CENTRE NATIONAL D'ORIENTATION ET DE PREPARATION AUX CONCOURS D'ENTREE DANS LES GRANDES ECOLES ET FACULTES DU CAMEROUN

Préparation au Concours d'Entrée en Troisième Année de l'ENSP et FGI

Support de Cours

SYSTÈME D'EXPLOITATION

Avec Intelligentsia Corporation, Il suffit d'y croire !!!

☎ 698 222 277 / 671 839 797
fb : Intelligentsia Corporation
email : contact@intelligentsia-corporation.com

" Vous n'êtes pas un passager sur le train de la vie, vous êtes l'ingénieur. "

-- Elly Roselle --

Instructions :

Il est recommandé à chaque étudiant de traiter les exercices de ce recueil (du moins ceux concernés par la séance) avant chaque séance car ***le temps ne joue pas en notre faveur.***

I. INTRODUCTION

Un système d'exploitation est un ensemble de composants logiciels et API (Application Programming Interface) permettant aux utilisateurs d'un ordinateur ou de tout autre matériel doté d'un tel système d'avoir une vue simplifié de l'utilisation de l'outil avec une multitude d'options de gestion et de contrôle déjà automatisées. Pour ce faire, elle gère et contrôle les composants matériels de l'appareil et offre une abstraction ce dernier (machine virtuelle) à travers une IHM (Interface Homme Machine) simplifiée.

II. LES FONCTIONS D'UN SYSTEME D'EXPLOITATION

Un système d'exploitation effectue fondamentalement trois tâches indépendantes : il permet de charger les programmes les uns après les autres, il émule une machine virtuelle et il gère les ressources. Précisons chacune de ces tâches.

- *Chargement des programmes*

Les premiers micro-ordinateurs étaient fournis sans système d'exploitation. Les tous premiers micro-ordinateurs n'avaient qu'un seul programme : un interpréteur du langage BASIC qui était contenu en mémoire ROM. Lors de l'apparition des lecteurs de cassettes puis, de façon plus fiable, des lecteurs de disquettes, cela commença à changer : si une disquette exécutable était placée dans le lecteur de disquettes, ce programme était exécuté (il fallait éventuellement ensuite remplacer cette disquette par une disquette de données), sinon l'interpréteur BASIC reprenait la main

Avec cette façon de faire, chaque changement de programme exigeait le redémarrage du micro- ordinateur avec la disquette du programme désiré dans le lecteur de disquettes. C'était le cas en particulier de l'Apple II

Les micro-ordinateurs furent ensuite, en option, fournis avec un système d'exploitation. Celui- ci, contenu sur disquette ou en mémoire RAM, affichait une invite à l'écran. On pouvait alors remplacer la disquette système de démarrage par une disquette contenant le

programme dé- siré : en écrivant le nom du programme sur la ligne de commande et en appuyant sur la touche Retour, le programme était chargé et exécuté. À la fin de l'exécution de ce programme, on pouvait charger un nouveau programme, sans devoir redémarrer le système. Ceci permet, par exemple, d'écrire un texte avec un traitement de texte puis d'appeler un autre programme pour l'imprimer

- *Le système d'exploitation en tant que machine virtuelle*

La gestion d'un système informatique donné, par exemple l'IBM-PC, se fait a priori en lan- gage machine. Ceci est primaire et lourd à gérer pour la plupart des ordinateurs, en particulier en ce qui concerne les entrées-sorties. Bien peu de programmes seraient développés si chaque programmeur devait connaître le fonctionnement, par exemple, de tel ou tel disque dur et toutes les erreurs qui peuvent apparaître lors de la lecture d'un bloc. Il a donc fallu trouver un moyen de libérer les programmeurs de la complexité du matériel. Cela consiste à enrober le matériel avec une couche de logiciel qui gère l'ensemble du système. Il faut présenter au programmeur une API (pour l'anglais Application Programming interface, interface de programmation d'application), ce qui correspond à une machine virtuelle plus facile à comprendre et à programmer

Considérons par exemple la programmation des entrées-sorties des disques durs au moyen du contrôleur IDE utilisé sur l'IBM-PC.

Le contrôleur IDE possède 8 commandes principales qui consistent toutes à charger entre 1 et 5 octets dans ses registres. Ces commandes permettent de lire et d'écrire des données, de déplacer le bras du disque, de formater le disque ainsi que d'initialiser, de tester, de restaurer et de recalibrer le contrôleur et les disques.

