Mise à niveau
Systèmes d'Exploitation

Master 1
Informatique des Organisations - MIDO

Joyce EL HADDAD
elhaddad@lamsade.dauphine.fr

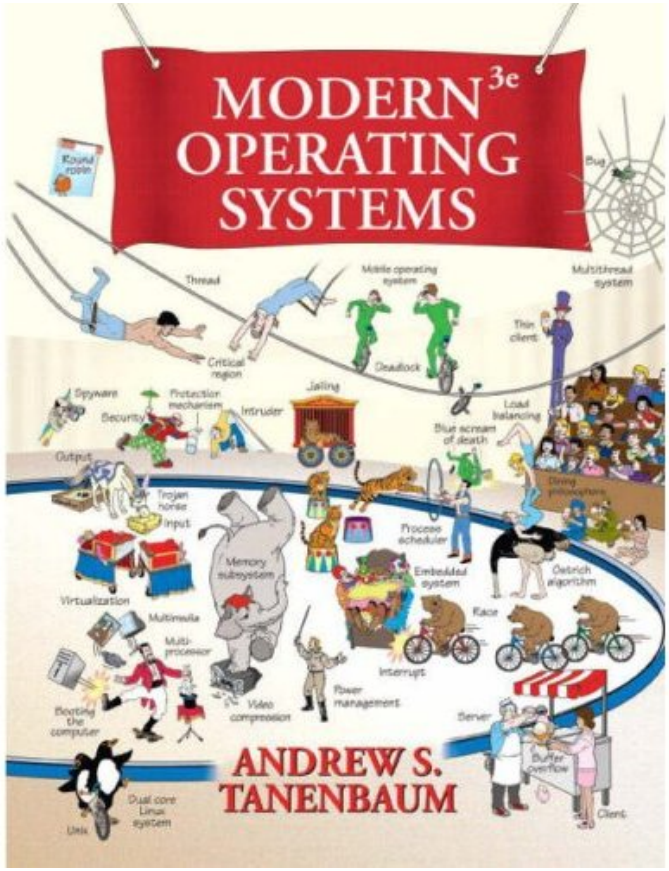
Plan

- Généralités
  1. Historique
  2. Fonctions
  3. Architectures

- Gestion des processus
  1. Définition
  2. Ordonnancement
  3. Concurrence
  4. Communication

LA Référence

- Andrew Tanenbaum, *Modern operating system*, Prentice Hall, 3th Edition, 2007



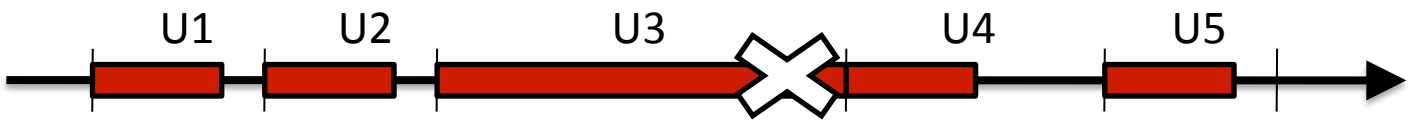
Plan

- Généralités
  1. Historique
  2. Fonctions
  3. Architectures

- Gestion des processus
  1. Définition
  2. Ordonnancement
  3. Concurrence
  4. Communication

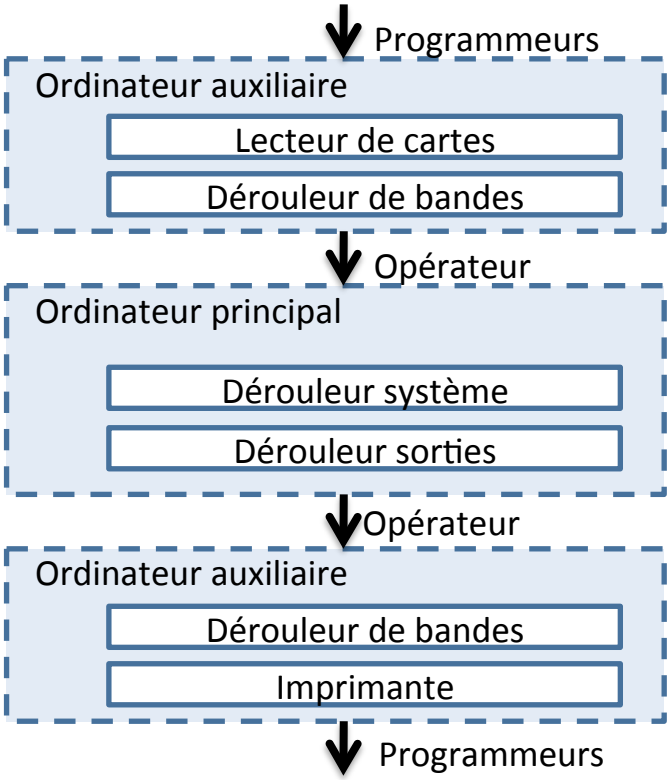
Historique

- 1945-1955 : premiers ordinateurs
  - Peu répandus, sans interface, peu fiable, programmes en binaire
  - Gestion multi-utilisateurs par réservation
  - Utilisateurs « multifonctions » : insertion du programme et des données, exécution, interprétation des résultats…
  - Inconvénients : temps d'inactivité, tâches perdues



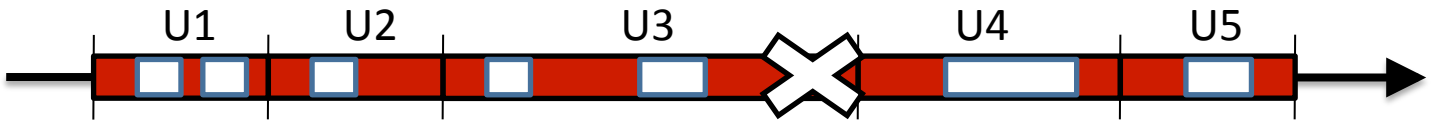
Historique

- 1955-1965 : traitement par lots
  - Programmes en langages évolués (Fortran, …)
  - Lots de cartes à l'opérateur : cartes Fortran en entrée et cartes en langage assembleur en sortie



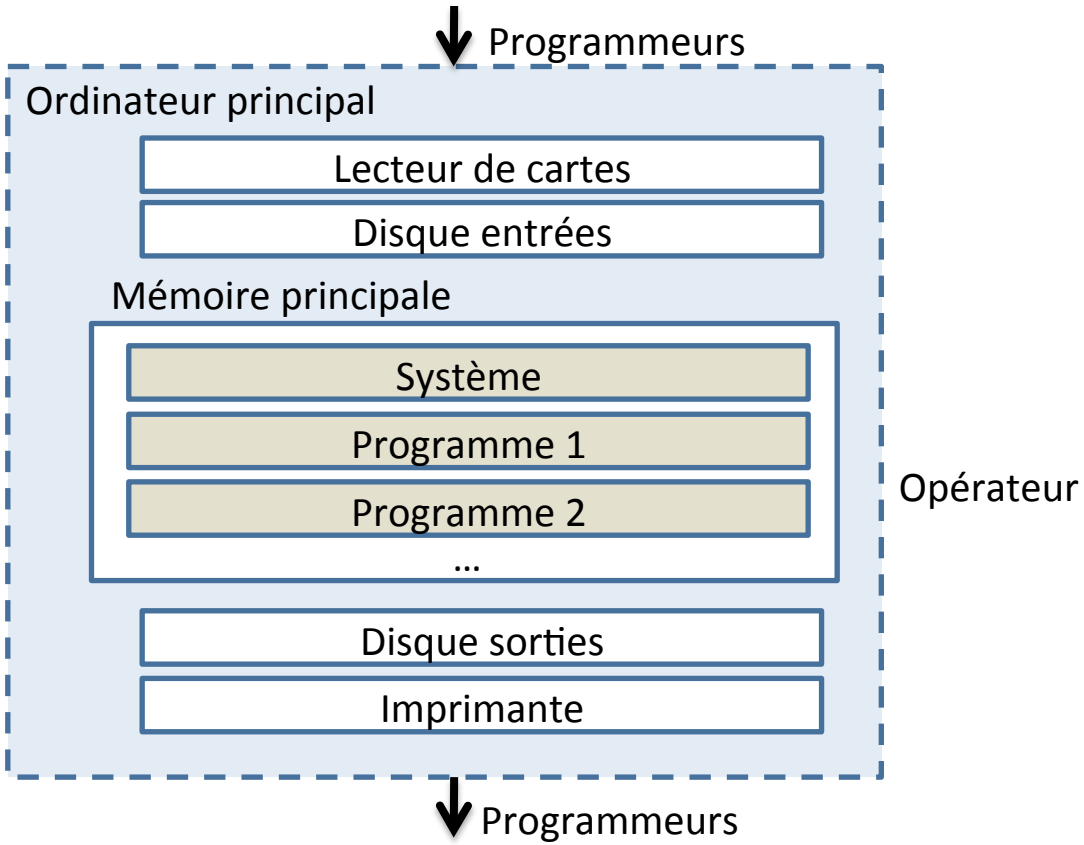
Historique

- 1955-1965 : traitement par lots
  - Traitements par lots séquentiel : exécution successives d'un ensemble de programmes
  - Contrainte : assigner à chaque programme une durée maximale. Les tâches inachevées sont abandonnées
  - Inconvénients : temps d'inactivité entre les Entrées/ Sorties (E/S), tâches perdues



