Linuxによる組込みシステム構築手法

MONTAVISTA™
SOFTWARE

Linux
CONFERENCE

2002年9月18日

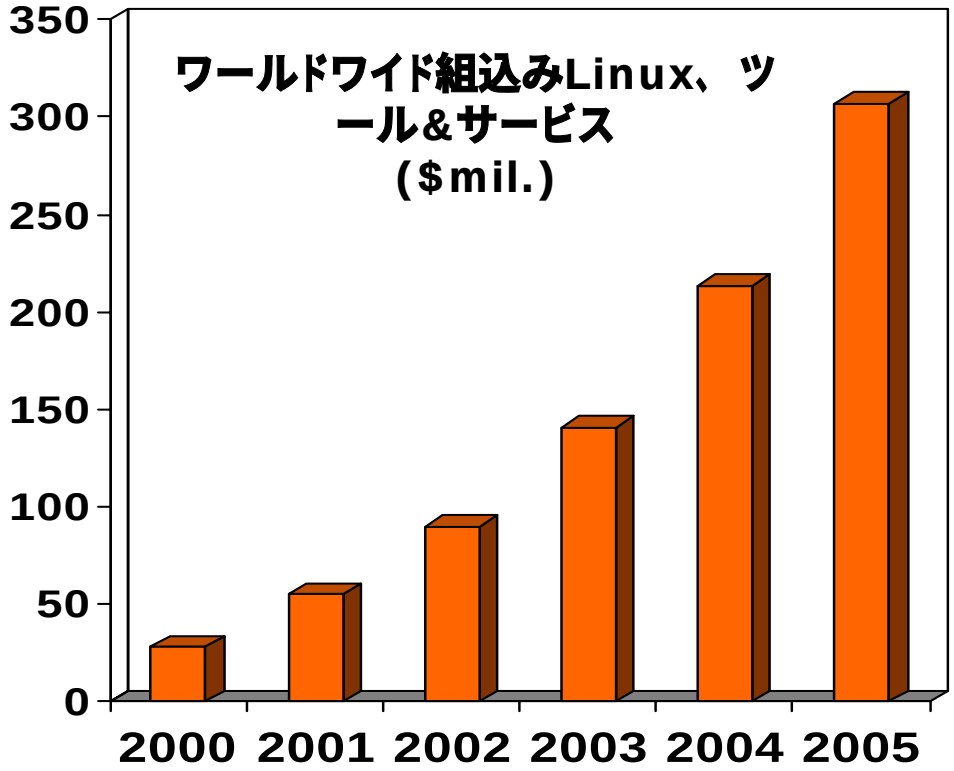
モンタビスタソフトウエア ジャパン(株)
木内 志朗

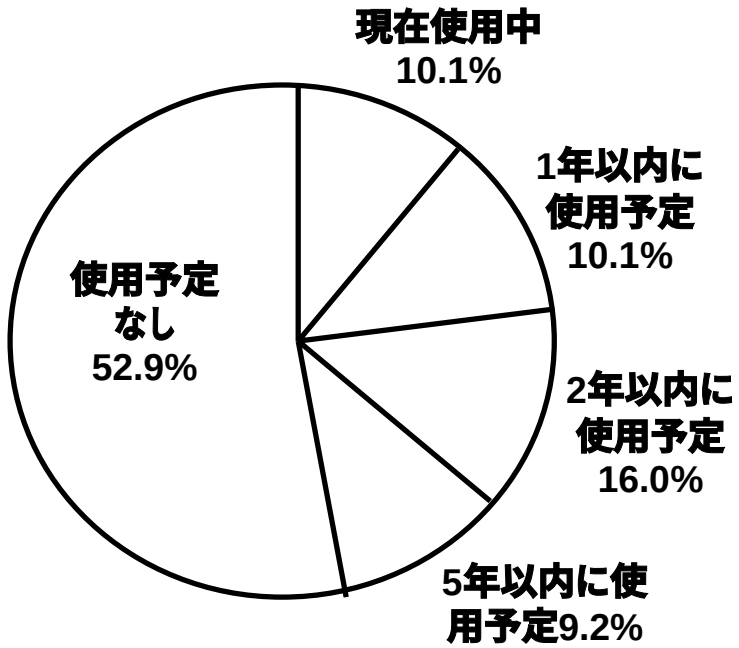
Linux搭載製品...


MONTAVISTA
SOFTWARE
FLORA-ie
55mi
Pre - Order
TS2000
Introducing
the all NEW
VR3
it...

Linux市場成長

MONTAVISTA SOFTWARE





組込み開発者の36%が2年以内にLinuxを使用予定

Data from VDC, Linux's Future in the Embedded Systems Market, May 2001

アプリケーション別市場予測

アプリケーション分野別　ワールドワイド組込みLinuxの出荷予測

（ツール＆サービスを除く）　　（$Mil.）

分野	2000	2005	CAGR
Telecom/Datacom	13.2	126.7	57.3%
Consumer Electronics	8.9	121.1	68.5%
Industrial Automation	1.4	14.2	60.1%
Retail Automation	1.0	8.7	55.5%
Office Automation	0.9	6.6	49.7%
Military/Aerospace	0.9	9.4	58.5%
Automotive	0.7	7.4	62.1%
Information Automation	0.6	6.6	60.7%
Medical	0.4	4.0	56.5%

Data from VDC, Linux's Future in the Embedded Systems Market, May 2001

Notes:

- Consumer Electronics will have high penetration but typically has higher cancellation of projects. Top sub-markets: set-top boxes, home networking gateways, Internet access appliances, PDAs, digital TV sets, smart phones, & MP3s.
- Automotive includes in-car computing, navigation systems, & audio/visual/multimedia as key growth sub-segments.
- Information Automation are storage & sharing of data such as Server Appliances (file servers, email servers and firewalls).
- Industrial Automation is the classic embedded area that is still hardware focused & has real-time concerns but in recent years is moving toward innovation.
- Telecom/Datacom is the highest penetration today and will continue to see growth for comms infrastructure equipment.

日本における組込みLinux

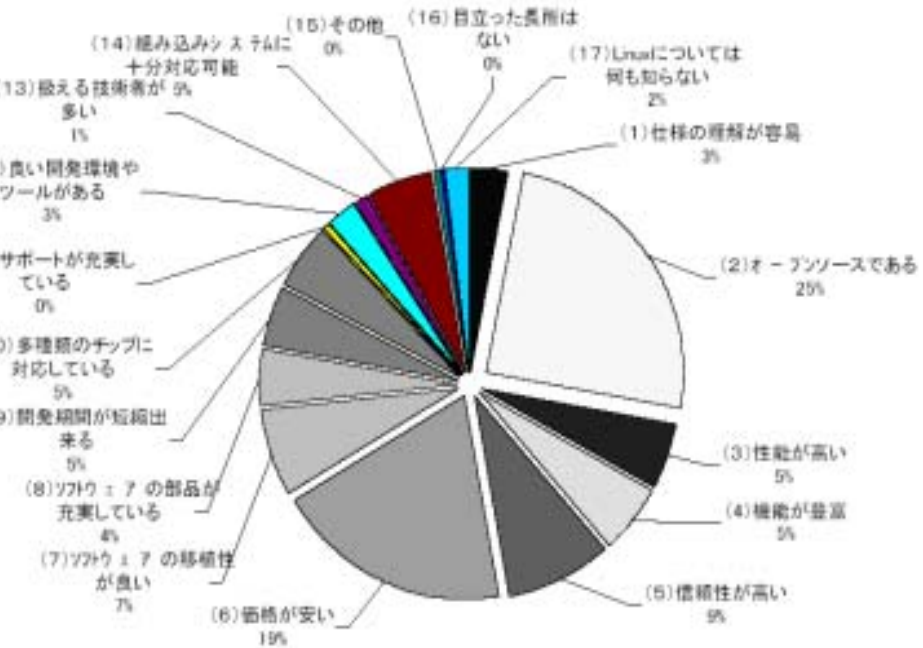
- 日本でも2000年から急速に組込みLinuxの導入及び検討が始まっている。
- 2002年後半から、多数の組込みLinux搭載製品が市場に・・・・

- なぜ組込みでLinuxを使用するのか？
  - ソースが入手可能
  - 多くの情報が入手できる
  - ネットワークが豊富
  - ロイヤリティフリー
  - iTRONや商用RTOSからの置き換え

