

**Andrew Tanenbaum**

Université libre d'Amsterdam

# **Systemes d'exploitation**

**3<sup>e</sup> édition**

Coordination de la traduction : Jean-Alain Hernandez, René Joly  
et Michel Dupuy

Traduction :  
Jean-Luc Bourdon  
Philippe Laroque  
Emmanuelle Burr  
Véronique Campillo  
Véronique Warion  
Patrick Fabre

**NOUVEAUX  
HORIZONS**

Le présent ouvrage est la traduction de *Modern Operating Systems, 3<sup>rd</sup> edition*, de Andrew S. Tanenbaum, publié par Pearson Education Inc. publishing as Prentice Hall, Copyright © 2008.

Authorized translation from the English language edition entitled *Modern Operating Systems, 3<sup>rd</sup> edition*, by Andrew Tanenbaum, published by Pearson Education, Inc., publishing as Prentice Hall, Copyright © 2008.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

French language edition published by Pearson Education France, copyright © 2008.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Titre original : *Modern Operating Systems, 3<sup>rd</sup> edition*

ISBN original : 0-13-600663-9

Copyright © 2008 by Andrew Tanenbaum

Tous droits réservés

Mise en pages : TyPAO

**Copyright © 2008 Pearson Education France**

**Tous droits réservés**

Nouveaux Horizons est la branche édition des services culturels du département d'État américain. Notre but est de rendre accessibles les livres d'auteurs américains en Afrique francophone et en Haïti. Pour connaître nos points de vente ou pour toute autre information, consultez notre site : <http://ars-paris.state.gov>.

Distribution Nouveaux Horizons – ARS, Paris, pour l'Afrique francophone et Haïti.

ISBN : 978-2-915236-90-3

4<sup>e</sup> tirage, 2016

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2° et 3° a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

# Table des matières

<b>L'auteur</b> .....	XIX
<b>Préface</b> .....	1
<b>1. Introduction</b> .....	3
1.1 Qu'est-ce qu'un système d'exploitation ? .....	5
1.1.1 Le système d'exploitation comme machine étendue .....	6
1.1.2 Le système d'exploitation comme gestionnaire de ressources .....	8
1.2 L'historique des systèmes d'exploitation .....	9
1.2.1 La première génération (1945-1955) : tubes à vide .....	9
1.2.2 La deuxième génération (1955-1965) : transistors et traitement par lots .....	10
1.2.3 La troisième génération (1965-1980) : circuits intégrés et multiprogrammation .....	12
1.2.4 La quatrième génération (1980-aujourd'hui) : les ordinateurs personnels .....	17
1.3 La structure matérielle d'un ordinateur .....	20
1.3.1 Le processeur .....	21
1.3.2 La mémoire .....	25
1.3.3 Les disques .....	28
1.3.4 Les bandes magnétiques .....	29
1.3.5 Les périphériques d'E/S .....	30
1.3.6 Les bus .....	33
1.3.7 Démarrage de l'ordinateur .....	35
1.4 Le bestiaire des systèmes d'exploitation .....	36
1.4.1 Les systèmes d'exploitation des mainframes .....	36
1.4.2 Les systèmes d'exploitation des serveurs .....	37
1.4.3 Les systèmes d'exploitation des multiprocesseurs .....	37
1.4.4 Les systèmes d'exploitation des PC .....	37

1.4.5	Les systèmes d'exploitation des assistants personnels . . . .	38
1.4.6	Les systèmes d'exploitation embarqués . . . . .	38
1.4.7	Les systèmes d'exploitation des objets communicants . . . .	38
1.4.8	Les systèmes d'exploitation temps réel . . . . .	39
1.4.9	Les systèmes d'exploitation pour smart cards . . . . .	39
1.5	Les concepts de base des systèmes d'exploitation . . . . .	40
1.5.1	Les processus . . . . .	40
1.5.2	Espace d'adressage . . . . .	42
1.5.3	Les fichiers . . . . .	43
1.5.4	Les entrées/sorties . . . . .	46
1.5.5	La sécurité . . . . .	46
1.5.6	Le shell . . . . .	46
1.5.7	L'ontogenèse résume la phylogenèse . . . . .	48
1.6	Les appels système . . . . .	51
1.6.1	Les appels système pour la gestion des processus . . . . .	55
1.6.2	Les appels système pour la gestion des fichiers . . . . .	57
1.6.3	Les appels système pour la gestion des répertoires . . . . .	58
1.6.4	Autres appels système . . . . .	59
1.6.5	L'API Win32 de Windows . . . . .	60
1.7	La structure d'un système d'exploitation . . . . .	63
1.7.1	Les systèmes monolithiques . . . . .	63
1.7.2	Les systèmes en couches . . . . .	64
1.7.3	Les micronoyaux . . . . .	65
1.7.4	Le modèle client-serveur . . . . .	68
1.7.5	Les machines virtuelles . . . . .	68
1.7.6	Les exonoyaux . . . . .	72
1.8	Le monde du langage C . . . . .	72
1.8.1	Le langage C . . . . .	73
1.8.2	Les fichiers d'en-tête . . . . .	73
1.8.3	Développement de gros programmes . . . . .	74
1.8.4	Le modèle d'exécution . . . . .	76
1.9	Les recherches sur les systèmes d'exploitation . . . . .	76
1.10	Organisation de l'ouvrage . . . . .	77
1.11	Le système métrique . . . . .	78
1.12	Résumé . . . . .	79
	Exercices . . . . .	80

<b>2. Processus et threads</b>	<b>85</b>
2.1 Les processus	85
2.1.1 Le modèle de processus	86
2.1.2 La création d'un processus	88
2.1.3 La fin d'un processus	90
2.1.4 La hiérarchie des processus	91
2.1.5 Les états des processus	92
2.1.6 L'implémentation des processus	94
2.1.7 La modélisation de la multiprogrammation	96
2.2 Les threads	98
2.2.1 L'utilisation des threads	98
2.2.2 Le modèle de thread classique	104
2.2.3 Les threads de POSIX	108
2.2.4 L'implémentation de threads dans l'espace utilisateur	110
2.2.5 L'implémentation de threads dans le noyau	113
2.2.6 Les implémentations hybrides	114
2.2.7 Les activations de l'ordonnanceur	115
2.2.8 Les threads spontanés	116
2.2.9 Du code monothread au code multithread	117
2.3 La communication interprocessus	121
2.3.1 Les conditions de concurrence	121
2.3.2 Les sections critiques	123
2.3.3 L'exclusion mutuelle avec attente active	124
2.3.4 Le sommeil et l'activation	129
2.3.5 Les sémaphores	132
2.3.6 Les mutex	135
2.3.7 Les moniteurs	140
2.3.8 L'échange de messages	146
2.3.9 Les barrières	149
2.4 L'ordonnancement	150
2.4.1 Introduction à l'ordonnancement	150
2.4.2 L'ordonnancement sur les systèmes de traitement par lots	157
2.4.3 L'ordonnancement des systèmes interactifs	159
2.4.4 L'ordonnancement des systèmes temps réel	166
2.4.5 La politique et le mécanisme d'ordonnancement	167
2.4.6 L'ordonnancement des threads	168

2.5	Les problèmes classiques .....	170
2.5.1	Le dîner des philosophes .....	170
2.5.2	Le problème des lecteurs et des rédacteurs .....	173
2.6	Les recherches sur les processus et les threads .....	175
2.7	Résumé .....	175
	Exercices .....	176

### **3. La gestion de la mémoire .....** 183

3.1	Abstraction de la mémoire ? .....	184
3.2	Une abstraction de la mémoire : les espaces d'adressage .....	187
3.2.1	La notion d'espace d'adressage .....	187
3.2.2	Le va-et-vient .....	189
3.2.3	Gérer la mémoire libre .....	192
3.3	La mémoire virtuelle .....	195
3.3.1	La pagination .....	196
3.3.2	Les tables des pages .....	200
3.3.3	Accélérer la pagination .....	202
3.3.4	Les tables des pages des grandes mémoires .....	206
3.4	Les algorithmes de remplacements de pages .....	209
3.4.1	L'algorithme optimal de remplacement de page .....	210
3.4.2	L'algorithme de remplacement de la page non récemment utilisée .....	211
3.4.3	L'algorithme de remplacement de page premier entré, premier sorti .....	212
3.4.4	L'algorithme de remplacement de page de la seconde chance .....	213
3.4.5	L'algorithme de remplacement de page de l'horloge .....	214
3.4.6	L'algorithme de remplacement de la page la moins récemment utilisée (LRU) .....	214
3.4.7	La simulation logicielle de l'algorithme LRU .....	216
3.4.8	L'algorithme de remplacement de page « ensemble de travail » .....	217
3.4.9	L'algorithme de remplacement de page WSClock .....	221
3.4.10	Résumé des algorithmes de remplacements de pages .....	223
3.5	La conception des systèmes de pagination .....	224
3.5.1	Les politiques d'allocations locale et globale .....	225
3.5.2	Contrôle du chargement .....	227

3.5.3	La taille des pages . . . . .	228
3.5.4	Séparation des espaces d'instructions et des données . . . .	229
3.5.5	Les pages partagées . . . . .	230
3.5.6	Les bibliothèques partagées . . . . .	232
3.5.7	Les fichiers placés (mappés) en mémoire . . . . .	234
3.5.8	La politique de nettoyage . . . . .	234
3.5.9	L'interface de mémoire virtuelle . . . . .	235
3.6	Les problèmes de l'implantation . . . . .	236
3.6.1	Le rôle du système d'exploitation dans la pagination . . . .	236
3.6.2	Gérer les défauts de pages . . . . .	237
3.6.3	Retourner au début d'une instruction . . . . .	238
3.6.4	Verrouiller des pages en mémoire . . . . .	239
3.6.5	La mémoire auxiliaire . . . . .	239
3.6.6	Séparer la stratégie et le système . . . . .	241
3.7	La segmentation . . . . .	243
3.7.1	Implantation de la segmentation pure . . . . .	247
3.7.2	La segmentation avec pagination : MULTICS . . . . .	248
3.7.3	La segmentation avec pagination : le Pentium d'Intel . . . .	251
3.8	Les recherches sur la gestion mémoire . . . . .	256
3.9	Résumé . . . . .	256
	Exercices . . . . .	257
<b>4.</b>	<b>Systèmes de fichiers . . . . .</b>	<b>265</b>
4.1	Les fichiers . . . . .	267
4.1.1	Les noms des fichiers . . . . .	267
4.1.2	La structure des fichiers . . . . .	269
4.1.3	Les types de fichiers . . . . .	270
4.1.4	L'accès aux fichiers . . . . .	273
4.1.5	Les attributs des fichiers . . . . .	273
4.1.6	Les opérations sur les fichiers . . . . .	275
4.1.7	Un exemple de programme utilisant des appels au système de fichiers . . . . .	276
4.2	Les répertoires . . . . .	279
4.2.1	Les systèmes à un seul niveau de répertoire . . . . .	279
4.2.2	Les systèmes à répertoire hiérarchique . . . . .	280
4.2.3	Les chemins d'accès . . . . .	280
4.2.4	Les opérations sur les répertoires . . . . .	283

4.3	Architecture d'un système de fichiers .....	284
4.3.1	L'organisation du système de fichiers .....	284
4.3.2	Mise en œuvre des fichiers .....	285
4.3.3	Mise en œuvre des répertoires .....	291
4.3.4	Les fichiers partagés .....	294
4.3.5	Les systèmes de fichiers LFS .....	297
4.3.6	Les systèmes de fichiers journalisés .....	299
4.3.7	Les systèmes de fichiers virtuels .....	300
4.4	Gestion et optimisation d'un système de fichiers .....	304
4.4.1	Gérer l'espace disque .....	304
4.4.2	La sauvegarde du système de fichiers .....	311
4.4.3	La cohérence du système de fichiers .....	317
4.4.4	L'efficacité du système de fichiers .....	320
4.4.5	La défragmentation des disques .....	325
4.5	Quelques systèmes de fichiers .....	325
4.5.1	Les systèmes de fichiers des CD-ROM .....	326
4.5.2	Le système de fichiers de MS-DOS .....	331
4.5.3	Le système de fichiers d'UNIX V7 .....	335
4.6	Les recherches sur les systèmes de fichiers .....	338
4.7	Résumé .....	338
	Exercices .....	339

## **5. Entrées/sorties** ..... 343

5.1	Les aspects matériels des E/S .....	343
5.1.1	Les périphériques d'E/S .....	344
5.1.2	Le contrôleur de périphérique .....	345
5.1.3	Les E/S projetées en mémoire .....	346
5.1.4	L'accès direct à la mémoire (DMA) .....	350
5.1.5	Un retour sur les interruptions .....	354
5.2	Les principes des logiciels d'E/S .....	358
5.2.1	Les objectifs des logiciels d'E/S .....	358
5.2.2	Les E/S programmées .....	360
5.2.3	Les E/S pilotées par les interruptions .....	362
5.2.4	Les E/S avec DMA .....	363
5.3	La structure en couches des logiciels d'E/S .....	363
5.3.1	Les gestionnaires d'interruptions .....	363



5.3.2	Les pilotes de périphériques .....	365
5.3.3	Le logiciel d'indépendance par rapport au matériel .....	369
5.3.4	Les logiciels d'E/S de l'espace utilisateur .....	375
5.4	Les disques .....	377
5.4.1	Les différents types de disques .....	377
5.4.2	Le formatage des disques .....	393
5.4.3	Les algorithmes d'ordonnancement du bras du disque ...	397
5.4.4	La gestion d'erreurs .....	401
5.4.5	Le stockage stable .....	404
5.5	Les horloges .....	407
5.5.1	L'horloge : la partie matérielle .....	407
5.5.2	L'horloge : la partie logicielle .....	409
5.5.3	Les temporisateurs logiciels .....	412
5.6	L'interface utilisateur : clavier, souris, écran .....	413
5.6.1	Les logiciels d'entrée .....	414
5.6.2	Les logiciels de sortie .....	419
5.7	Les clients légers .....	435
5.8	La gestion de l'alimentation .....	438
5.8.1	Les aspects liés au matériel .....	439
5.8.2	Les aspects liés au système d'exploitation .....	440
5.8.3	Les aspects liés aux applications .....	446
5.9	Les recherches dans le domaine des E/S .....	447
5.10	Résumé .....	448
	Exercices .....	449
<b>6.</b>	<b>Interblocages .....</b>	<b>457</b>
6.1	Les ressources .....	458
6.1.1	Les ressources retirables et non retirables .....	458
6.1.2	L'acquisition de ressources .....	459
6.2	Introduction aux interblocages .....	461
6.2.1	Les conditions pour provoquer un interblocage .....	462
6.2.2	La modélisation des interblocages .....	462
6.3	La politique de l'autruche .....	465
6.4	La détection et la reprise des interblocages .....	465
6.4.1	Détecter un interblocage avec une ressource de chaque type .....	466

6.4.2	Détecter des interblocages avec plusieurs ressources de chaque type .....	468
6.4.3	Reprendre un interblocage .....	471
6.5	L'évitement des interblocages .....	473
6.5.1	Les trajectoires de ressources .....	473
6.5.2	Les états sûrs et non sûrs .....	474
6.5.3	L'algorithme du banquier pour une ressource unique ....	476
6.5.4	L'algorithme du banquier pour plusieurs ressources .....	477
6.6	La prévention des interblocages .....	478
6.6.1	S'attaquer à la condition de l'exclusion mutuelle .....	479
6.6.2	S'attaquer à la condition de détention et d'attente .....	479
6.6.3	S'attaquer à la condition de non-préemption .....	480
6.6.4	S'attaquer à la condition de l'attente circulaire .....	480
6.7	Autres considérations .....	482
6.7.1	Le verrouillage en deux phases .....	482
6.7.2	Les interblocages de communication .....	482
6.7.3	Interblocage actif ( <i>livelock</i> ) .....	484
6.7.4	La privation de ressources .....	486
6.8	Les recherches sur les interblocages .....	486
6.9	Résumé .....	487
	Exercices .....	487

<b>7. Systèmes d'exploitation multimédias</b> .....	<b>493</b>
7.1 Introduction au multimédia .....	494
7.2 Les fichiers multimédias .....	498
7.2.1 Le codage vidéo .....	499
7.2.2 Le codage audio .....	502
7.3 La compression vidéo .....	504
7.3.1 La norme JPEG .....	504
7.3.2 La norme MPEG .....	507
7.4 La compression audio .....	510
7.5 L'ordonnancement de processus multimédias .....	514
7.5.1 L'ordonnancement de processus homogènes .....	514
7.5.2 L'ordonnancement temps réel .....	514
7.5.3 L'ordonnancement RMS .....	516
7.5.4 L'ordonnancement EDF .....	518

7.6	Les paradigmes du système de fichiers multimédia .....	520
7.6.1	Les fonctions de commandes VCR .....	521
7.6.2	La quasi-vidéo à la demande (NVOD) .....	523
7.6.3	La quasi-vidéo à la demande avec des fonctions VCR ....	524
7.7	Le placement de fichier .....	526
7.7.1	Placer un fichier sur un disque unique .....	527
7.7.2	Deux autres stratégies d'organisation de fichiers .....	528
7.7.3	Placer des fichiers NVOD .....	531
7.7.4	Placer plusieurs fichiers sur un seul disque .....	533
7.7.5	Placer des fichiers sur plusieurs disques .....	535
7.8	La mise en cache .....	537
7.8.1	Mettre en cache des blocs .....	538
7.8.2	Mettre en cache des fichiers .....	539
7.9	L'ordonnancement du disque pour le multimédia .....	540
7.9.1	L'ordonnancement statique du disque .....	540
7.9.2	L'ordonnancement dynamique du disque .....	542
7.10	Les recherches sur le multimédia .....	543
7.11	Résumé .....	544
	Exercices .....	545
<b>8.</b>	<b>Systèmes multiprocesseurs .....</b>	<b>551</b>
8.1	Les multiprocesseurs .....	554
8.1.1	Le matériel d'un multiprocesseur .....	554
8.1.2	Les types de systèmes d'exploitation pour les multiprocesseurs .....	563
8.1.3	La synchronisation des multiprocesseurs .....	567
8.1.4	L'ordonnancement des multiprocesseurs .....	571
8.2	Les multi-ordinateurs .....	578
8.2.1	La configuration matérielle d'un multi-ordinateur .....	578
8.2.2	Les logiciels de communication de bas niveau .....	582
8.2.3	Les logiciels de communication au niveau utilisateur ....	584
8.2.4	L'appel de procédure à distance .....	588
8.2.5	La mémoire partagée distribuée .....	590
8.2.6	L'ordonnancement sur un multi-ordinateur .....	595
8.2.7	L'équilibrage de la charge .....	595

8.3	La virtualisation .....	598
8.3.1	Les conditions de la virtualisation .....	600
8.3.2	Les hyperviseurs de type 1 .....	601
8.3.3	Les hyperviseurs de type 2 .....	602
8.3.4	La paravirtualisation .....	603
8.3.5	La virtualisation de la mémoire .....	605
8.3.6	La virtualisation des E/S .....	607
8.3.7	Les serveurs applicatifs virtuels .....	608
8.3.8	Les machines virtuelles sur des UC multicœurs .....	608
8.3.9	La question des licences .....	609
8.4	Les systèmes distribués .....	609
8.4.1	Le matériel réseau .....	612
8.4.2	Les services et les protocoles réseau .....	615
8.4.3	Le middleware fondé sur le document .....	619
8.4.4	Le middleware fondé sur le système de fichiers .....	620
8.4.5	Le middleware basé sur un objet partagé .....	625
8.4.6	Le middleware fondé sur la coordination .....	627
8.4.7	Les ordinateurs en grille .....	633
8.5	Les recherches sur les systèmes multiprocesseurs .....	633
8.6	Résumé .....	634
	Exercices .....	635
<b>9.</b>	<b>Sécurité .....</b>	<b>641</b>
9.1	L'environnement de sécurité .....	643
9.1.1	Les menaces .....	643
9.1.2	Les intrus .....	645
9.1.3	Les pertes accidentelles de données .....	646
9.2	Les bases de la cryptographie .....	646
9.2.1	La cryptographie à clé secrète .....	648
9.2.2	La cryptographie à clé publique .....	648
9.2.3	Les fonctions à sens unique .....	649
9.2.4	La signature numérique .....	650
9.2.5	Une plate-forme digne de confiance .....	652
9.3	Les mécanismes de protection .....	652
9.3.1	Les domaines de protection .....	653
9.3.2	Les listes de contrôle d'accès .....	655
9.3.3	Les capacités .....	658
9.3.4	Les systèmes de confiance .....	661

9.3.5	La base informatique de confiance .....	662
9.3.6	Les modèles formels .....	664
9.3.7	La sécurité multiniveaux .....	665
9.3.8	Les canaux cachés .....	668
9.4	L'authentification .....	673
9.4.1	L'authentification par mot de passe .....	674
9.4.2	L'authentification avec un objet physique .....	683
9.4.3	L'authentification biométrique .....	686
9.5	Les attaques depuis l'intérieur .....	689
9.5.1	Les bombes logiques .....	689
9.5.2	Les portes dérobées .....	690
9.5.3	L'usurpation d'identité .....	691
9.6	L'exploitation des bogues .....	692
9.6.1	Le débordement de mémoire .....	693
9.6.2	Les formats de chaînes de caractères .....	695
9.6.3	Les retours de fonction de la <i>libc</i> .....	697
9.6.4	Les dépassements de capacité .....	699
9.6.5	L'injection de code .....	699
9.6.6	L'élévation des privilèges .....	701
9.7	Les logiciels malveillants .....	701
9.7.1	Les chevaux de Troie .....	704
9.7.2	Les virus .....	706
9.7.3	Les vers .....	717
9.7.4	Les logiciels espions .....	720
9.7.5	Les outils de dissimulation d'activité .....	723
9.8	Les parades .....	729
9.8.1	Les pare-feu .....	729
9.8.2	Les antivirus et les techniques anti-antivirus .....	731
9.8.3	La signature du code .....	739
9.8.4	L'emprisonnement .....	740
9.8.5	La détection d'intrusion par modèle .....	741
9.8.6	L'encapsulation du code mobile .....	743
9.8.7	La sécurité de Java .....	748
9.9	Les recherches sur la sécurité .....	750
9.10	Résumé .....	751
	Exercices .....	752

