Chapitre 3

Le système Linux étudié

La plupart des articles et des livres consacrés au noyau Linux prennent toujours en exemple la dernière version disponible au moment où ils sont écrits, qui n'est déjà plus la dernière version au moment où ils paraissent. De plus, comme le source est alors d'une taille très importante, une seule partie de celui-ci est étudiée. Je ne pense pas que ce soit une bonne idée. Si nous voulons vraiment comprendre la structure d'un système d'exploitation, nous avons intérêt à considérer le système le plus simple possible et à l'étudier en entier. C'est pourquoi j'ai choisi la toute première version du noyau Linux : le noyau 0.01. Je donnerai cependant quelques indications sur l'évolution du noyau, mais ce n'est pas le but essentiel.

1 Le système Linux à étudier

1.1 Noyau et distribution

Le système d'exploitation Linux est un gros logiciel et, comme tel, difficile à appréhender par une seule personne. Mais, en fait, il faut distinguer plusieurs niveaux.

Ce que l'on entend par Linux, le plus souvent, concerne une **distribution**, telle que Red Hat, Suse, Mandrake, Debian... Une distribution comprend le système d'exploitation proprement dit, plus exactement le noyau, les utilitaires traditionnellement associés à Unix (un éditeur de texte, un compilateur C...), l'interface graphique X Window System[1] et beaucoup de logiciels utilisateur.

Notre but, dans ce livre, est uniquement d'étudier le noyau Linux.

1.2 Noyau minimal

Même pour le seul noyau, les sources ont une taille non négligeable : 58 Mo pour la version 2.2.18, par exemple. Ceci s'explique, en particulier, par le grand nombre de périphériques pris en compte. Il est évidemment inutile de s'occuper de tous les périphériques et de tous les types de tels périphériques du point de vue pédagogique. Nous étudierons donc un noyau minimal, ne mettant pas nécessairement toutes les activités en application et ne contenant que quelques périphériques à titre d'exemple.

Le noyau Linux 0.01 est intéressant du point de vue pédagogique. Il ne concerne que le microprocesseur Intel `80386` (et ses successeurs), il ne prend en compte qu'un nombre très limité de

Linux 0.01

[1] On utilise souvent, à tort, le terme *X Window*. Le consortium X, auteur du programme, recommande plutôt d'employer les termes « X » ou « X Window System » pour évoquer ce produit.

périphériques, qu'un seul système de fichiers et qu'un seul type d'exécutables, mais ces défauts pour l'utilisateur deviennent un avantage lorsqu'on veut étudier les sources en entier.

1.3 Obtention des sources

L'ensemble des sources des noyaux de Linux, depuis le tout premier jusqu'au dernier, se trouve sur le site : `http://ftp.cdut.edu.cn/pub/linux/kernel/history/`

Linux 0.01

Nous étudierons, dans une première étape, les sources du tout premier noyau, nettement moins imposant et contenant évidemment l'essentiel. Les sources du noyau 0.01 se trouvent également à l'adresse suivante : `http://www.kernel.org/pub/linux/kernel/Historic/`

1.4 Programmation Linux

L'obtention du noyau Linux et la compréhension des fichiers source nous entraînent à faire un détour par la programmation sous Linux.

A priori, nous ne devrions rien à avoir à dire sur la programmation. Que ce soit Linux ou un autre système d'exploitation, nous devrions avoir des sources portables. En fait, ce n'est pas le cas pour des raisons historiques dues à certains choix initiaux de Linus Torvalds, jamais remis en cause ensuite.

Les sources reposent sur des fichiers `make` (un outil pour gérer les grands logiciels répartis sur de nombreux fichiers), sur des fichiers en langage C non standard (il s'agit du langage C de GCC avec quelques utilisations de particularités de ce compilateur), sur des fichiers en langage d'assemblage, pour Intel 80386 pour ce qui nous concerne et, enfin, sur des scripts *bash* modifiés. La syntaxe du langage d'assemblage ne suit pas celle de l'assembleur MASM de Microsoft (qui fut longtemps la référence) mais celle de `gas` dans un style dit ATT.

1.5 Versions du noyau Linux

Linux distingue les noyaux stables des noyaux en développement avec un système de numérotation simple. Chaque version est caractérisée par trois nombres entiers séparés par des points. Les deux premiers identifient la **version**, le troisième la **parution** (*release* en anglais).