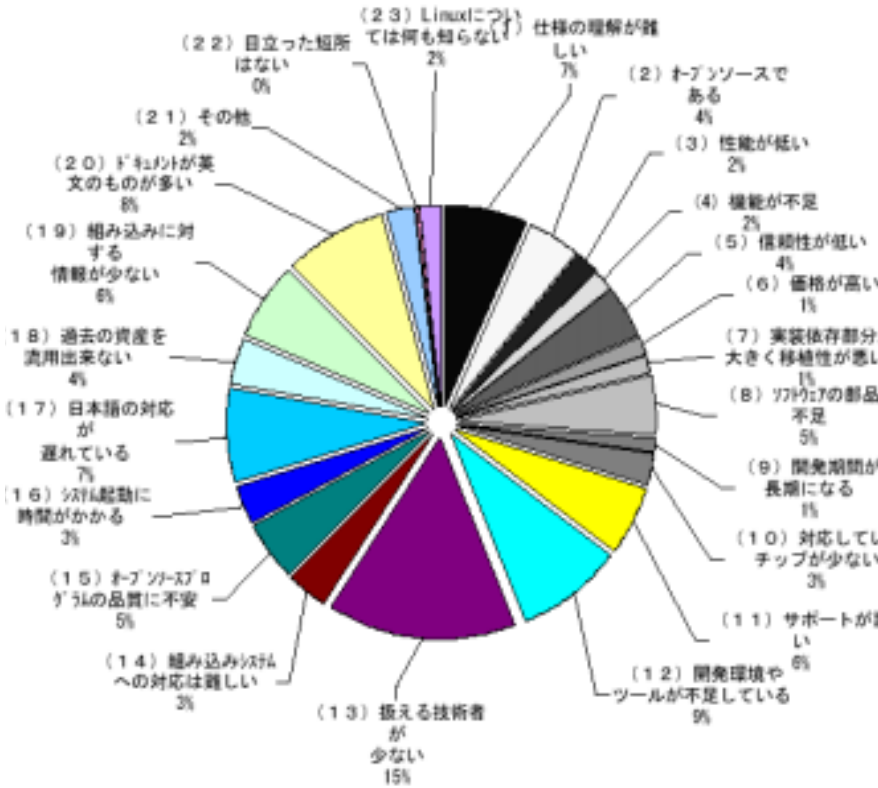
組込みLinuxの長所/短所

MONTAVISTA SOFTWARE

- Linuxの長所



- Linuxの短所



(財)日本システムハウス協会

「Embedded Linuxにおける技術動向（平成13年動向調査報告書」

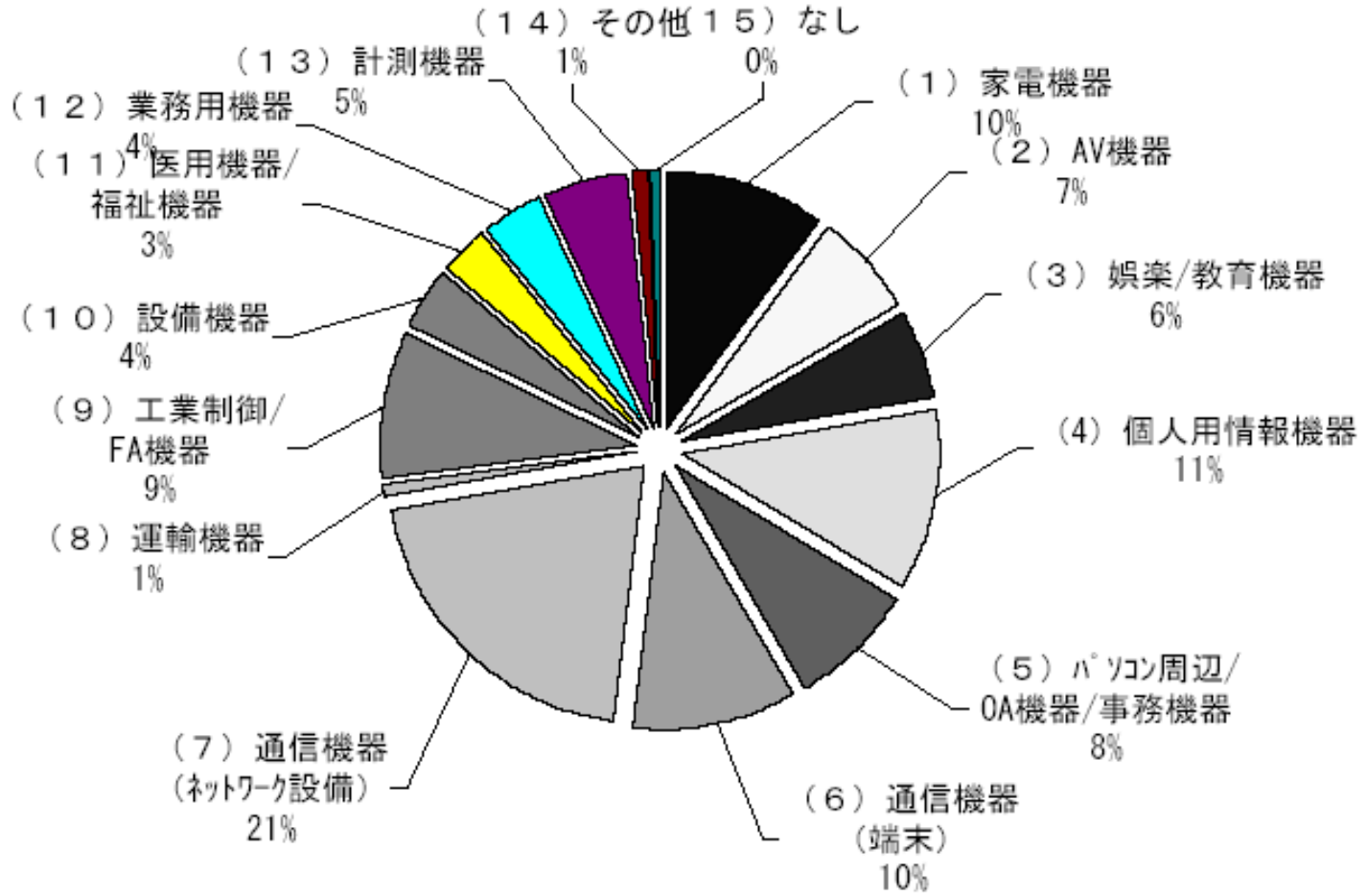
Linuxと日本市場

世界をリードする様々なコンスーマ市場

- **ホームサーバ**
- STB／PVR
- **インターネットアプライアンス**
- **デジタル**TV
- **プリンタ**／LBP
- **ゲーム機**
- PDA
- **など**

組込みLinux**の普及が最も期待される市場！**

今後Linuxが普及する分野



(財)日本システムハウス協会

「Embedded Linuxにおける技術動向（平成１３年動向調査報告書」

Linuxの実装

開発の流れ

組込みLinuxベンダから提供可能

(1)組込みLinux情報収集
(2)開発環境の整備
- 開発ツール(GNU binutilsなど)
- カーネルソース入手(http://www.kernel.orgなど)
- ブートコード開発

(3)カーネルの移植
(4)デバイスドライバの開発
(5)ユーザーランドの整備
(6)アプリケーション開発
(7)ROM/Flash 化
(8)テスト、評価

ブートローダー

- Linux自身はブートする機構をもっていない。
- PCや市販ボードではファームを利用することが可能
  - BIOS、LILO、PPC-BUG、PMON など
- オープンソースのブートローダも幾つか存在
  - PPCBOOT (PowerPC http://ppcboot.sourceforge.net)
  - RMON (MIPS http://www.carmel.com/pmon)
  - Bootldr (AtrongARM http://www.handhelds.org/)
  - GNU GRUB (IA-32)
- 最低限のブート／ローダーに必要な機能

  (1)ボード、CPUの初期設定

  (2)メモリー初期化

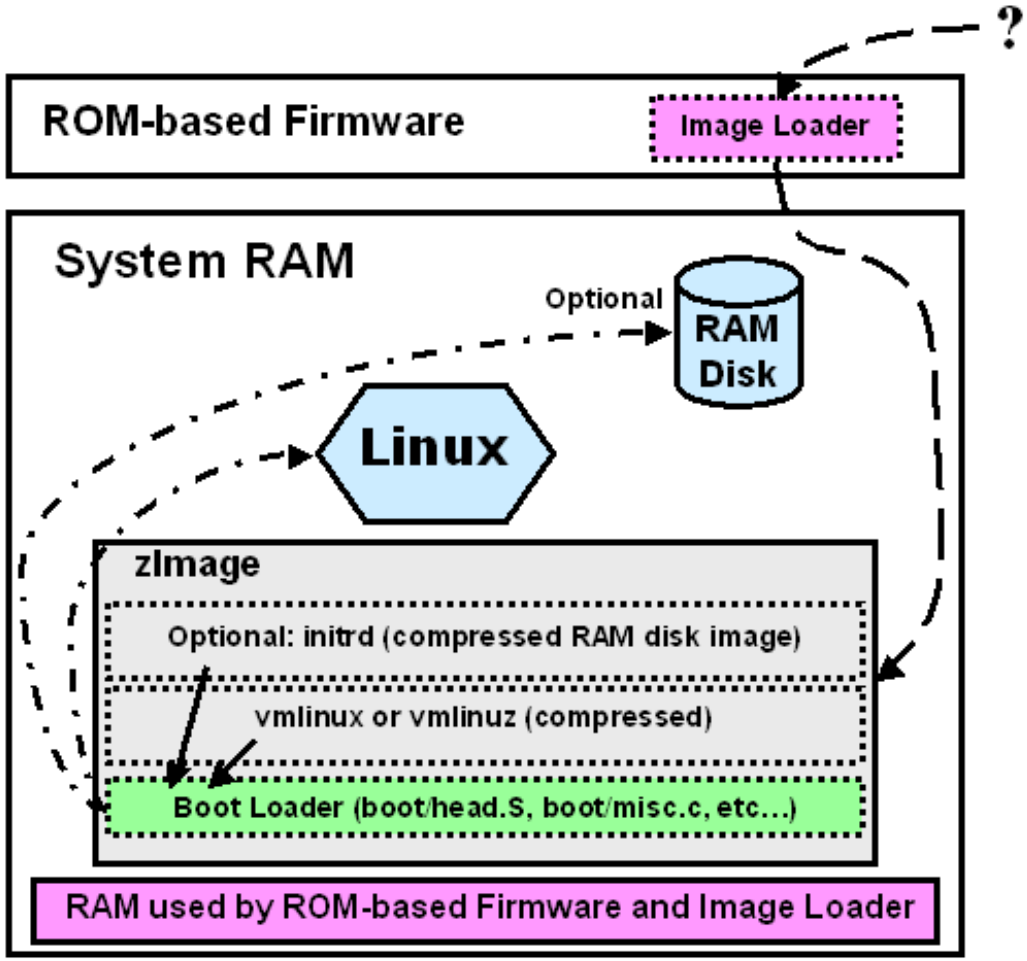
  (3)カーネルイメージをメモリー上に展開

  (4)カーネルにジャンプ

ブートの概要

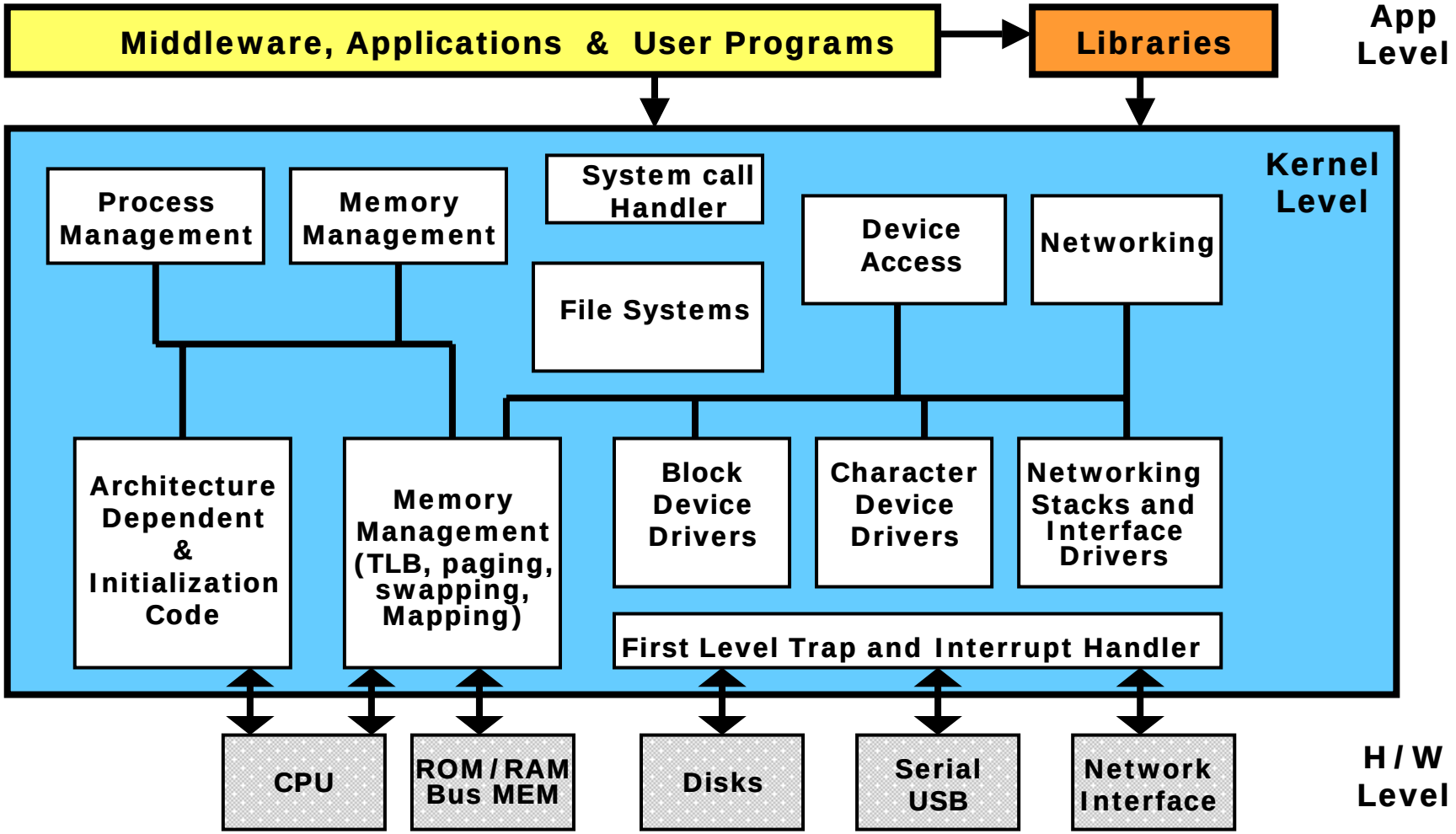
- 電源ON あるいは リセット
  - CPUイニシャル状態
- ROMやFlashなどから起動コードの実行
  - CPU、ボード依存コード
  - 必要なデバイスの初期化
    - 最低限、DRAMなどのメモリアクセス
    - マシン / ボードによっては、周辺デバイスの初期化まで必要な場合もある
- イメージのロード
  - プロンプトあるいは、固定的なロードメディアの選択（？）
    - ネットワーク（BOOTP、TFTP）
    - ROM、Flash、CF
    - HDD
    - Download
  - カーネルイメージ（zImage）にジャンプ
- ディスクからのブート時はLILOを使用

