CNAM IDF

# *Principes fondamentaux des Systèmes d'exploitation*

UTC502

« Julien Boudani »

# Contenu de la formation et Organisation

- Objectifs pédagogiques
  - Comprendre les principes fondamentaux des systèmes d'exploitation multiprogrammés
- Compétences visées
  - Appréhender les mécanismes fondamentaux des systèmes d'exploitation
- Modalités de validation
  - La note finale de l'unité est obtenue comme suit
    - 2 devoirs maison noté sur 5 points
    - un examen sur table noté sur 15 points

# Contenu de la formation et Organisation

- Contenu
  - Notions de base sur les systèmes d'exploitation,  Mise en œuvre de la  protection/isolation : notion d'espace d'adressage, de modes d'exécution user/superviseur, introduction des appels système.
  - Gestion des exécutions programmes, processus, ordonnancement,  threads
  - Synchronisation
  - Gestion de la mémorisation, mémoire centrale  pagination, problèmes de gestion mémoire et d'allocation de blocs de tailles variables
  - Notion de base en administration système,  comptes, droits, etc gestion des I/O asynchrones et des interruptions.

# Ressources

- Les systèmes d'exploitation, Unix, Linux et Windows XP, avec C et Java, Dunod 2003 (566 pages), ISBN : 2100071890, S. Bouzefrane
- Linux. Programmation système et réseau : Cours et exercices corrigés, Joëlle Delacroix
- Modern Multithreading Wiley 2005 (465 pages), R. H. Carver, Kuochung Tai
- Virtual Machines - versatile platforms for systems and processes, Elsevier, J.E. Smith, R. Nair
- Hardware and Software Support for Virtualization - Morgan and Claypool Life Sciences ISBN: 1627056939, E. Bugnon, J. Nich, D. Tsafrir
- Notes de cours, P. Ingels, Université de Rennes I
- Les systèmes Informatiques, Christian Carrez

# Ressources

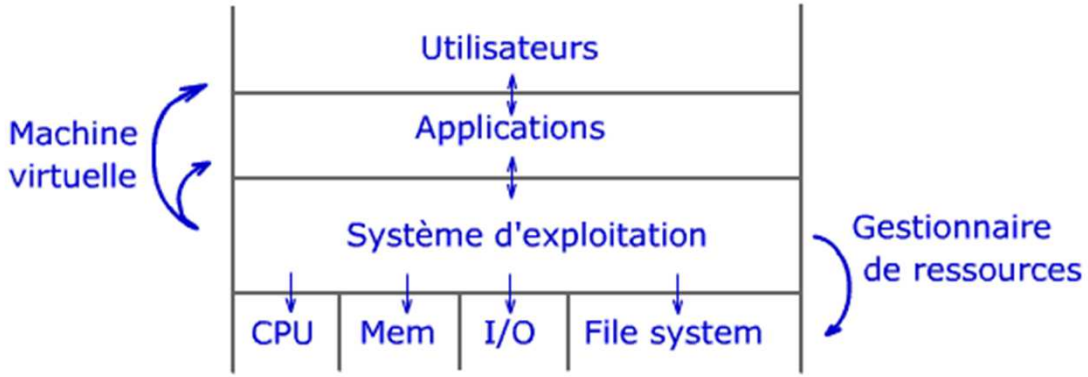
- Computer Architecture, a first course, Van Nostrand Reinhold, ISBN 2 225 80929 1
- Ecrire des applications réseau sous Linux, thème Système et réseaux de Linux Magazine, octobre 99, par Alain Basty.
- man d'Unix, notons ici que lorsque l'on doit programmer les systèmes d'exploitation le man d'Unix est la meilleure référence pour le programmeur. Il n'y a rien d'équivalent sur papier.
- J.M. Rifflet, La programmation sous Unix, 3e éd., McGraw-Hill, 1993 est cependant une bonne référence et un ouvrage assez complet
- Charles Petzold, Programming Windows 95, Microsoft Press, 1996 est considéré comme une bonne référence pour apprendre la programmation Windows. Elle a été la première du genre. Actuellement, il y a des dizaines d'ouvrages équivalents.

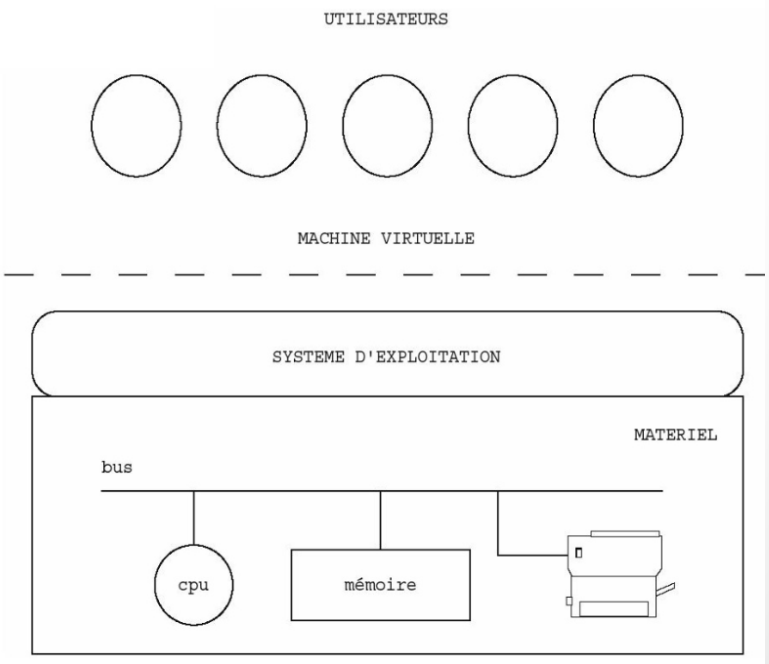
# Evolution des Systèmes d'Exploitation

# Présentation générale

- Le système d'exploitation est un ensemble de programmes qui réalise l'interface entre le matériel de l'ordinateur et les utilisateurs. Il a 2 objectifs principaux :
  - prise en charge de la gestion de plus en plus complexe des ressources et partage de celles-ci ;
  - construction au-dessus du matériel d'une machine virtuelle plus facile d'emploi et plus conviviale.

  La machine physique et ses différents composants, s'ils offrent des mécanismes permettant de faciliter leur partage entre différents programmes, ne sont malgré tout pas conçus pour supporter et gérer d'eux-mêmes ce partage. C'est là le premier rôle du système d'exploitation dans un environnement multiprogrammé que de gérer le partage de la machine physique et des ressources matérielles entre les différents programmes. Cette gestion doit assurer l'équité d'accès aux ressources matérielles et assurer également que les accès des programmes à ces ressources s'effectuent correctement, c'est-à-dire que les opérations réalisées par les programmes sont licites pour la cohérence des ressources : on parle alors de *protection des ressources*.


Utilisateurs
Applications
Système d'exploitation
CPU
Mem
I/O
File system
Machine virtuelle
Gestionnaire de ressources


UTILISATEURS
MACHINE VIRTUELLE
SYSTEME D'EXPLOITATION
MATERIEL
bus
cpu
mémoire

Le partage des ressources va concerner principalement le processeur, la mémoire centrale et les périphériques d'entrées-sorties. Plus précisément, les questions suivantes vont devoir être résolues :

- Dans le cadre du partage du processeur : parmi tous les programmes chargés en mémoire centrale, lequel doit s'exécuter ?
- Dans le cadre du partage de la mémoire centrale : comment allouer la mémoire centrale aux différents programmes ? Comment disposer d'une quantité suffisante de mémoire pour y placer tous les programmes nécessaires à un bon taux d'utilisation du processeur ? Comment assurer la protection entre ces différents programmes utilisateurs ? Par protection, on entend ici veiller à ce qu'un programme donné n'accède pas à une plage mémoire allouée à un autre programme.
- Dans le cadre du partage des périphériques : dans quel ordre traiter les requêtes d'entrées-sorties pour optimiser les transferts ?

Faciliter l'accès à la machine physique constitue le second rôle du système d'exploitation. Par le biais d'une interface de haut niveau, composée d'un ensemble de primitives attachées à des fonctionnalités qui gèrent elles-mêmes les caractéristiques matérielles sous-jacentes et offrent un service à l'utilisateur, le système d'exploitation construit au-dessus de la machine physique, une machine virtuelle plus simple d'emploi et plus conviviale. Ainsi, pour réaliser une opération d'entrées-sorties, l'utilisateur fait appel à une même primitive `ECRIRE(données)` quel que soit le périphérique concerné. C'est la primitive `ECRIRE` et la fonction de gestion des entrées-sorties du système d'exploitation à laquelle cette primitive est rattachée qui feront la liaison avec les caractéristiques matérielles.

Comme son nom le suggère, le système d'exploitation a en charge l'exploitation de la machine pour en faciliter l'accès, le partage et pour l'optimiser.

Les fonctionnalités du système d'exploitation utilisent les mécanismes offerts par le matériel de la machine physique pour réaliser leurs opérations. Notamment, le système d'exploitation s'interface avec la couche matérielle, par le biais du mécanisme des interruptions, qui lui permet de prendre connaissance des événements survenant sur la machine matérielle.

Par ailleurs, le système d'exploitation s'interface avec les applications du niveau utilisateur par le biais des fonctions prédéfinies que chacune de ses fonctionnalités offre. Ces fonctions que l'on qualifie de routines systèmes constituent les points d'entrées des fonctionnalités du système d'exploitation et sont appelables depuis les applications de niveau utilisateur. Ces appels peuvent se faire à deux niveaux :

- dans le code d'un programme utilisateur à l'aide d'un *appel système*, qui n'est autre qu'une forme d'appel de procédure amenant à l'exécution d'une routine système ;
- depuis le prompt de l'interpréteur de commandes à l'aide d'une *commande*. L'*interpréteur de commandes* est un outil de niveau utilisateur qui accepte les commandes de l'utilisateur, les analyse et lance l'exécution de la routine système associée.

Les systèmes d'exploitation multiprogrammés peuvent être classés selon différents types qui dépendent des buts et des services offerts par les systèmes. Trois grandes classes de systèmes peuvent être définies :

- les systèmes à traitements par lots ;
- les systèmes multi-utilisateurs interactifs ;
- les systèmes temps réels.

## a) Systèmes à traitements par lots

Les *systèmes à traitement par lots* ou *systèmes batch* constituent en quelque sorte les ancêtres de tous les systèmes d'exploitation. Ils sont nés de l'introduction sur les toutes premières machines de deux programmes permettant une exploitation plus rapide et plus rentable du processeur, en vue d'automatiser les tâches de préparation des travaux à exécuter. Ces deux programmes sont d'une part le chargeur dont le rôle a été initialement de charger automatiquement les programmes dans la mémoire centrale de la machine depuis les cartes perforées ou le dérouleur de bandes et d'autre part, le moniteur d'enchaînement de traitements, dont le rôle a été de permettre l'enchaînement automatique des travaux soumis en lieu et place de l'opérateur de la machine.