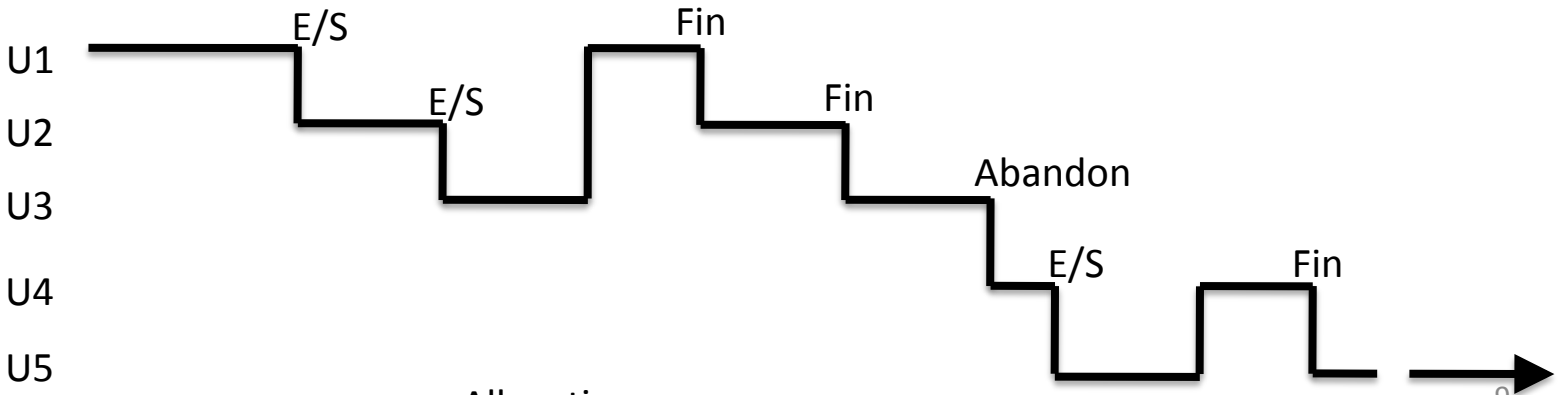
Historique

- 1965-1980 : multiprogrammation
  - Traitements par lots parallèle : exécutions parallèles d'un ensemble de programmes



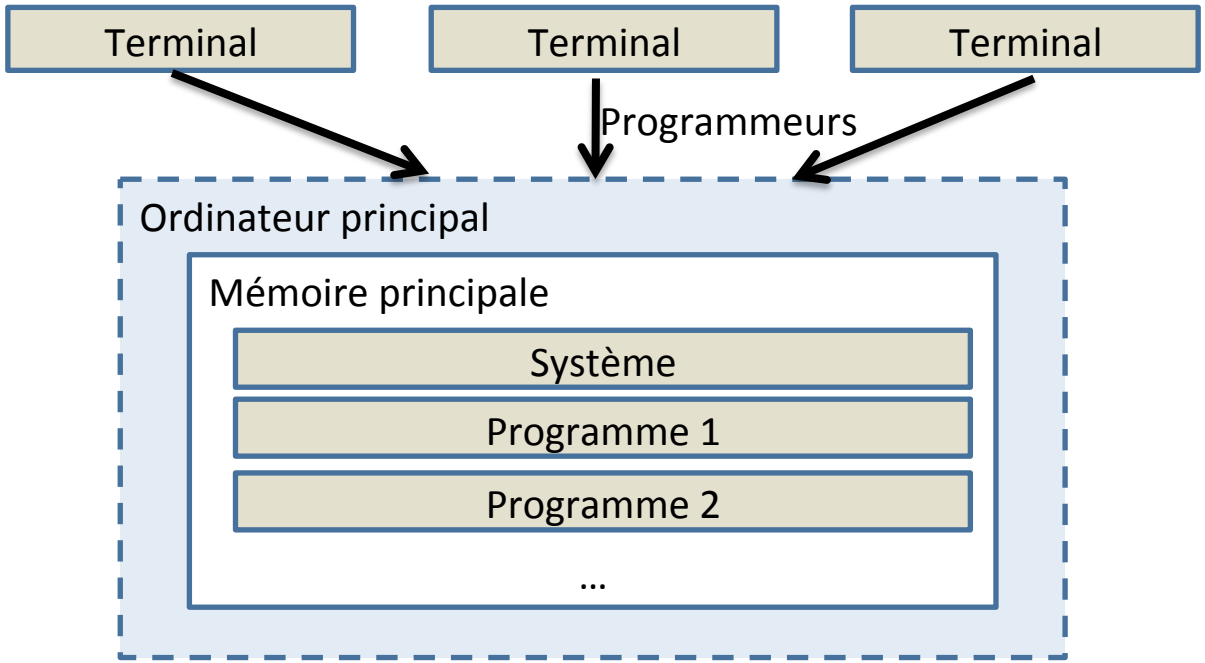
Historique

- 1965-1980 : multiprogrammation
  - À chaque E/S ou Fin, changement de programme
  - Fins des E/S signalées par des interruptions
  - Les tâches inachevées sont abandonnées
  - Inconvénients : temps d'inactivité entre les E/S pour les dernières tâches, tâches perdues, temps de réponse indépendant de la durée de la tâche



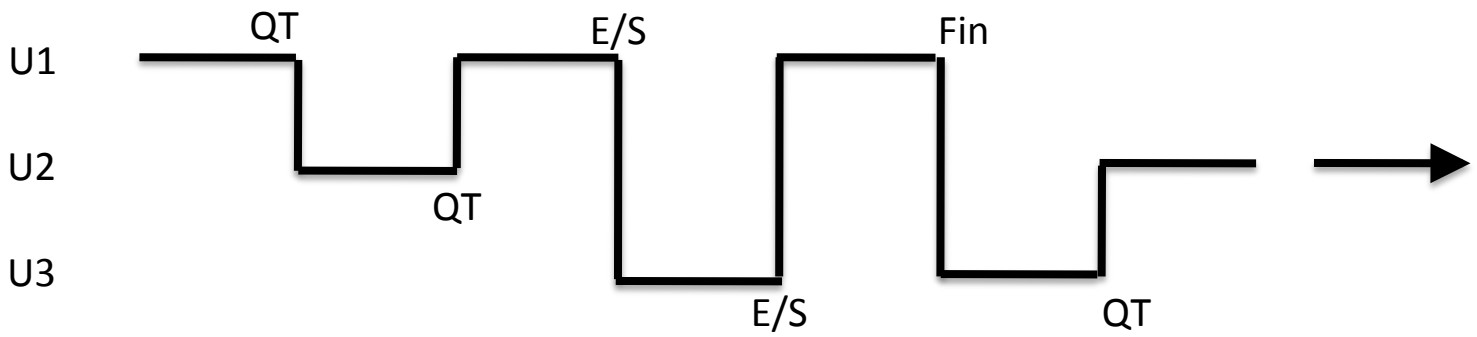
Historique

- 1980- : temps partagé
  - Les programmes n'ont plus de temps d'exécution maximum
  - Le partage du temps en quanta



Historique

- 1980- : temps partagé
  - Affectation du processeur à un programme durant un quantum
  - Interruption du programme si E/S, Fin, ou quantum épuisé
  - Avantage : prise en compte rapide des nouveaux programmes