<b>10. Étude de cas n° 1 : Linux</b> .....	759
10.1 L'historique d'UNIX et de Linux .....	760
10.1.1 UNICS .....	760
10.1.2 L'UNIX du PDP-11 .....	761
10.1.3 UNIX et la portabilité .....	762
10.1.4 L'UNIX de Berkeley .....	763
10.1.5 La standardisation d'UNIX .....	764
10.1.6 MINIX .....	765
10.1.7 Linux .....	766
10.2 Survol de Linux .....	768
10.2.1 Les buts de Linux .....	769
10.2.2 Les interfaces dans Linux .....	770
10.2.3 Le shell .....	772
10.2.4 Les utilitaires Linux .....	775
10.2.5 La structure du noyau .....	777
10.3 Les processus dans Linux .....	779
10.3.1 Les concepts fondamentaux .....	780
10.3.2 Les appels système Linux pour la gestion de processus ...	782
10.3.3 L'implémentation des processus et des threads sous Linux	786
10.3.4 L'ordonnancement sous Linux .....	793
10.3.5 Démarrer Linux .....	796
10.4 La gestion de la mémoire sous Linux .....	799
10.4.1 Les concepts fondamentaux .....	799
10.4.2 Les appels système pour la gestion de la mémoire sous Linux .....	802
10.4.3 L'implémentation de la gestion mémoire sous Linux ....	803
10.4.4 Pagination dans Linux .....	810
10.5 Les entrées/sorties sous Linux .....	813
10.5.1 Les concepts fondamentaux .....	814
10.5.2 Réseau .....	815
10.5.3 Les appels système pour les entrées/sorties sous Linux ...	817
10.5.4 L'implémentation des E/S sous Linux .....	818
10.5.5 Les modules dans Linux .....	821
10.6 Le système de fichiers Linux .....	822
10.6.1 Les concepts fondamentaux .....	822
10.6.2 Les appels système Linux pour le système de fichiers ....	827
10.6.3 L'implémentation du système de fichiers Linux .....	831
10.6.4 NFS : le système de fichiers en réseau .....	840

10.7	La sécurité sous Linux .....	846
10.7.1	Les concepts fondamentaux .....	846
10.7.2	Les appels système pour la sécurité sous Linux .....	849
10.7.3	L'implémentation de la sécurité sous Linux .....	850
10.8	Résumé .....	851
	Exercices .....	852
<b>11.</b>	<b>Étude de cas n° 2 : Windows Vista .....</b>	<b>859</b>
11.1	Historique de Windows Vista .....	859
11.1.1	Années 1980 : MS-DOS .....	860
11.1.2	Années 1990 : Windows MS-DOS .....	861
11.1.3	Années 2000 : Windows NT .....	861
11.1.4	Windows Vista .....	865
11.2	Programmer Windows Vista .....	866
11.2.1	L'interface de programmation d'applications NT native ..	869
11.2.2	L'API Win32 .....	872
11.2.3	Le registre Windows .....	877
11.3	Structure du système .....	880
11.3.1	Structure du système d'exploitation .....	880
11.3.2	Démarrer Windows Vista .....	896
11.3.3	Implémentation du gestionnaire d'objets .....	898
11.3.4	Sous-systèmes, DLL et services en mode utilisateur .....	909
11.4	Processus et threads dans Windows Vista .....	912
11.4.1	Les concepts fondamentaux .....	912
11.4.2	Les appels API pour la gestion des jobs, processus, threads et fibres .....	917
11.4.3	Implémentation des processus et des threads .....	923
11.5	Gestion de la mémoire .....	932
11.5.1	Concepts fondamentaux .....	932
11.5.2	Les appels système pour la gestion de la mémoire .....	937
11.5.3	Implémentation de la gestion mémoire .....	938
11.6	Mise en cache dans Windows Vista .....	948
11.7	Entrées/sorties dans Windows Vista .....	951
11.7.1	Concepts fondamentaux .....	951
11.7.2	Appels d'API d'E/S .....	953
11.7.3	Implémentation des E/S .....	956

11.8	Le système de fichiers Windows NT .....	962
11.8.1	Concepts fondamentaux .....	962
11.8.2	Implémentation du système de fichiers NT .....	963
11.9	La sécurité dans Windows Vista .....	974
11.9.1	Concepts fondamentaux .....	975
11.9.2	Appels d'API de sécurité .....	978
11.9.3	Implémentation de la sécurité .....	979
11.10	Résumé .....	981
	Exercices .....	983
<b>12.</b>	<b>Conception du système d'exploitation .....</b>	<b>987</b>
12.1	La problématique de la conception .....	987
12.1.1	Les objectifs .....	988
12.1.2	La conception d'un système d'exploitation : les raisons de la complexité .....	989
12.2	La conception de l'interface .....	991
12.2.1	Les principes directeurs .....	991
12.2.2	Les paradigmes .....	993
12.2.3	L'interface d'appel système .....	997
12.3	L'implémentation .....	999
12.3.1	La structure du système .....	999
12.3.2	Les procédés et la politique .....	1003
12.3.3	L'orthogonalité .....	1004
12.3.4	Le nommage .....	1005
12.3.5	Les phases de liaisons .....	1007
12.3.6	Les structures statiques ou dynamiques .....	1008
12.3.7	L'implémentation descendante ou ascendante .....	1009
12.3.8	Les techniques utiles .....	1010
12.4	Les performances .....	1016
12.4.1	Pourquoi les systèmes d'exploitation sont-ils lents ? .....	1016
12.4.2	Que faudrait-il améliorer ? .....	1017
12.4.3	Un compromis espace/temps .....	1018
12.4.4	Le cache .....	1021
12.4.5	Les indices .....	1022
12.4.6	L'exploitation de la spatialité .....	1022
12.4.7	L'optimisation du cas le plus courant .....	1023



12.5	La gestion de projet .....	1024
12.5.1	Le mythe de l'homme-mois .....	1024
12.5.2	La structure de l'équipe .....	1025
12.5.3	Le rôle de l'expérience .....	1027
12.5.4	Pas de solution miracle .....	1028
12.6	Les tendances de la conception des systèmes d'exploitation .....	1028
12.6.1	La virtualisation .....	1029
12.6.2	Les puces multicœurs .....	1029
12.6.3	Les systèmes d'exploitation à vaste espace d'adressage ...	1030
12.6.4	Les réseaux .....	1030
12.6.5	Les systèmes parallèles et distribués .....	1031
12.6.6	Les systèmes multimédias .....	1031
12.6.7	Les ordinateurs alimentés par batterie .....	1032
12.6.8	Les systèmes embarqués .....	1032
12.6.9	Les nœuds capteurs .....	1033
12.7	Résumé .....	1033
	Exercices .....	1034
	<b>Bibliographie</b> .....	1039
	<b>Index</b> .....	1041





## L'auteur

**Andrew S. Tanenbaum** est professeur d'informatique à l'Université libre d'Amsterdam et responsable du groupe Systèmes informatiques. Il dirige également l'institut *Advanced School for Computing and Imaging*, établissement d'enseignement et de recherche sur les systèmes parallèles et distribués pour l'image. Il est mondialement connu dans le domaine des systèmes d'exploitation puisqu'il a été à l'origine du *Kit de compilation d'Amsterdam*, outil d'aide à l'écriture de compilateurs qui a connu un immense succès, mais aussi de MINIX, clone d'UNIX destiné aux étudiants en informatique. Plus récemment, avec l'appui de ses doctorants et de son équipe de programmeurs, il a contribué à la définition du système d'exploitation distribué Amoeba, disponible gratuitement sur l'internet, tout comme l'est MINIX. Il s'intéresse aujourd'hui aux très gros systèmes distribués, ceux qui sont destinés à des millions d'utilisateurs. Au titre de ces travaux, il a déjà publié 85 articles de recherche et cinq ouvrages.

Andrew S. Tanenbaum, membre de l'Académie royale des arts et des sciences des Pays-Bas, a le titre de *Fellow* de l'ACM (*Association for Computing Machinery*) et de l'IEEE (*Institute of Electrical and Electronics Engineers*). Lauréat du prix Karl V. Kalstrom 1994 de l'ACM qui distingue un enseignant hors pair, il a également été reçu, en 1997, la médaille de l'ACM/SIGSE pour l'ensemble de son apport au monde de l'enseignement de l'informatique. Il figure au *Who's Who in the World*.





# Préface

Cette troisième édition de *Systèmes d'exploitation* est bien différente de la précédente. En premier lieu, les chapitres ont été réordonnés de façon à ce que le matériel soit traité plus tôt. Ensuite, nous avons plus insisté sur le côté « créateur d'abstractions » du système d'exploitation. Le chapitre 1, qui a été substantiellement mis à jour, expose l'ensemble des concepts. Le chapitre 2 présente cette abstraction de l'UC que sont les processus. Le chapitre 3 est consacré aux espaces d'adressage (mémoire virtuelle) en tant qu'abstraction de la mémoire physique. Le chapitre 4 présente les fichiers, abstraction du disque. Les processus, les espaces d'adressage virtuel et les fichiers sont les concepts clés fournis par les systèmes d'exploitation et, à ce titre, ils se devaient d'être présentés plus tôt qu'ils ne l'étaient auparavant.

Le chapitre 1 a été modifié et mis à jour : il comprend désormais une introduction au langage C plus particulièrement destinée au lecteur qui ne connaîtrait que Java.

Au chapitre 2, l'analyse des threads a été approfondie, ce qui reflète l'importance qu'ils ont pris. Ce chapitre inclut notamment une section sur le standard IEEE Pthreads.

Le chapitre 3, consacré à la gestion de la mémoire, a été réorganisé pour mieux tenir compte de cette idée fondamentale : le rôle du système d'exploitation en la matière est de fournir à chaque processus l'abstraction d'un espace d'adressage virtuel. En conséquence, nous avons supprimé ce qui concernait les vieux systèmes de traitement par lots (*batch*) et recentré les aspects d'implémentation de la pagination sur ce qui est crucial aujourd'hui : la gestion d'immenses espaces d'adressage et la nécessité d'aller considérablement plus vite qu'autrefois.

Les chapitres 4 à 7 ont été mis à jour pour tenir compte des évolutions du domaine. Cela se traduit, entre autres, par nombre d'exercices et problèmes entièrement nouveaux.

Le chapitre 8 a lui aussi été bien réactualisé, en particulier par l'adjonction d'une analyse des systèmes multicœurs ainsi que de toute une section sur la technologie de virtualisation, les hyperviseurs et les machines virtuelles. Pour cette dernière section, c'est VMware qui a été choisi comme exemple.

Le chapitre 9 a été complètement restructuré. Il contient une part importante de nouveautés, essentiellement autour de l'exploitation des vulnérabilités et les façons de s'en défendre.

Le chapitre 10 (sur Linux) est une révision de l'ancien chapitre 10 (sur UNIX et Linux). Nous ne traitons pratiquement plus que de Linux avec beaucoup de nouvelles sections.

Le chapitre 11, consacré à Windows Vista, est une révision de l'ancien chapitre 11 (Windows 2000). Il s'agit donc d'une mise à jour de la façon de traiter de Windows.

Le chapitre 12, sur la conception des systèmes d'exploitation, reprend pour l'essentiel les idées de l'édition précédente.

Beaucoup de gens m'ont aidé à réaliser cette nouvelle édition. D'abord et avant tout, il convient de remercier mon éditrice, Tracy Dunkelberger. J'ai déjà publié 18 ouvrages, j'ai donc une certaine expérience de ce type de travail : elle a su, tour à tour ou simultanément, trouver des contributeurs, gérer les rapports d'évaluation, s'occuper de la contractualisation de toutes les parties prenantes du processus, gérer les relations avec ma maison d'édition, Prentice Hall. Bref, faire en sorte que cet ouvrage sorte en temps et en heure sans pour autant se départir de son calme ni de son sourire. Merci Tracy, je vous suis vraiment reconnaissant.

Ada Gavrilovska de Georgia Tech, experte en Linux, a mis à jour le chapitre 10 qui s'appuie maintenant sur Linux et non plus sur UNIX FreeBSD. Une partie de ce chapitre est certes générique, et s'applique donc à tous les systèmes dérivés des idées d'UNIX, mais Linux est bien plus répandu aujourd'hui chez les étudiants que FreeBSD, d'où l'intérêt de ce changement.

Dave Probert, de Microsoft, a mis à jour le chapitre 11 (Windows Vista) en partant de celui que j'avais écrit sur Windows 2000. Ces deux systèmes certes se ressemblent, mais ils présentent tout de même des différences significatives. Dave connaît très bien le monde Windows et il a beaucoup amélioré la qualité de ce chapitre.

Si Ada, Dave et Mike sont intervenus chacun sur un chapitre, Shivakant Mishra de l'université du Colorado à Boulder s'est plutôt comporté comme un système distribué, lisant et commentant nombre de chapitres, inventant des exercices, etc.

Hugh Lauer mérite également une mention spéciale. Quand nous lui avons demandé s'il avait une idée sur la façon dont devait se faire la révision de la deuxième édition, nous ne nous attendions pas à recevoir un rapport de 23 pages ! Nombre de changements, et notamment tout ce qui concerne cette idée d'abstraction que nous évoquions plus haut, trouvent leur origine dans ce mémo.

Merci également à tous ceux qui ont bien voulu relire les diverses versions du manuscrit, proposer de nouveaux exercices, donner de nouveaux éléments pour la couverture, etc. et, parmi eux, Steve Armstrong, Jeffrey Chastine, John Connelly, Mischa Geldermans, Paul Gray, James Griffioen, Jorrit Herder, Michael Howard, Suraj Kothari, Roger Kraft, Trudy Levine, John Masiyowski, Shivakant Mishra, Rudy Pait, Xiao Qin, Mark Russinovich, Krishna Sivalingam, Leendert van Doorn et Ken Wong.

Toute l'équipe de Prentice Hall s'est, comme d'habitude, montrée disponible et compréhensive, en particulier Irwin Zucker et Scott Disanno à la production et David Alick, ReeAnne Davies et Melinda Haggerty à l'éditorial.

Enfin, *last but not least*, Barbara et Marvin ont été une fois encore adorables, chacun à sa façon bien sûr. Et puis comment ne pas remercier Suzanne de son amour et de sa patience, sans parler des *druiven* et *kersen* qui ont récemment remplacé les *sinaasappels*.

Andrew Tanenbaum

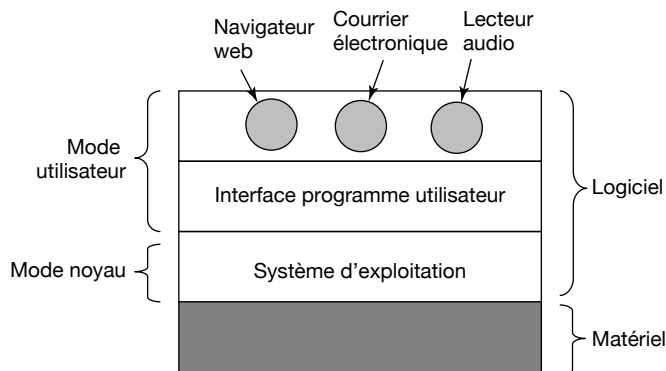
# Introduction

Un système informatique moderne comprend un ou plusieurs processeurs, de la mémoire principale, des disques, des imprimantes, un clavier, un écran, des interfaces réseau et autres périphériques d'entrées/sorties. Tout bien considéré, c'est un système complexe. Écrire des programmes qui prennent en compte tous ces composants et les utilisent correctement, de surcroît de façon optimale, est une tâche extrêmement difficile. Pour cette raison, les ordinateurs sont équipés d'une couche logicielle appelée **système d'exploitation** (SE) dont le rôle est de gérer tous les périphériques et de fournir aux programmes utilisateur une interface simplifiée avec le matériel. Ces systèmes sont le sujet de cet ouvrage.

Les lecteurs qui ont une certaine expérience d'un système d'exploitation, comme Windows, Linux, FreeBSD ou Mac OS X, risquent d'être quelque peu déçus. En effet, le programme avec lequel ils interagissent, appelé couramment le **shell** lorsque l'interaction est en mode texte ou **GUI** (*Graphic User Interface*) lorsqu'elle est en mode graphique ou qu'elle utilise des icônes, ne fait pas partie à proprement parler du système d'exploitation. En réalité, ce programme utilise le système d'exploitation pour effectuer son travail.

Une vue simplifiée des principaux composants dont on parle dans ce chapitre apparaît à la figure 1.1. Tout en bas, on trouve le matériel ou *hardware* qui comprend les circuits intégrés, les cartes d'entrée/sortie, les périphériques (le clavier, la souris, l'écran, les disques, etc.), les alimentations et tous les autres objets matériels de l'ordinateur.

Au-dessus du matériel, on trouve le logiciel ou *software*. La plupart des ordinateurs ont deux modes opératoires : le mode noyau, ou *kernel*, et le mode utilisateur. Le système d'exploitation est le composant logiciel fondamental d'un ordinateur, il fonctionne en **mode noyau** dit aussi **mode superviseur**. Dans ce mode, il a un accès complet et total à toutes les ressources matérielles et peut exécuter n'importe quelle instruction que la machine est capable de traiter. Tous les autres logiciels fonctionnent en **mode utilisateur**, dans lequel une partie seulement des instructions machines sont accessibles. En particulier les instructions qui affectent la commande de la machine ou les entrées/sorties sont inaccessibles en mode utilisateur.



**Figure 1.1** • Place du système d'exploitation au sein d'un ordinateur.

Nous reviendrons fréquemment sur les différences entre mode noyau et mode utilisateur tout au long de cet ouvrage.

L'interface programme utilisateur, le shell ou GUI, est le logiciel de plus bas niveau du mode utilisateur. Il permet à un utilisateur de lancer divers programmes, par exemple un navigateur Web, un gestionnaire de courrier électronique ou un lecteur audio dont les programmes font une utilisation intensive du système d'exploitation.

La place du système d'exploitation (SE) au sein d'un ordinateur est illustrée à la figure 1.1. Le SE fonctionne en lien étroit et direct avec le matériel et constitue la base fonctionnelle de tous les autres logiciels. Il se présente comme une couche logicielle qui cache partiellement le matériel et fournit au développeur un jeu d'instructions plus pratique pour son travail. Par exemple, lire les blocs d'un fichier est une tâche plus simple conceptuellement que devoir tenir compte du déplacement des têtes de lecture-écriture d'un disque, de la latence, etc.

Une importante distinction entre le système d'exploitation et un quelconque logiciel est que, par exemple, si un utilisateur n'aime pas un certain gestionnaire de courrier électronique, il peut librement en choisir un autre, voire en écrire un lui-même ; en revanche, il ne peut pas écrire son propre gestionnaire d'interruption d'horloge qui est partie prenante du système d'exploitation et qui est protégé par le matériel contre toute tentative de modification externe.

Cette distinction, cependant, est parfois vague ou assez floue dans des systèmes *embarqués* ou *enfouis* (qui souvent ne disposent pas d'un mode noyau) ou interprétés (comme les SE basés sur Java, qui fondent la séparation des composants sur l'interprétation et non sur du matériel).

On trouve dans nombre de machines des programmes qui tournent en mode utilisateur tout en aidant le système d'exploitation ou en effectuant des tâches privilégiées. Par exemple, un programme permet aux utilisateurs de modifier leur mot de passe. Ce programme ne fait pas partie du système d'exploitation, il ne fonctionne pas en mode noyau, mais il accomplit clairement une tâche sensible et doit être protégé de manière particulière.



Dans certains systèmes, cette idée est poussée à l'extrême, et des portions de ce que l'on a coutume de regrouper dans le système d'exploitation, comme le système de fichiers, fonctionnent dans l'espace utilisateur. Pour de tels systèmes, il est difficile de tracer une frontière claire. Tout ce qui s'exécute en mode noyau est bien à inclure dans le système d'exploitation, mais il peut en être de même pour certains programmes fonctionnant en mode utilisateur.

Les différences entre systèmes d'exploitation et programmes utilisateur (ou programmes d'application) ne portent pas uniquement, loin de là, sur leur localisation dans la machine. En particulier, les systèmes d'exploitation sont des programmes énormes et complexes. Le code source d'un système d'exploitation comme Linux ou Windows est de l'ordre de cinq millions de lignes de code. Si l'on imprimait ces cinq millions de lignes à raison de 50 lignes par page et 1 000 pages par volume (un peu comme ce livre), il faudrait 100 volumes : une véritable bibliothèque !

Les systèmes d'exploitation ont des durées de vie relativement longues et évoluent sur une longue période de temps. Windows 95/98/Me est un système d'exploitation de Microsoft. Windows NT/2000/XP/Vista est également un système d'exploitation de Microsoft, mais complètement différent, même si pour l'utilisateur lambda ces deux systèmes se ressemblent puisque Microsoft a voulu que l'interface utilisateur de Windows 2000/XP soit analogue à celle de Windows 98. De bonnes raisons ont toutefois conduit Microsoft à se débarrasser de Windows 98. C'est ce que nous verrons lorsque nous étudierons Windows en détail au chapitre 11.

Un autre exemple important que nous étudierons et utiliserons dans ce livre (à côté de Windows), c'est UNIX, ses variantes et ses clones. Ce système d'exploitation a lui aussi évolué au fil des ans, notamment les versions System V, Solaris et FreeBSD qui sont des dérivés du système UNIX d'origine. Quant à Linux, il est basé sur un nouveau code source, même s'il s'inspire beaucoup d'UNIX avec qui il est plus ou moins compatible. Nous utilisons de nombreux exemples issus d'UNIX tout au long de cet ouvrage et nous analysons Linux en détail au chapitre 10.