Le principe du traitement par lots s'appuie sur la composition de lots de travaux ayant des caractéristiques ou des besoins communs, la formation de ces lots visant à réduire les temps d'attente du processeur en faisant exécuter les uns à la suite des autres ou ensemble, des travaux nécessitant les mêmes ressources.

Par ailleurs, dans un système à traitements par lots, la caractéristique principale est qu'il n'y a pas d'interaction possible entre l'utilisateur et la machine durant

l'exécution du programme soumis. Le programme est soumis avec ses données d'entrées et l'utilisateur récupère les résultats de l'exécution ultérieurement, soit dans un fichier, soit sous forme d'une impression. Là encore, ce mode de fonctionnement est directement influencé par le mode de travail des premiers temps de l'informatique : le programmeur soumettait son programme sous forme de cartes perforées avec ses données à l'opérateur de la machine, qui faisait exécuter le programme et rendait les résultats de l'exécution ensuite au programmeur.

L'objectif poursuivi dans les systèmes à traitements pas lots est de maximiser l'utilisation du processeur et le débit des travaux, c'est-à-dire le nombre de travaux traités sur une tranche de temps.

## b) Systèmes interactifs

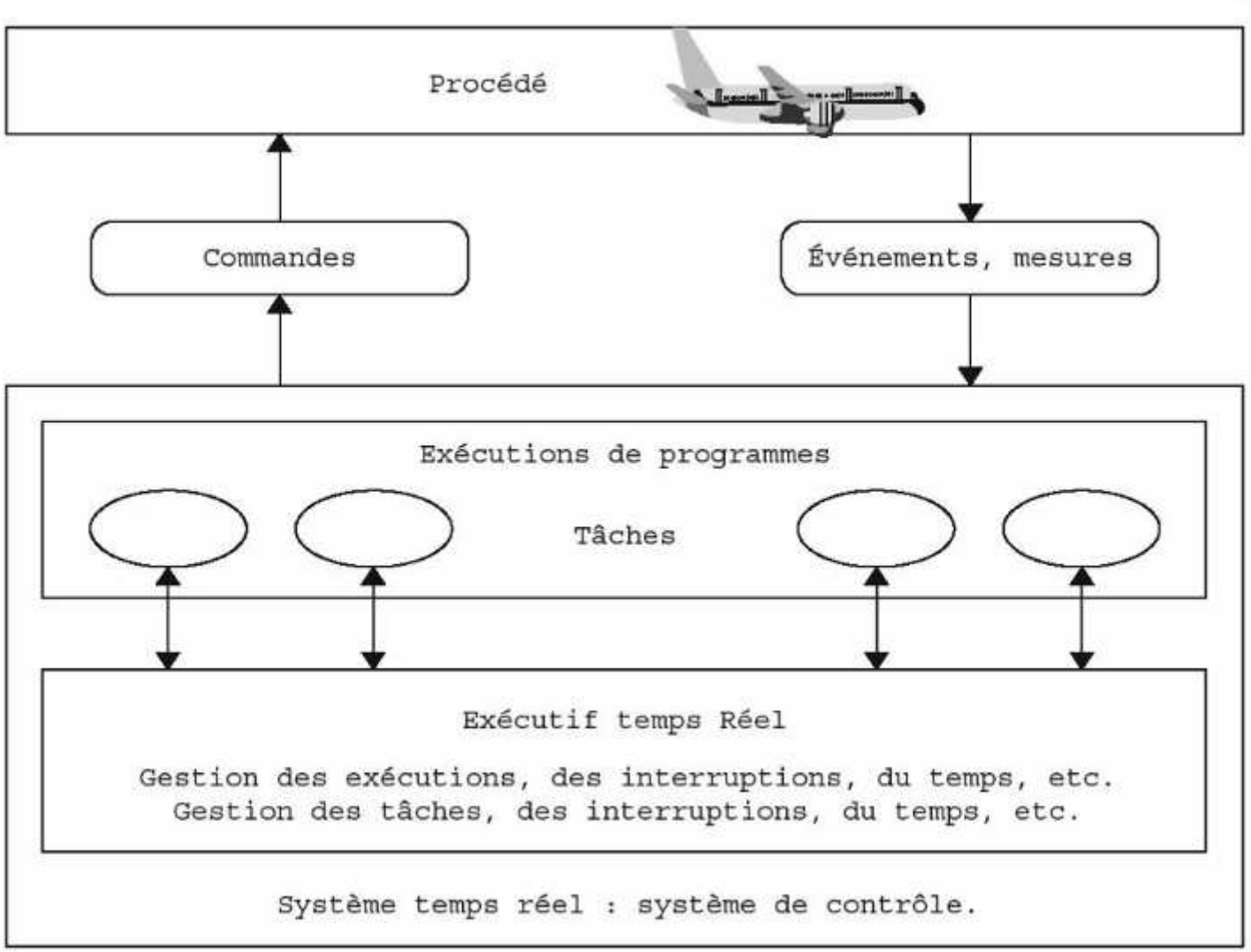
La particularité d'un système d'exploitation interactif est qu'au contraire des systèmes précédents, l'utilisateur de la machine peut interagir avec l'exécution de son programme. Typiquement, l'utilisateur lance l'exécution de son travail et attend derrière le clavier et l'écran, le résultat de celle-ci. S'il s'aperçoit que l'exécution n'est pas conforme à son espérance, il peut immédiatement agir pour arrêter celle-ci et analyser les raisons de l'échec.

Puisque l'utilisateur attend derrière son clavier et son écran et que par nature, l'utilisateur de la machine est un être impatient, le but principal poursuivi par les systèmes interactifs va être d'offrir pour chaque exécution le plus petit temps de réponse possible. Le temps de réponse d'une exécution est le temps écoulé entre le moment où l'exécution est demandée par l'utilisateur et le moment où l'exécution est achevée. Pour parvenir à ce but, la plupart des systèmes interactifs travaillent en temps partagé.

Un système en temps partagé permet aux différents utilisateurs de partager l'ordinateur simultanément, tout en ayant par ailleurs la sensation d'être seul à utiliser la machine. Ce principe repose notamment sur un partage de l'utilisation du processeur par les différents programmes des différents utilisateurs. Chaque programme occupe à tour de rôle le processeur pour un court laps de temps (*le quantum*) et les exécutions se succèdent suffisamment rapidement sur le processeur pour que l'utilisateur ait l'impression que son travail dispose seul du processeur.

- C) systèmes temps réel

Les systèmes temps réel sont des systèmes liés au contrôle de procédé pour lesquels la caractéristique primordiale est que les exécutions de programmes sont soumises à des contraintes temporelles, c'est à dire qu'une exécution de programme est notamment qualifiée par une date butoir de fin d'exécution, appelée échéance, au-delà de laquelle les résultats de l'exécution ne sont plus valides. Des exemples de système temps réel sont par exemple le pilotage automatique d'un train ou encore par le dispositif de surveillance d'un réacteur de centrale nucléaire.


Procédé
Commandes
Événements, mesures
Exécutions de programmes
Tâches
Exécutif temps Réel
Gestion des exécutions, des interruptions, du temps, etc.
Gestion des tâches, des interruptions, du temps, etc.
Système temps réel : système de contrôle.

Il ne s'agit pas de rendre le résultat le plus vite possible, mais simplement à temps. L'échelle du temps relative à la contrainte temporelle varie d'une application à l'autre : elle peut être par exemple de l'ordre de la microseconde dans des applications de contrôle radars, mais peut être de l'ordre de l'heure pour une application de contrôle chimique. Par contre, il est souvent primordial de respecter la contrainte temporelle, sous peine de graves défaillances, pouvant mettre en péril le procédé lui-même et son environnement. Tout retard de réaction vis-à-vis d'une situation anormale au sein du réacteur nucléaire, peut évidemment mener à une situation catastrophique !

On distingue deux types de contraintes temporelles :

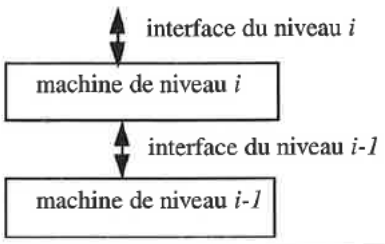
- les contraintes temporelles strictes (*temps réel strict ou dur*) : les fautes temporelles (non respect d'une contrainte temporelle) sont intolérables pour la validité du système. Elles mettent en péril le système temps réel lui-même voire son environnement. Par exemple, dans un système de lancement de missiles, si le lancement du missile est retardé, la cible risque d'être ratée ;

- les contraintes temporelles relatives (*temps réel relatif ou mou*) : quelques fautes temporelles peuvent être supportées sans remettre en cause la validité du système. Il s'ensuit généralement une baisse de performance du système. Par exemple, dans un système audio, si un paquet de données est perdu, la qualité du son sera dégradée, mais il restera audible.

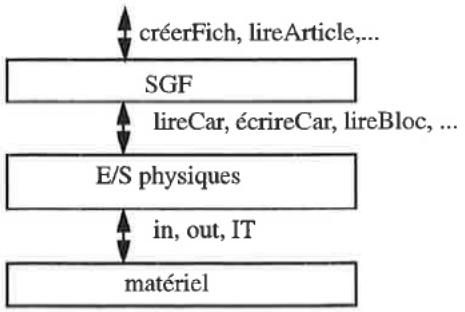
Pour être en mesure de respecter les contraintes temporelles associées aux exécutions de programmes, le système temps réel doit offrir un certain nombre de méca-nismes particuliers, dont le but est de réduire au maximum tout indéterminisme au niveau des durées des exécutions et de garantir par là même que les contraintes temporelles seront respectées. Par exemple, le processeur sera de préférence alloué à l'exécution la plus urgente et les choix de conception du système lui-même seront tels que les ressources physiques comme la mémoire ou encore les périphériques seront gérés avec des méthodes simples, limitant les fluctuations et les attentes indéterminées.

Le système temps réel est souvent qualifié d'*exécutif temps réel*.