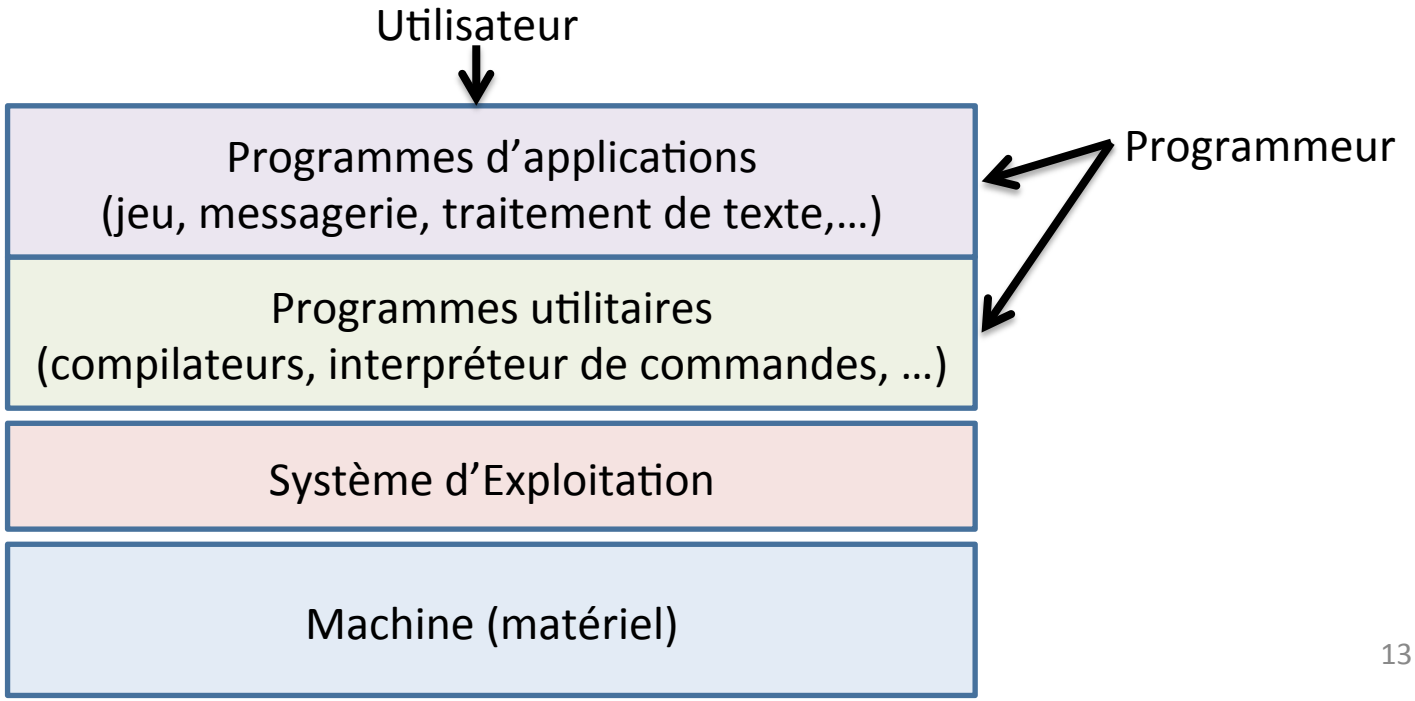
Allocation processeur

Historique

- 1980- : les ordinateurs personnels (deux principaux systèmes d'exploitation MSDOS et UNIX)
- 1990- : les ordinateurs personnels portables (systèmes d'exploitation micronoyau ou modulaires)
- 2000- : les ordinateurs de poches et les tablettes (systèmes d'exploitation iOS, Android,…)
- … et l'évolution continue

Fonctions d'un S.E.

- Deux catégories de programmes :
  - Les programmes utilitaires ou systèmes (éditeurs, compilateurs,…) destinés aux programmeurs
  - Les programmes d'applications (tableurs, traitement de texte,…) destinés aux utilisateurs finaux



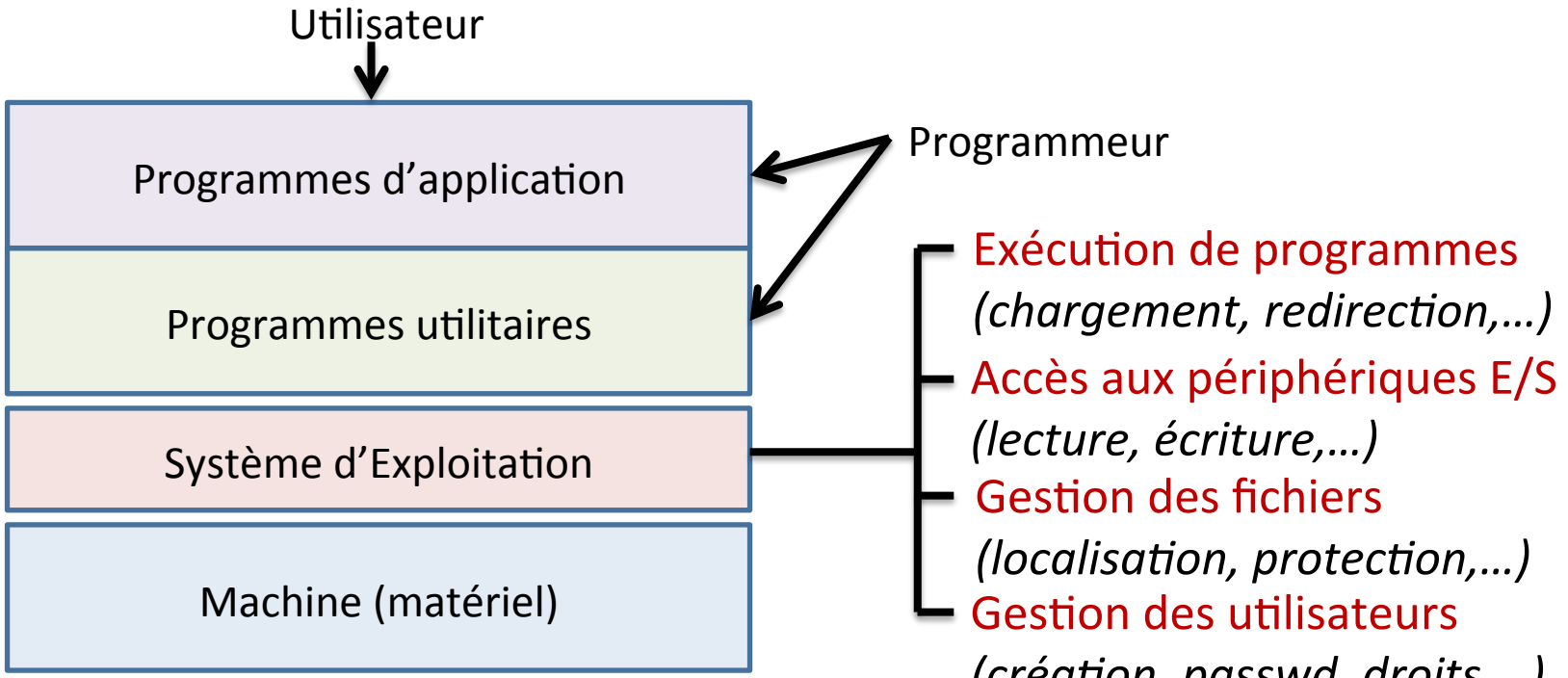
Fonctions d'un S.E.

- Un système d'exploitation (S.E.) est un programme qui :
  - Contrôle l'exécution des programmes utilitaires
  - Contrôle les ressources de l'ordinateur
  - Fournit la base sur laquelle seront construits les programmes d'applications

- Deux fonctions d'un S.E.
  - Machine étendue : Interface entre l'utilisateur et la machine
  - Gestionnaire des ressources de la machine

Fonctions d'un S.E.

- Interface Machine/Utilisateur
  - Rôle : masquer des éléments fastidieux liés au matériel, comme les interruptions, les horloges, la gestion de la mémoire, la gestion des périphériques…
  - Un S.E. fournit des services évolués; ces services sont d'autant plus évolués que l'utilisateur est novice



Fonctions d'un S.E.

- Interface Machine/Utilisateur : Appels Systèmes
  - Les programmes applications et utilitaires s'exécutent en mode utilisateur alors que le S.E. s'exécutent en mode noyau ou superviseur (kernel en anglais, la partie fondamentale d'un S.E.)
  - Pour obtenir un service d'un S.E., un utilisateur doit faire un Appel Système (System Call en anglais)
  - Un appel système est une fonction fournie par le noyau d'un S.E. et utilisée par les programmes s'exécutant dans l'espace utilisateur

Fonctions d'un S.E.

- Interface Machine/Utilisateur : Appels Systèmes
  - Les appels systèmes sont des procédures atomiques classiques :
    - appelées depuis un programme de l'espace utilisateur;
    - exécutées dans l'espace noyau;
    - dont le retour est effectué dans le programme appelant dans l'espace utilisateur.
  - Ils permettent au S.E. de réaliser un service demandé :
    - Gestion de processus *(fork, wait, exit, kill,…)*
    - Gestion du système des fichiers *(open, close, read, write,…)*
    - Gestion des utilisateurs et des protections *(chmod,…)*
    - …

Fonctions d'un S.E.

- Interface Machine/Utilisateur : Interpréteur
  - Une connexion à un terminal active un programme (processus) qui:
    - attend une commande *(ex. cat file1 file2 | sort > /dev/lp)*
    - l'interprète et
    - l'exécute directement ou indirectement par création d'un processus *(ex. concaténation de file1 et file2, le résultat est envoyé à sort qui trie dans l'ordre alphabétique et le résultat est envoyé à l'imprimante)*
  - Ce processus exécute un programme appelé l'interpréteur de commandes

Fonctions d'un S.E.

- Interface Machine/Utilisateur : Interpréteur
  - Un interpréteur traite les commandes tapées au clavier par l'utilisateur et lui masque les appels systèmes
  - Un interpréteur est un programme utilitaire ne faisant pas partie du noyau d'un S.E. mais faisant partie des composants de base
  - Plusieurs interpréteurs possibles : *sh (shell), csh (cshell), ksh (korn shell) sous Unix, cmd.exe sous Windows NT et ses dérivés , tcsh ou bash sous Mac OS X et ses dérivés.*

Fonctions d'un S.E.

