Chapitre 1

Introduction

UN système d'exploitation est un programme qui doit permettre aux utilisateurs d'utiliser les fonctionnalités d'un ordinateur. Mais il doit aussi aider le programmeur à développer des logiciels de la façon la plus efficace possible. Un système d'exploitation est mis en route dès qu'on met en marche l'ordinateur. Dans les grosses machines, il est toujours en exécution. Le système constitue donc une interface entre l'utilisateur et la machine physique.

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

Malgré les différences de point de vue, de forme, de taille et de type, les ordinateurs se composent de matériel et de logiciels, comme illustré à la figure 1.1. Un ordinateur sans logiciels n'est que de la ferraille plutôt chère. Le système d'exploitation est donc une composante logicielle très importante.

Le matériel d'un système informatique est composé de processeurs qui exécutent les instructions, de la mémoire centrale qui contient les données et les instructions à exécuter, de la mémoire secondaire qui sauvegarde les informations, et des périphériques d'Entrées/Sorties (clavier, souris, écran, modem, etc.) pour introduire ou récupérer des informations. Les logiciels sont à leur tour divisés en **programmes système** qui font fonctionner l'ordinateur : le **système d'exploitation** et les utilitaires (compilateurs, éditeurs, interpréteurs de commandes, etc.) et en **programmes d'application** qui résolvent des problèmes spécifiques des utilisateurs.

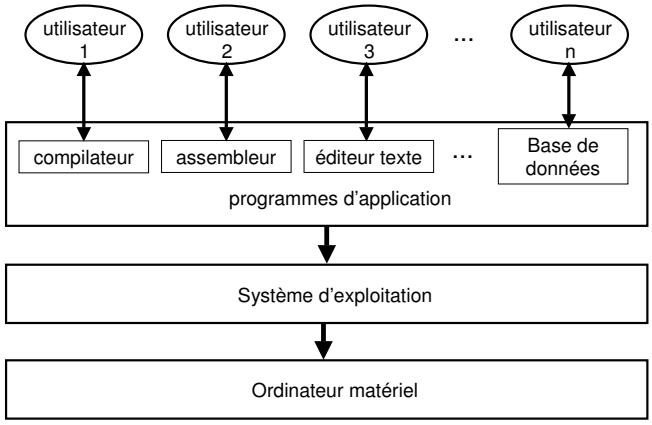

FIG. 1.1 – Les couches d'un système informatique.

1.1.1 Le système d'exploitation

Le **système d'exploitation** gère et contrôle les composants de l'ordinateur. Il fournit une base appelée **machine virtuelle**, sur laquelle seront construits les programmes d'application et les utilitaires au moyen des **services** ou **appels système**. Le but d'un système d'exploitation consiste donc à *développer des applications sans se soucier des détails de fonctionnement et de gestion du matériel*. Ainsi, par exemple, on peut effectuer la lecture d'un fichier par un simple appel système : `read(fd, buf, 255);` et peu importe que le fichier en question se trouve dans un disque magnétique, un DVD ou un disque en RAM.

Les fonctions d'un système d'exploitation

Les systèmes d'exploitation modernes sont constitués de centaines de milliers, voire de millions de lignes de code et requièrent une grande quantité d'homme-années de travail. Ils ont comme fonctions principales :

- Chargement et lancement des programmes
- Gestion des processeurs, de la mémoire, des périphériques
- Gestion des processus (programmes en cours d'exécution) et des fichiers
- Protection contre les erreurs et la détection des erreurs, etc.

Exemples de systèmes d'exploitation

Il y a plusieurs types de systèmes d'exploitation actuellement :
- MS-DOS, Windows
- OS/2, Mac-OS
- Unix (AIX, Xenix, Ultrix, Solaris, etc.)
- Linux

Il y a aussi des systèmes d'exploitation éducatifs, comme Minix[1] crée par Andrew S. Tanembaum ou des simulateurs de systèmes d'exploitation comme Nachos[2].

1.2 Évolution des systèmes d'exploitation

Exploitation porte ouverte : 1945-1955

Les machines à calculer étaient construites au moyen de tubes électroniques. Ces machines étaient énormes, coûteuses, très peu fiables et beaucoup moins rapides car le temps de cycle se mesurait en secondes. Elles n'avaient pas de systèmes d'exploitation. Les programmes étaient écrits directement en langage machine : ils étaient chargés en mémoire, exécutés et mis au point à partir d'un pupitre de commande. Le mode de fonctionnement consistait à réserver une tranche de temps pour chaque programmeur (un seul programme en mémoire). Au début de 1950, la procédure s'est améliorée grâce à l'introduction de cartes perforées. Les problèmes évidents étaient dus au mode d'exploitation : peu économique, le matériel coûteux, la longue durée d'entrée des programmes et une mise au point pénible.

