# Architecture des ordinateurs
# Environnement Windows :
# le système d'exploitation

# Table des matières

# 1. Introduction

## *a) Qu'est-ce qu'un Système d'Exploitation ?*

Le système d'exploitation (noté SE ou OS, abréviation du terme anglais ***Operating System***), est chargé d'assurer la liaison entre les ressources matérielles, l'utilisateur et les applications (traitement de texte, jeu vidéo, ...).

Ainsi lorsqu'un programme désire accéder à une ressource matérielle, il ne lui est pas nécessaire d'envoyer des informations spécifiques au périphérique, il lui suffit d'envoyer les informations au système d'exploitation, qui se charge de les transmettre au périphérique concerné via son **pilote**.

En l'absence de pilotes il faudrait que chaque programme reconnaisse et prenne en compte la communication avec chaque type de périphérique.

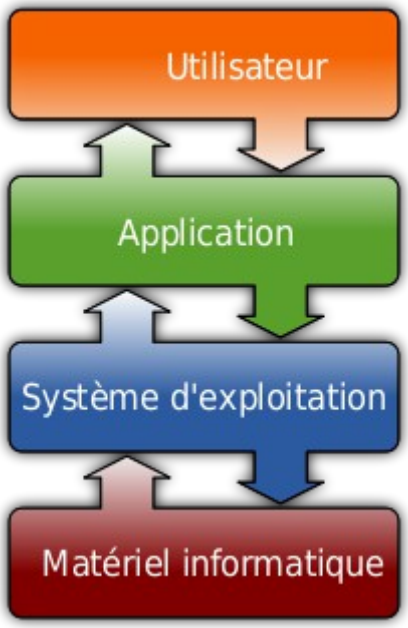



***Illustration 1: Rôle d'un système d'exploitation***

## b) *Les rôles d'un S.E.*

### *Gestion du processeur*

Le système d'exploitation est chargé de gérer l'allocation du processeur entre les différents programmes grâce à un **algorithme d'ordonnancement**. Le type d'ordonnanceur est totalement dépendant du système d'exploitation, en fonction de l'objectif visé.

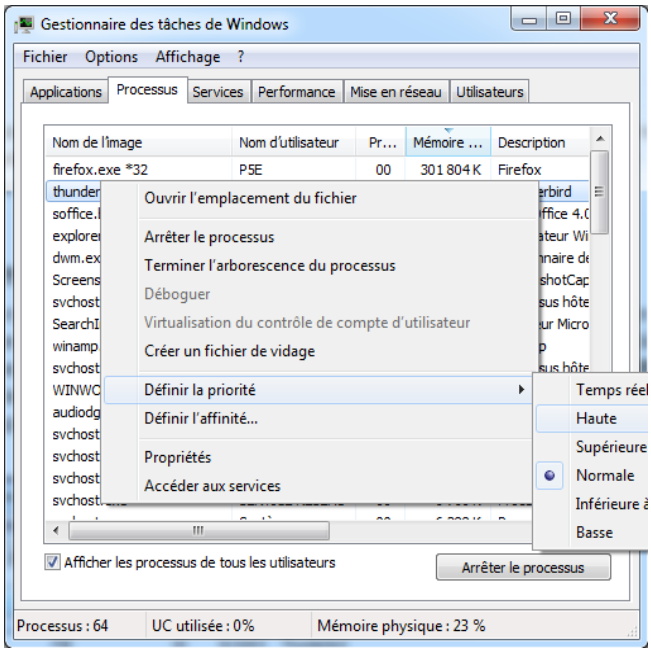



*Illustration 2: Gestion des priorités*

### *Gestion de la mémoire vive*

Le système d'exploitation est chargé de gérer l'espace mémoire alloué à chaque application et, le cas échéant, à chaque usager. En cas d'insuffisance de mémoire physique, le système d'exploitation peut créer une zone mémoire sur le disque dur, appelée «**mémoire virtuelle**». La mémoire virtuelle permet de faire fonctionner des applications nécessitant plus de mémoire qu'il n'y a de mémoire vive disponible sur le système. En contrepartie cette mémoire est beaucoup plus lente.

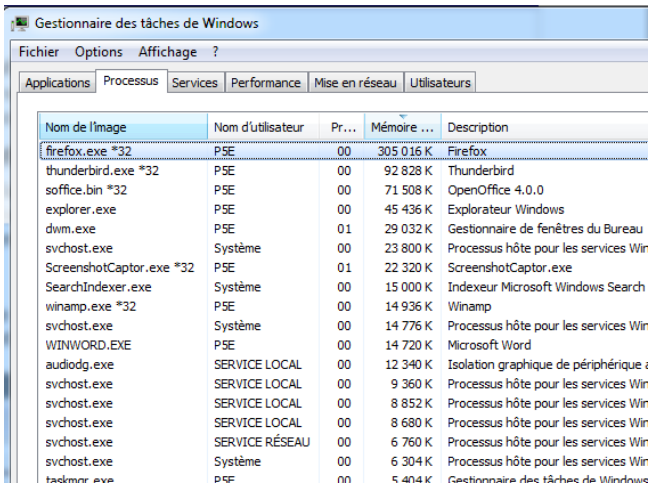



*Illustration 3: Gestion de la mémoire*

## Gestion des entrées/sorties

Le système d'exploitation permet d'unifier et de contrôler l'accès des programmes aux ressources matérielles par l'intermédiaire des pilotes (appelés également gestionnaires de périphériques ou gestionnaires d'entrée/sortie).

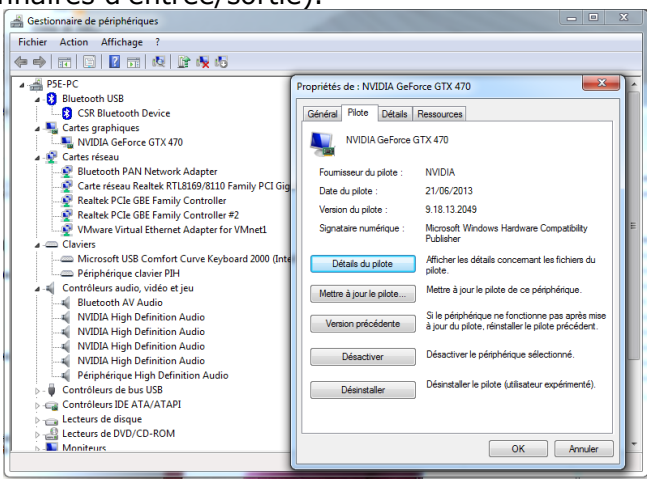



*Illustration 4: Gestion des périphériques et pilotes*

## Gestion de l'exécution des applications

Le système d'exploitation est chargé de la bonne exécution des applications en leur affectant les ressources nécessaires à leur bon fonctionnement. Il permet à ce titre de «tuer» une application ne répondant plus correctement.

