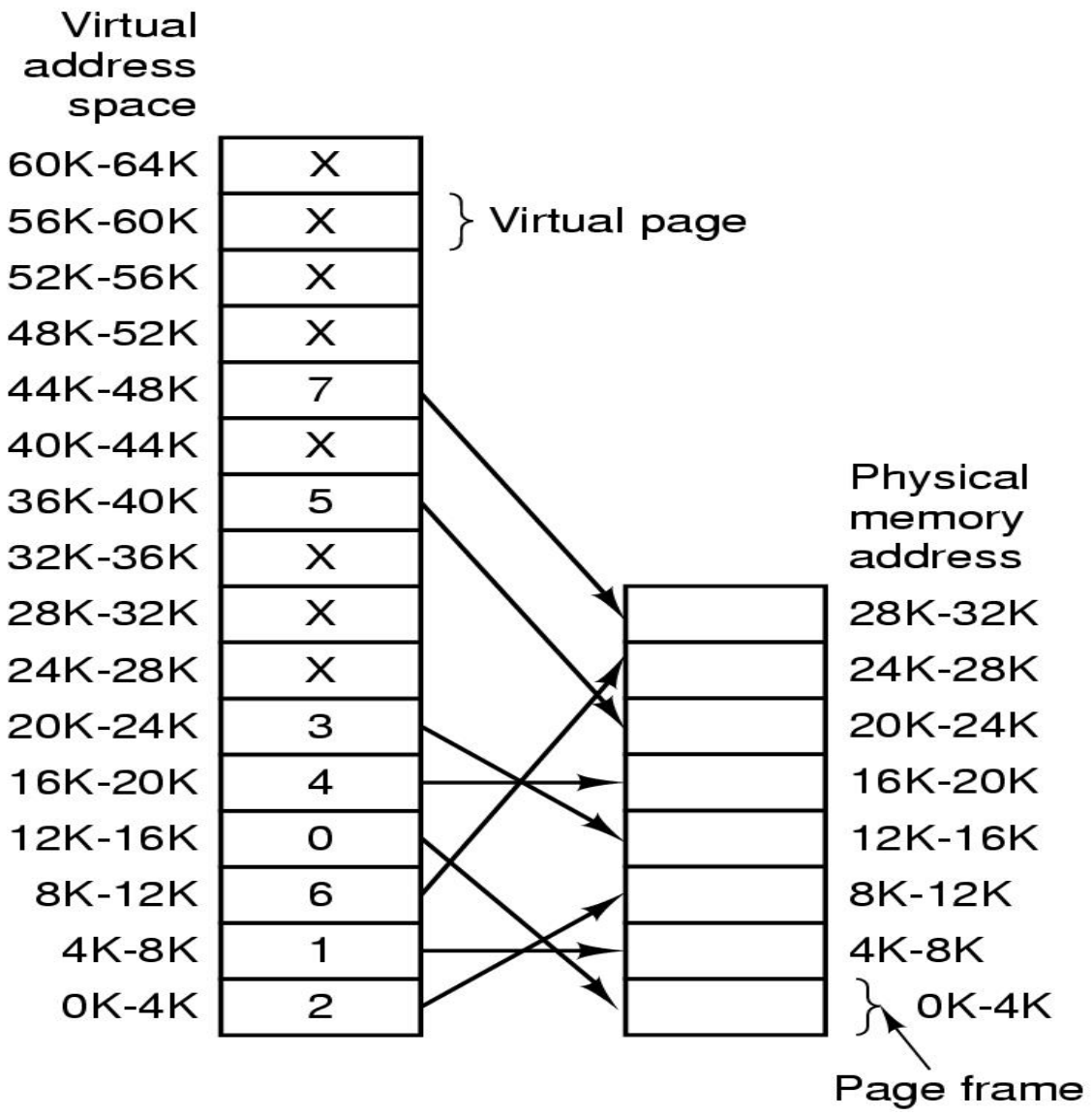
Multiprogrammation et partage de temps : 1965-1980 (2)



Exécuter des programmes dont la taille dépasse celle de la mémoire physique

Mémoire virtuelle (recherche universitaire de 1965 à 1975)

- Chaque processus a un espace d'adressage virtuel privé = {ensemble d'adresses virtuelles attribuées, par le compilateur, l'éditeur de lien et durant l'exécution aux données, aux instructions, etc.}
- Sa taille peut être beaucoup plus grande que celle de la mémoire physique.
- À l'exécution, une partie de cet espace virtuel est en mémoire physique ➔ Chargement à la demande.
- Le mécanisme de translation d'adresses est intégré au processeur.