Traitement par lots : 1955-1965

Les machines étaient construites au moyen de transistors et dotées d'unités de bandes magnétiques. Elles étaient plus fiables mais toujours énormes (enfermées dans des salles climatisées). Les programmes étaient écrits en Fortran ou en assembleur sur des cartes perforées. Le mode de fonctionnement utilisé, montré à la figure 1.2, était le traitement par lots qui consiste à :
- Transférer les travaux sur une bande magnétique
- Monter la bande sur le lecteur de bandes

[1] http ://www.cs.vu.nl/~ ast/minix.html
[2] http ://www.cs.washington.edu/homes/tom/nachos/

– Charger en mémoire un programme spécial (l'ancêtre des systèmes d'exploitation) qui lit puis exécute, l'un à la suite de l'autre, les programmes de la bande. Les résultats sont récupérés sur une autre bande, à la fin de l'exécution de tout le lot.
– Imprimer les résultats.

Ce mode d'exploitation nécessitait deux machines dont la plus puissante était réservée aux calculs et l'autre, moins chère, s'occupait des périphériques lents. Le problème est que le processeur restait inutilisé pendant les

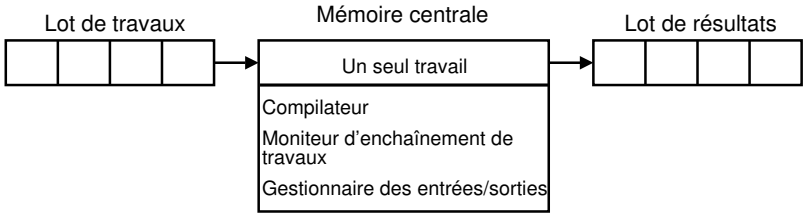

FIG. 1.2 – Traitement par lots et organisation de la mémoire.

opérations d'entrée et sortie (E/S), comme on le voit sur la figure 1.3. Le temps d'attente des résultats était trop long et en plus il n'y avait pas d'interaction avec l'utilisateur.

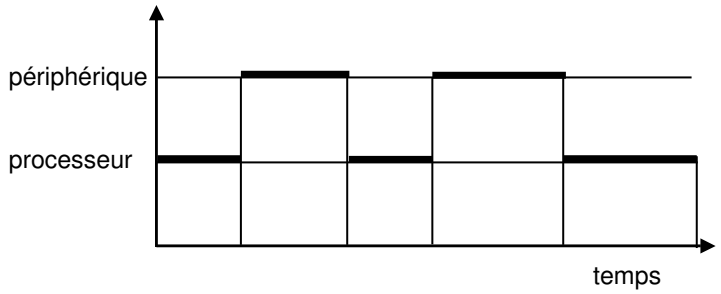

FIG. 1.3 – Utilisation du processeur et des périphériques.

Multiprogrammation et traitement par lots : 1965-1980

L'introduction des circuits intégrés dans la construction des machines a permis d'offrir un meilleur rapport coût/performance. L'arrivée sur le marché des unités de disques, qui offrent l'accès aléatoire et des capacités de

stockage importantes, a contribué à une série de développements des systèmes, notamment la possibilité de transférer les travaux vers le disque dès leur arrivée dans la salle machine. Cette technique s'appelle le *spool* (*Simultaneous Peripheral Operation On Line*) et est également utilisée pour les sorties. Cette notion de *spooling* ou traitement par lots subsiste dans les systèmes d'exploitation modernes.

Tous les travaux résidants sur le disque en attente d'exécution sont conservés dans le *pool* des travaux en entrée. La mémoire est organisée en un ensemble de n partitions (figure 1.4). Chaque partition peut contenir au plus un travail. Le système d'exploitation réside aussi dans une partition. S'il y a une partition libre et des travaux dans le *pool* d'entrée, le système

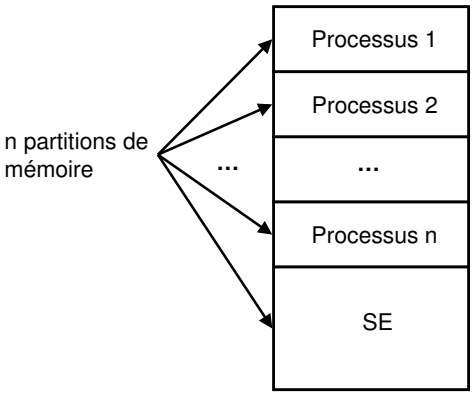

FIG. 1.4 – Partitions de mémoire.