Les commandes fondamentales sont la lecture et l'écriture, chacune demandant sept para- mètres regroupés dans six octets. Ces paramètres spécifient les éléments tels que l'adresse du premier secteur à lire ou à écrire, le nombre de secteurs à lire ou à écrire, ou si l'on doit essayer de corriger les erreurs. À la fin de l'opération, le contrôleur retourne 14 champs d'état et d'erreur regroupés dans 7 octets.

La plupart des programmeurs ne veulent pas se soucier de la programmation des disques durs. Ils veulent une abstraction simple de haut niveau : considérer par exemple que le disque contient des fichiers nommés ; chaque fichier peut être ouvert en lecture ou en écriture ; il sera lu ou écrit, et finalement fermé. La partie machine virtuelle des systèmes d'exploitation soustrait le matériel au regard du programmeur et offre une vue simple et agréable de fichiers nommés qui peuvent être lus et écrits.

- *Le système d'exploitation en tant que gestionnaire de ressources*

Les ordinateurs modernes se composent de processeurs, de mémoires, d'horloges, de disques, de moniteurs, d'interfaces réseau, d'imprimantes, et d'autres périphériques qui peuvent être utilisés par plusieurs utilisateurs en même temps, le travail du système d'exploitation consiste à ordonner et contrôler l'allocation des processeurs, des mémoires et des périphériques entre les différents programmes qui y font appel.

Imaginez ce qui se produirait si trois programmes qui s'exécutent sur un ordinateur essayaient simultanément d'imprimer leurs résultats sur la même imprimante. Les premières lignes impri- mées pourraient provenir du programme 1, les suivantes du programme 2, puis du programme

3 et ainsi de suite. Il en résulterait le désordre le plus total. Le système d'exploitation peut éviter ce chaos potentiel en transférant les résultats à imprimer dans un fichier tampon sur le disque. Lorsqu'une impression se termine, le système d'exploitation peut alors imprimer un des fichiers se trouvant dans le tampon. Simultanément, un autre programme peut continuer à générer des résultats sans se rendre compte qu'il ne les envoie pas (encore) à l'imprimante.

III. CARACTERISTIQUE D'UN SYSTEME D'EXPLOITATION

- *Systèmes multi-tâches*

La plupart des systèmes d'exploitation modernes permettent l'exécution de plusieurs tâches à la fois : un ordinateur peut, pendant qu'il exécute le programme d'un utilisateur, lire les données d'un disque ou afficher des résultats sur un terminal ou une imprimante. On parle de système d'exploitation multi-tâches ou multi-programmé dans ce cas.

- *Processus*

La notion fondamentale des systèmes d'exploitation multi-tâches est celle de processus. La notion de programme ne suffit pas. Rien n'empêche que le même programme soit exécuté plusieurs fois en même temps : on peut vouloir, par exemple, deux fenêtres emacs ou deux fenêtres gv pour comparer des textes.

Un processus est une instance de programme en train de s'exécuter.

Un processus est représenté par un programme (le code), mais également par ses données et par les paramètres indiquant où il en est, lui permettant ainsi de continuer s'il est interrompu (pile d'exécution, compteur ordinal...). On parle de l'environnement du programme.

Un processus s'appelle aussi tâche (task en anglais) dans le cas de Linux.

Téléphone : 698 222 277 / 671 839 797 **Facebook :** Intelligentsia Corporation **Email :** contact@intelligentsia-coropration.com

