Les systèmes d'exploitation

Philippe Arnould

Philippe.Arnould@univ-pau.fr

Les systèmes d'exploitation

Plan :

- *les processus*
- *la gestion de la mémoire*
- *les fichiers*
- *les Entrées/Sorties*

Introduction

Sans logiciels, un ordinateur est un morceau de métal inutile.
Grâce à ses logiciels, il peut mémoriser, exécuter des applications les plus diverses.

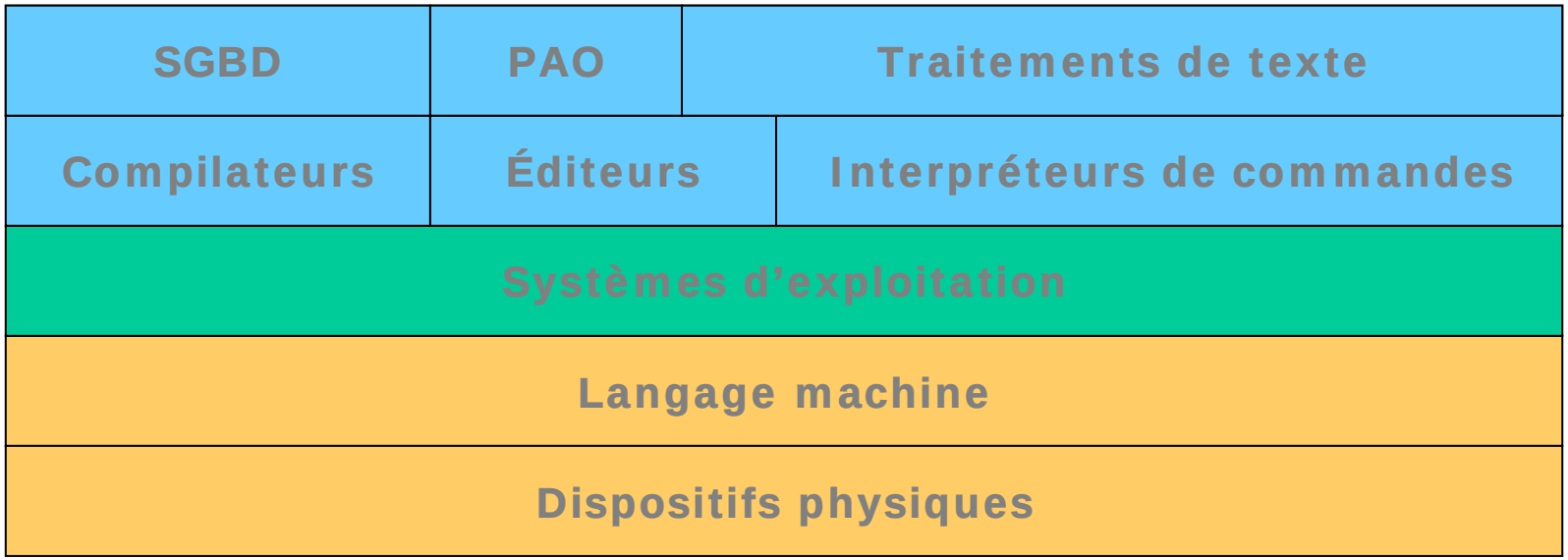


Figure n°1 : structure d'un ordinateur

Les systèmes d'exploitation

Exemples de fonctionnalités :

- machine virtuelle : machine plus facile à programmer (Application Program Interface).

- gestion simultanée de plusieurs programmes.

Les systèmes d'exploitation

Historique :

La « machine analytique » de Charles Babbage (1792-1871)

1. *La première génération (1945-1955) : les tubes à vides et cartes perforées.*

2. *La deuxième génération (1955-1965) : les transistors et le traitement par lots*

3. *La troisième génération (1965-1980) : les circuits intégrés et la multiprogrammation.*

4. *La quatrième génération (1980-1999) : les ordinateurs personnels*

Les systèmes d'exploitation

1. La première génération (1945-1955) : les tubes à vides et cartes perforées.

En 1940 : technologie des tubes électroniques.

Un seul groupe de personnes concevait, construisait, programmait, utilisait et effectuait la maintenance d'une machine.

La programmation en langage machine (relier électriquement des cartes).
Les systèmes d'exploitation étaient inconnus.

En 1950, on introduit les cartes perforées.

2. La deuxième génération (1955-1965) : les transistors et le traitement par lots

Transistor : fiabilité -> commercialisation

Traitement par lots :

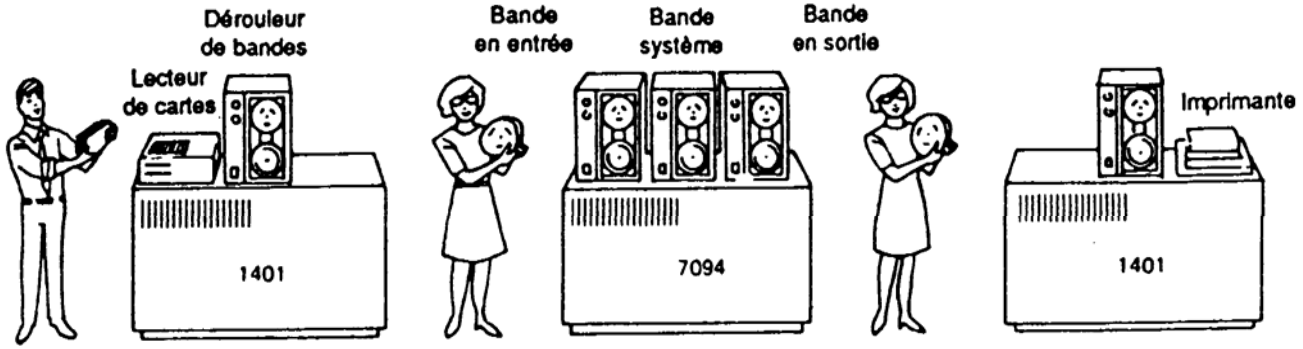


Figure n°2 : Traitement par lots

Les systèmes d'exploitation

3. La troisième génération (1965-1980) : les circuits intégrés et la multiprogrammation.

Avec la série 360, IBM : première génération d'ordinateurs conçus avec des circuits intégrés : amélioration du rapport coût/performance. Utilisation pour des applications scientifiques et commerciales, le système d'exploitation était énorme et très complexe (OS/360).
Introduction de la multiprogrammation et du SPOOL (Simultaneous Peripheral Operation On Line).

MIT : introduction du temps partagé CTSS (Compatible Time Sharing System)

Développement de MULTICS

4. La quatrième génération (1980-1999) : les ordinateurs personnels

Développement des circuits LSI (Large Scale Integration).
L'architecture est similaire au mini-ordinateur, la grosse différence se situe au niveau du prix.
Le micro-ordinateur permet à chacun d'avoir un ordinateur.
L'amélioration des performances et de l'interface permettent à des utilisateurs peu compétents d'utiliser un ordinateur (Macintosh, Windows,...).

Développement des réseaux : exploitation des ressources calculs des ordinateurs connectés sur le réseau

Les processus

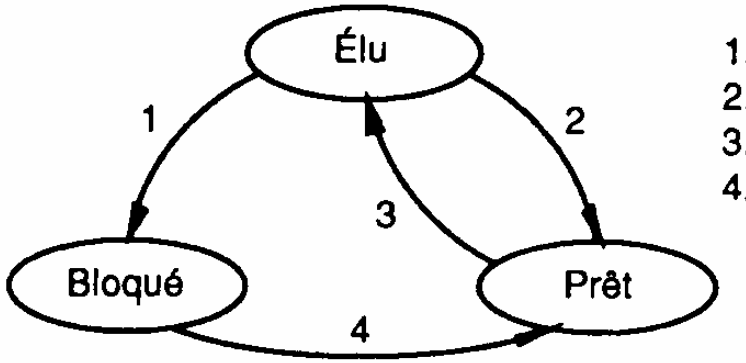
1. Introduction

Un processus est fondamentalement un programme qui s'exécute :

il consiste en un programme exécutable, ses données et sa pile d'exécution, son compteur ordinal, son pointeur de pile et autres registres, ainsi que toutes les autres informations nécessaires à l'exécution du programme.

Les ordinateurs modernes peuvent exécuter plusieurs programmes simultanément, c'est le multitâche. Conceptuellement, chaque processus a son propre processeur virtuel, en réalité, le vrai processeur commute entre plusieurs processus.

2. États d'un processus



1. Le processus se bloque en attente de données.
2. L'ordonnanceur choisit un autre processus.
3. L'ordonnanceur choisit le processus initial.
4. Les données deviennent disponibles.

Figure n°3 : état d'un processus

2. La communication inter-processus

a. Nécessité d'une méthode de communication

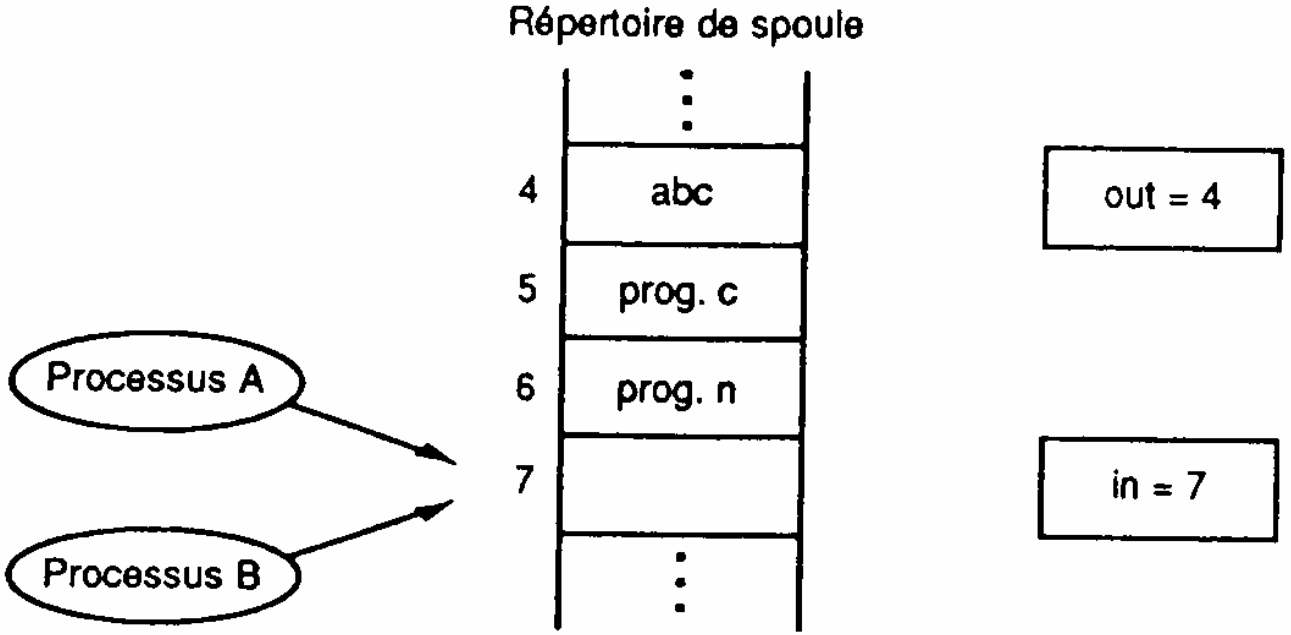


Figure n°4: Deux processus veulent accéder à une mémoire partagée au même moment

Les processus

Processus A

-lit et mémorise IN (7)

- place au 7iéme emplacement son nom de fichier à imprimer
- incrémente IN (8)

Processus B

- lit et mémorise IN (7)
- place au 7iéme emplacement son nom de fichier à imprimer
- incrémente IN (8)

Le fichier du processus B ne sera jamais imprimé

Les processus

Les situations, où deux processus ou plus lisent ou écrivent des données partagées et où le résultat dépend de l'ordonnancement des processus, sont qualifiées **d'accès concurrents**.

Pour éviter les problèmes de partage de la mémoire, de fichiers: interdire la lecture et l'écriture des données partagées à plus d'un processus à la fois.

Donc il nous faut une **exclusion mutuelle** qui empêche les autres processus d'accéder à un objet partagé si cet objet est en train d'être utilisé par un processus. Dans l'exemple précédent le processus B a utilisé une variable partagée avant que A ait fini de l'utiliser.

La partie du programme où se produisent les conflits d'accès s'appelle une **zone critique**. Le problème des conflits d'accès peut être résolus, si on peut s'assurer que les deux processus ne sont pas en même temps en zone critique.

Les processus

b) Méthodes d'exclusion mutuelle

- Le masquage des interruptions

Masquer les interruptions avant de rentrer dans une zone critique et les restaurer à la fin. L'interruption horloge ne pourra pas avoir lieu et ainsi le processeur ne pourra plus être alloué à un autre processus.

Méthode dangereuse, car si un processus oublie de les restaurer le système ne fonctionne plus

Les processus

- Les variables de verrouillage

Utilisation d'une variable (verrou) partagée, unique, qui a initialement la valeur 0. Un processus doit tester ce verrou avant d'entrer dans la zone critique.
Si le verrou vaut 0, le processus le met à 1 et entre dans la zone critique. Si le verrou est déjà à 1, le processus attend qu'il repasse à 0.

Cette solution ne convient pas, car elle a le même défaut que le « spool » d'impression. Si un processus lit le verrou et qu'il vaut 0. Avant qu'il puisse le mettre à 1 un autre processus est élu et exécuté. Ce processus met le verrou à 1. Lorsque le processus initial est réactivé, il mettra lui aussi le verrou à 1.

Les processus

- Les sémaphores

E.W. Dijkstra (1965) suggéra l'emploi d'une variable entière pour compter le nombre de processus voulant accéder à la zone critique : le sémaphore.
Avec le sémaphore, il proposa deux opérations, DOWN et UP.

DOWN décrémente la valeur d'un sémaphore si cette dernière est supérieure à 0, puis, elle poursuit l'exécution normale. Si le sémaphore vaut 0, le processus est mis en attente. Le test du sémaphore, le changement de sa valeur et la mise en attente éventuelle sont effectués en une seule opération atomique indivisible. Aucun autre processus ne peut accéder à un sémaphore tant qu'une opération sur ce sémaphore n'est pas terminée ou bloquée. Cette atomicité est absolument primordiale pour résoudre les problèmes de synchronisation et éviter les problèmes d'accès concurrents.

Les processus

- Les sémaphores

UP incrémente la valeur du sémaphore concerné. Si un ou plusieurs processus étaient en attente sur ce sémaphore, bloqués par une opération DOWN, l'un d'entre eux sera choisit par le système pour terminer son DOWN.