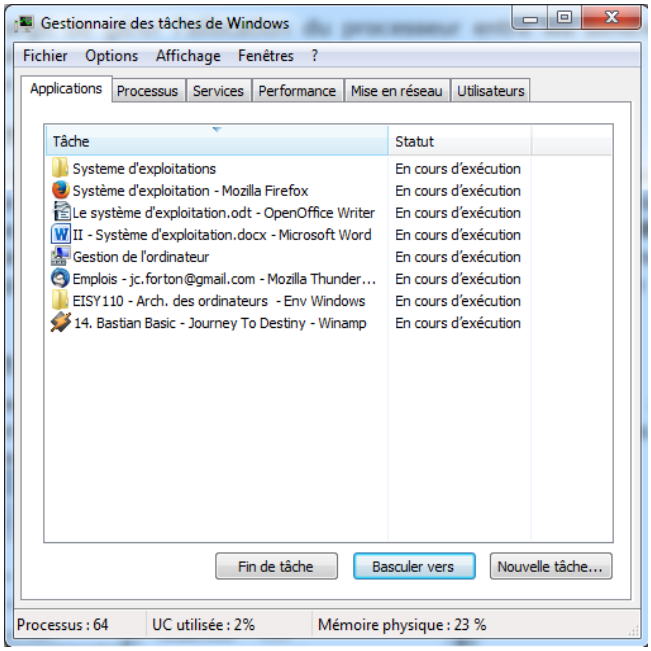



*Illustration 5: Gestion des processus*

## *Gestion des droits*

Le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes en garantissant que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.

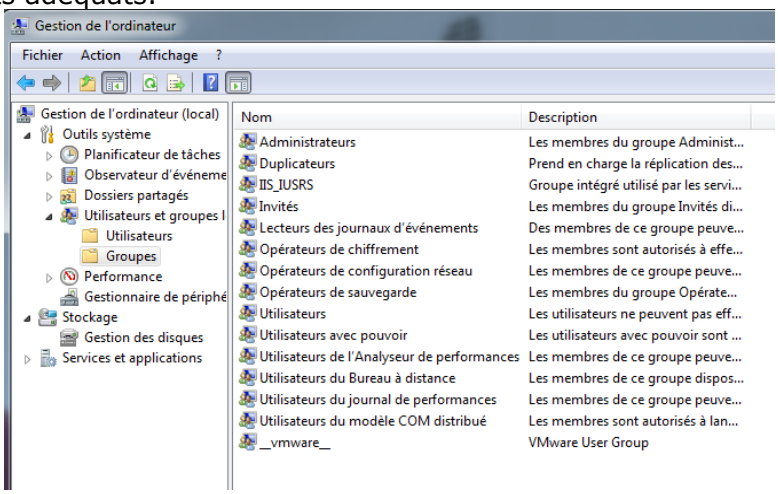



***Illustration 6: Gestion des utilisateurs sous Windows 7***

## *Gestion des fichiers*

Le système d'exploitation gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.

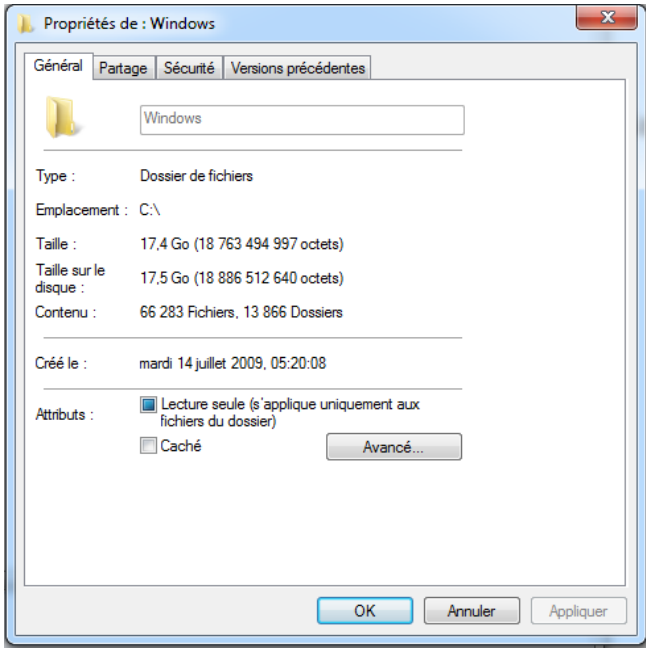



***Illustration 7: Gestion des fichiers***

### ***Gestion des informations***

Le système d'exploitation fournit un certain nombre d'indicateurs permettant de diagnostiquer le bon fonctionnement de la machine.

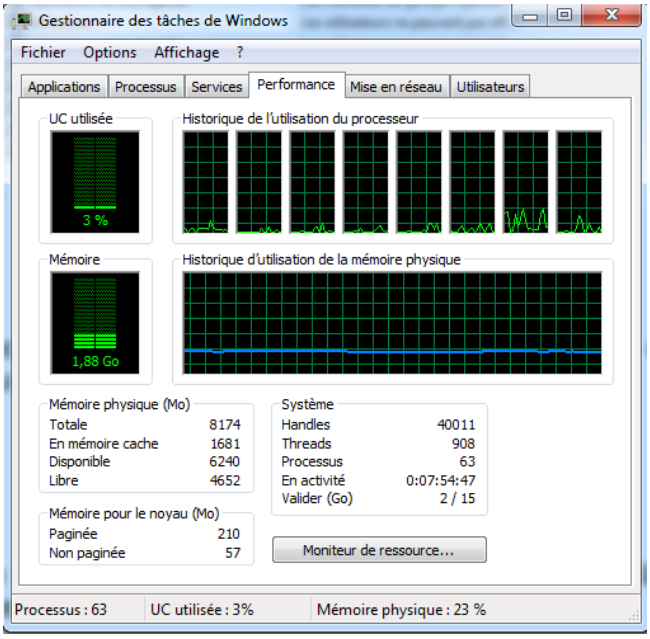



***Illustration 8: Gestion des performance***

# 2. Les composantes d'un S.E.

Le système d'exploitation est composé d'un ensemble de logiciels permettant de gérer les interactions avec le matériel. Parmi cet ensemble de logiciels on distingue généralement les éléments suivants :

- Le **noyau** (en anglais **kernel**) représentant les fonctions fondamentales du système d'exploitation
- L'**interpréteur de commande** (en anglais **shell**, par opposition au noyau) permettant la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes
- Le **système de fichiers** (en anglais «*file system*», noté *FS*), permettant d'enregistrer les fichiers dans une arborescence.

### *a) Le noyau*

Le noyau est composé de plusieurs module qui assure chacun une fonctionnalité.

L'allocateur (***dispatcheur***) du CPU qui est responsable de la répartition du temps disponible de l'unité de traitement entre les différents processus. Il doit aussi sauvegarder l'état de la machine lorsque le processus s'interrompt et indiquer au CPU le processus suivant ; le processus interrompu est sauvegardé dans un bloc d'information appelé vecteur d'état ou descripteur.