Autres besoins

Exploitation en réseau, distribuée, en temps réel

INF3405 (Réseaux informatiques)
INF8601 (Systèmes informatiques parallèles)
INF4404 (Systèmes répartis et infonuagiques)
INF3610 (Systèmes embarqués)

Exploitation en réseau

- Chaque système est doté d'une interface réseau qui lui permet de communiquer avec d'autres machines selon le modèle client-serveur.

Exploitation distribuée

- Un système d'exploitation réparti gère et contrôle un réseau d'ordinateurs connectés (leurs processeurs, leurs mémoires, les disques, etc.) => Ce réseau d'ordinateurs apparait aux utilisateurs comme une machine monoprocesseur.

Exploitation en 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).

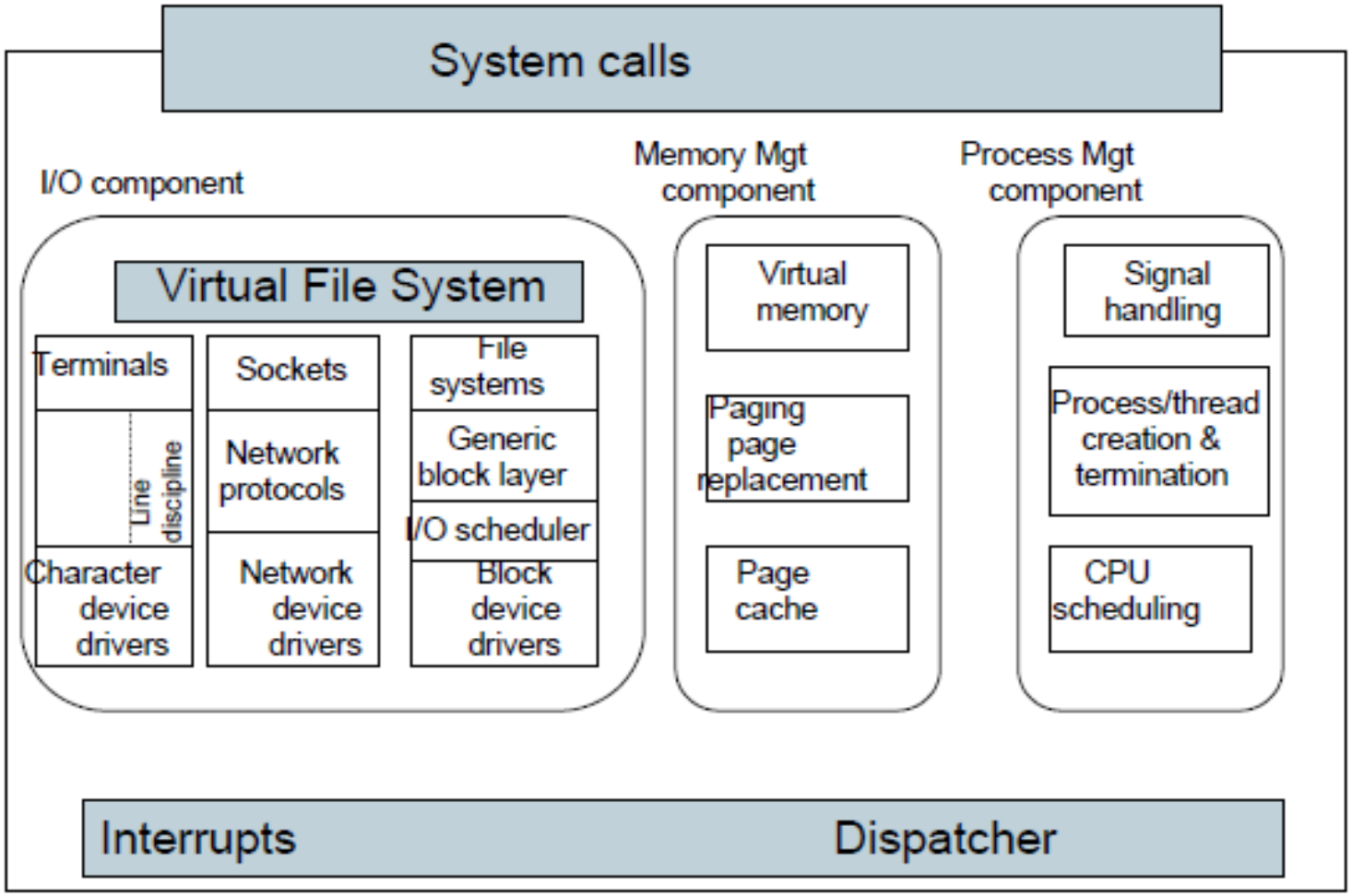
➔ RT-Linux, **VxWorks**, QNX et **MicroC**.