カーネル起動までの動作



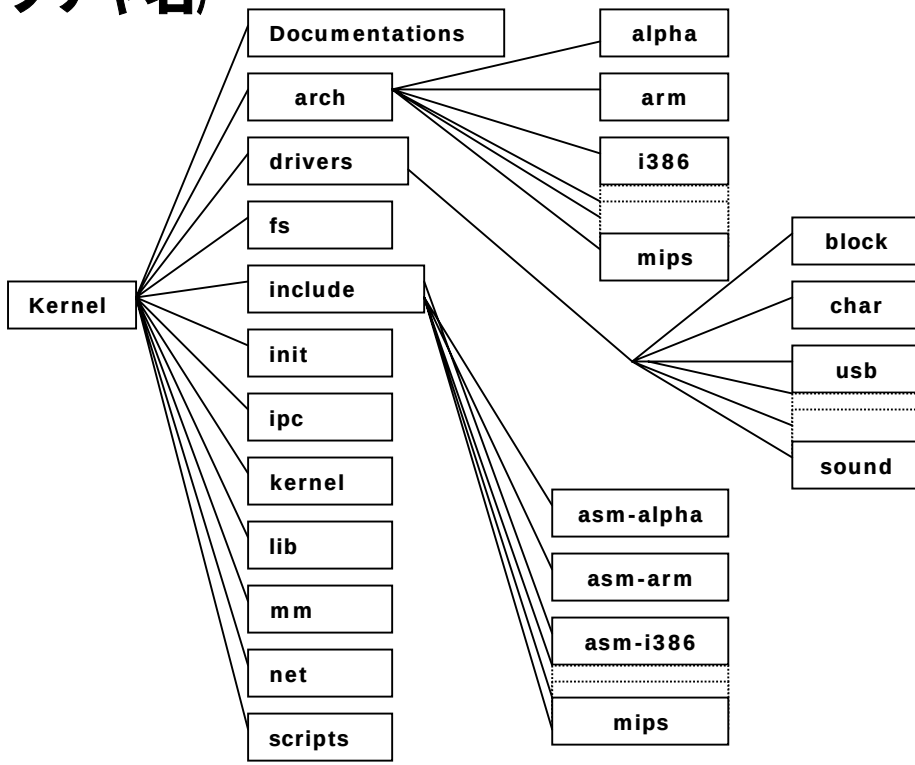
- Image Loader
  - zImageをRAMにコピー
  - ブートローダにジャンプ
- Boot Loader
  - RAMディスクが有れば展開
  - vmlinuxを展開
  - vmlinuxにジャンプ
- vmlinux
  - ディスクマウント
  - Init実行

Linux カーネル アーキテクチャ



ターゲットボードへのカーネルの移植

- 基本的にはC言語またはアセンブラ
- アーキテクチャ依存部
  - arch/ (アーキテクチャ名)
  - Include/asm-(アーキテクチャ名)

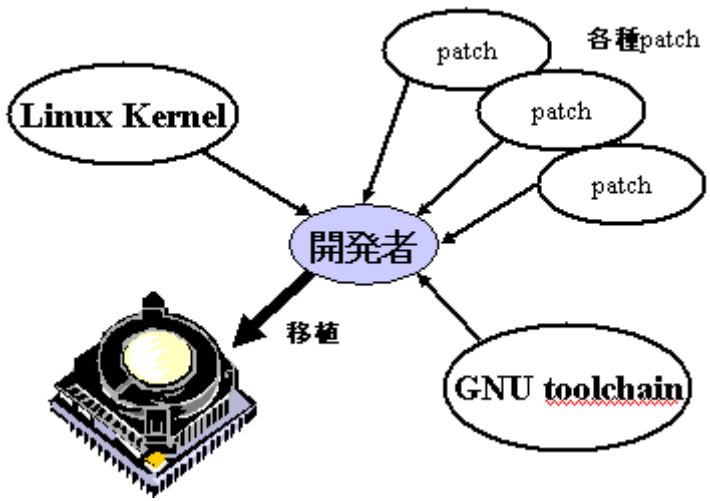


カーネルソースコード (Ver2.x 概算)

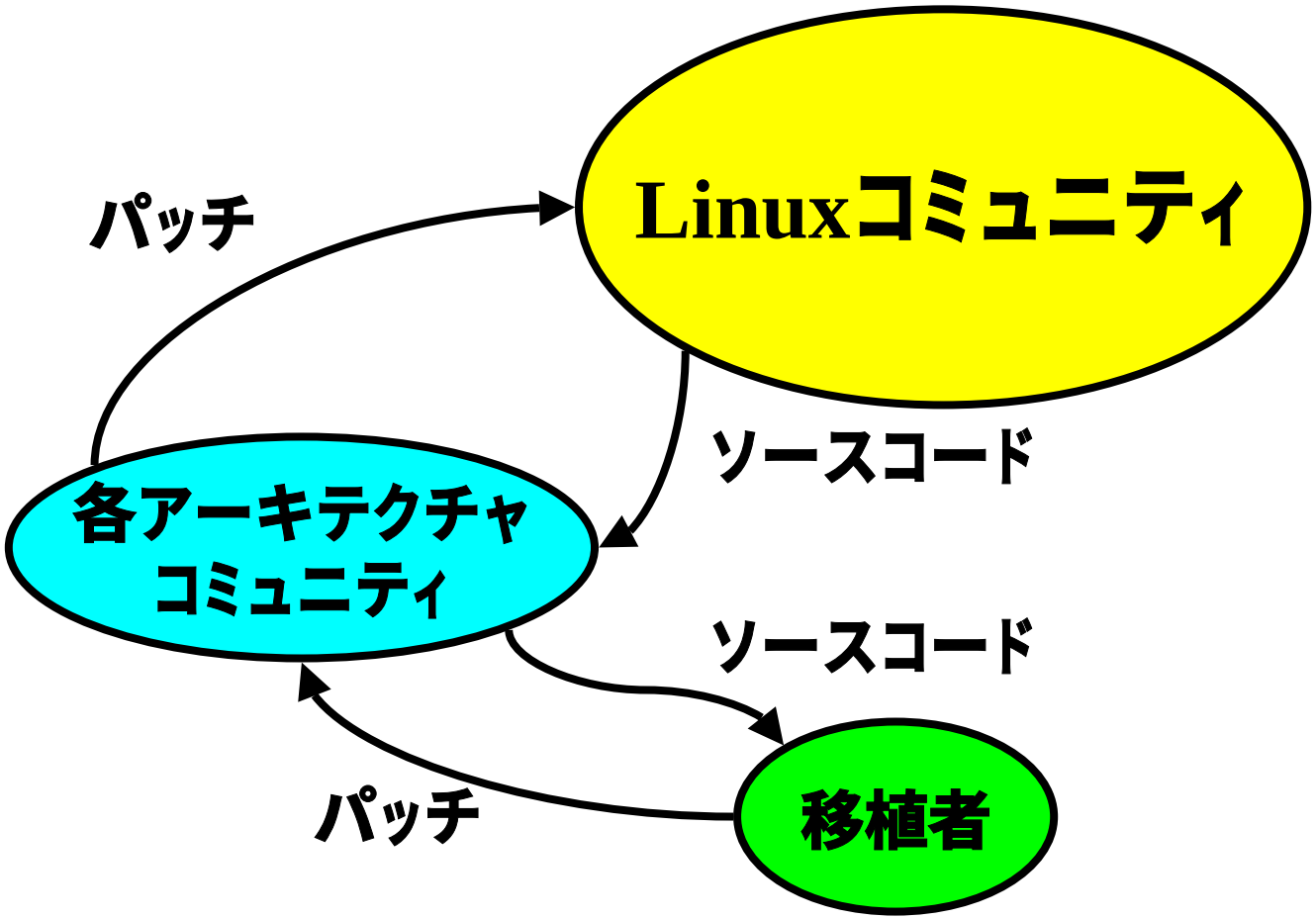
	Cコード	アセンブラ
- デバイスドライバ	400,000	200
- ネットワーク	28,000	
- VFS レイヤ	14,000	
- ファイルシステム	55,000	
- イニシャル	4,000	3,000
- 数値演算コプロ		4,000
- システム管理、サービス	22,000	
- インクルードファイル	??	