Un second numéro pair identifie un noyau stable ; impair, il dénote un noyau de développement. Les nouvelles parutions d'une version stable visent essentiellement à corriger des erreurs signalées par les utilisateurs ; les algorithmes principaux et les structures de données du noyau ne sont pas modifiés.

Les versions de développement, en revanche, peuvent différer les unes des autres de façon importante. Les développeurs du noyau sont libres d'expérimenter différentes solutions qui peuvent éventuellement conduire à des changements drastiques du noyau.

La notation de la version 0.01 ne suit pas le principe de la numérotation décrit ci-dessus avec trois nombres entiers séparés par des points. Linus Torvalds voulant seulement indiquer que nous sommes très loin d'une version stable, qui porterait le numéro 1.0, il a choisi le numéro 0.01 pour laisser de la place pour encore d'autres versions intermédiaires.

2 Les sources du noyau 0.01

2.1 Vue d'ensemble sur l'arborescence

Les sources du noyau 0.01 occupent 230 Ko, comportent 11 dossiers et 88 fichiers.

Le premier niveau de l'arborescence du source est simple :

Linux 0.01

```
/boot
/fs
/include
/init
/kernel
/lib
/mm
/tools
```

Elle s'inspire de l'arborescence du source de MINIX ([TAN-87], p. 104). Nous avons vu que les systèmes d'exploitation se composent de quatre parties principales : le gestionnaire des processus, le gestionnaire de la mémoire, le gestionnaire des fichiers et le gestionnaire des périphériques d'entrée-sortie. Le répertoire *kernel* correspond aux couches 1 et 2 de MINIX (processus et périphériques d'entrée-sortie). Les procédures des bibliothèques standard C utilisées par le noyau (**open()**, **read()**,...) se trouvent dans le répertoire *lib* (pour *LIBrary*). Les répertoires *mm* (pour *Memory Management*) et *fs* (pour *File System*) comportent le code du gestionnaire de mémoire et du gestionnaire de fichiers.

Le répertoire *include* contient les fichiers d'en-têtes nécessaires au système Linux. Il sert à la constitution du noyau, mais également à la programmation Linux une fois le noyau constitué.

Les trois derniers répertoires contiennent les outils de mise en place : le répertoire *boot* permet de démarrer le système ; le répertoire *init* d'initialiser le système (il ne contient que la fonction principale **main()**) ; le répertoire *tools* permet de construire le noyau.

2.2 L'arborescence détaillée

Le répertoire *boot*

Pour le noyau 0.01, ce répertoire ne contient que deux fichiers en langage d'assemblage : *boot.s* et *head.s*. Voici les fonctions des ces deux fichiers :

- *boot.s* contient le code du secteur d'amorçage de la disquette à partir de laquelle on démarre Linux, de l'initialisation des périphériques de l'ordinateur, de la configuration de l'environnement pour pouvoir passer en mode protégé des micro-processeurs Intel et, enfin, du passage au mode protégé ;

  il donne ensuite la main au code **startup_32()** contenu dans le fichier suivant :

Linux 0.01

```
|       boot.s
|
| boot.s is loaded at 0x7c00 by the bios-startup routines, and moves itself
| out of the way to address 0x90000, and jumps there.
|
| It then loads the system at 0x10000, using BIOS interrupts. Thereafter
| it disables all interrupts, moves the system down to 0x0000, changes
| to protected mode, and calls the start of system. System then must
| RE-initialize the protected mode in its own tables, and enable
| interrupts as needed.
```

- *head.s* permet de configurer l'environnement d'exécution pour le premier processus Linux (processus 0) puis de passer à la fonction **start_kernel()**, qui est la fonction principale du code C :

Linux 0.01

```
/*
 *  head.s contains the 32-bit startup code.
 *
 * NOTE!!! Startup happens at absolute address 0x00000000, which is also
 * where the page directory will exist. The startup code will be
 * overwritten by the page directory.
 */
```

Le répertoire *init*

Le répertoire *init* contient un seul fichier : le fichier *main.c* qui, comme son nom l'indique, contient la fonction principale du code C. Cette fonction initialise les périphériques (en mode protégé) puis fait appel au processus 1.