Dans ce chapitre, nous verrons les points importants des systèmes d'exploitation, notamment ce qu'ils sont, leur historique, leur environnement, leurs concepts de base et leur structure. Nous reviendrons ensuite sur tous ces points plus en détail dans les chapitres suivants.

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

---

Il est difficile de dire précisément en quoi consiste un système d'exploitation. Le problème vient pour partie du fait que le système d'exploitation remplit deux tâches *a priori* sans relations : il offre aux programmeurs d'applications (et aux logiciels d'applications) un ensemble clair de ressources d'une part, et d'autre part il gère les ressources matérielles de l'ordinateur. Suivant l'interlocuteur, on s'intéresse à l'un ou l'autre de ces aspects. Examinons-les tous les deux.

### 1.1.1 Le système d'exploitation comme machine étendue

L'**architecture** (ensemble d'instructions, organisation de la mémoire, E/S et structure du bus) de la plupart des ordinateurs, au niveau du langage machine, est primitive et assez fastidieuse à programmer, particulièrement en ce qui concerne les entrées/sorties. Pour fixer les idées, penchons-nous un instant sur les E/S disquettes réalisées par le contrôleur NEC PD765 qui était présent sur la plupart des ordinateurs personnels à base de microprocesseur Intel. Nous prenons la disquette comme exemple car, bien qu'étant obsolète, elle est beaucoup plus simple à expliquer et à comprendre qu'un disque dur moderne actuel. Donc, ce contrôleur PD765 comprend 16 commandes, spécifiées en chargeant de 1 à 9 octets dans un registre de périphérique. Elles permettent la lecture et l'écriture de données, le déplacement du bras, le formatage des pistes ainsi que l'initialisation, la détection, la remise à zéro et le recalibrage du contrôleur et des périphériques.

Les commandes les plus simples sont *read* et *write*, qui réclament chacune 13 paramètres compactés dans 9 octets. Ces paramètres précisent des informations aussi variées que l'adresse du bloc à lire/écrire, le nombre de secteurs par piste, le mode d'enregistrement utilisé par le support physique, l'espacement entre secteurs, et le mode de prise en compte des marqueurs d'adresses ou de données détruites. Si ce jargon vous est totalement étranger, pas d'inquiétude : c'est effectivement très ésotérique. Quand l'opération demandée se termine, le contrôleur retourne 23 champs de statuts et d'erreurs compactés dans 7 octets. Comme si cela ne suffisait pas, le programmeur du lecteur de disquettes doit également être constamment conscient de l'état d'activité du moteur. Si ce dernier est inactif, il doit être démarré (ce qui entraîne un délai de latence important) avant que les données puissent être lues ou écrites. Le moteur ne peut cependant pas être constamment actif, sinon la disquette serait vite usée. Le programmeur doit donc jongler entre le désagrément de devoir attendre longtemps la réaction du lecteur et une usure trop rapide des disquettes (ce qui entraîne la perte des informations qu'elles contiennent).

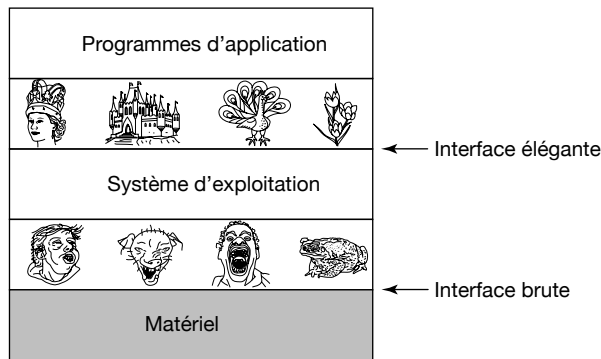
Sans rentrer véritablement dans les détails, il apparaît clairement que le programmeur moyen n'a que très modérément envie d'être confronté de trop près à la programmation des lecteurs de disquettes (ou des disques durs, qui sont encore plus complexes tout en étant très différents). Au contraire, ce que le programmeur souhaite, c'est une interface de programmation plus abstraite (de plus haut niveau). Dans le cas des disques, l'abstraction typique consiste à les considérer comme des collections de fichiers. Chaque fichier peut être ouvert en lecture ou en écriture, puis accédé, et enfin fermé. Les détails comme l'utilisation éventuelle de modulation de fréquence modifiée pour l'enregistrement ou l'état courant du moteur sont cachés à l'utilisateur par cette abstraction.

L'**abstraction** est la clé de gestion de la complexité. L'abstraction idéale n'existe pas. La bonne abstraction résulte d'un compromis entre deux tâches à optimiser : celle qui consiste à définir et à implémenter l'abstraction et celle qui consiste à utiliser l'abstraction pour résoudre le problème posé. Une abstraction que la quasi-totalité des utilisateurs comprend est le fichier. C'est l'élément d'information essentiel dans un ordinateur, tel qu'une photo numérique, un message électronique ou une page Web. Évoquer directement un fichier est bien plus simple que de faire référence aux

détails des multiples actions qui lui sont liées, comme on l'a vu avec la disquette décrite plus haut.

Le rôle du système d'exploitation est d'offrir un bon niveau d'abstraction permettant d'implémenter et de bien gérer les objets abstraits qu'il a créés. Nous faisons souvent référence à des abstractions au fil de cet ouvrage. C'est une bonne façon de comprendre les systèmes d'exploitation.

Ainsi, le système d'exploitation est le programme qui cache les détails de fonctionnement du matériel et présente au programmeur une interface simple et élégante à base de fichiers à lire et/ou à écrire. De même, c'est lui qui l'affranchit de tâches pesantes à propos des interruptions, des compteurs de temps, ou *timers*, de la gestion de la mémoire et autres caractéristiques de bas niveau. À chaque fois, l'abstraction présentée par le système d'exploitation est plus simple à comprendre et à utiliser que celle fournie par le matériel sous-jacent. On peut ainsi dire du système d'exploitation qu'il transforme le matériel brut en élégantes abstractions, comme le montre la figure 1.2.



**Figure 1.2** • Le système d'exploitation transforme le matériel brut en élégantes abstractions.

Les vrais utilisateurs du système d'exploitation sont les programmes d'application. Ce sont eux qui traitent directement avec lui et ses différentes abstractions. En revanche, l'utilisateur final ne traite qu'avec les abstractions fournies par l'interface utilisateur (lignes de commande du shell ou interface graphique GUI). Les abstractions offertes à l'interface utilisateur peuvent, ou non, sembler comparables à celles du système d'exploitation. Pour mieux comprendre cela, prenons par exemple le Bureau de Windows et une ligne de commande. Les deux sont des programmes qui s'exécutent sur le système d'exploitation Windows et qui utilisent certaines de ses abstractions, mais chacun présente une interface utilisateur très différente. Il en est de même de l'utilisateur de Linux qui, exécutant Gnome ou KDE, voit une interface utilisateur très différente de celle de X Window, même si dans ces deux cas les abstractions sous-jacentes sont identiques.

Dans cet ouvrage, nous étudierons en détail les abstractions proposées aux programmes d'application sans nous étendre outre mesure sur celles des interfaces utilisateurs, sujet important mais qui se trouve plus à la périphérie des systèmes d'exploitation.

### 1.1.2 Le système d'exploitation comme gestionnaire de ressources

La vue développée précédemment est une vue descendante. À l'inverse, on peut adopter un point de vue ascendant consistant à dire que le système d'exploitation doit gérer l'ensemble des éléments d'un système fort complexe. Les ordinateurs modernes sont constitués d'un ou plusieurs processeurs, d'une mémoire, de timers (temporisateurs), de disques, d'une souris, d'une ou plusieurs interfaces réseau et imprimantes, et le cas échéant de divers autres périphériques. Dans ce second point de vue, le système d'exploitation doit gérer de manière équitable et optimale l'allocation des processeurs, de la mémoire et des périphériques d'E/S aux différents programmes concurrents qui les sollicitent.

Les systèmes d'exploitation modernes permettent à plusieurs programmes de s'exécuter simultanément. Imaginons ce qui se passerait si trois programmes s'exécutant sur une machine essayaient en même temps d'imprimer sur la même imprimante. Les premières lignes proviendraient du premier programme, les lignes suivantes du second, puis on verrait quelques lignes du troisième programme et ainsi de suite. Le résultat serait totalement chaotique. Le système d'exploitation est là pour mettre de l'ordre dans ce chaos prévisible en plaçant dans un buffer (une mémoire tampon) sur le disque toutes les lignes de sortie destinées à l'imprimante. Quand un programme se termine, le système d'exploitation peut alors recopier le tampon sur l'imprimante, alors que les autres programmes continuent de remplir le disque de leurs propres sorties.

Quand un ordinateur (ou un réseau) sert plusieurs utilisateurs, le besoin de gérer et de protéger la mémoire, les périphériques d'E/S et les autres ressources augmente encore, dans la mesure où les différents utilisateurs augmentent le risque de conflits d'accès aux ressources. De plus, les utilisateurs ne se contentent pas en général de partager le matériel : ils souhaitent également pouvoir partager l'information (les fichiers, les bases de données, etc.). En clair, cet aspect du rôle du système d'exploitation met en évidence la nécessité de savoir qui utilise quelle ressource, d'autoriser ou de refuser l'accès à telle ressource, de mesurer l'utilisation par chacun de chaque ressource et de résoudre les conflits d'accès entre différents programmes et utilisateurs.

La gestion des ressources implique deux dimensions du **partage** (on parle de **multiplexage**) des ressources : dans le temps et dans l'espace. Quand une ressource est partagée dans le temps, les candidats à son utilisation prennent leur tour l'un après l'autre. Ainsi, avec une seule unité centrale (de traitement), **UC** ou **CPU** (*Central Processing Unit*) et plusieurs programmes concurrents, le système d'exploitation alloue l'UC en premier à un programme, puis à d'autres tour à tour, et revient le cas échéant au premier programme. Le choix du programme qui va bénéficier de l'UC lors du prochain changement – et pour combien de temps – est du ressort du système d'exploitation. Un autre exemple est le partage de l'imprimante. Il est nécessaire de choisir celui parmi les jobs candidats à l'impression qui sera le prochain à être servi.

L'autre catégorie de multiplexage est le multiplexage spatial. Plutôt que les clients attendent leur tour, chacun d'eux reçoit, immédiatement, une partie de la ressource. Par exemple, la mémoire principale est d'ordinaire partagée entre les quelques

programmes actifs de façon que chacun d'eux soit résident en même temps (afin, par exemple, que chacun puisse, à son tour, utiliser l'UC). Si l'on suppose qu'il y a assez de mémoire pour satisfaire les besoins de tous les programmes candidats, il est plus efficace de tous les maintenir en mémoire en même temps que de donner la totalité de la mémoire à chacun d'eux, tour à tour, surtout s'ils n'utilisent qu'une petite fraction de celle-ci. Bien entendu, cela soulève des questions de bon comportement, de protection, etc. C'est au système d'exploitation de les résoudre.

Une autre ressource partagée de manière spatiale est le disque dur. Dans la plupart des systèmes, un même disque peut abriter des fichiers de plusieurs utilisateurs en même temps. L'allocation d'espace disque et la mémorisation de « qui utilise quels blocs » est une tâche représentative des activités de gestion de ressources du système d'exploitation.

## 1.2 L'histoire des systèmes d'exploitation

---

Les systèmes d'exploitation ont évolué au fil des ans. Dans les sections suivantes, nous allons brièvement décrire quelques-unes des étapes de cette évolution. Dans la mesure où ils ont été historiquement très liés à l'architecture des ordinateurs sur lesquels ils étaient implémentés, nous décrirons les générations successives d'ordinateurs et observerons à quoi ressemblait leur système d'exploitation. Cette projection (on parle de *mappage*) des générations successives de systèmes d'exploitation sur les générations correspondantes d'ordinateurs est assez grossière, mais elle a le mérite de structurer ce qui, autrement, pourrait apparaître plus aléatoire.

L'évolution présentée ci-dessous est purement chronologique, mais elle correspond en réalité à une progression chaotique.

Le premier ordinateur réellement numérique a été conçu par le mathématicien anglais Charles Babbage (1792-1871). Bien que Babbage ait passé le plus clair de sa vie, et de sa fortune, à construire son « moteur analytique », il n'est jamais parvenu à le faire fonctionner correctement parce qu'il était entièrement mécanique et que la technologie de l'époque ne pouvait lui fournir les rouages et autres outils au niveau de précision que sa machine nécessitait. Il est inutile de préciser que cette machine analytique ne disposait pas de système d'exploitation.

Détail historique intéressant, Babbage a eu l'intuition qu'il aurait besoin d'un logiciel pour son moteur analytique. Il a alors loué les services d'une jeune femme nommée Ada Lovelace, fille du célèbre poète Lord Byron, en tant que premier programmeur au monde. Le langage Ada a été ainsi nommé en hommage à cette femme.

### 1.2.1 La première génération (1945-1955) : tubes à vide

Après les efforts infructueux de Babbage, peu d'avancées ont été réalisées dans le domaine des ordinateurs numériques avant la Seconde Guerre mondiale, laquelle stimula une profusion d'activité dans ce domaine. C'est ainsi que le professeur John Atanasoff et l'un de ses étudiants, Clifford Berry, de l'université de l'État d'Iowa,

construisirent une machine qui apparaît aujourd'hui comme le premier véritable ordinateur numérique. Cette machine utilisait 330 tubes à vide. À peu près au même moment, Konrad Zuse à Berlin construisit l'ordinateur Z3 basé sur des relais mécaniques. En 1944, l'ordinateur Colossus fut construit par un groupe d'ingénieurs de Bletchley Park en Angleterre, de même que le Mark I construit par Howard Aiken à Harvard, et l'ENIAC par le professeur William Mauchley avec l'un de ses étudiants, J. Presper Eckert, à l'université de Pennsylvanie. Certains ordinateurs utilisaient des tubes à vide, d'autres des relais mécaniques. Certains étaient programmables. Tous étaient très primitifs et mettaient de une à plusieurs secondes pour réaliser des calculs simples.

Dans ces premiers temps, la même équipe (le plus souvent des ingénieurs) concevait, construisait, programmait, administrait et maintenait chaque machine. Tout programme était conçu en langage machine absolu, souvent en basculant des tableaux d'interrupteurs pour contrôler les fonctions de base de la machine. Les langages de programmation étaient inconnus (y compris le langage d'assemblage). On n'avait jamais entendu parler de système d'exploitation. Le protocole classique d'utilisation consistait pour le programmeur à s'inscrire pour une certaine durée d'utilisation sur un papier de la salle machine, puis de s'y rendre à l'heure dite, insérer son tableau d'interrupteurs dans l'ordinateur et prier pour que, dans les quelques heures qui suivraient, aucun des environ 20 000 tubes ne grille pendant l'exécution. À peu près tous les problèmes codés étaient de simples calculs numériques, comme remplir des tables de sinus ou de logarithmes.

Au début des années 1950, le protocole s'est amélioré quelque peu avec l'introduction des cartes perforées. Il était devenu possible d'écrire ses programmes sur des cartes et de les faire lire par la machine au lieu d'utiliser les tableaux d'interrupteurs. Mis à part cela, la procédure restait la même.

## 1.2.2 La deuxième génération (1955-1965) : transistors et traitement par lots

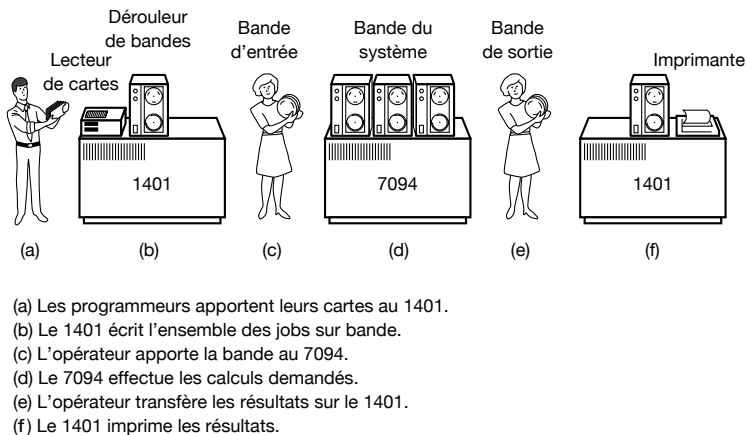
L'apparition du transistor au milieu des années 1950 a radicalement changé la donne. Les ordinateurs devenaient suffisamment fiables pour être produits et vendus dans l'idée qu'ils fonctionneraient suffisamment longtemps pour produire un retour sur investissements. Pour la première fois, on a assisté à une séparation nette entre concepteurs, constructeurs, opérateurs, programmeurs et personnel de maintenance.

Ces machines, appelées de nos jours **mainframes** ou ordinateurs centraux, étaient enfermées dans des locaux spéciaux climatisés, accessibles uniquement des opérateurs. Seules les grandes firmes ou établissements publics pouvaient investir les millions de dollars nécessaires à leur achat. Pour soumettre un *job* (un programme, ou un ensemble de programmes), le programmeur commençait par écrire son programme sur papier (en FORTRAN ou en assembleur), puis il le codait sur des cartes perforées. Il apportait ensuite son paquet de cartes dans la salle de soumission des jobs et le confiait à l'opérateur présent. Il pouvait ensuite aller boire un café en attendant le résultat.

Quand l'ordinateur finissait un job, un opérateur récupérait sur une imprimante la trace d'exécution correspondante et la stockait dans la salle des résultats afin que le programmeur la récupère. Il prenait ensuite l'un des paquets de cartes de la salle de soumission et le faisait lire par la machine. Si le compilateur FORTRAN était nécessaire, l'opérateur devait également le charger depuis un robot de stockage de fichiers. Une grande quantité de temps était gaspillée pendant ces diverses manipulations humaines.

Étant donné l'importance de l'investissement, il n'est pas étonnant que l'on se soit rapidement mis à réfléchir sur la façon de réduire cette perte de temps. La solution la plus fréquemment adoptée a alors été le système de *batch* ou de **traitement par lots**. L'idée était d'utiliser une machine intermédiaire peu onéreuse comme l'IBM 1401, qui était plus performant pour les entrées/sorties que pour les calculs numériques, pour lire les cartes de l'ensemble des jobs soumis et les écrire sur une bande magnétique.

On donnait ensuite au vrai calculateur plus coûteux et performant, comme l'IBM 7094, la bande contenant l'ensemble des jobs et, grâce à un programme spécial (l'ancêtre des systèmes d'exploitation actuels), le calculateur 7094 lisait le premier job et l'exécutait, puis le second et l'exécutait, et ainsi de suite jusqu'au dernier job. Le résultat des jobs était de nouveau écrit sur une bande au lieu d'être imprimé. Quand l'ensemble des jobs était terminé, l'opérateur reprenait les bandes, remplaçant une autre bande de soumission dans le calculateur et le relançait. Il allait ensuite imprimer les résultats en rechargeant la seconde bande sur l'ordinateur intermédiaire, l'IBM 1401. On parle dans un tel cas d'impression **off-line** (c'est-à-dire hors ligne ou non connectée à l'ordinateur). L'ensemble de cette procédure est illustrée à la figure 1.3.

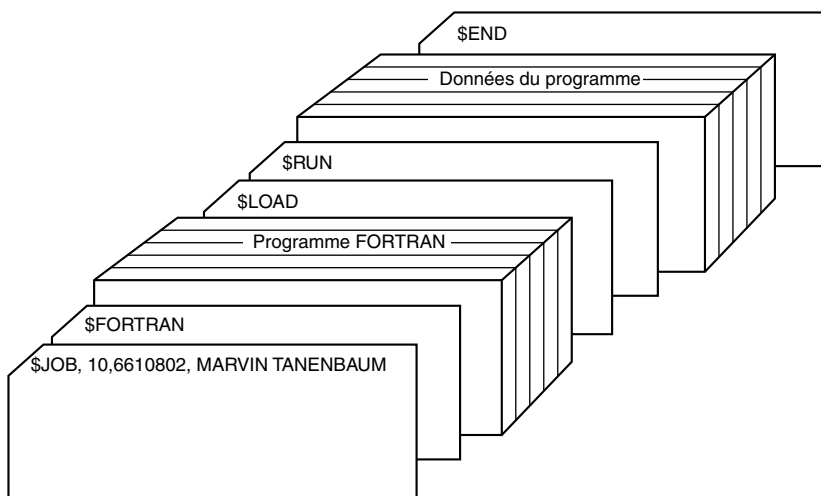


**Figure 1.3** • Un ancien système de batch.

La structure d'un job typique est reproduite à la figure 1.4. Il débutait par une carte \$JOB spécifiant le temps maximal imparti au job (en minutes), le compte à débiter et le nom du programmeur. On trouvait ensuite une carte \$FORTRAN indiquant au

système de charger le compilateur FORTRAN depuis la bande système. Ensuite venait le programme à compiler, puis une carte \$LOAD demandant le chargement du programme qui venait d'être compilé (les programmes compilés étaient fréquemment écrits sur des bandes spécifiques et devaient être chargés explicitement). On avait ensuite une carte \$RUN déclenchant l'exécution du programme avec les données qui la suivaient. Enfin, la carte \$END indiquait la fin du job. Ce système de contrôle primitif était le précurseur des langages modernes de contrôle de job et d'interprétation de commandes.

Les gros ordinateurs de la seconde génération étaient utilisés principalement pour des calculs scientifiques ou d'ingénierie, comme la résolution d'équations aux dérivées partielles. Ils étaient dans leur grande majorité programmés en FORTRAN et assembleur. Les systèmes d'exploitation de l'époque étaient notamment FMS (*Fortran Monitor System*) et IBSYS, le système d'exploitation de l'IBM 7094.