La priorité d'un processus est attribuée par le planificateur (***scheduleur***) selon l'urgence et les ressources requises.

Le ***gestionnaire d'interruptions*** a pour tâche de déterminer la source de l'interruption et d'activer la procédure de service correspondante.

Le noyau possède deux modes de fonctionnement :

- Le mode **noyau** ou **superviseur,** qui n'impose pas de restrictions sur les instructions exécutées ;
- Le mode **utilisateur,** qui limite ce que peuvent faire les instructions.

Cette partition entre espace utilisateur et espace noyau est l'élément de base du contrôle d'accès : les applications de l'espace utilisateur ne peuvent accéder à une zone mémoire ne leur appartenant pas. Une telle action déclenche immédiatement une ***trappe*** du noyau, qui doit envoyer un signal particulier au programme pour y mettre fin.

Pour que ce mécanisme fonctionne, il faut que les processeurs disposent d'une unité de gestion mémoire (***MMU***) exploitable par le noyau. La trappe est en effet déclenchée par une interruption matérielle. Le mécanisme de protection mémoire ne peut être implémenté efficacement de façon logicielle.

## b) L'interpréteur de commande

### Sous Windows

Sous Windows, seule l'invite ***DOS*** (***D***isk ***O***perating ***S***ystem) existe, pour des raisons historiques. Elle se lance par l'utilitaire COMMAND.COM ou « cmd.exe ».

Jusqu'à Windows 3.x, Windows n'était qu'une interface graphique du DOS, mais a commencé à proposer plus de fonctionnalités que lui à partir de Windows 95.

La famille de Windows NT, jusqu'à Windows XP qui en est la version 5.1, se passe presque intégralement de la ligne de commande, et l'invite de commandes qu'elle propose n'est qu'un émulateur, largement bridé, de MS-DOS.

Depuis le 24 mars 2009, ***PowerShell*** 1.0 est distribué comme une mise à jour logicielle facultative par le service *Windows Update* de *Microsoft* pour Windows XP et Vista, et son intégration est native sous Windows 7 en version 2.0.

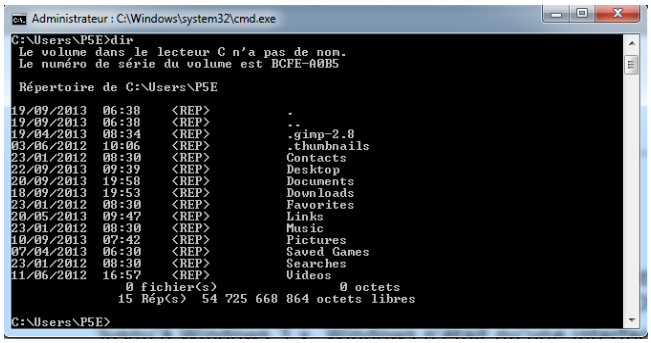



***Illustration 9: Invite PowerShell***

### Sous Unix

Sous UNIX, la ligne de commande a toujours été le moyen privilégié de communication avec l'ordinateur. Le ***Bourne shell*** (***sh***) est l'interpréteur originel de l'environnement UNIX. À son époque, sa grande originalité était l'utilisation de tubes (caractère "|"), qui permettent de connecter la sortie d'une commande à l'entrée d'une autre. On peut ainsi écrire des commandes complexes à partir de commandes simples.

D'autres langages de scripts tels que Perl, Python ou Ruby, remplacent progressivement les interpréteurs qui sont encore prédominants dans les environnements de démarrage de systèmes UNIX.

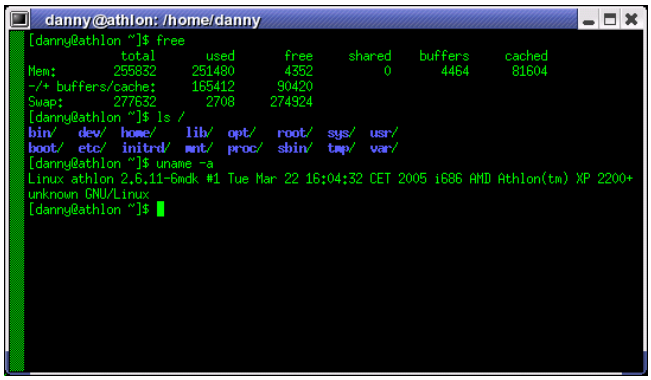



***Illustration 10: Shell Unix***

### c) *Le système de fichiers*

Pour l'utilisateur, un système de fichiers est vu comme une arborescence : les fichiers sont regroupés dans des répertoires (concept utilisé par la plupart des systèmes d'exploitation).

Ces répertoires contiennent soit des fichiers, soit récursivement d'autres répertoires.

Il y a donc un répertoire racine et des sous-répertoires. Une telle organisation génère une hiérarchie de répertoires et de fichiers organisés en arbre.

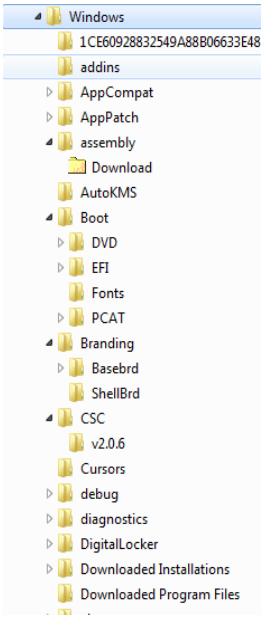



*Illustration 11: Arborescence du système de fichier*

Le nom d'un fichier est une chaîne de caractères sous Windows, de taille limitée et aujourd'hui, quasiment l'ensemble des caractères du répertoire Unicode est généralement utilisable, mais certains caractères spécifiques ayant un sens pour le système d'exploitation peuvent être interdits ou déconseillés comme les caractères « : », « / » ou « \ ».

Le système de gestion des fichiers assure plusieurs fonctions :

- **Manipulation des fichiers** : créer/détruire des fichiers, insérer, supprimer et modifier un article dans un fichier ;
- **Allocation d'espace** : les fichiers étant de taille différente et cette taille pouvant être dynamique, le SGF alloue à chaque fichier un nombre variable de blocs mémoire.
- **Localisation des fichiers** : chaque fichier possède un ensemble d'informations descriptives (nom, adresse…) appelées métadonnées, regroupées dans un ***inode*** et qui permet de le retrouver.
- **Sécurité et contrôle des fichiers** : un nom et une clé de protection sont associés à chaque fichier afin de le protéger contre tout accès non autorisé ou mal intentionné lors du partage des fichiers.

# 3. Les différents systèmes

## a) Selon les services rendus

### Système multitâche

Un système d'exploitation est multitâche (***multitasking***) s'il permet d'exécuter, de façon apparemment simultanée, plusieurs programmes informatiques. On parle également de multiprogrammation. Cette fonction est **indépendante du nombre de processeurs** présents physiquement dans l'ordinateur.