d'exploitation choisit un travail puis lance son chargement en mémoire. Il conserve en mémoire plusieurs travaux et gère le partage du processeur entre les différents travaux chargés en mémoire (la multiprogrammation). Le processeur est alloué à un travail jusqu'à ce qu'il demande une E/S (premier arrivé, premier servi). Lorsqu'un travail demande une E/S (en attente de la fin d'une E/S), le processeur est alloué à un autre travail en mémoire (le suivant). A la fin d'une E/S, une interruption se produit et le système d'exploitation reprend le contrôle pour traiter l'interruption et lancer ou poursuivre l'exécution d'un travail. Dès qu'un travail se termine, le système d'exploitation peut lancer le chargement, à partir du disque, d'un nouveau travail dans la partition qui vient de se libérer.

Supposons trois travaux **A**, **B** et **C**. Dans un système multiprogrammé, trois activités peuvent être donc menées en parallèle : 1. Le chargement du travail **C** en mémoire, 2. L'exécution du travail **B** et 3. L'édition des résultats

du travail **A**.

La multiprogrammation nécessite des circuits de contrôle pour protéger chaque travail contre les intrusions et les erreurs des autres. Les ordinateurs de cette époque possédaient ces circuits.

▶ **Exemple 1.** Considérons les travaux **A**, **B** et **C** en mémoire, qui sont montrés sur la figure 1.5. L'exécution de **A** est entamée en premier puis lorsqu'il

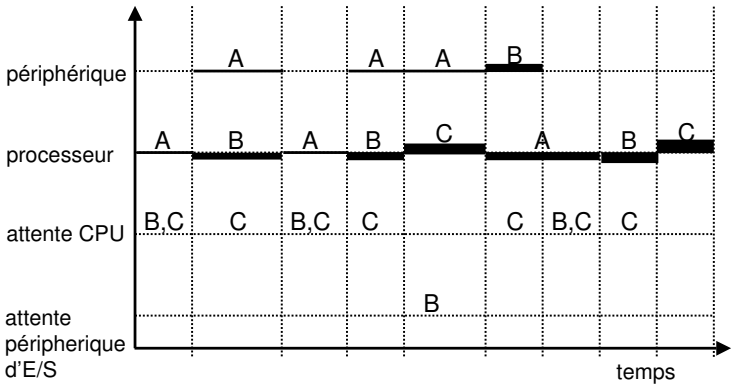

FIG. 1.5 – Commutation du processeur.

demande une Entrée/Sortie (E/S), le processeur commute sur **B**. A la fin de l'E/S demandée par **A**, le processeur suspend l'exécution de **B** pour commuter sur **A**. On suppose qu'**A** est prioritaire. Après un certain temps de calcul, **A** demande de nouveau une E/S ; ce qui provoque la commutation du processeur sur **B**. Durant l'exécution de l'E/S de **A**, le travail **B** demande une E/S. Il se met donc en attente car le périphérique est occupé. Le processeur commute alors sur le travail **C**. Lorsque l'exécution de l'E/S demandée par **A** s'est terminée, le processeur commute sur **A** et le traitement de la demande d'E/S du travail **B** est entamé par le périphérique d'E/S.

Multiprogrammation et partage de temps : 1965-1980

Le désir d'un temps de réponse plus rapide et d'interactivité de l'exploitation a introduit la technique de partage de temps (systèmes temps partagé ou multi-utilisateurs) : plusieurs utilisateurs peuvent se connecter à la machine par l'intermédiaire de leurs terminaux et travailler en même temps.

Le processeur est alloué, à tour de rôle, pendant un certain temps à chacun des travaux en attente d'exécution. Au bout de ce temps, même si le travail en cours ne s'est pas terminé, son exécution est suspendue. Le processeur est alloué à un autre travail. Si plusieurs utilisateurs lancent à partir de leurs terminaux leurs programmes simultanément, ce mode d'exploitation donne l'impression que les programmes s'exécutent en parallèle. Le cas de trois travaux A, B et C est montré sur la figure 1.6. Les temps de réponse pour chaque utilisateur sont acceptables.