**Figure 1.4** • Structure d'un job typique FMS.

### 1.2.3 La troisième génération (1965-1980) : circuits intégrés et multiprogrammation

Au début des années 1960, la plupart des constructeurs d'ordinateurs proposaient deux lignes de produits totalement incompatibles. D'un côté, on trouvait des ordinateurs orientés mots, qui étaient destinés à des tâches de calculs numériques scientifiques intensifs, comme le 7094. De l'autre, on trouvait des ordinateurs orientés caractères qui étaient à vocation plus commerciale, comme le 1401. Ils étaient très utilisés pour la gestion des bandes magnétiques et l'impression, tant par les banques que par les compagnies d'assurances.

Le développement et la maintenance de deux lignes de produits aussi distinctes présentaient un surcoût important pour les constructeurs. En outre, de nombreux



clients ne pensaient initialement avoir besoin que d'une petite machine, mais demandaient rapidement une machine plus performante, capable de faire tourner tous leurs programmes plus rapidement.

IBM a tenté de répondre simultanément aux deux types de problèmes en présentant son *system/360*. Le 360 consistait en une série de machines compatibles au niveau logiciel, dont la taille s'étendait de celle du 1401 à bien au-delà du 7094. Ces machines différaient uniquement au niveau du prix et des performances (mémoire maximale, vitesse du processeur, nombre de périphériques tolérés, etc.). Comme toutes ces machines possédaient la même architecture et le même jeu d'instructions, tous les programmes écrits pour l'une étaient – au moins en principe – capables de fonctionner sur n'importe quelle autre. De plus, le 360 était conçu pour gérer aussi bien des applications scientifiques que commerciales. La même série de machines pouvait donc satisfaire tous les clients potentiels. Dans les années qui ont suivi, IBM a continué de proposer des successeurs compatibles 360, en utilisant des technologies plus évoluées, comme les 370, 4300, 3080 et 3090. La série Z est le plus récent descendant de cette ligne de produits, bien qu'il ait très largement divergé du modèle d'origine.

Les 360 ont été les premières machines à utiliser des **circuits intégrés**, ce qui leur donnait un immense avantage au plan prix/performance sur les machines de seconde génération, entièrement conçues à base de transistors. Le succès a été immédiat, et l'idée d'une ligne de produits compatibles a aussitôt été reprise par tous les principaux constructeurs. Les descendants de ces machines sont encore utilisés de nos jours dans les centres informatiques. On les utilise d'ordinaire pour gérer de très grandes bases de données (par exemple, pour la gestion des réservations de billets d'avion) ou comme serveurs Web pour des sites devant gérer des milliers de requêtes par seconde.

La plus grande force du concept de « famille unique » a aussi été sa plus grande faiblesse. Le but était que tout logiciel, y compris le système d'exploitation **OS/360**, devait fonctionner sur tous les modèles, depuis de petits systèmes destinés à remplacer les 1401 dans leurs travaux de gestion des bandes magnétiques et des imprimantes, jusqu'aux très grands systèmes qui remplaçaient avantageusement les 7094 dans les domaines des prévisions météorologiques ou d'autres calculs intensifs. Il devait être performant pour des systèmes à faible – ou à très grand – nombre de périphériques. Il devait bien fonctionner et être efficace dans des environnements commerciaux comme auprès des scientifiques.

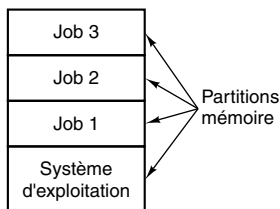
Il était impossible, pour IBM ou qui que ce soit d'autre, de produire un logiciel satisfaisant des contraintes aussi contradictoires. Le résultat a donné un système d'exploitation énorme et extraordinairement complexe, de deux ou trois ordres de grandeur au-dessus de FMS. Il était constitué de millions de lignes d'assembleur écrites par des milliers de programmeurs, et contenait des milliers de bogues entraînant sans arrêt de nouvelles versions de maintenance corrective. Bien entendu, chaque nouvelle version introduisait de nouveaux bogues et il est probable que le nombre total est globalement resté constant au fil du temps.

L'un des concepteurs d'OS/360, Fred Brooks, a publié à la suite de cette expérience un petit ouvrage décrivant ses aventures avec le système. Même s'il est impossible de

résumer ici l'ouvrage, il suffit de mentionner que la couverture montrait une horde d'animaux préhistoriques pris dans une mare de goudron. La couverture de l'ouvrage de Silberschatz en 2000 reprit la même idée en montrant les systèmes d'exploitation comme des dinosaures.

En dépit de sa taille gigantesque et de ses problèmes, OS/360 et les autres systèmes de troisième génération concurrents ont globalement répondu à l'attente de leurs clients. Ils ont également généralisé certaines techniques fondamentales absentes de leurs correspondants de la génération précédente. La plus importante d'entre elles est probablement la **multiprogrammation**. Sur le 7094, quand le job actif s'arrêtait pour attendre une entrée/sortie sur une bande ou tout autre périphérique, l'UC attendait simplement que l'E/S ait terminé. Dans les applications de calcul intensif, les E/S sont peu fréquentes et cette perte de temps est négligeable. Dans les applications commerciales, le temps consacré aux E/S peut représenter parfois 80 % à 90 % du temps total. Il fallait donc réagir pour utiliser cette si coûteuse UC de façon plus intelligente.

La solution a consisté à partitionner la mémoire en plusieurs morceaux, chaque job pouvant accéder à l'un des morceaux, comme le montre la figure 1.5. Quand un job attendait la réalisation d'une opération d'E/S, un autre pouvait s'emparer de l'UC. Si un nombre suffisant de jobs pouvait être chargés en mémoire simultanément, on pouvait arriver à un taux d'utilisation de l'UC voisin de 100 % en permanence. Maintenir de façon sûre plusieurs jobs en mémoire simultanément requiert des mécanismes spéciaux de protection matérielle des jobs contre l'intrusion des jobs concurrents. Le 360 et ses concurrents ont été équipés de ce type de matériel spécifique.



**Figure 1.5** • Un système multiprogrammé avec trois jobs en mémoire.

Une autre caractéristique importante des systèmes d'exploitation de troisième génération est leur capacité à stocker sur le disque les jobs écrits sur cartes au fur et à mesure de leur arrivée. Ainsi, dès qu'un job était terminé, le système d'exploitation pouvait en charger un nouveau depuis le disque et l'exécuter sur la partition libérée. Cette technique est appelée **spoulage** ou **spoule** (de *spool*, *Simultaneous Peripheral Operation On Line*) et est également utilisée en sortie. Avec le spoulage, le 1401 n'était plus nécessaire et la plupart des manipulations de bandes disparurent.

Bien que ces systèmes d'exploitation soient bien adaptés au calcul intensif et au traitement de grandes masses d'informations commerciales, ils n'étaient encore que des systèmes batch. De nombreux utilisateurs regrettaient les beaux jours de la première

génération, où ils avaient la machine pour eux seuls pendant quelques heures et pouvaient donc rapidement mettre au point leurs programmes. Avec la troisième génération, la durée entre la soumission d'un job et la récupération de ses résultats se chiffrait en heures, et une simple erreur de « ; » pouvait entraîner l'échec de la compilation et la perte d'une demi-journée.

Ce désir de réponse rapide a conduit au **temps partagé**, une variante de la multiprogrammation dans laquelle chaque utilisateur dispose d'un terminal en ligne. Dans un tel système, si 20 utilisateurs sont en ligne et que 17 d'entre eux réfléchissent, discutent ou boivent un café, l'UC peut servir tour à tour les 3 jobs qui réclament effectivement des ressources. Dans la mesure où, lors d'une mise au point de programme, on exécute fréquemment des commandes courtes (par exemple, la compilation d'une procédure de 5 pages plutôt que le tri d'un fichier contenant un million d'enregistrements), l'ordinateur peut fournir un service rapide et interactif à plusieurs utilisateurs et même sans doute travailler sur de gros jobs batch en tâche de fond, quand l'UC était auparavant inactive. Le premier système réellement à temps partagé, **CTSS** (*Compatible Time Sharing System*), a été développé au MIT sur un 7094 modifié. Cela dit, le temps partagé n'est devenu réellement populaire que lorsque les mécanismes de protection matérielle nécessaires ont été systématisés dans les systèmes de troisième génération.

Devant le succès de CTSS, le MIT, Bell Labs et General Electric (qui était à l'époque un acteur majeur de la construction d'ordinateurs) ont décidé de travailler à la réalisation d'un « outil informatique », une machine capable de supporter des centaines d'utilisateurs simultanés. Leur modèle empruntait beaucoup à celui de la distribution d'électricité – quand on a besoin de courant, on branche simplement une prise dans le mur, et immédiatement, la quantité désirée d'énergie est disponible. Les concepteurs de ce système, connu sous le nom de **MULTICS** (*MULTIplexed Information and Computing Service*), voyaient une seule énorme machine fournissant de la puissance de calcul à toute la zone de Boston. L'idée que des machines 10 000 fois plus puissantes que leur GE-645 seraient vendues 1 000 € au grand public 40 ans plus tard relevait de la science-fiction, à peu près au même titre que le serait de nos jours celle d'un train sous-marin supersonique traversant l'Atlantique.

MULTICS a obtenu un succès mitigé. Il était conçu pour supporter des centaines d'utilisateurs sur une machine à peine plus puissante qu'un PC à base d'Intel i386, même si elle était dotée de plus de capacités d'E/S. Ce n'est pas aussi délirant qu'il n'y paraît, car on savait à l'époque écrire des programmes concis et efficaces, capacité qui a quasiment disparu depuis. Parmi les raisons qui ont fait que MULTICS ne s'est pas développé dans le monde entier, le fait qu'il était écrit en PL/1 n'est pas la moindre : le compilateur PL/1 est arrivé des années en retard sur les prévisions, et il fonctionnait tout juste correctement. En outre, MULTICS était extraordinairement ambitieux pour l'époque, un peu à la manière de la machine de Babbage pour le XIX<sup>e</sup> siècle.

Pour résumer, MULTICS a introduit en germe de nombreuses innovations, mais convertir celles-ci en un produit opérationnel à grand succès commercial s'est révélé beaucoup plus compliqué que ce à quoi on s'attendait. Bell Labs s'est retiré du projet, et G.E. a abandonné carrément le domaine de la construction d'ordinateurs.

Le MIT a cependant persisté, et est parvenu à produire une version opérationnelle de MULTICS. Celle-ci a été commercialisée par la société qui venait de racheter les activités informatiques de G.E., Honeywell, et installée dans 80 grandes entreprises et universités de par le monde. Bien que peu nombreux, les utilisateurs de MULTICS ont été enthousiastes et fidèles. General Motors, Ford et la NSA (*National Security Agency*), par exemple, n'ont interrompu définitivement leurs derniers systèmes MULTICS qu'à la fin des années 1990, soit près de trente ans après la création du système.

Pour l'instant, cette idée d'un « outil informatique » n'est pas à l'ordre du jour, mais elle pourrait bien revenir sous la forme de groupes massifs de serveurs internet auxquels se connecteraient des terminaux ne possédant qu'un minimum d'intelligence, la plus grande partie du travail étant réalisée par le serveur. L'idée vient du fait que la plupart des utilisateurs refusent d'avoir à maintenir un système de plus en plus complexe et préfèrent déléguer cette tâche à une équipe de professionnels employés par l'entreprise gérant le serveur. Le commerce électronique fait déjà un pas dans cette direction.

Malgré ce relatif échec commercial, MULTICS a exercé une influence énorme sur l'évolution des systèmes d'exploitation.

La percée des mini-ordinateurs est une autre caractéristique importante de la troisième génération. Elle a commencé avec le DEC PDP-1 en 1961. Le PDP-1 ne possédait que 4 K-mots de 18 bits, mais à 120 000 dollars pièce (soit moins de 5 % du prix d'un 7094), il s'est vendu comme des petits pains. Pour certaines tâches non scientifiques, il atteignait presque les performances du 7094 et il a donné naissance à toute une nouvelle industrie. Il a rapidement été suivi de toute une famille d'autres PDP (qui, au contraire des IBM, n'étaient pas compatibles entre eux), culminant avec le PDP-11.

L'un des informaticiens de Bell Labs, qui avaient travaillé sur le projet MULTICS, Ken Thompson, est tombé sur un PDP-7 inutilisé et a commencé à écrire une version allégée, mono-utilisateur de MULTICS. Ce travail a servi de base au système **UNIX**, qui est devenu célèbre dans le monde académique, dans nombre d'administrations et d'entreprises.

L'histoire d'UNIX a été souvent contée. Le lecteur en trouvera une partie au chapitre 10. Pour l'instant, il nous suffit de dire qu'en raison de la publication du code source, plusieurs organismes ont développé leurs propres versions (incompatibles), ce qui a conduit au chaos. Deux versions principales se sont développées, **System V** d'AT&T et **BSD** (*Berkeley Software Distribution*) de l'Université de Berkeley, en Californie. Afin de rendre possible l'écriture de programmes qui puissent fonctionner sur tout système UNIX, l'IEEE a développé un standard appelé **POSIX**, auquel la plupart des versions actuelles d'UNIX se conforment. POSIX définit une interface minimale d'appels système que tout système compatible doit implémenter. En fait, des systèmes autres qu'UNIX supportent également l'interface POSIX.

À ce sujet, notons que l'auteur de cet ouvrage a proposé en 1987 un petit clone d'UNIX appelé **MINIX**, dans un but pédagogique. Fonctionnellement, MINIX est

proche d'UNIX, y compris en ce qui concerne le support de POSIX. Depuis, le système MINIX a évolué. Il en est à sa version 3, MINIX 3, qui est très modulaire et présente un haut degré de fiabilité. Le système MINIX 3 est disponible gratuitement – y compris le code source – sur l'internet à l'URL [www.minix3.org](http://www.minix3.org).

Le besoin d'une version de production, par opposition à la version d'éducation évoquée ci-dessus, de MINIX a conduit un étudiant finlandais, Linus Torvalds, à écrire **Linux**. Ce système a été écrit à partir de MINIX et supportait au départ plusieurs caractéristiques de ce système (par exemple son système de fichiers). Il a depuis été considérablement étendu mais conserve une bonne part de sa structure de base commune avec MINIX et UNIX. La grande majorité de ce qui sera dit dans ce livre sur UNIX s'applique donc à System V, BSD, MINIX, Linux et d'autres versions et clones d'UNIX.

### 1.2.4 La quatrième génération (1980-aujourd'hui) : les ordinateurs personnels

Avec le développement des circuits intégrés LSI (*Large Scale Integration circuits*), on a assisté à l'apparition de puces contenant des milliers de transistors sur un millimètre carré de silicium : l'âge des ordinateurs personnels était arrivé. En termes d'architecture, les ordinateurs personnels (appelés au départ micro-ordinateurs) n'étaient pas très éloignés des mini-ordinateurs du type PDP-11, mais les échelles de prix étaient fort différentes. Là où le mini permettait à un département d'une entreprise ou d'une université l'acquisition d'un ordinateur dédié, le microprocesseur a permis à un individu de posséder sa propre machine.

En 1974, quand Intel a sorti le 8080, premier microprocesseur 8 bits généraliste, il a cherché un système d'exploitation, ne serait-ce que pour le tester. Intel a demandé à l'un de ses consultants, Gary Kildall, d'en écrire un. Kildall et l'un de ses amis ont commencé par écrire un contrôleur pour le tout récent lecteur de disquettes 8 pouces de Shugart & associés, et ont adjoint ce lecteur au 8080, créant ainsi le premier micro-ordinateur équipé d'un disque. Kildall a alors écrit un système d'exploitation orienté disque, appelé **CP/M** (*Control Program for Microcomputers*), pour le gérer. Comme Intel pensait que les micro-ordinateurs équipés d'un disque n'avaient que peu de chance de prospérer, quand Kildall a demandé les droits sur CP/M, ceux-ci lui ont été cédés. Kildall a alors fondé une entreprise, Digital Research, pour développer et commercialiser CP/M.

En 1977, Digital Research a réécrit CP/M pour qu'il puisse fonctionner sur les nombreuses plates-formes utilisant le 8080, le Zilog Z80 et d'autres microprocesseurs. De nombreuses applications ont été écrites pour CP/M, faisant de lui l'acteur principal du marché de la micro-informatique pendant cinq ans.

Au début des années 1980, IBM a conçu l'IBM PC et a cherché des logiciels qui puissent fonctionner dessus. Des contacts ont été pris avec Bill Gates pour son interpréteur BASIC. On lui a également demandé s'il connaissait un système d'exploitation pouvant tourner sur le PC. Gates a conseillé aux gens d'IBM de prendre contact avec Digital Research, qui était devenue la plus grosse entreprise au monde dans le

domaine des systèmes d'exploitation. Kildall a refusé, et cela a sans doute été la pire décision stratégique de l'histoire, de rencontrer IBM : il a délégué un de ses collaborateurs à sa place. Pire, son avocat a refusé de signer l'engagement de non-publication relatif au PC, qui n'était pas encore annoncé. IBM s'est donc à nouveau tourné vers Gates et lui a demandé de leur fournir un système d'exploitation.

Suite à cette demande, Gates a appris qu'un petit constructeur de Seattle, *Seattle Computer Products*, possédait un système qui pouvait convenir, **DOS** (*Disk Operating System*). Il leur a proposé de l'acheter (75 000 \$, d'après la rumeur), ce qu'ils ont accepté immédiatement. Gates proposa à IBM un package DOS/BASIC, qu'il accepta. IBM souhaita quelques modifications, aussi Gates a-t-il engagé la personne qui avait écrit DOS, Tim Paterson, dans sa toute nouvelle entreprise, Microsoft. La version modifiée a été baptisée **MS-DOS** (*Microsoft Disk Operating System*) et a très vite dominé le marché de l'IBM PC. Le facteur décisif a été la décision de Gates (*a posteriori* fort judicieuse) de vendre MS-DOS aux constructeurs pour le fournir avec leur matériel, plutôt que – comme l'avait fait Kildall – de le vendre aux particuliers.

Quand l'IBM PC/AT a vu le jour en 1983, équipé du processeur Intel 80286, MS-DOS était solidement implanté et CP/M vivait ses derniers moments. MS-DOS a ensuite été largement utilisé avec le 80386 et le 80486. Même si la première version de MS-DOS était très primitive, les versions suivantes ont inclus des caractéristiques plus sophistiquées, certaines empruntées à UNIX (Microsoft n'ignorait rien du monde UNIX, il a même vendu dans ses premières années une version pour microprocesseur appelée XENIX).

CP/M, MS-DOS et les autres systèmes équipant les premiers micro-ordinateurs étaient tous fondés sur une interface de type texte. Ce schéma a changé suite aux recherches menées par Doug Engelbart, de l'institut de recherches de Stanford, dans les années 1960. Engelbart a inventé le concept d'**IHM graphique** (interface homme-machine) ou **GUI** (*Graphical User Interface*) avec ses fenêtres, ses icônes, ses menus et sa souris. Ses idées ont été reprises par des chercheurs de Xerox PARC et incorporées aux machines qu'ils produisaient.

Un jour, Steve Jobs (le co-inventeur, dans son garage, de l'Apple) a visité le PARC. Lorsqu'il a aperçu une IHM graphique, il a immédiatement pris conscience de la plus-value potentielle qu'elle apportait, ce qui n'a pas été le cas des dirigeants de Xerox. Jobs s'est mis à construire un Apple pourvu d'une IHM graphique. Ce projet a abouti à Lisa, qui était trop chère et a été un échec commercial. Sa seconde tentative, le Macintosh d'Apple, a été un énorme succès, non seulement parce qu'il était vendu beaucoup moins cher que Lisa, mais aussi parce qu'il était convivial, ce qui signifie ici qu'il était destiné à des utilisateurs qui, non seulement ne connaissaient rien aux ordinateurs, mais n'avaient en outre aucunement l'intention d'en apprendre quoi que ce soit.

Quand Microsoft a décidé de construire le successeur de MS-DOS, il a grandement été influencé par le succès du Macintosh. Il a présenté un système à base d'IHM graphique appelé Windows, qui fonctionnait au départ comme une surcouche de

MS-DOS (c'était plus un interpréteur de commandes graphique qu'un véritable système d'exploitation). Pendant une dizaine d'années, entre 1985 et 1995, Windows est resté un environnement graphique au-dessus de MS-DOS. En 1995, une nouvelle version de Windows, appelée Windows 95, a été présentée. Elle incluait bon nombre des caractéristiques des systèmes d'exploitation et n'utilisait plus MS-DOS que pour le démarrage et l'exécution d'anciens programmes. En 1998, une version légèrement modifiée a vu le jour, appelée Windows 98. Ces deux versions contiennent encore cependant une grande quantité de code assembleur Intel 16 bits.

Autre système d'exploitation Microsoft, **Windows NT** (pour *New Technology*) est compatible jusqu'à un certain point avec Windows 95, mais constitue une réécriture totale en interne. C'est un système totalement 32 bits. Le responsable de sa conception, David Cutler, a été l'un des concepteurs du système d'exploitation VMS fonctionnant sur les VAX. Certaines des idées de VMS se retrouvent donc dans NT. Microsoft espérait que la première version de NT suffirait à faire totalement oublier MS-DOS et les versions précédentes de Windows puisque c'était un système de plus haute qualité, mais il en a été pour ses frais. Il lui a fallu attendre NT 4.0 pour que ce rêve commence à prendre forme, particulièrement dans les réseaux d'entreprises. La version 5 de NT a reçu le nom de Windows 2000 au début de l'année 1999. Le but était de succéder à la fois à Windows 98 et à NT. Ce but n'a pas été encore totalement atteint, et Microsoft s'est décidé à sortir une nouvelle version de Windows 98, appelée **Windows Me** (*Millennium Edition*).