カーネルの移植

- アーキテクチャの最新ソースコードの入手
- 開発ツールの入手
- 各種patchの入手
- 移植
- Linuxカーネルへのフィードバック



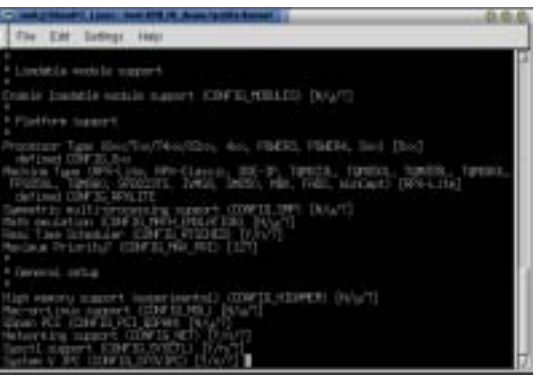
Linuxカーネルのフィードバック



カーネルコンフィグレーション

- カーネルを新しいカーネルのコンフィグレーション
  - make config　　テキストベースの対話式
  - make menuconfig　　簡易グラフィック表示のコンフィグレーション
  - make xconfig　　X-Win上のコンフィグレーション

make xconfig



make menuconfig

make xconfig

カーネルコンフィグレーション

カーネルビルド手順

ーネルコンフィグレーション項目例

Code maturity level options
Loadable module support
Palatform support
General setup
Memory Technology Devices (MTD)
Plug and Play configuration
Block devices
Multi-device support (RAID and LVM)
Networking options
ATA/IDE/MFM/RLL support
SCSI support
Network device support
Amateur Radio support
IrDA (infrared) support
ISDN subsystem
Old CD-ROM drivers (not SCSI, not IDE)
Console drivers
Input core support
Character devices
Multimedia devices
File systems
Sound
USB support
Kernel hacking

- **カーネルコンパイルの手順**

(1) カーネルコンフィグレーション

make menuconfig

(2)ソースの依存関係の計算

make dep

(3)カーネルのコンパイル

make vmlinux

make vmlinux
make zImage
make zImage.initrd

カーネルコンフィグレーション

ドライバの追加

- 例えばキャラクタデバイスドライバ mydriver.cを追加
  - (1)drivers/char/mydriver.c を作成
  - (2)drivers/char/Config.in に以下を追加

```
tristate 'Support MyDriver'  CONFIG_MYDRIVER
```

  - (3)drivers/char/Makefile に以下を追加

```
ifeq ($(CONFIG_MYDRIVER),y)
   L_OBJS += mydriver.o
else
     ifeq ($CONFIG_SAMPLE),m)
     M_OBJS += mydriver.o
     endif
  endif
```

  - (4)カーネルコンフィグレーション（xconfig, menuconfigなど）でmydriver.oを選択が可能になる

    Charcter devices -> Support MyDriver

- カーネルコンフィグレーションの結果が、.config ファイルに反映される

デバイスドライバ

- 豊富なデバイスドライバーをサポート
  - ethernet、serial、timer、USB、IEEE1394、PCMCIA、Flash Mem、PCI ・・・・
- スタティックあるいは、ダイナミックロード可能
- 既存のデバイスドライバを流用する場合の問題
  - アーキテクチャに依存
  - カーネルバージョンへの依存
  - エンディアン
  - 割り込み関連処理
  - 割り込み応答性
  - ドライバの品質
  - BIOSに依存

デバイスドライバのソース

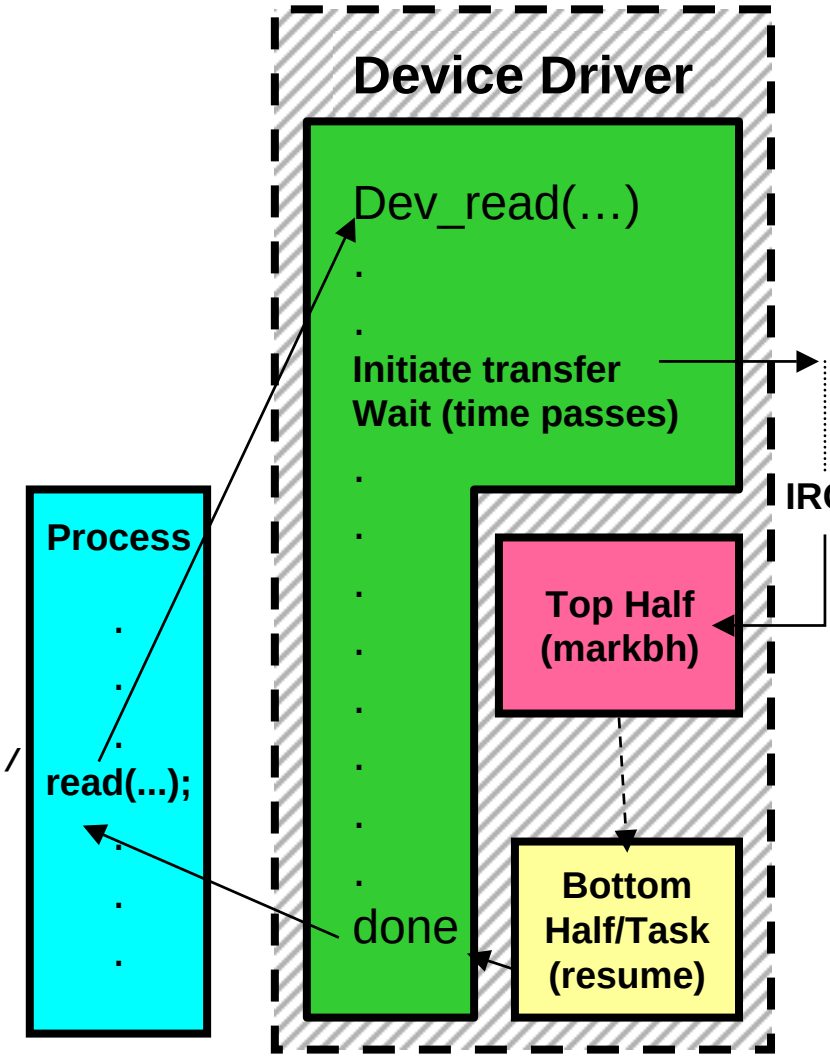
MONTAVISTA SOFTWARE

```
$ cd /opt/hardhat/devkit/ppc/8xx/kernel/linux-*/drivers
$ ls -l
total 96
-rw-r--r--   1 root     root         2282 Aug  9  1999 Makefile
drwxr-xr-x   6 root     root         4096 Jul 27 08:08 acorn/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 ap1000/
drwxr-xr-x   3 root     root         4096 Jul 27 08:08 block/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 cdrom/
drwxr-xr-x   6 root     root         4096 Jul 27 08:08 char/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 dio/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 fc4/
drwxr-xr-x  11 root     root         4096 Jul 27 08:08 isdn/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 macintosh/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 misc/
drwxr-xr-x   5 root     root         4096 Jul 27 08:08 net/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 nubus/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 pci/
drwxr-xr-x   2 root     root         4096 Jul 27 08:08 pnp/
drwxr-xr-x   4 root     root         4096 Jul 27 08:08 sbus/
drwxr-xr-x   3 root     root         8192 Jul 27 08:09 scsi/
drwxr-xr-x   3 root     root         4096 Jul 27 08:09 sgi/
drwxr-xr-x   3 root     root         4096 Jul 27 08:09 sound/
drwxr-xr-x   2 root     root         4096 Jul 27 08:09 tc/
drwxr-xr-x   3 root     root         4096 Jul 27 08:09 usb/
drwxr-xr-x   2 root     root         4096 Jul 27 08:09 video/
drwxr-xr-x   2 root     root         4096 Jul 27 08:09 zorro/
$ find . -type f -print | wc -l
   1451
$
```

デバイスドライバの構造

- ドライバ機能

```
#include <linux/fs.h>
static struct file_operations device_fops = {
        NULL,                   /* llseek */
        device_read,            /* read */
        device_write,           /* write */
        NULL,                   /* readdir */
        device_poll,            /* poll */
        device_ioctl,           /* ioctl */
        NULL,                   /* mmap */
        device_open,            /* open */
        NULL,                   /* flush */
        device_release,         /* release */
        NULL,                   /* fsync */
        NULL,                   /* fasync */
        NULL,                   /* check_media_change */
        NULL,                   /* revalidate */
        NULL,                   /* lock */
};
```



デバイスドライバ
モジュールのイニシャル / 終了

- スタティックにリンクされている場合、カーネル初期時に呼ばれる。
- ダイナミックにリンクする場合、insmod / rmmodにより呼ばれる。

```
/*************************************************************************
 *   <ドライバ初期設定>
 *   この関数は、モジュール登録時 (insmod) によばれます。ドライバをスタティックリンクしてある
 *   場合は、カーネルの初期化時に呼ばれます。
 *   処理としては、カーネルにデバイスドライバを登録し、必要な変数や構造体などを初期化します。
 */
static int __init my_init_module(void)
{
     /* カーネルにデバイスドライバを登録します。その際にドライバ名とデバイスのメジャー番号
        を指定します。
     */
     res = register_chrdev(MYDRIVER_MAJOR, MYDRIVER_NAME, &my_fops);
     if (res) {
             printk("init_mydriver: register_chrdev() failed, rc==%d¥n", res);
             return -EIO;
     }
     return 0;
}

/*************************************************************************
 *   <ドライバ終了>
 *   この関数は、モジュール終了時に呼ばれます。 通常、登録されたドライバを削除します。
 */
static void __exit my_cleanup_module(void)
{
     /* カーネルからデバイスを削除します */
     unregister_chrdev(MYDRIVER_MAJOR, MYDRIVER_NAME);
     return;
}

module_init(my_init_module);                    /* モジュール／ドライバ登録 */
module_exit(my_cleanup_module);                 /* モジュール／ドライバ終了 */
```

デバイスドライバ
デバイス 読み込み<open()/close()>

- Open()/close()システムコールにより呼ばれる。
- 割り込みの設定/リリース、ワークエリアの確保/リリース、タイマー登録/リリースなど

```
/*********************************************************************
 * <オープン処理>
 * この関数は、アプリケーションからopen()が呼ばれたときにコールされます。
 */
static int mydriver_open(struct inode *inode, struct file *file)
{
     int rc;
     /* デバイスの初期化やワークエリアの確保、初期化
        タイマなどの設定などを書きます。*/

     /* デバイスの割込みハンドラを登録します。 */
     rc = request_irq(IRQXX, mydriver_interrupt, SA_SHIRQ, MYDRIVE_NAME, NULL);
     if (rc) {
          return -EBUSY;
     }

     MOD_INC_USE_COUNT;
     return 0;
}

/*********************************************************************
 * <リリース処理>
 * この関数は、アプリケーションからclose()が呼ばれたときにコールされます。
 */
static int mydriver_release(struct inode *inode, struct file *file)
{
     /* ワークエリアの開放、タイマーの削除などの処理を
        書きます。*/
     MOD_DEC_USE_COUNT;

     return 0;
}
```

デバイスドライバ
デバイス 読み込み<read()>

- read()システムコールにより呼ばれる。
- 通常、デバイスからの読み込み処理を実装する。
- ノンブロック / ブロックの実装
- カーネルスペースからユーザースペースのデータコピー (copy_to_user)

```
/*****************************************************************
 * <読み込み処理>
 * この関数は、アプリケーションからread()が呼ばれたときにコールされます。
 */
static ssize_t mydriver_read(struct file *file, char *buf, size_t count, loff_t *offset)
{
     unsigned char data[10];

     /* デバイスからデータを読み込みます。ここでは、実際にデータをよみこむのは、デバイスドライバと仮定して、スリープさせます。 */
     interruptible_sleep_on(&mydriver_zz);

     /* ペンディングされているシグナルを確認します。 */
     if (singnal_pending(current))
             return -EINTR;

     /* ボトムハーフ処理で発行された wake_up_interruptible() によって、起床し以下のコードが実行されます。 */

     /* dataはカーネル空間ですので、カーネル空間からユーザ空間のbufにデータをコピーします。 */
     if (copy_to_user(buf, &data, sizeof(data)))
                     return -EFAULT;
             return mydriver_final_count;
     } else {
             return -1;
     }
}
```