- *Temps partagé*

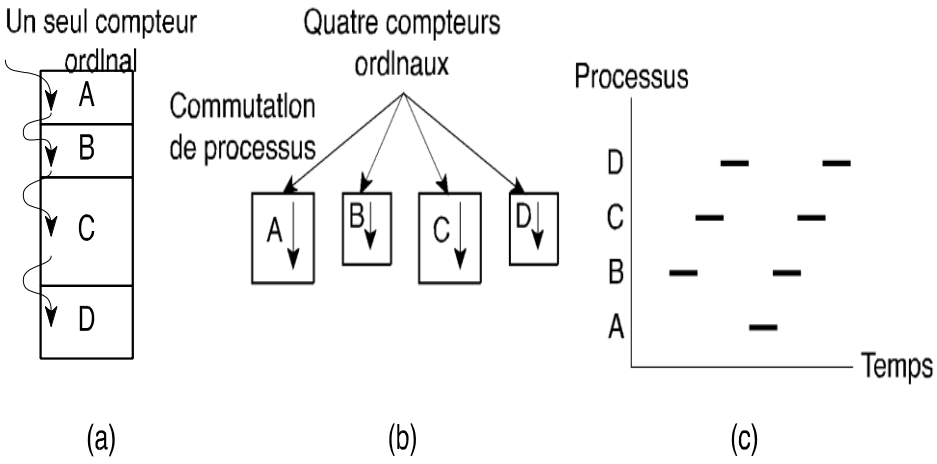
La plupart des systèmes d'exploitation multi-tâches sont implémentés sur un ordinateur ayant un seul micro-processeur. Celui-ci, à un instant donné, n'exécute réellement qu'un seul pro- gramme, mais le système peut le faire passer d'un programme à un autre en exécutant chaque programme pendant quelques dizaines de millisecondes ; ceci donne aux utilisateurs l'impression que tous les programmes sont exécutés en même temps. On parle alors de système à temps partagé.

Certains qualifient de pseudo-parallélisme cette commutation très rapide du processeur d'un programme à un autre, pour la différencier du vrai parallélisme qui se produit au niveau du matériel lorsque le processeur travail en même temps que certains périphériques d'entrée sortie.

- *Abstraction du déroulement*

Conceptuellement, chaque processus a son propre processeur virtuel. Bien sûr, le vrai processeur commute entre plusieurs processus. Mais, pour bien comprendre le système, il est préférable de penser à un ensemble de processus qui s'exécutent en (pseudo-) parallélisme plutôt qu'à l'allocation du processeur entre différents processus. Cette commutation rapide est appelée multi-programmation.

La figure 1.1 montre quatre processus s'exécutant en même temps. La figure (b) présente une abstraction de cette situation. Les quatre programmes deviennent quatre processus indépendants disposant chacun de leur propre contrôle de flux (c'est-à-dire leur compteur ordinal). À la figure (c), on peut constater que, sur un intervalle de temps assez grand, tous les processus ont progressé, mais qu'à un instant donné, il n'y a qu'un seul processus actif.



(a) Multiprogrammation de quatre programmes. (b) Modèle conceptuel de quatre processus séquentiels indépendants. Un seul programme est actif à un instant donné.

Figure1. : Processus

- ***Variables d'environnement***

Comme nous l'avons déjà dit, la donnée du programme est insuffisante pour la détermination d'un processus. Il faut lui indiquer toute une série de variables d'environnement : les fichiers sur lesquels il opère, où en est le compteur ordinal, etc. Ces variables d'environnement sont nécessaires pour deux raisons :

- La première est que deux processus peuvent utiliser le même code (deux fenêtres emacs par exemple) mais les fichiers concernés peuvent être différents, le compteur ordinal ne pas en être au même endroit…

· La seconde est due au caractère multi-tâches, traité par pseudo-parallélisme. Périodique- ment, le système d'exploitation décide d'interrompre un processus en cours afin de démarrer l'exécution d'un autre processus. Lorsqu'un processus est temporairement suspendu de cette manière, il doit pouvoir retrouver plus tard exactement l'état dans lequel il se trouvait au moment de sa suspension. Il faut donc que toutes les informations dont il a besoin soient sauvegardées quelque part pendant sa mise en attente. S'il possède, par exemple, plusieurs fichiers ouverts, les positions dans ces fichiers doivent être mémorisées. La liste des variables d'environnement dépend du système d'exploitation en question, et même de sa version. Elle se trouve dans le descripteur du processus (en anglais process descriptor).

- ***Espace mémoire d'un processus***

Dans de nombreux systèmes d'exploitation, chaque processus possède son propre espace mé- moire, non accessible aux autres processus. On parle de l'espace d'adressage du processus.

- *Incidence sur le traitement des durées*

Puisque le processeur commute entre les processus, la vitesse d'exécution d'un processus ne sera pas uniforme et variera vraisemblablement si les mêmes processus sont exécutés à nouveau. Il ne faut donc pas que les processus fassent une quelconque présomption sur le facteur temps.

