1

何が新しくなったのか，何が追加されたのか

カーネルの役割とLinux Kernel2.6の新機能

岸 哲夫

カーネルといえば，OSの中核部分として，アプリケーション・ソフトウェアや周辺機器，ディスクやメモリなどの資源の管理，割り込み処理，プロセス間通信など，オペレーティング・システムとしての基本機能を提供するものです．本章では，Linuxカーネルの概要を説明し，Kernel2.6での変更点を説明します．現在の最新版Kernelは2005年3月にリリースされた2.6.11です．（筆者）

Linuxを構成する要素を大きく分けると以下のようになります．

- カーネル
  メモリ，プロセスの管理など
- ドライバ
  あらゆる種類のデバイスにアクセスするインターフェース
- シェル
  ユーザ・インターフェース
- デーモン
  バックグラウンドで動作するサービス
- コマンド
  ユーザが目的の処理を行うための機能

どのようなOSにも主記憶上にロードされ，ユーザ・アプリケーションにさまざまなサービスを提供するソフトウェアが存在しますが，Linuxの場合はそれをカーネルと呼びます．

カーネルは一般に，LILOやGRUBというブート・ローダによって主記憶上にロードして実行されます．筆者は，Fedora2をPentium III 800MHzで動作させています．ブート・ローダにはGRUBを使い，この設定ファイルは/boot/grub/grub.confに指定されています（**リスト1**）．

リスト1の中でkernel /vmlinuz-2.6.5-1.358とありますが，これがカーネルの本体です．また，/bootの下にはカーネルの関連ファイルがあります（**図1**）．

このディストリビューションの場合，バージョンは2.6.5です．

カーネルのソース・コードは/usr/src/linuxにあります．一般的なオプションでインストールすれば，そこにソース一式が入るはずです．

カーネルから見た場合，カーネルがメイン・ルーチンならばすべてのLinux上で実行されているタスクはそこから呼び出されることになります．それぞれのタスクは互いのタスクの資源にアクセスし，消費し，改変します．

OSの利用者から見た場合，ps -alとコマンドを入力して表示される各プロセスは，そのままそのマシン上で動いている「プログラム」です．し

リスト1　grub.conf

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/hda2
#          initrd /initrd-version.img
#boot=/dev/hda
default=0
timeout=10
splashimage=(hd0,0)/grub/splash.xpm.gz
title Fedora Core (2.6.5-1.358)
        root (hd0,0)
        kernel /vmlinuz-2.6.5-1.358 ro root=LABEL=/ rhgb quiet
        initrd /initrd-2.6.5-1.358.img
```

```
# ls /boot -Al
合計 1657
-rw-r--r--  1 root root  239593  5月  8  2004 System.map-2.6.5-1.358
-rw-r--r--  1 root root   46375  5月  8  2004 config-2.6.5-1.358
drwxr-xr-x  2 root root    1024 11月 19 05:08 grub
-rw-r--r--  1 root root  188042 11月 19 04:40 initrd-2.6.5-1.358.img
drwx------  2 root root   12288 11月 19 13:35 lost+found
-rw-r--r--  1 root root 1199031  5月  8  2004 vmlinuz-2.6.5-1.358
#
```

図1　/bootの下にあるファイル

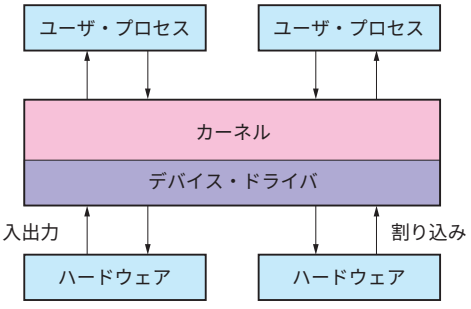


図2　カーネルの概念

KEYWORD —Linux Kernel 2.6，μClinux，プリエンプション，マルチプロセッサ，ジャーナリング・ファイル・システム

1 カーネルの役割と Linux Kernel 2.6の新機能

```
# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      33) SIGRTMIN    34) SIGRTMIN+1
35) SIGRTMIN+2  36) SIGRTMIN+3  37) SIGRTMIN+4  38) SIGRTMIN+5
39) SIGRTMIN+6  40) SIGRTMIN+7  41) SIGRTMIN+8  42) SIGRTMIN+9
43) SIGRTMIN+10 44) SIGRTMIN+11 45) SIGRTMIN+12 46) SIGRTMIN+13
47) SIGRTMIN+14 48) SIGRTMIN+15 49) SIGRTMAX-15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
#
```

図3　利用できるシグナル

```
# ls | sort | more
DIR_COLORS
DIR_COLORS.xterm
FreeWnn
Muttrc
X11
a2ps-site.cfg
a2ps.cfg
acpi
adjtime
aep
aep.conf
aeplog.conf
alchemist
aliases
aliases.db
alternatives
anacrontab
ant.conf
asound.state
at.deny
auto.master
auto.misc
auto.net
bashrc
blkid.tab
bonobo-activation
canna
cdrecord.conf
cpuspeed.conf
cron.d
cron.daily
cron.hourly
```

図4　パイプ処理

かし，OSの内部から見た場合にはカーネルのタスクなのです．簡単に言えば，こうしてマルチタスクが実現されているのです．

つまりカーネルとは，OSそのものであるとも言え，Linuxの存在そのものであるプログラムです(**図2**)．

カーネルのしくみ

PCを例にとって説明します．これはほかの組み込み用のボードで動かすときにも同じような処理になります．

起動時の処理は少し複雑になっています．まず電源を入れると，BIOSが起動します．ディスプレイを利用できるように初期化し，ハードディスク・コントローラを初期化します．CPU側から見ると0xfffffff0番地にジャンプし，ROMの中のBIOSを実行しているわけです．その後，ハードディスクのMBR(マスタ・ブート・レコード)から起動用のコード(ブートストラップ)を読み込んで実行します．

カーネルは，起動されると必要な初期化処理を実行してからイベント待ちになります．そして，大きく分けて以下のような事象が起きるのを待ちます．

- プロセス間通信(シグナル)
- システム・コール
- 割り込み

● プロセス間通信(シグナル)のしくみ

まず，プロセス間通信(シグナル)について，ごく簡単に説明します．カーネルから見たシグナルはプロセスの状態を変化させるために使用されます．それは逆にプロセスから見た場合，ソフトウェア割り込みの一つでもあります．

シグナルとパイプがプロセス間通信の代表例です．

シグナルは，プロセスに対して非同期イベントを伝達するために使用されます．そのシステムで利用できるシグナルはkill -lとコマンドを入力すると表示されます(**図3**)．

たとえば，sighupはログアウトしたときにその端末がもっているシェルから実行したプロセスにsighupが通知され，通常はそのプロセスが終了します．つまり，シェルのプロセスと終了させたいプロセスとの間で通信をしているわけです．

パイプとは，たとえばあるコマンドの出力をパイプ処理して別のコマンドの標準入力に送り込むしくみのことです．通常はこのような使い方をしませんが，例として`ls | sort | more`とコマンドを打つと**図4**のようになります．

● システム・コール ――カーネルの機能を実行

システム・コールとは，ユーザがカーネルの機能を実行させるための手続きのことです．簡単に言うと，主記憶に常駐している「タスク構造体」の中身を読み出したり，画面に文字を表示したり，データ・ファイルに書き込みを行う際に使用します．C言語のライブラリ関数などが実際にシステム・コールを発行するので，一般のシステム開発ではブラック・ボックスになっていますが，よりカーネル寄り，システム寄りのシステムを開発する際には理解していなければならない機能です．システムがサポートしているシステム・コール一覧は/usr/src/linux-2.6.5-1.358/include/asmi386/unistd.hを見るとわかります．**リスト2**にその一部を示します．

このソースを編集することで，新しいシステム・コールをカーネルに追加することもできます．

● 割り込み ――ハードウェア割り込み

割り込みとは，ここではハードウェア割り込みのことを言います．簡単に言えば，たとえばLANからパケットを受け取ったとき，キーボードからキーを打鍵したとき，そしてタイマ割り込みなどが，これにあたります．それぞれの事象が起きた際にどのタスクを起動するかを記述してあるのが割り込みハンドラです．

このように，カーネルはLinuxシステムの基幹であり，これが安定していないと大きな問題となります．

また，Linuxの機能そのものなので，カーネルを変更すれば新しい機能を追加することが可能になります．

Kernel2.6の新機能

Kernel2.4と比較して，カーネルの内部仕様が大幅に変更さ

Information ――「TOPPERS/FI4 カーネル」がローランド(株)のディジタル・ピアノに採用
TOPPERSプロジェクトが開発したRTOS「TOPPERS/FI4 カーネル」がローランド(株)のディジタル・ピアノに採用された．また，TOPPERSプロジェクトは，5/27にアルカディア市ヶ谷(東京都千代田区)にて「TOPPERS カンファレンス 2005」を開催する．

リスト2　システム・コールの一覧（抜粋）