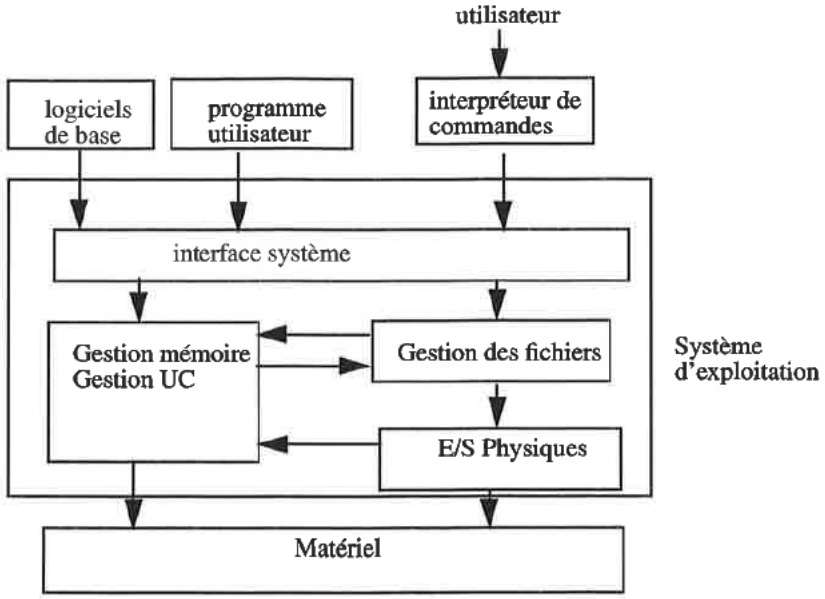
# Structuration des systèmes d'exploitation
# les premiers systèmes




Exemple de structure pour le système d'E/S

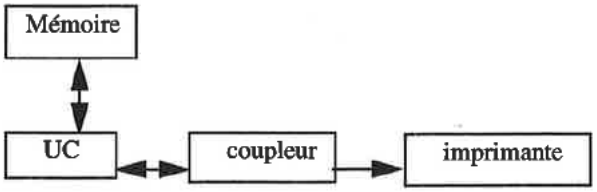



Machine virtuelle et Interface

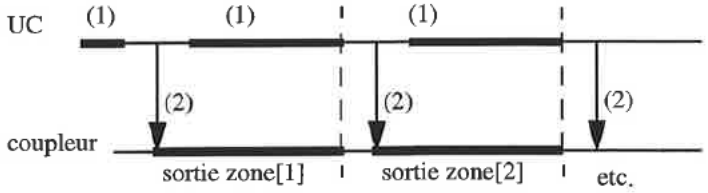



Structure d'un système mono-utilisateur

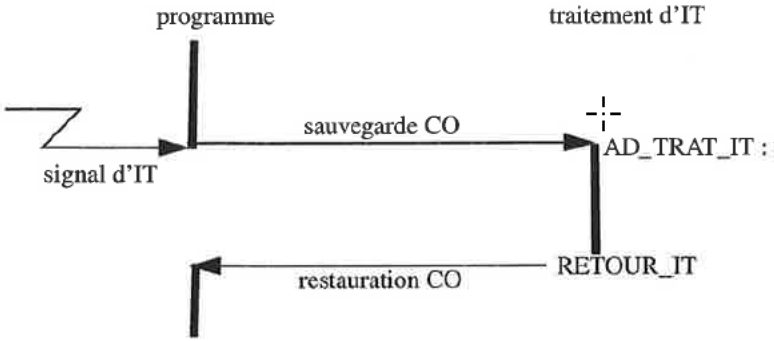
# Synchronisation en UC et périphérique



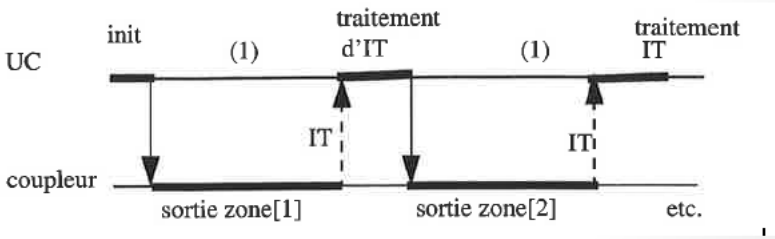

Configuration matérielle




Synchronisation par attente active

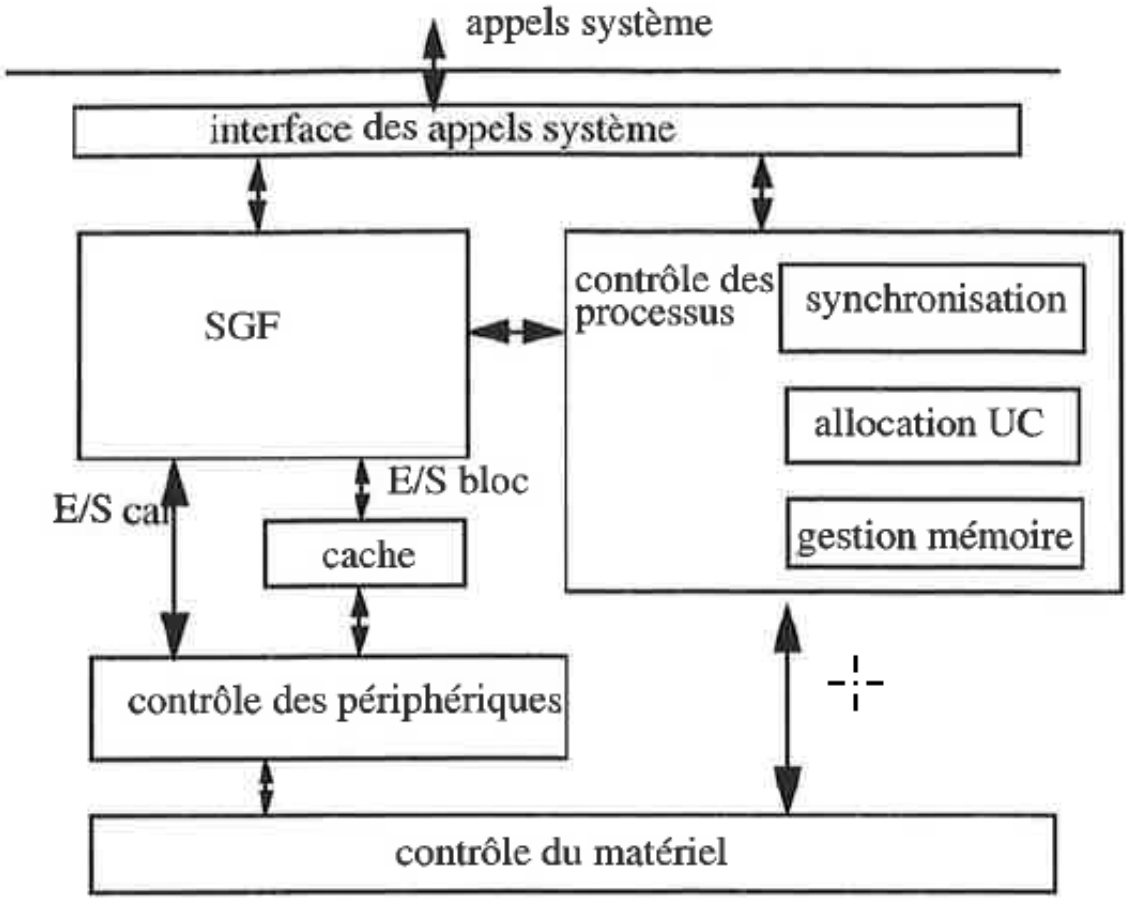



Le mécanisme d'interruption




Synchronisation par Interruption

# Structure d'un système multiprogrammé (Linux)

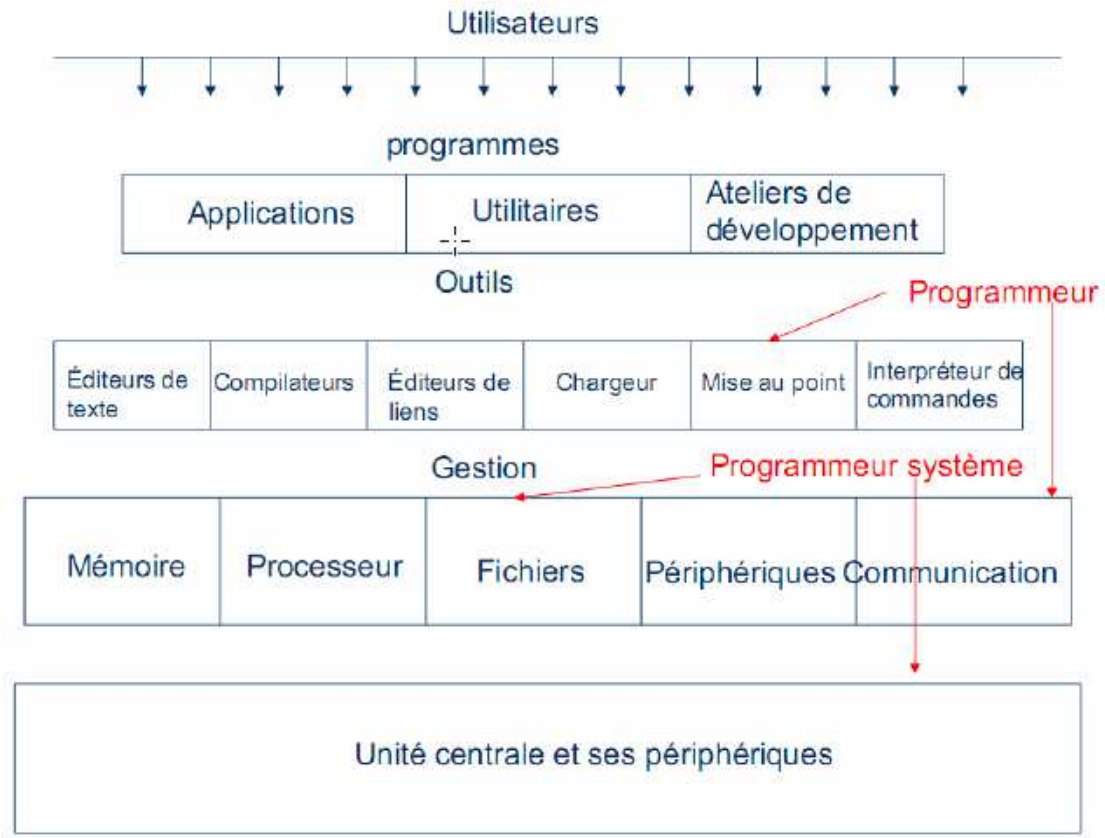


# Un peu d'histoire

- Nous allons montrer que les problèmes évoqués précédemment ne sont pas nouveaux, ni les solutions adaptées d'ailleurs

- On va également donner un panorama rapide de ce qui existe en matière de systèmes d'exploitation et de la façon dont ils peuvent évoluer

# Structure générale d'un système informatique



# Définition d'un système Informatique

- Un système informatique
  - Matériel
    - Unité centrale, disques, réseau, périphériques
  - Système d'exploitation
    - Gestion utilisateurs, SGF, ressources (accès, partage, distribution, échange…)
  - Programmes d'application
  - Utilisateurs
    - Login, droits, accès

# Systèmes d'exploitations

- En anglais « Operating System (OS) »
- Qu'est-ce que c'est?