Considérons le cas d'un processus d'entrée-sortie qui met en marche le moteur d'un lecteur de disquettes, exécute 1 000 fois une boucle pour que la vitesse de la disquette se stabilise, puis demande la lecture du premier enregistrement. Si le processeur a aussi été alloué à un autre processus pendant l'exécution de la boucle, le processus d'entrée-sortie risque d'être réactivé trop tard, c'est-à-dire après le passage du premier enregistrement devant la tête de lecture.

Lorsqu'un processus a besoin de mesurer des durées avec précision, c'est-à-dire lorsque certains événements doivent absolument se produire au bout de quelques millisecondes, il faut prendre des mesures particulières pour s'en assurer. On utilise alors des minuteurs, comme nous le verrons.

Cependant, la plupart des processus ne sont pas affectés par la multi-programmation du processeur et par les différences de vitesse d'exécution qui existent entre eux.

- *Systèmes multi-utilisateurs*

Un système multi-utilisateurs est capable d'exécuter de façon (pseudo-) concurrente et indépendante des applications appartenant à plusieurs utilisateurs. « Concurrente » signifie que les applications peuvent être actives au même moment et se disputer l'accès à différentes ressources comme le processeur, la mémoire, les disques durs… « Indépendante » signifie que chaque application peut réaliser son travail sans se préoccuper de ce que font les applications des autres utilisateurs.

Un système multi-utilisateurs est nécessairement multi-tâches mais la réciproque est fausse : le système d'exploitation MS-DOS est mono-utilisateur et mono-tâche ; les systèmes MacOS 6.1 et Windows 3.1 sont mono-utilisateurs mais multi-tâches ; Unix et Windows NT sont multi- utilisateurs.

Mise en place

Comme pour les systèmes multi-tâches, la multi-utilisation est émulée en attribuant des laps de temps à chaque utilisateur. Naturellement, le fait de basculer d'une application à l'autre ralentit chacune d'entre elles et affecte le temps de réponse perçu par les utilisateurs.

Mécanismes associés

Lorsqu'ils permettent la multi-utilisation, les systèmes d'exploitation doivent prévoir un certain nombre de mécanismes :

- un mécanisme d'authentification permettant de vérifier l'identité de l'utilisateur ;
- un mécanisme de protection contre les programmes utilisateur erronés, qui pourraient bloquer les autres applications en cours d'exécution sur le système, ou mal intentionnés, qui pourraient perturber ou espionner les activités des autres utilisateurs ;
- un mécanisme de comptabilité pour limiter le volume des ressources allouées à chaque utilisateur.

Utilisateurs

Dans un système multi-utilisateurs, chaque utilisateur possède un espace privé sur la machine : généralement, il possède un certain quota de l'espace disque pour enregistrer ses fichiers, il reçoit des courriers électroniques privés, etc. Le système d'exploitation doit assurer que la partie privée de l'espace d'un utilisateur ne puisse être visible que par son propriétaire. Il doit, en particulier, assurer qu'aucun utilisateur ne puisse utiliser une application du système dans le but de violer l'espace privé d'un autre utilisateur.

Chaque utilisateur est identifié par un numéro unique, appelé l'identifiant de l'utilisateur, ou UID (pour l'anglais User IDentifier). En général, seul un nombre limité de personnes est autorisé à utiliser un système informatique. Lorsque l'un de ces utilisateurs commence une session de travail, le système d'exploitation lui demande un nom d'utilisateur et un mot de passe. Si l'utilisateur ne répond pas par des informations valides, l'accès lui est refusé.

Groupe d'utilisateurs

Pour pouvoir partager de façon sélective le matériel avec d'autres, chaque utilisateur peut être membre d'un ou de plusieurs groupes d'utilisateurs. Un groupe est également identifié par un numéro unique dénommé identifiant de groupe, ou GID (pour l'anglais Group IDenti- fier). Par exemple, chaque fichier est associé à un et un seul groupe. Sous Unix, il est possible par exemple de limiter l'accès en lecture et en écriture au seul possesseur d'un fichier, en lec- ture au groupe, et d'interdire tout accès aux autres utilisateurs.