```
#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
#define __NR_mknod               14
#define __NR_chmod               15
#define __NR_lchown              16
#define __NR_break               17
#define __NR_oldstat             18
#define __NR_lseek               19
#define __NR_getpid              20
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
#define __NR_ptrace              26
#define __NR_alarm               27
#define __NR_oldfstat            28
#define __NR_pause               29
#define __NR_utime               30
#define __NR_stty                31
#define __NR_gtty                32
#define __NR_access              33
#define __NR_nice                34
#define __NR_ftime               35
#define __NR_sync                36
#define __NR_kill                37
#define __NR_rename              38
#define __NR_mkdir               39
#define __NR_rmdir               40
#define __NR_dup                 41
#define __NR_pipe                42
#define __NR_times               43
#define __NR_prof                44
#define __NR_brk                 45
#define __NR_setgid              46
#define __NR_getgid              47
#define __NR_signal              48
#define __NR_geteuid             49
#define __NR_getegid             50
#define __NR_acct                51
#define __NR_umount2             52
#define __NR_lock                53
#define __NR_ioctl               54
#define __NR_fcntl               55
#define __NR_mpx                 56
#define __NR_setpgid             57
#define __NR_ulimit              58
#define __NR_oldolduname         59
#define __NR_umask               60
#define __NR_chroot              61
#define __NR_ustat               62
#define __NR_dup2                63
#define __NR_getppid             64
#define __NR_getpgrp             65
#define __NR_setsid              66
#define __NR_sigaction           67
#define __NR_sgetmask            68
#define __NR_ssetmask            69
#define __NR_setreuid            70
#define __NR_setregid            71
#define __NR_sigsuspend          72
#define __NR_sigpending          73
#define __NR_sethostname         74
#define __NR_setrlimit           75
#define __NR_getrlimit           76  /* Back compatible 2Gig limited rlimit */
#define __NR_getrusage           77
#define __NR_gettimeofday        78
#define __NR_settimeofday        79
#define __NR_getgroups           80
#define __NR_setgroups           81
#define __NR_select              82
#define __NR_symlink             83
#define __NR_oldlstat            84
#define __NR_readlink            85
#define __NR_uselib              86
#define __NR_swapon              87
#define __NR_reboot              88
#define __NR_readdir             89
#define __NR_mmap                90
#define __NR_munmap              91
#define __NR_truncate            92
#define __NR_ftruncate           93
#define __NR_fchmod              94
#define __NR_fchown              95
#define __NR_getpriority         96
#define __NR_setpriority         97
#define __NR_profil              98
#define __NR_statfs              99
#define __NR_fstatfs            100
                        （以下略）
```

れています．また，JFSやXFSといったファイル・システムが追加され，USAGI（IPv6対応）の統合，新デバイスのサポートなども行われました．**表1**（章末）に，これまでLinux Kernelがたどってきた道筋を示します．

では，Kernel2.6で，どのような機能が追加されたのかを説明します．

● マルチプロセッサ・システムを構成するには

Kernel2.6では，CPUが32個でも問題なく動作するそうです．

●スケジューラ

Kernel2.4までは，システム全体で一つの実行可能なプロセスのリストを用いてスケジューリングを行う，非常にシンプルなスケジューラが使用されてきました．しかし，この方式では，プロセスの数が多くなればなるほどスケジューラの処理に時間がかかってしまうという問題がありました．

この問題を解決するために，Kernel2.6からはスケジューリング方式が変更されました．プロセス優先度でソートされたプロセス・キューの配列でラン・キューを実装し，優先度の高いプロセスから動作させる構造になりました．また，プロセッサごとにラン・キューをもつように変更されたので，マルチプロセッサ時の性能向上も期待されます（**図5**）．

論理的には，二つのプロセッサをもつ，Intel社のPentium4で採用されている，HTテクノロジの場合でも効率が良くなります．

●RCU（Read Copy Update）

Kernel2.6では，RCUと呼ばれる新しい同期機構が追加されました．

カーネルには，複数のプロセッサによる同時アクセスからデータ構造を保護するため，スピンロックをはじめとする同期機構が用意されています．従来のスピンロックではロック/アンロックの操作にデータの書き換えを伴うため，プロセッサ間のキャッシュ同期処理が必要となり，大規模なマルチプロセッサ環境では大きな負荷になる場合がありました．

この問題を解決するために，Kernel2.6 から導入されたのが RCU という考え方です．簡単に説明すると，データ読み込み処理の際にはロックをせず，データ書き込みの際にのみロック処理を行う排他処理方式のことです．処理にかかるメモリ・アクセスを少なくできるため，効率が良くなります．

● pagevec

Kernel2.6 では，複数のページをバッチ的に処理するための pagevec という構造体が導入されました．これにより，システム・グローバルな構造の更新を抑えることができ，システムの効率が良くなりました．