- Gestion des ressources
  - Un ordinateur est constitué d'un ensemble de ressources
    - Processeur (CPU), Mémoire (principale), Périphériques E/S (disques, imprimantes,…)
  - Plusieurs fonctionnalités de gestion
    - Processeur : allocation du processeur aux différents programmes
    - Fichiers
    - Entrées/Sorties : accès aux périphériques
    - Mémoire : segmentation et pagination
    - Concurrence : synchronisation pour l'accès à des ressources partagées
    - Protection : respect des droits d'accès aux ressources

Fonctions d'un S.E.

- Gestion des ressources
  - Le noyau d'un S.E. gère l'allocation des ressources aux programmes avec pour objectifs :
    - Efficacité : utilisation maximale des ressources
    - Equité : pas de programme en attente indéfinie
    - Cohérence : entre les accès consécutifs
    - Protection : contre des accès interdits

Fonctions d'un S.E.

- Gestion des ressources : Processus
  - Processus = un programme en cours d'exécution
  - Un processus est caractérisé par : son code exécutable, ses données et sa pile, son compteur ordinal et son pointeur de pile, l'utilisateur qui l'a lancé, sa durée d'exécution, les fichiers qu'il accède…
  - Un processus est crée par un autre processus
  - Les processus sont les « éléments actifs » du système

Fonctions d'un S.E.

- Gestion des ressources : Processus
  - Le système tient à jour l'état des processus : en exécution, prêt, en attente d'évènement, en mémoire principale ou secondaire
  - Gestion de processus : création d'un processus (exécution), terminaison d'un processus (retour), attente de terminaison (père), définition de comportement sur évènement (interruption clavier), allocation mémoire, communication entre processus (envoi de signaux),…

Fonctions d'un S.E.

- Gestion des ressources : Fichiers
  - Fichier = un ensemble de données conservées indépendamment des exécutions des programmes
  - Un fichier est caractérisé par : la structure et le contenu de ses enregistrements, sa localisation, son propriétaire, ses droits d'accès, ses dates de création, modification,…
  - Les fichiers sont les principaux « éléments passifs » du système
  - Gestion des fichiers : création, destruction, ouverture, fermeture, lecture, écriture, accès à un fichier, protection d'un fichier, …

Fonctions d'un S.E.

- Gestion des ressources : Fichiers
  - Les fichiers sont regroupés en Système de Fichiers organisés de manière hiérarchique (un fichier contenant d'autres est un répertoire)
  - Un SGF contient différents types de fichiers :
    - Les fichiers de données : structure et contenu sont accessibles aux processus utilisateurs
    - Les répertoires : seulement le contenu est accessible aux processus utilisateurs
    - Les fichiers spéciaux (périphériques) : inaccessibles aux processus utilisateurs
    - Les fichiers de communication (pipe) : durée de vie ou contenu limitée à l'exécution du processus

Fonctions d'un S.E.

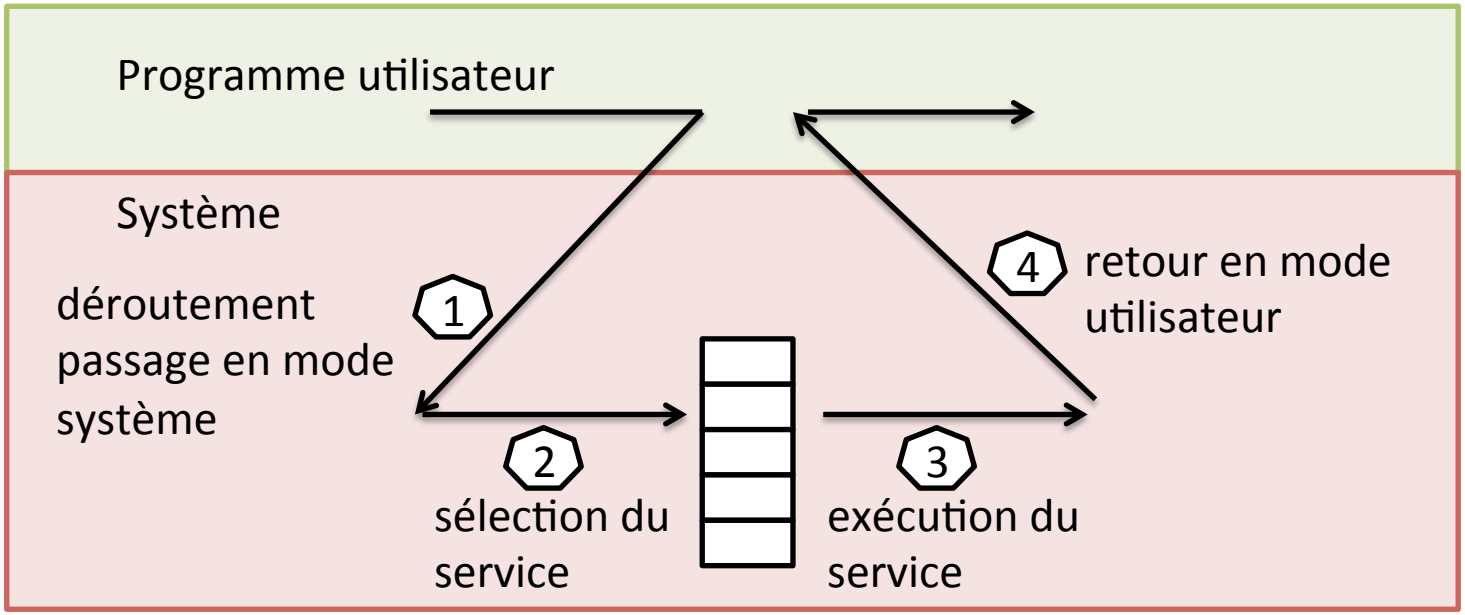
- Gestion des ressources : Fichiers
  - Gestion du système de fichiers : rattachement/ détachement d'un fichier à un répertoire, montage/ démontage d'un système de fichiers,…
  - Gestion des protections : consultation/modification des protections (ou du propriétaires) d'un fichier, consultation /modification de l'utilisateur associé à un processus,…

Architecture d'un S.E.

- Objectifs de l'architecture
  - Efficacité de l'exécution : rapide (pas de perte de temps), stable (pas de bug), complet (accès à toutes les E/S)
  - Taille réduite du code
  - Maintenance et évolution aisée : compatibilité avec les versions successives, corrections des bugs, introduction de nouvelles possibilités (E/S)
- Typologie de l'architecture
  - Systèmes monolithiques
  - Systèmes en couches
  - Machines virtuelles
  - Modèle client/serveur

Architecture d'un S.E.

- Systèmes Monolithiques
  - Ensemble de procédures à interface bien définie (paramètres, code retour) s'appelant mutuellement et qui, compilées, forment le système
  - Pb : difficile d'introduire de nouvelles fonctionnalités, de réécrire certaines procédures (bugs)



Architecture d'un S.E.

- Systèmes en Couches
  - Une description en couches : chaque couche offre des services à la couche supérieure et utilise les services de la couche inférieure
  - Premier système en couches : le système de traitement par lots avec opérateur THE (Dijkstra 1968)
    - Première tentative de créer un S.E. conçu sur des superpositions de niveaux d'abstraction nettement séparés

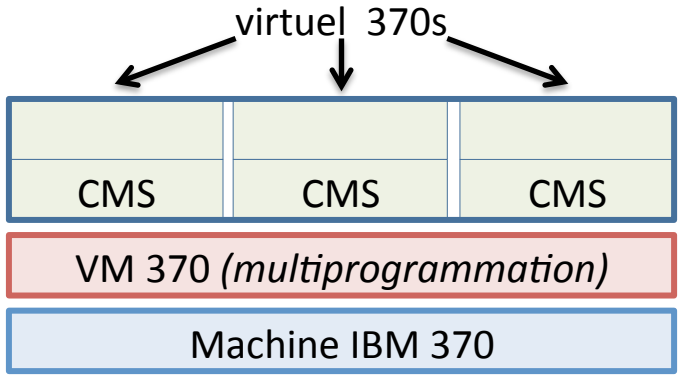
5 : Processus operateur du système
4 : Gestion processus utilisateur
3 : Gestion des E/S
2 : Gestion communication processus-operateur
1 : Gestion de la mémoire
0 : Allocation processeur et multiprogrammation

Architecture d'un S.E.