La simultanéité apparente est le résultat de l'alternance rapide d'exécution des processus présents en mémoire. Le passage de l'exécution d'un processus à un autre est appelé commutation de contexte. Ces commutations peuvent être initiées par les programmes eux-mêmes (***multitâche coopératif***) ou par le système d'exploitation lors d'événements externes (***multitâche préemptif***).

Le multitâche préemptif est plus robuste que le multitâche coopératif car une tâche ne peut bloquer l'ensemble du système. Le système d'exploitation peut aussi utiliser plus efficacement les ressources disponibles, par exemple si des données sont disponibles via un périphérique d'entrée, le processus devant traiter ces données peut être immédiatement activé. De la même façon, une tâche en attente de données ne consommera pas de temps processeur avant que ses données ne soient réellement disponibles.

### Système multi-utilisateurs

Un système d'exploitation multi-utilisateurs est conçu pour permettre à plusieurs utilisateurs d'utiliser l'ordinateur simultanément, tout en limitant les droits d'accès de chacun afin de garantir l'intégrité de leurs données.

Le terme opposé est mono-utilisateur qui est utilisé lorsque l'on parle de systèmes d'exploitation utilisable par un seul utilisateur à la fois ou encore en référence à une licence de logiciel prévue pour un utilisateur.

## b) Selon leur architecture

### Système centralisé

Dans ce cas le système ne gère que les ressources de la machine sur laquelle il est présent. C'est le cas de la majeure partie des systèmes installés.

*Illustration 12: Système centralisé*

### *Système réparti*

C'est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources. Le système s'exécute sur un ensemble de machines sans mémoire partagée, mais qui apparaissent à l'utilisateur comme une seule et unique machine.

*Illustration 13: Cluster de machines formant un système d'information réparti*

## c) Selon l'architecture matérielle qui les supporte

### *Systèmes multiprocesseurs*

On appelle SMP (***S***ymetric **M**ulti**p**rocessor) une architecture **parallèle** dans laquelle tous les processeurs accèdent à un espace mémoire partagé.

Les ordinateurs multiprocesseurs permettent un parallélisme de tâches, où un processus peut être exécuté sur chaque processeur. On obtient ainsi une plus grande puissance de calcul qu'avec un ordinateur monoprocesseur, qui peut être utilisée soit pour plusieurs programmes qui disposeraient chacun d'un processeur, soit pour des programmes spécialement conçus, qui sont capables de répartir leurs calculs sur les différents processeurs.

### *Systèmes embarqués*

Les systèmes embarqués sont prévus pour fonctionner sur des machines de petite taille, telles que des PDA ou des appareils électroniques autonomes (sondes spatiales, robot, ordinateur de bord de véhicule, etc.), possédant une autonomie réduite. Ces systèmes électroniques et informatiques, souvent temps réel, sont spécialisés dans une tâche bien précise.

### *Systèmes temps réels*

Les systèmes temps réel (***R***eal ***T***ime ***S***ystem), essentiellement utilisés dans l'industrie, sont des systèmes dont l'objectif est de fonctionner dans un environnement contraint temporellement.

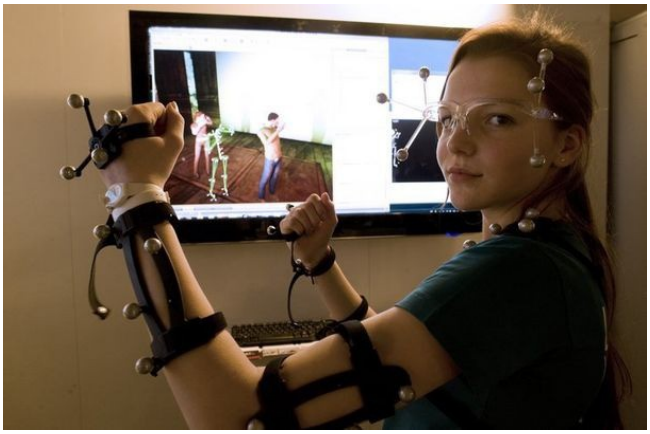


*Illustration 14: Utilisation d'un RTS pour une simulation*

## *d) Gestion des ressources matérielles*

L'une des fonctions primordiales des systèmes d'exploitation est de lancer les programmes et répartir les ressources (processeur, mémoire, périphériques, …) entre les différents programmes qui s'exécutent en même temps, et cela de manière efficace et harmonieuse.

### *Gestion des entrées-sorties*

Cette fonction consiste à prendre en charge le transfert d'information entre l'unité centrale et les périphériques et le réseau. Dans un premier temps, il s'agit de configurer le matériel par le biais de fichiers systèmes pour ensuite gérer les échanges entre ces composants (eg. la mémoire centrale, le processeur, ...).

### *Notion d'interruption*

Une interruption matérielle (IRQ) est un signal qu'un périphérique envoie au système d'exploitation pour dire qu'il a terminé une opération. C'est au système d'exploitation de gérer l'interception des interruptions. Les interruptions matérielles sont utilisées en informatique lorsqu'il est nécessaire de pouvoir réagir en temps réel à un événement asynchrone, ou bien, de manière plus générale, afin d'économiser le temps d'exécution lié à une boucle de consultation (polling loop).

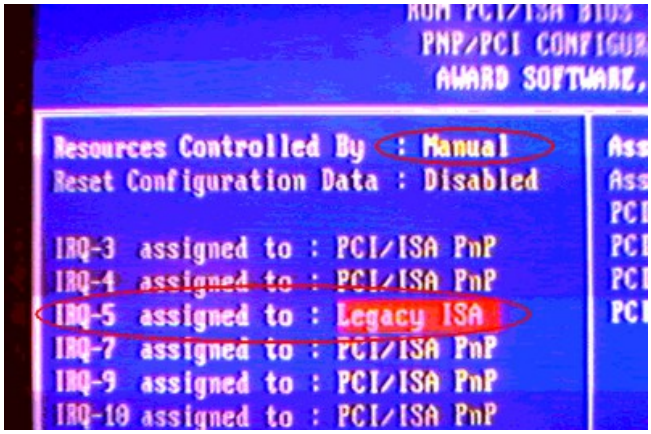



*Illustration 15: Gestion des IRQ dans le BIOS*

### *Notion de processus*

Un processus est un programme qui est en cours d'exécution. L'exécution d'un processus dure un certain temps, avec un début et (parfois) une fin.

Un processus peut être démarré par un utilisateur par l'intermédiaire d'un périphérique ou bien par un autre processus : les « applications » utilisateur sont des (ensembles de) processus.

Le lancement d'un programme suppose qu'il ait été chargé en mémoire centrale (car seule les instructions se trouvant mémoire centrale peuvent être traitées par le processeur).

### *Les systèmes multitâches*

Les systèmes d'exploitation multitâches peuvent exécuter plusieurs programmes en même temps.