Super-utilisateur

Un système d'exploitation multi-utilisateurs prévoit un utilisateur particulier appelé super- utilisateur ou superviseur (root en anglais). L'administrateur du système doit se connecter en temps que super-utilisateur pour gérer les comptes des utilisateurs et

réaliser les tâches de maintenance telles que les sauvegardes et les mises à jour des programmes. Le super- utilisateur peut faire pratiquement n'importe quoi dans la mesure où le système d'exploitation ne lui applique jamais les mécanismes de protection, ceux-ci ne concernant que les autres utilisateurs, appelés utilisateurs ordinaires. Le super-utilisateur peut, en particulier, accéder à tous les fichiers du système et interférer sur l'activité de n'importe quel processus en cours d'exécution. Il ne peut pas, en revanche, accéder aux ports d'entrée-sortie qui n'ont pas été prévus par le noyau, comme nous le verrons.

IV. STRUCTURE EXTERNE D'UN SYSTEME D'EXPLOITATION

- *Noyau et utilitaires*

Le système d'exploitation comporte un certain nombre de routines (sous-programmes). Les plus importantes constituent le noyau (kernel en anglais). Celui-ci est chargé en mémoire vive à l'initialisation du système et contient de nombreuses procédures nécessaires au bon fonctionnement du système. Les autres routines, moins critiques, sont appelées des utilitaires.

Le noyau d'un système d'exploitation se compose de quatre parties principales : le gestionnaire de tâches (ou des processus), le gestionnaire de mémoire, le gestionnaire de fichiers et le gestionnaire de périphériques d'entrée-sortie. Il possède également deux parties auxiliaires : le chargeur du système d'exploitation et l'interpréteur de commandes.

- *Le gestionnaire de tâches*

Sur un système à temps partagé, l'une des parties les plus importantes du système d'exploitation est le gestionnaire de tâches (en anglais scheduler) ou ordonnanceur. Sur un système à un seul processeur, il divise le temps en laps de temps (en anglais slices, tranches). Périodiquement, le gestionnaire de tâches décide d'interrompre le processus en cours et de démarrer (ou reprendre) l'exécution d'un autre, soit parce que le premier a épuisé son temps d'allocation du processus soit qu'il est bloqué (en attente d'une donnée d'un des périphériques).

Le contrôle de plusieurs activités parallèles est un travail difficile. C'est pourquoi les concepteurs des systèmes d'exploitation ont constamment, au fil des ans, amélioré le modèle de parallélisme pour le rendre plus simple d'emploi.

Certains systèmes d'exploitation permettent uniquement des processus non préemptifs, ce qui signifie que le gestionnaire des tâches n'est invoqué que lorsqu'un processus cède volontairement le processeur. Mais les processus d'un système multi-utilisateur doivent être préemptifs.

- *Le gestionnaire de mémoire*

La mémoire est une ressource importante qui doit être gérée avec prudence. Le moindre micro- ordinateur a, dès la fin des années 1980, dix fois plus de mémoire que l'IBM 7094, l'ordinateur le plus puissant du début des années soixante. Mais la taille des programmes augmente tout aussi vite que celle des mémoires.

La gestion de la mémoire est du ressort du gestionnaire de mémoire. Celui-ci doit connaître les parties libres et les parties occupées de la mémoire, allouer de la mémoire aux processus qui en ont besoin, récupérer la mémoire utilisée par un processus lorsque celui-ci se termine et traiter le va-et-vient (swapping en anglais, ou pagination) entre le disque et la mémoire principale lorsque cette dernière ne peut pas contenir tous les processus.

- *Le gestionnaire de fichiers*

Comme nous l'avons déjà dit, une des tâches fondamentales du système d'exploitation est de masquer les spécificités des disques et des autres périphériques d'entrée-sortie et d'offrir au programmeur un modèle agréable et facile d'emploi. Ceci se fait à travers la notion de fichier.

- *Le gestionnaire de périphériques*

Le contrôle des périphériques d'entrée-sortie (E/S) de l'ordinateur est l'une des fonctions primordiales d'un système d'exploitation. Ce dernier doit envoyer les commandes aux périphériques, intercepter les interruptions, et traiter les erreurs. Il doit aussi fournir une interface simple et facile d'emploi entre les périphériques et le reste du système qui doit être, dans la mesure du possible, la même pour tous les périphériques, c'est-à-dire indépendante du périphérique utilisé. Le code des entrées-sorties représente une part importante de l'ensemble d'un système d'exploitation.