- Machines Virtuelles
  - Une machine virtuelle est une illusion d'un appareil informatique créée par un logiciel d'émulation
  - Le logiciel simule la présence de ressources matérielles et logicielles telles que la mémoire, le processeur, le disque dur, le système d'exploitation et les pilotes, permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine simulée
  - Les machines virtuelles sont utilisées pour exploiter une machine unique comme s'il y en avait plusieurs (*virtualisation*)

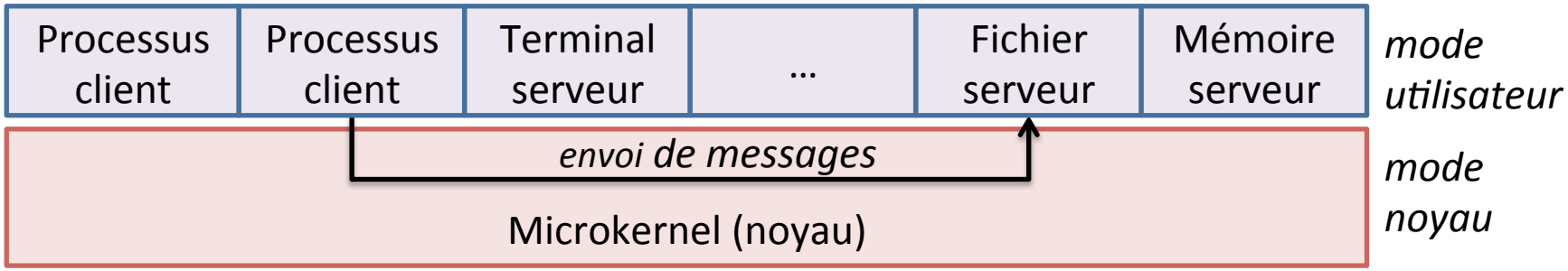
Architecture d'un S.E.

- Machines Virtuelles
  - La création de plusieurs environnements d'exécution sur un seul ordinateur, dont chacun émule l'ordinateur hôte *(le système produit n copies de la machine réelle)*
  - Chaque utilisateur a l'illusion de disposer d'un ordinateur complet alors que chaque machine virtuelle est isolée des autres
  - Le système VM/370 (IBM 1970)
    - CMS (Conversational Monitor System) : un S.E. léger mono-utilisateur utilisé pour le temps partagé



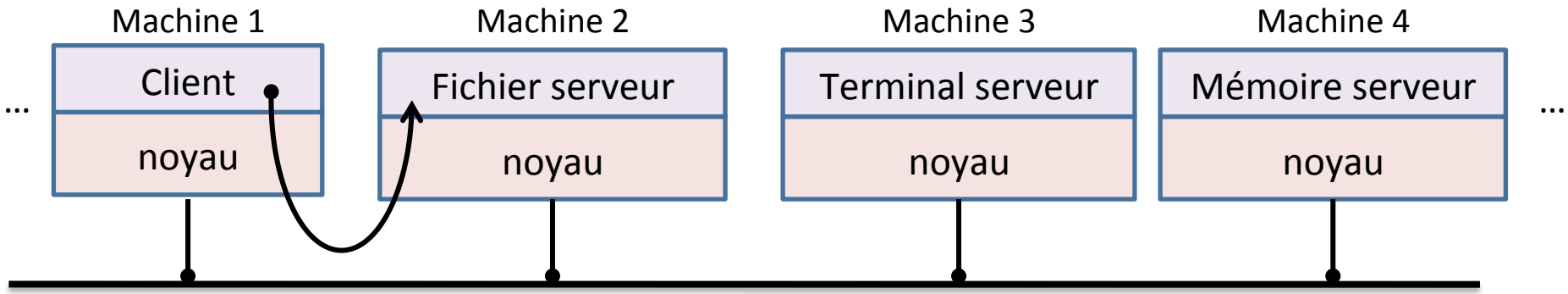
Architecture d'un S.E.

- Modèle Client/Serveur
  - Recherche de simplicité et de flexibilité
  - Réduire le système à un noyau minimal *microkernel* et bâtir les autres fonctions évolués par des processus utilisateurs spécialisés : *les serveurs*
  - Le noyau gère la communication entre clients et serveurs



Architecture d'un S.E.

- Modèle Client/Serveur
  - Avantages :
    - Plus de flexibilité
    - Plus grande facilité de maintenance
    - Tolérance aux pannes (un serveur peut planter sans que cela n'entraîne la panne de la machine)
    - adapté à un environnement distribué



Client/serveur dans un système distribué

Architecture d'un S.E.

- Fonctions/Architecture d'un S.E.
  - Chaque couche a une interface bien définie par en ensembles d'appels système

4 : IHM, Interpréteur de commandes
3 : Gestion des fichiers
2 : Gestion des E/S
1 : Gestion de la mémoire
0 : Gestion des processus *(ordonnancement, gestion des interruptions, E/S de bas niveau)*

Plan

- Généralités
  1. Historique
  2. Fonctions
  3. Architectures

- Gestion des processus
  1. Définition
  2. Ordonnancement
  3. Concurrence
  4. Communication

Définition d'un processus

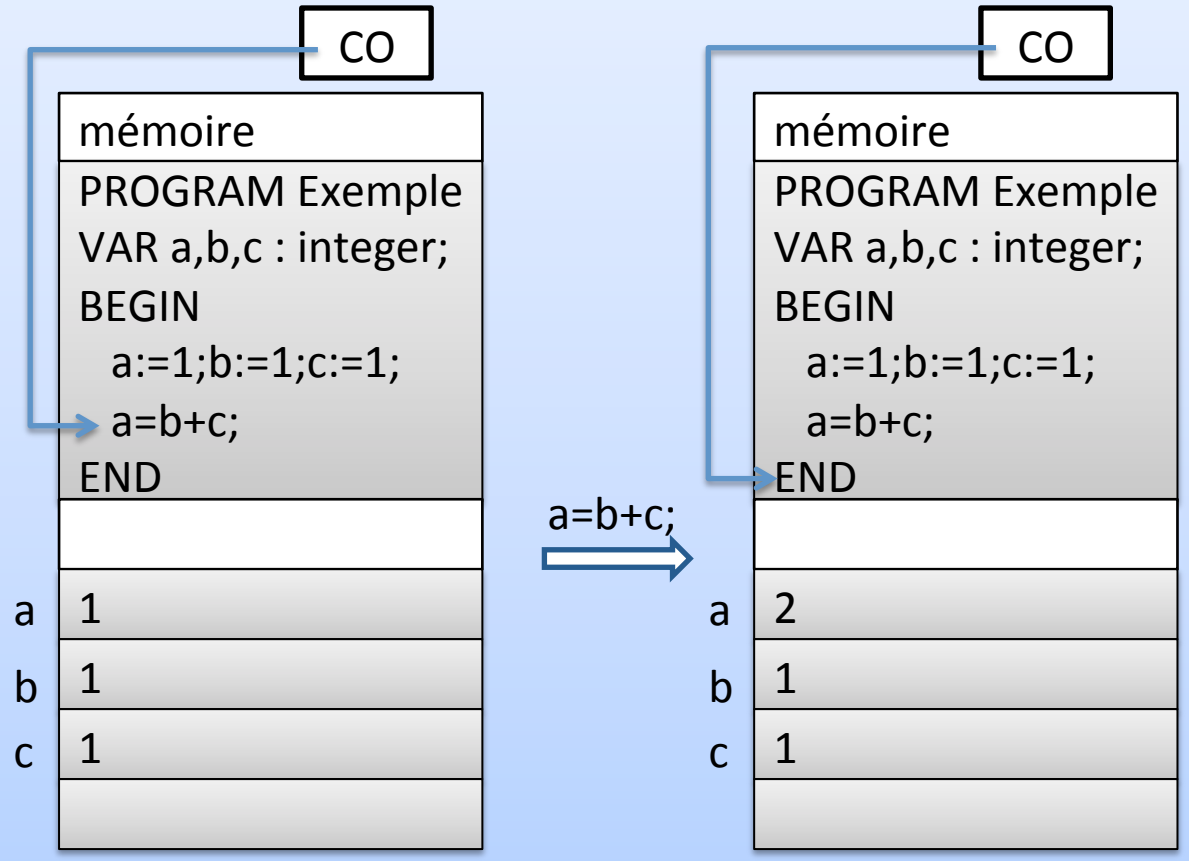
- Un programme est l'expression d'un algorithme à l'aide d'un langage de programmation

- Une tâche est la jonction d'un programme et des données auxquelles il s'applique

- Un processus est l'exécution d'une tâche. Il est caractérisé par la tâche et son contexte d'exécution (son compteur ordinal, ses données, sa place en mémoire,…)

Définition d'un processus

```
PROGRAM Exemple
VAR a,b,c : integer;
BEGIN
  a:=1;b:=1;c:=1;
  a=b+c;
END
```



Définition d'un processus

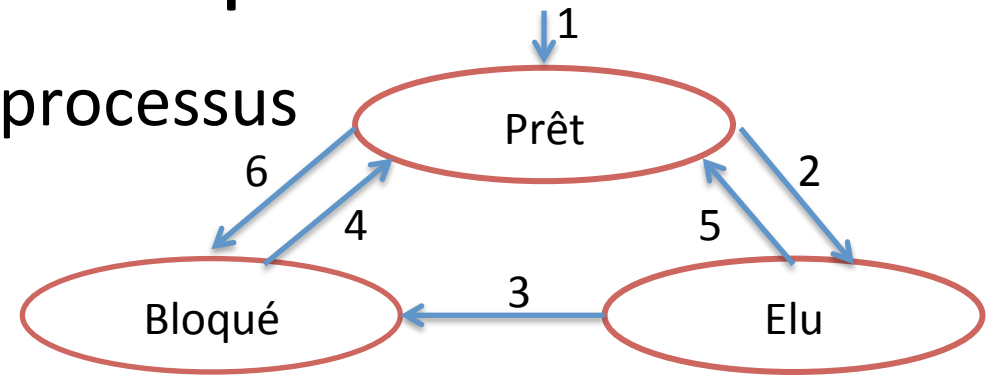
- Création et hiérarchie entre processus sous Unix
  - Un processus peut en créer un autre (processus père)
  - Le processus **init** est le premier lancé. Il crée un certain nombre de processus système (daemon) sans utilisateur qui se chargent des tâches : surveillance d'une connexion réseau, attente d'une connexion au clavier, sauvegardes périodiques,…
- Le S.E. permet aux processus de communiquer entre eux
- Le S.E. tient à jour une table des processus contenant leur emplacements mémoires, les fichiers ouverts, l'utilisateur, leur *états* …