● マルチスレッド対応のソフトウェアを生かす

● NGPT

Next Generation POSIX Threading（NGPT）は，その名のとおり次世代 POSIX スレッドと呼ばれる規格名です．

「POSIX pthreads liblary」と呼ばれるライブラリを利用可能なマルチスレッド・アプリケーションにおいて，パフォーマンスの向上が期待できます．

従来から Linux では POSIX に準拠したマルチスレッドを用意していましたが，Kernel2.6 では NGPT がサポートされます．これによりマルチスレッドに対応したソフトウェアのパフォーマンスの向上が期待できるほか，1 台のサーバ上で複数のプロセッサを稼働させる SMP（Symmetric MultiProcessor）環境での機能向上に役立ちます．

簡単に言えば，スレッドをスリープさせた場合，実アドレスが刻々と変化しますが，その実アドレスと，仮想アドレスの変換を行う関数が充実したのです．これが「futex システム・コール」と呼ばれるものです．この機能でスレッドの同期実時間が減少し，スレッドを用いたアプリケーションの性能が良くなるはずです．

● vcache

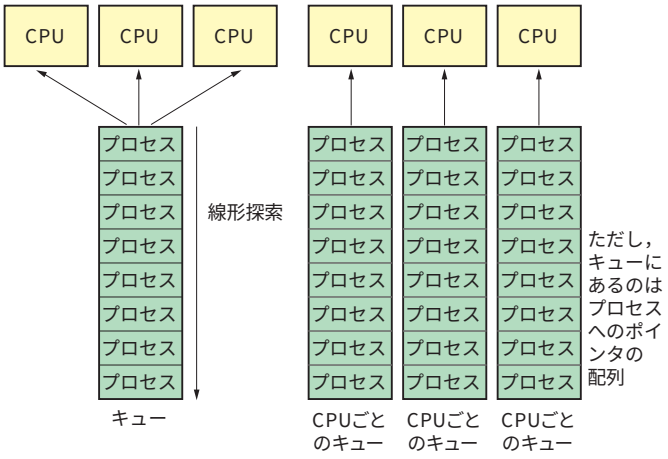
前述の futex のための機能追加です．仮想アドレスと物理ページの対応付けの変更をコール・バック関数によって通知するしくみです．

● 入出力の管理――ブロック I/O の改善

● Multi page block I/O

Linux Kernel2.4 までは，ブロック入出力を行うキューが一つしかなかったので，キューが使用中のときにはロックされてしまい，ほかのブロック・デバイスに対する入出力はロックが外れるまで待たなければなりませんでした．また，ページ・キャッシュ層からブロック I/O 層へリクエストを渡す際，複数ページからなるバッファをそのまま渡すことができず，いったんブロック・サイズに分割したバッファをブロック I/O 層で再びつなぎ合わせるというむだな処理を行っていました．

Kernel2.6 では，このブロック I/O のしくみを改善し，複数のブロック・デバイスに対してデバイスごとに並列にブロック I/O 処理できるようになりました．複数ページからなるバッファに対する I/O を効率的に行うため，BIO と呼ばれる構造体が新設されました．



（a）Kernel2.6 以前のスケジューラ　（b）Kernel2.6のスケジューラ

図5　スケジューラ

これは，従来の Buffer head に相当するものですが，ページの配列をもつようになっていて，複数のページをまとめてブロック I/O 層へ渡すことができます．また，これに合わせた各デバイス・ドライバの修正も行われ，Direct I/O のコードも BIO を使用するように大幅に書き直されました．これにより，ブロックへの分割と再構成をするといったむだな処理を省くことが可能になりました．このブロック I/O の改善は，多くのハードディスクを搭載して入出力を頻繁に行うサーバ処理に適したものといえます．

図6のように I/O が簡略化されたことが，パフォーマンスの改善に大きく役立っています．

● 非同期入出力

Linux の AIO 関連システム・コール，および，ほぼ対応する POSIX AIO 関数は以下のようになっています．

システム・コール	POSIX AIO 関数
io_setup	なし
io_destroy	なし
io_submit	lio_listio
io_cancel	aio_cancel
io_getevents	aio_suspend

Direct I/O と組み合わせて，データベースなどの用途で性能の向上が期待できます．

● readv/writev システム・コール

readv/writev については，地味な改良ですが，ログの出力など同期書き込みを多用する場面で性能向上が期待できます．

また，カーネル内部でも NFS サーバなどで使われています．Direct I/O との組み合わせでも動作するようになっているので，データベース用途などでの高速化も期待できます．Kernel2.6 では Ext2 などの主要なファイル・システムで readv/writev が実装されています．

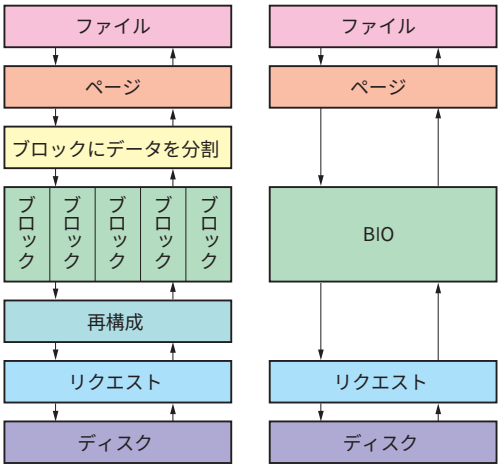


図6 ブロックI/Oの概略

(a) Kernel2.6以前のブロックI/Oの模式図　(b) Kernel2.6のブロックI/Oの模式図

新しい形式が追加された ジャーナリング・ファイル・システム

ジャーナルとは，ファイル・システムに対する変更などを記録するデータベースのことです．マシンの障害時などには，ファイル・システムの整合性をこのジャーナルから確認することにより，スムーズな復旧を促します．

従来のKernel2.4では，ジャーナリング・ファイル・システムとして，Ext3形式とReiserFS形式がサポートされています．そしてKernel2.6では，新たにIBM社が開発したJFS形式とSGI社が開発したXFS形式が標準カーネルに取り込まれています．

JFSはIBM社のAIXやOS/2で利用されているファイル・システムです．

● ジャーナル機能を追加したExt3形式ファイル・システム

Ext3ファイル・システムは，従来のExt2ファイル・システムにジャーナル機能を追加したものです．

たとえば，今まではシステムが異常終了した場合，再起動時fsckコマンドが動作して，長時間に渡ってファイル・システムをチェックしていましたが，Ext3ファイル・システムでは通常の起動と変わらない時間で起動します．

また，Ext3ファイル・システムの特徴の一つとして，Ext2ファイル・システムの上位互換であることが挙げられます．つまり，Ext3に対応していないカーネルであっても，Ext2ファイル・システムとしてのアクセスが可能です．また，ルート・ファイル・システムがExt3ファイル・システムの場合には，特に設定などを書き換える必要はありません．

それ以外のパーティションは，/etc/fstabにExt3ファイル・システムであることを明示しなければなりませんが，明示しなかったとしても，単なるExt2ファイル・システム・パーティションとしてアクセスできます．

以前のExt2を主として使用している場合には，カーネルのバージョンを上げてExt3パッケージを導入し，ファイル・システムの信頼性を上げるべきだと思います．

● 古典的なReiserFS形式ファイル・システム

古典的なBツリー・アルゴリズムを使ったファイル・システムです．特徴はファイルの高速アクセスやジャーナル機能です．

ジャーナル機能をもつファイル・システムのすべてについて言える利点ですが，突然の電源断などによるファイル・システムの不整合発生を防止できます．そのため，Windows95や98などで電源断してしまったときのように，fsckプログラムで延々とファイル・システムの整合性をチェックする必要がありません．

パフォーマンスとしては，小さいファイルが多いパーティションのディスク使用効率に優れている点が挙げられます．ReiserFSは従来よく使われてきたExt2と互換性がないので，ReiserFSパーティションにアクセスするにはOSの対応が必須となります．

● IBM社が開発したJFS形式ファイル・システム

IBM社が開発し，GPLで公開しているファイル・システムです．ディスク障害を高速に検知できるので，高スループットのサーバ環境に適しています．64ビット・ファイル・システムであることが特徴です．つまり，最大32Pバイトのディスク・スペースや，最大4Pバイトのファイル，Bツリーを利用したファイルの高速なアクセスが可能です．

また，迅速な障害復旧を可能にするメタ・データ・ジャーナリングにも対応しています．OS/2で利用していたJFSパーティションをそのまま利用することもできます．Ext2との互換性はありません．

● SGI社が開発したXFS形式ファイル・システム

SGI社のUNIX系OSであるIRIXで運用するために開発されたもので，現在ではGPLのもとでソース・コードが公開され，だれでも無償で利用できるようになっています．同社の公開したコードを元に，Linuxなどのほかの UNIX系OSへの移植が進んでいます．

ジャーナル機能に対応しているため，ファイル・システムの修復が速く，64ビット・ファイル・システムを採用しているため，最大1800Tバイトのディスク・スペースや，最大900Tバイトのファイルも扱うことができます．Ext2との互換性はありません．

上記のファイル・システムは次のような特徴をもっています．

●大規模ストレージに対応

4Tバイト以上のパーティションを作成できます．これだけあれば大規模なサーバが構築できます．

●大容量ファイル・サイズ対応

動画や音声配信にはたくさんの容量が必要ですが，XFS/JFSは4Tバイト以上のファイル・サイズをサポートしているので，安心して扱うことができます．

●POSIX ACLや拡張属性のサポート

POSIX ACLがサポートされることになりました．これはWindowsにおけるアクセス制御に似ています．

POSIX ACLの例として，**図7**のような操作が可能です．

●オンライン・リサイズ機能のサポート

システム実行中に，ファイル・システムがあるパーティションのサイズを変更する機能をオンライン・リサイズと言います．現在はXFSでオンライン・リサイズ機能が利用できます．ほかのファイル・システムでもサポートされる予定です．