デバイスドライバ
デバイス 書き込み<write()>

- write()システムコールにより呼ばれる。
- 通常、デバイスの書き込み処理を実装する。
- ノンブロック / ブロックの実装
- カーネルスペースからユーザースペースのデータコピー (copy_from_user)

```
/*****************************************************************
 * <書き込み処理>
 * この関数は、アプリケーションからwrite()が呼ばれたときにコールされます。
 */
static ssize_t mydriver_write(struct file *file, const char *buf, size_t count, loff_t *offset)
{
     char data[10];

     /* デバイスに書き込むデータ buf は、ユーザ空間のデータですので、カーネル空間の data にコピーします */
     if (copy_from_user(&data, buf, sizeof(data)))
             return -EFAULT;

     /* ここで、data の値をデバイスに書き込みます */
    XXX_Device_Write();

     return sizeof(data);
}
```

デバイスドライバ
割り込み、タスクレット（ボトムハーフ）

- 割込み処理で長い間ブロックしたくない場合、、割込みの後処理（ボトムハーフ）を使用する
- 遅延処理としてタスクレット（ボトムハーフ）を使用

```
/*********************************************************************
 * ＜割込みハンドラ（トップハーフ）＞
 */
void mydriver_interrupt(int irq, void *dev_id, struct pt_regs *fp)
{
     /* デバイスからデータを取り込みます */

     /* 割込みの後処理としてタスクレットの実行フラグをセットします。 */
     tasklet_init(&mydriver_tasklet, mydriver_task, (unsigned long)dev_id);
     tasklet_schedule(&mydriver_tasklet);
     return;
}


/*********************************************************************
 * ＜割込み後処理（タスクレット）＞
 */
void mydriver_task(unsigned long data)
{
     /* 割込みハンドラが取り込んだデータを加工したりします。 */

     /* スリープしている プロセスを起床させます。 */
     wake_up_interruptible(&mydriver_zz);
     return;
}
```

カーネル／デバイスドライバのデバッグ

MONTAVISTA SOFTWARE

- カーネルの機能を利用
  - printk
  - oopsメッセージ
  - /proc
  - kgdb
- ICE、J-TAGなどの利用
  - Linuxへ対応している環境がそろいだしてきている
    - MMUのサポート
    - ドライバのダイナミックロード時のアドレス解決
  - Emblix 開発環境ワーキンググループで標準化

printkの使用

- 重要度による分類が可能
  - KERN_EMERG, KERN_ALERT, KERN_CRIT, KERN_ERR, KERN_WARNING, KERN_NOTICE, KERN_INFO, KERN_DEBUG の8種類が定義
- 出力レベルの制御が可能

  ```
  # echo 8 > /proc/sys/kernel/printk
  ```
- 利用が簡単
- あらかじめ記述が必要

```
#define MYDEBUG(n, args...)     ¥
        do { if (my_debug > (n)) printk(KERN_DEBUG args); } while (0)

int __init my_init(void)
{       MYDEBUG(1, "loading my module.¥n"); return 0;}

void __exit my_exit(void)
{       MYDEBUG(2, "cleanup my module.¥n");}

static int __init my_setup(char *str)
{       my_debug = simple_strtol(str, NULL, 0); return 1;}
```

Oopsメッセージ

- レジスタおよびstackの情報を表示
- アーキテクチャにより出力が異なる
- 出力の解析が難しい

(一部のアーキテクチャでは解析ツールがある)

```
  options:  [pci] [cardbus]
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
Scheduling in interrupt
kernel BUG at sched.c:666!
Unable to handle kernel paging request at virtual address 00000000, epc == 8001ea78, ra == 8001ea78
Oops in fault.c:do_page_fault, line 204:
$0 : 00000000 1000fc00 0000001b 1000fc00 801ff890 00000001 00000001 00000fa8
$8 : 00000000 00000000 81177c86 00000000 00000000 00000000 8021e8a9 fffffff9
$16: 00000000 00000000 81176000 801f4000 80200b48 8000a000 801fcf88 f7f48a6d
$24: 81177c3a ffffffff                   81176000 81177d78 81177d78 8001ea78
Hi : 00000003
Lo : 33333336
epc  : 8001ea78    Not tainted
Status: 1000fc02
Cause : 0000830c
Process keventd (pid: 2, stackpage=81176000)
Stack: 801a5d30 801a5e54 0000029a 00000f8a 801ffa64 00000000 00000000 801f4000
       80200b48 80200b54 00000001 f7f48a6d 81177db0 80020100 801f44e0 00000026
       00000ef8 801f4000 81177d78 80013b5c 00000000 801f4000 80200b48 80200b54
       801ffa64 80013bd0 801f4020 00000000 00000ef8 801f4000 801f4000 801a1b28
       8002f21c 76757e73 7a79f877 fffffffd5 81162000 81177e60 00000000 1000fe01
       80200000 ...
Call Trace: [<801a5d30>] [<801a5e54>] [<80020100>] [<80013b5c>] [<80013bd0>] [<801a1b28>]
 [<8002f21c>] [<80012e18>] [<80012e0c>] [<80122efd>] [<c01afe48>] [<8007d81c>]
 [<8007faac>] [<8010ed94>] [<8010ec54>] [<80012aa8>] [<8010ea7c>] [<8010ef34>]
 [<80035c40>] [<8000f088>] [<8002ade0>] [<80036238>] [<80036220>] [<80036034>]
 [<8000ca8c>] [<80036418>] [<8000ca8c>]

Code: 24a55e54  0c00801d  2406029a <ac000000> 8e42001c  0440000c  00000000  8f820004  2442ffff
Kernel panic: Aiee, killing interrupt handler!
In interrupt handler - not syncing
 □
```

カーネル / ドライバのデバッグ

kgdb

- リモートデバッグ
  - ホストマシンとターゲットをシリアル接続
- ステップ動作が可能
- すべての環境で利用できるわけではない
  - アーキテクチャごとの対応が必要

ファイルシステム

- Linuxは豊富なファイルシステムをサポート
  - minix、ext2、FAT、NTFS、hpfs、ISO9660、nfs、cramfs、raiserFS、JFS、JFFS、ext3、raw ・・・・
- デスクトップ環境ではext2 / ext3が標準的
- 組込みシステムでは・・・・
  - RAM Disk
  - ROM FS
  - cramfs
  - JFFS / JFFS2
  - Raiserfs
  - XFS
  - など

ルートファイルシステム

- ターゲット上にルートファイルシステムを構築 (クロス環境)
  - ブート時、NFSマウントでコンソールよりファイルシステム構築
  - ホスト上でファイルシステムイメージ作成 －＞ Flash 書き込み
  - RAM ディスク、initrdなどの利用
- 一般的なディレクトリ構成

/bin /dev /etc /home /lib /mnt /proc /sbin /usr /var

ファイルシステム パッケージ ツール

(ターゲット コンフィグレーションツール：MontaVista)

MONTAVISTA SOFTWARE

ミドルウエア

- グラフィックス
- ブラウザ
- Java
- プロトコル

GUI

MONTAVISTA SOFTWARE

- デスクトップではXFree86 (X-Window) が標準的
- 組込みシステムではメモリサイズが問題
  - X-Window (XFree86)
    - 8〜16Mbyte
  - Tiny-X
    - X-Windowをスケーラブルに構成
    - Xサーバーは最小で1Mbyte程度
  - DirectFB
    - GTK+が、X-Windowを使用せずに直接FrameBufferをアクセス
    - GtkFBライブラリは約2Mbyte、簡単はボタンWidgetなどが約1.7Mbyte
  - Qt/Embedded
    - X-Windowを使用せずに直接FrameBufferをアクセス
    - Qt/X、Qt/Winと互換
    - 800K〜3Mbyte
  - Microwindows
    - Win32、X-WinライクなAPIをサポート
    - 200K〜1.2Mbyte

Browser

- Webブラウザが要求されるデバイス
  - セットトップボックス / インターネットTV
  - ホームサーバ
  - モバイルデバイス
  - PDA、WebPad
  - その他
- 幅広い選択肢
  - オープンソース：ViewML、Mozilla、NonoZilla・・・・
  - プロプライエタリ：Netscape、Opera、SpyBlass、NetFront・・・
- 必要機能
  - HTML / Java / Java script
  - SSL、WAP、フォント、plug-in、日本語化、FEP

Java

- フリーLinux Java
  - Blackdown
    - http://www.blackdown.org/java-linux.html
  - Kaffe project
    - http://www.kaffe.org
- 商用Linux Java
  - Visual Age Micro Edition（OTI/IBM)
    - http://www.embedded.oti.com
  - Jeode（Insignia Solutions Inc.)
    - http://www.insignia.com
  - intent（TAO Group)
    - http://www.tao-group.com

Linuxにおけるミドルウェア(1)

Item	Short Description	Available for Linux	Open Source	Comments	URL
BlueTooth	Wireless Interface	Yes	Yes	AXIS Communications	http://axis.com
Browser	Embedded HTTP client	Yes	Yes/No	ViewML Netscape/Mozilla Opera Espial NetFront	http://www.viesml.com http://www.netscape.com http://www.opera.com http://espial.com http://www.access.co.jp
xDSL	Digital Subscriber Line client S/W	Yes	No	SiliconAutomationSystems	http://www.sasi.com
FireWire	IEEE1394 Serial Interface	Yes	Yes/No	Multiple sources; InToto	--- http://www.intoto.com
GUI	Embedded Graphical Interface	Yes	Yes/No	Microwindows QT/Embedded Mojo Designs Eyelet Xfree	http://www.microwindows.org http://www.trolltech.com http://www.mojodesigns.com http://www.xfree86.org
HTTP	Wab Server	Yes	Yes/No	Apache standard GoAhead Magma	http://www.apache.org http://www.goahead.com http://www1.magma.ca
IrDA	Infra Red	Yes	Yes	Linux IrDA Project	http://www.cs.uit.no/linux-irda
Java	Embedded and Personal Java interpreters and JITs	Yes	Yes/No	IBM/OTI J9 Insignia Jeode TAO intent	http://www.embedded.oti.com http://www.insignia.com http://www.tao-group.com
MPEG	Video Encoder/Decoder	Yes	Yes/No	Open source encoders and decoders exists, with premium binary products from vendors like PocketVideo	