De nombreux systèmes d'exploitation offrent un niveau d'abstraction qui permet aux utilisateurs de réaliser des entrées-sorties sans entrer dans le détail du matériel. Ce niveau d'abs- traction fait apparaître chaque périphérique comme un fichier spécial, qui permettent de traiter les périphériques d'entrée-sortie comme des fichiers. C'est le

cas d'Unix. Dans ce cas, on appelle fichier régulier tout fichier situé en mémoire de masse.

- *Le chargeur du système d'exploitation*

En général, de nos jours, lorsque l'ordinateur (compatible PC ou Mac) est mis sous tension, il exécute un logiciel appelé BIOS (pour Basic Input Output System) placé à une adresse bien déterminée et contenu en mémoire RAM. Ce logiciel initialise les périphériques, charge un secteur d'un disque, et exécute ce qui y est placé. Lors de la conception d'un système d'exploitation, on place sur ce secteur le chargeur du système d'exploitation ou, plus exactement, le chargeur du chargeur du système d'exploitation (ou pré-chargeur) puisque le contenu d'un secteur est insuffisant pour le chargeur lui-même.

La conception du chargeur et du pré-chargeur est indispensable, même si ceux-ci ne font pas explicitement partie du système d'exploitation.

- *L'interpréteur de commandes*

Le système d'exploitation proprement dit est le code qui permet de définir les appels système. Les programmes système tels que les éditeurs de texte, les compilateurs, les assembleurs, les éditeurs de liens et les interpréteurs de commandes ne font pas partie du système d'exploitation. Cependant l'interpréteur de commandes (shell en anglais) est souvent considéré comme en faisant partie.

Sous sa forme la plus rudimentaire, l'interpréteur de commandes exécute une boucle infinie qui affiche une invite (montrant par là que l'on attend quelque chose), lit le nom du programme saisi par l'utilisateur à ce moment-là et l'exécute.

V. STRUCTURE EXTERNE D'UN SYSTEME D'EXPLOITATION

Après avoir examiné un système d'exploitation de l'extérieur (du point de vue de l'interface présentée à l'utilisateur et au programmeur), nous allons examiner son fonctionnement interne.

- *Les systèmes monolithiques*

Andrew Tannenbaum appelle système monolithique (d'un seul bloc) un système d'exploitation qui est une collection de procédures, chacune pouvant à tout moment appeler n'importe quelle autre procédure, en remarquant que c'est l'organisation (plutôt chaotique) la plus répandue.

Pour construire le code ob jet du système d'exploitation, il faut compiler toutes les procédures, ou les fichiers qui les contiennent, puis les réunir au moyen d'un éditeur de liens. Dans un système monolithique, il n'y a aucun masquage de l'information : chaque procédure est visible de toutes les autres, par opposition aux structures constituées de modules ou d'unités de pro- grammes et dans lesquelles les informations sont locales aux modules et où il existe des points de passage obligés pour accéder aux modules.

MS-DOS est un exemple d'un tel système.

- *Systèmes à mode noyau et utilisateur*

Dans beaucoup de systèmes d'exploitation, il existe deux modes : le mode noyau et le mode utilisateur. Le système d'exploitation démarre en mode noyau, ce qui permet d'initialiser les périphériques et de mettre en place les routines de service pour les appels système, et commute ensuite en mode utilisateur. En mode utilisateur, on ne peut pas avoir accès directement aux périphériques : on doit utiliser ce qu'on appelle des appels système pour avoir accès à ce qui a été prévu par le système : le noyau reçoit cet appel système, vérifie qu'il s'agit d'une demande valable (en particulier du point de vue des droits d'accès), l'exécute, puis renvoie au mode utilisateur. Le mode noyau ne peut être changé que par une compilation du noyau ; même le super-utilisateur agit en mode utilisateur.

Unix et Windows (tout au moins depuis Windows 95) sont de tels systèmes. Ceci explique pourquoi on ne peut pas tout programmer sur un tel système.

Les micro-processeurs modernes aident à la mise en place de tels systèmes. C'est l'origine Aide du mode protégé des micro-processeurs d'Intel depuis le 80286 : il existe plusieurs niveaux de privilèges avec une vérification matérielle, et non plus seulement logicielle, des règles de passage d'un niveau à l'autre.