Définition d'un processus

- Etats standards d'un processus
  - Un processus peut être dans les états suivants : élu, prêt, bloqué
  - L'unique processus élu est le processus qui s'exécute
  - Les processus prêts sont susceptibles d'être exécutés s'ils sont choisis par le système
  - Les processus bloqués ne peuvent être choisis
    - ils leurs manquent une ressources matérielle *(mémoire)* ou système *(accès à un fichier), soit*
    - ils sont en attente d'un évènement matériel *(fin E/S)* ou système *(message ou signal)*

Définition d'un processus

- Etats standards d'un processus



1. Le processus a les ressources nécessaires à son exécution
2. Le processus est élu : l'ordonnanceur sélectionne le processus pour lui donner le processeur
3. Attente d'un évènement (ressource, fin d'un autre processus, demande d'une E/S) : au cours de son exécution le processus se bloque
4. L'évènement attendu arrive : l'origine du blocage du processus a disparu
5. Le temps alloué au processus est épuisé (fin de quantum), le processus perd le processeur qui lui est retiré par l'ordonnanceur
6. Le système retire une ressource nécessaire à l'exécution d'un processus

Ordonnancement

- L'ordonnanceur (Scheduler en anglais) est le composant du noyau du S.E. qui choisit les processus qui vont être exécutés par le(s) processeur(s) d'un ordinateur

- Le rôle de l'ordonnanceur se décompose en deux parties
  - Le mécanisme de commutation entre les processus
  - Le choix du prochain processus à être exécuter

- Objectifs de l'ordonnanceur
  - Efficacité de la commutation: maximiser le taux d'utilisation du CPU et minimiser le temps système
  - Satisfaction des utilisateurs : minimiser le temps de réponse

Ordonnancement

- Dans un système en batch, il n'y a commutation que si le processus actif doit effectué une E/S

- Dans un système en temps partagé, un processus peut perdre le CPU s'il fait une E/S ou s'il épuise son quantum

- Les interruptions sont utilisées pour commuter les processus
  - Une interruption périodique est déclenchée par une horloge (quantum) et l'ordonnanceur est alors mis en action. Il peut commuter les processus en modifiant le processus de retour de l'interruption

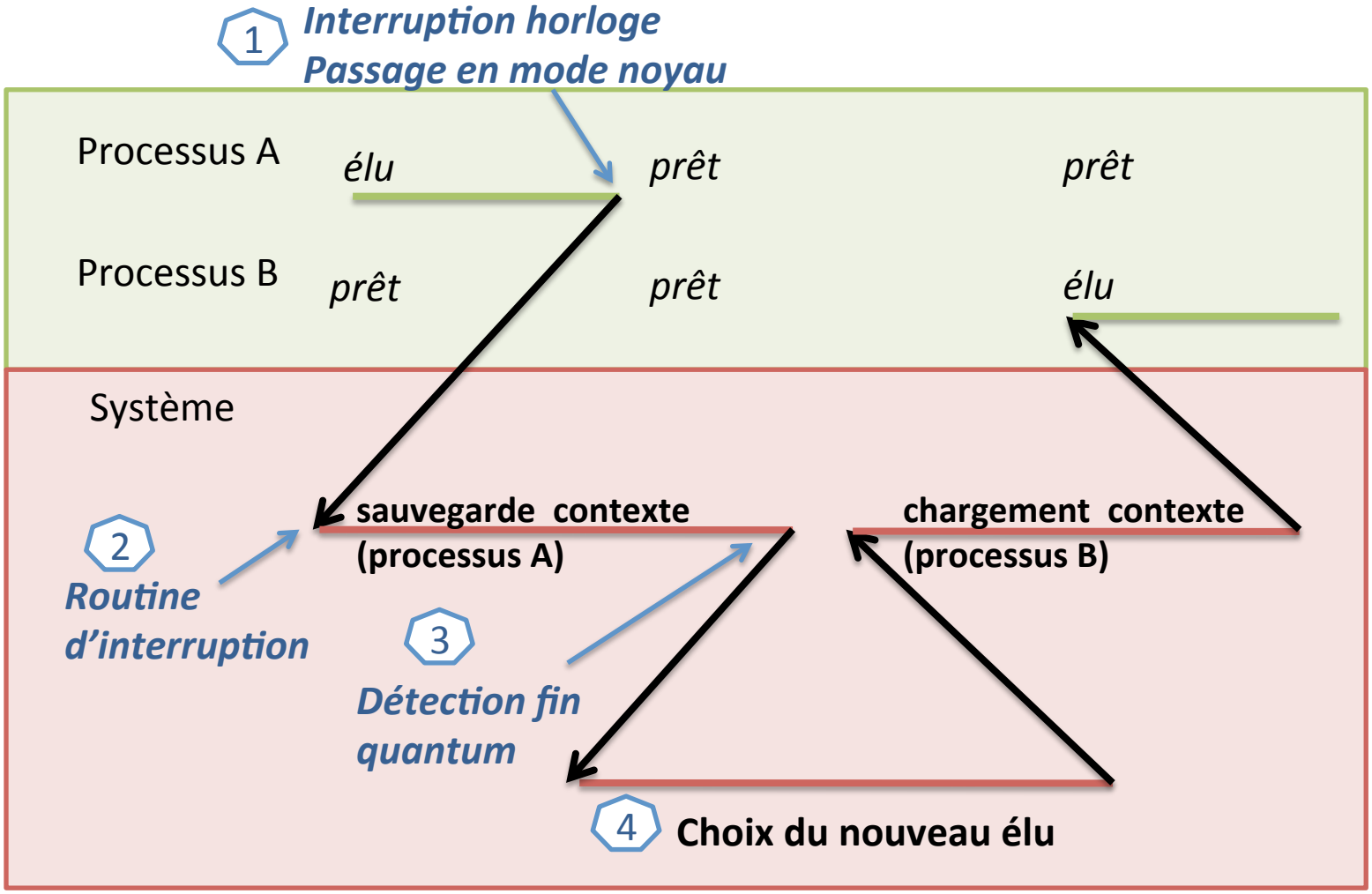
- Une stratégie doit être adoptée par l'ordonnanceur pour le choix du prochain processus

Ordonnancement

- Mise en œuvre de la commutation :
  - Une interruption est un arrêt temporaire de l'exécution d'un processus par le processeur afin d'exécuter un autre processus, appelé routine d'interruption
  - Lors d'une interruption, le processeur sauve tout ou une partie de son état interne, et exécute ensuite la routine d'interruption
  - La routine se finit par une (ou plusieurs) instruction de retour d'interruption, qui restaure l'état sauvé et soit fait repartir le processeur de l'endroit où il avait été interrompu soit modifie les adresses de retour, pour effectuer des commutations de processus

Ordonnancement

- Mise en œuvre de la commutation :



Ordonnancement

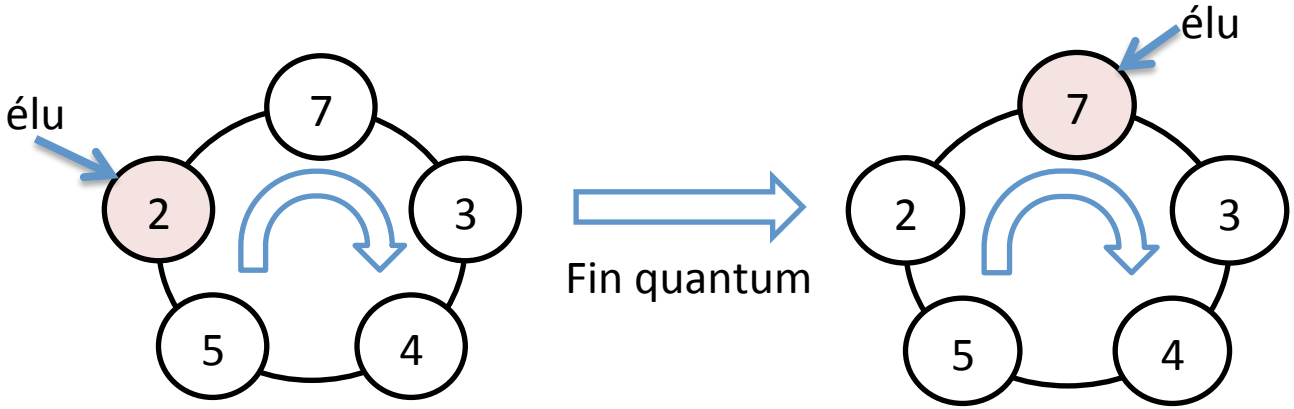
- Les stratégies typiques :
  - First In First Out (FIFO) *(premier arrivé, premier servi)*
    - Exécution suivant les ordres d'arrivée
    - Inconvénients: temps de réponse dépendant du processus qui s'exécute (tant qu'il ne se bloque pas, les autres doivent attendre), pénalise les processus courts (proportion temps d'attente/temps d'exécution) et favorise les processus longs
  - Shortest Job Next (SJN) *(plus court processus en premier)*
    - Le choix se fait en fonction du temps d'exécution estimé du processus
    - L'ordonnanceur va laisser passer d'abord le plus court des processus de la file d'attente