  « Programme assurant la gestion de l'ordinateur et de ses périphériques »

  [www.dicofr.com]

- A quoi ca sert?
  - à simplifier la vie des utilisateurs et des programmeurs
  - à gérer les ressources de la machine d'une manière efficace

# Le rôle du système d'exploitation

- Le SE est une interface entre les applications et le matériel

- Le SE a pour responsabilités de :
  - gérer les ressources (mémoire, disque, périphérique, processeur, etc.)
  - associer les protections nécessaires à la gestion des ressources
  - fournir un accès équitable aux différentes ressources partagées
  - ordonnancer les programmes pour assurer une bonne qualité de service

# Caractéristiques des SE modernes

- Multi-threading
  - Processus légers
- Multi-processus
- Multi-coeur, partage mémoire (multi-processeur)
- Architecture à micro-noyaux (micro-kernel)
  - Gestion de l'adressage
  - Communication interprocess
  - Ordonnancement
- Système distribué

# Abstraction

- Cacher la complexité des machines pour l'utilisateur afin d'utiliser la machine sans savoir ce qui est derrière

- Abstraction du terme « Machine » selon Coy:
  - machine réelle = Unité centrale + périphériques
  - machine abstraite = machine réelle + système d'exploitation
  - machine utilisable = machine abstraite + application

# Exigences à un Système d'exploitation

- Généralités
  - Satisfaire les utilisateurs et les programmeurs
  - Gérer 2D, 3D, vidéo, audio, réseau, CD, DVD, clé USB, ...
  - Plusieurs utilisateurs (itinérants) --> multi-utilisateurs
  - être extensible

- De plus en plus gros et complexe :
  - Efficace, évolutif, maintenable

# Exigences de l'utilisateur

- « Faut que ça marche ! »
  (comme j'en ai envie ...)
- « Ça imprime pas ... »
- = Machine utilisable (machine étendu)

# Exigences du programmeur

- Simplifier l'accès aux ressources de la machine :
  - Mémoire, processeur, périphériques, fichiers, programmes, réseaux, communication interne
  - Modèle de programmation simple et unifié
- Efficacité dans tous les cas
- = Machine étendue

# Quelques définitions

- Traitement par lots (Batch)
- Systèmes Multi-tâches (Multiprogrammés)
- Systèmes Multi-utilisateurs (centralisé)
  - Temps partagé, transactionnel
- Systèmes Multi-processeurs
- Systèmes temps réel
- Systèmes distribués
- Systèmes Embarqués/connectés

# Traitement par lots (Batch processing)

- Un utilisateur donne plusieurs commandes (« Jobs ») dans une queue d'éxécution de programmes

- Entièrement séquentielle
- p.ex. pour faire plusieurs calculs pendant la nuit

- p.ex. **`autoexec.bat`**

# Systèmes Multi-tache (Multitasking)

- Assurer l'éxécution de **plusieurs programmes** en **meme temps (c-à-d. plusieurs processus)**

- Chaque processus a besoin du processeur
  - situation concurrente
  - solution: « scheduling »

# Systèmes Multi-processeurs

- système avec plusieurs processeurs
  - parallèle
  - vrai multi-tache
  - doit assurer qu'il y a l'éxecution d'autant de processus que processeurs en meme temps
- contrairement: système avec un seul processeur
  - quasi-parallèle
  - arrêter et reprendre les différentes processus
    - Gestion avec le « scheduler » (ordonnancement des processus)

# Systèmes Multi-utilisateurs (« time-sharing »)

- permettre a **différentes personnes** de travailler avec **un ordinateur** en **même temps**
- connexion par
  - via le terminal de l'ordinateur lui-même
  - à distance (telnet, ssh, ftp, ...)
- donner l'impression à chaque utilisateur qu'il est seul
- exige une gestion des droits
  - de fichiers (pour éviter la destruction des fichiers etc.)
  - de processus

# Multi-utilisateurs

- Login
- Type:
  - Administrateur (« root »)
  - Groupes
  - Utilisateurs
- pour gérer les droits

# Systèmes Temps réels

- Sert pour le pilotage et le contrôle des déroulements externes (p.ex. centrale électrique)
- doit garantir des temps de réactions données pour des signaux extérieur urgents
- plusieurs systèmes d'exploitations n'y arrivent pas car l'interruption de certaines activités met le système dans un état instable

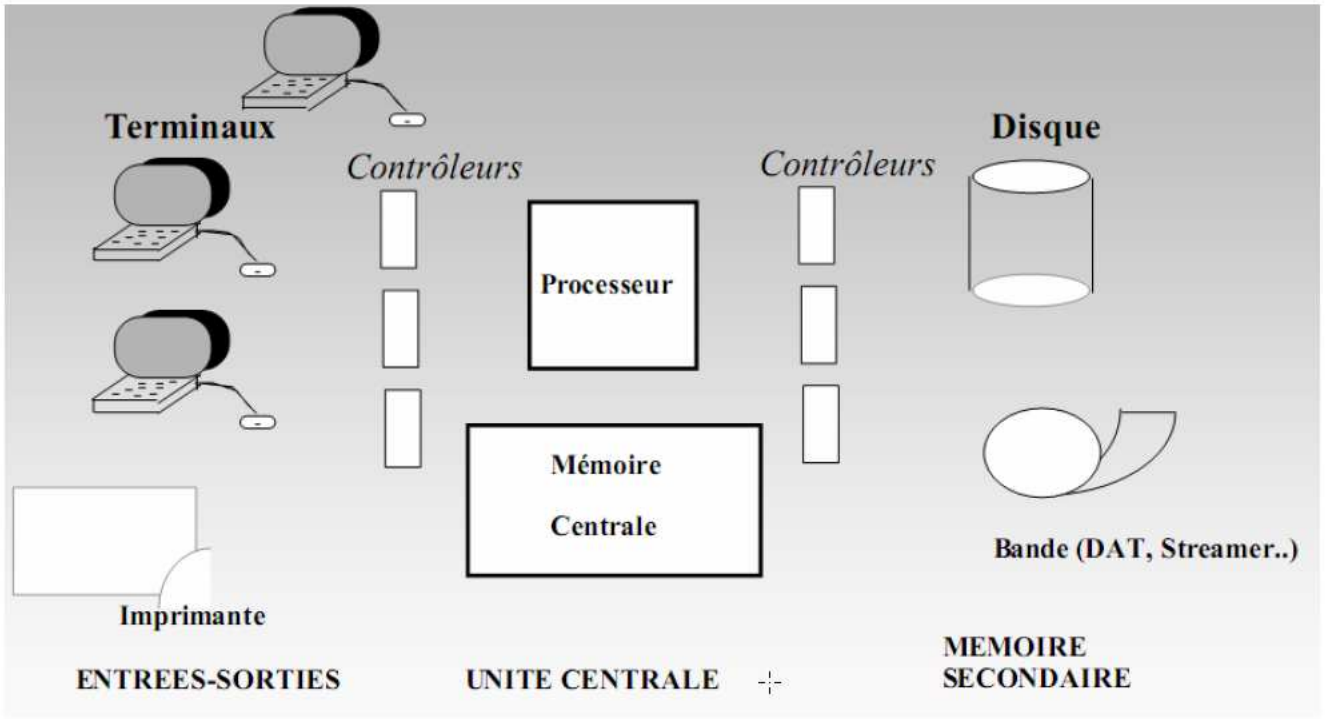
# Systèmes distribués

- doit permettre l'éxecution d'un **seul programme** sur **plusieurs machines**
- distribuer les processus et les remettre ensemble
- pour gros calculs, p.ex. inversion de grandes matrices

# Eléments d'un S.E.

| Matériel | Services de l'OS | Abstraction utilisateur |
|---|---|---|
| Processeur | Gestion de processus, Ordonnancement, signaux, Protection, Synchronisation, Allocation | Processus |
| Mémoire | Allocation, protection, Mémoire virtuelle | Espace d'adressage |
| E/S | Gestion d'iT ; DMA ; Synchronisation | Dispositif physique d'E/S<br>Appels système |
| Disque | Gestion de la persistance des données ; SGF | Fichiers |
| Réseau | Protocole de sécurité ;<br>SGF distribué | RPC ; Serveur de fichiers |