Structure des systèmes d’exploitation

Structure des systèmes d'exploitation

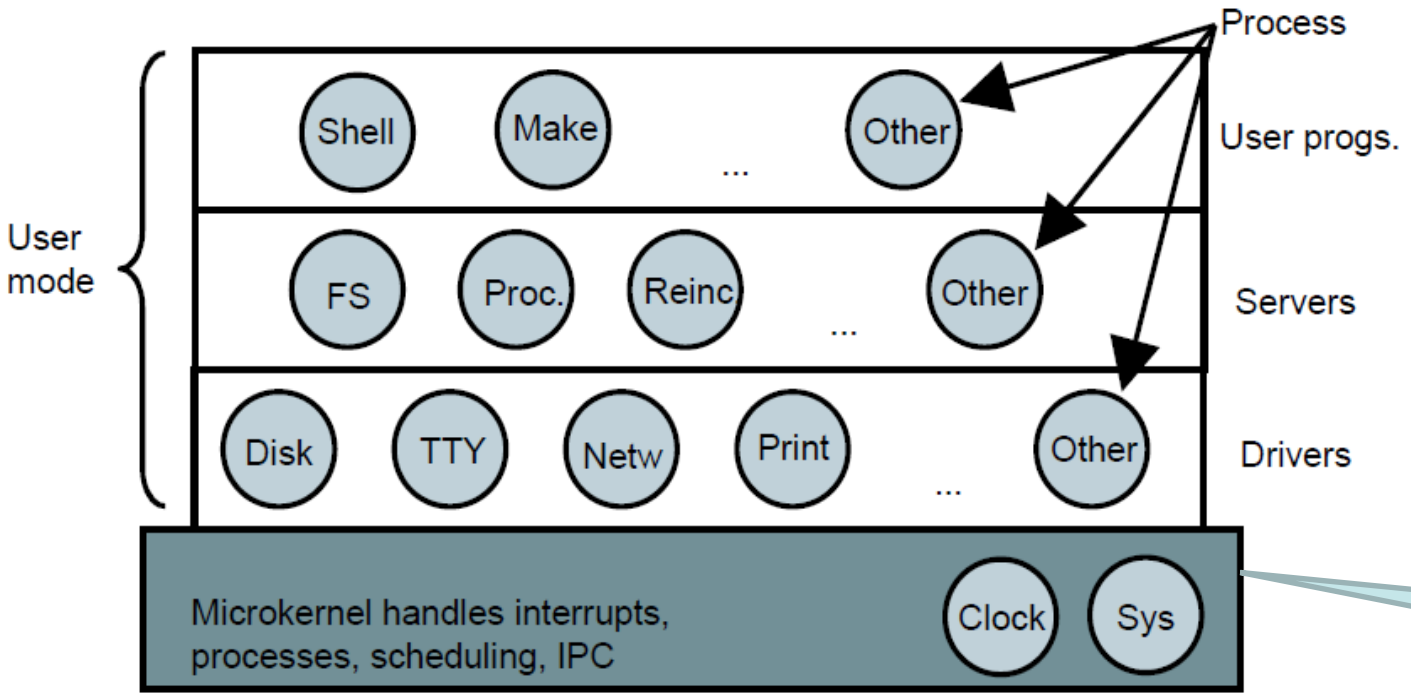
- **Noyau monolithique modulaire**: est un ensemble de modules chargeables dynamiquement qui s'exécutent en mode noyau (**Linux**, **BSD**, **Solaris**).