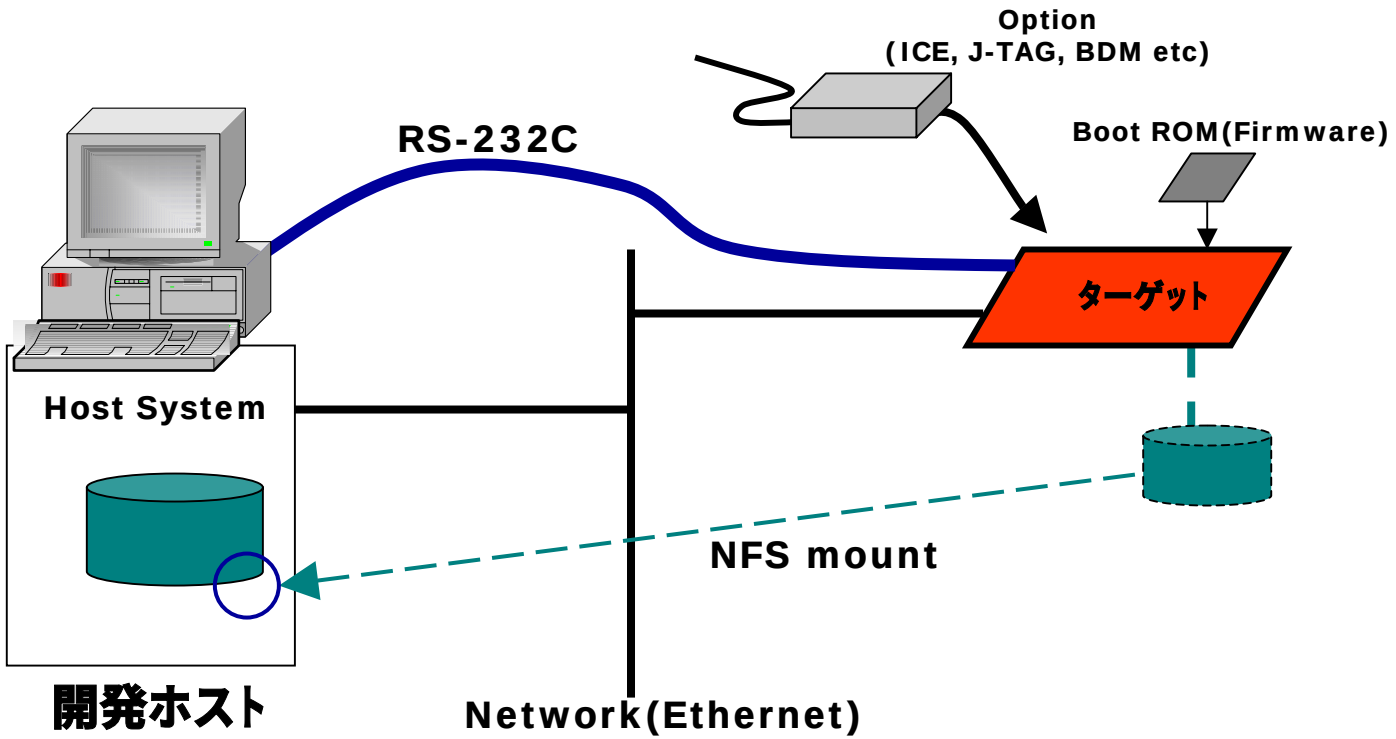
Linuxにおけるミドルウェア(2)

MONTAVISTA SOFTWARE

Item	Short Description	Available for Linux	Open Source	Comments	URL
Power Management	Disk, Display, CPU and Motherboard quiesce and sleep	Partial	Partly	New power management option exist in the 2.3 kernel, mostly to address notebook PC	
Real Audio	Streaming Audio standard	Yes	No	Real Networks	http://www/realnetworks.com
SSL	Security	Yes	Yes	OpenSSL	http://www.openssl.org/
Telephony Protocols ATM family, SS7, etc		Yes	Yes/No	Trillium Harris and Jeffries Spider Software Open IO	http://www.trillium.com http://hjinc.com http://www.spider.com http://www.nortelnetworks.com
TWAIN	Scanner interface S/W	Yes	Yes	Various open source drivers	
USB	Peripheral Interface	Yes	Yes/No	Linux Project Intoto Phenix Technologies	http://www.kernel.org http://www.intoto.com http://www.phoenix.com
VPN	Virtual Private Network	Yes	Yes/No		

クロス開発環境

- ターゲットでネットワークをサポートしていた方が効率的
  - ネットワーク経由でカーネルダウンロード
  - ルートファイルシステムをNFSマウント
- ICE、J-TAGなどが使用可能な場合
  - カーネルダウンロード、Flash書き込みなど



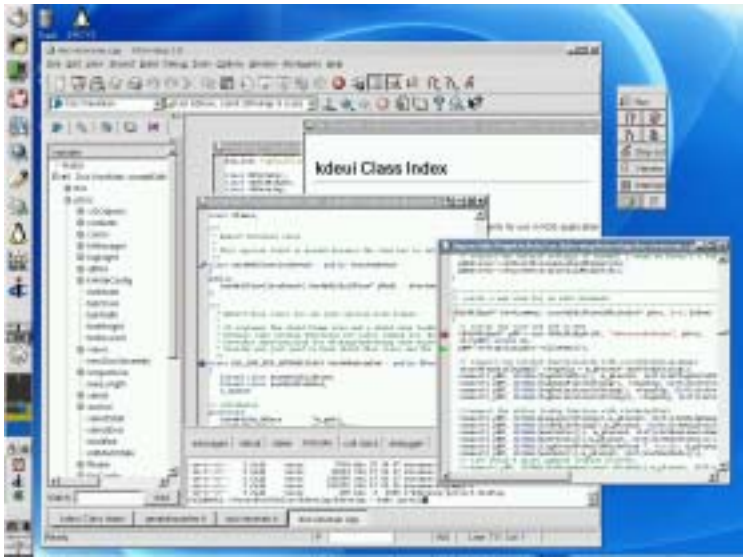
開発環境 (1)

- Gnu ツール
  - gcc / g++ コンパイラ、アセンブラ、ライブラリアン、リンカーなど
  - gdb ソースレベルデバッガ
    - カーネルおよびアプリケーション
    - オプションでH / Wデバッグのサポート
- DDD (Data Display Debugger)
  - gdbのグラフィカルなフロントエンド

開発環境(2)

MONTAVISTA SOFTWARE

- 統合開発環境
  - Code Crusader
  - Kdevelop
  - Code Warrior
  - SNiFF+
  - Source Navigator
  - Kylix
  - など



KDevelop

- トレースツール
  - Linux Trace Toolkit

- ソースコード解析
  - cscope/cbrowser
  - cprof
  - cflow

アプリケーションのデバッグ

- gccをサポートしているデバッガ
  - GNUツールのgdb、dddを利用
  - クロス開発では、gdbserverをターゲット上で起動
- ICE、J-TAT/BDMなどを使用できる環境が少ない
  - MMUへの対応
  - ロード、ページング、スワップ時のアドレス解決
  - Emblix 開発環境WGで標準化を検討

組込みシステムでの問題点

- 既存RTOSからの移行
- メモリサイズ
- リアルタイム性
- 起動時間、ファイルシステム
- ライセンス／GPL
- 開発環境の整備

MONTAVISTA SOFTWARE

RTOSから組込みLinuxへの移行

- 移植について
  - 例えば、VxWorks、pSOSystem、μiTRON
  - アーキテクチャ、API
  - タスク、プロセス、スレッド
  - メモリプロテクションとI/Oアクセス
  - メモリフットプリント
  - リアルタイム性
- 開発パラダイムの変更

既存RTOSからの移行

タスク、プロセス、スレッド

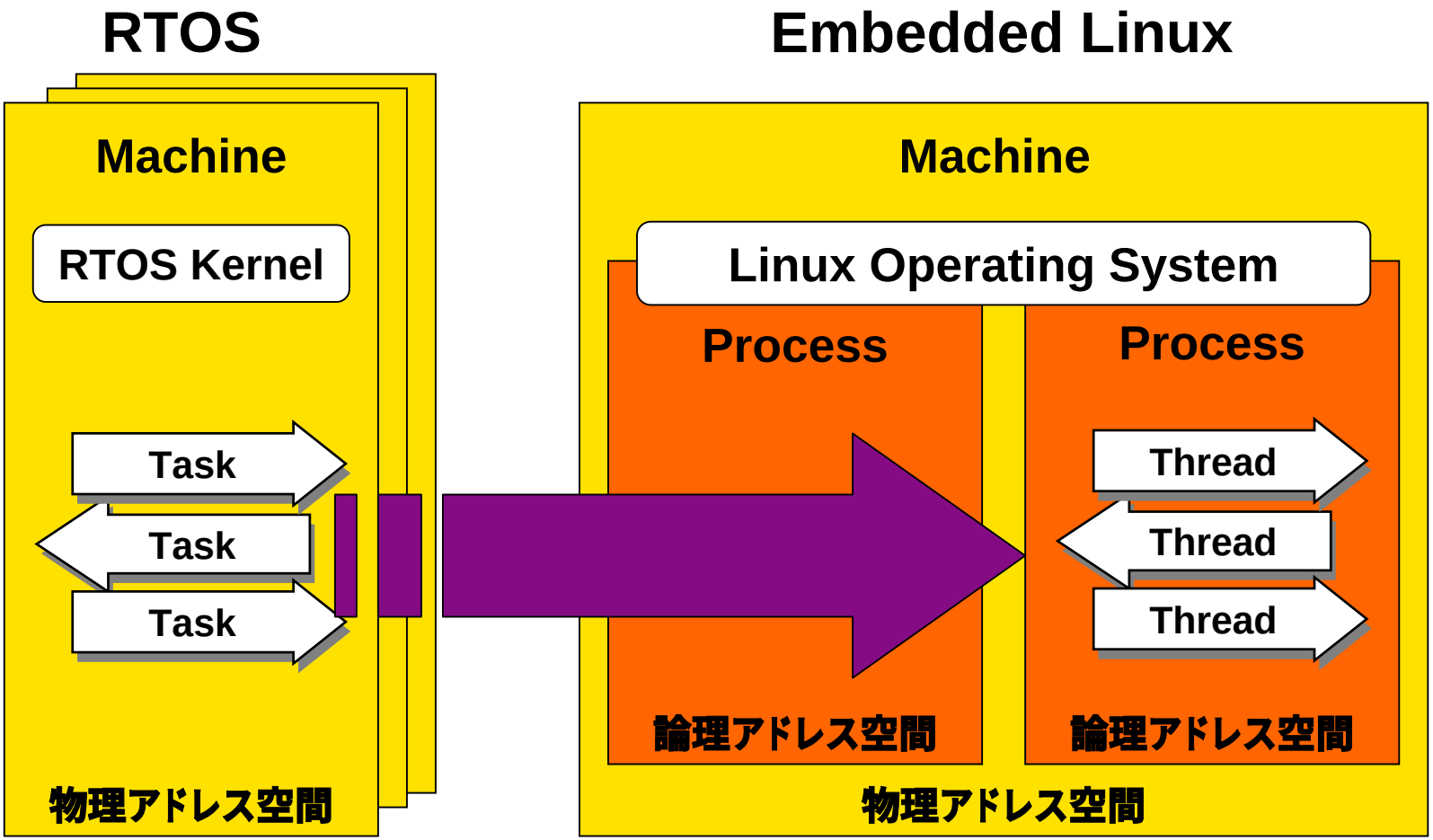
- タスク
  - RTOSによってスケジューリングされる単位 / 実体
  - レジスタ、スタック、PCなどの軽いコンテキスト
- プロセス
  - 一般的なUNIX / Linuxスケジュールの実体
  - メモリマッピングについてもコンテキストされる
- スレッド
  - POSIX.1c pthread スケジュール単位
  - RTOSのタスクとほぼ同じ