En 2001, une version un peu plus moderne de Windows 2000, appelée Windows XP, a été introduite. Cette version a tenu plus de six ans, plus longtemps donc que les précédentes. En 2007, Microsoft a sorti le successeur de XP, appelé Vista, doté d'une nouvelle interface graphique, Aero, et de nombreuses autres nouvelles fonctionnalités. Microsoft espère que Vista remplacera totalement Windows XP, mais cela prendra plusieurs années.

L'autre grand acteur du monde de l'informatique personnelle est UNIX (et ses nombreux clones). UNIX est le plus à sa place sur les stations de travail ou sur les serveurs réseau. Il est aujourd'hui plus particulièrement présent dans les pays en fort développement comme l'Inde ou la Chine. Sur les machines à base de processeurs Pentium, Linux commence à devenir une alternative crédible à Windows pour les étudiants et un nombre croissant d'utilisateurs professionnels. (À propos, nous utiliserons par la suite le terme **Pentium** pour désigner un processeur de la famille Pentium, Pentium II, Pentium III ou Pentium 4, aussi bien que ses successeurs comme Core 2 et Duo. De même, le terme **x86** est lui aussi assez souvent utilisé pour évoquer les processeurs Intel issus du 8086.) FreeBSD est une version dérivée et très répandue d'UNIX. Elle provient de la version BSD développée à Berkeley. Tous les Macintosh récents fonctionnent sous une version modifiée de FreeBSD. UNIX est devenu le standard sur les stations de travail à base des processeurs RISC à hautes performances, comme celles vendues par Hewlett-Packard et Sun Microsystems.

Bien que de nombreux utilisateurs d'UNIX, surtout les plus expérimentés, préfèrent utiliser une interface en ligne de commande plutôt qu'une IHM graphique, pratiquement tous les systèmes UNIX actuels disposent d'un système de fenêtrage appelé

**X Window**, connu également sous l'appellation **X11**, et produit au MIT. Ce système prend en charge les manipulations de base des fenêtres et permet aux utilisateurs de créer, détruire, déplacer ou retailler des fenêtres à l'aide de la souris. Le plus souvent, une IHM complète, comme **Gnome** ou **KDE**, est présente au-dessus de X11 pour donner à UNIX un aspect proche de celui du Macintosh ou de Windows, si l'utilisateur le désire.

Un développement intéressant a débuté vers le milieu des années 1980 : la croissance des réseaux d'ordinateurs personnels fonctionnant sous des **systèmes d'exploitation en réseau** ou des **systèmes d'exploitation distribués**. Dans un système d'exploitation en réseau, les utilisateurs connaissent l'existence individuelle des multiples machines mises à contribution, et peuvent se connecter sur des machines distantes et transférer des fichiers d'une machine à une autre du réseau. Chaque machine fonctionne sous son propre système d'exploitation et gère ses propres utilisateurs.

Les systèmes d'exploitation en réseau ne diffèrent pas en profondeur des systèmes d'exploitation classiques, pour un processeur unique. Ils requièrent la présence d'une interface réseau et de logiciels de bas niveau pour la piloter, ainsi que de programmes permettant la connexion et/ou l'accès aux fichiers à distance, mais cela ne remet pas en cause en profondeur la structure du système.

En revanche, un système d'exploitation distribué apparaît à l'utilisateur identique à un système classique à un processeur. L'utilisateur ne sait pas sur quelle machine s'exécute son programme, ni où se trouvent ses fichiers. Ces détails sont cachés (et gérés efficacement) par le système d'exploitation.

Les véritables systèmes distribués ne se réduisent pas à l'ajout de code à un système classique, car les différences avec les systèmes centralisés sont profondes. Par exemple, un système distribué permet à une application de s'exécuter sur plusieurs processeurs en parallèle, ce qui nécessite des algorithmes d'ordonnancement plus complexes, ne serait-ce que pour optimiser le degré de parallélisme.

Les délais de communication sur le réseau impliquent que ces algorithmes doivent souvent fonctionner avec des informations obsolètes, erronées ou même fausses. Ce contexte est radicalement différent de celui des systèmes monoprocesseurs, où le contrôle de l'état du système est fiable en permanence.

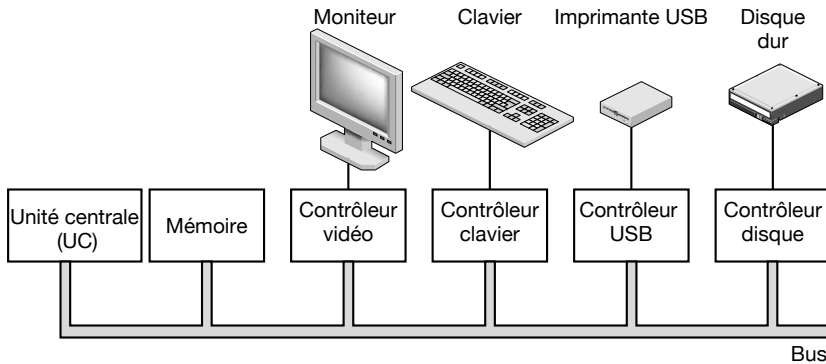
## 1.3 La structure matérielle d'un ordinateur

---

Le système d'exploitation est étroitement lié au matériel de l'ordinateur qu'il fait fonctionner. Il en étend le jeu d'instructions et gère ses ressources. Pour cela, il doit connaître en détail ce matériel, ou tout au moins l'aspect qu'il présente au programmeur. Pour cette raison, il est bon de passer en revue les principaux éléments matériels des ordinateurs personnels. Nous pourrions ensuite commencer plus efficacement l'étude des systèmes d'exploitation : ce qu'ils font et comment ils fonctionnent.



Conceptuellement, un ordinateur personnel simple peut être symboliquement schématisé par la description de la figure 1.6. Le processeur ou l'UC, la mémoire et les périphériques d'E/S sont connectés par un bus système et communiquent entre eux *via* ce bus. Les machines modernes ont une structure plus sophistiquée, composée de plusieurs bus, que nous détaillerons plus loin. Pour l'instant, ce modèle simple est suffisant. Dans les sections qui suivent, nous passons ces différents composants en revue en les montrant sous l'angle d'approche du concepteur de système d'exploitation. Il est utile de dire qu'il ne s'agira que d'un résumé sommaire. De nombreux ouvrages ont été écrits sur le sujet, dont *Architecture de l'ordinateur*, publié par Pearson Education France en 2006.



**Figure 1.6** • Quelques-uns des composants d'un ordinateur personnel simple.

### 1.3.1 Le processeur

Le « cerveau » de l'ordinateur est le processeur ou UC. Il extrait des instructions de la mémoire et les exécute. Le cycle de base de tout processeur est d'extraire la première instruction de la mémoire, la décoder pour connaître son type et ses opérandes, l'exécuter, puis recommencer pour les instructions qui suivent. C'est ainsi qu'un programme s'exécute.

Chaque processeur possède un ensemble spécifique d'instructions exécutables. C'est pourquoi un Pentium ne peut pas exécuter un programme destiné à un SPARC, et *vice versa*. Comme le temps d'accès à la mémoire est très largement supérieur à celui nécessaire pour exécuter une instruction, tous les processeurs contiennent des registres permettant de stocker des variables importantes et des résultats temporaires. L'ensemble des instructions classiques comprend des instructions pour charger un mot mémoire depuis la mémoire dans un registre et pour stocker le contenu d'un registre dans la mémoire. D'autres instructions combinent deux paramètres provenant de la mémoire, de registres ou des deux, et stockent le résultat obtenu en mémoire ou dans un registre.

Outre les registres ordinaires décrits précédemment, la plupart des ordinateurs disposent de registres spéciaux accessibles par le programmeur. L'un d'eux est le **compteur**

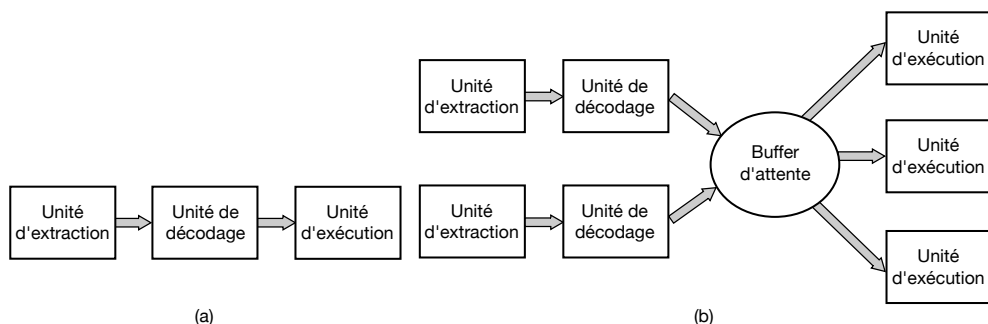
**ordinal** (*program counter*), qui contient l'adresse de la prochaine instruction à extraire de la mémoire. Ce compteur est mis à jour à chaque exécution d'instruction.

Dans un autre registre, on trouve le **pointeur de pile** (*stack pointer*), qui contient l'adresse courante du sommet de pile en mémoire. Cette pile contient un enregistrement par procédure dans laquelle le programme est entré et d'où il n'est pas encore ressorti. Chaque enregistrement contient les paramètres d'entrée, variables locales et temporaires de la procédure qui ne sont pas stockées dans des registres.

Un autre registre contient le **mot d'état** du programme (**PSW**, *Program Status Word*). Il comprend des bits de condition (positionnés par des instructions de comparaison), la priorité de l'UC, le mode (utilisateur ou noyau) et d'autres bits de contrôle. Les programmes utilisateur peuvent en principe lire le contenu entier de ce mot, mais n'ont accès en écriture qu'à un sous-ensemble de ses champs. Le PSW joue un rôle très important dans les appels système et les E/S.

Le système d'exploitation doit connaître l'ensemble des registres. Lors du multiplexage temporel du processeur, le système d'exploitation interrompra fréquemment un programme pour en démarrer un autre. À chaque interruption, le système doit sauvegarder tous les registres afin qu'ils soient restaurés quand le programme pourra à nouveau reprendre son exécution.

Pour améliorer les performances, les concepteurs de processeurs ont depuis longtemps abandonné ce modèle simple extraction-décodage-exécution en séquence. De nombreux processeurs modernes ont des caractéristiques qui leur permettent d'exécuter plus d'une instruction à la fois. Ainsi, une même UC peut posséder des unités séparées pour l'extraction, le décodage et l'exécution, de façon à pouvoir décodifier l'instruction  $n+1$  et extraire l'instruction  $n+2$  pendant l'exécution de l'instruction  $n$ . Une telle structure est nommée **pipeline** et est décrite à la figure 1.7(a) dans le cas d'un pipeline à 3 niveaux ou étages. De plus longs pipelines ne sont pas rares. Dans la plupart des conceptions de pipelines, une fois l'instruction chargée dans le pipeline, elle est exécutée même si l'instruction précédente était un branchement conditionnel. La présence de pipelines rend la conception des compilateurs et des systèmes d'exploitation beaucoup plus compliquée.



**Figure 1.7** • (a) Un pipeline à trois niveaux. (b) Un processeur superscalaire.

Autre degré de sophistication au-dessus du pipe-line : les processeurs **superscalaires**, décrits à la figure 1.7(b). Dans cette approche, on dispose de plusieurs unités d'exécution, par exemple une pour l'arithmétique entière, une pour l'arithmétique flottante et une pour les opérations logiques. Deux ou plusieurs opérations sont extraites à la fois, décodées, et placées dans un buffer d'attente (une mémoire tampon) jusqu'au moment de leur exécution. Dès qu'une unité d'exécution est libre, elle consulte le buffer pour voir si une instruction de type correspondant est en attente ; si c'est le cas, elle l'extrait du buffer et l'exécute. La conséquence de cette approche est que les instructions sont souvent exécutées dans le désordre. La plupart du temps, il revient principalement au matériel de vérifier que le résultat obtenu est équivalent à celui fourni par une architecture séquentielle, mais une partie non négligeable (et complexe) du travail échoit tout de même au système d'exploitation, comme nous le verrons par la suite.

La plupart des processeurs, mis à part les plus simples d'entre eux, utilisés dans les systèmes embarqués ont deux modes de fonctionnement : le mode utilisateur et le mode noyau. Ce mode est en général mémorisé dans un des bits du PSW. En mode noyau, le processeur peut exécuter n'importe quelle instruction de son jeu et utiliser toutes les caractéristiques du matériel sous-jacent. Le système d'exploitation tourne en mode noyau, ce qui lui donne accès à l'ensemble du matériel.

En revanche, les programmes utilisateur tournent en mode utilisateur, qui ne permet l'accès qu'à un sous-ensemble des instructions et des ressources de la machine. En général, toutes les instructions relatives aux E/S et à la protection mémoire sont inaccessibleles en mode utilisateur. Il est aussi évidemment interdit de basculer sur « noyau » le bit de mode du PSW.

Pour accéder aux services offerts par le système d'exploitation, un programme utilisateur doit effectuer un **appel système**, qui bascule en mode noyau et invoque le système d'exploitation. L'instruction *trap* effectue cette bascule. Quand le travail du système d'exploitation est terminé, le contrôle est rendu au programme utilisateur pour l'instruction suivante. Nous expliquerons les détails de gestion des appels système plus loin dans ce chapitre. Détail typographique : nous utiliserons une police à espacement fixe pour les appels système, comme dans le cas de *read*.

Il faut noter que les ordinateurs ont d'autres déroutements que l'instruction *trap* permettant d'effectuer un appel système. La plupart d'entre eux sont déclenchés par le matériel pour avertir d'une situation imprévue, comme par exemple une tentative de division par 0 ou un débordement de pile. Dans tous les cas, le système d'exploitation prend le contrôle et décide de la conduite à adopter. Parfois, le programme se termine sur une erreur. D'autres fois, l'erreur peut être ignorée (par exemple, un nombre flottant inférieur à la précision minimale peut être mis à 0). Enfin, quand le programme a annoncé au départ qu'il souhaitait prendre en charge certaines conditions exceptionnelles, le contrôle peut lui être donné le cas échéant.

## Technique du multithread et circuits intégrés multicœurs

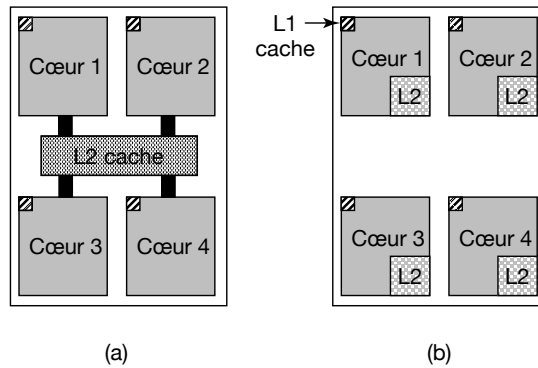
La **loi de Moore** affirme que le nombre de transistors d'un circuit intégré double tous les 18 mois. Cette loi n'est pas une loi physique comme la conservation de la quantité de mouvement, mais une loi qui résulte de l'observation faite par Gordon Moore, le cofondateur de la société Intel. Moore a observé de combien les ingénieurs concepteurs des circuits intégrés étaient capables de réduire la taille des transistors et ainsi d'en augmenter leur nombre sur une même surface de silicium. La loi de Moore a été vérifiée pendant plus de 30 ans. Elle devrait encore tenir pendant une dizaine d'années.

L'abondance de transistors est source de problème : que faire avec eux ? Nous avons vu ci-dessus une approche qui peut être un élément de réponse : l'architecture superscalaire dotée de plusieurs unités fonctionnelles. D'autres réponses sont possibles, comme créer de volumineux caches dans les processeurs. C'est ce qui s'est passé, mais finalement on a atteint le point de rendement décroissant. Alors que faire de plus ?

Une autre approche est la duplication ; non seulement d'unités fonctionnelles mais également de l'unité de commande. Le Pentium 4 et d'autres processeurs ont mis en place cette technique qui a donné lieu au **multithreading** ou **hyperthreading** (nom donné par Intel à cette technique). En première approximation, cette technique consiste à permettre à l'UC de maintenir les états de deux threads différents et ainsi de pouvoir passer de l'un à l'autre en un temps très court, de l'ordre de la nanoseconde. Un **thread** est une sorte de processus léger, qui correspond à un programme exécutable par le processeur. (Nous étudions en détail les threads au chapitre 2.) Par exemple si l'un des processus (ou thread) a besoin de lire un mot mémoire (ce qui peut prendre quelques cycles d'horloge), une UC dotée de la technique de multithreading peut passer très rapidement à l'autre processus pour éviter l'attente. Ce qui a pour effet d'améliorer les performances. Même si elle s'en approche, il ne faut toutefois pas confondre cette technique avec le parallélisme car en multithreading un seul processus est exécuté à la fois.

Le multithreading a une forte conséquence sur le système d'exploitation, car chaque thread lui apparaît comme une UC séparée. Considérons par exemple un système qui comprend deux UC, chacune traitant deux threads. Le système d'exploitation voit cela comme quatre UC. S'il y a suffisamment de travail pour maintenir les deux UC occupées à un instant donné, le système d'exploitation peut, par mégarde, faire exécuter les deux threads sur la même UC, laissant la seconde UC oisive. Ce choix est bien moins efficace que de faire exécuter un thread par chaque UC. Le successeur du Pentium 4, le Core 2, ne pratique pas la technique de l'hyperthreading, mais Intel a annoncé que le successeur du Core 2 la mettrait en œuvre.

Au-delà du multithreading on trouve des UC qui disposent de deux ou quatre processeurs intégrés complets, voire plus. On dit de ces circuits qu'ils sont **multicœurs**. Le circuit multicœur de la figure 1.8 comprend quatre processeurs, chacun disposant d'une UC indépendante. (Nous expliquons le rôle des caches ci-dessous.) L'utilisation de ces circuits multicœurs nécessite impérativement un système d'exploitation multiprocesseur.

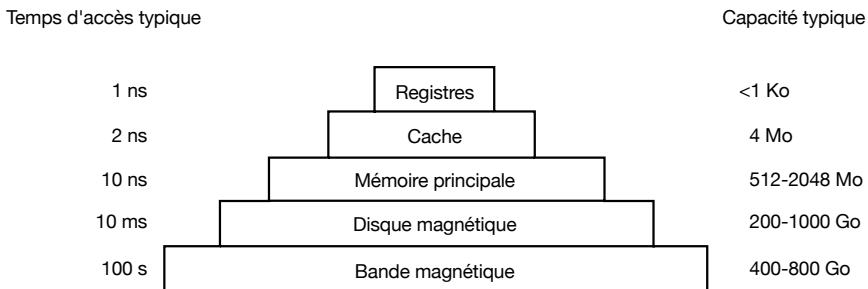


**Figure 1.8** • (a) Un circuit à quatre cœurs avec un cache L2 partagé. (b) Un circuit à quatre cœurs avec des caches L2 indépendants.

### 1.3.2 La mémoire

Le second constituant fondamental de tout ordinateur est la mémoire. Dans l'idéal, la mémoire devrait être extrêmement rapide (plus rapide que le temps d'exécution d'une instruction, de façon à ce que l'UC ne soit pas freinée par les accès mémoire), disponible en grande quantité et peu onéreuse. Aucune des technologies actuelles ne satisfait ces trois critères, il faut donc biaiser. La mémoire est construite comme une hiérarchie de couches, comme le montre la figure 1.9. Les mémoires de la couche supérieure sont très rapides, de faible capacité et présentent des coûts par bit bien plus élevés que celles des couches inférieures.


La couche supérieure est constituée des registres internes à l'UC. Ils sont fabriqués dans les mêmes matériaux que le processeur et sont donc du même niveau de performance. Leur capacité de stockage est de l'ordre de  $32 \times 32$  bits (sur un processeur 32 bits) ou  $64 \times 64$  bits (sur un processeur 64 bits). Elle est en tout cas inférieure à 1 Ko. Les programmes gèrent les registres eux-mêmes (c'est-à-dire décident de ce qu'ils y stockent).



**Figure 1.9** • Un découpage hiérarchique classique de la mémoire. Les valeurs mentionnées sont approximatives.

Vient ensuite la **mémoire cache** (ou simplement *cache*), principalement contrôlée par le matériel. Un *cache* est une mémoire rapide de petite taille qui contient les mots mémoire les plus récemment utilisés, accélérant ainsi l'accès à ces mots. Un *cache* est divisé en **lignes de cache**, le plus souvent de 64 octets chacune, adressées de manière contiguë (0 à 63 pour la première, 64 à 127 pour la deuxième, etc.). Les lignes les plus fréquemment utilisées sont stockées dans un *cache* à très hautes performances, généralement situé tout près de l'UC (voire à l'intérieur même de celle-ci). Quand le programme a besoin d'accéder à un mot mémoire, le matériel vérifie que la ligne demandée est dans le *cache*. Si c'est le cas, on parle de réussite de lecture ou **cache hit**. La requête étant satisfaite par le *cache*, aucun accès à la mémoire principale n'est effectué. Un *cache hit* prend en général deux cycles d'horloge. Si la ligne n'est pas dans le *cache*, on parle d'échec de lecture ou **cache miss** ; on doit alors accéder à la mémoire principale, au prix d'une perte de temps non négligeable. La taille du *cache* est limitée par le prix élevé de ce type de mémoire. Certaines machines ont deux, voire trois niveaux de *cache*, chaque niveau étant plus lent mais de taille plus importante que le précédent.