Le problème du producteur et du consommateur :

Deux processus partagent une mémoire tampon de taille fixe. L'un d'entre eux, le producteur, met des informations dans la mémoire tampon, et l'autre, le consommateur, les retire.

Les problèmes surviennent lorsque le tampon et vide ou plein.

```
#define N 100                       /* nb d'emplacements ds le tampon */

typedef int semaphore;              /* les sémaphores constituent */
                                    /* un type d'entier particulier */
semaphore mutex = 1;                /* contrôle accès section critique */
semaphore vide = N;                 /* nb d'emplacements libres */
semaphore plein = 0;                /* nb d'emplacements occupées */

void producteur(void)
{
   int objet;

   while (TRUE) {                   /* TRUE est la constante 1 */
      produire_objet(&objet);       /* produire l'objet suivant */
      down(&vide);                  /* déc. nb d'emplacements libres */
      down(&mutex);                 /* entrée section critique */
      mettre_objet(objet);          /* mettre l'objet dans le tampon */
      up(&mutex);                   /* sortie section critique */
      up(&plein);                   /* inc. nb d'emplacements occupés */
   }
}

void consommateur(void)
{
   int objet;

   while (TRUE) {                   /* boucle infinie */
      down(&plein);                 /* déc. nb d'emplacements occupés */
      down(&mutex);                 /* entrée section critique */
      retirer_objet(&objet);        /* retirer l'objet du tampon */
      up(&mutex);                   /* sortie section critique */
      up(&vide);                    /* inc. nb d'emplacements occupés */
      utiliser_objet(objet);        /* utiliser l'objet */
   }
}
```

Figure n°5 : Solution du problème du producteur et du consommateur avec les sémaphores

Les processus

- L ’échange de messages

Avec les processus distribués (machines en réseaux), il est impossible d’utiliser les sémaphores, la solution est fournie par **l’échange de messages**. Cette méthode utilise deux primitives **SEND** et **RECEIVE**. La première envoie un message au destinataire spécifié. Le récepteur peut se bloquer si aucun message n’est disponible.

Pour résoudre le problème du producteur-consommateur avec l’échange de message, on supposera que le nombre maximal de messages est N. Sur le figure n°6, on présente une solution du producteur consommateur avec échange de message.

```
#include "prototypes.h"

#define N 100                       /* nb d'emplacements ds le tampon */
#define TAILLE 4                    /* taille message */

typedef int message[TAILLE];

void producteur(void)
{
   int objet;
   message m;                       /* tampon des messages */

   while (TRUE) {
      produire_objet(&objet);       /* produire un objet */
      receive(consommateur, &m);    /* attendre un message vide */
      faire_message(&m, objet);     /* construire message à envoyer */
      send(consommateur, &m);       /* envoyer msg au consommateur */
   }
}

void consommateur(void)
{
   int objet, i;
   message m;

   for (i = 0; i < N; i++)          /* envoyer N messages vides */
      send(producteur, &m);
   while (TRUE) {
      receive(producteur, &m);      /* attendre un message */
      retirer_objet(&m, &objet);    /* retirer l'objet du message */
      send(producteur, &m);         /* renvoyer une réponse vide */
      utiliser_objet(objet);        /* utiliser l'objet */
   }
}
```

Figure n°6 : Solution du problème du producteur et du consommateur avec l'échange de messages

Les processus

3 L'ordonnancement des processus

L'ordonnanceur prend des décisions de telle façon que :

- chaque processus reçoit sa part du temps processeur,
- le processeur est utilisé à 100 %,
- le temps de réponse soit le minimum pour les utilisateurs en mode interactif,
- l'attente des utilisateurs qui travaillent par lots soit la plus petite possible,
- le nombre de travaux effectués en une heure soit maximum.

On voit que le troisième et le quatrième objectif sont contradictoires, car pour réduire le temps de réponse en mode interactif, il ne faut pas exécuter des traitements par lots.

Les processus

Fonctionnement de l ’ordonnancement

Pour s’assurer qu’aucun processus ne s’exécutera pendant trop de temps, les ordinateurs disposent d’une **horloge** (**timer**) qui génère une interruption périodiquement.
A chaque interruption, le système d’exploitation reprend la main et décide si processus courant doit poursuivre son exécution ou s’il a consommé le temps processeur qui lui était imparti.
Dans ce dernier cas, le processus est suspendu et le processeur est alloué à un autre processus.
Cette stratégie qui permet de suspendre les processus prêts est appelée **ordonnancement avec réquisition (preemptive scheduling)** et s’oppose à la méthode d’exécution avec achèvement des premiers systèmes à traitement par lots, **l’ordonnancement sans réquisition.**

Les processus

L'ordonnancement circulaire (tourniquet)

L'un des algorithmes les plus anciens, les plus simples, les plus fiables et les plus utilisés, est le **tourniquet (round robin)**.

Chaque processus possède un intervalle de temps, son **quantum**, pendant lequel il est autorisé à s'exécuter.

Si un processus s'exécute toujours au bout de son quantum, le processeur est réquisitionné et alloué à un autre processus.

Si, en revanche, le processus se bloque ou se termine avant la fin de son quantum, le processeur immédiatement alloué à un autre processus.

Les processus

Le seul problème intéressant du modèle du tourniquet, est **la durée du quantum**. La commutation requiert un certain temps qui est utilisé pour gérer cette communication.

En supposant que cette commutation de processus (appelée aussi **commutation de contexte**) dure 5 ms et que le quantum soit fixé à 20 ms. Le processeur travaille alors pendant 20 ms, puis passe 5 ms sur la commutation de processus. On a donc 20 % du temps processeur qui est gâché pour la commutation (**overhead** important).

Pour améliorer l'efficacité du processeur, on peut fixer la valeur du quantum à 500 ms. La perte de temps est alors de 1%. Mais si 10 utilisateurs appuient en presque en même temps sur la touche retour-chariot, le premier d'entre eux sera lancé immédiatement, le deuxième devra attendre pendant 1/2 s et ainsi de suite. Le dernier attendra 5s si tous les autres utilisent entièrement leur quantum.

Les processus

Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur.

Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif.

Les processus

L'ordonnancement avec priorité

Le modèle du tourniquet suppose que tous les processus sont d'égale importance. La prise en compte de facteurs externes aux calculateurs conduit à un ordonnancement avec priorité. C'est-à-dire chaque processus a une priorité et on lance le processus prêt dont la priorité est la plus élevée.

Pour empêcher les processus de priorité élevée de s'exécuter indéfiniment, l'ordonnanceur diminue la priorité du processus élu à chaque top d'horloge. Si cette priorité devient inférieure à celle du deuxième processus de plus haute priorité, la commutation a lieu.

Les processus

Généralement, les processus les plus prioritaires sont associés aux traitements de défauts du système (panne, reconfiguration, alarme, etc.).

Ensuite sont considérées les processus associés aux événements matériels tels que les exceptions et les interruptions.

Enfin restent les processus dédiés à la gestion des périphériques et les processus utilisateurs.

Les processus

Les files multiples

Dans cette méthode, on cherche à accorder au processus le plus court un privilège encore plus grand que la technique du tourniquet. Pour cela, on introduit des files d'attentes auxquelles sont associées des quantum de temps différents.

Couramment, on utilise trois files d'attente avec comme quantum Q, 2Q et 4Q.

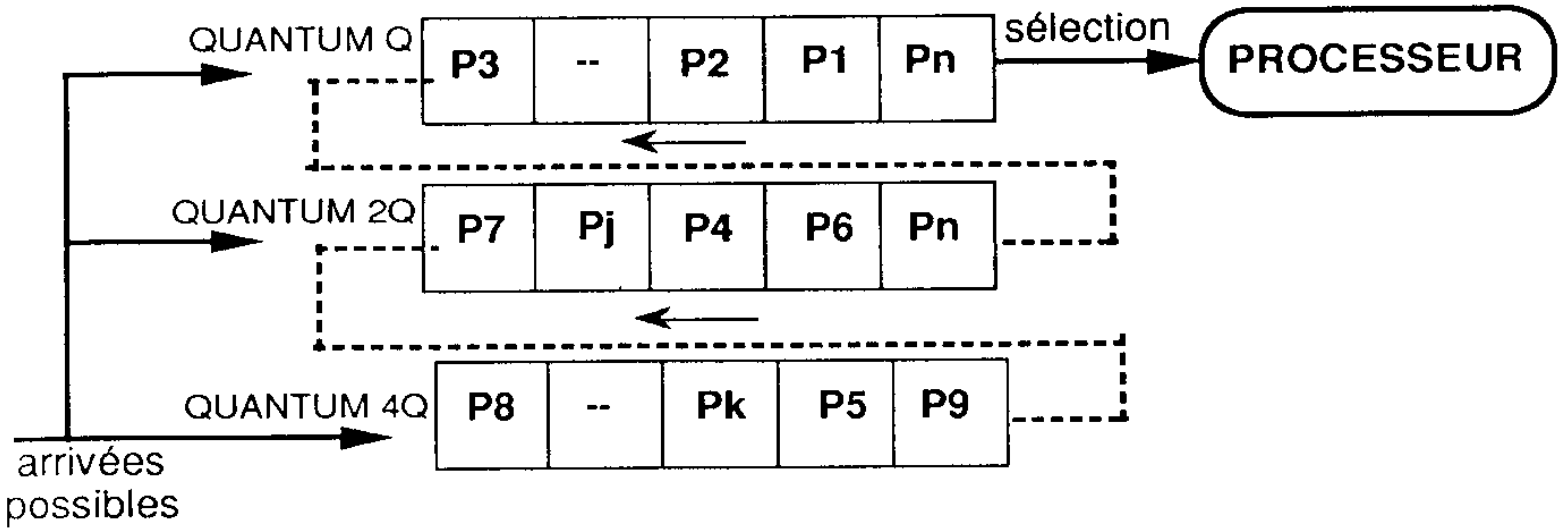


Figure n°7 : l'ordonnancement avec files multiples

Les processus

L'ordonnancement du plus court d'abord

La plupart des ordonnancements présentés précédemment sont conçus pour les systèmes interactifs. Mais dans les systèmes où le temps d'exécution est connu à l'avance (par exemple, dans une compagnie d'assurances, on peut prévoir le temps d'exécution d'un travail portant sur 100 plaintes, car ce travail est exécuté tous les jours) l'ordonnanceur doit choisir le plus court d'abord.

Soient quatre travaux A, B, C et D dont les temps d'exécution respectifs sont 8, 4, 4 et 4 minutes.
En l'exécutant dans cet ordre A se termine au bout de 8 mn, B au bout 12 mn, C de 16 mn et D de 20 mn. La moyenne est donc de 14 mn.
Si on exécute les travaux en commençant par le plus court, les temps sont 4, 8, 12 et 20 mn. La moyenne est de 11 mn. On prouve donc que cet ordre est optimal.

Les processus

En l'exécutant dans cet ordre A se termine au bout de 8 mn, B au bout 12 mn, C de 16 mn et D de 20 mn. La moyenne est donc de 14 mn.

Si on exécute les travaux en commençant par le plus court, les temps sont 4, 8, 12 et 20 mn. La moyenne est de 11 mn. On prouve donc que cet ordre est optimal.

Les processus

4 Réalisation des processus

Pour mettre en œuvre le modèle des processus, le système d'exploitation a une table (tableau de structure).

La table des processus dont chaque entrée correspond à un processus particuliers.

Chaque entrée comporte des informations sur l'état des processus, son compteur ordinal, son pointeur de pile, son allocation mémoire, l'état de ses fichiers ouverts et tout ce qui doit être sauvegardé lorsqu'un processus passe de l'état élu à l'état prêt.

LA GESTION DE LA MEMOIRE

Les systèmes de gestion de mémoires se répartissent en d'eux catégories :

•les systèmes qui déplacent les processus entre la mémoire principale et le disque (va-et-vient et pagination)

•ceux qui ne le font pas. Cette dernière méthode est la plus simple mais ne permet pas d'avoir plusieurs processus en mémoire.

LA GESTION DE LA MEMOIRE

1 Gestion de la mémoire sans va-et-vient ni pagination

La gestion la plus simple consiste à avoir un seul processus en mémoire à un instant donné et à lui permettre d'utiliser toute la mémoire disponible. Cette approche, courante avant 1960, n'est plus utilisée, car chaque processus doit contenir les pilotes des E/S qui l'utilise.

Une des solutions est de placer les pilotes d'E/S en ROM en haut de la mémoire et le système d'exploitation au bas de la RAM. L'IBM PC utilise ce principe avec les pilotes situés dans un bloc de 8 Ko en haut de l'espace d'adressage de 1 Mo (BIOS : Basic Input Output System)

Il ne peut donc avoir qu'un seul processus qui s'exécute à un moment donné. L'utilisateur tape une commande sur son terminal et le système d'exploitation le charge en mémoire puis l'exécute. Lorsque le processus se termine, le système d'exploitation affiche une invite (prompt) et attend la commande suivante.

LA GESTION DE LA MEMOIRE

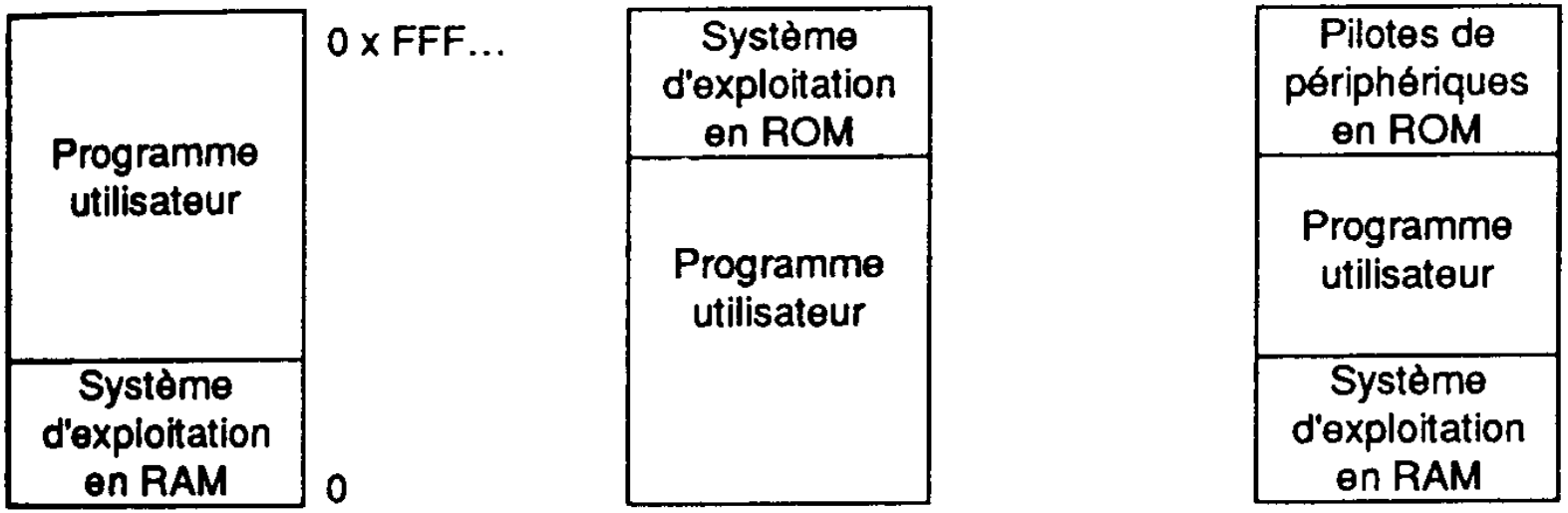


Figure n°8 : Gestion de la mémoire sans va-et-vient ni pagination

2) Le va-et-vient

Avec les systèmes à temps partagé, la mémoire ne pouvant pas contenir tous les processus, il faut placer quelques processus sur le disque. Il faut bien sûr les ramener ces processus en mémoire principale avant de les exécuter. Le mouvement des processus entre la mémoire principale et le disque est appelé **va-et-vient (swapping)**.