Ordonnancement

- Les stratégies typiques :
  - Shortest Remaining Time (SRT) *(plus court temps restant en premier)*
    - Le processus dont le temps restant est le plus petit est choisi
    - Quand un processus arrive dans la file d'attente, sa durée d'exécution est comparer à la durée d'exécution restante du processus en cours. Si la durée du processus entrant est plus courte que la durée d'exécution restante du processus en cours, alors ce dernier est suspendu et le processus entrant prend sa place
    - SJN avec préemption (priorité 0/1)

Ordonnancement

- Les stratégies typiques :
  - Tourniquet (*Round Robin* en anglais)
    - FIFO avec préemption
    - Les processus prêt sont mis dans une file d'attente circulaire
    - Des tranches de temps (quantum) sont attribués à chaque processus en proportion égale, sans accorder de priorité aux processus
    - Une fois le quantum épuisé, le processus passe la main et retourne à la fin dans la file d'attente



Ordonnancement

- Les stratégies typiques :
  - Tourniquet
    - Choix du quantum : quantum court implique trop de commutation, quantum long implique temps de réponse (du processus) long
    - Cette stratégie suppose une équité entre les processus
    - ... plus de raffinement… les *priorités*

Ordonnancement

- Les stratégies typiques :
  - Principe des méthodes avec priorité
    - A chaque processus est assignée une priorité (fonction du temps)
    - L'ordonnanceur choisit le processus prêt de plus haute priorité
  - Priorités statiques
    - Une file d'attente par niveau de priorité *(ex. temps de calcul estimé)*
    - A leur création, les processus sont affectés à un niveau de priorité et sont insérés dans la file correspondante
    - Risque de famine : pour élire une tâche d'une file, il faut que toutes les files de niveau supérieur soient vides

Ordonnancement

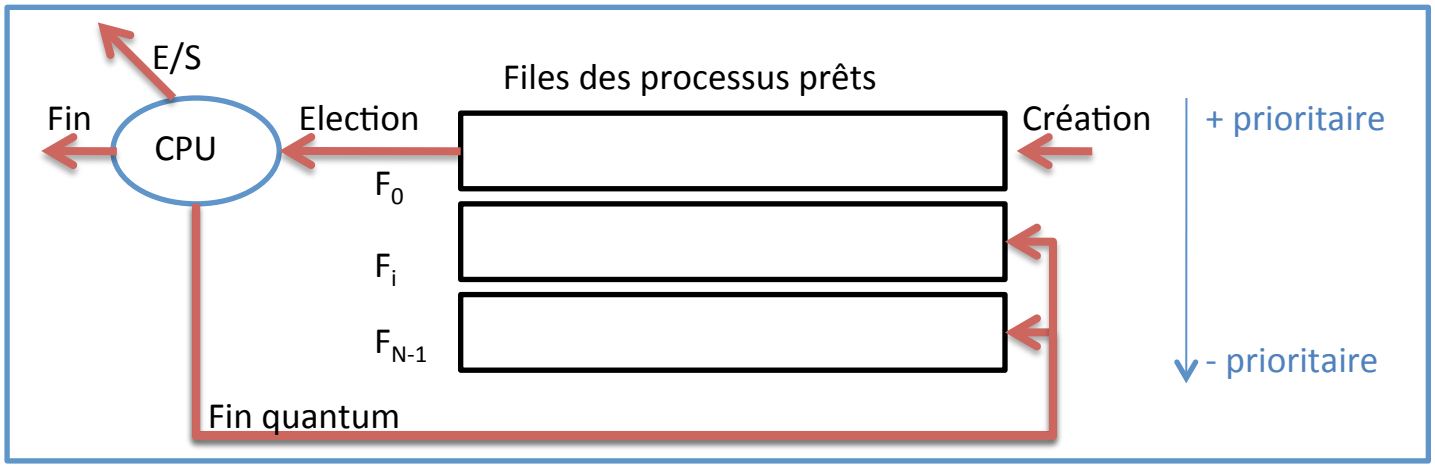
- Les stratégies typiques :
  - Priorités dynamiques
    - Priorité calculée avant et pendant l'exécution (ex. -1 à chaque élection)
    - La modification des priorités des processus permet de prendre en compte, au moins de manière approchée, le comportement des processus (ex. pénalisation des processus qui utilisent de nombreuses ressources)
    - Diminution du temps alloué aux processus qui ont été élus
    - Favoritisme envers les processus courts

Ordonnancement

- Les stratégies typiques :
  - Files d'attente multiples
    - Principe : N files (F_0 à F_{N-1}) correspondant à N niveaux de priorité (0 étant le niveau le plus prioritaire)
    - Les files sont rangées par ordre de priorité
    - A la création, les processus sont rangés dans des files en fonction de leur temps d'exécution
    - Les processus changent de file au cours de temps :
      - Le premier processus prêt de la première file non vide et la plus prioritaire s'exécute
      - Les processus dans la file F_0 s'exécutent durant 1 quantum, ceux de la file F_1 s'exécutent durant 2 quantum…
      - Fin des i quantums d'un processus issue de F_i, alors réinsertion du processus dans la file F_{i+1}

Ordonnancement

- Les stratégies typiques :
  - Files d'attente multiples
    - Des nombreux paramètres doivent être définis :
      - le nombre de niveaux, la ou les stratégies de sélection utilisées pour sélectionner un processus à l'intérieur de chaque file, les conditions qui vont provoquer un passage d'un processus dans un niveau inférieur ou supérieur, le mécanisme pour déterminer dans quel niveau se situe un nouveau processus,…

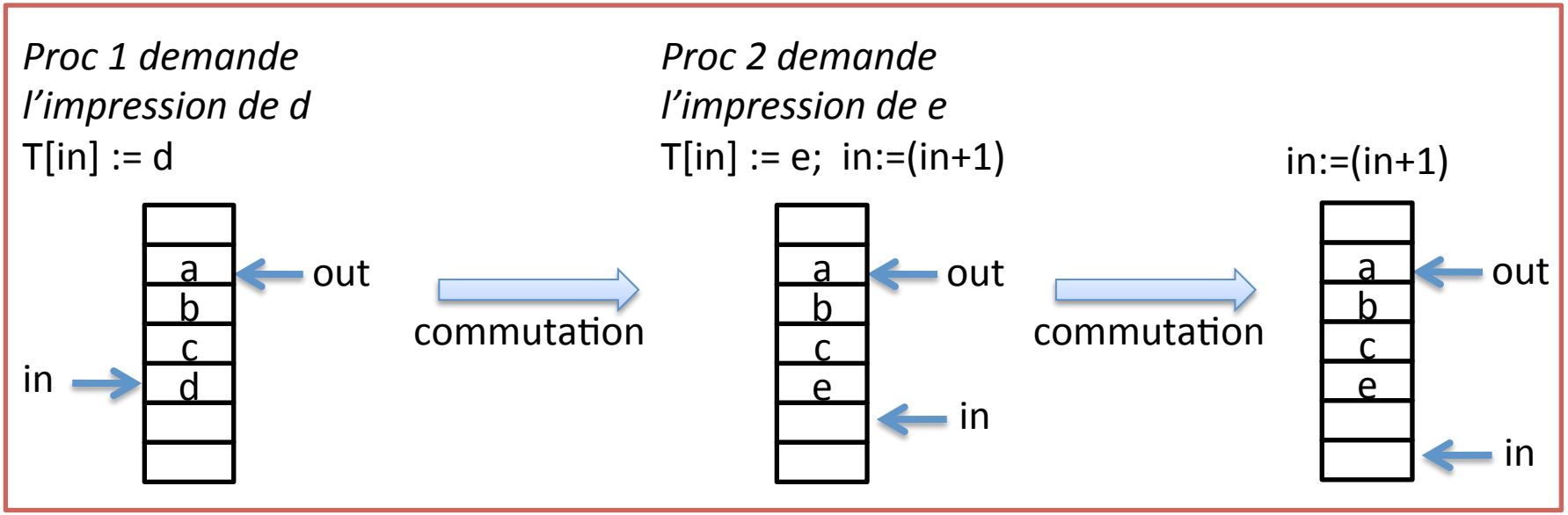


Ordonnancement

- Les stratégies typiques :
  - Files d'attente multiples
    - Avantage : plus une tâche s'exécute, moins elle est prioritaire, cela permet de favoriser les tâches courtes => Approximation de SJN
    - Risque de famine : Il est possible qu'une tâche reste bloquée dans une file intermédiaire (F_1 à F_{N-1})
      - Solution : régulièrement (tous les X quanta), le système remonte d'un niveau toutes les tâches. Après un certain temps les tâches se retrouveront donc à nouveau dans F_0
  - Ordonnancement dans systèmes actuels
    - Trois ordonnancements sont normalisés et cohabitent dans les systèmes actuels : FIFO *(processus temps réel)*, Tourniquet *(processus système)* et Files d'attente multiples *(processus utilisateur)*