Dans ce cas là, tout se complique pour le système d'exploitation:

- il faut d'une part qu'il gère l'allocation de la mémoire ;
- comme le processeur ne peut exécuter qu'une instruction à la fois, les processus doivent se partager le processeur.

Un système est dit préemptif lorsqu'il possède un ordonnanceur (aussi appelé planificateur), qui répartit, selon des critères de priorité, le temps machine entre les différents processus qui en font la demande.

## *e) Gestion des fichiers*

Une autre forme de mémoire que le système doit gérer en dehors de la mémoire centrale, est la mémoire dite permanente ou mémoire de masse. Le système d'exploitation sert d'intermédiaire entre le haut niveau (les applications) et le bas niveau (les pilotes des disques). La partie du système d'exploitation qui se charge de cela se nomme **système de gestion de fichiers** (***SGF***).

Un ***SGF*** est une façon de **stocker** les informations et de les **organiser** dans des fichiers sur ce que l'on appelle des mémoires secondaires (disque dur, SSD, CD-ROM, clé USB, disquette, etc.). Une telle gestion des fichiers permet de traiter, de conserver des quantités importantes de données ainsi que de les partager entre plusieurs programmes informatiques.

Il offre à l'utilisateur une vue **abstraite** sur ses données et permet de les localiser à partir d'un chemin d'accès.

### *Sous Windows (File Allocation Table)*

Différentes méthodes permettent d'associer un nom de fichier à son contenu. Dans le cas du système de fichiers FAT, ancien système de fichiers de MS-DOS et de Windows encore largement utilisé sur les supports amovibles comme les clés USB, chaque répertoire contient une table associant les noms de fichiers à leur taille et un index pointant vers la table d'allocation de fichiers, une zone réservée du disque indiquant pour chaque bloc de données l'index du bloc suivant du même fichier.

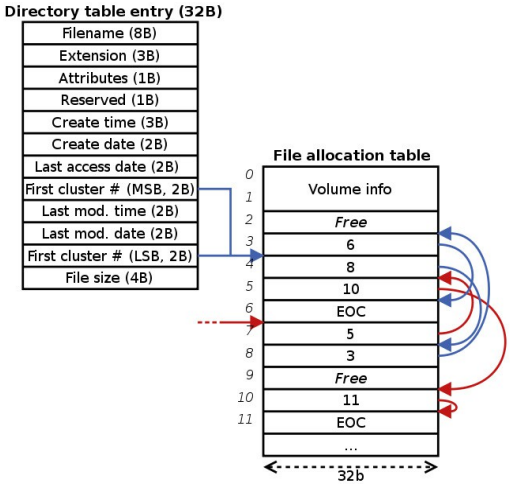



***Illustration 16: Gestion des index en FAT***

### *Sous Linux (Extended-FileSystem)*

Dans le cas des systèmes de fichier Linux, les fichiers et les répertoires sont identifiés par un numéro unique, le numéro **d'inode**.

Ce numéro permet d'accéder à une structure de données (**inode**) regroupant toutes les informations sur un fichier à l'exception du nom, notamment la protection d'accès en lecture, en écriture ou des listes de dates, ainsi que le moyen d'en retrouver le contenu.

Le nom est stocké dans le répertoire associé à un numéro d'inode. Cette organisation présente l'avantage qu'un fichier unique sur disque peut être connu du système sous plusieurs noms.

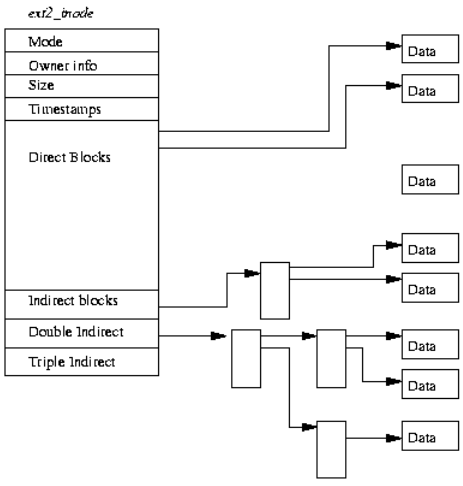



***Illustration 17: Gestion des inodes sous EXT-FS***

### *La fragmentation*

Le disque dur d'un ordinateur est divisé en **secteurs** et le système d'exploitation les rassemble pour former des **clusters** (blocs). Ainsi, un fichier est contenu sur un nombre entier de clusters et le dernier cluster n'est pas toujours entièrement rempli de données.

**Chaque fichier** utilise donc **plusieurs clusters**, l'adresse de ces clusters est conservée par le système d'exploitation dans la ***Master File Table*** (MFT), ce qui lui permet de retrouver l'adresse physique du fichier sur le disque dur.

Si les **clusters** contenant le fichier sont **contigus**, celui-ci n'est **pas fragmenté**. Toutefois, en pratique, les clusters d'un fichier sont généralement éparpillés par groupes, d'où la fragmentation du fichier.

*Illustration 18: Schéma représentant la fragmentation d'un fichier (en rouge)*

#### *Cas d'Unix*

Le noyau calcule le nombre de blocs nécessaires au stockage de chaque fichier sur le disque dur :

- si le nombre de clusters libres contigus est trouvé sur le disque, il stocke le fichier sur ces clusters et le fichier n'est pas fragmenté ;
- s'il ne trouve pas assez de clusters libres contigus, il scinde le fichier en plusieurs groupe de clusters et éparpille ces groupes sur le disque dur en tentant de minimiser le nombre de groupes et par conséquent remplit les plus grands espaces de clusters vides contigus en premier.

#### *Cas de Windows*

Le noyau NT essaie de combler les trous dans le sens où il fragmente le fichier pour ne pas laisser au début du disque des zones avec des clusters libres.

La fragmentation trouve sa source dans les multiples suppressions, modifications, copies de fichiers sur le disque dur, qui favorisent l'apparition de zones de *clusters* libres, et par conséquent la fragmentation.

#### *Performances*

Si la fragmentation est si problématique, c'est parce qu'elle engendre des problèmes au niveau physique.

Lorsqu'un fichier n'est pas fragmenté, la tête de lecture du disque dur n'a pas besoin de se déplacer, ou très peu, puisque les clusters sont les uns à la suite des autres.

À l'inverse, si le fichier est fragmenté, la tête de lecture va faire de multiples aller-retours pour lire chacun des groupes de clusters : se déplacer prend du temps, donc plus le fichier est fragmenté, plus le temps pour accéder à son contenu est élevé.

# 4. La mémoire

## *a) Introduction*

La gestion de la mémoire est un difficile compromis entre les performances (temps d'accès) et la quantité (espace disponible). On désire en effet tout le temps avoir le maximum de mémoire disponible, mais l'on souhaite rarement que cela se fasse au détriment des performances.

Le rôle du gestionnaire de la mémoire est de :

- permettre le partage de la mémoire (pour un système multitâches) ;
- permettre d'allouer des blocs de mémoire aux différentes tâches ;
- protéger les espaces mémoire utilisés (empêcher par exemple à un utilisateur de modifier une tâche exécutée par un autre utilisateur) ;
- optimiser la quantité de mémoire disponible, notamment par des mécanismes d'extension de la mémoire.