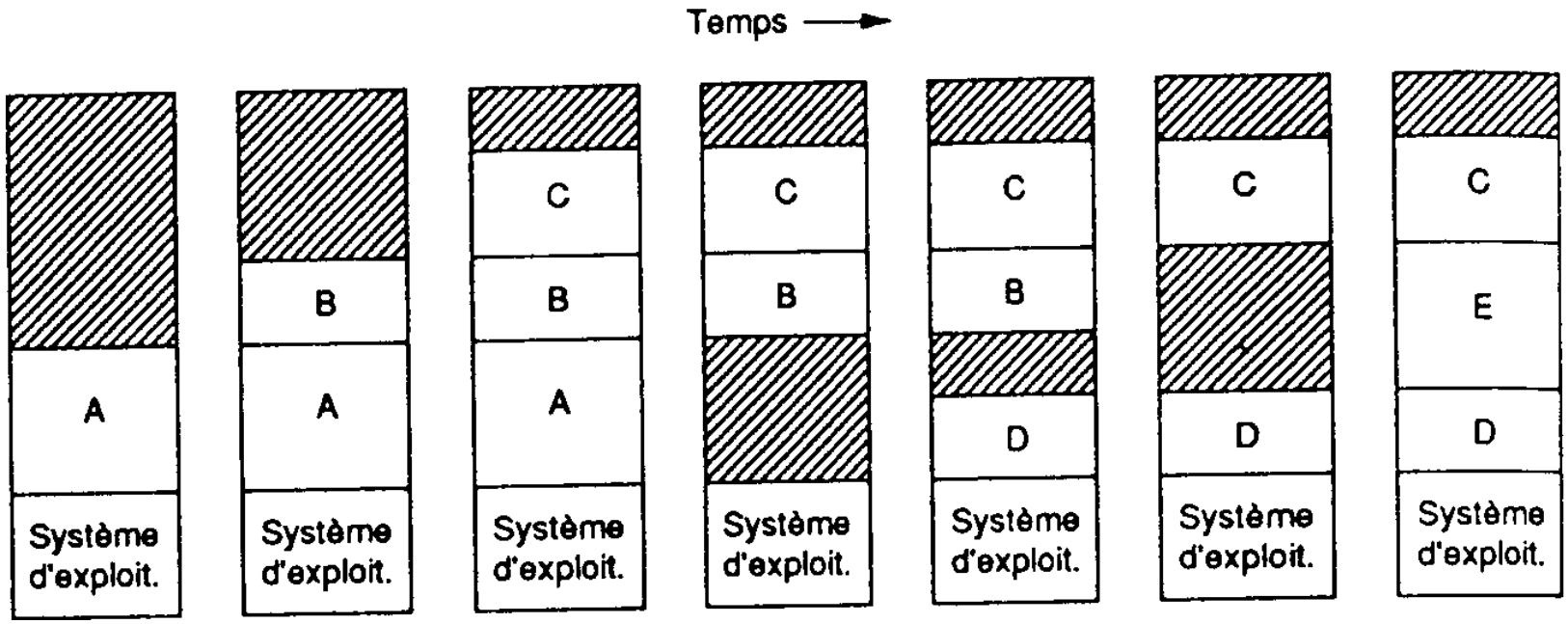
Pour réaliser le va-et-vient, il est nécessaire de diviser la mémoire.

Les partitions peuvent être fixes mais lorsque la mémoire est limitée on perd beaucoup de place avec les programmes plus petits que les partitions.

Pour optimiser la mémoire, il faut donc utiliser des partitions variables.

LA GESTION DE LA MEMOIRE

Avec des partitions variables, le nombre et la taille des processus en mémoire varie au cours du temps :



LA GESTION DE LA MEMOIRE

Pour pouvoir modifier dynamiquement les zones mémoires, il est nécessaire de mémoriser l'occupation de la mémoire. Il existe trois méthodes de mémorisation :

- les tables de bits (bit maps),
- les listes
- les subdivisions (buddy).

LA GESTION DE LA MEMOIRE

A) la gestion de la mémoire par tables de bits.

La mémoire est divisée en unités d'allocation dont la taille peut varier de quelques mots à plusieurs kilo-octets. A chaque unité, on fait correspondre un bit dans la table de bits qui est à 0 si l'unité est libre et à 1 si l'unité est occupée.
La taille de l'unité d'allocation joue un rôle important. Plus elle est faible, plus la table de bits est importante. Et si on prend une unité trop grande, on réduit la taille de la table de bits, mais on perd beaucoup de place mémoire chaque fois qu'un processus n'est pas un multiple de l'unité d'allocation.
Le problème survient lorsqu'on doit ramener en mémoire un processus de k unités. Le gestionnaire de mémoire doit alors parcourir la table de bits à la recherche de k zéros consécutifs. Cette recherche est lente, ce qui fait en pratique, on utilise rarement les tables de bits.

LA GESTION DE LA MEMOIRE

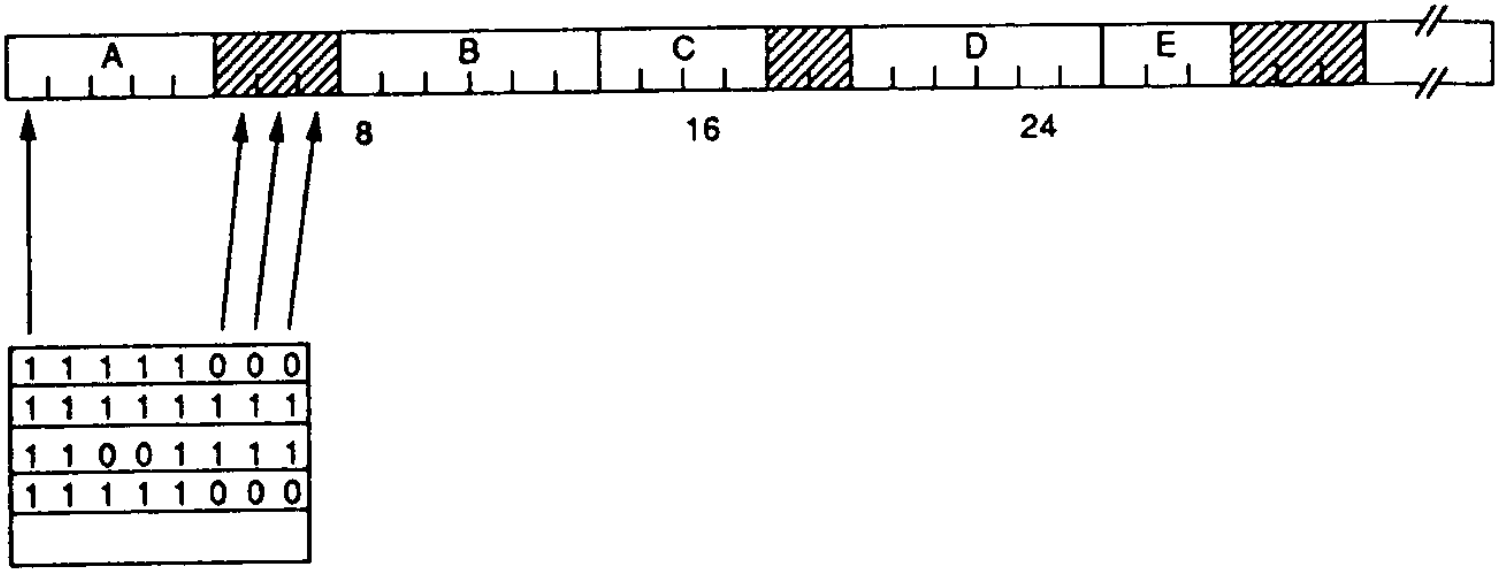


Figure n°9 : Gestion de la mémoire par table de bits

LA GESTION DE LA MEMOIRE

B) la gestion de la mémoire par listes chaînées.

Cette méthode consiste à gérer une liste chaînée des segments libres et occupés., un segment étant un processus ou un espace libre entre deux processus.
Chaque entrée de la liste spécifie une zone libre (H pour hole) ou un processus (P), son adresse de départ, sa longueur et un pointeur sur l'entrée suivante.

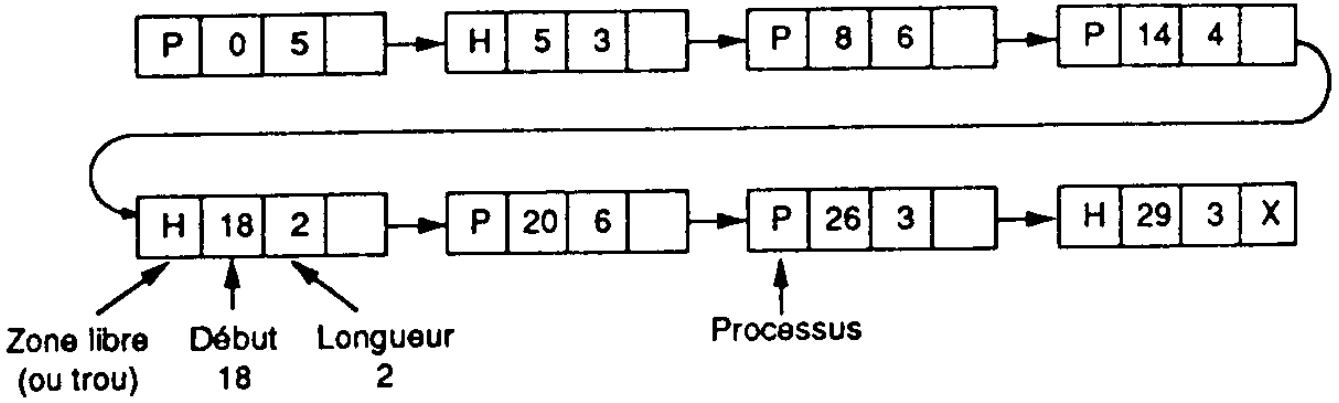


Figure n°10 : Gestion de la mémoire par listes chaînées

LA GESTION DE LA MEMOIRE

Dans cet exemple, la liste des segments est triée sur les adresses. Ce tri permet de mettre à jour facilement lorsqu'un processus se termine ou est déplacé sur le disque.

Un processus a normalement deux voisins (sauf au début et à la fin de la mémoire). Pour remplacer un processus , il suffit de mettre à jour la liste en remplaçant le P par H.

Comme la table des processus contient des pointeurs sur des positions des différents processus de cette liste chaînée, il est plus commode d'avoir un double chaînage plutôt qu'une simple liste chaînée. Ceci permet de voir si le précèdent est une zone libre, dans le but de réaliser une fusion.

LA GESTION DE LA MEMOIRE

Pour permettre d'allouer de la mémoire en utilisant cette méthode de liste chaînée, il existe plusieurs algorithmes.

Le premier et le plus simple est celui de la première zone libre (first fit). Le gestionnaire de la mémoire parcourt la liste des segments à la recherche de la première zone libre qui peut contenir le processus. Cette zone est scindée en deux parties : la première contient le processus et la deuxième, l'espace mémoire inutilisé.

LA GESTION DE LA MEMOIRE

C) La gestion de la mémoire par subdivisions

L'allocation par subdivision est algorithme de gestion qui s'appuie sur le fait que les ordinateurs utilisent des adresses binaires de manière à accélérer la fusion lors de la libération de la mémoire (processus terminé ou déplacé sur le disque).

Le gestionnaire mémorise une liste de blocs libres dont la taille est 1, 2, 4, 8, 16, octets.

L'avantage de ce type de gestion par subdivisions est lorsqu'un bloc de 2^k octets est libéré, il suffit d'examiner uniquement la liste des zones libres de 2^k pour reconstruire un bloc plus grand. Avec les autres algorithmes, il faut parcourir toute la liste.