Les *caches* jouent un rôle essentiel dans les systèmes informatiques, et pas seulement pour améliorer les performances entre le processeur et la mémoire principale. Mais, par exemple, à chaque fois qu'on peut diviser une ressource en plusieurs éléments (ou blocs) et que certains éléments sont plus fréquemment utilisés que d'autres, le *cache* apparaît comme améliorant les performances. Les systèmes d'exploitation utilisent sans cesse des *caches*. C'est ainsi que la plupart conservent en mémoire centrale les blocs des fichiers les plus souvent référencés plutôt que de les extraire du disque à maintes reprises. C'est une technique de type *cache*. De façon similaire, le résultat de la conversion d'un long chemin d'accès comme

 /home/ast/projects/minix3/src/kernel/clock.c

en une adresse disque où se trouve le fichier référencé peut être mise en *cache* pour éviter de refaire la conversion à chaque utilisation. De même, lorsque l'adresse d'une page Web (l'URL) est convertie en adresse IP, cette dernière peut être conservée en *cache* pour une utilisation future.

Dès qu'on utilise un système de *cache*, plusieurs questions se posent :

1. Quand mettre un nouvel item dans le *cache* ?
2. Dans quelle ligne de *cache* insérer le nouvel item ?
3. Quel item sortir du *cache* quand on a besoin de place pour en mettre un nouveau ?
4. Où ranger l'item expulsé du *cache* en mémoire ?

Chaque question ne concerne pas forcément tous les types de *caches*. Dans le *cache* d'une UC, un item est systématiquement inséré dans la ligne du *cache* à chaque fois qu'une situation d'échec de lecture (*cache miss*) apparaît. La ligne de *cache* à utiliser est souvent calculée à partir de quelques bits de poids fort de l'adresse mémoire référencée. Par exemple avec un *cache* de 4 096 lignes de 64 octets et 32 bits d'adresse, les bits 6 à 17 de l'adresse peuvent être utilisés pour déterminer la ligne du *cache*, et les bits 0 à 5 pour la position de l'octet dans la ligne. Dans ce cas, l'item à

enlever est celui qui est rangé à la position que l'on va occuper, mais dans d'autres systèmes ce ne serait pas si simple. Cependant, quoi qu'il en soit, lorsqu'on réécrit une ligne de cache en mémoire, l'emplacement mémoire de réécriture est uniquement déterminé par l'adresse en question.

La plupart des UC actuelles ont deux niveaux de caches. Le premier niveau, L1, est interne au processeur et comprend le plus souvent deux caches, un cache d'instructions et un cache de données. Chaque cache fait en général 16 Ko. Il y a, en outre, très souvent un cache de second niveau, L2, entre les caches de niveau 1 et la mémoire principale. Sa capacité est de l'ordre de quelques mégaoctets. Il contient les mots mémoire les plus récemment utilisés. La différence entre les deux niveaux tient au facteur temps : avec L1 les accès se font sans délai alors qu'il faut un ou deux cycles d'horloge avec L2.

À la figure 1.8(a), chaque cœur compte un cache L1 privé et peut accéder à un cache L2 partagé par tous. C'est l'approche utilisée par les circuits multicœurs d'Intel. En revanche, à la figure 1.8(b), chaque cœur dispose de deux caches privés, L1 et L2. C'est l'approche utilisée par AMD. Chaque stratégie a ses partisans et ses détracteurs. L'approche d'Intel nécessite un contrôleur de cache L2 plus complexe que dans celle retenue par AMD, qui au contraire rend plus complexe la garantie de cohérence des caches L2.

On trouve ensuite sur la hiérarchie de la figure 1.9 la mémoire principale, point central du système de mémoire. Elle est souvent appelée mémoire **RAM** (*Random Access Memory*). Les anciens la nomment souvent *core memory* (mémoire noyau), car les ordinateurs des années 1950 et 1960 utilisaient de minuscules noyaux de ferrite magnétisable pour la mémoire principale. À l'heure actuelle, la capacité de ces mémoires s'étend de quelques centaines de mégaoctets à plusieurs gigaoctets et croît rapidement. Toutes les requêtes non satisfaites par les accès aux caches sont reportées ensuite vers la mémoire principale.

En plus de la mémoire RAM, qui est volatile, la plupart des ordinateurs disposent d'une quantité relativement faible de mémoire non volatile. Celle-ci, contrairement à la RAM, est une mémoire qui ne perd pas ses données lorsque l'alimentation électrique est coupée. Il s'agit de la mémoire **ROM** (*Read Only Memory*) qui ne peut être que lue. Programmé une fois pour toutes lors de sa fabrication en usine, le contenu d'une ROM ne peut pas être modifié. Comme la RAM, elle est très rapide et peu coûteuse. En général, on trouve dans ces mémoires ROM le *bootstrap* qui est le programme d'amorçage servant à démarrer la machine. De même, certains programmes de gestion de périphériques se trouvent eux aussi en mémoire ROM, sur des cartes d'entrées/sorties.

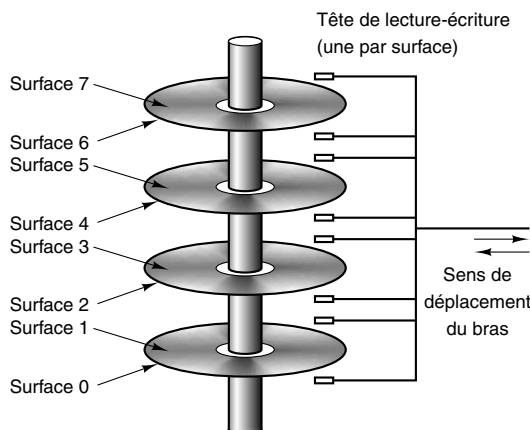
Il existe d'autres mémoires non volatiles : les **EEPROM** (*Electrically Erasable programmable ROM*, mémoire en lecture seule mais programmable après effacement électrique) et les **mémoires Flash** qui, contrairement aux ROM, peuvent être effacées et reprogrammées plusieurs fois. Le temps d'écriture dans ces mémoires étant beaucoup plus important que dans les mémoires RAM, on les utilise comme des mémoires ROM.

Les mémoires Flash sont très utilisées comme mémoire de stockage de masse sur divers appareils électroniques, comme les caméscopes ou les lecteurs MP3. Leurs performances sont intermédiaires entre la RAM et le disque dur. Toutefois, contrairement aux disques, si elles sont effacées beaucoup trop souvent elles finissent par s'user et présenter des défaillances.

La **mémoire CMOS** est une autre catégorie de mémoire volatile, mais à très faible consommation, fort utilisée dans les ordinateurs pour conserver l'heure et la date courantes. La mémoire CMOS et le circuit d'horloge sont alimentés par une petite batterie permettant le maintien de l'heure et du jour même lorsque l'ordinateur est hors tension. La mémoire CMOS permet également de conserver des paramètres de configuration, par exemple ceux de périphériques. L'avantage de ce type de mémoire est sa très faible consommation électrique : les batteries d'usine fonctionnent fréquemment pendant plusieurs années. Cependant, lorsqu'elles commencent à faiblir, le comportement de l'ordinateur peut devenir étrange, comme s'il oubliait des choses pourtant bien connues auparavant.

### 1.3.3 Les disques

L'étape suivante dans la hiérarchie de la figure 1.9, c'est le disque magnétique ou disque dur. Le coût du stockage sur disque est 100 fois inférieur à celui de la mémoire RAM et sa capacité de stockage 100 fois supérieure. Le seul inconvénient est le temps d'accès aux données, de trois ordres de grandeur plus long que dans le cas des accès RAM, en raison de contraintes mécaniques, comme le montre la figure 1.10.



**Figure 1.10** • Structure d'un disque.

Un disque est constitué de un ou plusieurs plateaux tournant à 5 400, 7 200 ou 10 800 tours/min. Un bras mécanique se déplace au-dessus des plateaux, à la manière du bras des anciens électrophones. Le bras compte plusieurs têtes de lecture-écriture. L'information est écrite sur le disque en une série de cercles concentriques. À une



position donnée du bras, chaque tête peut lire une zone circulaire appelée **piste**. L'ensemble des pistes accessibles à une position donnée s'appelle un **cylindre**.

Chaque piste est divisée en secteurs, d'une taille de l'ordre de 512 octets. Sur les disques modernes, les cylindres extérieurs contiennent davantage de secteurs que les cylindres intérieurs. Le déplacement du bras d'un cylindre au suivant prend environ 1 ms. Le déplacement vers un cylindre quelconque est de l'ordre de 5 à 10 ms. Une fois le bras sur la piste souhaitée, on doit attendre que la tête se trouve au-dessus du secteur concerné, ce qui prend également un temps de l'ordre de 5 à 10 ms. Enfin, la lecture (ou l'écriture) effective se fait à un débit allant de 50 Mo/s (pour les plus lents) à 160 Mo/s (pour les plus rapides).

De nombreux ordinateurs mettent en œuvre la technique de mémoire virtuelle que nous étudions en détail au chapitre 3. Cette technique rend possible l'exécution d'un programme plus gros que la capacité physique de la mémoire principale. Le programme est enregistré sur le disque. La mémoire principale est alors utilisée comme un cache qui contient les parties du programme les plus fréquemment utilisées. Cela nécessite une conversion à la volée des adresses virtuelles générées par le programme en adresses physiques (réelles) en mémoire principale. Cette conversion d'adresses est réalisée par un circuit spécialisé de gestion mémoire appelé MMU (*Memory Management Unit*), comme le montre la figure 1.6. Ce MMU se trouve sur le processeur lui-même ou très près mais, logiquement, il est placé entre l'UC et la mémoire.

La présence d'un cache et d'un MMU a un impact majeur sur les performances. Dans les systèmes multiprogrammés, quand il faut passer d'un programme à l'autre (on parle de **changement de contexte**), il peut être nécessaire de vider tous les blocs modifiés du cache et changer le registre de mappage du MMU. Ces deux actions sont coûteuses et complexes à réaliser. Nous verrons plus loin dans l'ouvrage comment les réaliser efficacement.

### 1.3.4 Les bandes magnétiques

La dernière couche de la hiérarchie des mémoires est constituée des bandes magnétiques. Ce médium est principalement utilisé pour l'archivage des données des disques et pour le stockage de très grandes quantités de données. Pour accéder à une bande, il faut d'abord la placer dans un lecteur, manuellement ou à l'aide d'un robot (installations fréquentes dans les cas de manipulations de grandes bases de données). La bande doit alors être déroulée rapidement jusqu'à l'endroit où se trouve le bloc recherché. Le temps nécessaire à ces deux opérations se chiffre en minutes. Les gros avantages de la bande sont son coût à l'octet extraordinairement faible et son caractère extractible, ce qui permet un stockage hors site favorisant la sécurité en cas d'accident (incendie, inondation, etc.).

La hiérarchie que nous venons de décrire est représentative, mais toutes les installations ne possèdent pas tous les niveaux cités, ou en possèdent de différents (par exemple, des disques optiques). Dans tous les cas cependant, quand on descend dans la hiérarchie, le temps d'accès s'accroît considérablement, la taille des informations stockables également, et le coût à l'octet baisse de façon tout aussi spectaculaire.

Il est donc probable que cette hiérarchie ne soit pas bouleversée en profondeur dans les années qui viennent.

### 1.3.5 Les périphériques d'E/S

La mémoire n'est pas la seule ressource que le système d'exploitation ait à gérer. Les périphériques d'E/S lui sont également étroitement connectés. Comme nous l'avons vu à la figure 1.6, les périphériques d'E/S se présentent généralement en deux parties : un contrôleur et le périphérique lui-même. Le contrôleur est une puce (ou un petit ensemble de composants) sur une carte qui contrôle physiquement le périphérique. Il reçoit des commandes du système d'exploitation, par exemple pour lire des données, et les exécute.

Dans de nombreux cas, le contrôle effectif du périphérique est très complexe, aussi le contrôleur doit-il présenter une interface simplifiée au système d'exploitation. Ainsi, un contrôleur disque peut accepter une commande pour lire le secteur 11 206 du disque 2. Le contrôleur doit convertir ce numéro linéaire de secteur en un triplet cylindre/secteur/tête. Cette conversion peut être compliquée par le fait que les cylindres peuvent comporter un nombre variable de secteurs et par la présence de secteurs défectueux. Il doit alors déterminer sur quel cylindre se trouve le bras et lui indiquer de se déplacer du nombre correspondant de cylindres. Il doit attendre que le secteur recherché se trouve sous la tête de lecture avant de lire les bits et de les stocker au fur et à mesure de leur lecture, après avoir laissé de côté le préambule et calculé une somme de contrôle. Il doit enfin assembler les bits en mots pour les stocker dans la mémoire. Pour mener à bien ce type de tâche, les contrôleurs sont souvent équipés de petits ordinateurs embarqués programmés spécialement pour cela.

L'autre composant est le périphérique lui-même. Il possède en général une interface plutôt simple, à la fois parce qu'il n'assure pas beaucoup de fonctionnalités et afin de préserver une certaine homogénéité. Cette dernière propriété est nécessaire si l'on veut que tout contrôleur **IDE** (*Integrated Drive Electronics*, le type de disque le plus courant sur les systèmes à base de Pentium, entre autres) puisse piloter n'importe quel disque dur IDE. Comme la véritable interface du périphérique est cachée derrière le contrôleur, tout ce que voit le système d'exploitation c'est l'interface du contrôleur, qui peut être très différente.

Comme chaque type de contrôleur est spécifique, du logiciel spécifique est requis pour le contrôle de chacun d'eux. Le logiciel qui communique avec le contrôleur se nomme **pilote de périphérique** (*device driver*). Chaque fabricant de contrôleur doit fournir un pilote pour chaque système d'exploitation qu'il déclare supporter. Ainsi, un scanner peut être accompagné de pilotes pour Windows 98, Windows 2000, Windows XP, Vista et Linux par exemple.

Pour être actif, le pilote doit être placé dans le système d'exploitation afin de fonctionner en mode noyau. En théorie, les pilotes peuvent fonctionner en dehors du noyau mais peu de systèmes utilisent cette possibilité, parce que cela réclame la possibilité d'autoriser un « programme utilisateur » à accéder au périphérique de manière contrôlée, une caractéristique rarement disponible. Il y a trois façons de placer le

pilote dans le noyau. La première méthode consiste à refaire une édition de liens du noyau en y incorporant le code du nouveau pilote et de redémarrer la machine. De nombreux systèmes UNIX fonctionnent ainsi. La seconde méthode consiste à placer une nouvelle entrée dans un fichier spécial indiquant que le système a besoin de ce pilote, puis là aussi de redémarrer le système. À l'amorçage, le système d'exploitation cherche et charge les pilotes nécessaires. C'est le mode de fonctionnement de Windows. La troisième méthode nécessite un système d'exploitation capable d'intégrer un nouveau pilote à la volée, sans nécessiter de redémarrage. Elle était très peu répandue mais devient de plus en plus commune. Les périphériques « *hot-plug* » (connectables à chaud), comme ceux des familles USB et IEEE 1394, décrites plus loin, nécessitent de tels pilotes chargés dynamiquement.

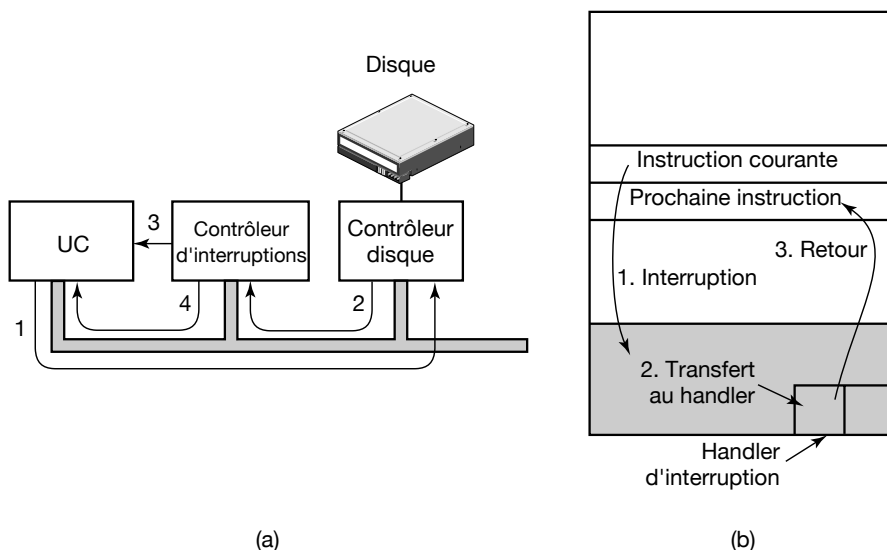
Chaque contrôleur possède un petit nombre de registres utilisables pour communiquer avec lui. Ainsi, un contrôleur disque minimal comprendrait des registres pour renseigner l'adresse disque, l'adresse mémoire, le numéro de secteur et le type d'opération (lecture ou écriture). Pour activer le contrôleur, le pilote reçoit une commande du système d'exploitation et transfère les paramètres de la commande dans les registres correspondants du périphérique. L'ensemble de ces registres constitue l'espace des ports d'E/S, un sujet que nous reprenons en détail au chapitre 5.

Sur certains ordinateurs, les registres de périphériques sont mappés sur l'espace d'adressage du système d'exploitation, et peuvent donc être accédés comme n'importe quel mot mémoire. Sur ces machines, aucune instruction spéciale d'E/S n'est nécessaire et les programmes utilisateur peuvent être maintenus hors d'état d'accéder directement au périphérique simplement en configurant leurs registres de base et de limite de sorte que ces adresses soient hors de la région accessible. Sur d'autres, les registres d'E/S sont placés dans un espace spécial de ports d'E/S, chaque registre correspondant à une adresse de port. Sur ces machines, des instructions spéciales IN et OUT fonctionnant en mode noyau sont disponibles pour permettre aux pilotes de lire ou d'écrire les registres. Le premier modèle élimine le besoin d'instructions spécifiques, mais au prix d'une consommation d'espace d'adressage. Le second économise la mémoire mais nécessite des instructions particulières. Les deux modèles sont largement utilisés.

La lecture et l'écriture peuvent s'effectuer de trois façons différentes. La méthode la plus simple consiste pour un programme à faire un appel système, que le noyau traduit en un appel à une procédure du pilote approprié. Le pilote démarre alors l'E/S, puis boucle en attente sur le résultat de l'opération (en général, un bit indique que le périphérique est occupé). Quand celle-ci se termine, le pilote range le cas échéant les informations là où elles sont attendues, puis termine. Le système d'exploitation rend alors le contrôle à l'appelant. Cette méthode se nomme **attente active** et présente l'inconvénient d'occuper l'UC à ne rien faire pendant qu'elle attend la fin de l'opération du périphérique.

La seconde méthode consiste à demander au périphérique de lancer une interruption quand il termine l'opération d'E/S. Ensuite, le pilote termine. Le système d'exploitation bloque alors l'appelant et cherche autre chose à faire. Quand le contrôleur détecte la fin du transfert, il génère l'interruption.

Les **interruptions** sont fondamentales dans le fonctionnement d'un système d'exploitation, et nous allons les examiner d'un peu plus près afin de mieux en comprendre le principe. La figure 1.11(a) représente le traitement d'une E/S en trois étapes. À l'étape 1, le pilote informe le contrôleur de ce qu'il doit faire en positionnant les registres du périphérique. Le contrôleur démarre alors le périphérique. Lorsqu'il a terminé de lire ou d'écrire le nombre d'octets demandé, le contrôleur le signale au contrôleur d'interruptions à l'aide de certaines lignes du bus ; c'est la seconde étape. Si le contrôleur d'interruptions est prêt à recevoir une interruption (ce qui n'est pas toujours le cas, il peut être occupé par une tâche de plus haute priorité), il informe l'UC de la réception de l'interruption sur une broche particulière (étape 3). À l'étape 4, le contrôleur d'interruptions place le numéro du périphérique sur le bus pour que l'UC sache quel périphérique vient de terminer une opération d'E/S (de nombreux périphériques travaillent en général en parallèle).



**Figure 1.11** • (a) Étapes du démarrage d'un périphérique d'E/S et réception d'une interruption. (b) La gestion d'une interruption implique l'interception de l'interruption, le déclenchement du gestionnaire d'interruptions et le retour au programme utilisateur.

Quand l'UC a décidé de prendre en charge l'interruption, le compteur ordinal et le PSW sont empilés et l'UC passe en mode noyau. Le numéro de périphérique peut être utilisé pour indexer une zone de mémoire permettant de trouver l'adresse du gestionnaire d'interruptions pour ce périphérique. Cette zone de mémoire est appelée **vecteur d'interruption**. Le gestionnaire d'interruptions (une partie du pilote), une fois démarré, dépile le compteur ordinal et le PSW et les sauvegarde, puis interroge le périphérique pour connaître son état. Quand il termine, il rend le contrôle au programme qui était en cours, au niveau de la prochaine instruction qu'il devait exécuter. Ces étapes sont décrites à la figure 1.11(b).

La troisième technique de réalisation d'une E/S utilise une puce spéciale, dite **DMA** (*Direct Memory Access*, accès direct à la mémoire), qui contrôle le flux de bits entre la mémoire et un contrôleur quelconque sans nécessiter d'intervention de l'UC. Cette dernière initialise la puce DMA en lui donnant le nombre d'octets à transférer, le périphérique et l'adresse mémoire concernés et le sens de transfert. Quand la puce a terminé son travail, elle provoque une interruption. Le DMA et le matériel spécifique aux E/S sont décrits en détail au chapitre 5.