## *b) Gestion de la mémoire*

### *Gestion avec segmentation*

La segmentation permet la **séparation des données** et du programme (entre autres segments) dans des espaces **logiquement indépendants** facilitant alors la programmation, l'édition de liens et le **partage inter-processus**. La segmentation permet également d'offrir une plus **grande protection** grâce au niveau de privilège de chaque segment.

Un **segment mémoire** est un espace d'adressage indépendant défini par deux valeurs :

- L'adresse où il commence (*base*) ;
- Sa taille ou son *décalage* (*limite* ou *offset*).

Lorsque la ***MMU*** doit **traduire** une **adresse logique** en **adresse linéaire**, l'unité de segmentation doit dans un premier temps utiliser la première partie de l'adresse, c'est-à-dire le **sélecteur de segment**, pour retrouver les caractéristiques du segment (base, offset, ...) dans la GDT (***Global Descriptor Table***).

Puis il utilise la valeur de décalage (sur 32 bit) qui référence l'adresse à l'intérieur du segment.

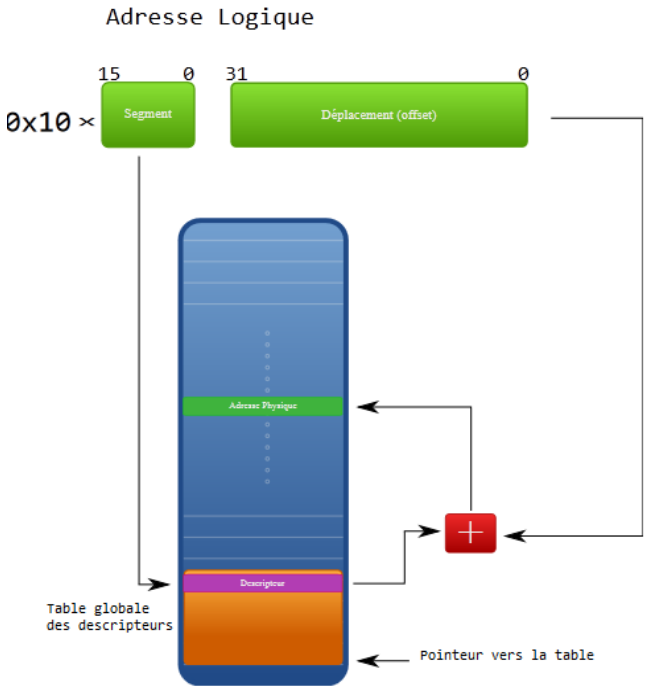



*Illustration 19: Gestion de la mémoire par segmentation*

### *Gestion avec pagination*

Les adresses mémoires émises par le processeur sont des **adresses virtuelles**, indiquant la position d'un mot dans la mémoire virtuelle. Cette mémoire virtuelle est formée de zones de même taille, appelées pages. Une adresse virtuelle est un couple **numéro de page / déplacement dans la page**.

La mémoire vive est également composée de zones de même taille, appelées cadres (***frames***), dans lesquelles prennent place les pages (un cadre contient une page: taille d'un cadre = taille d'une page).

Un mécanisme de traduction (translation) assure la conversion des adresses virtuelles en adresses physiques, en consultant la PTE (***pages table entry***) pour connaître le numéro du cadre qui contient la page recherchée. L'adresse physique obtenue est le couple numéro de cadre / déplacement.

**Il peut y avoir plus de pages que de cadres** (c'est là tout l'intérêt) : les pages qui ne sont pas en mémoire sont stockées sur un autre support (ex. disque), elles seront ramenées dans un cadre quand on en aura besoin.

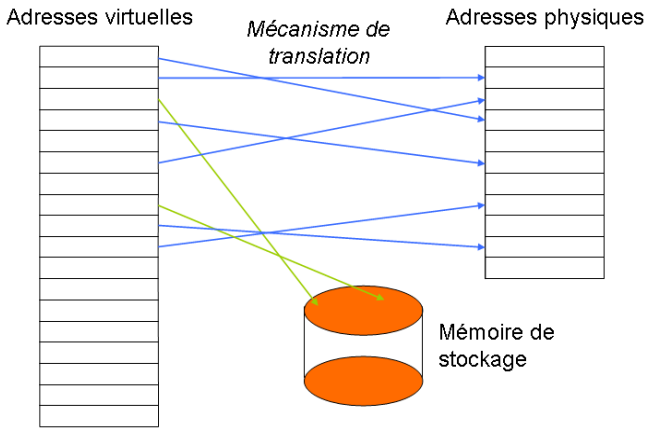



*Illustration 20: Gestion de la mémoire par pagination*

Le système d'exploitation réalise cette opération en créant un fichier temporaire (appelé fichier ***SWAP***) dans lequel sont stockées les informations lorsque la quantité de mémoire vive n'est plus suffisante.

Cette opération se traduit par une **baisse considérable des performances**, étant donné que le temps d'accès du disque dur est extrêmement plus faible que celui de la RAM.

Lors de l'utilisation de la mémoire virtuelle, il est courant de constater que la LED du disque dur reste quasiment constamment allumée et dans le cas du système Microsoft qu'un fichier appelé « ***win386.swp »*** d'une taille conséquente, proportionnelle aux besoins en mémoire vive, fait son apparition.

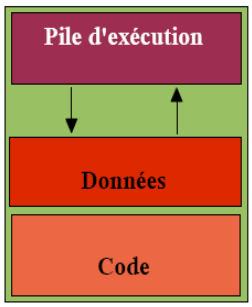
# 5. Les processus

## a) *Introduction*

Un processus modélise l'exécution d'un programme sur un processeur disposant

- de son propre compteur ordinal (@ de la prochaine instruction exécutable) ;
- de ses registres ;
- de ses variables.

Un processus fournit l'image de l'état d'avancement de l'exécution d'un programme. Ce processus est alors appelé processus lourd.




Un processus lourd peut être divisé en plusieurs processus (dit processus légers). On aboutit ainsi à une arborescence de processus.

Les processus sont composés d'un espace de travail (espace d'adressage) en mémoire formé de 3 segments et visible par l'utilisateur/programmeur (cf. ci-joint).