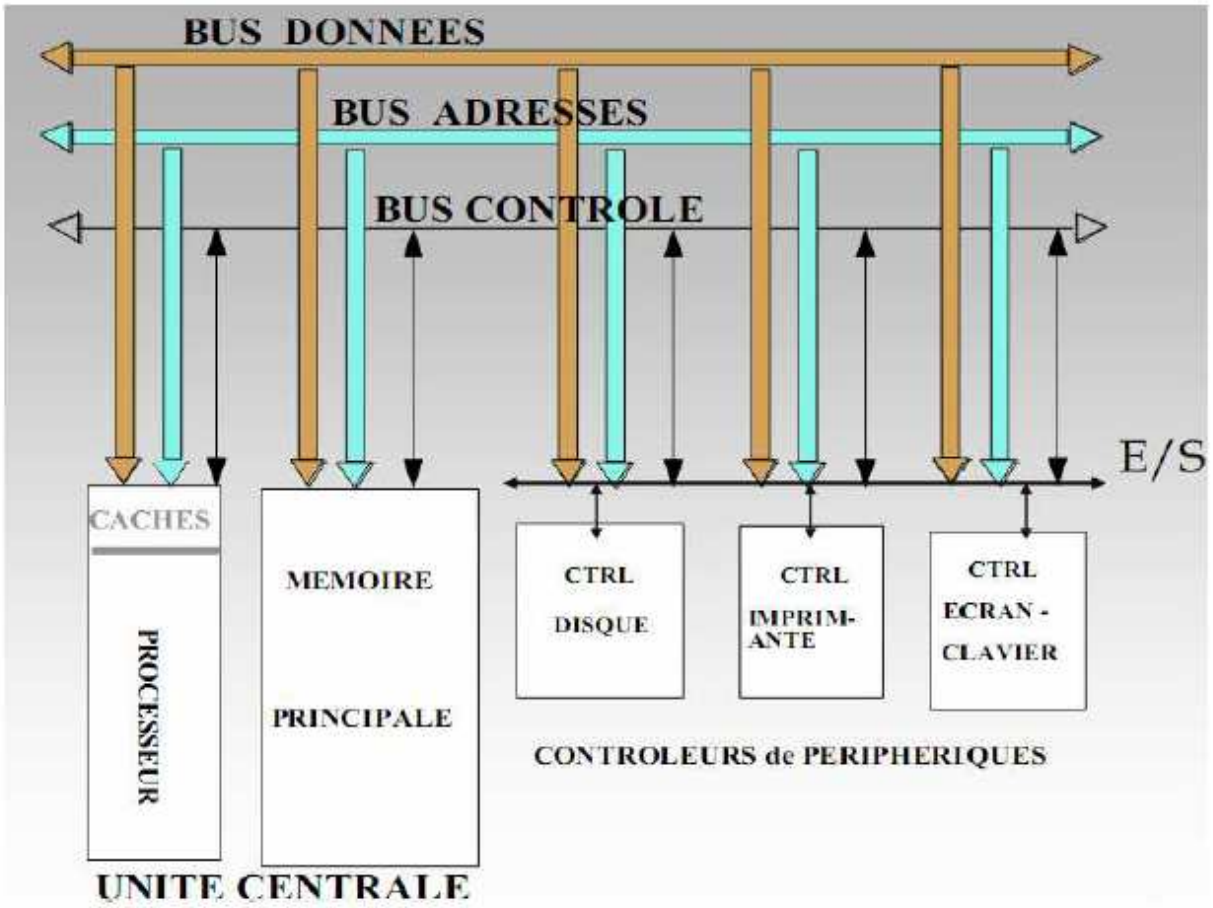
# Les constituants d'un ordinateur



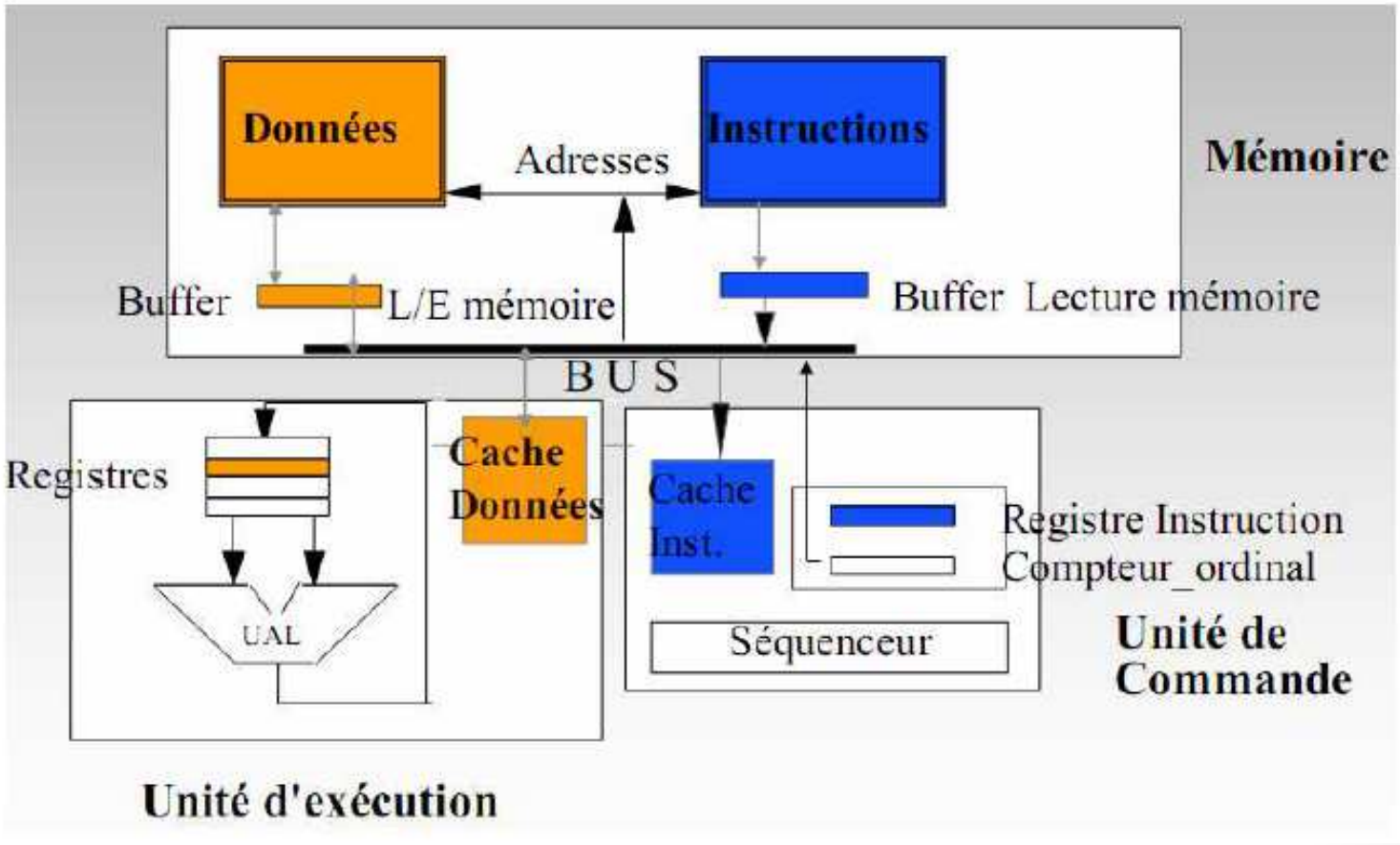
# Le processeur et la mémoire

- Caractéristiques principales
  - La largeur du bus données (64 bits) et la largeur du bus adresse (32/64bits)
  - Le nombre de mots de la mémoire
  - Les temps de cycle (processeur, bus mémoire)
  - Le nombre de cycles par instruction
  - Le débit d'instructions
  - Le processeur est un chip VLSI ~ 100 millions de transistors
  - La mémoire centrale est réalisée à partir de boîtiers de 16, 64, 256, 1024 Méga bits

# Le bus de données/adresses/contrôle



# Architetcure de Von Neuman



# Exemples de processeurs

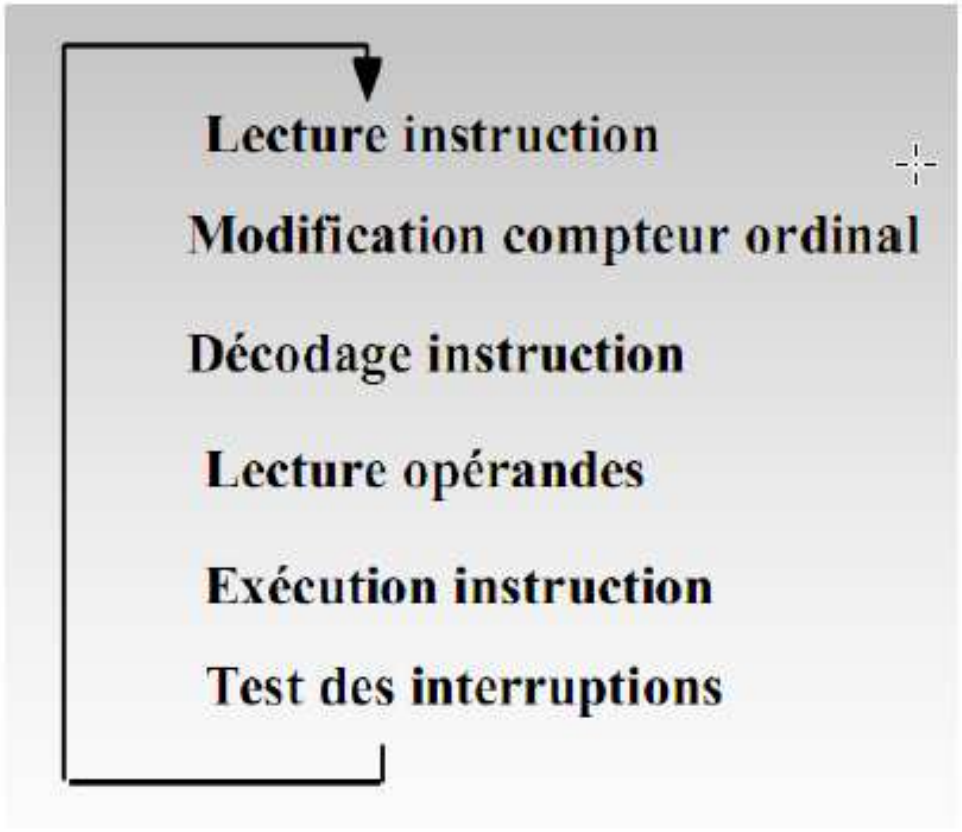
- *Pentium IV 2002*
  - 64 bits données et 32 bits adresses
  - H = jusqu'à 2.53 Ghz.
  - Cache L1 :12K/8K - L2 : 512 K on chip
  - Nb transistors = 42 millions


- *IBM Power IV*
  - 64 bits adresses et données
  - H = 1.3 Ghz
  - Cache 64 K/64K - L2 : 1,4 M. on chip 128 M off chip
  - Nb transistors = 170 millions
  - 2 procs + cache L2 on chip

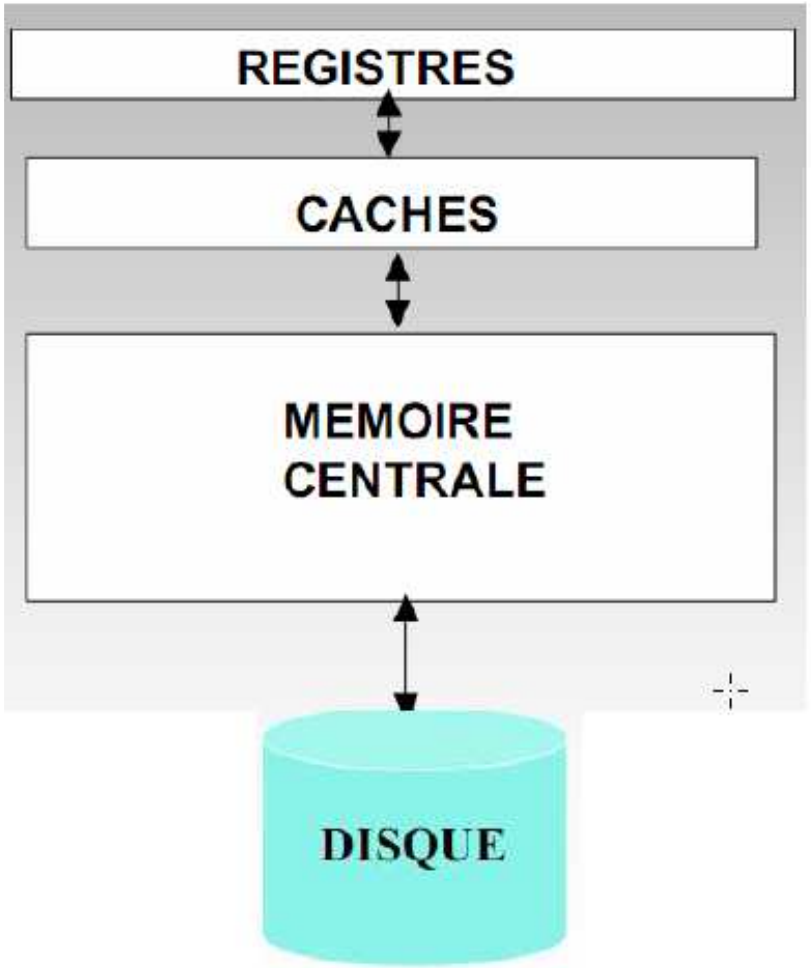
- *Pentium4 2004*
  - 64 bits adresses et 32/64 données
  - H = 3.6 Ghz
  - Cache L1 :12K/16K - L2 : 1M on chip
  - Nb transistors = 100 millions


- *IBM Power5 (Bi-processeur)*
  - 64 bits adresses et données
  - H = 1.9 Ghz
  - Cache 64 K/32K - L2 : 1,9 M. on chip 36 M off chip
  - 2 procs + cache L2 on chip

# Algorithme exécuté par le processeur



# La hiérarchie mémoire



# La mémoire centrale

- La mémoire, contenant des instructions et des données, est organisée en un ensemble de mots numérotés consécutivement à partir de 0
- Le N° associé à chacun de ces mots est l'adresse physique
- L'ensemble de ces adresses forme l'espace d'adressage physique de la mémoire

# Gestion de la mémoire centrale

- Taille mémoire limitée même si de plus en plus importante

- Découpage de la mémoire en :
  - zones de taille variable
  - pages de taille fixe : la pagination
  - segments (chaque segment contient un nombre fixe de pages mémoire) : la segmentation

# Gestion des E/S et des fichiers

- Les dispositifs d'E/S
- Intégration des E/S dans le SE
- Le Système de Gestion de fichiers
  - Le type fichier
  - Les méthodes d'accès
  - L'allocation sur le disque
  - Le RAID
- Exemples UNIX, Linux, Windows

# Gestion des E/S et des fichiers

- Un écran vidéo ; un clavier
- Un disque magnétique
- Une imprimante

- Caractéristiques communes :
  - Les périphériques d'E/S sont beaucoup plus lents que l'unité de traitement

- Libérer l'UC des opérations d'E/S
  - Comment ?