Linux



Structure des systèmes d'exploitation (2)

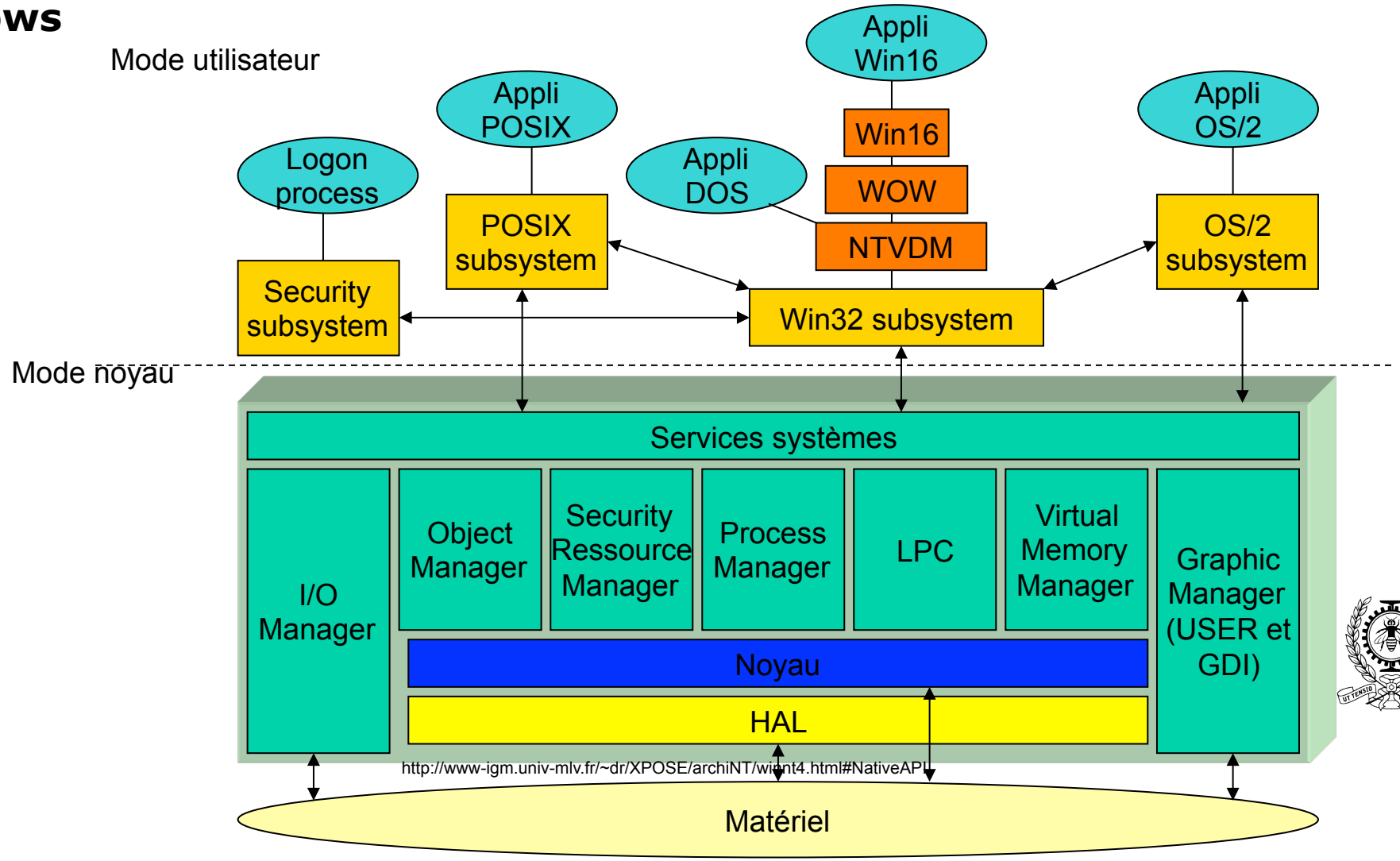
- **Micronoyau :** contient le strict minimum qui fonctionne en mode noyau **(MINIX)**.