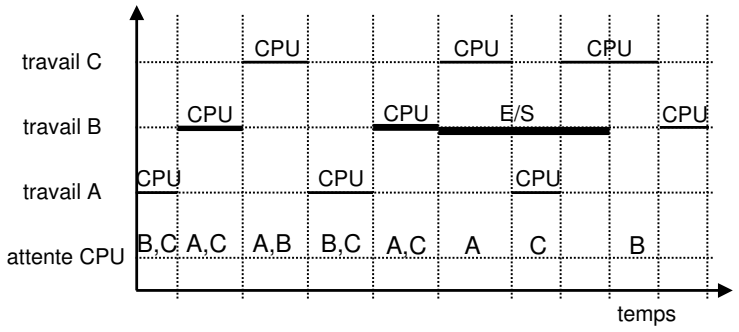

FIG. 1.6 – Temps partagé.

Systèmes d'exploitation d'ordinateurs personnels

Ces systèmes d'exploitation mono-utilisateur ne mettaient pas l'accent sur l'usage du processeur, ni sur la protection. Leur but était de fournir une interface conviviale et une rapidité de réaction. Ensuite, les fonctionnalités des gros systèmes ont été peu à peu transférées vers les microordinateurs.

Exploitation en réseau

Les réseaux d'ordinateurs personnels qui fonctionnent sous des systèmes d'exploitation en réseau permettent de se connecter sur une machine distante et de transférer des fichiers d'une machine à une autre. Ces systèmes d'exploitation requièrent une interface réseau, un logiciel de contrôle de bas niveau, ainsi que des programmes qui permettent une connexion distante et un accès aux fichiers qui se trouvent sur les différentes machines.

Exploitation en distribué

Les réseaux d'ordinateurs qui fonctionnent sous des systèmes d'exploitation distribués apparaissent aux yeux des utilisateurs comme une machine monoprocesseur, même lorsque ce n'est pas le cas. Le système d'exploitation distribué gère et contrôle l'ensemble des composants de tous les ordinateurs connectés (les processeurs, les mémoires, les disques, etc.).

Systèmes multiprocesseurs

Ces systèmes sont composés de plusieurs processeurs reliés au bus de l'ordinateur. Ils se caractérisent par leur capacité de traitement et leur fiabilité : la panne d'un processeur n'arrêtera pas le système

Système d'exploitation temps réel

Ce sont des systèmes spécialisés dans la conduite d'appareillage industriel ou dans la commande de processus où le temps joue un rôle critique (des contraintes temporelles strictes ou souples à respecter).

1.3 Interactions utilisateur/système

Pour un utilisateur, le système d'exploitation apparaît comme un ensemble de procédures, trop complexes pour qu'il les écrive lui-même. Les bibliothèques des **appels système** sont alors des procédures mises à la disposition des programmeurs. Ainsi un programme *C/C++* peut utiliser des appels système d'Unix/Linux comme `open()`, `write()` et `read()` pour effectuer des Entrées/Sorties de bas niveau.

L'**interpréteur de commandes** constitue une interface utilisateur/système. Il est disponible dans tous les systèmes. Il est lancé dès la connexion au système et invite l'utilisateur à introduire une commande. L'**interpréteur de commandes** récupère puis exécute la commande par combinaison d'appels système et d'outils (compilateurs, éditeurs de lien, etc.). Il affiche les résultats ou les erreurs, puis se met en attente de la commande suivante. Par exemple, la commande de l'interpréteur (shell) d'Unix suivante permet d'afficher à l'écran le contenu du fichier appelé `archive` : `cat archive`[3]

L'introduction du graphisme dans les interfaces utilisateur a révolutionné le monde de l'informatique (attendons celle du son !). L'interface

[3] La commande équivalente sous MS-DOS est : `type archive`.

graphique a été rendue populaire par le Macintosh de Apple. Elle est maintenant proposée pour la plupart des machines.

1.4 Appels système

En général, les processeurs ont deux modes de fonctionnement :
- Le **mode superviseur** (noyau, privilégié ou maître) : pour le système d'exploitation, où toutes les instructions sont autorisées.
- Le **mode utilisateur** (esclave) : pour les programmes des utilisateurs et les utilitaires, où certaines instructions ne sont pas permises.

Ces modes de fonctionnement assurent la protection du système d'exploitation contre les intrusions et les erreurs. Ce n'est pas le cas des systèmes mono-utilisateur comme MS-DOS ou MacOS qui ont un seul mode de fonctionnement : le mode utilisateur. Ils ne sont pas protégés et donc peu fiables.

Un **appel système** consiste en une interruption logicielle (instruction `TRAP`) qui a pour rôle d'activer le système d'exploitation. Il a pour but : changer de mode d'exécution pour passer du mode utilisateur au mode maître, récupérer les paramètres et vérifier la validité de l'appel, de lancer l'exécution de la fonction demandée, de récupérer la (les) valeur(s) de retour et de retourner au programme appelant avec retour au mode utilisateur.