```
# getfacl test.txt
# file: test.txt
# owner: kishi
# group: kishi
user::rw-
group::r--
other::r--

# setfacl -m u::w test.txt
# getfacl test.txt
# file: test.txt
# owner: kishi
# group: kishi
user::-w-
group::r--
other::r--

# setfacl -m u::wrx test.txt
# getfacl test.txt
# file: test.txt
# owner: kishi
# group: kishi
user::rwx
group::r--
other::r--
```

図7 POSIX ACLの例

ファイル・システムを共有化するネットワーク・ファイル・システム

NFS（ネットワーク・ファイル・システム）とは，ファイル・システムを共有するためのプロトコルです．

Kernel2.6では，ネットワーク・ファイル・システムの機能が強化されました．リモート・ホスト上のファイル・システムをマウントしたり，ローカルのファイル・システムをリモート・ホストからマウントできるようにする機能のことです．

● NFSv4のサポート

NFSv4がサポートされました．従来使用されていたNFSの上位互換です．ただし，まだクライアント機能だけが動作しています．

● AFSのサポート

AFSは，1980年代に開発された広域分散ファイル・システムですが，Kernel2.6で新たにサポートされました．

● CIFSクライアント機能のサポート

CIFS（Common Internet File System）は，Windowsのファイル共有などで使用されているプロトコルです．SMB（Server Message Block）の後継プロトコルであり，TCP/IP上に実装されています．

Kernel2.6では，CIFSのクライアント機能がカーネル自身に内蔵されています．これにより，通常のボリュームと同様，mountコマンドでWindowsの共有フォルダをマウントすることが可能となります．

● そのほかのネットワーク関連での追加機能

●zerocopy NFS

Kernel2.6では，ページ・キャッシュに使われているページを直接プロトコル層に渡し，可能であればそのままネットワーク・ドライバまで渡すようになりました．このため，むだなコピー処理のオーバヘッドが削減され，プロセッサ内部のキャッシュ・メモリからほかの有用なデータが追い出されることが減ることが期待できます．

●epoll

epollはpollの一種であり，代替となるべく追加されました．エッジ・トリガ・インターフェース，またはレベル・トリガ・インターフェースとして使うことが可能です．また，監視するファイル・ディスクリプタの数が多い場合にも使えます．epollセットを設定したり制御するために，epoll_create，epoll_ctl，epoll_waitの三つのシステム・コールが提供されています．

メモリ管理の効率化

● reverse map ――ページ・アウト処理の効率を改善

従来のカーネルでは，メモリ不足時のページ・アウト処理の際，全アドレス空間をスキャンしていました．ある物理ページがどのメモリ空間から参照されているか，そしてそのページがダーティであるか否かを簡単に知る方法がなかったのです．Kernel2.6では，page構造体にそのページを参照しているリストを保持することになったので，ページ・アウト処理の効率が改善されました．

● mempool ――ある分量のメモリをつねに確保

mempoolは，初期化時に指定したバックエンド・アロケータを使用し，指定したメモリをつねに保持しておくことで，ある分量のメモリがつねに確保できることを保証します．

これによって，高負荷時の安定性の向上が期待されます．メモリが不足した際に，不要なメモリの解放処理を行いますが，これがうまく動作するようになります．解放処理自体にさらなるメモリ割り当てが必要となる場合がありますが，これで問題なく動作するはずです．

対応するアーキテクチャの見直し

Kernel2.6では，64ビット系のCPUや，組み込み系のCPUなどもサポートされるようになりました．

● 64ビット系CPUへの対応

●IA-64 CPU対応の拡充

Kernel2.4からIA-64をサポートしていますが，Kernel2.6ではIA-64ネイティブのアプリケーションとIA-32のアプリケーションの両方が動作します．

●AMD系64ビットCPUに対応

Kernel2.6では，AMDのx86-64アーキテクチャ上でLinuxを動作させるための機能が追加されています．IA-32を64ビットに拡張したアーキテクチャを採用しています．64ビットのア

ドレス空間や64ビット拡張レジスタが利用できます．

●PowerPC64系CPU対応の拡充

非連続メモリがサポートされました．

● 組み込み系CPU対応の拡充

近年は携帯電話やディジタル家電において，組み込みLinuxの需要が大きくなってきています．このような背景を受けて，Kernel2.6では以下のような組み込み系分野への対応強化がなされています．

●組み込み系CPU

IBM PowerPC 405GPやIntel XScale系のCPUサポートが標準カーネルに統合されました．

● μClinuxの統合

MMUを搭載していないCPUでもUNIX/Linuxのアプリケーションを利用できるようにしたのがμClinux（uClinux）です．メモリ管理系の置き換えや，プロセス実行系の置き換えを行い，MMUなしで動作するように設定されています．もちろん，MMUが前提で動作するようになっているシステム・コールは動作しません．

ACPIによる電源管理

● ACPIによる電源管理

Kernel2.6では，ACPI（Advanced Configuration and Power Interface）に基づく電源管理がデフォルトとなります．電源ボタン押下時の動作設定をサスペンド/シャットダウンなどから選択できます．

● ソフトウェア・サスペンド機能

ソフトウェア・サスペンドは，実行時のメモリ・イメージをディスク（swapパーティション）に退避し，再起動時に復元することで実現されています．正常動作時の状態を保存しておき，異常発生時に復元するなどの使い方が考えられます．

対応デバイスの追加

Linuxは現時点でも多くのデバイスに対応していますが，Kernel2.6ではさらに多種多様なデバイスに対応するようになりました．カーネルの再構築を行えば，IEEE1394からDVカメラの映像キャプチャも可能です．

● USBドライバに対応

USB2.0に正式に対応しています．これでハイ・スピード転送（480Mbps）が可能になりました．USB-MIDIのサポートも追加されているので，フリー・ソフトウェアでソフト・シンセサイザを使ってマルチトラック録音を行うことも，MIDIでつないだ外部音源を制御することも十分に可能です．

● ALSAドライバに対応

従来のOSS（Open Sound System）よりも多くのデバイスに対応しています．また，OSS互換インターフェースをもっており，OSS準拠のアプリケーションの動作が可能です．そしてSMP環境での動作を保証しています．Kernel2.6では，ALSA（Advanced Linux Sound Architecture）ドライバによるサウンド・カード・サポート機能が標準のLinuxカーネルに統合されました．

● ネットワーク・デバイスへの対応

NAPIは，ネットワークに高負荷がかかった時点でドライバの動作を割り込み駆動からポーリング駆動に切り替えることで，ネットワーク負荷に伴うシステム全体の応答性能の低下を防ぎます．

カーネル・レベル・プリエンプション

組み込み分野でリアルタイムLinuxの需要もありますが，プロセスをせいぜい20ms周期で起床させるようなものであれば，リアルタイムLinuxを使う必要がないかもしれません．Kernel2.6にはそれだけの機能があります．

リアルタイムLinuxでの開発で混乱するよりは，Kernel2.6でスムースに開発を終えたほうが良いかもしれません．しかし，対象のシステムの状況にもよるので，十分な調査をしてから使用することを勧めます．

また，そのような用途であれば，カーネルの再構築でむだなコードを取り除き，最小の構成で動作させることを勧めます．

そのほかの追加機能

● Video For Linux ――ディジタルTV放送などに対応

Video For Linux（V4L）のAPIがバージョン2にアップしました．これはビデオ・キャプチャやディジタル・ビデオ放送受信用ハードウェアに対する対応が強化されたためです．なお，対応ビデオ・キャプチャ・カードの種類も大幅に増えました．SAA5249，Philips SAB3036，Stradis 4：2：2などのチップセット，またディジタル・テレビ放送の受信カードである，ALPS BSRU6/BSRV2/TDLB7の各DVBカードにも対応しました．xawtvやtranscode，xineをインストールしてDVDレコーダとして使うことも可能です．大画面で見たい場合は，「ネットワーク・プレーヤ」をつなぎましょう．

● LVMの強化

LVM（Logical Volume Manager）とは，複数の物理ボリューム（ディスク・パーティション）を論理的な単一のボリュームに見せかける技術です．Kernel2.4のものとは大幅に変わっています．

● IPv6を実装するなどTCP/IP関連の機能も強化

Kernel2.6のIPv6実装には，USAGIプロジェクトの成果が取り入れられました．現在のインターネット上ではIPv4方式でIPアドレスが附番されています．例として`192.168.0.1`は32ビットの番号が8ビットずつの10進数に分けられていま

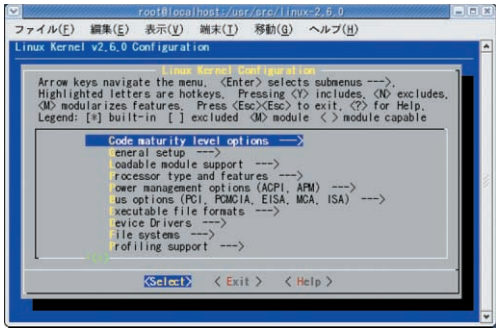


図8 menuconfig ── X Window System がなくても動作する設定

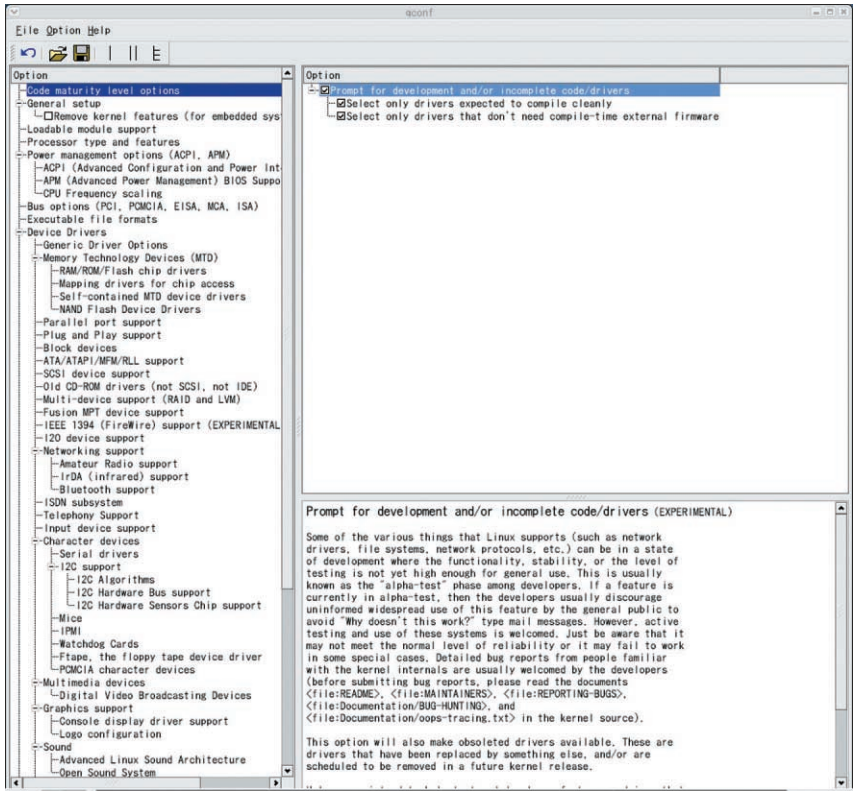


図9 xconfig

す．IPv6では128ビットの番号を16ビット単位に区切った16進数で表現します．これで接続できるコンピュータの数はおよそ10^{38}になります．IPv4方式では足りなくなる貴重なIPアドレスに余裕ができ，家電などに附番することが可能になります．

下記の機能が改善，または新規対応になりました．

- IPv6上のICMP機構
- 経路制御
- エニーキャスト
- モバイルIP機能
- IPv6 Over IPv6
- IPv6 Over ATM
- ネット・フィルタのIPv6サポート
- SCTP
- IPsec

Kernel2.6を再構築する

Kernel2.4とKernel2.6では，再構築の手順が少し違います．以下に，その手順を示します．

まず，ソース・ディレクトリに移動して，