Concurrence

- Accès concurrents des processus
  - Soit une pile T des demandes d'impression gérer par un processus système
  - Un processus utilisateur qui souhaite imprimer l'indique dans la cellule *in* puis l'incrémente



Concurrence

- Accès concurrents des processus
  - Une *Section Critique* (S.C.) est une suite d'instructions qui doit être exécutée de manière indivisible

```
<début_section_critique>
  T[in] := d;
  in:=in+1;
<fin_section_critique>
```

  - Pb : Comment réaliser des sections critiques?
  - Solutions : les interruptions, les variables partagées, les sémaphores,…

Concurrence

- Accès concurrents des processus
  - Les interruptions :
    - Les commutations ont lieu sur les interruptions : masquer, puis démasquer les interruptions assure qu'un processus ne sera pas interrompu pendant son accès à la section critique
    - Inconvénient : une très longue section critique monopolise le processeur
  - Les variables partagées
    - Deux processus p et q désirent entrer en section critique

Algo. de p	**Algo. de q**
dem[p] :=vrai;	**dem[q] :=vrai;**
Attendre (dem[q] ==faux);	**Attendre (dem[p] ==faux);**
<section_critique>	**<section_critique>**
dem[p] :=faux;	**dem[q] :=faux;**

    - => Blocage !!

Concurrence

- Accès concurrents des processus
  - Les variables partagées
    - Introduction d'un tour (initialement au processus p)

Algo. de p	**Algo. de q**
Attendre (tour ==p);	**Attendre (tour==q);**
<section_critique>	**<section_critique>**
tour:=q;	**tour :=p;**

    - => dépendance des processus : si le processus p désire exécuter 5 sections critiques et q ne désire exécuter que 2 sections critiques alors p sera bloqué indéfiniment !!

Concurrence

- Accès concurrents des processus
  - Les variables partagées
    - Solution : Algorithme de Peterson qui combine les demandes et le tour

Algo. de p

```
dem[p] :=vrai;
tour :=q;
Attendre ((dem[q] ==faux)
                or  (tour==p))
<section_critique>
dem[p] :=faux;
```

Algo. de q

```
dem[q] :=vrai;
tour :=p;
Attendre ((dem[p] ==faux)
                or(tour==q))
<section_critique>
dem[q] :=faux;
```

    - Fonctionnement :
      - Si pas de concurrence, les variables « dem » autorisent l'entrée en S.C.
      - Si concurrence, la variable « tour » permet l'entée en S.C.

Concurrence

- Accès concurrents des processus
  - Les sémaphores [Dijkstra 1965] :
    - Un sémaphore s est une variable composé d'un compteur et d'une file de processus bloqués
    - Un sémaphore est manipulé par trois primitives :

      I(s,nb) : initialisation du compteur à nb (positif ou nul) autorisations

      P(s) - Puis-je (*Proberen*) - contrôle d'autorisation + blocage éventuel du demandeur :

      masquer l'interruption,

      nb=nb-1,

      si nb < 0 alors blocage du processus et insertion dans la file,

      démasquer l'interruption.

Concurrence

- Accès concurrents des processus
  - Les sémaphores [Dijkstra 1965] :
    - Un sémaphore est manipulé par trois primitives :

      V(s) - Vas-y (*Verhogen*) - ajout d'une autorisation + déblocage éventuel d'un demandeur :

      masquer l'interruption,

      nb=nb+1,

      si nb <= 0 alors extraction de la file et déblocage d'un processus,

      démasquer l'interruption.

    - Solution simple et efficace par sémaphore s initialisé à 1

Algo. de p	**Algo. de q**
P(s);	**P(s);**
<section_critique>	**<section_critique>**
V(s);	**V(s);**

Communication

- Communication entre processus
  - Les moyens de communications entre processus dépendent de la nature de la communication :
    - Synchrone/Asynchrone *(attente d'un des interlocuteurs)*
    - Personnelle/impersonnelle *(connaissance de l'interlocuteur)*
    - Volume des informations
  - Plusieurs modes de communications :
    - Les signaux
    - Les tubes
    - Les messages
    - Les mémoires partagées

Communication

- Communication entre processus
  - Les signaux :
    - Mécanisme par lequel un processus est informé d'un évènement par un autre processus
    - Événements externes qui changent le déroulement d'un processus de manière *asynchrone (à n'importe quel instant lors de l'exécution du processus)*
    - Similaire à une interruption logicielle : le processus sauvegarde son contexte, réagit au signal et restaure son contexte
    - Les signaux transportent peu d'information (le type du signal et rien d'autre)

Communication

- Communication entre processus
  - Les signaux :
    - Il existe différents signaux, identifiés par un numéro :

      SIGHUP 1 : Hang-up (fin de connexion) *terminaison du processus leader de session*

      SIGQUIT 3: Interruption forte (ctrl-\) *frappe quit sur terminal de contrôle*

      SIGKILL 9 : Interruption très forte (ne peut être ignorée) *signal de terminaison*

      SIGCHLD 20 : *Un des processus fils est mort ou a été arrêté (peut être ignoré)*

    - La commande kill envoie un signal à un processus :

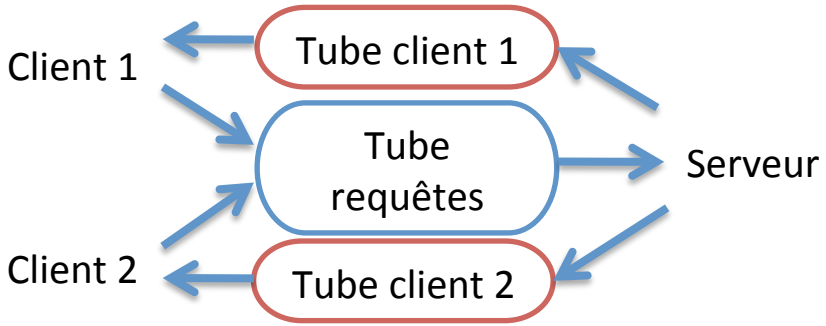
      int kill (pid_t pid, int sig); où pid est l'identité du processus et sig est le nom d'un signal ou un entier entre 1 et NSIG (nb de signaux ≈ 30).

Communication

- Communication entre processus
  - Les tubes
    - Mécanisme de communication *synchrone et unidirectionnel*
    - Un tube n'a pas toujours de nom : tube ordinaire (sans nom) et tube nommé
    - Un tube ordinaire a une durée de vie limitée aux processus créateurs et leurs descendants
    - Un tube nommé est un *fichier* dont la durée de vie va au-delà de la terminaison des processus créateurs et leurs descendants. Il sera désigné par une référence dans le système de gestion des fichiers et sera supprimé lorsque plus aucun processus ne l'utilise.

Communication

- Communication entre processus
  - Les tubes
    - L'existence d'un tube correspond à la possession d'un descripteur acquis soit par un appel à la primitive de création de tube *pipe*; soit par héritage (un processus fils hérite de son père les descripteurs de lecture et d'écriture de tubes)
    - Un tube possède deux extrémités, une pour y lire et l'autre pour écrire
    - La lecture dans un tube est destructrice: l'info lue est supprimée du tube
    - Avantages tubes nommés : peuvent être utilisés par des processus différents et sans lien de parenté *(exemple clients/ serveur)*



Communication

- Communication entre processus
  - Les messages
    - Mécanisme de boîte aux lettres
    - Un mode de communication *personnalisé* : envoie à un processus et réception d'un processus
    - Une information de taille limitée accompagne le message
    - Émission Synchrone (l'émetteur attend que son message soit lu) vs. Asynchrone (le message est rangé dans une file du récepteur)
    - Réception Synchrone (bloquante jusqu'à l'arrivée d'un message) vs. Asynchrone (non bloquante)

Communication

- Communication entre processus
  - Les mémoires partagées
    - Un segment de mémoire accessible à plusieurs processus par un nom prédéfini et une clé d'accès. Les divers processus pourront y lire et/ou y écrire
    - La communication est *impersonnelle*
    - Façon rapide pour échanger des données entre processus.
    - Très efficace pour les algorithmes à variables partagées (ex. l'algorithme de Peterson)
    - Une mémoire partagée peut contenir des variables et des informations structurées (struct, tableau,…)
    - Intérêt : pas de primitives bloquantes

Information

- Cours et TD disponibles à l'adresse

  http://www.lamsade.dauphine.fr/~elhaddad/Courses/SAR/