Le répertoire *include*

Le répertoire *include* est évidemment le répertoire par défaut des fichiers d'en-têtes C qui ne font pas partie de la bibliothèque C standard. Il s'agit des fichiers d'en-têtes qui sont propres à Linux (propres à UNIX pour la plupart) ou, faisant partie de la bibliothèque C standard, qui doivent être implémentés suivant le système. Ces fichiers se trouvent soit dans le répertoire lui-même, soit dans l'un des trois sous-répertoires :

- *asm* contient des fichiers d'en-têtes dont le code est écrit en langage d'assemblage ;
- *linux* contient des fichiers d'en-têtes propres à Linux (n'existant pas sur les autres distributions UNIX) ;
- *sys* est un sous-répertoire classique d'UNIX, contenant les fichiers d'en-têtes concernant le système.

Le répertoire lui-même contient d'abord des fichiers d'en-têtes faisant partie de la bibliothèque standard C mais qu'il faut implémenter suivant le système (voir [PLAU-92] pour plus de détails) :

- *ctype.h* (pour *Character TYPEs*) permet le traitement des caractères en distinguant des classes de caractères (chiffre, alphabétique, espace...) ;
- *errno.h* (pour *ERRor NumerO*) permet d'associer un numéro à des constantes symboliques représentant les erreurs rencontrées ;
- *signal.h* définit les valeurs de code d'un ensemble de signaux ;
- *stdarg.h* (pour *STandarD ARGument*) définit des macros permettant d'accéder aux arguments d'une fonction, telle la fonction **printf()**, acceptant une liste variable d'arguments ;
- *stddef.h* (pour *STandarD DEFinitions*) contient un certain nombre de définitions standard (*sic*) ;
- *string.h* contient des fonctions permettant de manipuler les chaînes de caractères ;
- *time.h* concerne les calculs sur l'heure et la date.

Il contient ensuite des fichiers d'en-têtes propres à UNIX :

- *a.out.h* contient le format propre au type d'exécutable *a.out*, qui était le plus utilisé avant l'arrivée du format ELF ;

- *const.h* contient diverses valeurs de constantes ;
- *fcntl.h* contient les fonctions permettant de manipuler les descripteurs de fichiers ;
- *termios.h* contient les constantes et les fonctions concernant les terminaux ;
- *unistd.h* (pour *UNIx STandarD*) contient les constantes et les fonctions standard d'UNIX ;
- *utime.h* (pour *User TIME*) permet de changer la date et l'heure d'un nœud d'information.

Le sous-répertoire asm contient quatre fichiers :

- *io.h* (pour *Input/Output*) contient la définition des macros, en langage d'assemblage, permettant d'accéder aux ports d'entrée-sortie ;
- *memory.h* contient la définition de la macro **memcpy()** ;
- *segment.h* contient la définition des fonctions en ligne d'écriture et de lecture d'un octet, d'un mot ou d'un mot double ;
- *system.h* contient la définition de fonctions nécessaires à l'initialisation.

Le sous-répertoire linux contient neuf fichiers :

- *config.h* contient les données nécessaires au démarrage du système (concernant la capacité mémoire et le disque dur) ;
- *fs.h* (pour *File System*) contient les définitions des tableaux de structures pour les fichiers ;
- *hdreg.h* (pour *Hard Disk REGisters*) contient des définitions pour le contrôleur de disque dur de l'IBM PC-AT ;
- *head.h* contient des constantes nécessaires pour le fichier *head.s* ;
- *kernel.h* contient la déclaration de fonctions nécessaires pour le mode noyau (comme la fonction **printk()**) ;
- *mm.h* (pour *Memory Management*) contient la déclaration de fonctions de manipulation de la mémoire ;
- *sched.h* (pour *SCHEDuler*) contient la définition des structures et la déclaration des fonctions nécessaires à la manipulation des processus ;
- *sys.h* (pour *SYStem call*) contient la déclaration des appels système ;
- *tty.h* contient la définition de structures et la déclaration de fonctions concernant le terminal (`tty` pour *TeleTYpe*), nécessaires pour le fichier *tty_io.c* ci-dessous.