La table 1.1 énumère quelques appels système du système Unix conformes à la norme Posix[4]. Les appels système manipulent divers objets gérés par le système d'exploitation. Les processus et les fichiers sont les plus importants de ces objets.

1.4.1 Les processus

Un **processus** est un programme en cours d'exécution. Il est composé d'un **programme exécutable** (code), un **compteur ordinal** (**co**), un ensemble de données, une pile d'exécution et autres registres et informations nécessaires à l'exécution.

Les appels systèmes permettent notamment la création et l'arrêt des processus. Un processus peut créer un ou plusieurs processus fils qui, à leur tour, peuvent créer des processus fils dans une structure arborescente. Les processus peuvent se synchroniser et communiquer entre eux. Actuellement, on utilise de plus en plus le concept de **processus légers** (on parlera

[4] Posix est un ensemble de procédures standard d'Unix qui seront vues toute au long du texte.

Appel système	Description
fork	Créer un processus
waitpid	Attendre la terminaison d'un processus
wait	
execve	Exécuter un autre programme
exit	Terminer l'exécution
open	Créer ou ouvrir un fichier
close	Fermer un fichier
read	Lecture de données
write	Écriture de données
lseek	Pointeur dans un fichier
stat	Obtenir l'état des attributs
mkdir	Créer un répertoire
rmdir	Éliminer un répertoire
link	Liens vers un fichier
unlink	Éliminer un fichier
mount	Monter un système de fichiers
umount	Démonter un système de fichiers
chdir	Changer de répertoire
chmod	Changer les permissions d'accès
kill	Signaux
time	Obtenir le temps

TAB. 1.1 – Quelques appels système Posix.

des processus et des processus légers dans le Chapitre **??**). Les processus légers sont un moyen de raffiner et de diviser le travail normalement associé à un processus.

Lorsqu'on lance une application, le processus qui est créé par le système d'exploitation exécute les instructions du programme en mode utilisateur. À certains moments, on fait des appels système qui requièrent l'exécution d'instructions (et l'accès à des données) qui appartiennent au système d'exploitation. Pour ce faire, le même processus passe alors au mode superviseur. Il est à noter qu'il s'agit bien du même processus, même s'il exécute des instructions du système d'exploitation. On peut considérer que des routines du système d'exploitation sont exécutées au nom du processsus en question. La figure 1.7 illustre ce principe.

La figure 1.8 illustre de manière plus détaillée ce qui se produit typiquement lors d'un appel système. Dans cet exemple, une application exé-

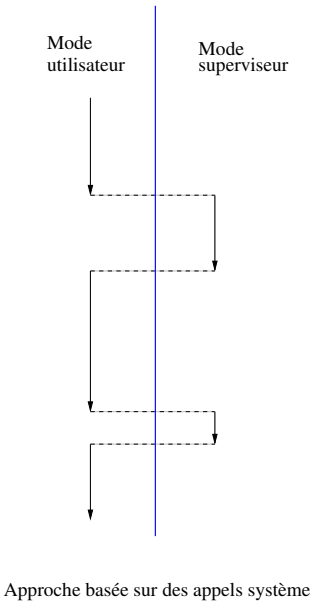

FIG. 1.7 – Alternance de modes d'exécution d'un processus

cute `fork()`. Il s'agit ici d'une fonction d'une librairie qui a été compilée avec l'application. Cette fonction prépare le terrain pour exécuter le véritable appel système, c'est-à-dire une instruction assembleur TRAP qui prend comme paramètre un pointeur dont la valeur est ici indiquée par SYS_FORK. Essentiellement, cette instruction met d'abord le processeur en mode superviseur, puis cherche dans une table (qui a été créé lors du chargement du système d'exploitation), à la position indiquée par SYS_FORK, l'adresse de la routine qui doit être exécutée. Puis le compteur ordinal se positionne à cette adresse et continue l'exécution du processus, cette fois-ci avec des instructions qui appartiennent au système d'exploitation. Lorsque le service demandé est terminé, on revient à l'application, tout en rétablissant le mode utilisateur.

1.4.2 Les fichiers

Un **fichier** est un ensemble de données enregistrées de façon à être lues et traitées par ordinateur. Les fichiers peuvent être regroupés dans des répertoires. Un **répertoire** peut contenir soit des fichiers, soit d'autres répertoires dans une structure arborescente.