LA GESTION DE LA MEMOIRE

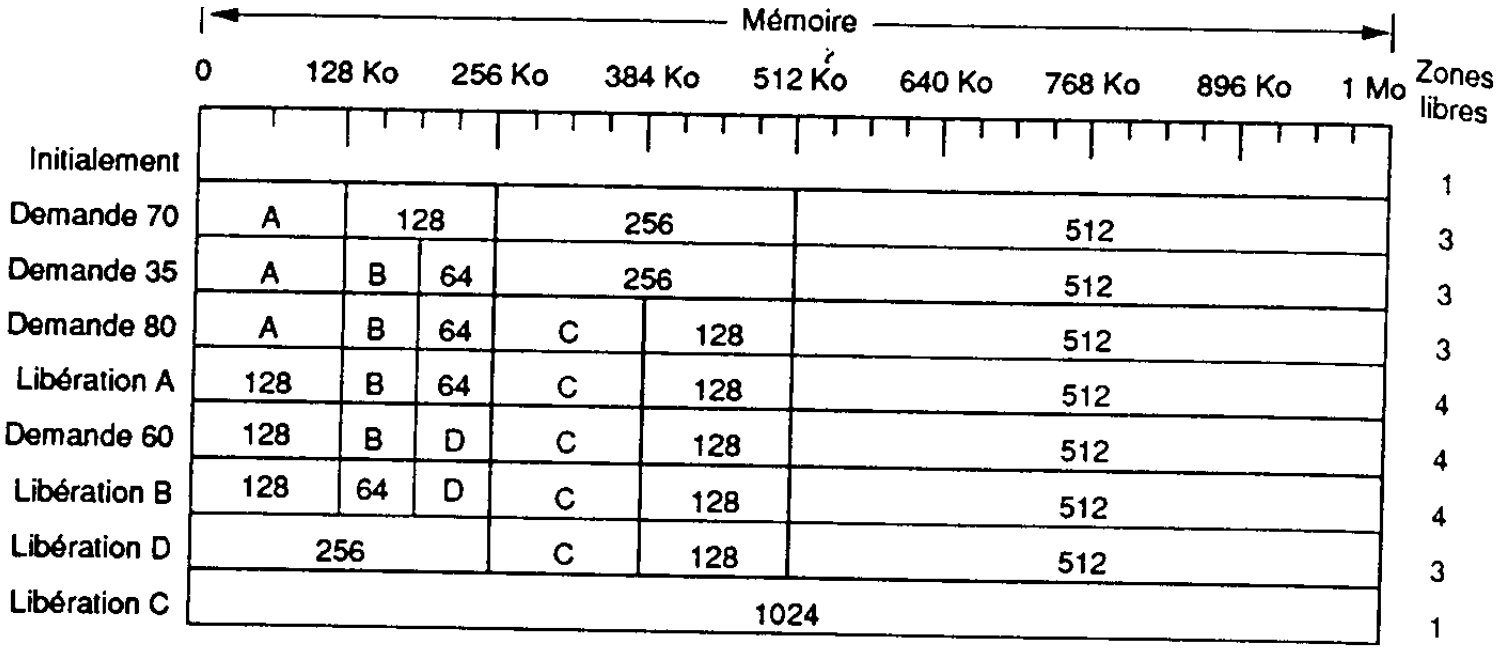


Figure n°11 : Gestion de la mémoire par subdivisions

LA GESTION DE LA MEMOIRE

3) La mémoire virtuelle

Sur les premiers ordinateurs, les programmes étaient trop volumineux pour les faire rentrer en mémoire. La solution fut de diviser les programmes en plusieurs parties **appelées segments de recouvrement (overlay)**. Le segment 0 s'exécute en premier. Lorsqu'il se termine, il appelle un autre segment de recouvrement.
En 1961, on adopta **le principe de mémoire virtuelle**.
L'idée de base est que la taille du programme, des données et de la pile peuvent dépasser la mémoire disponible.
Le système d'exploitation garde en mémoire les parties du programme qui sont utilisées et stocke le reste sur le disque.

Par exemple, on peut exécuter un programme de 1 Mo sur une machine 256 Ko en choisissant judicieusement les 256 Ko du programme à mettre en mémoire principale à tout instant. Les autres parties effectuent des va-et-vient entre le disque et la mémoire.

LA GESTION DE LA MEMOIRE

A) La pagination

La plupart des systèmes à mémoire virtuelle ont recours à la pagination. Sur tout ordinateur, les programmes peuvent générer un certain nombre d'adresses. Quand un programme effectue une instruction comme :

MOVE REG, 1000

il copie le contenu de l'adresse 1000 dans REG (ou l'inverse). Les adresses peuvent être générées à l'aide d'indexations, de registre de base, de segments ou d'autres.

Ces adresses manipulées par les programmes sont appelées des adresses virtuelles et constituent l'espace d'adressage virtuel.

LA GESTION DE LA MEMOIRE

Sur les ordinateurs sans mémoire virtuelle, ces adresses sont directement placées sur le bus de la mémoire et provoquent une lecture ou une écriture.

Lorsque la mémoire virtuelle est utilisée, les adresses virtuelles sont envoyées non pas sur le bus, mais à **l'unité de gestion de la mémoire ou MMU (Memory Management Unit)**. Cette unité permet de traduire les adresses virtuelles en adresses physiques.

LA GESTION DE LA MEMOIRE

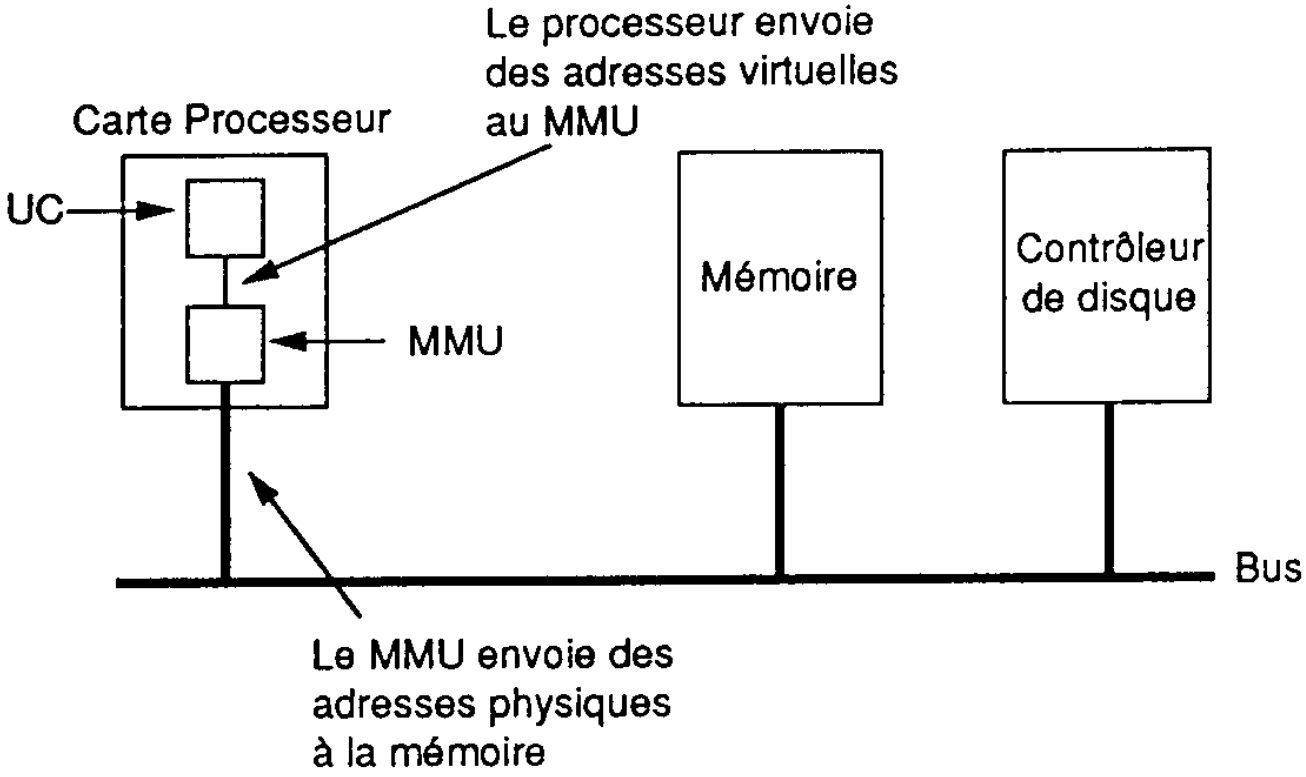


Figure n°12 : Unité de gestion de la mémoire

LA GESTION DE LA MEMOIRE

Un exemple de conversion est donné à la figure n°13. L'ordinateur de cet exemple peut générer des adresses sur 16 bits de 0 à 64K (adresses virtuelles) avec seulement 32 K de mémoire physique. On peut donc écrire de programmes de 64 K mais on ne peut pas les charger entièrement en mémoire. Une partie du programme doit être stockée sur le disque.

L'espace des adresses virtuelles est divisé en petites unités appelées **pages**.
Les unités correspondantes de la mémoire physique sont les **cases mémoire** (**page frames**). Les pages et les cases sont toujours de la même taille. Ici, elles font 4K (ca varie de 512 o à 8 Ko).

LA GESTION DE LA MEMOIRE

Quand un programme essaie de lire l'adresse 0, par exemple avec:

MOVE REG, 0

l'adresse virtuelle 0 est envoyée au MMU. Ce dernier constate que cette adresse virtuelle se situe à la page 0 (0 à 4095) qui appartient à la case 2. Le MMU transforme l'adresse en 8192 et place cette valeur sur le bus.
De même l'instruction : MOVE REG, 8192 est transformée en MOVE REG, 24576.
8192 -> case 6 -> 24 K-> 24*1024=24576.
Idem l'adresse virtuelle 21500 se trouve à 21500/1024=20K +1020 .
Donc l'adresse physique sera 12K+1020=12288 +1020=13308.
Les " x " sur la figure indique que la mémoire n'est pas mappée.

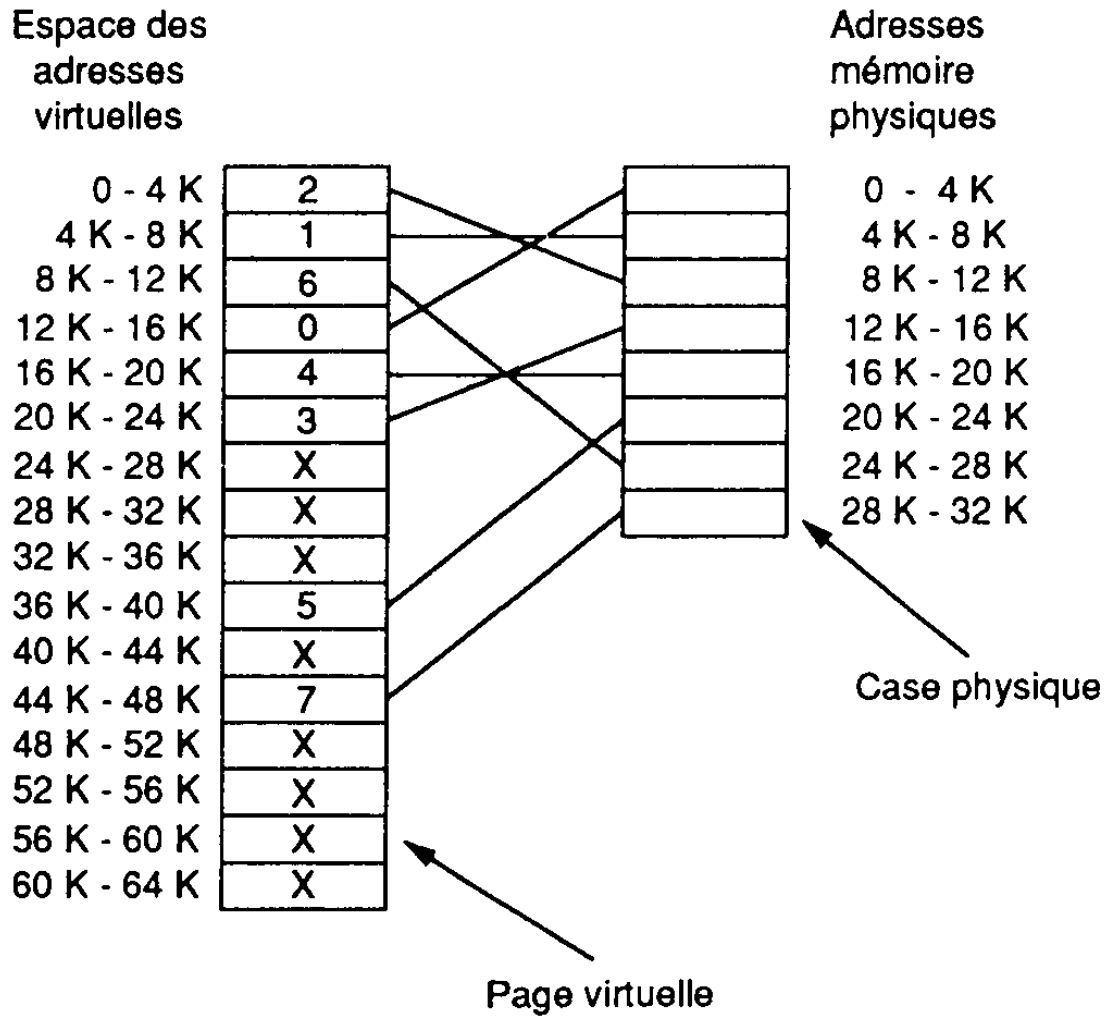


Figure n°13 : Pagination de 64 Ko virtuelles avec 32 Ko physiques

LA GESTION DE LA MEMOIRE

Que se passe-t-il si on adresse une mémoire non mappée ?

Par exemple, en effectuant la commande MOVE REG, 32780(32K + 12).

La MMU constate que cette page n'est pas mappée et provoque un **déroutement** (**trap**) : le processeur est restitué au système d'exploitation.

Ce déroutement est appelé un **défaut de page** (**page fault**).

Le système d'exploitation repère une case peu utilisée et recopie son contenu sur le disque.

LA GESTION DE LA MEMOIRE

Si le système d'exploitation choisit de déplacer la case 1, il charge la page virtuelle 8 à l'adresse physique 4K et effectue deux changements dans la map du MMU.

Il commence à indiquer que la page virtuelle 1 n'est pas mappée afin de pouvoir dérouter les prochains accès aux adresses de 4K à 8K.

Puis, il remplace le " x " de la page virtuelle 8 par un 1 pour pouvoir mapper l'adresse virtuelle 32780 sur l'adresse physique 4108 lorsque l'instruction déroutée sera reéxécutée.