Le sous-répertoire sys contient cinq fichiers :

- *stat.h* contient la déclaration des fonctions renvoyant les informations sur les fichiers ;
- *times.h* contient la déclaration de la fonction renvoyant le nombre de tops d'horloge écoulés depuis le démarrage du système ;
- *types.h* contient la définition d'un certain nombre de types ;
- *utsname.h* contient la déclaration de la fonction donnant le nom et des informations sur le noyau ;
- *wait.h* contient la déclaration des fonctions permettant de suspendre l'exécution du processus en cours jusqu'à ce un processus fils se termine ou qu'un signal soit envoyé.

Le répertoire *kernel*

Il contient dix-sept fichiers, outre le fichier *Makefile* :

- *asm.s* contient les routines de service de la plupart des 32 premières interruptions, c'est-à-dire de celles qui sont réservées par Intel :

Linux 0.01

```
/*
 * asm.s contains the low-level code for most hardware faults.
 * page_exception is handled by the mm, so that isn't here. This
 * file also handles (hopefully) fpu-exceptions due to TS-bit, as
 * the fpu must be properly saved/resored. This hasn't been tested.
 */
```

- *console.c* contient les paramètres, les variables et les fonctions nécessaires à l'affichage sur le moniteur (nécessite les structures définissant un terminal) :

Linux 0.01

```
/*
 *      console.c
 *
 * This module implements the console io functions
 *      'void con_init(void)'
 *      'void con_write(struct tty_queue * queue)'
 * Hopefully this will be a rather complete VT102 implementation.
 *
 */
```

- *exit.c* contient les fonctions nécessaires pour quitter un processus autrement que par `return` ;
- *fork.c* contient les fonctions nécessaires pour créer un processus fils :

Linux 0.01

```
/*
 *  'fork.c' contains the help-routines for the 'fork' system call
 * (see also system_call.s), and some misc functions ('verify_area').
 * Fork is rather simple, once you get the hang of it, but the memory
 * management can be a bitch. See 'mm/mm.c': 'copy_page_tables()'
 */
```

- *hd.c* contient le pilote du disque dur :

Linux 0.01

```
/*
 * This code handles all hd-interrupts, and read/write requests to
 * the hard-disk. It is relatively straigthforward (not obvious maybe,
 * but interrupts never are), while still being efficient, and never
 * disabling interrupts (except to overcome possible race-condition).
 * The elevator block-seek algorithm doesn't need to disable interrupts
 * due to clever programming.
 */
```

- *keyboard.s* contient la routine de service associée à `IRQ1`, c'est-à-dire à l'interruption matérielle provenant du clavier ;
- *mktime.c* contient la fonction permettant de transformer la date exprimée en secondes depuis 1970 en année, mois, jour, heure, minute et seconde :

Linux 0.01

```
/*
 * This isn't the library routine, it is only used in the kernel.
 * as such, we don't care about years<1970 etc, but assume everything
 * is ok. Similarly, TZ etc is happily ignored. We just do everything
 * as easily as possible. Let's find something public for the library
 * routines (although I think minix times is public).
 */
/*
 * PS. I hate whoever though up the year 1970 - couldn't they have gotten
 * a leap-year instead? I also hate Gregorius, pope or no. I'm grumpy.
 */
```

- *panic.c* contient une fonction utilisée par le noyau pour indiquer un problème grave :

Linux 0.01

```
/*
 * This function is used through-out the kernel (includeinh mm and fs)
 * to indicate a major problem.
 */
```

- *printk.c* contient une fonction analogue à la fonction **printf()** du langage C mais qui peut être utilisée par le noyau :

Linux 0.01

```
/*
 * When in kernel-mode, we cannot use printf, as fs is liable to
 * point to 'interesting' things. Make a printf with fs-saving, and
 * all is well.
 */
```

- *rs_io.c* contient la routine de service associée aux interruptions matérielles des ports série (*rs* rappelant la norme RS232) :

Linux 0.01

```
/*
 *      rs_io.s
 *
 * This module implements the rs232 io interrupts.
 */
```

- *sched.c* contient le séquenceur (*scheduler* en anglais) qui permet de changer de processus pour rendre le système d'exploitation multi-tâches :

Linux 0.01

```
/*
 * 'sched.c' is the main kernel file. It contains scheduling primitives
 * (sleep_on, wakeup, schedule etc) as well as a number of simple system
 * call functions (type getpid(), which just extracts a field from
 * current-task
 */
```

- *serial.c* contient l'implémentation de deux fonctions servant aux ports série :