既存RTOSからの移行

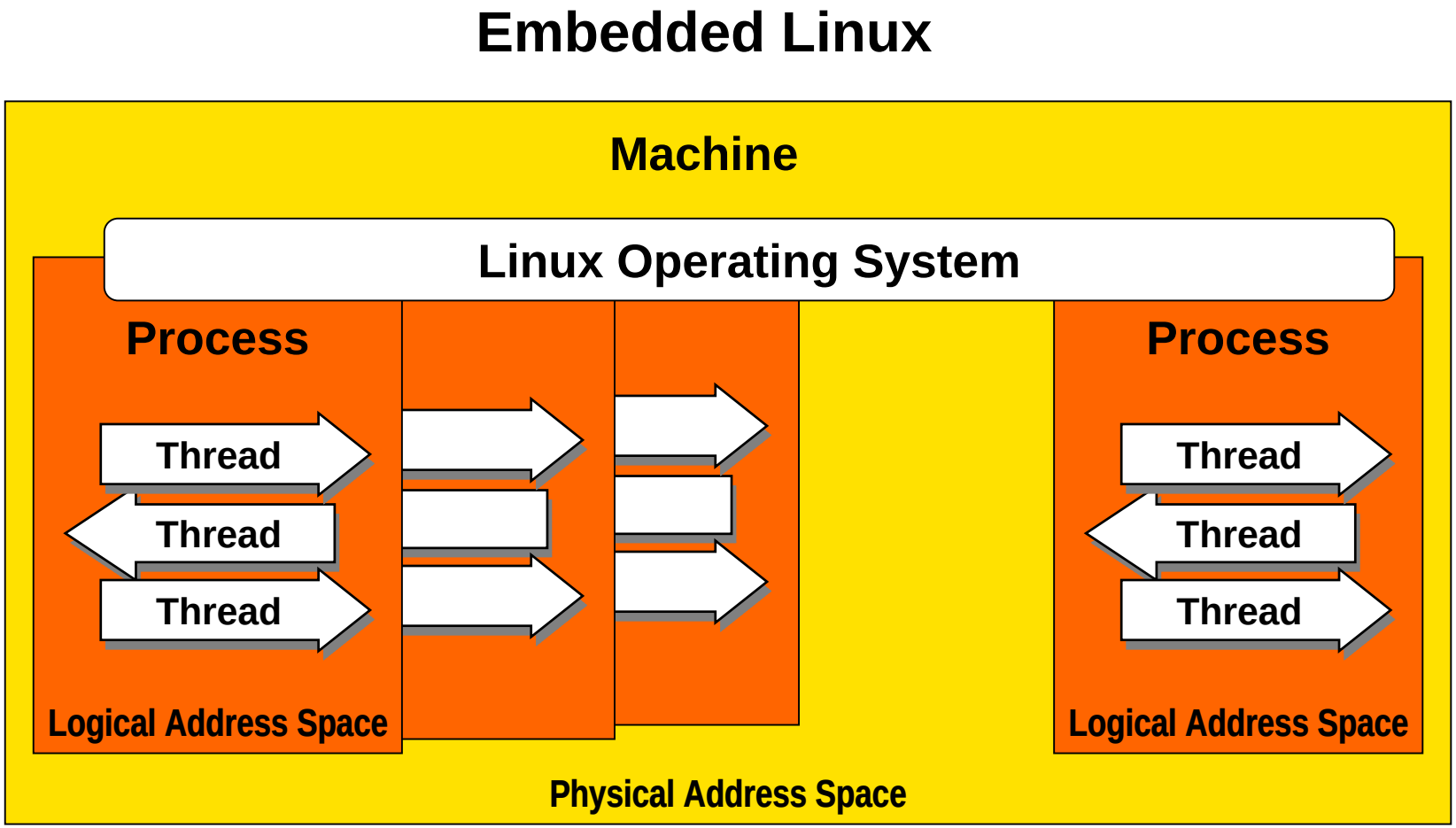
プロセスモデルとスレッドモデル

- Linuxではスレッドとプロセスは同一の実装方法
- 資源の共有が異なる (プロセスでは資源のコピーを行う。スレッドでは同じ資源を共有する)
- スレッドモデルはスレッドを同一の論理アドレス空間に配置
- プロセスモデルはプロセスを別の論理アドレス空間に配置
- RTOSタスク　　Linuxスレッド

既存RTOSからの移行
最初の段階の移植

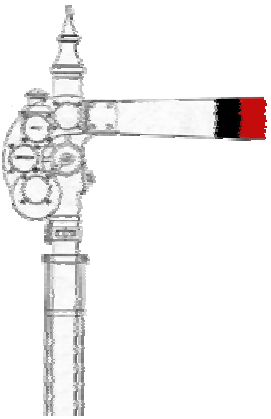


次の段階の移植


Embedded Linux
Machine
Linux Operating System
Process
Thread
Thread
Thread
Logical Address Space
Process
Thread
Thread
Thread
Logical Address Space
Physical Address Space

既存RTOSからの移行

同期メカニズム



RTOS IPC	Linux IPC
セマフォ	SVR4 セマフォ、 POSIX1.c セマフォ
ミューテックス	POSIX1.c ミューテックス、 コンディション変数
メッセージキュー	パイプ / FIFO、SVR4キュー
共有メモリ	共有メモリ
イベント	シグナル、RTシグナル
タイマ、タスクディレイ	POSIXタイマ / アラーム、 sleep ()、nanosleep ()
ウォッチドック、タスクレジスタ、 パーティション / バッファ	ツールキットによりエミュレーション

MONTAVISTA SOFTWARE

既存RTOSからの移行

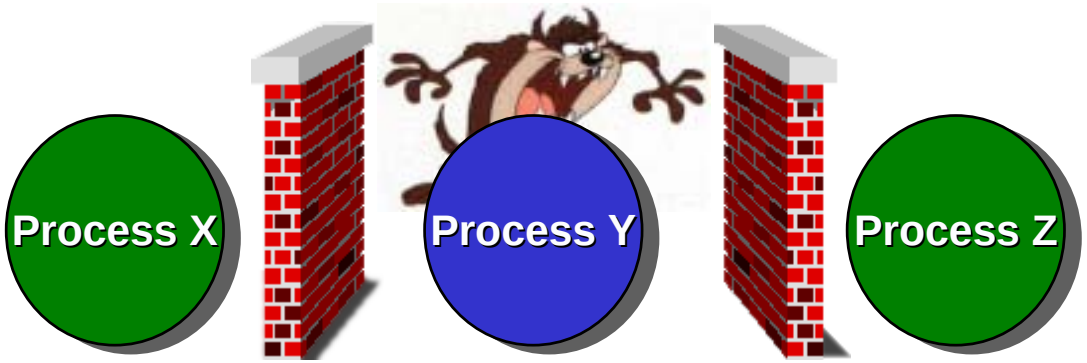
アドレッシング機構

- 伝統的な“フラット” RTOSシステム
  - MMUが存在しても使用していない
  - 物理アドレスと論理アドレスは等しい
  - 全てのマップされたアドレス空間にアクセス可能
- 制限付きあるいは、メモリ保護無し
  - インラインコードは、特権アドレスにアクセス可能
  - 特定メモリブロックに依存した“protection domains”
  - デバイスドライバモデルの使用を推奨しない

既存RTOSからの移行

メモリ プロテクション

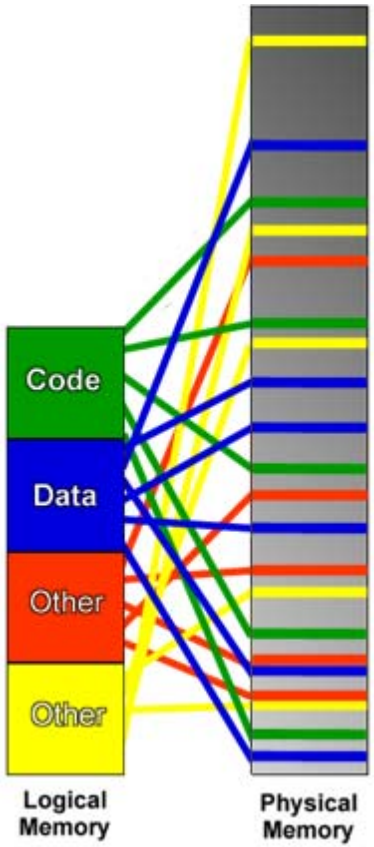
- セキュリティと堅牢性を提供
  - Offers greater security and robustness
  - 単一あるいはグループ プログラミング
- MMU により、コード / データをプロテクト
  - コード / データへのアクセスを制限



MONTAVISTA SOFTWARE

既存RTOSからの移行

アドレッシング機構



Paged

- Linuxは、仮想メモリ
  - 物理メモリは論理メモリにマップされる
    - 1ページ4 Kbyte単位で増加
    - システムメモリプールを有効に利用
  - カーネル、ドライバ、ユーザースペース
- ユーザーレベルのアプリケーションとは独立
  - それぞれのプログラムは仮想アドレス空間を持つ
  - 共有メモリ以外で、他のアプリケーションのメモリスペースにアクセスは不可能
  - カーネルとアプリケーションコードはリードオンリ

既存RTOSからの移行

H/W アクセスおよびI/O

- 伝統的なＲＴＯＳコードは、メモリマップドI／Ｏに直接アクセス
  - ドライバ標準形式が定義されていない
  - 大半がH/Wに、直接タスクからアクセス
  - 品質と移植性への挑戦
- Linuxでは、物理I/Oへのアクセスには特権が必要
  - H/W へのアクセスはデバイスドライバ使用
  - メモリマップされたデバイスへは mmap()などのシステムコールを使用しアクセス
  - 抽象化することにより、品質、H/Wへの依存性をなくす。

メモリーサイズ

- カーネル
  - 圧縮イメージで500Kbyte程度(PPC)
  - ランタイム時:700K〜1.2Mbye
  - RAM:4Mbyte〜
- ユーザーランド
  - glibc：800K〜1Mbyte
  - それ以外はアプリケーションに依存

メモリサイズ

最小メモリシステム

	Memory	Ramdisk	Kernel	Compressed Ramdisk	Compressed Kernel	Boot Image Size
X86	2.3M	389K	678K	151K	257K	X
PPC	2.1M	451K	743K	167K	237K	503K

- カーネルに含まれるコンポーネント
  - RAMディスクのみ (ネットワーク無し)
- アプリケーション
  - メモリ使用量を表示する単純なアプリケーション