```
#make mrproper
```

として初期設定をします．

```
#make menuconfig
```

これは，X Window Systemがなくても動作する設定です(**図8**)．そしてXを動かしている場合には，次のようにします．

```
#make xconfig
```

Kernel2.4のものとGUIが大きく変わりました(**図9**)．こちらのほうが見やすいと思います．これはQtベースで作られています．

また，新たに設定されたコンフィグ画面があります(**図10**)．これはGTKベースで作られています．

```
#make gconfig
```

Fedoraの Kernel2.4では存在したハイパースレッディング・

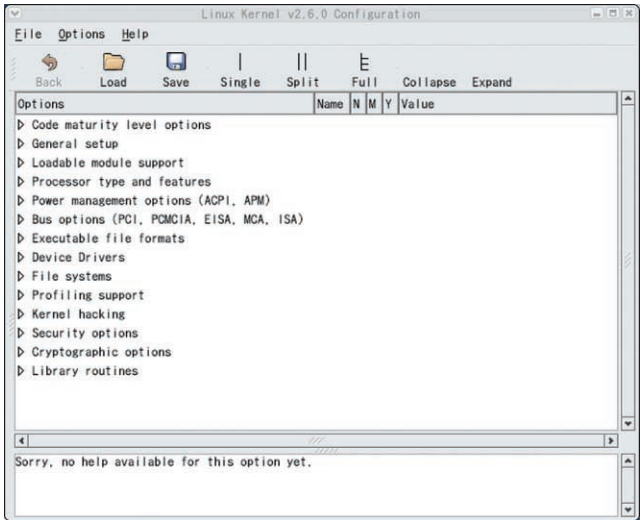


図10 gconfig

サポートの項目がなくなっています．つまり，Kernel2.6ではかならずサポートするということです．

ほかにも以下のようなものがありますが，理解して使えば便利です．

```
#make oldconfig
```

前回の.configファイルとの差分を設定します．

```
#make defconfig
```

前回の.configファイルとの差分をすべてデフォルトに設

New Products ──マルチプロセッサ対応の「TOPPERS/FDMP カーネル」が発表に
TOPPERS プロジェクトは，マルチプロセッサ対応の RTOS「TOPPERS/FDMP カーネル」を 4/18 に発表した．同 OS は，オープン・ソース・ソフトウェアとして配布される．

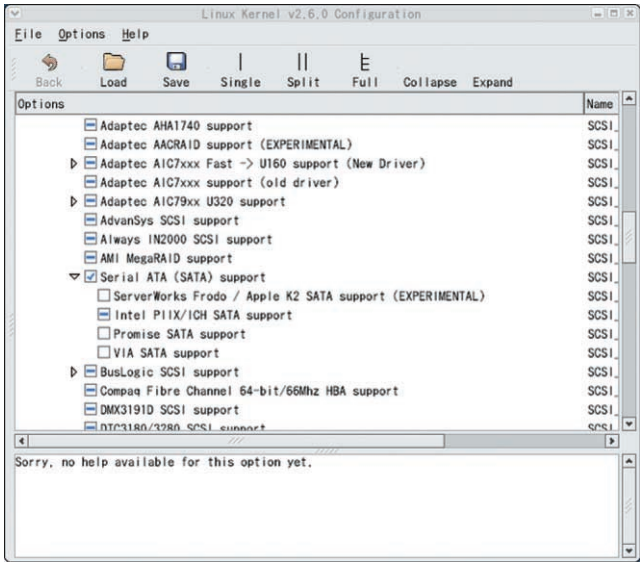


図11　シリアルATAの設定

定します．