L'accès aux fichiers se fait en spécifiant un **chemin d'accès**, c'est-à dire la

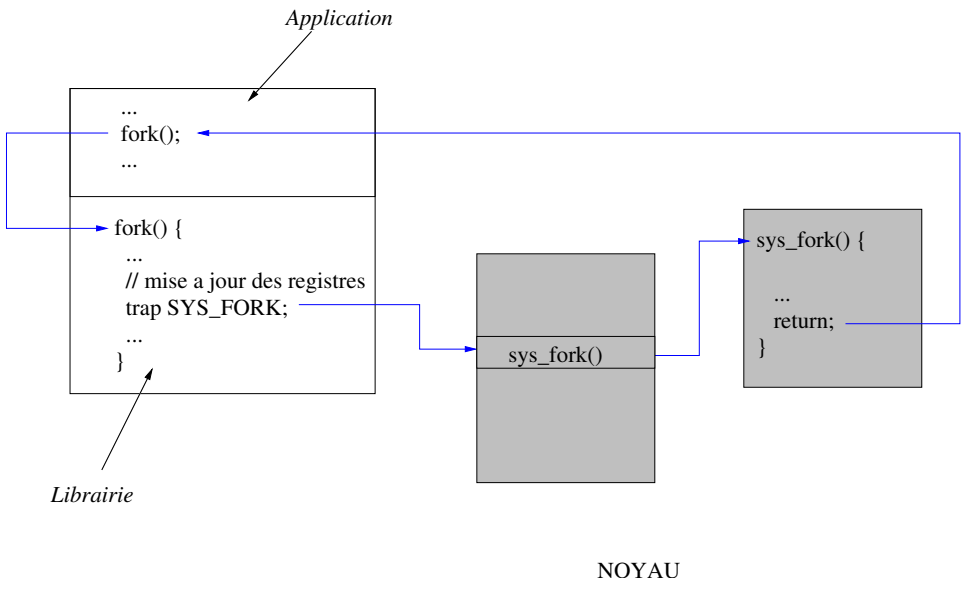

FIG. 1.8 – Étapes d'un appel système

liste des répertoires à traverser pour accéder au fichier. Un chemin d'accès est **absolu** si le point de départ est le répertoire racine. Un chemin d'accès est **relatif** si le point de départ est le répertoire courant. Les appels système permettent de créer les fichiers et les répertoires, ainsi que de les supprimer, de les ouvrir, de les lire et de les modifier.

1.5 Structure d'un système d'exploitation

1.5.1 Structure en couches

Le système d'exploitation est structuré en couches. Chaque couche utilise les fonctions des couches inférieures. La principale difficulté est la définition des différentes couches. Par exemple, on peut l'organiser en cinq couches, comme montré sur la figure 1.9 :

- Au plus bas niveau on trouve le **noyau**, l'interface entre le matériel et le logiciel. Il se charge, en utilisant les fonctions fournies par le matériel, de gérer la UCT, les interruptions et les processus (la communication et la synchronisation). Il doit entièrement résider en mémoire.
- Au second niveau, on trouve le gestionnaire de la mémoire qui se charge du partage de la mémoire entre les processus en attente d'exé-

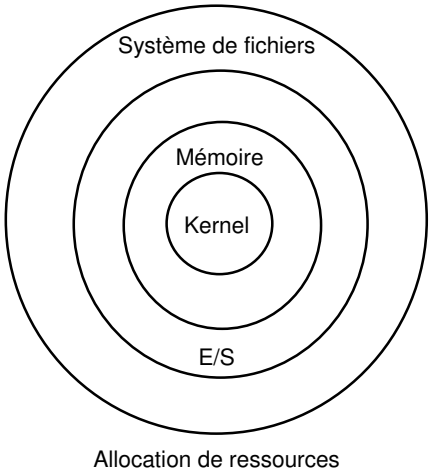

FIG. 1.9 – Structure en couches.

cution.
- Au troisième niveau, on a le module de gestion des entrées/sorties qui se charge de gérer tous les périphériques (clavier, écran, disques, imprimantes, etc.).
- Au quatrième niveau, on trouve le gestionnaire de fichiers qui se charge de la gestion de l'espace du disque, de la manipulation des fichiers tout en assurant l'intégrité des données, la protection des fichiers, etc.
- Au cinquième niveau, on a le module d'allocation de ressources qui se charge d'assurer une bonne utilisation des ressources ; de comptabiliser et de fournir des statistiques sur l'exploitation des ressources principales ; de créer de nouveaux processus et leur attribuer un niveau de priorité : de permettre à chaque processus existant dans le système d'obtenir les ressources nécessaires dans des limites de temps raisonnables ; d'exclure mutuellement les processus qui demandent une ressource non partageable et d'éviter les situations de blocage.