Linux 0.01

```
/*
 *      serial.c
 *
 * This module implements the rs232 io functions
 *      void rs_write(struct tty_struct * queue);
 *      void rs_init(void);
 * and all interrupts pertaining to serial IO.
 */
```

- *sys.c* contient la définition de beaucoup de fonctions de code **sys_XX()** d'appels système ;
- *system_call.s* contient du code en langage d'assemblage permettant d'implémenter les appels système :

Linux 0.01

```
/*
 *  system_call.s  contains the system-call low-level handling routines.
 * This also contains the timer-interrupt handler, as some of the code is
 * the same. The hd-interrupt is also here.
 *
 * NOTE: This code handles signal-recognition, which happens every time
 * after a timer-interrupt and after each system call. Ordinary interrupts
 * don't handle signal-recognition, as that would clutter them up totally
 * unnecessarily.
 *
 * ---------------------------------------------------------------------
 */
```

- *traps.c* contient le code en langage C des routines de service associées aux 32 premières interruptions, c'est-à-dire celles réservées par Intel :

Linux 0.01

```
/*
 * 'Traps.c' handles hardware traps and faults after we have saved some
 * state in 'asm.s'. Currently mostly a debugging-aid, will be extended
 * to mainly kill the offending process (probably by giving it a signal,
 * but possibly by killing it outright if necessary).
 */
```

- *tty_io.c* contient les fonctions nécessaires au fonctionnement du terminal :

Linux 0.01

```
/*
 * 'tty_io.c' gives an orthogonal feeling to tty's, be they consoles
 * or rs-channels. It also implements echoing, cooked mode etc (well,
 * not currently, but ...)
 */
```

- *vsprintf.c* contient le code permettant de définir à la fois les fonctions **printk()** et **printf()** :

Linux 0.01

```
/* vsprintf.c -- Lars Wirzenius & Linus Torvalds. */
/*
 * Wirzenius wrote this portably, Torvalds fucked it up:-)
 */
```

Le répertoire *lib*

Le répertoire *lib* contient onze fichiers, outre le fichier *Makefile* :

- *_exit.c* contient la définition de la fonction associée à l'appel système de terminaison d'un processus **_exit()** ;
- *close.c* contient la définition de la fonction associée à l'appel système de fermeture d'un fichier **close()** ;
- *ctype.c* contient la définition du tableau de définition des types de chacun des 256 caractères (majuscule, chiffre...) ;
- *dup.c* contient la définition de la fonction associée à l'appel système **dup()** ;
- *errno.c* contient la déclaration de la variable `errno` ;
- *execv.c* contient la définition de la fonction associée à l'appel système **execv()** ;
- *open.c* contient la définition de la fonction associée à l'appel système d'ouverture d'un fichier **open()** ;
- *setsid.c* contient la définition de la fonction associée à l'appel système **setsid()** ;
- *string.c* contient des directives de compilation ;
- *wait.c* contient la définition de la fonction associée à l'appel système **wait()** ;
- *write.c* contient la définition de la fonction associée à l'appel système d'écriture sur un fichier **write()**.

Le répertoire *fs*

Le répertoire *fs* contient dix-huit fichiers, outre le fichier *Makefile* :

- *bitmap.c* contient le code permettant de gérer les tables de bits d'utilisation des nœuds d'information et des blocs :

Linux 0.01

```
/* bitmap.c contains the code that handles the inode and block bitmaps */
```

- *block_dev.c* contient le code permettant de gérer les périphériques bloc ;
- *buffer.c* contient le code permettant de gérer l'antémémoire de blocs :

```
/*
 *  'buffer.c' implements the buffer-cache functions. Race-conditions have
 * been avoided by NEVER letting an interrupt change a buffer (except for
 * the data, of course), but instead letting the caller do it. NOTE! As
 * interrupts can wake up a caller, some cli-sti sequences are needed to
 * check for sleep-on-calls. These should be extremely quick, though
 * (I hope).
 */
```