```
#make allmodconfig
```

すべてmodule設定にします．

```
#make allyesconfig
```

すべてY設定にします．

```
#make allnoconfig
```

すべてN設定にします．

ところで，最近のマザーボードにはシリアルATAが装備されています．装備されている以上，使わなくてもドライバが必要になることを忘れてはいけません．使っているハードウェアを完全に把握していないと，困ったことになります．

図11のようにSCSIドライバの設定の中で，シリアルATAの設定を行います．HTテクノロジ対応のマザーボードには，たいていシリアルATAが装備されているはずなので，注意しなければなりません．この設定をモジュールにしておかないと時間をかけて動かないカーネルを作ることになってしまいます．

コンフィグレーションが終わったら，

```
#make
```

とするだけでカーネルもモジュールも作ってくれます．

```
#make modules_install
```

これでモジュールをインストールします．

```
#make install
```

カーネルを/bootの下にコピーし，initrdもコピーし，/boot/grub/grub.confも変更されます．

このようにとても簡単になっています．

Kernel2.6を使ってみる

ビジネスの分野はもとより，映像分野でも，音楽の分野でも，Kernel2.6が活躍することでしょう．たとえばオープン・ソースで作品を作るということは，環境の制約がないということです．

これで高速なハイパースレッディング採用のPCやXeonを数個使った環境で作品を次々と作ることができます．

音楽の分野ではオープン・オーディオ・ライセンスという流れができています．まるでオープン・ソースとして作られたライブラリのように，素材として，または作品として音楽を流通させている人々がいます．インターネット上でのコラボレートで作品を作ることも可能なのです．リミックス，サンプリングなどの技法によって，新たな楽曲を作成することもできます．そのような用途にもKernel2.6を使ったLinuxは大いに役立ちます．ネットワーク・ファイル・システムで互いのPCをマウントし，ほぼリアルタイムで作業ができます．ファイル・システムの規格である，POSIX ACLのサポートでWindowsとSambaで接続してWindowsで加工したデータの受け渡しが楽になるのです．

製作者側は，オープン・ソース・コミニュティが作り上げたもので，良い作品を残すことがお返しになるわけです．高速なスレッド処理は音楽の非破壊エフェクト処理をすばやく行い，製作者の思考を妨げることがないはずです．製作過程でのツールにとらわれないことで，作品の幅も広がると思います．

ディジタル楽器の世界でも，OSに組み込みLinuxを使ったものが一般的になり，操作性も良くなっていくことでしょう．スタジオの床に叩きつけたくなるような操作性のものが多いなか，なんとかそうなって欲しいものです．

従来のカーネルからKernel2.6にアップグレードする際は，ただ単にカーネルのソースを持ってきてコンパイルする方法ではうまくいかない場合があります．そんなときには，一気に新しいディストリビューションを入れてしまいましょう．

もちろん，カーネルから構築し，デバイス・ドライバや，そのほかのコマンドをすべてコンパイルする方法でも問題はありません．ファイル・システムも堅牢になるうえ，カーネルは安定していて，デメリットは特になさそうです．

Kernel2.6はリリースされて1年以上経ったので，そろそろ枯れてきていると思います．そのため，ビジネスにも映像/音声にもお勧めです．

● HTプロセッサのベンチマーク

Kernel2.6マルチプロセッサでパフォーマンスが良くなるとのことです．そこで，動画を処理する際に使うツールtranscodeを使って処理速度を測定してみることにします．

このソフトウェアは基本部分がマルチスレッド対応なので，試験するという目的には合っているはずです．

今回は，以下のような試験をしてみました．

- Kernel2.4での試験
  - ▶HT機能ON
    - ●transcodeを単独で実行
    - ●transcodeを並行して実行
  - ▶HT機能OFF
    - ●transcodeを単独で実行

Information――ザイリンクス，アットマークテクノとコンサルタント契約を締結

ザイリンクス(株)は，(株)アットマークテクノとコンサルタント契約を締結した．これにより，アットマークテクノは東京に事務所を設立し，5月から本格営業を開始する．

- transcode を並行して実行

● Kernel2.6 での試験

▶ HT 機能 ON

- transcode を単独で実行
- transcode を並行して実行

▶ HT 機能 OFF

- transcode を単独で実行
- transcode を並行して実行

HT 機能の ON/OFF は BIOS 設定で行うことができます．Kernel2.4 は CPU/マルチプロセッサ/HT 機能で最適化したものです．

Kernel2.6 は CPU/マルチプロセッサで最適化したものを使用しました．また，動画素材として，ディジタル・ビデオからキャプチャした 10 秒分の DV フォーマットで作成された動画を用いました．

図 12 に測定結果を示します．

この結果を見ると並行実行の結果は不定です．それは内部でどのようにマルチスレッド化がなされているかがわからない以上，参考にしかなりません．

Kernel2.6 で HT 機能を切って並列動作させないほうが良いという結果が出ました．

しかし，単独実行では予想したとおりの結果が出ました．

順位は次のとおりです．

(1) Kernel2.6　HT 機能 ON
(2) Kernel2.4　HT 機能 ON
(3) Kernel2.4　HT 機能 OFF
(4) Kernel2.6　HT 機能 OFF

Kernel2.6 では，マルチプロセッサシステム対応の強化，マルチスレッド対応の強化を謳っているだけあって，HT 機能 ON の場合の動作が特に速いようです．

カーネルのバージョン	HT 機能	単独実行/並行実行	CODEC	所要時間 (秒)	並行実行時の平均値 (秒)	対比結果 (秒)
2.4	ON	単独	Xvid	10	→	10
	ON	単独	DivX	12	→	12
	ON	並行 1	Xvid	21	25	25
	ON	並行 1	DivX	20	17.5	17.5
	ON	並行 2	Xvid	29	—	—
	ON	並行 2	DivX	15	—	—
	OFF	単独	Xvid	11	→	11
	OFF	単独	DivX	14	→	14
	OFF	並行 1	Xvid	18	16	16
	OFF	並行 1	DivX	32	26	26
	OFF	並行 2	Xvid	14	—	—
	OFF	並行 2	DivX	20	—	—
2.6	ON	単独	Xvid	9	→	9
	ON	単独	DivX	12	→	12
	ON	並行 1	Xvid	19	17	17
	ON	並行 1	DivX	18	21.5	21.5
	ON	並行 2	Xvid	15	—	—
	ON	並行 2	DivX	25	—	—
	OFF	単独	Xvid	13	→	13
	OFF	単独	DivX	15	→	15
	OFF	並行 1	Xvid	13	22.5	22.5
	OFF	並行 1	DivX	38	27	27
	OFF	並行 2	Xvid	32	—	—
	OFF	並行 2	DivX	16	—	—

図 12　エンコードによるベンチマーク

Kernel2.4 の結果を見てわかるとおり，HT 機能は ON にしたほうが速いという当然の結果が出ました．

また，HT 機能に最適化されている Kernel2.6 で HT 機能を切ると Kernel2.4 よりも遅くなるという結果が出ました．これも当然の結果です．

以上の結果を見ると，Linux で動画や音声を扱う際には，HT テクノロジ対応のシステムを使うと良いでしょう．

きし・てつお

表 1　Linux Kernel の歴史

1991 年 6 月	Linus Torvalds 氏によって MINIX を参考に Linux の原型が完成
	comp.os.minix ニュース・グループにてカーネルの性能や開発方針などについての論争があった
1991 年 9 月	Linux Kernel 0.01 公開
	参考 URL：http://www.vision25.demon.co.uk/prog/linuxbirthday.html
1991 年 10 月	Linux Kernel 0.02 リリース
	ソースだけでなく，バイナリもリリースされた．まだ MINIX を必要とした
1991 年 10 月	Linux Kernel 0.03 リリース
	Linux 上で gcc を使用することが可能になった
1991 年 11 月	Linux Kernel 0.10 リリース
	MINIX なしで動くようになった
1991 年 12 月	Linux Kernel 0.11 リリース
	デマンド・ページング，フロッピ・ドライバ，VGA ドライバ，mkfs/fsck/fdisk などが機能した
1992 年 1 月	Linux Kernel 0.12 リリース