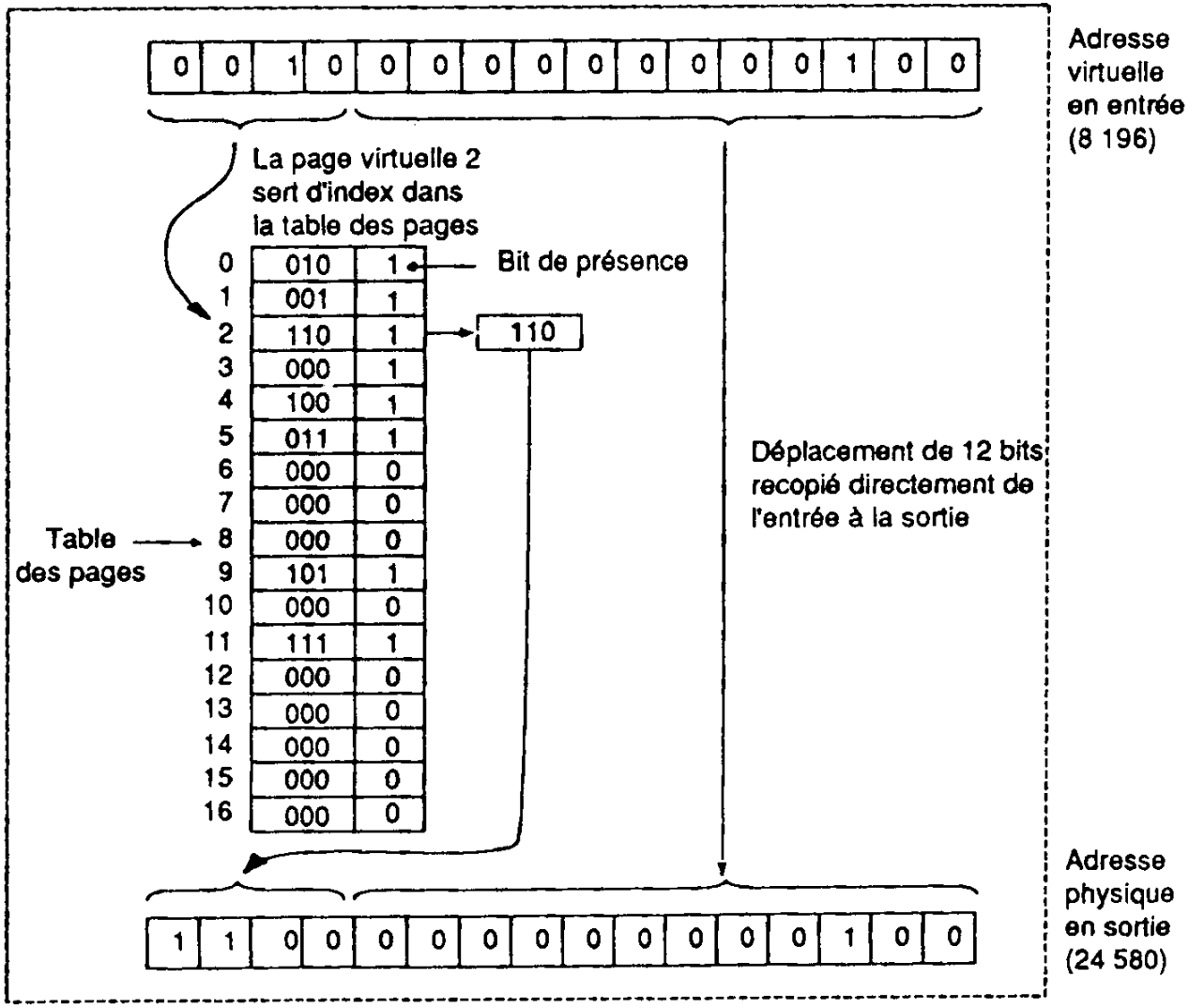


Figure n°14 : Exemple de pagination

LA GESTION DE LA MEMOIRE

A) La segmentation

La mémoire virtuelle étudiée jusqu'à ici est à une dimension, les adresses virtuelles étant comprises entre 0 et une adresse maximale. Dans de nombreux cas, il est préférable d'avoir davantage d'espaces d'adresses virtuelles. Par exemple, un compilateur a besoin de plusieurs tables. Et chacune de ces tables grandit au cours du programme.
Une méthode simple consiste à doter la machine de plusieurs espaces d'adresses indépendants appelés **segments**.
Chaque **segment** est une suite d'adresses continues de 0 à une adresse maximale.
La taille d'un **segment** est comprise entre 0 et l'adresse maximale autorisée.

LA GESTION DE LA MEMOIRE

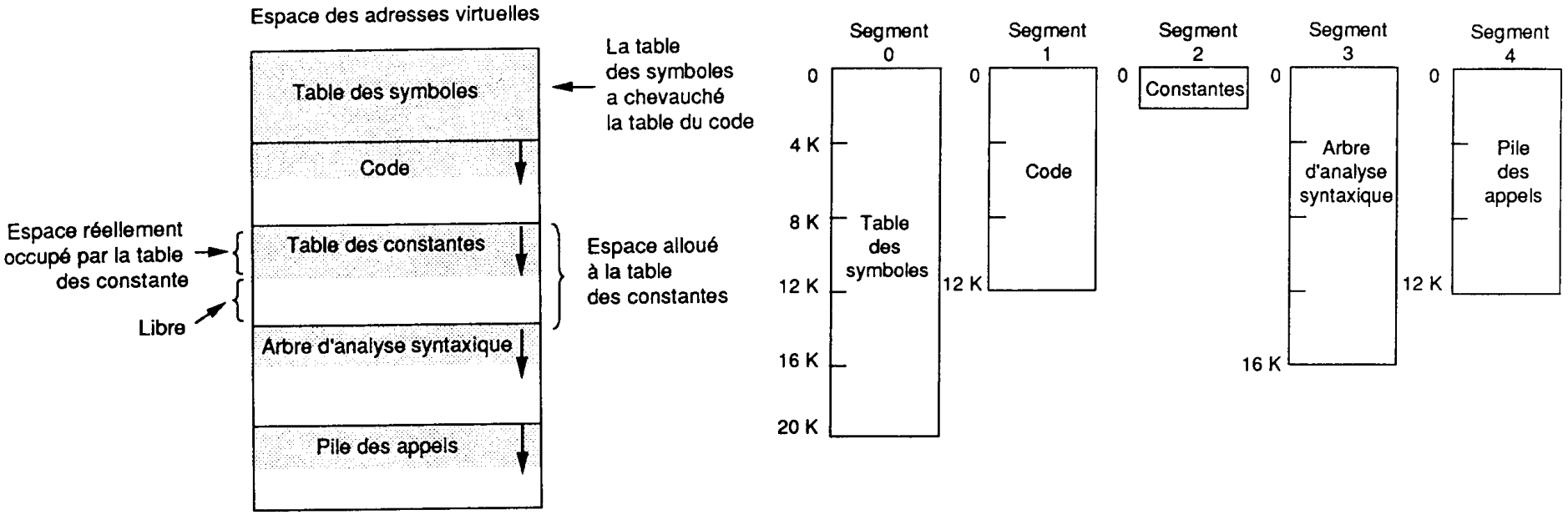


Figure n°15 : Utilité de la segmentation

LA GESTION DE LA MEMOIRE

Un segment est une entité logique que le programmeur doit manipuler. Un segment peut contenir une procédure, un tableau, une pile ou une suite de variables, mais en règle générale ne contient pas un mélange d'objet de type différent.
Une mémoire segmentée simplifie le traitement de structures de données dont la taille varie.
La segmentation simplifie le partage des procédures et des données entre processus. Par exemple, une bibliothèque graphique partagée peut-être placée dans un segment et partagés entre plusieurs processus, ce qui évite de l'avoir dans l'espace d'adressage des processus. Dans un système paginé, pour réaliser des bibliothèques partagées, il faut simuler la segmentation.
Chaque segment étant une entité logique connue du programmeur, peuvent avoir des protections différentes (lecture/écriture). Ce type de protections est utile pour détecter les erreurs de programmation.

Considérations	Pagination	Segmentation
Le programmeur doit-il connaître la technique utilisée ?	Non	Oui
Combien y a-t-il d'espaces d'adressage linéaires ?	1	Plusieurs
L'espace d'adressage total peut-il dépasser la taille de la mémoire physique ?	Oui	Oui
Les procédures et les données peuvent-elles être distinguées et protégées séparément ?	Non	Oui
Peut-on traiter simplement les tables dont les tailles varient ?	Non	Oui
Le partage de procédures entre utilisateurs est-il simplifié ?	Non	Oui
Pourquoi cette technique a-t-elle été inventée ?	Pour obtenir un grand espace d'adressage linéaire sans avoir à acheter de la mémoire physique	Pour permettre la séparation des programmes et des données dans des espaces d'adressage logiquement indépendants et pour faciliter le partage et la protection

Figure n°16 : Pagination et segmentation

Le système de fichiers

Toutes les applications informatiques doivent enregistrer et retrouver des informations. Les problèmes soulevés pour le stockage sont les suivants :

-il faut pouvoir stocker des informations de très grande taille.

-les informations ne doivent pas disparaître lorsque le processus qui les utilise se termine.

-plusieurs processus doivent pouvoir accéder simultanément aux informations.

La solution consiste à stocker les informations dans des fichiers sur des disques ou autres supports.

Les fichiers sont gérés par le **système de fichiers** (**file system**) du système d'exploitation.

L'utilisateur attache plus d'importance à l'interface (changement de nom, effacement,..) offerte par le système d'exploitation qu'a la mise en œuvre proprement dite (nombre de blocs, de secteurs,..).

Le système de fichiers

1) **Les fichiers**

A) L'affectation des noms de fichiers

Un processus qui crée un fichier lui attribue un nom.
Lorsque le processus se termine, le fichier existe toujours et un autre processus peut y accéder au moyen de son nom.
Les règles d'affectation des noms de fichiers varient d'un système à un autre.
Sous UNIX, le nom est une chaîne de 1 à 255 caractères (n'importe quel caractère) avec une distinction majuscule/minuscule.
Sous MS-DOS, le nom est de la forme 8.3 : de 1 à 8 caractères suivis d'un point et de 1 à 3 caractères. L'ensemble des trois caractères après le point s'appelle l'extension et permet de distinguer le type de fichier.
Sous UNIX, on peut avoir plusieurs extensions (toto.c.Z).

Le système de fichiers

.bak fichier de sauvegarde
.tif fichier image au format TIFF
.gif fichier image au format GIF
.ps fichier image au format Postscript
.c fichier source en C
.obj fichier objet crée par le compilateur.
.lib bibliothèque de fichiers .obj utilisé par l'éditeur de liens.
.html fichier source en langage HTML.

L'extension est surtout obligatoire pour les fichiers sources en programmation (.h, .c, .cpp).

B) La structure des fichiers

Les fichiers peuvent être structurés de différentes manières.

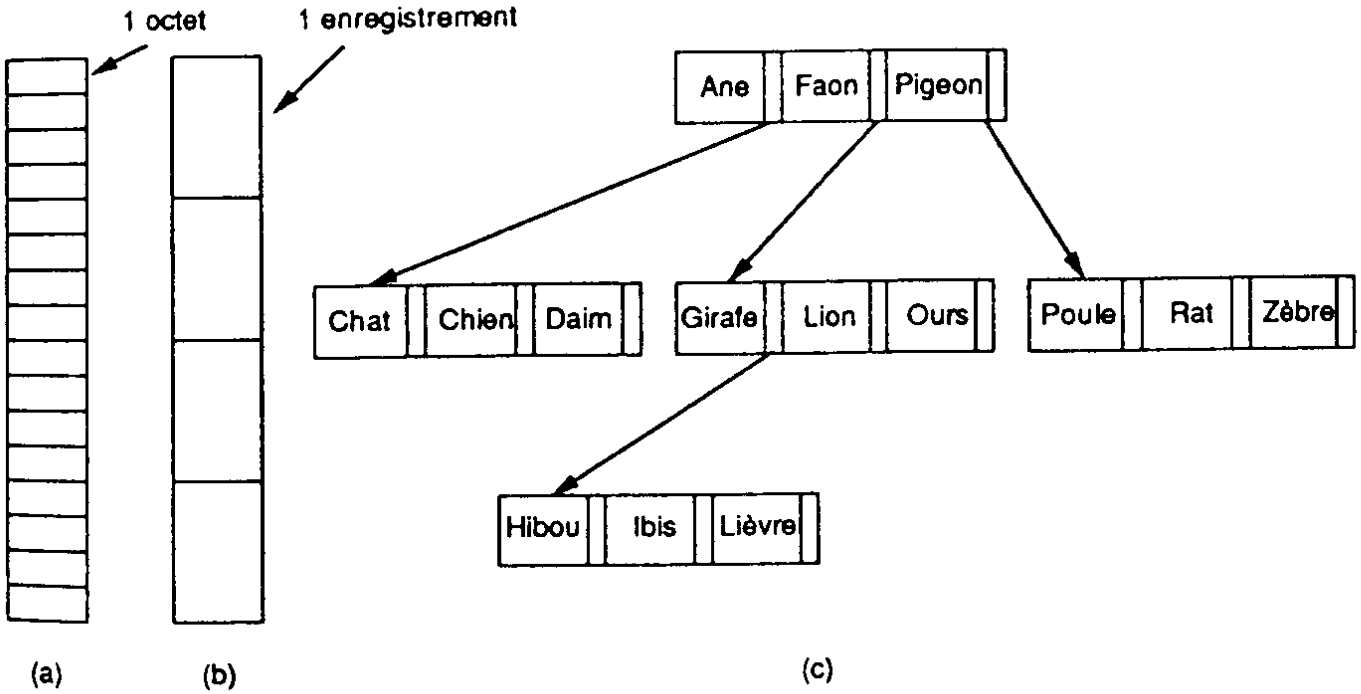


Figure n°17 : Structure des fichiers

Le système de fichiers

(a) est une simple suite d'octets.
Le système d'exploitation ne connaît pas et ne s'occupe pas du contenu du fichier. Il ne gère que des octets. Les structures internes sont réalisées par les programmes utilisateurs. Cette approche offre la plus grande souplesse.
UNIX et MS-DOS sont structurés ainsi.

Le premier niveau de structuration est donné en (b).
Dans ce modèle, un fichier est une suite d'enregistrements de taille fixe qui ont chacun une structure interne.
Le concept de base des enregistrements de taille fixe est qu'une opération de lecture renvoie un enregistrement et qu'une opération d'écriture ajoute un enregistrement.
A l'époque des cartes perforées de 80 colonnes, de nombreux systèmes d'exploitation avaient des fichiers constitués d'enregistrement de 80 caractères.

Le système de fichiers

La troisième méthode est présentée en (c).
Un fichier est une arborescence d'enregistrements qui n'ont pas tous nécessairement la même taille et qui contienne une clef à une position donnée.
L'arbre est trié en fonction de cette clef afin d'accélérer la recherche d'une clef donnée.
Ce type de structure est surtout réalisé pour chercher non pas l'enregistrement suivant mais l'enregistrement qui contient une clef à une position donnée (base de données).
Lors de l'ajout d'un enregistrement, c'est le système d'exploitation qui décide de l'emplacement et non pas l'utilisateur.

Le système de fichiers

C) Les types de fichiers

Il existe différents types de fichiers :

-les fichiers ordinaires qui contiennent les informations des utilisateurs,

- les catalogues sont des fichiers systèmes qui maintiennent la structure du système de fichiers,

- les fichiers spéciaux caractères sont liés aux E/S et permettent de modéliser les périphériques d'E/S série tels que les terminaux, les imprimantes et les réseaux,

- les fichiers spéciaux blocs modélisent les disques.