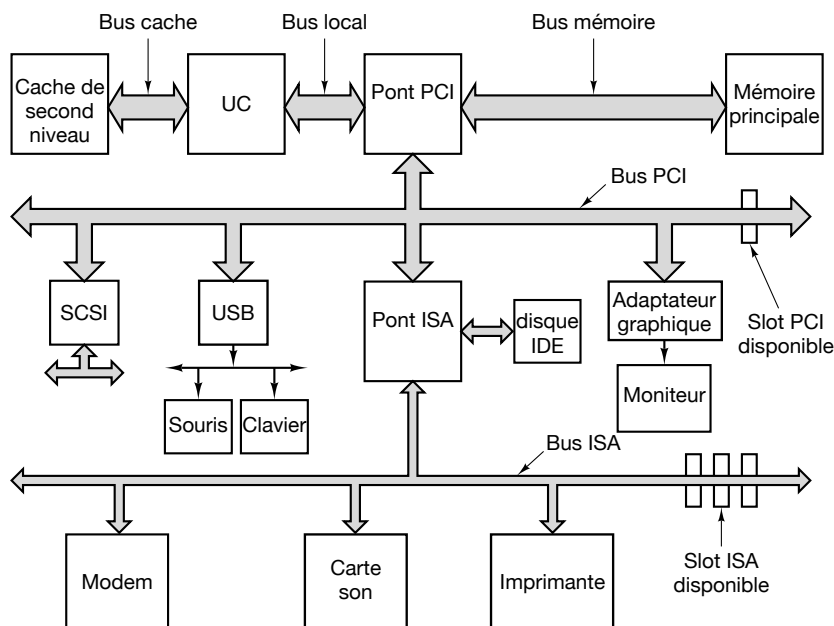
Les interruptions arrivent souvent au plus mauvais moment, par exemple pendant qu'un autre gestionnaire d'interruptions s'exécute. C'est pourquoi l'UC dispose d'un moyen pour désactiver et réactiver les interruptions. Quand elles sont désactivées, les interruptions continuent d'être signalées par les divers périphériques mais ne sont plus prises en compte par l'UC. Quand celle-ci les réactive, le gestionnaire d'interruptions choisit, parmi celles qui se seraient signalées pendant la désactivation, celle qui doit être traitée d'abord. Ce choix repose en général sur un système de priorités affectées de manière statique aux différents périphériques. C'est celui qui a la plus haute priorité qui gagne.

### 1.3.6 Les bus

L'organisation décrite à la figure 1.6 a été celle des mini-ordinateurs pendant des années, ainsi que celle des premiers IBM PC. Cependant, avec l'augmentation des performances des processeurs et des mémoires, un unique bus n'a rapidement plus été suffisant pour assurer le trafic. On a dû avoir recours à des bus supplémentaires, tant pour réaliser des E/S performantes que pour accroître le débit entre le processeur et la mémoire. De fait, un système moderne à base de Pentium ressemble davantage à la figure 1.12. Ce système possède huit bus (cache, local, mémoire, PCI, SCSI, USB, IDE et ISA), chacun étant dévolu à un rôle particulier et doté de performances en rapport. Le système d'exploitation doit les connaître tous, tant pour la configuration du système que pour la gestion. Les deux principaux bus sont le bus d'origine des PC IBM (le bus **ISA**, *Industry Standard Architecture*) et son successeur (le bus **PCI**, *Peripheral Component Interconnect*). Le bus ISA, qui équipait les PC/AT au départ, fonctionne à 8,33 MHz et peut transférer deux octets à la fois, ce qui donne un taux de transfert maximal de 16,67 Mo/s. Il n'est plus présent que pour des raisons de compatibilité avec d'anciennes cartes d'E/S. Le bus PCI a été mis au point par Intel pour succéder au bus ISA. Il peut atteindre 66 MHz et transfère 8 octets à la fois, ce qui donne un débit maximal de 528 Mo/s. La plupart des périphériques d'E/S modernes utilisent ce bus de nos jours. On trouve même des ordinateurs non équipés de processeurs Intel et néanmoins pourvus d'un bus PCI, en raison du grand nombre de cartes d'E/S disponibles pour celui-ci. Les PC les plus modernes utilisent une version nouvelle du bus PCI appelé **PCI Express**.

Dans cette configuration, le processeur dialogue avec une puce spéciale appelée pont PCI sur le bus local, et le pont PCI avec la mémoire à l'aide d'un bus mémoire dédié fonctionnant fréquemment à 100 MHz. Les systèmes Pentium possèdent un cache de

premier niveau sur le processeur et un cache de second niveau, beaucoup plus grand, à l'extérieur, connecté à l'UC par le bus cache.



**Figure 1.12** • Structure d'un gros système Pentium.

En outre, ce système à base de Pentium utilise trois bus spécialisés : IDE, USB et SCSI. Le bus **IDE** permet de connecter au système des périphériques comme des disques ou des CD-ROM. C'est une évolution de l'interface de contrôle des disques du PC/AT, qui est devenue le standard sur pratiquement toutes les plates-formes Pentium pour les disques et, fréquemment, les CD-ROM.

Le bus **USB** (*Universal Serial Bus*) a été créé pour connecter les périphériques lents (comme le clavier ou la souris) à l'ordinateur. Il utilise un petit connecteur à quatre câbles, dont deux servent à l'alimentation électrique des périphériques USB. Il s'agit d'un bus série (transmission en série) à gestion centralisée, dans lequel un périphérique maître consulte toutes les millisecondes chaque périphérique sur le bus pour détecter un trafic éventuel. Le trafic total maximal est de 1,5 Mo/s. Tous les périphériques partagent le même pilote, ce qui rend superflue l'installation d'un pilote particulier lors de la connexion d'un nouveau périphérique USB. Autre conséquence intéressante, il n'est plus nécessaire de redémarrer la machine lors de l'ajout d'un périphérique.

Le bus **SCSI** (*Small Computer System Interface*) est un bus à hautes performances destiné aux périphériques nécessitant une large bande passante comme les disques, les scanners, etc. Il peut fonctionner à des taux de transfert atteignant 160 Mo/s.

Il figurait sur les premiers Macintosh, mais il est également très populaire dans le monde UNIX et sur certains systèmes à base Intel.

Un autre bus, qui ne figure pas sur la figure 1.12, est le bus **IEEE 1394**. On l'appelle parfois bus **Firewire**, même s'il s'agit du nom qu'Apple utilise pour sa propre implémentation de 1394. Comme le bus USB, il s'agit d'un bus série mais dont les taux de transfert peuvent atteindre 100 Mo/s, ce qui le rend apte à connecter des périphériques multimédias comme les caméscopes numériques. Cependant, à la différence du bus USB, le bus IEEE 1394 ne possède pas de contrôle centralisé. Pour fonctionner dans un environnement comme celui représenté à la figure 1.12, le système d'exploitation doit connaître tout ce qui constitue cet environnement et le configurer. Cette contrainte a conduit Intel et Microsoft à concevoir une technologie applicable au PC appelée **plug and play** (branchez et ça marche), à partir d'une idée similaire déjà implémentée sur le Macintosh d'Apple. Avant le plug and play, chaque carte d'E/S possédait un niveau d'interruption prédéfini et des adresses fixes pour ses registres d'E/S. Ainsi, le clavier utilisait l'interruption 1 et les adresses E/S 0x60 à 0x64 ; le lecteur de disquettes occupait l'interruption 6 et les adresses 0x3F0 à 0x3F7, et l'imprimante l'interruption 7 et les adresses 0x378 à 0x37A, etc.

Au départ, tout allait bien. Les problèmes surgissaient lorsque l'utilisateur connectait une carte son et une carte modem qui utilisaient par exemple toutes deux l'interruption 4. Elles entraient en conflit et il était impossible de les faire fonctionner ensemble. La solution consistait à manipuler de petites bascules ou des cavaliers inclus sur les cartes afin de modifier leur niveau d'interruption et leur plage d'adresses pour qu'elles n'entrent plus en conflit avec les autres périphériques du système. À part quelques adolescents qui avaient décidé de consacrer leur vie à la maîtrise des arcanes de l'architecture matérielle du PC, personne ne pouvait trouver la configuration permettant d'annuler tous les conflits, ce qui a amené à la plus grande confusion.

L'approche plug and play consiste à laisser le système collecter automatiquement l'information sur les périphériques d'E/S présents, affecter de manière centralisée niveaux d'interruption et plages d'adresses E/S, puis redistribuer à chaque carte d'E/S les paramètres qui la concernent. Cela ressemble de façon très proche à ce qui se passe au démarrage du système.

### 1.3.7 Démarrage de l'ordinateur

Très schématiquement, examinons le processus de démarrage du Pentium. Chaque Pentium est fixé sur une carte mère (*motherboard*). Sur cette carte mère, on trouve le **BIOS** (*Basic Input Output System*), un logiciel de bas niveau dédié aux E/S contenant des procédures pour lire le clavier, écrire à l'écran et réaliser des E/S disques, entre autres. De nos jours, le BIOS est contenu dans une mémoire Flash RAM non volatile, mais que le système d'exploitation peut mettre à jour lorsque des bogues du BIOS sont détectés.

Au démarrage du système, le BIOS est lancé. Il commence par vérifier la taille de la mémoire RAM et la présence, et la réponse, du clavier et d'autres périphériques de base. Il utilise alors les bus ISA et PCI pour détecter les périphériques qui y sont

connectés. Certains de ces périphériques sont spécifiques, dans la mesure où ils ont été conçus avant le plug and play, et ont donc des niveaux d'interruption et des plages d'adresses E/S fixes. Ces périphériques et leurs paramètres sont enregistrés. Les périphériques plug and play sont également listés. Si la liste est différente de ce qu'elle était au démarrage précédent, les nouveaux périphériques sont configurés.

Le BIOS détermine alors le périphérique d'amorçage en consultant une liste stockée dans la mémoire CMOS. L'utilisateur peut modifier cette liste à l'aide d'un programme de configuration du BIOS au moment du boot. Typiquement, un boot à partir du lecteur de disquettes est d'abord tenté. En cas d'échec, le lecteur de CD-ROM est essayé, puis le disque dur. Le premier secteur du périphérique de boot est lu en mémoire et exécuté. Ce secteur contient un programme qui examine la table des partitions (à la fin du secteur de boot) pour connaître la partition active. Un second chargeur de boot est alors lu en mémoire depuis cette partition. Ce chargeur lit le système d'exploitation en mémoire et le lance.

Le système d'exploitation consulte alors le BIOS pour avoir les informations de configuration. Pour chaque périphérique, il vérifie la présence d'un pilote. S'il n'en trouve pas, il demande à l'utilisateur de fournir une disquette ou un CD-ROM le contenant. Quand il a trouvé tous les pilotes nécessaires, il les charge dans le noyau. Il initialise ensuite ses tables internes, crée les processus nécessaires en tâche de fond, puis démarre un programme de login (en mode texte ou graphique) sur chaque terminal.

## 1.4 Le bestiaire des systèmes d'exploitation

---

L'histoire des systèmes d'exploitation s'étale sur les 50 dernières années. Durant ces années, une large palette de systèmes, plus ou moins célèbres, ont vu le jour. Nous allons dans cette section évoquer brièvement neuf familles de systèmes. Il y sera fait allusion à plusieurs reprises dans la suite de cet ouvrage.

### 1.4.1 Les systèmes d'exploitation des mainframes

À l'une des extrémités du spectre, on trouve les systèmes d'exploitation des mainframes, ces ordinateurs qui remplissent une pièce et que l'on trouve encore dans les centres de calcul de certaines grandes sociétés. Ces machines se distinguent principalement des ordinateurs personnels de par leurs capacités d'E/S. Il n'est pas rare de voir un mainframe gérer un millier de disques pour plusieurs téraoctets de mémoire au total. Les mainframes vivent en outre une seconde jeunesse en tant que serveurs Web, de commerce électronique ou B2B (*Business to Business*).

L'accent est mis dans leurs systèmes d'exploitation sur la capacité à gérer de manière optimale plusieurs jobs en même temps, chacun d'eux demandant de grandes ressources d'E/S. Ils offrent typiquement trois types de services : le batch, le transactionnel et le temps partagé. Un système batch exécute des jobs sans interaction avec l'utilisateur. La gestion des déclarations dans une compagnie d'assurances ou les statistiques de ventes dans une chaîne de magasins sont des exemples de ce type de travail.



Le transactionnel doit gérer en permanence un très grand nombre de petites requêtes concurrentes. Le traitement des chèques dans une banque ou un système de réservation de billets d'avion en sont deux exemples types. Le temps partagé permet à plusieurs utilisateurs, éventuellement distants, d'exécuter des travaux en même temps sur la même machine. Ces trois fonctions sont étroitement reliées, et les systèmes pour mainframes sont fréquemment capables de rendre les trois types de services, à l'image de l'OS/390, un descendant d'OS/360 d'IBM. Il faut toutefois noter que les systèmes pour mainframes sont progressivement remplacés par des variantes du système UNIX comme Linux.

### **1.4.2 Les systèmes d'exploitation des serveurs**

Un cran au-dessous des précédents, on trouve les systèmes d'exploitation des serveurs. Ils fonctionnent sur des machines-serveurs, qui sont soit de gros micro-ordinateurs, soit des stations de travail, voire des mainframes. Ils servent en parallèle de nombreux utilisateurs à travers le réseau et permettent à ces derniers de partager des ressources matérielles et logicielles. Ils proposent en général un service d'impression, un service de fichiers ou un service Web. Les fournisseurs d'accès à l'internet utilisent nombre de ces machines pour servir leurs clients. Les sites Web en utilisent pour stocker et distribuer des pages HTML et gérer les requêtes clientes. Les représentants courants de cette famille sont Solaris, FreeBSD, Linux et Windows Serveur 200x.

### **1.4.3 Les systèmes d'exploitation des multiprocesseurs**

Le recours à plusieurs processeurs sur une même plate-forme pour augmenter la puissance de calcul est une technique de plus en plus courante. En fonction du type de connexions, et de ce qui est partagé, on appelle ces systèmes « ordinateurs parallèles », multi-ordinateurs ou multiprocesseurs. Ils nécessitent des systèmes d'exploitation spéciaux qui sont le plus souvent des variantes de systèmes serveurs, améliorés au niveau de la connectivité et des communications.

L'arrivée des circuits multicœurs au sein des PC et des agendas électroniques influe sur leurs SE qui s'imprègnent de plus en plus des concepts des systèmes multiprocesseurs. C'est le cas notamment de Windows et de Linux.

### **1.4.4 Les systèmes d'exploitation des PC**

Leur rôle est de fournir à l'utilisateur une interface conviviale. Ils sont principalement dédiés au traitement de texte, à l'utilisation de tableurs et à l'accès à l'internet. Les systèmes les plus représentatifs sont Windows 2000, Windows Vista, Mac OS, FreeBSD et Linux. Ils sont si répandus qu'il n'est sans doute pas nécessaire d'en dire beaucoup plus. Nombreux sont d'ailleurs les utilisateurs qui pensent qu'ils constituent la seule catégorie existante de systèmes d'exploitation. Les systèmes les plus modernes supportent la multiprogrammation et permettent l'exécution simultanée d'une douzaine de programmes.

### 1.4.5 Les systèmes d'exploitation des assistants personnels

En descendant l'échelle de taille, on trouve les assistants numériques personnels (ou, plus simplement, assistants personnels) et les systèmes embarqués. Un assistant personnel ou **PDA** (*Personal Digital Assistant*) est un petit ordinateur qui tient dans une poche et sert de carnet d'adresses électronique, de bloc-notes, etc. Les PDA et les téléphones mobiles sont en train de converger : ils ont tous deux des processeurs 32 bits et des SE sophistiqués et ne diffèrent essentiellement que par le poids, la taille et les capacités d'E/S (écran/clavier).

C'est ainsi que les SE des assistants personnels permettent, outre les fonctions classiques d'agenda et de carnet d'adresses, de téléphoner, de prendre des photos, d'écouter de la musique, etc. Certains offrent en outre la possibilité d'exécuter de petites applications. Ces systèmes d'exploitation commencent à ressembler à ceux des PC d'il y a une dizaine d'années. La majeure différence entre assistant personnel et PC se situe au niveau des capacités mémoire centrale et disque dur. Les deux représentants types de leurs systèmes d'exploitation sont Symbian OS et Palm OS.

### 1.4.6 Les systèmes d'exploitation embarqués

Les systèmes embarqués tournent sur des ordinateurs qui pilotent des périphériques, lesquels d'ordinaire ne sont pas dépendants d'un ordinateur, comme une télévision, un four à micro-ondes, un téléphone portable ou un lecteur MP3. Ils ont souvent des caractéristiques temps réel mais leurs dimensions, taille mémoire et restrictions en termes d'alimentation électrique les rangent à part. Vis-à-vis des assistants personnels ces systèmes n'autorisent aucunement l'exécution d'un programme quelconque externe. On ne peut bien entendu pas modifier, ni charger un nouveau programme sur un four à micro-ondes dont le programme de fonctionnement est inscrit en mémoire ROM. Les systèmes QNX et VxWorks sont deux célèbres systèmes d'exploitation embarqués.

### 1.4.7 Les systèmes d'exploitation des objets communicants

Les réseaux d'objets communicants (capteurs, etc.) sont en plein développement. Ces objets, nœuds de communications, sont de véritables petits ordinateurs qui communiquent par radio entre eux et avec une station centrale (appelée station de base) qui se trouve dans leur environnement. Ces réseaux sont par exemple utilisés pour mettre en place des systèmes de protection ou de surveillance d'immeuble, de site industriel, d'incendie de forêt, etc.

Ces objets sont alimentés par pile, ont une puissance limitée et peuvent fonctionner sur une longue période de temps. Ils disposent d'un processeur, de mémoires RAM et ROM, et d'un petit système d'exploitation temps réel leur permettant de répondre et d'agir face à des événements extérieurs qu'ils peuvent détecter ou mesurer. Tout comme les systèmes embarqués, leur programme d'application est chargé en mémoire ROM à la fabrication et ils n'acceptent pas d'exécuter autre chose. TinyOS est le plus connu des SE de réseaux de capteurs.

### 1.4.8 Les systèmes d'exploitation temps réel

Ces systèmes se caractérisent par le respect de strictes contraintes temporelles. Par exemple, dans un système de contrôle de processus, des ordinateurs temps réel doivent regrouper des données relatives à un processus de fabrication pour contrôler les machines dans une usine. Il y a fréquemment des limites qui doivent impérativement être respectées (on parle de temps réel « dur »). Ainsi, dans le cas d'un robot soudeur, une soudure faite trop tôt ou trop tard peut sérieusement compromettre un véhicule. Sur une chaîne de montage automobile, si une action doit absolument avoir lieu à tel instant (ou dans un certain délai), on se trouve dans une situation de **temps réel dur**.

L'autre catégorie relève du **temps réel mou**, dans lequel il est admissible de dépasser occasionnellement certains délais. Les systèmes audionumériques et multimédias se rangent dans cette catégorie, de même que les systèmes de téléphones numériques.

Comme le respect des délais est très strict sur les systèmes temps réel, leurs systèmes d'exploitation sont souvent réduits à la plus simple expression : des bibliothèques associées aux programmes d'application, avec des couplages relativement forts entre les différents éléments. Le système e-Cos est un représentant type de système d'exploitation temps réel.

Les systèmes d'exploitation des assistants personnels, des systèmes embarqués et des systèmes temps réel ont nombre de points en commun. Presque tous ont des orientations temps réel. Les programmes qu'exécutent les systèmes embarqués et les assistants personnels sont uniquement ceux intégrés à la conception. Les utilisateurs ne peuvent pas y intégrer leurs propres programmes, ce qui rend leur protection plus simple à réaliser. Bien qu'ils aient des points en commun, les assistants personnels comme les systèmes embarqués sont principalement destinés au grand public, tandis que les systèmes temps réel sont plutôt à usage industriel.

### 1.4.9 Les systèmes d'exploitation pour smart cards

Les plus petits systèmes d'exploitation se trouvent sur les smart cards, ou cartes à puce, des périphériques de la taille d'une carte de crédit contenant un processeur. Selon le mode de lecture, on rencontre des cartes à contact (introduction de la carte dans un lecteur) ou sans contact (la carte est passée devant un lecteur sans qu'il y ait contact mécanique). Ces systèmes sont sujets à de sévères contraintes de mémoire et de puissance de calcul. Certains ne savent remplir qu'une seule tâche, par exemple le paiement électronique, mais ce n'est pas toujours le cas. Leur SE est le plus souvent « propriétaire ».

Certaines smart cards reposent sur Java : la ROM de la carte contient une machine virtuelle Java (JVM, *Java Virtual Machine*). Les applets Java (de petits programmes Java) sont téléchargés sur la carte et interprétés par la JVM. Certaines cartes peuvent gérer plusieurs applets concurrents, ce qui oblige à utiliser la multiprogrammation et à mettre en place une politique d'ordonnancement des tâches. La gestion de ressources et la protection deviennent également, dans ce cas, d'actualité. Le système d'exploitation, même primitif, de la carte doit alors prendre en charge ces fonctionnalités supplémentaires.

## 1.5 Les concepts de base des systèmes d'exploitation

---

Tous les systèmes d'exploitation partagent certains concepts de base comme les processus, la mémoire ou les fichiers. La compréhension de ces concepts est nécessaire à celle du système d'exploitation. Dans les sections qui suivent, nous allons passer en revue certains de ces concepts, même brièvement, en guise d'introduction. Nous reviendrons sur chacun d'eux en détail plus loin dans cet ouvrage. Pour illustrer ces concepts, nous aurons de temps en temps recours à des exemples, généralement tirés d'UNIX et d'autres systèmes. Nous étudierons en outre plus en détail Windows Vista au chapitre 11.

### 1.5.1 Les processus

Le **processus** est un concept clé dans tout système d'exploitation. De façon schématique, un processus est un programme en cours d'exécution. Chaque processus est pourvu d'un **espace d'adressage**, un ensemble d'adresses mémoire allant de 0 à une limite donnée, dans lesquelles le processus peut lire et écrire. L'espace d'adressage contient le programme exécutable, ses données et sa pile. Les processus possèdent également un ensemble de registres, parmi lesquels le compteur ordinal, le pointeur de pile, d'autres registres matériels et toute l'information nécessaire pour exécuter le programme.