# Les types d'interruption

- Interne à l'unité de traitement
  - division par 0
  - dépassement de capacité numérique
  - erreur de parité sur la mémoire
  - erreur de transmission
  - défaut d'alimentation électrique

- La sollicitation d'une unité périphérique
  - commande pupitre (RAZ - INIT)
  - demande de service pour envoyer ou recevoir une donnée

- Les Appels superviseur provoqués par le programme
  - Lancement de tâches
  - Allocation de ressources matérielles (mémoire - disque - imprimante ...)
  - Réalisés par des instructions de type SVC (Supervisor Call)

- Moyen de communication entre unités de traitements dans une architecture multiprocesseur

# Kernel (Noyau) vs Shell

- Un noyau de système d'exploitation, ou simplement noyau, ou kernel en anglais, est une des parties fondamentales de certains systèmes d'exploitation. Il gère les ressources de l'ordinateur et permet aux différents composants — matériels et logiciels — de communiquer entre eux.

- En tant que partie du système d'exploitation, le noyau fournit des mécanismes d'abstraction du matériel, notamment de la mémoire, du (ou des) processeur(s), et des échanges d'informations entre logiciels et périphériques matériels. Le noyau autorise aussi diverses abstractions logicielles et facilite la communication entre les processus.

- Le noyau d'un système d'exploitation est lui-même un logiciel, mais ne peut cependant utiliser tous les mécanismes d'abstraction qu'il fournit aux autres logiciels. Son rôle central impose par ailleurs des performances élevées. Cela fait du noyau la partie la plus critique d'un système d'exploitation et rend sa conception et sa programmation particulièrement délicates. Plusieurs techniques sont mises en œuvre pour simplifier la programmation des noyaux tout en garantissant de bonnes performances.

# Kernel (Noyau) vs Shell

- Les premiers concepteurs informatiques avaient l'habitude de décrire les différentes couches logicielles d'un système par une analogie : celle du noyau de la noix.

- En anglais, le mot « kernel » désigne le cerneau, la partie comestible de la noix, c'est-à-dire son cœur. À l'inverse, la coque (partie non comestible de la noix) est une enveloppe très dure qui entoure la partie comestible. En anglais cette coque est appelée « shell ».

# Kernel (Noyau) vs Shell

- le noyau d'un système d'exploitation est le logiciel qui assure :
  - la communication entre les logiciels et le matériel ;
  - la gestion des divers logiciels (tâches) d'une machine (lancement des programmes, ordonnancement…) ;
  - la gestion du matériel (mémoire, processeur, périphérique, stockage…).

- La majorité des systèmes d'exploitation est construite autour de la notion de noyau. L'existence d'un noyau, c'est-à-dire d'un programme unique responsable de la communication entre le matériel et le logiciel, résulte de compromis complexes portant sur des questions de performance, de sécurité et d'architecture des processeurs.

# Le processus

# Représentation de processus

- Processus :
  - entité d'exécution (séquence d'actions) d'un programme
  - exécution sur un processeur
  - représentation de ce processus (par une structure de données) dans le moteur d'exécution (le noyau du système)

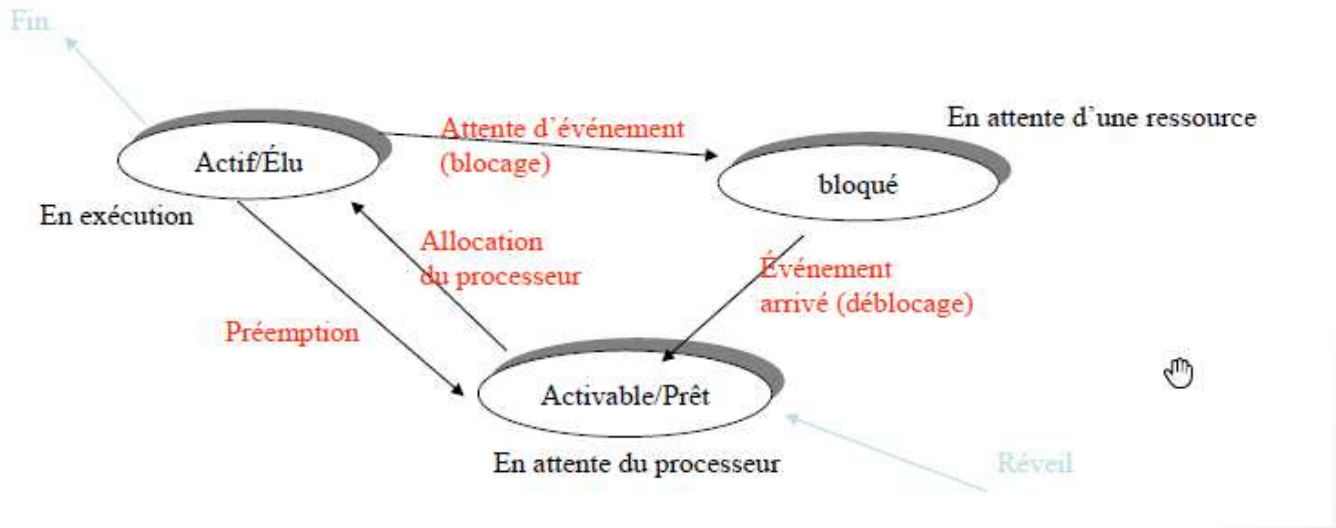
- Décomposition d'une application en processus
  - statique : le nombre des processus est fixe et connu dès le départ
  - dynamique : création et destruction pendant le déroulement

- Niveau et hiérarchie de processus
  - un seul niveau : tous égaux (ex. Concurrent Pascal)
  - hiérarchie de processus avec emboîtement (ex. Unix, Ada): relation père-fils et arborescence de la descendance, le père doit attendre que tous ses fils aient terminé

- Création de processus
  - par déclaration d'un modèle (type, classe) et instanciation d'un objet selon ce modèle (ex. Ada, Java)
  - par création d'une copie (clone) du père dans son état au moment de la création du fils (ex. Unix, Linux : fork)

# Gestion système des processus

- États d'un processus
  - prêt (activable): le processus possède toutes les ressources pour s'exécuter sauf le processeur
  - élu (actif): a obtenu le processeur qui exécute les instructions de son code
  - bloqué : processus bloqué en attente d'un événement (fin d'E/S)



# Création des processus

- Initialisation de processus
  - passage de paramètres à la création (ex. Unix/Linux : **exec**)
  - pas de paramètre, donc acquisition explicite par le processus une fois créé

  (ex. Unix/Linux : **fork** crée un processus avec duplication du code et données)

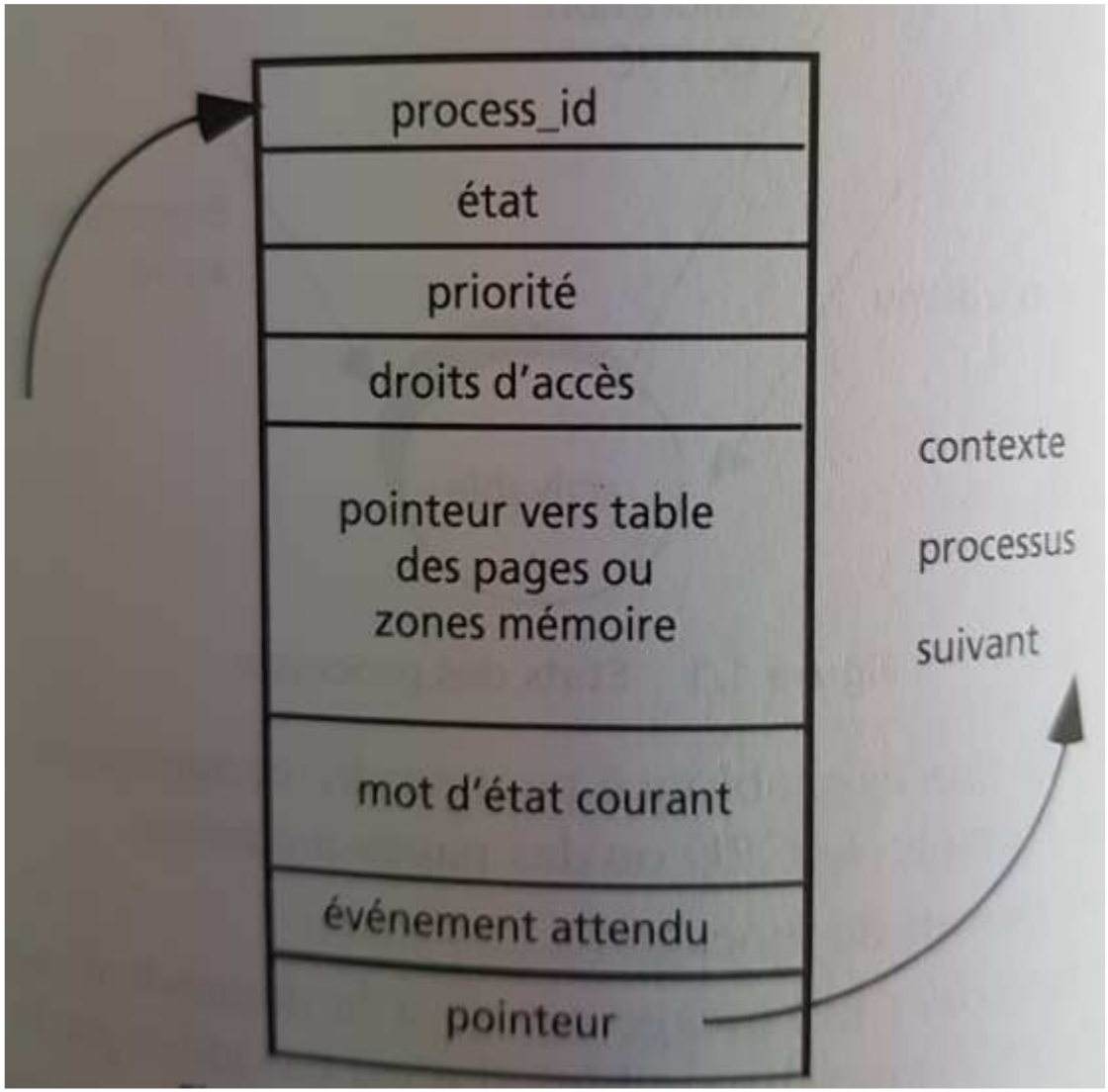
- Démarrage ou activation des processus
  - à la création (les processus sont créés actifs, en Ada)
  - à l'instanciation de classes (création de threads en Java)

- Mode de comportement des processus: séquentiel, cyclique, périodique, etc.