Linux 0.01

- *char_dev.c* contient le code permettant de gérer les périphériques caractère ;
- *exec.c* contient le code permettant d'exécuter un nouveau programme ;
- *fcntl.c* contient le code permettant de manipuler les descripteurs de fichiers ;
- *file_dev.c* contient les fonctions d'écriture et de lecture dans un fichier ordinaire ;
- *file_table.c* contient la déclaration de la table des fichiers ;
- *inode.c* contient la déclaration de la table des nœuds d'information en mémoire ainsi que les fonctions permettant de la gérer ;
- *ioctl.c* contient la déclaration de la table `ioctl[]` et quelques fonctions associées ;
- *namei.c* (pour *NAME I-node*) contient les fonctions permettant de nommer les fichiers ;
- *open.c* contient les fonctions permettant d'ouvrir et de changer les droits d'accès d'un fichier ;
- *pipe.c* permet de mettre en place les tubes de communication ;
- *read_write.c* contient les fonctions permettant de se positionner, de lire et d'écrire sur un fichier ;
- *stat.c* contient les fonctions permettant d'obtenir des informations sur un fichier ;
- *super.c* contient les définitions et les fonctions concernant les super-blocs ;
- *truncate.c* contient les fonctions permettant d'effacer un fichier ;
- *tty_ioctl.c* contient les fonctions permettant de paramétrer un terminal.

Le répertoire *mm*

Le répertoire *mm* contient deux fichiers, outre le fichier *Makefile* :

- *memory.c* contient les fonctions concernant la gestion des pages ;
- *page.s* contient la routine de service de l'interruption matérielle concernant le défaut de page :

```
/*
 * page.s contains the low-level page-exception code.
 * the real work is done in mm.c
 */
```

Linux 0.01

Le répertoire *tools*

Le répertoire *tools* contient un seul fichier : *build.c*. Il s'agit d'un programme C indépendant qui permet de construire l'image du noyau.

3 Vue d'ensemble sur l'implémentation

3.1 Caractéristiques

La version 0.01 de Linux n'a pas pour but d'être évoluée :

Architecture. Elle ne supporte que les micro-processeurs `80386` d'Intel et ses descendants, grâce à la compatibilité ascendante de ceux-ci. Elle ne gère évidemment que les systèmes à un seul micro-processeur.

Mémoire. Elle gère la mémoire grâce au mécanisme de pagination. La mémoire physique est limitée à 8 Mo, ce qui était déjà énorme pour 1991. Cette limitation peut cependant être étendue sans trop de modifications. La capacité de la mémoire physique n'est pas détectée par le noyau, il faut la configurer manuellement.

Disque dur. On ne peut utiliser que des disques `IDE` et seul le premier contrôleur est pris en charge. Comme pour la mémoire, les paramètres des disques durs doivent être entrés avant la compilation.

Périphériques. La version 0.01 ne gère que deux disques durs, la console (clavier et écran texte) et deux modems (*via* deux ports série). Elle ne gère ni le lecteur de disquettes, ni le port parallèle (donc pas d'imprimante), ni la souris, ni les cartes graphiques (autres que texte), ni les cartes son ou autres périphériques (ISA, PCI ou autre). Elle n'utilise pas la DMA (*Direct Memory Access*).

Gestion des processus. Linux 0.01 est multi-tâches et multi-utilisateurs. Il peut gérer 64 tâches simultanées (ce nombre est aisément extensible) et 65 536 utilisateurs. Aucun utilitaire gérant les utilisateurs (`login`, `su`, `passwd`,...) n'est fourni et une seule console est implémentée.

Système de fichiers. Le système de fichiers utilisé par Linux 0.01 est celui de la première version de Minix. Il gère des fichiers avec des noms de 14 caractères au plus et une taille maximale de 64 Mo par fichier.

Réseau. Le support réseau n'était pas implémenté sur Linux 0.01.

La toute première version est donc rudimentaire du point de vue de l'utilisation mais elle est suffisante à titre pédagogique pour étudier le principe de l'implémentation d'un système d'exploitation.

3.2 Étapes de l'implémentation

Linus Torvalds ne donne aucune indication sur l'implémentation de son système. L'étude des sources nous conduit à distinguer les étapes suivantes, ce qui correspondra au plan de notre étude.

Le système d'exploitation est presque entièrement écrit en langage C, mais il existe quelques fichiers et quelques portions de fichiers écrits en langage d'assemblage, et ceci pour deux raisons : soit pour piloter les périphériques, soit pour tenir compte des particularités du micro-processeur Intel `86386`. Ces particularités sont encapsulées dans des macros ou des fonctions C. La seconde partie de notre étude consiste à étudier ces particularités.