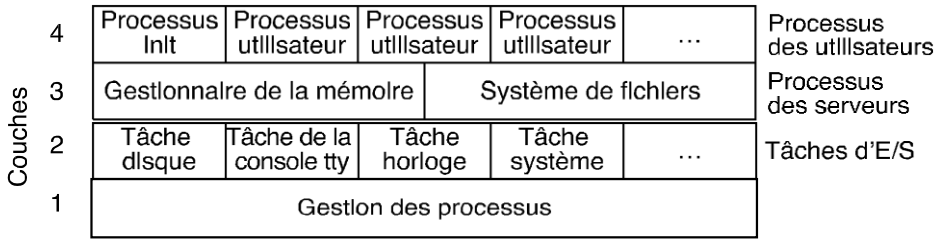
- *Systèmes à couches*

Les systèmes précédents peuvent être considérés comme des systèmes à deux couches et être généralisés en systèmes à plusieurs couches : chaque couche s'appuie sur celle qui lui est immédiatement inférieure.

Le premier système à utiliser cette technique a été le système THE développé au Technische Hogeschool d'Eindhoven (d'où son nom) aux Pays-Bas par Diskstra (1968) et ses élèves. Le système d'exploitation Multics, à l'origine d'Unix, était aussi un système à couches.

Le système d'exploitation Minix de Tanenbaum, schématisé sur la figure 1.2 ([TAN-87], p.100), qui inspira Linux, est un système à quatre couches :

- La couche 1, la plus basse, traite les interruptions et les déroutements (traps en anglais) et fournit aux couches du dessus un modèle constitué de processus séquentiels indépendants qui



MINIX est structuré en quatre couches.

communiquent au moyen de messages. Le code de cette couche a deux fonctions majeures : la première est le traitement des interruptions et des déroutements ; la deuxième est liée au mécanisme des messages. La partie de cette couche qui traite des interruptions est écrite en langage d'assemblage ; les autres fonctions de la couche, ainsi que les couches supérieures, sont écrites en langage C.

La couche 2 contient les pilotes de périphériques (device drivers en anglais), un par type de périphérique (disque, horloge, terminal...). Elle contient de plus une tâche particulière, la tâche système.

Toutes les tâches de la couche 2 et tout le code de la couche 1 ne forment qu'un seul pro- gramme binaire, appelé le noyau (kernel en anglais). Les tâches de la couche 2 sont totalement indépendantes bien qu'elles fassent partie d'un même programme o jet : elles sont sélectionnées

indépendamment les unes des autres et communiquent par envoi de messages. Elles sont regroupées en un seul code binaire pour faciliter l'intégration de Minix à des machines à deux modes.

La couche 3 renferme deux gestionnaires qui fournissent des services aux processus des utilisateurs. Le gestionnaire de mémoire (MM pour l'anglais Memory Manager) traite tous les appels système de Minix, tels que **fork()**, **exec()** et **brk()**, qui concernent la gestion de la mémoire. Le système de fichiers (FS pour l'anglais File System) se charge des appels système du système de fichiers, tels que **read()**, **mount()** et **chdir()**.

La couche 4 contient enfin tous les processus des utilisateurs : interpréteurs de commandes, éditeurs de texte, compilateurs, et programmes écrits par les utilisateurs.

Linux s'inspirera de cette division en couches, bien qu'on n'y trouve officiellement que deux couches : le mode noyau et le mode utilisateur.

- *Systèmes à micro-noyau*

Les systèmes d'exploitation à base de micro-noyau ne possèdent que quelques fonctions, en général quelques primitives de synchronisation, un gestionnaire des tâches simple, et un mécanisme de communication entre processus. Des processus système s'exécutent au-dessus du micro-noyau pour implémenter les autres fonctions d'un système d'exploitation, comme l'allo- cation mémoire, les gestionnaires de périphériques, les gestionnaires d'appels système, etc.

Le système d'exploitation Amoeba de Tanenbaum fut l'un des premiers systèmes à micro- noyau.

Ce type de systèmes d'exploitation promettait beaucoup ; malheureusement ils se sont révélés plus lents que les systèmes monolithiques, du fait du coût des passages de messages entre les différentes couches du système d'exploitation.