1.5.2 Structure monolithique

Les systèmes d'exploitation sous une **structure monolithique** (figure 1.10) sont un ensemble de procédures de —presque— le même niveau : une **procédure principale** qui appelle la **procédure de service** requise, des procédures de service qui exécutent les **appels système** et un ensemble de

procédures utilitaires qui assistent les procédures de service, par exemple la recherche de données des programmes utilisateur. Unix et Linux sont des

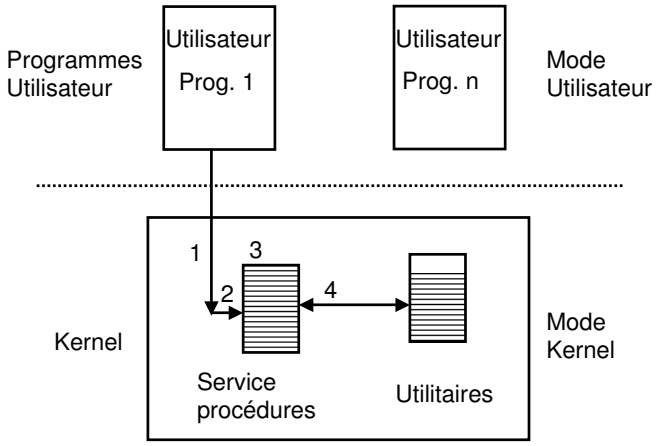

FIG. 1.10 – Structure monolithique. (1) Appel système (Mode utilisateur $\mapsto$ Mode kernel). (2) Vérification de paramètres. (3) Appel de la procédure de service. (4) Procédure de service appel utilitaires, et puis retourner au mode utilisateur.

exemples de systèmes monolithiques.

1.5.3 Micro-kernel

Une architecture plus moderne que celle monolitique est l'architecture **micro-kernel** (voir la figure 1.11) utilisée en MACH[5]/ HURD[6], Minix et NT. L'attribut principal qui distingue les micro-kernels des kernels monolitiques est l'implantation de leurs architectures respectives en mode superviseur (*kernel mode*) et en mode usager (*user mode*). L'architecture monolitique met en œuvre tous les services du Système d'exploitation (controleurs de dispositifs, mémoire virtuelle, système de fichiers, réseaux, etc) dans le domaine du mode superviseur de l'UCT. Par contre, l'architecture micro-kernel fait une division entre les services du Système d'exploitation, en les divisant en « haut-niveau » implantés dans le domaine de l'utilisateur et « bas-niveau » implantés dans l'espace du mode superviseur.

[5]http ://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html
[6]http ://www.gnu.ai.mit.edu/software/hurd/hurd.html

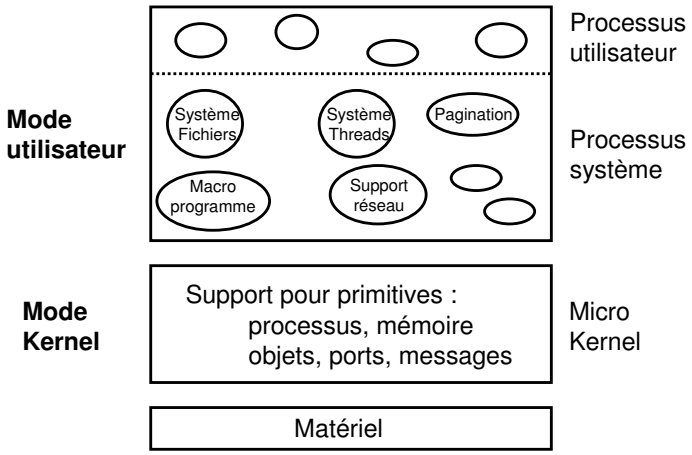

FIG. 1.11 – Structure de micro-kernel.

1.5.4 Le modèle client/serveur

Dans le modèle **client/serveur** Le système d'exploitation est composé d'un noyau et d'un ensemble de serveurs. Le noyau gère la communication entre les **clients** et les serveurs. Les clients sont les demandeurs de services. Par exemple, pour demander un service, comme la lecture d'un bloc d'un fichier, un processus utilisateur (aussi appelé processus client) envoie une requête à un processus serveur qui effectue le travail et renvoie une réponse. Les **serveurs** s'exécutent en mode utilisateur. Ils ne peuvent donc pas accéder directement au matériel. Par conséquent, une erreur ou un bogue dans le serveur de fichiers, par exemple, n'affectera pas, en général, l'ensemble de la machine. Les dégâts se limitent au serveur.