Dans le chapitre 4, nous voyons comment l'accès à la mémoire vive est encapsulée dans des macros et comment la segmentation est utilisée sous Linux. L'étude de la pagination est reportée au chapitre 17 sur l'utilisation de la mémoire virtuelle sous Linux. Dans le chapitre 5, nous

voyons comment l'accès aux ports d'entrée-sortie est encapsulé dans des macros et comment les interruptions, que ce soit les exceptions réservées par Intel, les interruptions matérielles ou la seule interruption logicielle de Linux sont initialisées sous Linux, sans étudier, pour l'instant, les gestionnaires associés.

La troisième partie de notre étude est consacrée aux grandes structures de données utilisées par Linux. Dans le chapitre 6, nous étudions en détail la structure des descripteurs de processus, la table des processus et la tâche initiale, c'est-à-dire ce qui concerne l'aspect statique des processus en mode noyau. Dans le chapitre 7, nous étudions la mise en place des fichiers, c'est-à-dire ce qui concerne l'aspect statique des fichiers en mode noyau, plus exactement nous entreprenons une étude générale des fichiers dans les divers types de systèmes d'exploitation, les caractéristiques des fichiers sous UNIX, la structure d'un disque MINIX (qui est le seul système de fichiers accepté par le noyau 0.01 de Linux), les structures de données liées aux fichiers en mode noyau (antémémoire, nœuds d'information, super-blocs et descripteurs de fichiers) et, enfin, la façon dont on désigne les fichiers de périphériques sous Linux. Dans le chapitre 8, nous étudions la mise en place des terminaux à haut niveau, ceci regroupant à la fois l'encapsulation du clavier, de l'affichage sur le moniteur et des deux liaisons série. Nous n'entrons pas, dans ce chapitre, dans le détail des pilotes pour ces trois types de périphériques.

La quatrième partie est consacrée à la mise en place de l'aspect dynamique du mode noyau qui ne donne pas lieu à affichage en cas d'erreur (tout simplement parce que nous n'avons pas vu comment celui-ci est mis en place). Dans le chapitre 9, nous voyons comment les appels système sont mis en place, sans les étudier un par un pour l'instant. Dans le chapitre 10, nous étudions la mise en place de la mesure du temps, que ce soit l'horloge temps réel ou les minuteurs. Dans le chapitre 11, nous étudions la commutation des tâches et l'ordonnancement des processus. Dans le chapitre 12, nous étudions la notion générale de signal puis la mise en place des signaux sous Linux.

La cinquième partie est consacrée à l'affichage. Dans le chapitre 14, nous étudions la mise en place du pilote d'écran sous Linux. Dans le chapitre 15, nous étudions la mise en place de l'affichage formaté, ce qui nous conduit à étudier la mise en place des fonctions de bibliothèque ayant un nombre variable d'arguments.

La sixième partie est consacrée à la mise en place de l'aspect dynamique du mode noyau faisant intervenir l'affichage de messages d'erreur. Dans le chapitre 16, nous étudions les gestionnaires des exceptions sauf celui concernant le défaut de page, reporté dans le chapitre suivant. Dans le chapitre 17, nous étudions la notion de mémoire virtuelle de façon générale puis sa mise en place sous Linux.

La septième partie est consacrée à l'étude des fichiers réguliers. Dans le chapitre 19, nous étudions la notion de cache du disque dur et sa mise en place sous Linux. Dans le chapitre 18, nous étudions la mise en place du pilote du disque dur, c'est-à-dire l'accès au disque dur à bas niveau. Dans le chapitre 20, nous étudions la mise en place des périphériques bloc, c'est-à-dire l'accès au disque dur à haut niveau. Dans le chapitre 21, nous étudions la gestion des nœuds d'information. Dans le chapitre 22, nous étudions la gestion des fichiers réguliers et des répertoires.

La huitième partie est consacrée à l'étude des périphériques caractère. Dans le chapitre 23, nous étudions le pilote du clavier. Dans le chapitre 24, nous étudions le pilote des liaisons série. Dans le chapitre 25, nous étudions les périphériques caractère.

La neuvième partie, à chapitre unique 26, est consacrée à l'étude de la communication par tubes entre processus.