	POSIX 準拠のジョブ・コントロール，仮想記憶の機能が追加された
1992 年 3 月	Linux Kernel 0.95 リリース
	BSD もこの時期に原型が完成した
1992 年 4 月	Linux Kernel 0.96 リリース
1992 年 8 月	Linux Kernel 0.97 リリース
1992 年 9 月	Linux Kernel 0.98 リリース
1992 年 10 月	Linux Kernel 0.99 リリース
	このころより日本のユーザの間で日本語化への模索が始まった
1994 年 2 月	Linux Kernel 0.99.15 リリース
	静かにユーザの間で話題になっていった
1994 年 3 月	Linux Kernel 1.0 リリース
	最初の公式版カーネルである
1994 年 4 月	Linux Kernel 1.1.0 リリース
	マイナ・バージョンが偶数のバージョンは安定性を重視した一般用，奇数のバージョンは機能追加をめざす開発用，と位置づけられた
1995 年 3 月	Linux Kernel 1.2.0 リリース
	このころより数多くの雑誌や書籍で Linux が取り上げられた
1996 年 5 月	Linux Kernel 2.0 リリース
	このころより一般雑誌にも取り上げられるなど，パーソナル・ユースとして Linux が多くのユーザに使われ始めた
	この間，無数のマイナ・バージョンアップ，パッチがいろいろなディストリビュータから配布された
1998 年 12 月	Linux Kernel 2.0 2.2.0-pre-patch1 リリース
	マルチプロセッサに対応した
	このころ MkLinux kernel GENERIC-06 がリリースされた

表1　Linux Kernelの歴史（つづき）

1999年1月	Linux Kernel 2.2.0 リリース
	この版には「セキュリティ・ホール」が存在したためすぐに2.2.1がリリースされた
	このころORACLEが「Oracle8 Workgroup Server for Linux Release8.0.5」を出荷した
1999年2月	Linux Kernel 2.2.2 リリース
1999年3月	Linux Kernel 2.23，2.2.4，2.2.5 リリース
1999年4月	Linux Kernel 2.2.6，2.2.7 リリース
1999年5月	Linux Kernel 2.2.8 リリース
	Cyrixや AMD社のプロセッサに対応した
	なお，この版には「ファイル・システムのバグ」が存在したためすぐに2.2.9がリリースされた
	開発版カーネルとして2.3.0から2.3.3が公開された
1999年6月	Linux Kernel 2.2.10 リリース
	ネットワーク経由のDoS攻撃に対処した
	開発版カーネル2.3.4から2.3.9まで公開された
1999年7月	開発版カーネル2.3.10から2.3.12まで公開された
1999年8月	Linux Kernel 2.2.11，2.2.12 リリース
	開発版カーネル2.3.13から2.3.15まで公開された
1999年9月	開発版2.3シリーズのカーネルは2.3.18で機能の追加は終了とLinus Torvalds氏が発表した
	新機能は2.4に盛り込まれることとなる
1999年10月	Linux Kernel 2.2.13 リリース
	開発版カーネル2.3.19から2.3.24まで公開された
1999年11月	開発版カーネル2.3.25から2.3.27まで公開された
1999年12月	開発版カーネル2.3.28から2.3.35まで公開された
2000年1月	Linux Kernel 2.2.14 リリース
	AMD Athlonプロセッサ用のチップセットである「AMD-751」のサポートやIntel Coppermineプロセッサの L2キャッシュに関するバグ・フィックスがなされた開発版カーネル2.3.36から2.3.41まで公開された
2000年2月	開発版カーネル2.3.42から2.3.48まで公開された
2000年3月	開発版カーネル2.3.49から2.3.99まで公開された
2000年5月	Linux Kernel 2.2.15 リリース
	開発版カーネル2.4.0-test1がリリースされた
2000年6月	Linux Kernel 2.2.16 リリース
	任意のコードを実行できるユーザがsetcap(2)システム・コールを通してroot権限を奪取できるという2.2.15のセキュリティ・ホールが修正された
	開発版カーネル2.4.0-test2がリリースされた
2000年7月	開発版カーネル2.4.0-test3から2.4.0-test5まで公開された
	2.4.0-test5でFATファイル・システムのファイル名をeuc-jpやshift_jisに変換する機能が追加された
	簡単に言えば日本語ファイル名を含むWindowsのファイル・システムをLinuxで扱うことが楽になった
2000年8月	開発版カーネル2.4.0-test6から2.4.0-test7まで公開された
2000年9月	Linux Kernel 2.2.17 リリース
	開発版カーネル2.4.0-test8から2.4.0-test8まで公開された
2000年10月	開発版カーネル2.4.0-test9から2.4.0-test10まで公開された
2000年11月	開発版カーネル2.4.0-test11まで公開された
2000年12月	Linux Kernel 2.2.18 リリース
	開発版カーネル2.4.0-test12まで公開された
2001年1月	Linux Kernel 2.4 リリース
	バグ・フィックス版2.4.1もリリースされた
2001年2月	Linux Kernel 2.4.2 リリース
	IBMの汎用機S/390への対応，ReiserFSの改良，USB関連の改良などが行われた
	米Red Hat社がカーネル2.4搭載のRed Hat Linux 7.0.9公開ベータ「Fisher」を国内配布
2001年3月	Linux Kernel 2.4.3 リリース
	USB関連の改良などが行われた
2001年4月	Linux Kernel 2.4.4 リリース
	Raw I/Oのフィックス，USBのアップデート，ReiserFSのアップデートが行われた
2001年5月	Linux Kernel 2.4.5 リリース
	米VERITAS Software社のファイル・システム「VxFS」に対応するなどの変更が施された
2001年7月	Linux Kernel 2.4.6 リリース
	Ext2ファイル・システムのディレクトリの内容をページ・キャッシュに移行した
	iノード割り当ての効率化を図った
	“knfsd”をアップデートし，ReiserFSへの出力を可能にした
	Linux Kernel 2.4.7 リリース
	IEEE1394 GUIDのcleanupを行った
	completion handler用IDEテープ・ドライバのアップデートを行った
2001年8月	Linux Kernel 2.4.8 リリース
	「EMU10K」チップ用サウンド・ドライバを統合した
	スワップイン/スワップオフの競合状態を解消した
	VMのバランシングを改善した
	Linux Kernel 2.4.9 リリース
	Sparcのアップデートを行った
	米ORiNOCOのドライバのアップデートを行った
	FAT fs，btaudio buildのバグ・フィックスを行った
2001年9月	Linux Kernel 2.4.10 リリース
	jffs2とNTFSのアップデートを行った
	ACPIのアップデートを行った
	“min()/max()”の最新バージョンへの変更を行った
	新しいmultipath RAID personalityを導入した
	バーチャル・メモリのアップデートを行った
2001年10月	Linux Kernel 2.4.11 リリース
	symlink attachの修正を行った
	emu10kドライバのアップデートを行った
	しかし，このバージョンにはバグがあったので，すぐに2.4.12がリリースされた
	Linux Kernel 2.4.13 リリース
	page write-outのスロットリングに対応した
	ymfpciサウンド・ドライバのアップデートを行った
	I2Oのsync-upに対応した
	x86 smp_call_functionパッチを復活させた
	handle VM write loadを向上させた
2001年11月	Linux Kernel 2.4.14 リリース
	SPARCとSCSIに関するさまざまな修正がなされた
	PCI idsメール・アドレスのアップデートを行った
	TCP hashの最適化が行われた
	alphaのアップデート(atomic_dec_and_lock etc)が行われた
	Linux Kernel 2.4.15 リリース
	quota SMP races with BKLが修正された
	SCHED_FIFO for UP/SMPが修正された
	Ext3/minix/sysv fsync behaviourが修正された
	UFS filesystem byteorderのcleanupが行われた
	なお，このバージョンからカーネル2.5が枝分かれした
2001年12月	Linux Kernel 2.4.16
	iput()のinodeの問題を解決した