Le système de fichiers

Les fichiers ordinaires sont de deux types ASCII ou binaires.

Les fichiers ASCII contiennent des lignes de texte. La fin de ligne est modélisée par un CR sous UNIX et par un LF et un CR sous DOS.
Le grand avantage des fichiers ASCII est qu'ils peuvent être affichés, imprimés sans modification et qu'ils peuvent être édité au moyen d'un éditeur de texte standard.

Les autres fichiers sont des fichiers binaires, ce qui signifie simplement qu'ils ne sont pas des fichiers ASCII. Leur impression donne une suite incompréhensible de signes. Ces fichiers ont en général une structure interne. La figure n°18 présente deux exemples.

Le système de fichiers

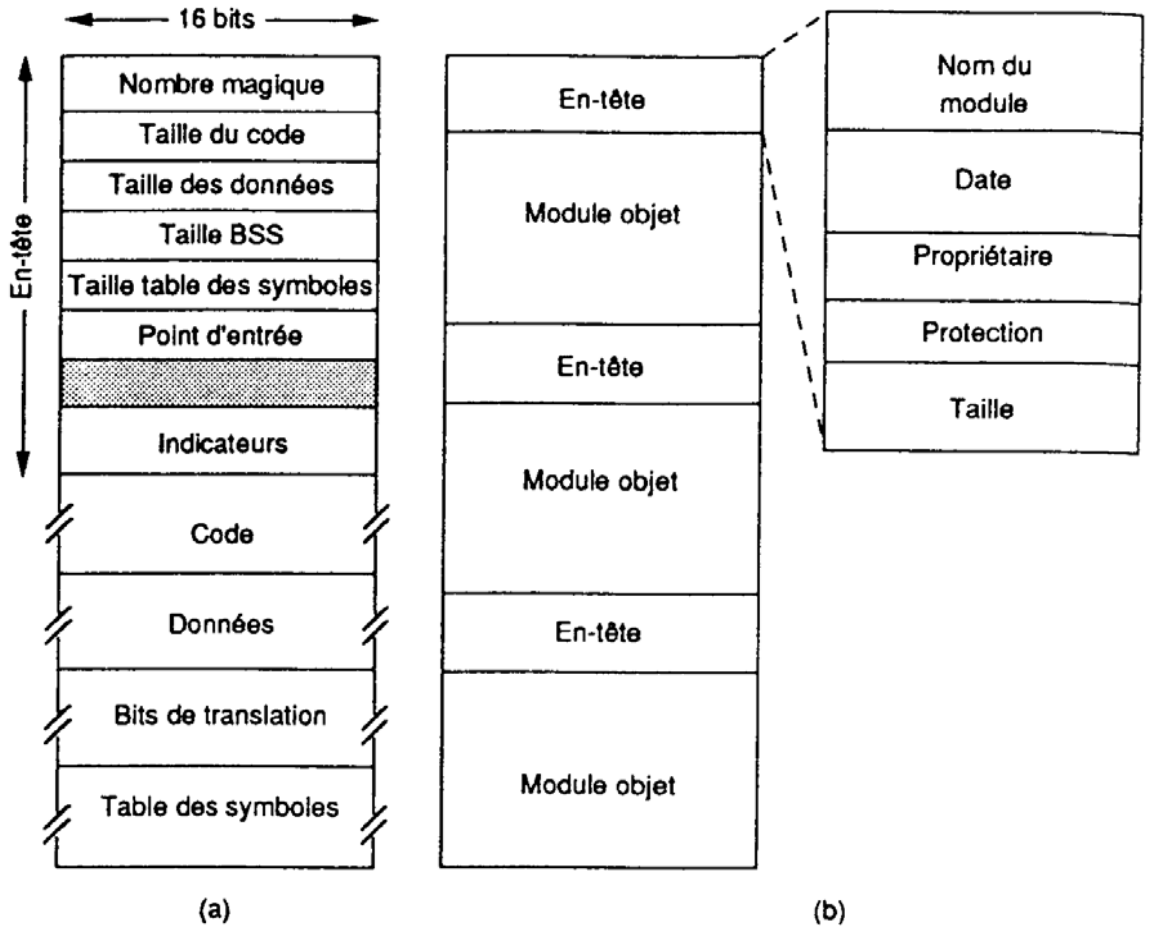


Figure n°18 : Structure internes des fichiers exécutables et bibliothèques

Le système de fichiers

D) l'accès aux fichiers

Il existe deux types d'accès :

- l'accès **séquentiel**, car lorsque l'on utilise des bandes magnétiques on est obligé de lire les enregistrements à la suite.
- l'accès **aléatoire** qui permet d'accéder directement aux fichiers (disque).

E) Les attributs des fichiers

Chaque fichier possède un nom et des données.
De plus, tous les systèmes d'exploitation associent des informations complémentaires à chaque fichier, par exemple la date de création, la taille, ce sont les **attributs des fichiers**.

Le système de fichiers

Champ	Signification
Protection	Qui peut accéder au fichier et de quelle façon
Mot de passe	Mot de passe requis pour accéder au fichier
Créateur	Personne qui a créé le fichier
Propriétaire	Propriétaire courant
Indicateur lecture seule	0 pour lecture/écriture, 1 pour lecture seule
Indicateur fichier caché	0 pour fichier normal, 1 pour ne pas l'afficher dans les listages
Indicateur fichier système	0 pour fichier normal, 1 pour fichier système
Indicateur d'archivage	0 le fichier a été archivé, 1 il doit être archivé
Indicateur fichier ASCII/binaire	0 pour fichier ASCII, 1 pour fichier binaire
Indicateur accès aléatoire	0 pour accès séquentiel, 1 pour accès aléatoire
Indicateur fichier temporaire	0 pour fichier normal, 1 pour supprimer le fichier lorsque le processus se termine
Indicateur de verrouillage	0 pour fichier non verrouillé, 1 pour fichier verrouillé
Longueur d'enregistrement	Nombre d'octets dans l'enregistrement
Position de la clé	Position relative de la clé dans chaque enregistrement
Longueur de la clé	Nombre d'octets du champ clé
Date de création	Date et heure de création du fichier
Date du dernier accès	Date et heure du dernier accès au fichier
Date de modification	Date et heure de la dernière modification
Taille courante	Nombre d'octets du fichier
Taille maximale	Taille maximale autorisée pour le fichier

Figure n°19 : Exemple d'attributs de fichier

F) Les opérations sur les fichiers

Les fichiers permettent de stocker des informations et de les rechercher ultérieurement. Les systèmes d'exploitation fournissent des méthodes variées pour réaliser le stockage et la recherche. Les principaux appels sont les suivants.

1. **CREATE**. Le fichier est crée sans données. Cet appel a pour but d'indiquer la création du fichier et de fixer un certain nombre de paramètres.
2. **DELETE**. Le fichier devenu inutile est supprimé pour libérer de 1'espace sur le disque. é existe toujours un appel système i cette fin. Certains systèmes d'exploitation offrent même la possibilité de supprimer automatiquement les fichiers non utilisés pendant n jours.

3. **OPEN**. Un fichier doit être ouvert avant qu'un processus puisse l'utiliser. L'appel OPEN permet au système de charger en mémoire les attributs et la liste des adresses du fichier sur le disque afin d'accélérer les accès ultérieurs.
4. **CLOSE**. Lorsqu'il n'y a plus d'accès au fichier, les attributs et la liste des adresses du fichier ne sont plus requis. Le fichier doit être fermé pour libérer de l'espace dans les tables internes. De nombreux systèmes incitent les utilisateurs à fermer les fichiers en imposant un nombre maximal de fichiers ouverts par processus.
5. **READ**. Des données sont lues à partir du fichier. En général, les octets sont lus à partir de la position courante. L'appelant doit spécifier le nombre d'octets demandés ainsi qu'une mémoire tampon de réception.

6. **WRITE**. Des données sont écrites dans le fichier à partir de la position courante. Si la position courante est située à la fin du fichier, la taille du fichier augmente. Si la position courante est au milieu du fichier, les anciennes données sont remplacées et définitivement perdues.
7. **APPEND**. Cet appel est une version restreinte de WRITE qui ajoute des données à la fin du fichier. Les systèmes qui ont un nombre restreint d'appels système ne disposent pas en général de l'appel APPEND. Il existe souvent plusieurs manières pour effectuer une opération de sorte que ces systèmes permettent néanmoins d'effectuer un APPEND.
8. **SEEK**. Pour les fichiers à accès aléatoire, il faut indiquer la position des données à lire ou à écrire. L'appel système SEEK, souvent utilisé, modifie la position courante dans le fichier. A la suite de cet appel, les données sont lues ou écrites à partir de la nouvelle position courante.

Le système de fichiers

9. **GET ATTRIBUTES**. Les processus doivent souvent lire les attributs des fichiers pour effectuer certaines opérations. Par exemple, le programme make d'UNIX est utilisé pour gérer des développements de logiciels informatiques constitués de plusieurs fichiers source. Lorsque make est appelé, il examine les dates de modification de tous les fichiers source et objet, et effectue le plus petit nombre de compilations nécessaires pour mettre à jour le projet. Pour effectuer son travail, il doit accéder aux attributs des fichiers, et plus particulièrement à leur date de modification.
10. **SET ATTRIBUTES**. Quelques attributs peuvent être modifiés par les utilisateurs et peuvent être renseignés après la création du fichier. Cet appel système réalise cette opération. Les informations relatives à la protection constituent un exemple évident d'attributs modifiables. La plupart des indicateurs le sont également.

11. **RENAME**. Il est fréquent de vouloir modifier le nom d'un fichier. Cet appel système réalise cette opération. Il n'est pas vraiment indispensable puisqu'un fichier peut toujours être copié dans un fichier qui porte le nouveau nom et l'ancien fichier supprimé, ce qui suffit en général.

```
/* copiefichier copie le fichier 'src' dans 'dst'. */
/* Les erreurs ne sont pas signalées */

#include <sys/types.h>          /* Définitions de types */
#include <fcntl.h>              /* Définit O_RDONLY, etc. */
#include <stdlib.h>             /* Prototypes des appels système */
#include <unistd.h>             /* Prototypes des appels système */

void main(int argc, char *argv[]);/* Prototype C ANSI */

#define TAILLE_BUF 4096         /* Unité de lecture/écriture */
#define MODE 0666               /* Mode de protection */
                                /* du fichier (rw-rw-rw-) */
void main(int argc, char *argv[])/* argc : nb d'arguments */
{                               /* argv : ptrs sur arguments */
   int src, dst, in, out;

   char buf[TAILLE_BUF];

   if (argc != 3) exit(1);      /* Mauvais nombre de paramètres */

   /* Ouvrir le fichier source et créer le fichier de destination. */
   src = open(argv[1], O_RDONLY);/* Source en lecture seule */
   if (src < 0) exit(2);        /* Impossible d'ouvrir la source */
   dst = open(argv[2], MODE);   /* Créer la destination */
   if (dst < 0) exit(3);        /* Impossible de créer la dest. */

   /* Tout s'est bien passé. Effectuer la copie. */
   while (1) {                  /* Boucler indéfiniment */
      in = read(src, buf, TAILLE_BUF);/* Lire la source */
      if (in <= 0) break;       /* Quitter en fin de fichier */
      out = write(dst, buf, in); /* Ecrire les octets ds la dest. */
      if (out <= 0) break;      /* Quitter en cas d'erreur */
   }

   close(src);                  /* Fermer la source */
   close(dst);                  /* Fermer la destination */
   exit(0);                     /* Quitter */
}
```

Figure n°19 : Exemple d'utilisation des appels systèmes

2) Les catalogues (répertoires)

Le système mémorise les noms, attributs et adresses des fichiers dans des catalogues qui sont eux-mêmes des fichiers.
Un répertoire contient un certain nombre d'entrées, une par fichier.

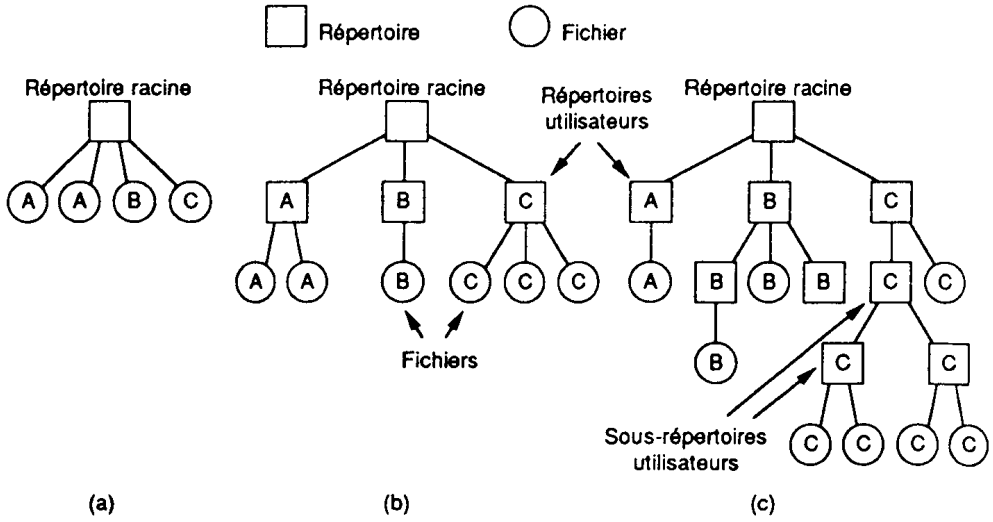


Figure n°19 : Structure des répertoires

Le système de fichiers

Lorsque le système de fichier se présente sous la forme d'un arbre de catalogue, il faut trouver un moyen pour spécifier les noms de fichiers.

Il existe deux méthodes : chemin absolu et le chemin relatif.

Le chemin **absolu** spécifie le fichier à partir de la racine de l'arborescence (notée / sous UNIX, \ sous MS-DOS). Tandis que le chemin **relatif** spécifie le fichier à partir du répertoire où on se trouve.

Généralement le signe **.** désigne le répertoire courant et le signe **..** le répertoire père.

3) La mise œuvre des systèmes de fichiers

a) Le stockage des fichiers

Le concept fondamental du stockage des fichiers est la mémorisation des adresses des blocs de chaque fichier.

Allocation contiguë

La méthode d'allocation la plus simple consiste à stocker chaque fichier dans une suite de blocs consécutifs. Un fichier de 50 Ko occuperait 50 blocs consécutifs sur un disque dont la taille des blocs est de 1 Ko. Les avantages sont la simplicité de mise œuvre puisqu'il suffit de mémoriser un nombre, l'adresse du premier bloc et les performances puisque tout fichier peut être lu en une seule opération. Ses inconvénients sont que la taille des fichiers doit être connue à la création et la fragmentation du disque qu'elle introduit.