La dixième partie est consacrée à la mise en place du mode utilisateur, c'est-à-dire à la mise en place des appels système et des fonctions de bibliothèques. Dans le chapitre 27, les appels système concernant le système de fichiers sont mis en place. Dans le chapitre 28, les appels système concernant les processus sont mis en place. Dans le chapitre 29, les autres appels système sont mis en place. Dans le chapitre 30, les fonctions de la bibliothèque C sont mises en place.

La onzième partie, à chapitre unique 31, est consacrée au démarrage du système.

4 Évolution du noyau

Linux 2.2.18

Les sources du noyau 2.2.18 occupent 4 500 fichiers de C et de langage d'assemblage contenus dans près de 270 sous-répertoires ; elles totalisent quelques deux millions de lignes de code représentant près de 58 Mo.

4.1 Cas du noyau 2.4.18

Linux 2.4.18

Les sources du noyau 2.4.18 occupent 122 Mo. Le premier niveau de l'arborescence est aussi simple que dans le cas du premier noyau :

- `/arch` concerne tout ce qui dépend de l'architecture de la puce, Linux ayant été adapté à plusieurs micro-processeurs ; c'est dans ce répertoire qu'on retrouve ce qui a trait au démarrage ;
- `/Documentation` contient de la documentation, en particulier sur les périphériques pris en compte ;
- `/drivers` renferme les divers pilotes de périphériques ;
- `/fs` contient ce qui concerne les systèmes de fichiers, plusieurs systèmes de fichiers étant pris en compte et non plus seulement Minix ;
- `/include` renferme les fichiers d'en-têtes, dont beaucoup dépendent d'une architecture de micro-processeur donnée ;
- `/init` ne contient toujours qu'un seul fichier `main.c` ;
- `/ipc` renferme la mise en place d'un mode de communication entre processus qui n'était pas pris en compte lors du noyau 0.01 ;
- `/kernel` a un contenu assez proche de ce qui s'y trouvait pour le noyau 0.01 ;
- `/lib` a toujours la même fonction ;
- `/mm` également, mais compte un peu plus de fichiers ;
- `/net` concerne la mise en place des réseaux, principalement de TCP/IP, thèmes non abordés lors du noyau 0.01 ;
- `/scripts` renferme un certain nombre de scripts.

Le contenu des répertoires `/boot` et `/tools` est passé dans le répertoire `/arch`.

4.2 Aide au parcours du code source

Il n'est pas toujours facile de s'y retrouver dans le code source, en particulier pour savoir où telle constante ou telle fonction est définie. Un bon outil est le site Internet *Cross-Referencing Linux* : `http://lxr.linux.no/` en cliquant sur « *Browse the code* » ou, plus lent, *Linux Cross Reference* : `http://www.iglu.org.il/lxr/`

4.3 Cas du noyau 2.6.0

Donnons le premier niveau de l'arborescence des sources du noyau 2.6.0, qui comporte 5 929 913 lignes de code pour 212 Mo : Linux 2.6.0

- */Documentation*;
- */arch*;
- */crypto* est un nouveau répertoire qui concerne la cryptographie ;
- */drivers*;
- */fs*;
- */include*;
- */init*;
- */ipc*;
- */kernel*;
- */lib*;
- */mm*;
- */net*;
- */scripts*;
- */security* est un nouveau répertoire relatif à la sécurité ;
- */sound* est un nouveau répertoire traitant du son ;
- */usr* est un nouveau répertoire pour les fichiers auxiliaires.

On y trouve donc quatre nouveaux répertoires : deux d'entre eux (*sound* et *usr*) permettent de mieux structurer les sources ; les deux autres (*crypto* et *security*) prennent en compte un thème très à la mode.

Conclusion

Il existe de très nombreux systèmes d'exploitation. Les versions successives de chacun d'eux permettent d'une part d'améliorer ce que l'on peut appeler le micro-noyau du système et, d'autre part, de prendre en compte les changements essentiels dans le matériel (par exemple les réseaux ou les périphériques USB). Nous avons expliqué pourquoi il vaut mieux s'intéresser, dans une première étape, au tout premier noyau, la version 0.01 de Linux, pour enchaîner sur ses évolutions (il en est actuellement à sa version 2.6). Nous verrons dans les deux chapitres suivants en quoi un système d'exploitation dépend du micro-processeur.