	pagecache readaheadサイズを調整可能にした
	PPC版のカーネルの問題を解決した
	Linux Kernel 2.4.17 リリース
	USBの問題を解決した
	devfsを修正した
	ネットワーク関連を修正した
	Linux Kernel 2.5.1 リリース
2002年1月	Linux Kernel 2.5.2 リリース
	周辺機器接続規格のUSB 2.0が，最新版のLinuxカーネル2.5.2でネイティブ・サポートされた
	Linux Kernel 2.5.3 リリース
2002年2月	Linux Kernel 2.4.18 リリース
	PPC MPC8260版について修正された
	ネットワーク関連を修正した

Information ―日新システムズら，SIP/V2IP製品開発向けのLinux開発環境提供と製品開発サービスを提供へ
(株)日新システムズ，フリースケール・セミコンダクタ・ジャパン(株)，(株)ソフトフロント，(株)ソフィアシステムズの4社は，各社のサービスおよび製品を組み合わせ，SIP/V2IP製品開発のためのLinux開発環境提供と，最終的な製品開発サービスを開始した．

表1 Linux Kernelの歴史（つづき）

2002年3月	Linux Kernel 2.5.6 リリース
	Linux Kernel 2.5.7 リリース
	Linux Kernel 2.5.8 リリース
2002年4月	Linux Kernel 2.5.9 リリース
	Linux Kernel 2.5.10 リリース
	Linux Kernel 2.5.11 リリース
2002年5月	Linux Kernel 2.5.12～19 リリース
2002年6月	Linux Kernel 2.5.20～24 リリース
2002年7月	Linux Kernel 2.5.25～29 リリース
2002年8月	Linux Kernel 2.4.19 リリース
	PPPの問題がフィックスした
	Linux Kernel 2.5.30～32 リリース
2002年9月	米Red Hat社がLinuxカーネル2.4.18対応の「Red Hat 8.0」をリリース
	Linux Kernel 2.5.33～39 リリース
2002年10月	Linux Kernel 2.5.40～44 リリース
2002年11月	Linux Kernel 2.4.20 リリース
	数々の修正がフィックスされた
	Linux Kernel 2.5.45～50 リリース
2002年12月	Linux Kernel 2.5.51～53 リリース
2003年1月	Linux Kernel 2.5.54～59 リリース
2003年2月	Linux Kernel 2.5.60～63 リリース
2003年3月	Linux Kernel 2.5.63～66 リリース
2003年4月	Linux Kernel 2.5.67～68 リリース
	米Red Hat社，「Red Hat Linux 9」をリリース
2003年5月	Linux Kernel 2.5.69～70 リリース
2003年6月	Linux Kernel 2.4.21 リリース
	AMDの64ビットプロセッサOpteronへの対応を追加した
	さまざまな新デバイスのサポートが追加された
	64ビット・ファイル・システムXFSに対応した
	Linux Kernel 2.5.71～73 リリース
2003年7月	Linux Kernel 2.5.74～75 リリース
	2.5系カーネルは2.5.75が最後になった
	Linux Kernel2.6テスト版がリリースされた
2003年8月	Linux Kernel 2.4.22 リリース
	ACPIのバグが修正された
	HFS+（Mac OSファイル・システム）がサポートされた
	64ビットSuperHがサポートされた
2003年11月	Linux Kernel 2.4.23 リリース
	IDEディスクのホット・プラグがサポートされた
	Linux IP Virtual Serverがサポートされた
	root権限を不正取得できてしまう脆弱性に対応した
2003年12月	**Linux Kernel 2.6 リリース**
2004年1月	Linux Kernel 2.6.1 リリース
	Radeon IGP345Mチップセットに対応した
	SGI IOC4チップセットに対応した
	BT8xxTVチューナ・カードに対応した
	Tekram DC390（T），Am53，79C974で使用するSCSIドライバが修正された
2004年1月	Linux Kernel 2.4.24 リリース
	仮想メモリが使用する機能にある脆弱性に対応した
2004年2月	Linux Kernel 2.6.2 リリース
	64ビットCPUについてのバグ・フィックスがなされた
	USBドライバの修正がなされた
	Bluetoothドライバの修正がなされた
	64ビット・ファイル・システムXFSが改良された
2004年2月	Linux Kernel 2.6.3 リリース
	数多くのALSAドライバが修正された
	mremap（2）システム・コールに関する脆弱性に対応した
	ACPI関連のコードが更新された
	PowerPC G5に対応した
2004年2月	Linux Kernel 2.4.25 リリース
	mremap（2）システム・コールに関する脆弱性に対応した
	XFS，USB関連のドライバが更新された
2004年3月	Linux Kernel 2.6.4 リリース
	HFSおよびHFS+のドライバが更新された
	XFSドライバが更新された
	IEEE1394，USB，PCI関連の各種ドライバが更新された
2004年4月	Linux Kernel 2.6.5 リリース
	無線LANカードのprism54ドライバが更新された
	USBドライバが更新された
	ALSAドライバが修正された
	PCMCIA，ACPI，AGP，I2Cのドライバが修正された
	x86-64，PowerPC64，IA64，s390に関する修正がなされた
2004年4月	Linux Kernel 2.4.26 リリース
	AMD64，IA64，PowerPCに関する修正がなされた
	XFS，JFS関連のドライバが更新された
2004年5月	Linux Kernel 2.6.6 リリース
	Video For Linuxドライバが修正された
	RTL8169ドライバが修正された
	SiS 964/180シリアルATAがサポートされた
	CFQ I/Oスケジューラがサポートされた
2004年6月	Linux Kernel 2.6.7 リリース
	浮動小数点の例外に関する脆弱性を修正した
2004年8月	Linux Kernel 2.6.8 リリース
	NFS関連のバグ修正がなされた
	各種ドライバが更新された
2004年8月	Linux Kernel 2.4.27 リリース
	Bluetooth関連が更新された
	各種ドライバが更新された
2004年10月	Linux Kernel 2.6.9 リリース
	SMC社製EtherPower II 10/100（SMC9432）用ネットワーク・ドライバepic100を更新した
	ALSAドライバを更新した
	ACPI関連を更新した
	組み込み用32ビットRISCチップM32R関連を更新した
2004年11月	Linux Kernel 2.4.28 リリース
	SMC社製EtherPower II 10/100（SMC9432）用ネットワーク・ドライバepic100を更新した
	ACPI，暗号化関連を更新した
	無線LANカード・ドライバprism54が更新された
	XFS，JFS関連のドライバが更新された
2004年12月	Linux Kernel 2.6.10 リリース
	Wavelan ISAドライバが更新された
	ethtoolコマンドが修正された
	USBドライバが更新された
	無線LANカード・ドライバprism54が更新された
2005年1月	Linux Kernel 2.4.29 リリース
	NETFILTERドライバを更新
	XFS関連のドライバが更新された
	無線LANカード・ドライバprism54が更新された
	SATA関連のドライバが更新された
2005年3月	Linux Kernel 2.6.11 リリース
	既知のバグすべてに対応した
	x86-64，IA64，PPC，ARM，MIPS関連の更新
	ACPI関連を更新した
	DRI，ALSA，SCSI，XFS関連を更新した
	相互接続アーキテクチャInfiniBandをサポートした
2005年4月	Linux Kernel 2.4.30 リリース
	IA64関連が更新された
	NETFILTERドライバを更新した
	USBドライバが更新された
	SMC社製EtherPower II 10/100（SMC9432）用ネットワーク・ドライバepic100を更新した
	JFS関連のドライバが更新された
	SATA関連のドライバが更新された

New Products ――エルピーダメモリ，データ転送速度が6.4Gビット/sと高速なDRAM「XDR DRAM」を発売
エルピーダメモリ（株）は，データ転送速度が6.4Gビット/sと高速なDRAM「XDR DRAM EDX5116ABSE」のサンプル出荷を開始した．容量は512Mビット（4M×16ビット×8バンク構成），電源電圧は1.8V．9月から量産出荷を開始する予定．