Le système de fichiers

L’allocation au moyen d’une liste chaînée

Cette méthode consiste à sauvegarder les blocs dans une liste chaînée. Le premier mot de chaque bloc est un pointeur sur le bloc suivant. Le reste du bloc contient les données. Contrairement à l’allocation contiguë, tous les blocs peuvent être utilisés. Il n’y a pas d’espace perdu par la fragmentation du disque. L’entrée du catalogue stocke simplement l’adresse du premier bloc, les autres blocs sont trouvés à partir de ce bloc. Si la lecture séquentielle d’un fichier est simple, accès aléatoire est extrêmement lent. Par ailleurs le pointeur sur le bloc suivant occupant quelques octets, l'espace réservé aux données dans chaque bloc n’est plus une puissance de deux.

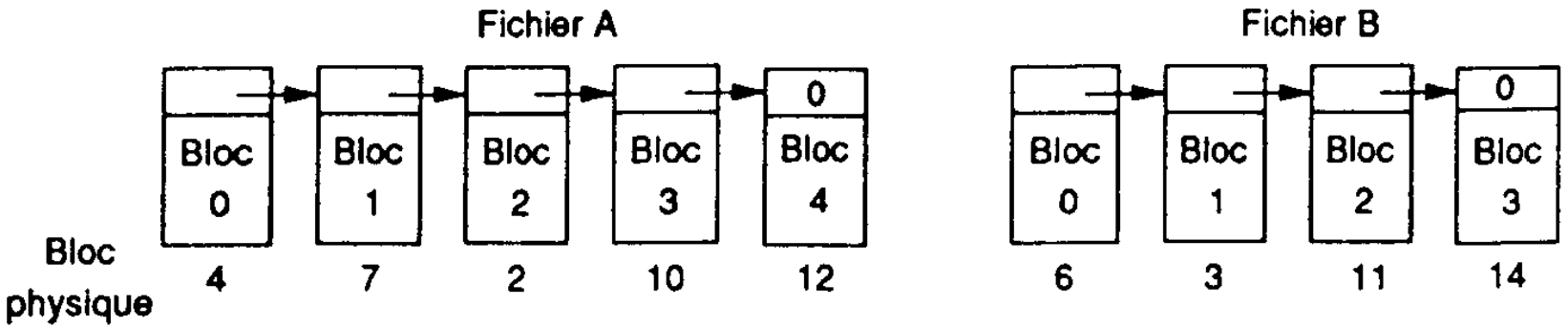


Figure n°20 : Allocation par listes chaînée

L'allocation au moyen d'une liste chaînée indexée

Les inconvénients de l'allocation au moyen d'une liste chaînée peuvent être éliminée en retirant le pointeur de chaque bloc pour le placer dans une table ou un index en mémoire.

La figure n°20 présente le stockage de deux fichiers A et B.
A occupe les blocs physiques 4, 7, 2, 10 et 12 et B 3, 11 et 14.

Cette méthode libère intégralement l'espace du bloc de données. Elle facilite également les accès aléatoires. La liste doit toujours être parcourue pour trouver un déplacement donné dans un fichier. MS_DOS utilise cette technique. Le principal inconvénient vient du fait que la table doit être entièrement en mémoire en permanence.

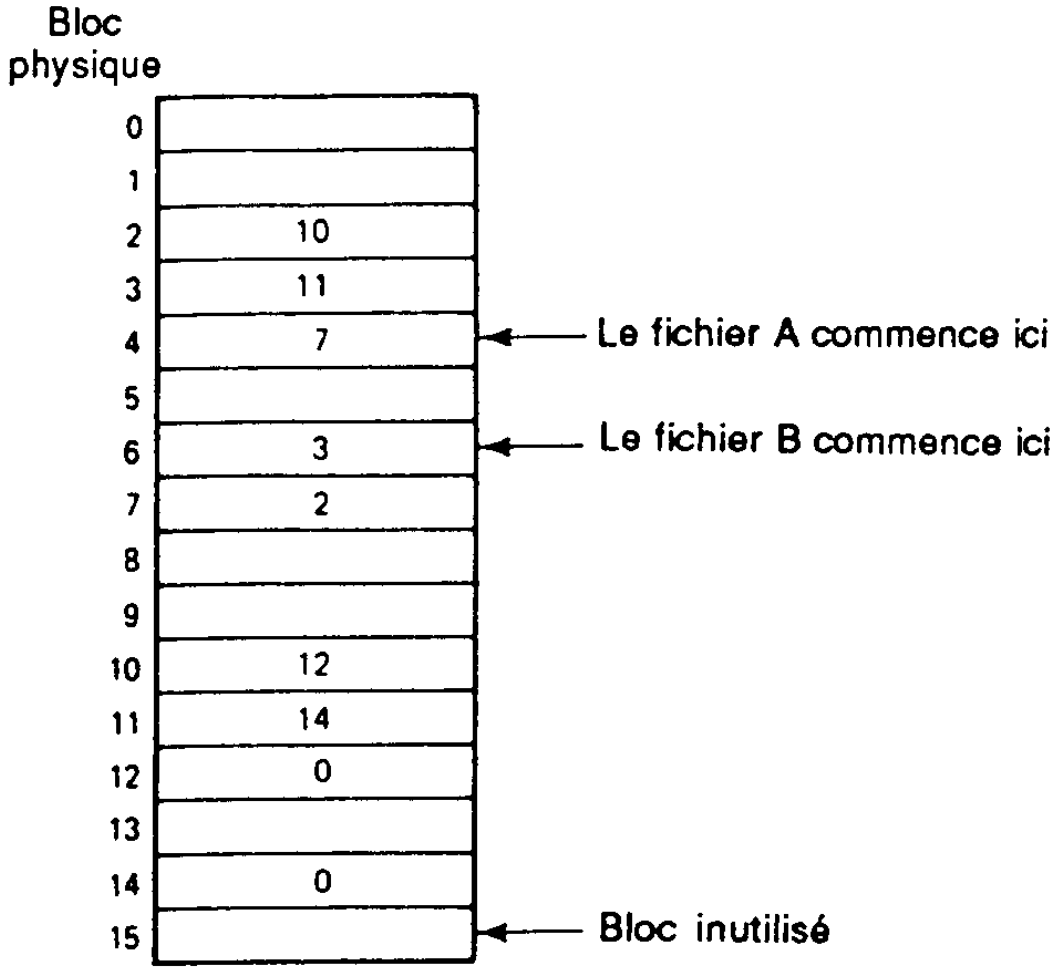


Figure n°21 : Allocation par listes chaînée indexée

Les nœuds d'informations

Cette méthode consiste à associer à chaque fichier une petite table appelée **nœud d'information (i-node)**. Cette table contient les attributs et les adresses sur le disque des blocs du fichier. Les premières adresses du disque sont contenues dans le nœud d'information (10) de sorte que les informations des petits fichiers y sont entièrement contenues lorsqu'il est chargé en mémoire à l'ouverture du fichier. Pour les fichiers plus importants, une des adresses du i-node est celle d'un bloc disque appelé bloc d'indirection qui contient les adresses additionnelles. On peut aller jusqu'à un bloc d'indirection triple avec UNIX.

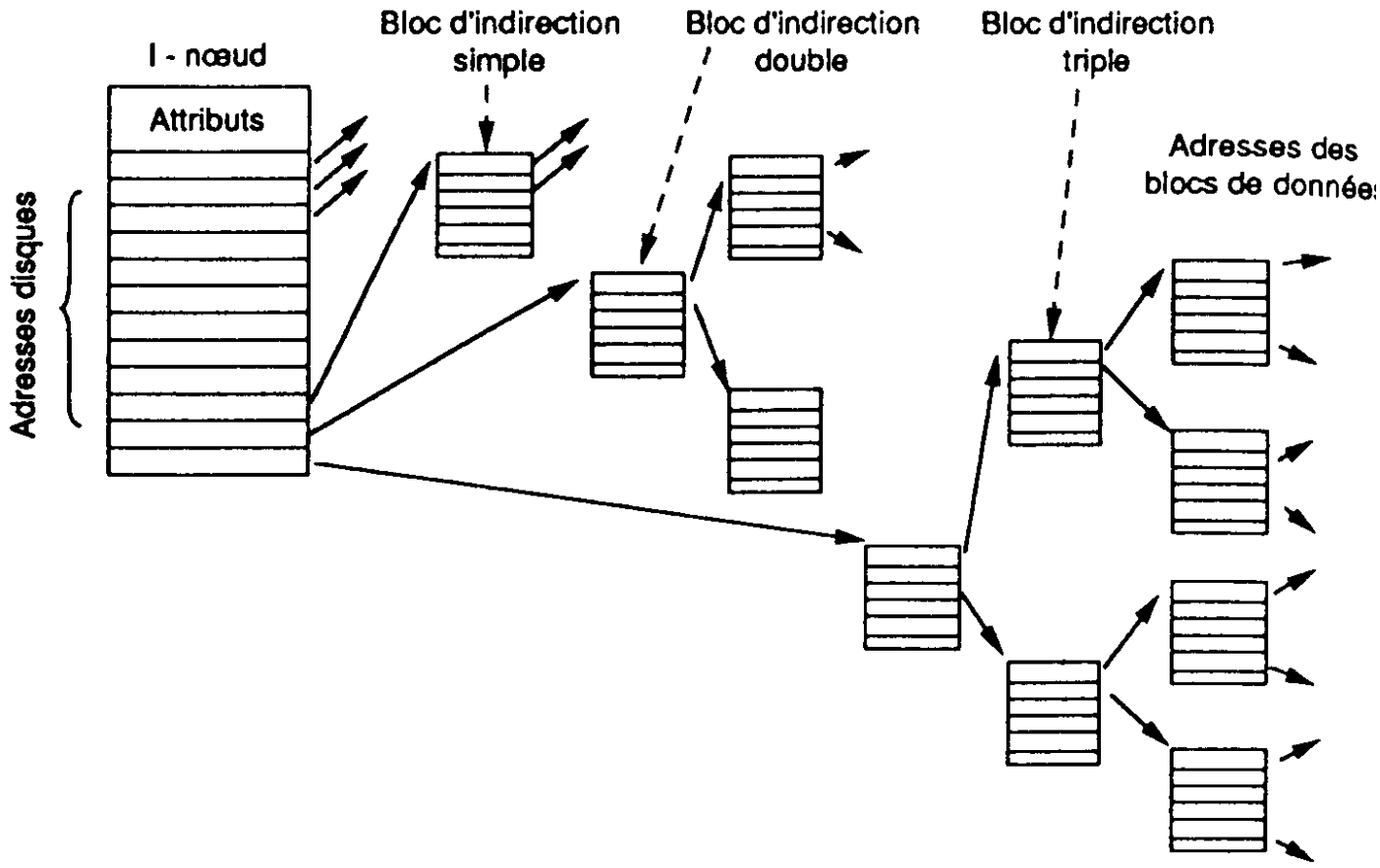


Figure n°22 : L'allocation par nœud d'information

B) Mise en œuvre des répertoires

Il faut ouvrir un fichier pour pouvoir le lire. Quand on ouvre un fichier, le système d'exploitation utilise le chemin accès donné pour localiser l'entrée dans le répertoire.

Cette entrée fournit les informations nécessaires pour retrouver les blocs sur le disque.

En fonction du système, ces informations peuvent être les adresses disques de tout le fichier (allocation contiguë), le numéro du premier bloc (listes chaînées) ou le numéro du nœud d'information.

Le système de fichiers

Sous MS-DOS, une entrée d'un répertoire est constituée de 32 octets. Elle comporte le nom, les attributs et le numéro du premier bloc.

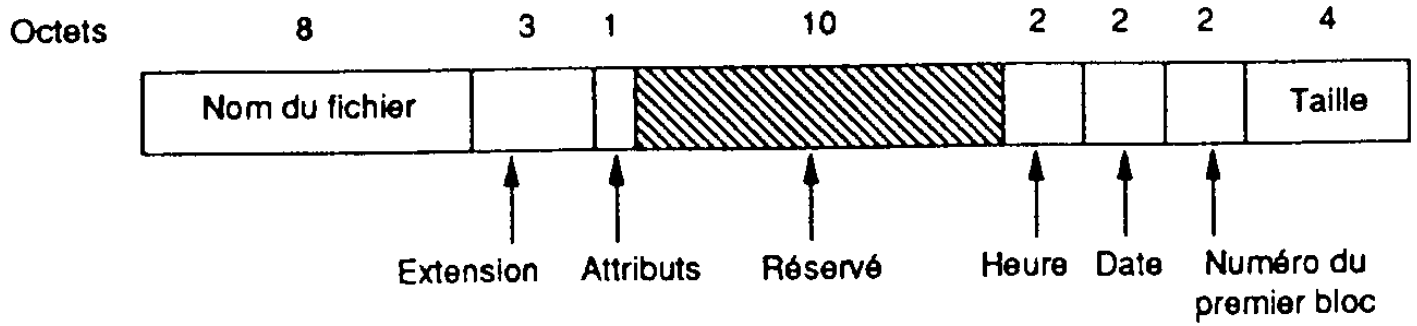


Figure n°23 : Les répertoires sous MS-DOS

Le système de fichiers

Sous UNIX, chaque entrée contient un nom de fichier et le numéro de son nœud d'information. Toutes les informations concernant le fichier, son type, sa date,... sont stockées dans le noeud d'information.

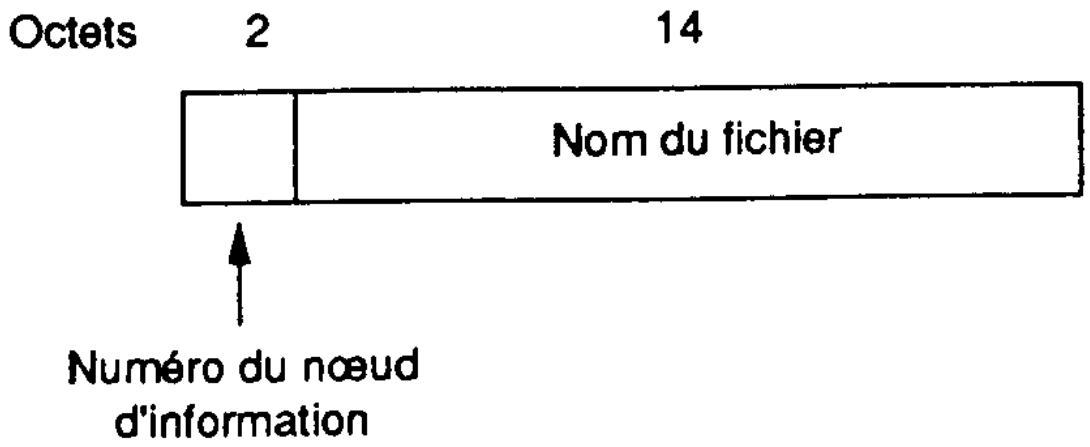


Figure n°24 : Les répertoires sous UNIX

Voyons maintenant comment le système d'exploitation trouve les blocs physiques sous UNIX. La figure n°12 présente cette recherche pour le fichier /usr/ast/courier.