De façon simplifiée, on peut imaginer les processus dans trois états :

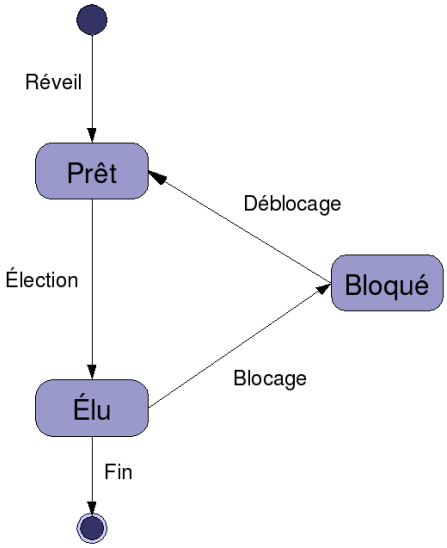



*Illustration 21: Diagramme d'état du cycle de vie d'un processus*

Lorsque le processus est invoqué, il se **réveil** et attends son **élection**. Une fois **élu** pour le prochain cycle d'horloge, le processus passe au processeur les instructions à exécuter. Si l'instruction dure plusieurs cycle, le processus passe alternativement de l'état **bloqué** à l'état **élu**. Lorsque toutes les instructions sont exécutées, le processus se **termine**.

### b) Contexte d'exécution

Un **contexte d'exécution** d'une tâche (processus, processus léger...) est constitué par l'**ensemble des données utilisées par la tâche** en question.

C'est l'ensemble minimal de données à sauvegarder pour permettre une interruption de la tâche à un moment donné, et une reprise de cette exécution au point où elle a été interrompue, et, dans la mesure du possible, indépendamment de la date de la reprise du processus.

Ces données sont situées :

- dans les registres du processeur sur lequel la tâche est exécutée ;
- dans la zone de la mémoire utilisée par la tâche ;
- pour certains systèmes d'exploitation, dans des registres de contrôle stockant les informations nécessaires au système pour gérer ce processus.

### c) Communication inter processus

Les communications inter processus (Inter-Process Communication ou **IPC**) regroupent un **ensemble de mécanismes** permettant à des **processus concurrents** (ou distants) de **communiquer**. Ces mécanismes peuvent être classés en trois catégories :

- les outils permettant aux processus de s'échanger des données ;
- les outils permettant de synchroniser les processus, notamment pour gérer le principe de section critique ;
- les outils offrant directement les caractéristiques des deux premiers (ie : permettant d'échanger des données et de synchroniser des processus).

### d) Les sémaphores

Un sémaphore est une variable (type de donnée abstrait) et constitue la méthode utilisée couramment pour restreindre l'accès à des ressources partagées (par exemple un espace de stockage) dans un environnement de programmation concurrente. Le sémaphore a été inventé par Edsger Dijkstra et utilisé pour la première fois dans le système d'exploitation ***THE Operating system***.

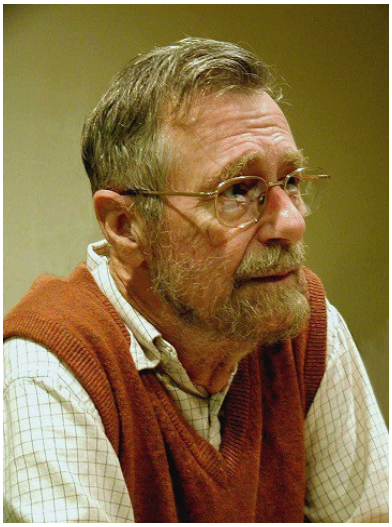


*Illustration 22: Edsger Dijkstra*

Pour pouvoir exister sous forme logicielle, ils nécessitent une implémentation matérielle (au niveau du microprocesseur), permettant de tester et modifier la variable protégée au cours d'un cycle insécable. En effet, dans un contexte de multiprogrammation, on ne peut prendre le risque de voir la variable modifiée par un autre processus juste après que le processus courant vient de la tester et avant qu'il ne la modifie.

### *Structure*

Un sémaphore est une structure à deux champs :

- une variable entière, ou valeur du sémaphore (K), un sémaphore est dit binaire si sa valeur ne peut être que 0 ou 1 ;
- une file d'attente de processus ou de tâches (Liste L).

Dans la plupart des cas, la valeur du sémaphore représente à un moment donné le nombre d'accès possibles à une ressource.

Seules deux fonctions permettent de manipuler un sémaphore :

- **WAIT** (ou **P**rend) :si s = 0 alors le processus qui exécute WAIT est bloqué sinon décrémente s ;
- **SIGNAL** (ou **V**end) : incrémente s et si s > 0 alors le processus qui est en attente (WAIT) est activé ;

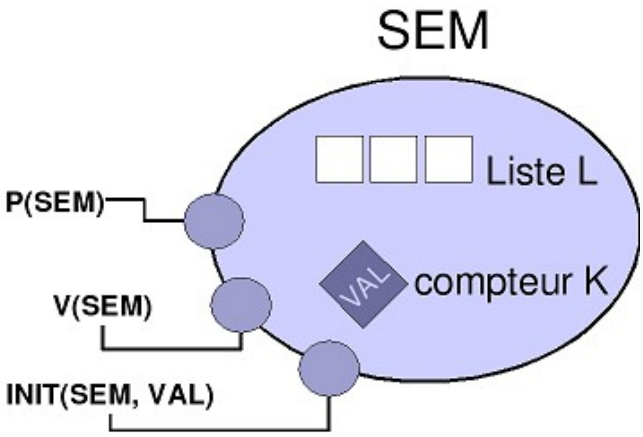



*Illustration 23: Le sémaphore SEM, son compteur K et sa liste L*

Une troisième fonction, **INIT**, est seulement utilisé pour initialiser le sémaphore et ne doit être appelée qu'une seule fois.

### *Principe*

Les fonctions **WAIT** et **SIGNAL** sont **insécables** (atomiques) et s'excluent mutuellement. Si WAIT et SIGNAL sont **appelées en même temps**, elles sont exécutées l'une après l'autre dans un **ordre imprévisible**.

Les processus ont besoin de communiquer, d'échanger des informations de façon plus structurée que par le biais d'interruptions. Un modèle de communication entre processus avec partage de zone commune (tampon) est le modèle **producteur – consommateur**.

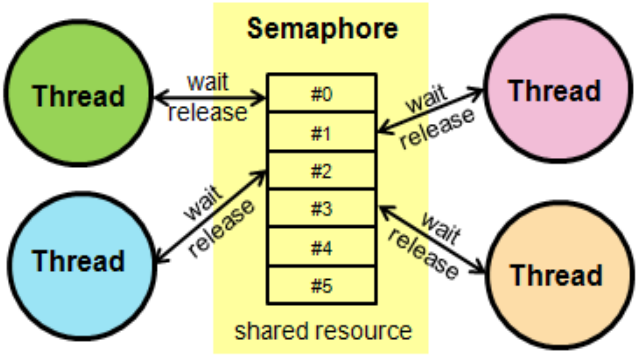



*Illustration 24: Système producteur / consommateur*

Le producteur doit pouvoir ranger en zone commune des données qu'il produit en attendant que le consommateur soit prêt à les consommer.

Le consommateur ne doit pas essayer de consommer des données inexistantes.