- Terminaison de processus
  - fin correcte du code
  - terminaison imposée par le noyau (erreur, dépassement de ressource)
  - auto terminaison (suicide) sur exception
  - terminaison forcée par un autre processus(« CTRL/C » sous Unix/Linux)
  - jamais : processus cyclique permanent ("démon")
  - jamais : boucle infinie par erreur

# Descripteur de processus

- Bloc de contrôle (utilisé par le système) comporte :
  - identificateur repérant de manière unique chaque processus
  - état du processus
  - environnement volatile (copie des registres du processeur, du compteur ordinal et du pointeur de pile, à la fin de la dernière élection)
  - liens permanents de chaînage vers les descripteurs de ressources


process_id
état
priorité
droits d'accès
pointeur vers table des pages ou zones mémoire
mot d'état courant
événement attendu
pointeur
contexte processus suivant

# Ordonnancement de processus

- Allocation du processeur
  - sans réquisition possible (préemption) jusqu'à fin de l'état actif
  - avec réquisition si :
    - activation d'un processus plus "prioritaire"
    - baisse dynamique de "priorité"
    - fin de quantum d'allocation

- Quelques règles de gestion des processus prêts:
  - ancienneté (FIFO)
  - quantum de taille fixe ou variable
  - priorités fixes ou dynamiques

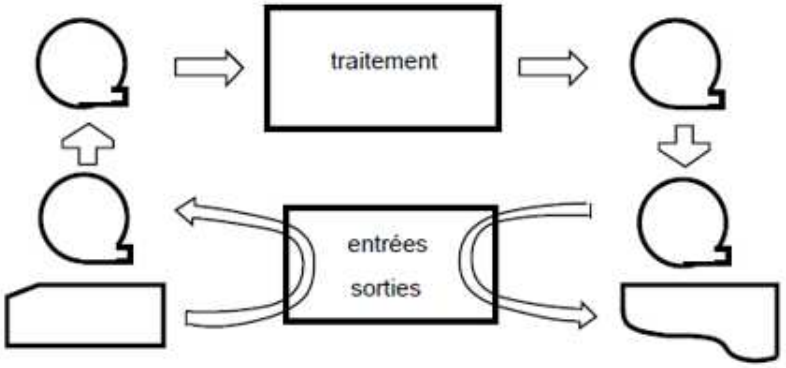
# Ordonnancement de processus Linux

- Trois politiques d'ordonnancement sous Linux :
  - SCHED_FIFO : pour ordonnancer des processus non préemptibles
  - SCHED_RR : pour ordonnancer des processus préemptibles (quantum de temps)
  - SCHED_OTHER : pour ordonnancer des processus quelconques
- Les processus de la SCHED_FIFO sont plus prioritaires que ceux de la SCHED_RR qui eux sont plus prioritaires que ceux de la SCHED_OTHER
- Les priorités :
  - varient de 1 à 99 dans la SCHED_FIFO et la SCHED_RR (le processus de priorité 99 est le plus prioritaire)
  - égales à 0 dans la SCHED_OTHER

# Session 2

# Exercices

# Evolution des performances




- Petit Système d'exploitation

# Gestion des processus

# Applications comportant des processus concurrents

- Motivations pour les processus concurrents
  - améliorer l'utilisation de la machine (cas monoprocesseur)
  - multiprogrammation sur des serveurs
  - parallélisme entre le calcul et les E/S sur PC
  - lancer des sous-calculs dans une machine parallèle ou répartie
  - créer des processus réactifs pour répondre à des requêtes ou à des événements externes (systèmes temps réel, systèmes embarqués et/mobiles)

- Construire des applications concurrentes, parallèles ou réparties

# Critères de découpage en processus concurrents

- Parallélisme physique présent dans l'application
  - un processus pour chaque organe d'entrée/sortie : imprimante, souris,
  - processus clients et processus serveurs dans un système réparti
  - processus de lecture de capteurs dans un système embarqué (réseaux de capteurs)

- Parallélisme logique de l'application
  - activités avec différentes échéances ou importances
  - activités avec différents comportements : périodiques, réactives, cycliques
  - activités de natures différentes : calcul, acquisition, présentation, client, serveur
  - activités distantes, réparties ou mobiles
  - regroupement des actions fonctionnellement proches pour limiter les interactions interprocessus

# Définitions

- Processus:
  - Instance d'un programme en exécution
  - et un ensemble de données
  - et un ensemble d'informations (BCP, bloc de contrôle de processus)

- Deux types de processus
  - Système: lancés par le système d'exploitation (démons) ou le super-utilisateur
  - Utilisateur: lancés par les utilisateurs

- Code de retour d'un processus
  - =0: fin normale
  - #0: comportement anormal (en cours d'exécution ou à la terminaison)

# Définitions

- La table des descripteurs de fichiers
  - Chaque processus peut posséder au maximum OPEN_MAX descripteurs de fichier
  - En plus des descripteurs qu'un processus hérite de son père, il peut en créer d'autres avec open, dup, …

- Enchaînement d'exécution des processus
  - Si les commandes associées sont séparées par ";"
  - Exemple: $commande_1$; $commande_2$; … ; $commande_n$
  - Ps –ef ; ls –l; pwd
  - Remarque:
    - Une erreur dans un processus n'affecte pas les suivants

- Processus coopérants et communication par tube (pipe)
  - –Communication par l'intermédiaire de tube. Les résultats d'un processus servent de données d'entrée à l'autre
  - –Exemple: $commande_1$ | $commande_2$ | … |$commande_n$
  - Ps –ef | wc -l

# Descripteur des processus Linux

- Bloc de contrôle (utilisé par Linux) comporte
  - État du processus
  - Priorité du processus
  - Signaux en attente
  - Signaux masqués
  - Pointeur sur le processus suivant dans la liste des processus prêts
  - Numéro du processus
  - Numéro du groupe contenant le processus
  - Pointeur sur le processus père
  - Numéro de la session contenant le processus
  - Identificateur de l'utilisateur réel
  - Identificateur de l'utilisateur effectif
  - Politique d'ordonnancement utilisé
  - Temps processeur consommé en mode noyau et en mode utilisateur
  - Date de création du processus, etc.

# Visualisation de processus

- La commande «**ps**» permet de visualiser les processus existant à son lancement sur la machine

- Sans option, elle ne concerne que les processus associés au terminal depuis lequel elle est lancée

```
$ ps -cflg
  F S UID      PID   PPID  PGID   SID  CLS PRI ADDR    SZ WCHAN  STIME TTY        TIME CMD
000 S invite  8389  8325  8389  8389    -  25    -   749 wait4  10:15 pts/2  00:00:00 /bin/bash
000 R invite  11364 8389 11364  8389    -  25    -   789 -      11:22 pts/2  00:00:00 ps -cflj

$
```

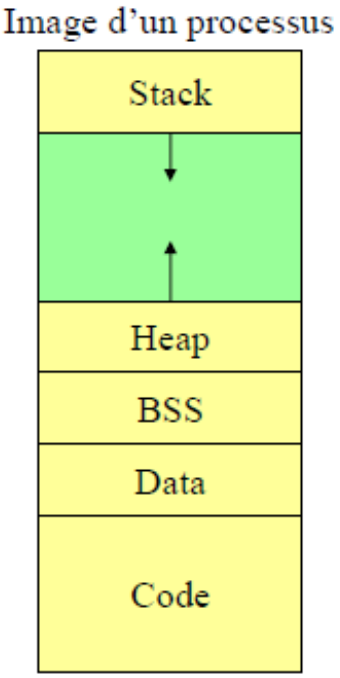
```
$ ps -l
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S   501  8389  8325  0  73   0    -   749 wait4  pts/2    00:00:00 bash
000 R   501 11370  8389  0  79   0    -   788 -      pts/2    00:00:00 ps

$
```

# Informations de la commande ***ps***

| Nom | Interprétation | Option(s) |
|---|---|---|
| S | Etat du processus<br>S: endormi, R: actif, T: stoppé, Z: terminé (zombi) | -l |
| F | Indicateur de l'état du processus en mémoire (swappé, en mémoire, …) | -l |
| PPID | Identificateur du processus père | -l –f |
| UID | Utilisateur propriétaire | -l –f |
| PGID | N° du groupe de processus | -j |
| SID | N° de session | -j |
| ADDR | Adresse du processus | -l |
| SZ | Taille du processus (en nombre de pages) | -l |
| WCHAN | Adresse d'événements attendus (raison de sa mise en sommeil s'il est endormi) | -l |
| STIME | Date de création | -l |
| CLS | Classe de priorité<br>(TS → temps partagé, SYS→ système, RT→ temps réel). | -cf –cl |
| C | Utilisation du processeur par le processus | -f –l |
| PRI | Priorité du processus | -l |
| NI | Valeur « nice » | -l |