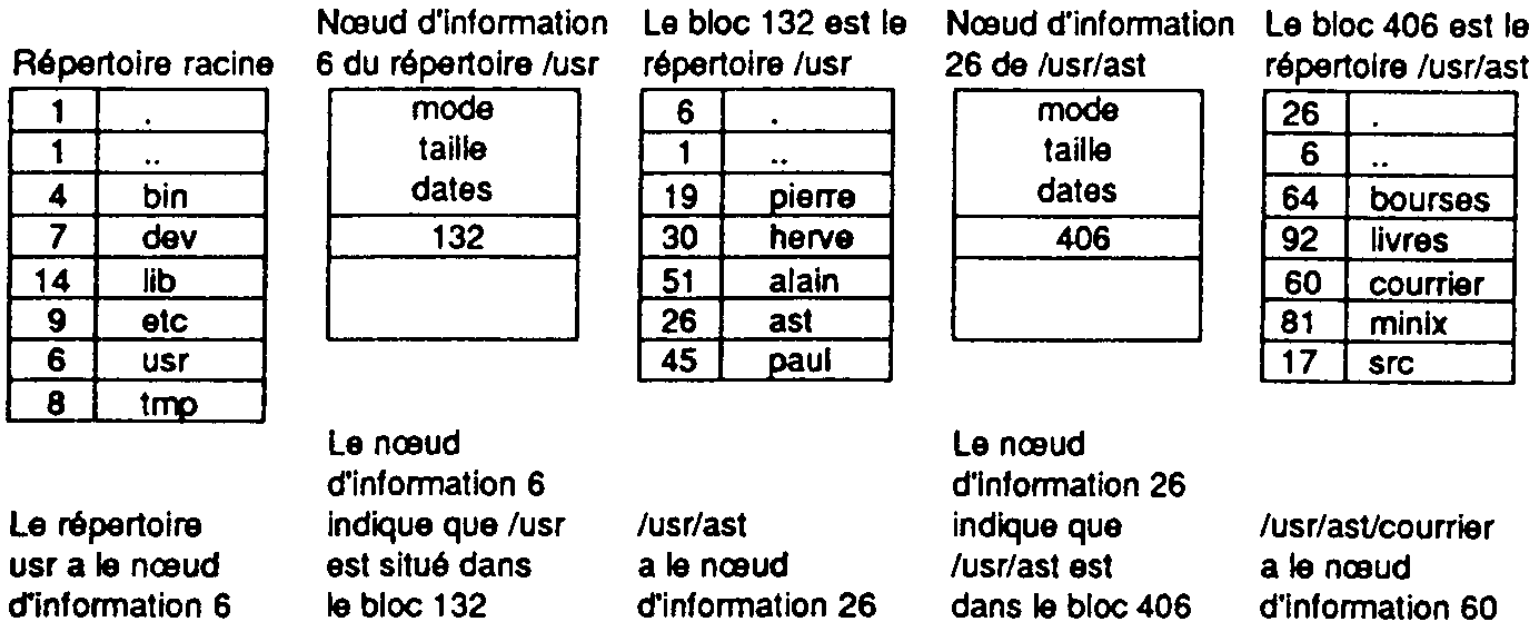


Figure n°25 : Les étapes de recherche du fichier /usr/ast/courier

LES ENTREES/SORTIES

Le contrôle des périphériques d'entrées/sorties de l'ordinateur est une fonction primordiale d'un système d'exploitation.

Le système d'exploitation doit :

- envoyer des commandes aux périphériques,
- intercepter les interruptions,
- traiter les erreurs.

Il doit aussi fournir une interface simple et facile d'emploi entre les périphériques et le reste du système. Cette interface doit être dans la mesure du possible, la même pour tous les périphériques utilisés, c'est-à-dire indépendante du périphérique utilisé.
Ce chapitre se divise en deux parties, les principes des E/S du point de vue :

- matériel,
- logiciel.

LES ENTREES/SORTIES

I Principe du matériel d'Entrées/Sorties.

Les périphériques d'Entrées/Sorties se répartissent en deux catégories : les périphériques **blocs** et les périphériques **caractères**.

Un périphérique bloc mémorise les informations dans les blocs de taille fixe, chaque bloc ayant une adresse propre.
Les tailles vont de 128 à 1024 octets.
La propriété fondamentale de ces périphériques et qu'ils permettent de lire et d'écrire un bloc indépendamment de tous les autres. Les disques sont des périphériques bloc.

La frontière entre les périphériques bloc et les autres n'est pas toujours bien définie.

Par exemple, si une bande magnétique contient des bloc de 1 Ko et si on veut lire le bloc N, le dérouleur doit rembobiner la bande et se positionner sur le bloc N. Cette recherche est analogue a celle sur un disque mais le temps est beaucoup plus long. De plus, on ne peut pas réécrire un bloc au milieu d'une bande. Donc les bandes peuvent être utilisées comme des périphériques blocs mais en général, elle ne le sont pas.

Le deuxième type de périphérique d'Entrées/Sorties est le périphérique **caractère**.
Il accepte un flot de caractères sans se soucier d'une quelconque structure de bloc. Il ne peut pas être adressé et ne possède pas de fonction de recherche. Les terminaux, les imprimantes, les bandes de papier, les cartes perforées, les interfaces réseaux, les souris, les rats(pour les expériences) et la plupart des périphériques qui se comportent pas comme des disques peuvent être considérés comme des périphériques caractères.

LES ENTREES/SORTIES

Cette classification n'est pas parfaite, quelques périphériques appartiennent à aucune catégories. Les horloges ne possèdent pas de blocs et n'acceptent pas non plus de flux de caractères. Elles ne font que générer des interruptions à des intervalles réguliers.

a) les contrôleurs de périphériques

Les unités d'Entrées/Sorties sont constitués de composants mécaniques et électronique.
Ces derniers sont appelés contrôleurs de périphérique ou adaptateurs (cartes de circuits imprimés que l'on insère dans l'ordinateur).
Les composants mécaniques constituent le périphérique lui-même, relié aux contrôleurs via une interface généralement normalisé.
Le SE communique toujours avec le contrôleur et non avec le périphérique.

LES ENTREES/SORTIES

La plupart des micro-ordinateur utilise le modèle à bus unique pour la communication entre le processeur et le contrôleur. (figure n°1). Les gros ordinateurs utilisent, souvent en revanche, un modèle à plusieurs bus ainsi que les ordinateurs dédiées aux E/S, appelés voies E.S (I/O channels), qui allège la tache du processeur principal.

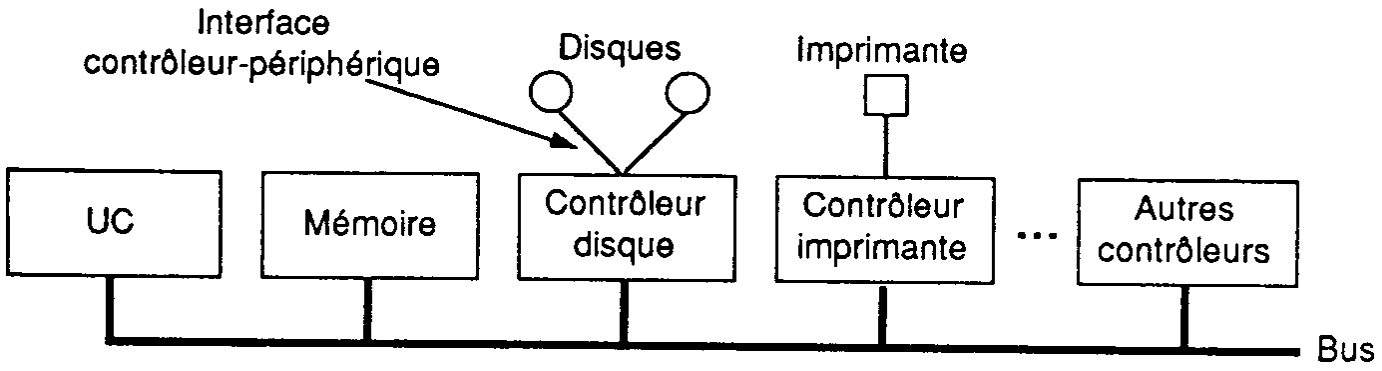


Figure n°26 : Modèle connexion entre le processeur, la mémoire, les contrôleurs et les périphériques

LES ENTREES/SORTIES

L'interface entre le contrôleur et le périphérique est souvent de très bas niveau.
Par exemple, un contrôleur d'écran fonctionne comme un contrôleur série de bas niveau. Il lit dans la mémoire les octets qui contiennent les caractères à afficher et génère des signaux pour moduler le faisceau d'électron qui dessine les caractères à l'écran. Il génère aussi à la fin de chaque ligne des signaux pour repositionner la faisceau au début de la ligne suivante...
Sans ce contrôleur, le programmeur d'un SE aurait à programmer le balayage du tube cathodique, alors qu'il suffit au SE d'initialiser le contrôleur avec quelques paramètres comme le nombre de caractères par ligne et le nombre de lignes par écran, pour qu'il se charge de diriger le faisceau.

LES ENTREES/SORTIES

Chaque contrôleur communique avec le processeur par l’intermédiaire de quelques registres.

Sur certains ordinateurs, ceux-ci se situent dans l'espace mémoire adressable (E/S mappées en mémoire). Le 680x0 utilise cette méthode, tandis que d'autre utilisent un espace mémoire particulier pour les E/S et allouent à chaque contrôleurs une partie de cet espace.(figure n°26)

LES ENTREES/SORTIES

Contrôleur d'E/S	Adresses d'E/S	Vecteurs d'interruption
Horloge	040 - 043	8
Clavier	060 - 063	9
RS232 auxiliaire	2F8 - 2FF	11
Disque dur	320 - 32F	13
Imprimante	378 - 37F	15
Ecran monochrome	380 - 3BF	–
Ecran couleur	3D0 - 3DF	–
Lecteur de disquettes	3F0 - 3F7	14
RS232 principale	3F8 - 3FF	12

Figure n°27 : Adresses et interruptions de l'IBM PC

B) L'accès direct à la mémoire (DMA)

La lecture d'un disque sans DMA : le contrôleur lit un bloc, bit après bit, jusqu'à ce que le tampon soit remplit. Puis il calcule une somme de contrôle pour vérifier la donnée, ensuite le contrôleur génère une interruption. Le SE peut alors, lire le tampon et recopier les données lues en mémoire au moyen d'une boucle.

Cette boucle exécutée par le processeur pour lire un octet après l'autre consomme beaucoup de temps processeur.

Le DMA a été crée pour libérer le processeur du travail de bas niveau.

Le contrôleur après avoir lu le bloc du périphérique, placé les données dans le tampon et vérifié la checksum, copie le premier octet dans la mémoire principale à l'adresse spécifié par au DMA.

Puis incrémente cette adresse et décrémente le compteur du DMA du nombre d'octet transférés.

Cette opération est répétée jusqu'à ce que le compteur du DMA atteigne la valeur 0.

Le contrôleur génère alors une interruption ainsi le SE n'aura pas à transférer les données

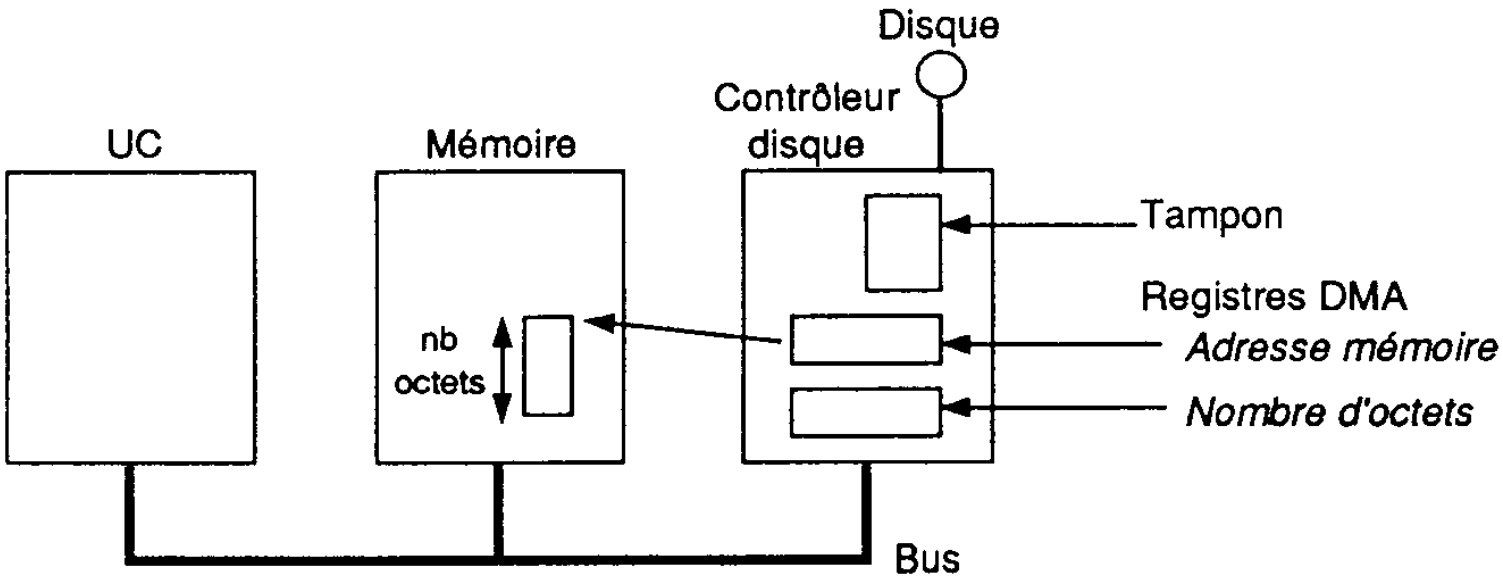


Figure n°28 : Principe du DMA