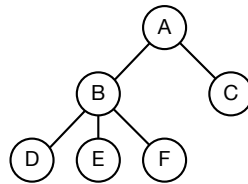
Nous reviendrons en détail sur la notion de processus au chapitre 2 mais, pour l'instant, la façon la plus simple de situer la notion de processus est de penser à un système multiprogrammé. Imaginez que vous ayez à convertir un volumineux fichier vidéo dans un certain format (ce qui peut durer environ une heure) et que pendant ce temps vous décidiez de naviguer sur le Web alors qu'en tâche de fond, périodiquement, votre courrier électronique est relevé. On distingue (au moins) trois processus actifs : le convertisseur de fichier vidéo, le navigateur Web et le releveur de courrier électronique. Périodiquement, le système d'exploitation décide d'interrompre un processus et d'en faire avancer un autre, par exemple parce que le premier a eu sa part de temps UC durant la dernière seconde.

Quand un processus est ainsi suspendu temporairement, il doit ensuite être redémarré exactement dans le même état. Cela suppose que toute l'information sur le processus soit explicitement sauvée quelque part pendant la suspension. Par exemple, le processus peut avoir ouvert plusieurs fichiers en lecture. À chacun de ces fichiers est associé un pointeur donnant la position courante (c'est-à-dire le numéro du prochain octet ou enregistrement à lire). Quand un processus est suspendu, tous ces pointeurs doivent être sauvegardés afin qu'un `read` après le redémarrage du processus fournisse bien l'information désirée. Dans de nombreux systèmes, toute l'information sur chaque processus, hormis le contenu de l'espace d'adressage, est stockée dans une table nommée **table des processus**, une pour chaque processus en cours.

Ainsi, un processus suspendu est composé de son espace d'adressage et de son entrée dans la table des processus, qui contient, entre autres choses, ses registres.

Les principaux appels système pour la gestion des processus couvrent la création et la disparition des processus. Considérons un exemple typique. Un processus appelé **interpréteur de commandes** (ou **shell**) lit des commandes depuis un terminal. L'utilisateur vient juste de taper une commande demandant la compilation d'un programme. Le shell doit alors créer un nouveau processus qui exécutera le compilateur. Quand ce processus a terminé la compilation, il exécute un appel système pour se terminer.

Si un processus peut créer un ou plusieurs autres processus (appelés **processus fils** ou **enfants**), et que ces derniers créent également des processus fils, on arrive à une structure arborescente du type de celle représentée à la figure 1.13. Des processus qui collaborent à la réalisation d'une tâche complexe ont souvent besoin de communiquer entre eux et de synchroniser leurs actions. Cette communication est appelée **communication interprocessus**, et sera décrite en détail au chapitre 2.



**Figure 1.13** • Une arborescence de processus. Le processus A a deux fils, B et C. Quant au processus B, il a trois fils, D, E et F.

D'autres appels système permettent de demander davantage de mémoire (ou d'en libérer lorsqu'elle n'est plus utilisée), d'attendre qu'un processus fils se termine, ou de recouvrir son programme par un autre.

Il arrive qu'il faille faire parvenir de l'information à un processus en cours d'exécution alors que ce processus n'attend pas cette information. Ainsi, un processus qui communique avec un autre situé sur un ordinateur différent utilise des messages véhiculés par un réseau. Pour éviter qu'un message (ou une réponse) soit perdu, l'émetteur peut demander que le système d'exploitation l'avertisse au bout d'un nombre donné de secondes, afin qu'il réémette le message si aucun accusé de réception n'est encore parvenu. Après avoir positionné ce **temporisateur** ou **timer**, le programme peut continuer et faire autre chose. Quand la durée spécifiée s'est écoulée, le système envoie un *signal d'alarme* au processus. Ce signal provoque une interruption momentanée du processus et la sauvegarde de ses registres sur la pile. Le processus démarre alors une procédure spéciale de gestion de ce signal : il réémet par exemple le message supposé perdu. Quand la procédure de gestion du signal d'alarme se termine, le processus est réactivé dans l'état où il se trouvait lors de la réception du signal. Les signaux sont donc l'équivalent logiciel des interruptions matérielles et peuvent être créés pour un grand nombre de raisons autres que l'expiration d'un timer. De nombreux *traps* (c'est-à-dire déroutements ou interruptions logicielles) détectés par le matériel, comme l'exécution d'une instruction illégale ou l'utilisation d'une adresse invalide, sont aussi convertis en signaux d'alarme envoyés au processus fautif.

Chaque personne autorisée à utiliser un système se voit attribuer un **identifiant d'utilisateur** ou **UID** (*User IDentification*) par l'administrateur système. Chaque processus hérite de l'UID de l'utilisateur qui l'a créé. Un processus enfant hérite de l'UID de son processus parent. Les utilisateurs peuvent être structurés en groupes, chaque groupe étant identifié par son **identifiant de groupe** ou **GID** (*Group IDentification*).

Un UID particulier, appelé **super-utilisateur** (surtout sous UNIX), possède des pouvoirs spéciaux et peut violer certaines règles de protection. Dans des grands environnements, seul l'administrateur système connaît le mot de passe associé à cet UID, mais de nombreux utilisateurs (notamment des étudiants en informatique) déploient d'importants efforts pour découvrir les points faibles du système qui pourraient leur permettre de devenir super-utilisateurs sans le mot de passe.

Nous étudierons les processus, la communication interprocessus et les aspects connexes au chapitre 2.

## 1.5.2 Espace d'adressage

Tout ordinateur possède une mémoire principale qui contient les programmes en cours d'exécution.

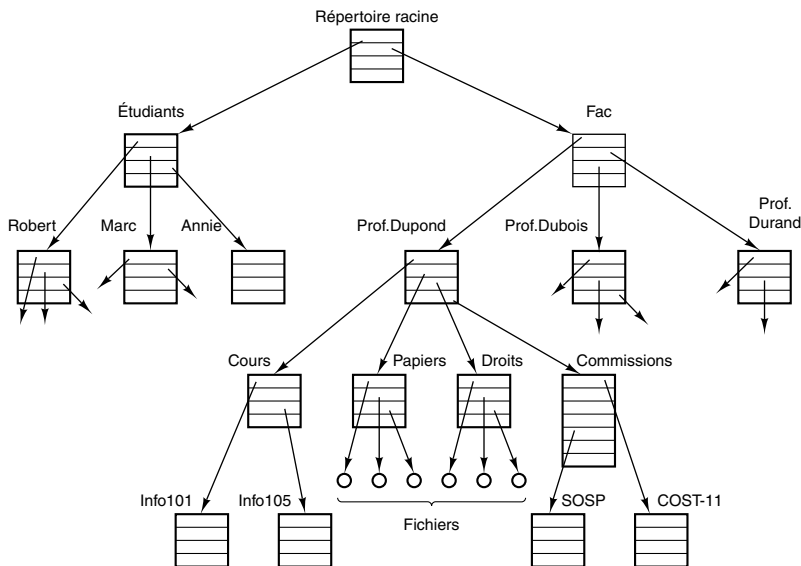
Dans les cas les plus simples, seul un programme à la fois peut se trouver en mémoire. Pour exécuter un second programme, on doit d'abord décharger le premier puis charger le second. Les systèmes plus sophistiqués autorisent plusieurs programmes à cohabiter en même temps en mémoire. Un mécanisme de protection doit alors être mis en place pour les empêcher d'interférer entre eux et avec le système d'exploitation. Bien que d'origine matérielle, ce mécanisme est contrôlé par le système d'exploitation. Cet aspect de protection de la mémoire principale n'est pas le seul à prendre en compte. La gestion de l'espace d'adressage des processus est un problème aussi important. Normalement, chaque processus est autorisé à utiliser une plage d'adresses, s'étendant en général de 0 à une borne donnée. Dans les cas les plus simples, cette plage est de taille inférieure à celle de la mémoire principale. Le processus peut alors remplir son espace d'adressage, qui tient intégralement en mémoire principale.

Cependant, sur de nombreux ordinateurs, les adresses sont sur 32 ou 64 bits, ce qui donne un espace d'adressage de  $2^{32}$  ou  $2^{64}$  mots suivant les cas. Que se passe-t-il si un processus, qui possède un espace d'adressage supérieur à celui de la mémoire principale, décide d'utiliser la totalité de cet espace ? Avec les premiers ordinateurs, un tel processus ne pouvait qu'échouer. De nos jours, une technique appelée mémoire virtuelle permet au système d'exploitation de garder une partie de l'espace d'adressage en mémoire principale et une partie sur disque, et d'assurer les allers-retours nécessaires entre ces deux parties. Cette fonction importante du système d'exploitation (ainsi que tout ce qui est relatif à la gestion de la mémoire) est décrite au chapitre 3.

### 1.5.3 Les fichiers

Un autre concept clé présent dans quasiment tous les systèmes d'exploitation est le système de fichiers. L'une des fonctions principales du système d'exploitation est en effet de masquer les particularités des disques et autres périphériques d'E/S et de présenter au programmeur un modèle abstrait homogène de fichiers indépendants du périphérique. Il faut bien sûr des appels système pour la création et la destruction de fichiers, et les opérations de lecture et d'écriture. Avant de pouvoir lire un fichier, on doit le trouver sur le disque et l'ouvrir ; après l'avoir lu, on doit le refermer. L'ouverture et la fermeture de fichiers est aussi faite à l'aide d'appels système.

La plupart des systèmes d'exploitation utilisent la notion de **répertoire** (*directory*) pour structurer l'espace des fichiers. Un étudiant peut par exemple grouper les fichiers relatifs à un cours dans un répertoire, son courrier électronique dans un autre, et son site Web personnel dans un troisième. Il faut donc disposer d'appels système pour la création et la suppression de répertoires, l'ajout et la suppression de fichiers dans un répertoire. Les entrées de répertoires pouvant être des fichiers mais aussi des répertoires, ce modèle conduit à une hiérarchie arborescente, le système de fichiers, comme le montre la figure 1.14.



**Figure 1.14** • Un système de fichiers pour un département d'informatique.

Les hiérarchies de processus et de fichiers sont toutes deux représentées de manière arborescente, mais la ressemblance s'arrête là. Les arborescences de processus sont en général peu profondes (rarement plus de trois niveaux) alors que les systèmes de fichiers arborescents ont fréquemment plus de cinq niveaux. Les arbres de processus ont une durée de vie brève ; celle des hiérarchies de fichiers se compte souvent en

années. Les notions de propriété et de droits d'accès sont aussi très différentes d'un environnement à l'autre. Typiquement, seul le processus parent peut contrôler, voire même accéder à, un processus fils, alors qu'on dispose presque systématiquement de mécanismes permettant l'accès aux fichiers par des groupes d'utilisateurs.

Chaque fichier de la hiérarchie est identifié par un **chemin** (*path name*) depuis le sommet de la hiérarchie, appelé **répertoire racine** (*root directory*). Un tel chemin (qualifié d'absolu) est composé de la suite des répertoires à traverser depuis la racine pour accéder au fichier, séparés par un caractère spécial (en général le *slash* « / »). La figure 1.14 donne pour le fichier `info101` le chemin `/Fac/Prof.Dupond/Cours/info101`.

Le « / » du début indique que le chemin est absolu (c'est-à-dire qu'il débute à la racine). MS-DOS et Windows utilisent le *backslash* « \ » comme séparateur. Dans ce cas, l'accès au fichier `info101` précédent devient `\Fac\Prof.Dupond\Cours\info101`.

Tout au long de ce livre nous n'utiliserons quasi uniquement que la convention relative à UNIX, c'est-à-dire celle avec le caractère *slash* « / ».

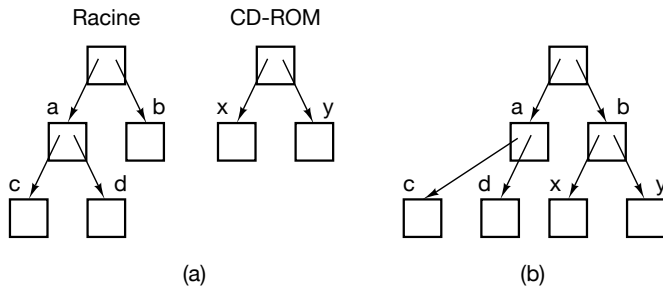
À tout instant, chaque processus possède un **répertoire de travail courant** (*working directory*), dans lequel sont recherchés les fichiers dont le chemin ne commence pas par un « / » (on parle de chemin *relatif*). Ainsi, si pour reprendre l'exemple de la figure 1.14, on considère que le répertoire courant est `/Fac/Prof.Dupond`, le fichier `info101` peut également être accédé sous le nom `Cours/info101`. Un processus peut changer de répertoire de travail grâce à un appel système particulier.

Avant de pouvoir être lu ou écrit, un fichier doit être ouvert. Les droits d'accès sont vérifiés lors de cette ouverture. En cas de succès, le système renvoie une valeur entière appelée **descripteur de fichier** qui sera utilisée pour toutes les opérations sur le fichier. En cas d'échec, un code d'erreur est renvoyé.

Le **montage** d'un système de fichiers est un concept important sous UNIX. Presque tous les ordinateurs disposent d'un ou plusieurs lecteurs de CD-ROM et DVD. Ils disposent également de plusieurs ports USB permettant d'y raccorder divers types de périphériques de stockage. Pour permettre une gestion élégante des systèmes de fichiers extractibles (CD-ROM notamment), UNIX permet d'attacher le système de fichiers d'un périphérique de ce type à l'arborescence principale. Ceci est illustré par la figure 1.15. Dans la partie (a), avant le montage, les deux systèmes de fichiers (celui du disque dur et celui du CD-ROM) sont séparés et sans liens.

Les fichiers qui se trouvent sur le CD-ROM ne peuvent pas être accédés depuis le répertoire racine, car il n'y a pas moyen d'y spécifier un chemin d'accès. UNIX ne permet pas d'allouer des lettres ou des chiffres à chaque périphérique d'E/S. La partie (b) de la figure montre le résultat du montage du système de fichiers du CD-ROM sur le répertoire `b`, permettant ainsi l'accès à `/b/x` et `/b/y`. Si `b` contenait des fichiers avant le montage, ceux-ci deviennent inaccessibles tant que le CD-ROM est monté sur ce répertoire. En règle générale, les systèmes de fichiers sont montés sur des répertoires vides. Si un système contient plusieurs disques durs, le même mécanisme permet de les faire apparaître comme une arborescence unique.

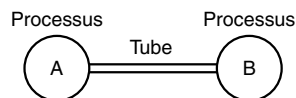




**Figure 1.15** • (a) Avant montage, les fichiers sur le CD-ROM sont inaccessibles. (b) Après montage, ils sont inclus dans l'arborescence générale.

La notion de **fichier spécial** est également importante sous UNIX. Leur rôle est de faire apparaître les périphériques d'E/S comme des fichiers. Ainsi, ils peuvent être lus et écrits à l'aide des mêmes appels système que ceux utilisés pour les fichiers ordinaires. Il y a deux catégories de fichiers spéciaux, ceux en mode **bloc** et ceux en mode **caractère**. Les fichiers spéciaux en mode bloc représentent des périphériques consistant en une collection de blocs accessibles de manière aléatoire, comme les disques. En ouvrant un fichier spécial en mode bloc et en lisant le bloc 4 de ce fichier, on peut accéder à ce bloc indépendamment de la structure de système de fichiers implantée sur le disque correspondant. De même, les fichiers spéciaux en mode caractère servent à représenter les imprimantes, les modems et autres périphériques traitant des flots de caractères. Par convention, les fichiers spéciaux sont placés dans le répertoire `/dev`. Par exemple, `/dev/lp` représente fréquemment l'imprimante.

La dernière caractéristique abordée dans cette introduction est reliée à la fois aux processus et aux fichiers : les tubes. Un **tube** (*pipe*) est une sorte de pseudo-fichier que l'on utilise pour relier deux processus, comme le montre la figure 1.16. Si les processus *A* et *B* souhaitent communiquer *via* un tube, ils doivent auparavant le créer. Quand *A* veut envoyer des données à *B*, il écrit dans le tube comme il le ferait dans un fichier de sortie. *B* peut alors lire les données comme il le ferait depuis un fichier d'entrée. La communication entre processus sous UNIX ressemble ainsi beaucoup à des lectures-écritures sur des fichiers. À tel point que la seule façon pour un processus de savoir que le fichier dans lequel il est en train d'écrire n'est pas un fichier ordinaire mais un tube est d'exécuter un appel système particulier. Les systèmes de fichiers sont très importants. Il en sera question en détail au chapitre 4, mais aussi dans les chapitres 10 et 11.



**Figure 1.16** • Deux processus reliés par un tube.

### 1.5.4 Les entrées/sorties

Tous les ordinateurs sont munis de périphériques pour l'acquisition de données en entrée et la production de données en sortie. De nombreux types de périphériques existent (claviers, écrans, imprimantes, etc.). La gestion de ces périphériques est de la responsabilité du système d'exploitation.

Ainsi, tout système d'exploitation possède un sous-système d'entrées/sorties responsable des périphériques. Une partie du logiciel est indépendante du périphérique et s'applique à plusieurs, voire à la totalité, des périphériques du système. Une autre partie (par exemple les pilotes) est spécifique à des périphériques particuliers. Nous examinerons les logiciels d'E/S au chapitre 5.

### 1.5.5 La sécurité

Les ordinateurs renferment de grandes quantités d'informations que leur propriétaire souhaite souvent garder confidentielles. Cela peut inclure du courrier électronique, des documents professionnels, d'imposition, etc. Il revient au système d'exploitation d'assurer la sécurité, de telle sorte que les fichiers, par exemple, ne soient accessibles que par les utilisateurs autorisés.

Prenons l'exemple d'UNIX pour avoir une idée du fonctionnement de la sécurité. Sous UNIX, les fichiers sont protégés par un code binaire à 9 bits. Ce code est composé de trois champs à 3 bits chacun, un champ pour le propriétaire du fichier, un pour les membres du groupe (les utilisateurs sont structurés en groupes par l'administrateur du système) auquel le fichier est rattaché, et un pour les autres utilisateurs. Chaque champ contient un bit pour le droit de lecture, un pour l'accès en écriture et un pour l'exécution. On les appelle fréquemment les **bits rwx** (*Read/Write/eXecute*). Par exemple, le code `rxwr-x--x` signifie que le propriétaire peut lire, écrire et exécuter le fichier, que les membres de son groupe peuvent le lire et l'exécuter, mais pas l'écrire, et que les autres utilisateurs ne peuvent que l'exécuter. Pour les répertoires, le « x » signifie droit de consultation. Un tiret indique que le droit correspondant est refusé.

Outre la protection des fichiers, il existe de nombreux domaines où la sécurité s'exerce. La protection du système contre les intrus humains ou non, comme les virus, en est un exemple. Nous en traiterons plusieurs au chapitre 9.

### 1.5.6 Le shell

Le système d'exploitation est le code qui gère les appels système. Les éditeurs, compilateurs, assembleurs, éditeurs de liens et interpréteurs de commandes ne font clairement pas partie de l'OS, même s'ils sont importants et utiles. Au risque d'entretenir cette confusion, nous allons dans cette section évoquer brièvement l'interpréteur de commandes d'UNIX, appelé **shell**. Il fait abondamment usage des caractéristiques du système d'exploitation et constitue de ce fait un bon exemple d'utilisation des appels système. Il constitue également la principale interface entre un utilisateur placé devant son terminal et le système d'exploitation, sauf si l'utilisateur a recours à une

interface graphique. De nombreux shells sont disponibles (sh, csh, ksh, bash, etc.). Tous supportent les fonctionnalités décrites ci-après, qui dérivent du shell original (sh).

Quand un utilisateur se connecte, un shell est lancé. Ce shell a pour entrée standard le clavier du terminal, et pour sortie standard son écran. Il commence par afficher un **prompt** (signe d'invite), une chaîne de caractères (par exemple un symbole « \$ ») qui indique à l'utilisateur que le shell est prêt à recevoir une commande. Si l'utilisateur tape alors par exemple,

```
└─ date
```

le shell crée un processus fils qui exécute alors le programme `date`. Pendant que le fils s'exécute, le shell attend sa terminaison. Quand elle a lieu, le prompt réapparaît et le shell attend de lire la commande suivante.

L'utilisateur peut demander que la sortie soit redirigée dans un fichier, comme

```
└─ date > fichier
```

De même, l'entrée de la commande peut être redirigée dans un fichier. Ainsi,

```
└─ sort < fichier1 > sortie2
```

demande à la commande `sort` de lire les données contenues dans `fichier1`, de les trier et d'écrire le résultat dans `fichier2`.

La sortie d'une commande peut être utilisée comme entrée d'une autre commande en connectant les deux commandes par un tube. Ainsi,

```
└─ cat fichier1 fichier2 fichier3 | sort > /dev/lp
```

demande à la commande `cat` de concaténer les trois fichiers (`fichier1`, `fichier2` et `fichier3`) et d'envoyer le résultat de l'opération à la commande `sort` qui triera les lignes dans l'ordre alphabétique. La sortie du tri sera alors envoyée sur l'imprimante.

Si l'utilisateur termine sa commande par un « & », le shell n'attend pas la fin de la commande ; il renvoie immédiatement le prompt. Ainsi,

```
└─ cat fichier1 fichier2 | sort > /dev/lp &
```

démontre le tri comme un job en tâche de fond, ce qui permet à l'utilisateur de continuer à travailler normalement pendant que le tri s'effectue. Le shell possède de nombreuses autres caractéristiques intéressantes, que nous ne pouvons pas développer ici en totalité. La plupart des ouvrages sur UNIX font une part importante à la description du shell.

Aujourd'hui, la plupart des PC utilisent des interfaces graphiques, de type GUI, qui sont des programmes s'exécutant au-dessus du système d'exploitation, comme le shell. Sur les systèmes Linux, l'utilisateur a le choix entre deux interfaces graphiques, Gnome ou KDE, ou pas du tout s'il utilise un terminal X.11. Dans le cas de Windows, il est également possible de changer l'interface graphique standard (Windows Explorer) par un autre programme en modifiant quelques valeurs dans un registre. Mais peu d'utilisateurs utilisent cette propriété.