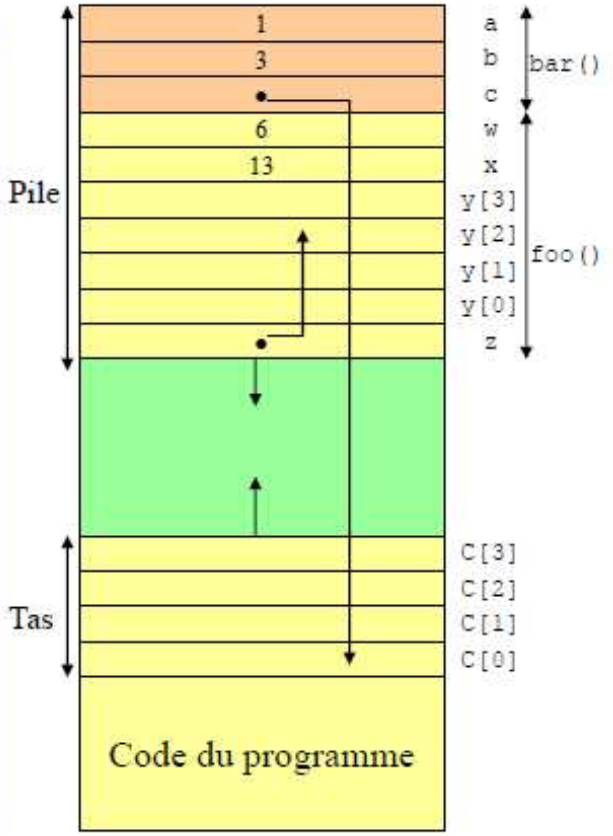
# Processus en mémoire

- Pile (stack):
  - Les appels de fonctions et variables locales aux fonctions exigent une structure de donnée dynamique
  - Variables locales et paramètres
  - Adresse de retour de fonction
- Tas (heap)
  - Allocation dynamique des variables
- Code (Text)
  - Code des fonctions
  - Lecture seulement (read-only)
- Données (Data+BSS)
  - Constantes
  - Variables initialisées / non-initialisées



# Processus en mémoire

```
void foo(int w) {
   int x = w+7;
   int y[4];
   int *z = &y[2];
}

void bar(int a) {
   int b = a+2;
   int *c = (int *) malloc(sizeof(int)*4);
   foo(b+3);
}

main() {
   bar(1);
}
```



# Ordonnancement de processus

- L'ordonnanceur (scheduller) d'Unix:
  - Associe une priorité dynamique à chaque processus, recalculée périodiquement
  - Utilise la stratégie du tourniquet (round robin) pour les processus de même priorité

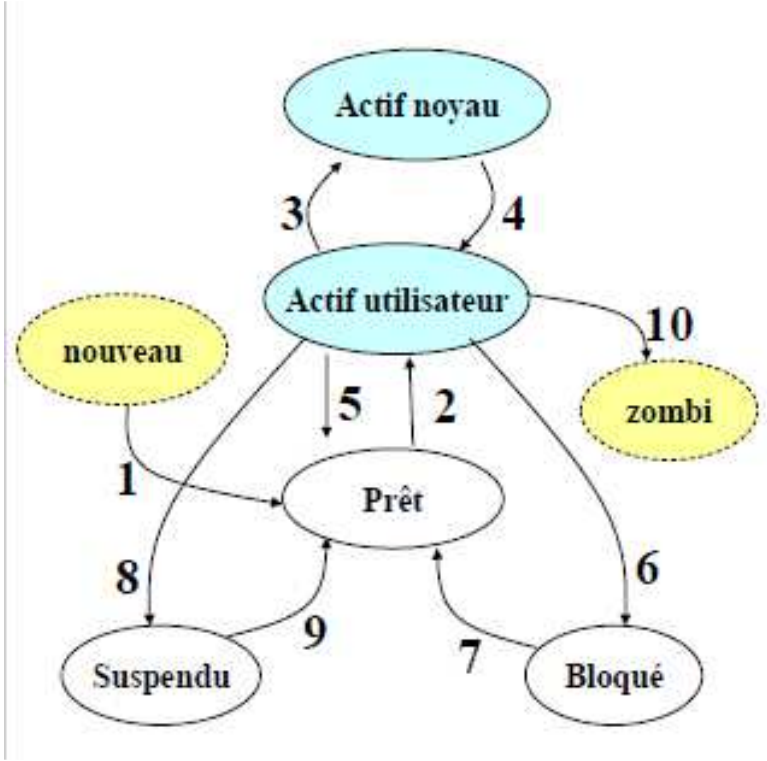
- La fonction de calcul de la priorité courante utilise les paramètres suivants:
  - Une priorité de base: correspondant au niveau d'interruption du processus
  - Le temps CPU consommé récemment
  - Une priorité ajustable par le processus lui-même grâce à:

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

  - qui permet d'augmenter la valeur de incr[0-39] et donc de diminuer sa priorité

- Le super-utilisateur peut augmenter la priorité en donnant une valeur <0 à incr.

# État d'un processus




1. Allocation de ressources
2. Élu par l'ordonnanceur
3. Le processus revient d'un appel système ou d'une interruption
4. Le processus a fait un appel système ou a été interrompu
5. A consommé son quantum de temps
6. Se met en attente d'un événement
7. Événement attendu est arrivé
8. Suspendu par un signal SIGSTOP
9. Réveil par le signal SIGCONT
10. fin

# Création de processus

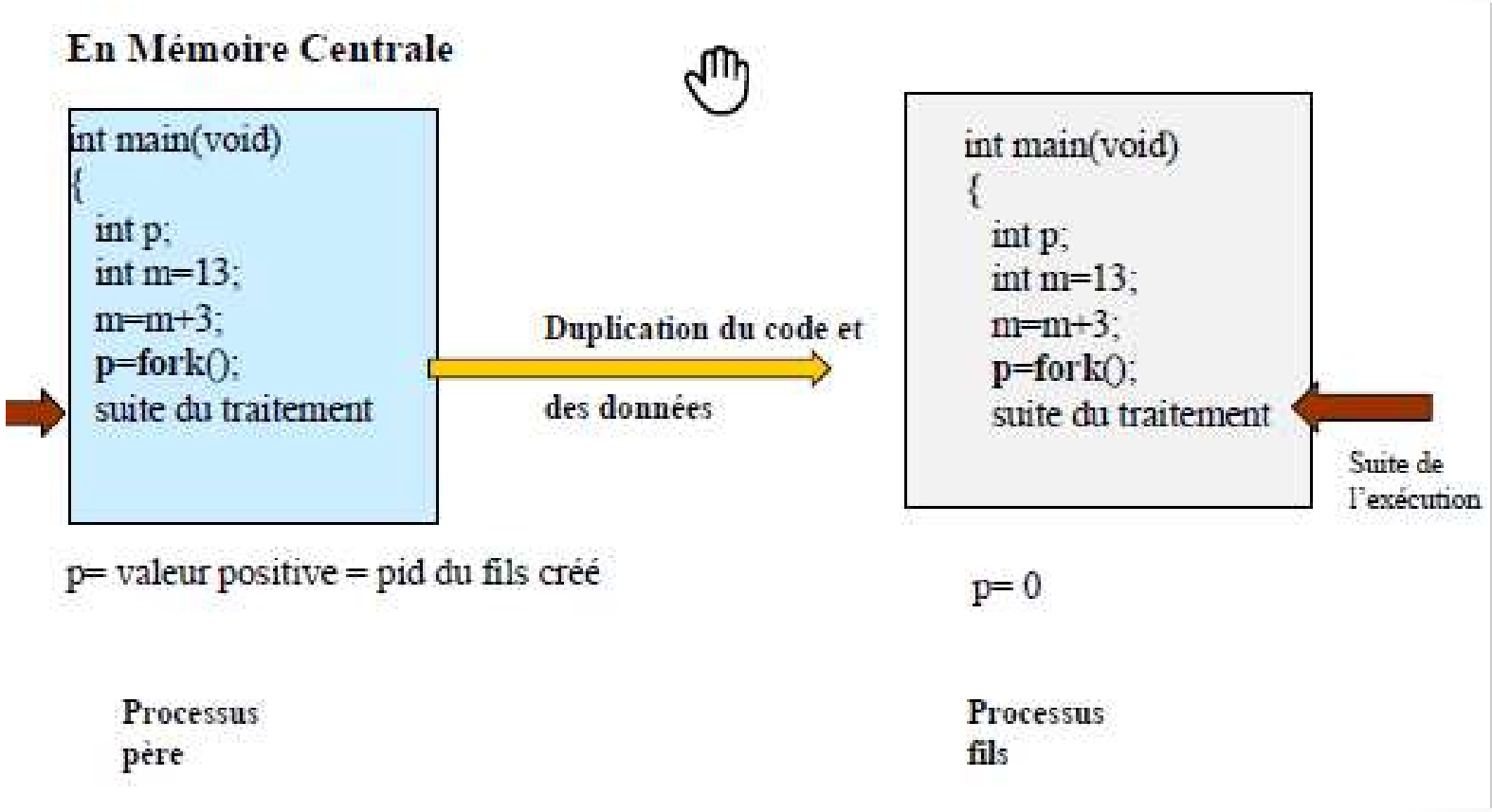
- La primitive **fork()** permet de créer un nouveau processus (fils) par duplication

- Afin de distinguer le père du fils, **fork()** retourne:
  - -1 : en cas d'erreur de création
  - 0 : indique le fils
  - > 0 : le **pid** du processus fils au processus père

- Les primitives **getpid()** et **getppid()** permettent d'identifier un processus et son père.

```
#include <unistd.h>
pid_t fork(void)     // crée un nouveau processus
pid_t getpid(void)   // donne le pid du processus
pid_t getppid(void)  // donne le pid du père du processus
```

# Création de processus



# Exemple 1 -Création d'un processus

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
   int p;

   p=fork();

   if (p==0) printf("je suis le processus fils\n");
   if (p>0) printf("je suis le processus père\n");

   return 0;
}
```



```
je suis le processus père
je suis le processus fils
```

# Exemple 1 -Création d'un processus

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
   int p;

   p=fork();

   if (p==0) printf("je suis le processus fils\n");
   if (p>0) printf("je suis le processus père\n");

   return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
   int p;

   p=fork();

   if (p==0) printf("je suis le processus fils\n");
   if (p>0) printf("je suis le processus père\n");

   return 0;
}
```

```
je suis le processus père
je suis le processus fils
```

# Exemple 2 -Création d'un processus

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void){
   int p;

   p = fork();

   if (p > 0)
      printf("processus père: %d-%d-%d\n", p, getpid(), getppid());

   if (p == 0) {
      printf("processus fils: %d-%d-%d\n", p, getpid(), getppid());
   }

   if (p < 0)
      printf("Probleme de creation par fork()\n");

   system("ps -l");
   return 0;
}
```

# Exemple 2 -Création d'un processus

```
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S   501  2402  2401  0  71   0    -   748 wait4  pts/1    00:00:00 bash
000 S   501 13681  2402  0  71   0    -   343 wait4  pts/1    00:00:00 exo2
040 S   501 13682 13681  0  71   0    -   343 wait4  pts/1    00:00:00 exo2
000 R   501 13683 13681  0  74   0    -   786 -      pts/1    00:00:00 ps
000 R   501 13684 13682  0  74   0    -   789 -      pts/1    00:00:00 ps

processus fils: 0-13682-13681

  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S   501  2402  2401  0  71   0    -   748 wait4  pts/1    00:00:00 bash
000 S   501 13681  2402  0  71   0    -   343 wait4  pts/1    00:00:00 exo2
044 Z   501 13682 13681  0  65   0    -     0 do_exi pts/1    00:00:00 exo2 <def>
000 R   501 13683 13681  0  75   0    -   788 -      pts/1    00:00:00 ps

processus père: 13682-13681-2402
```