1.5.5 Machines virtuelles

Le cœur du système se charge de la multiprogrammation en fournissant à la couche au dessus plusieurs machines virtuelles. Des systèmes d'exploitation différents peuvent tourner au dessus d'une machine virtuelle.

Exemples de machines virtuelles :

- **IBM VM** : offre à chaque usager sa propre machine virtuelle monotâche. Les machines virtuelles étaient planifiées avec du temps partagé.
- **Java** : les programmes compilés en Java tournent sur une machine virtuelle (JVM).

- **VMWare**[7] : sur PC, exécute en même temps des sessions Windows, Linux, OS/2, etc.
- **Nachos** : SE qui s'exécute dans une machine virtuelle MIPS, qui tourne sur Unix.

[7] http ://www.vmware.com

Sugestions de lecture

Référence : Silverwschatz A., Galvin P., Gagné G., *Applied Operating System Concepts*, Wiley, 2003.

Chapître 1

Section 1.1 What is an operating system ?

Section 1.2 Batch systems : une attention particulière au concept de multiprogrammation.

Section 1.3 Time-Sharing systems : Bien saisir la différence entre multiprogrammation et temps partagé. Bien saisir la complexité des systèmes à temps partagé.

Chapître 2

Section 2.1 Computer-System Operation.

Section 2.2 I/O Structure : Le concept d'interruption est très important. En ce qui concerne le DMA, il est bon de savoir ce que c'est.

Section 2.5 Hardware protection : Important, avec une attention toute particulière aux modes utilisateur et superviseur.

Section 2.6 General system architecture : Le concept d'appel système est très important.

Chapître 3

Section 3.1 Systems components : Bien comprendre le concept de processus.

Section 3.3 System Calls : Très important.

1.6 Exercices

1. Indiquez si les éléments logiciels suivants se situent parmi la catégorie des programmes d'application, des programmes système, ou des programmes de systèmes d'exploitation.
   (a) Le programme `more`
   (b) L'éditeur de texte `emacs`
   (c) La librairie de routines X-Window (Xlib)
   (d) Un script de shell
   (e) Le traitement de texte Ms-word
   (f) La fonction `read`
2. L'ordinateur personnel compatible IBM fournit une interface aux périphériques en utilisant des routines du BIOS (Basic Input and Output System) qui est stockée en mémoire morte (ROM). Quel avantage cette interface fournit-elle aux développeurs d'applications pour la plate-forme Intel 80x86 ?
3. Dans un environnement multiprogrammé, y-a-t-il partage des ressources entre les processus d'un même utilisateur ? Expliquez.
4. Le degré de multiprogrammation représente le nombre maximal de processus qu'un système uniprocesseur peut manipuler à tout moment. Expliquez quelques facteurs matériels et logiciels qui pourraient avoir une certaine influence sur le degré de multiprogrammation d'un système.
5. Décrivez le facteur principal qui sert de mesure du temps de réponse pour un système à soumission de lots et pour un système en temps partagé.
6. Expliquez la différence fondamentale qui existe entre un système d'exploitation multiprogrammé et un système d'exploitation réseau.
7. Une fenêtre (*window*) représente un certain niveau d'abstraction de l'écran car les applications interagissent avec celle-ci et non pas directement à l'écran. Décrivez trois prototypes de fonctions qu'une application pourrait utiliser pour interagir avec une fenêtre d'écran.
8. Quelle différence existe-il entre un système temps réel et une application en temps réel ? Donnez un exemple pour chaque cas.
9. Expliquez la raison principale de l'utilisation de deux modes (usager, système) d'opération dans les systèmes d'exploitation modernes.

10. Quel est le rôle d'un système d'exploitation ? Les interpréteurs de commandes et les compilateurs font-ils parties du système d'exploitation ?
11. Qu'est ce qu'un système multiprogrammé ? Un système de traitement par lots ? Un système en temps partagé ?
12. Dans le système Unix, les véritables appels système sont effectués à partir :
    (a) D'un programme utilisateur.
    (b) D'une commande shell.
    (c) D'une procédure de la bibliothèque standard.
    (d) Sont-ils exécutés en mode superviseur ou en mode utilisateur ?
13. Comment sont organisés les fichiers dans le système Unix ? Un utilisateur peut-il accéder à un fichier d'un autre utilisateur ? Si oui, comment ?
14. Est-ce qu'un Système d'Exploitation multitâche est nécessairement multiusager ? Et pour l'inverse ? Expliquez.