Structure des systèmes d'exploitation (3)

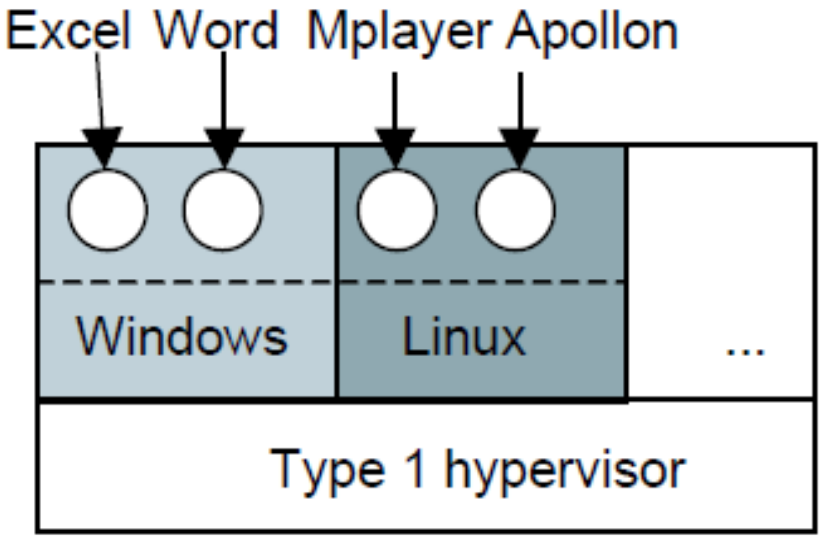
- **Noyau hybride** : (**Windows, Mac OS X** (Mach + une partie du BSD)).

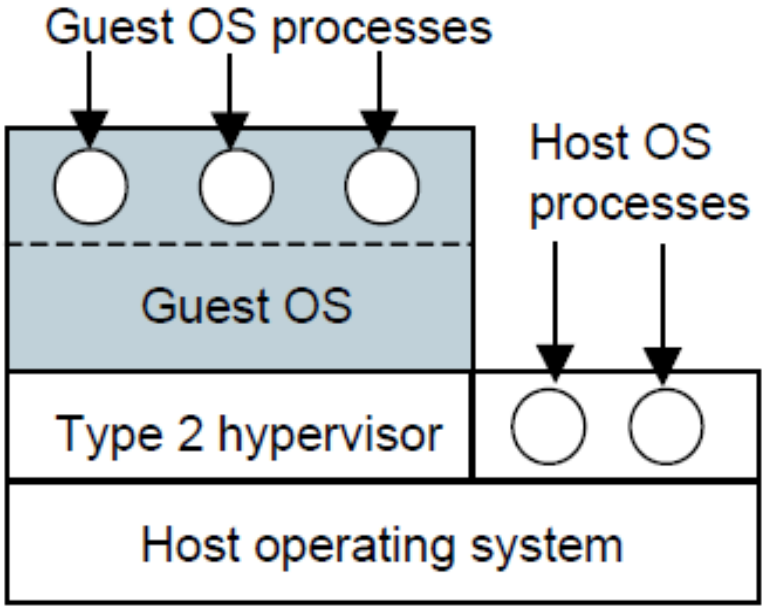
Windows



Structure des systèmes d'exploitation (4)

- **Machines virtuelles**





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

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 connecté), Windows 7 (2009, multi-utilisateur), etc.

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

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