### *Producteurs / Consommateurs*

Hypothèses :

- les données sont de taille constante ;
- les vitesses respectives des deux processus (producteur consommateur) sont quelconques ;
- le producteur ne peut pas ranger un objet si le tampon est plein ;
- Le consommateur ne peut pas prendre un objet si le tampon est vide ;
- Exclusion mutuelle au niveau de l'objet : le consommateur ne peut prélever un objet que le producteur est en train de ranger ;
- Si le producteur est en attente parce que le tampon est plein, il doit être averti dès que cette condition cesse d'être vraie.

### *Sans sémaphore*

Réfléchissons sur le comportement d'un producteur et d'un consommateur fonctionnant sans sémaphore.

| **PRODUCTEUR** | **CONSOMMATEUR** |
|---|---|
| Faire toujours<br>produire un objet<br>si nb d'objets dans tampon < N alors<br>déposer l'objet dans le tampon<br>finsi<br>Fait | Faire toujours<br>si nb d'objets dans tampon >0 alors<br>prendre l'objet<br>consommer l'objet<br>finsi<br>Fait |

*Illustration 25: Producteur / consommateur sans sémaphore*

### *Avec sémaphore*

Le tampon peut être représenté par une liste circulaire. On introduit donc deux variables caractérisant l'état du tampon :

- NPLEIN, nombre d'objets dans le tampon (début : 0) ;
- NVIDE : nombre d'emplacements disponibles dans le tampon (N au début).

On peut considérer NVIDE et NPLEIN comme des sémaphores :

| **PRODUCTEUR** | **CONSOMMATEUR** |
|---|---|
| Faire toujours<br>produire un objet<br>**Wait** (NVIDE)<br>déposer un objet<br>**Signal** (NPLEIN)<br>Fait | Faire toujours<br>**Wait** (NPLEIN)<br>prélever un objet<br>**Signal** (NVIDE)<br>consommer l'objet<br>Fait |

*Illustration 26: Producteur / consommateur avec sémaphore*

### *e) Ordonnancement des processus*

L'ordonnanceur définit l'ordre dans lequel les processus prêts utilisent l'UC et la durée d'utilisation, en utilisant un algorithme d'ordonnancement.

Un bon algorithme d'ordonnancement doit posséder les qualités suivantes :

- **équité** : chaque processus reçoit sa part du temps processeur ;
- **efficacité** : le processeur doit travailler à 100 % du temps ;
- **temps de réponse** : à minimiser en mode interactif ;
- **temps d'exécution** : minimiser l'attente des travaux en traitement par lots (batch) ;
- **rendement** : maximiser le nombre de travaux effectués par unité de temps.

#### *La problématique du dîner de philosophe*

Le problème du « dîner des philosophes » est un cas d'école classique sur le partage de ressources en informatique système. **Plusieurs philosophes** (ici 5) se trouvent autour d'une table. Chacun des philosophes a devant lui un plat de spaghetti et à gauche de chaque plat de spaghetti se trouve **une fourchette**.

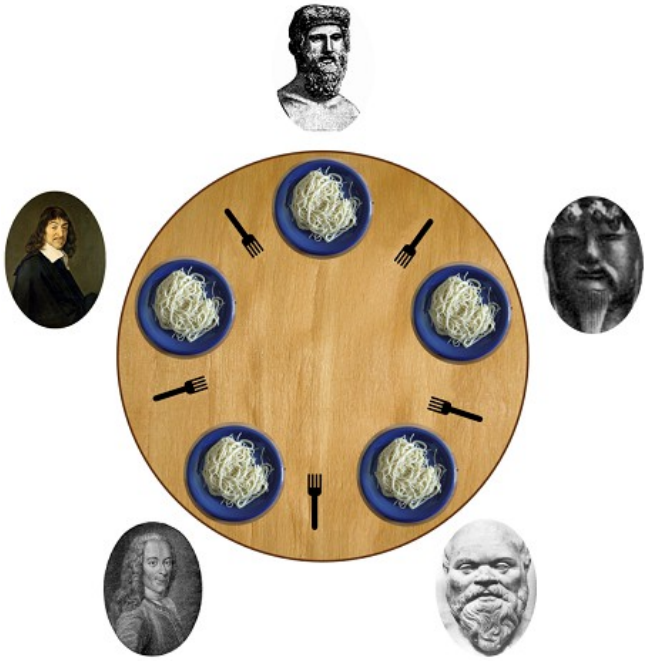


*Illustration 27: Le dîner de philosophe (Platon, Confucius, Socrate, Voltaire et Descartes)*

Un philosophe n'a que trois états possibles :

- penser pendant un temps indéterminé ;
- être affamé (pendant un temps déterminé et fini sinon il y a famine) ;
- manger pendant un temps déterminé et fini.

Des contraintes extérieures s'imposent à cette situation :

- **quand un philosophe a faim**, il va se mettre dans l'état « **affamé** » et attendre que les **fourchettes** soient **libres** ;
- pour **manger**, un philosophe a besoin de **deux fourchettes** : celle qui se trouve à gauche de sa propre assiette, et celle qui se trouve à droite ;
- si un philosophe n'arrive pas à s'emparer d'une fourchette, il reste affamé pendant un temps déterminé, en attendant de renouveler sa tentative.

Le problème consiste à trouver un ordonnancement des philosophes tel qu'ils puissent tous manger, chacun à leur tour.

### *Dead-lock*

Les philosophes, s'ils agissent tous de façons naïves et identiques, risquent fort de se retrouver en situation d'interblocage. En effet, il suffit que chacun saisisse sa fourchette de gauche et, qu'ensuite, chacun attende que sa fourchette de droite se libère pour qu'aucun d'entre eux ne puisse manger, et ce pour l'éternité.

### *Ordonnancement en temps partagé*

L'ordonnancement à temps partagé est présent sur la plupart des ordinateurs comme, par exemple, l'ordonnancement « **decay** » qui est celui par défaut sous ***Unix***.

*Il* consiste en un système de **priorités** adaptatives qui privilégie les tâches interactives pour que leur temps de réponse soit bon.

### *Ordonnancement circulaire ou tourniquet (round robin)*

Le round-robin est un jeu de parcs : un tourniquet. Chaque processus est sur le tourniquet et ne fait que passer devant le processeur, a son tour et pendant un temps fini.

Plus formellement :

- un **nouveau processus** est ajouté en **fin de liste** pour ne pas doubler des processus déjà existants, ce qui pourrait créer une possibilité de famine ;
- l'**utilisation** par un processus **du processeur** ne peut **pas dépasser** un certain **quantum de temps** ce qui nous assure de nouveau qu'il n'y aura pas de famine ;
- l'attente maximum est donnée par la multiplication du nombre de processus en cours multiplié par le quantum de temps accordé à chaque processus ;
- un processus qui vient de finir d'utiliser le processeur (quantum écoulé) est placé en fin de liste ;
- un processus qui a terminé son travail est sorti de la liste, par conséquent le temps d'attente pour les autres processus diminue.

Quand le processeur choisit un nouveau processus à traiter et le charge, cela s 'appelle la **commutation de contexte**.

# Annexes

## 6. Index des illustrations

### Index des illustrations