Pourtant, les micro-noyaux présentent des avantages théoriques sur les systèmes monolithiques. Ils nécessitent par exemple de la part de leurs concepteurs une approche modulaire, dans la mesure où chaque couche du système est un programme relativement indépendant qui doit interagir avec les autres couches via une interface logicielle propre et bien établie. De plus, un système à base de micro-noyau peut être porté assez aisément sur d'autres architectures dans la mesure où toutes les composantes dépendantes du matériel sont en général localisées dans le code du micro-noyau. Enfin, les systèmes à base de micro-noyau ont tendance à mieux utiliser la mémoire vive que les systèmes monolithiques.

- *Systèmes à modules*

Un module est un fichier objet dont le code peut être lié au noyau (et en être supprimé) en cours d'exécution. Ce code objet est en général constitué d'un ensemble de fonctions qui implémente un système de fichiers, un pilote de périphérique, ou tout autre fonctionnalité de haut niveau d'un système d'exploitation. Le module, contrairement aux couches externes d'un système à base de micro-noyau, ne s'exécute pas dans un processus spécifique. Il est au contraire exécuté en mode noyau au nom du processus courant, comme toute fonction liée statiquement dans le noyau.

La notion de module représente une fonctionnalité du noyau qui offre bon nombre des avantages théoriques d'un micro-noyau sans pénaliser les performances. Parmi les avantages des modules, citons :

- Une approche modulaire : puisque chaque module peut être lié et délié en cours d'exécution du système, les programmeurs ont dû introduire des interfaces

logicielles très claires permet- tant d'accéder aux structures de données gérées par les modules. Cela rend le développement de nouveaux modules plus simple.

- Indépendance vis-à-vis de la plateforme : même s'il doit se baser sur des caractéristiques bien définies du matériel, un module ne dépend pas d'une plateforme particulière. Ainsi, un pilote de disque basé sur le standard SCSI fonctionne aussi bien sur un ordinateur compatible IBM que sur un Alpha.

- Utilisation économique de la mémoire : un module peut être inséré dans le noyau lorsque les fonctionnalités qu'il apporte sont requises et en être supprimé lorsqu'elles ne le sont plus. De plus, ce mécanisme peut être rendu transparent à l'utilisateur puisqu'il peut être réalisé automatiquement par le noyau.

- Aucune perte de performances : une fois inséré dans le noyau, le code d'un module est équivalent au code lié statiquement au noyau. De ce fait, aucun passage de message n'est nécessaire lorsque les fonctions du module sont invoquées. Bien entendu, une petite perte de performance est causée par le chargement et la suppression des modules. Cependant, cette perte est comparable à celle dont sont responsables la création et la destruction du processus d'un système à base de micro-noyau.

- ***Les appels système***

L'interface entre le système d'exploitation et les programmes de l'utilisateur est constituée d'un ensemble d'« instructions étendues » fournies par le système d'exploitation, qualifiées d'appels système.

Les appels système créent, détruisent et utilisent divers objets logiciels gérés par le système d'exploitation, dont les plus importants sont les processus et les fichiers.

- ***Les signaux***

Les processus s'exécutant indépendamment les uns des autres, il s'agit de pseudo-parallélisme. Il faut cependant quelquefois fournir de l'information à un processus. Comment le système d'exploitation procède-t-il ? On a imaginé une méthode analogue à celle des interruptions logicielles pour les micro-processeurs, appelée signal.

Considérons, par exemple, le cas de l'envoi d'un message. Pour empêcher la perte des mes- sages, on convient que le récepteur envoie lui-même un acquittement dès qu'il reçoit une par- tie du message (d'une taille déterminée) ; on envoie à nouveau cette partie si l'acquittement ne parvient pas dans un temps déterminé. Pour mettre en place un tel envoi, on utilisera un processus : il envoie une partie du message, demande à son système d'exploitation de l'avertir lorsqu'un certain temps est écoulé, il vérifie alors qu'il a reçu l'acquittement du message et sinon l'envoie à nouveau.

Lorsque le système d'exploitation envoie un signal à un processus, ce signal provoque la sus- pension temporaire du travail en cours, la sauvegarde des registres dans la pile et l'exécution d'une procédure particulière de traitement du

signal reçu. À la fin de la procédure de traite- ment du signal, le processus est redémarré dans l'état où il se trouvait juste avant la réception du signal.