MONTAVISTA SOFTWARE

メモリサイズ
基本システム

	Minimum Memory	Filesystem Size	Kernel Size	Compressed Filesystem	Compressed Kernel
X86	2.8M	2.1M	909K	749K	411K
PPC	2.8M	2.4M	1020K	839K	489K

- カーネルに含まれるコンポーネント
  - ネットワーク、NFSによるルートファイルシステム
- アプリケーション
  - Telnetd
  - Bushbox (shell、基本ユーティティ)

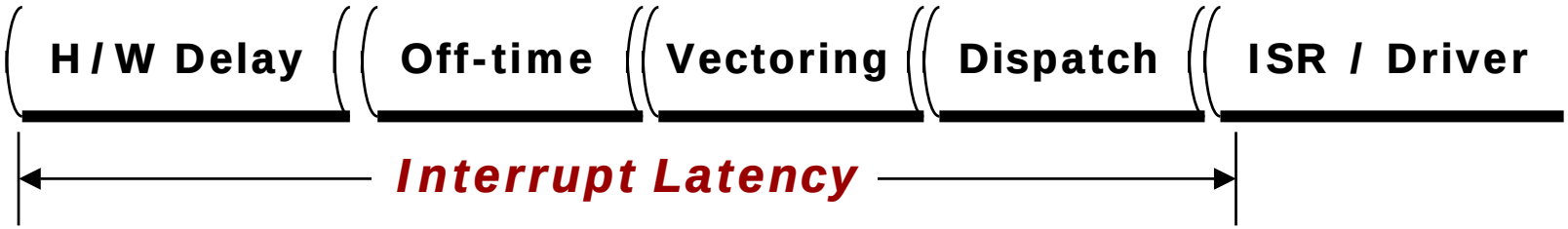
メモリサイズ削減

- 標準でいくつかの機能/サービスが動作
  - ターゲットの要求仕様を明確化
  - 不必要な機能の削減やサービスの停止
- XIP(eXecution In Place)
  FlashROMまたはROM上で直接実行
  - カーネル
  - ユーザランド
  - メモリの節約が可能(実行速度は低下)

リアルタイム性

- リアルタイム性の要求
  - 全ての組込みアプリケーションがパフォーマンスの向上を求めている
  - 30-40％ のアプリケーションがリアルタイム性を必要
  - ＜ 10％ がハードリアルタイム性が必要
- リアルタイム性のためのアプローチ
  - パフォーマンス チューニング
    - ドライバ、割込み禁止時間、カーネルスレッド・・・・
  - ハイブリッド カーネルへの置き換え
    - RTLinux、RTAI、・・・・・・
  - 標準Linuxのリアルタイム性の向上
    - MontaVista リアルタイムスケジューラ
    - MontaVista プリエンプタブル カーネル

リアルタイム性
割込み遅延時間



Cerelon 300MHz 割込みマスク時間

負荷：大きいUDPパケットターゲットにホストから送信 (5クライアント)

blocking time(us)	start file name	start file line	end file name	end file line
787	irq.c	445	softirq.c	71
174	ide-disk.c	431	ll_rw_blk.c	384
171	ide-disk.c	431	ide.c	1625
31	console.c	1862	console.c	1980
19	pc_keyb.c	485	pc_keyb.c	487
13	exit.c	394	exit.c	428
9	ll_rw_blk.c	759	ll_rw_blk.c	853
8	timer.c	665	timer.c	672
7	timer.c	290	timer.c	323
6	timer.c	290	timer.c	314

リアルタイム性
ハイブリッドOS化による拡張

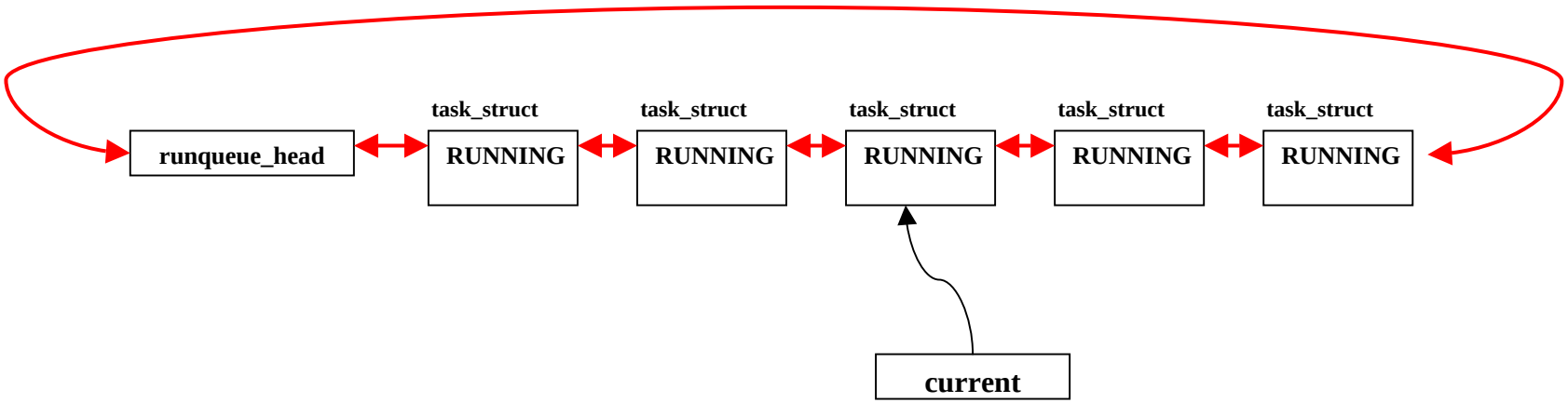
- RTLinux
  - リアルタイムOS上でLinuxが動作
  - リアルタイムアプリケーションはリアルタイムOS上でLinuxアプリケーションはLinux上で動作
  - リアルタイムOSとLinux間で通信機能を持つ
- RTAI
  - RTLinuxから派生
- Linux on ITRON
  - リアルタイムOSとしてITRON準拠のOSを搭載

MONTAVISTA SOFTWARE

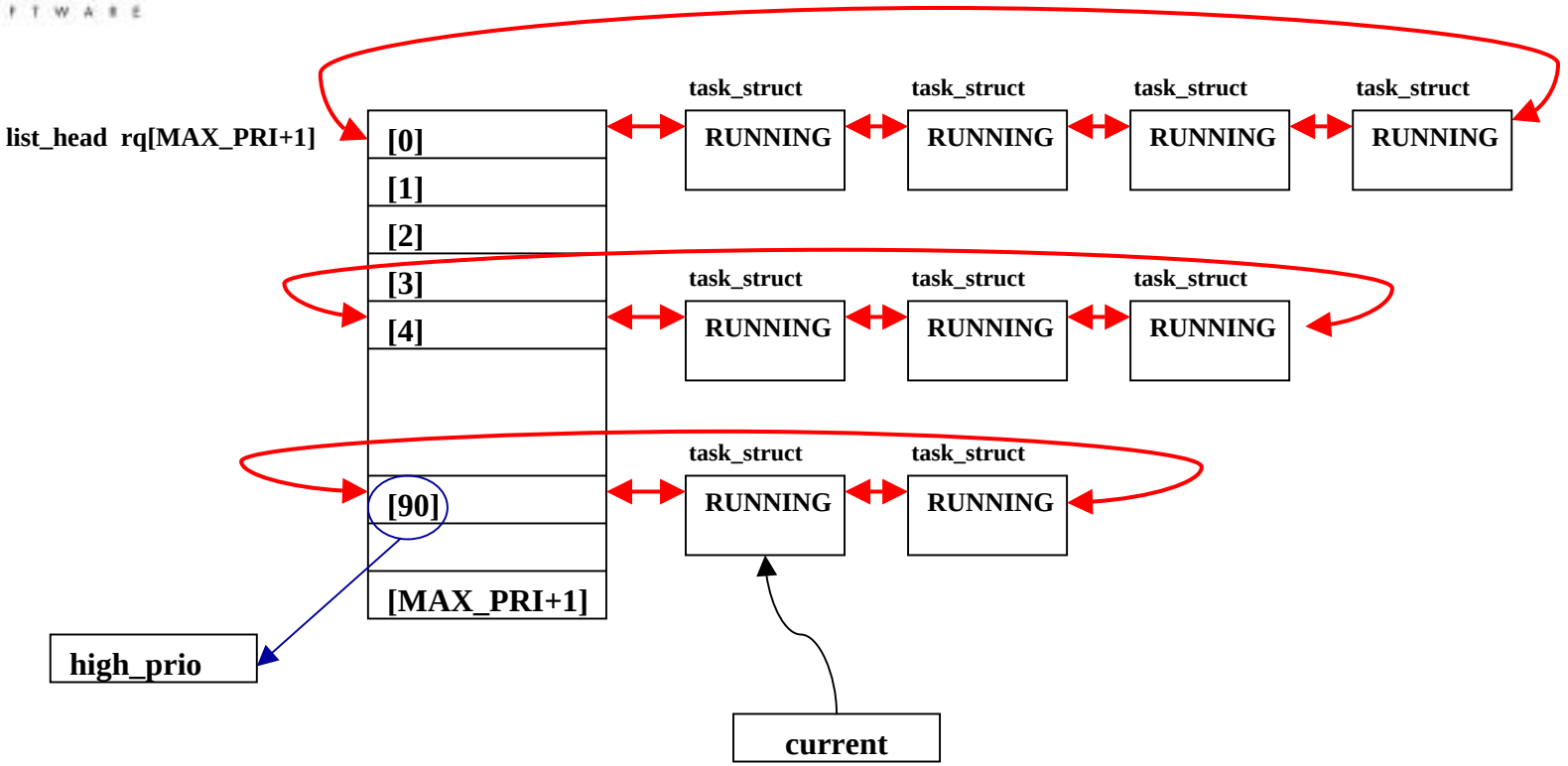
リアルタイム性
通常のスケジューラの実装 (1)

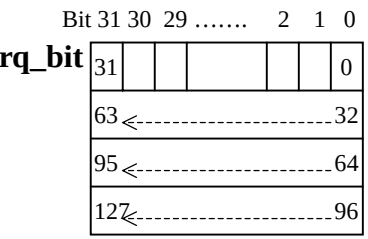
<Linux 2.4.x>

- スケジューラー schedul()
  - プロセス、スレッドの区別なくスケジューリングする
  - 実行可能プロセス (スレッド) はrunqueueにリンクされている
  - runqueueにつながれているプロセスの内、最も優先度の高いプロセスにCPUを割り当てる



MontaVista
リアルタイムスケジューラの実装 (1)





※rq[0]はTASK_OTHER

MontaVista

リアルタイムスケジューラの実装(2)

- runqueue をプライオリティ毎の構造
- タスクのステータスがTASK_OTHERは、プライオリティ＝0
- high_prio はRUNNNING状態の最も高い優先度
- rq_bitにより最も高いプライオリティを高速に検索可能
- リアルタイムプロセスのプライオリティは1～127
  (カーネルコンフィグレーションで、2047まで拡張可能)

MontaVista
リアルタイムスケジューラの実装 (3)

- スケジューラの実行時間

μSec

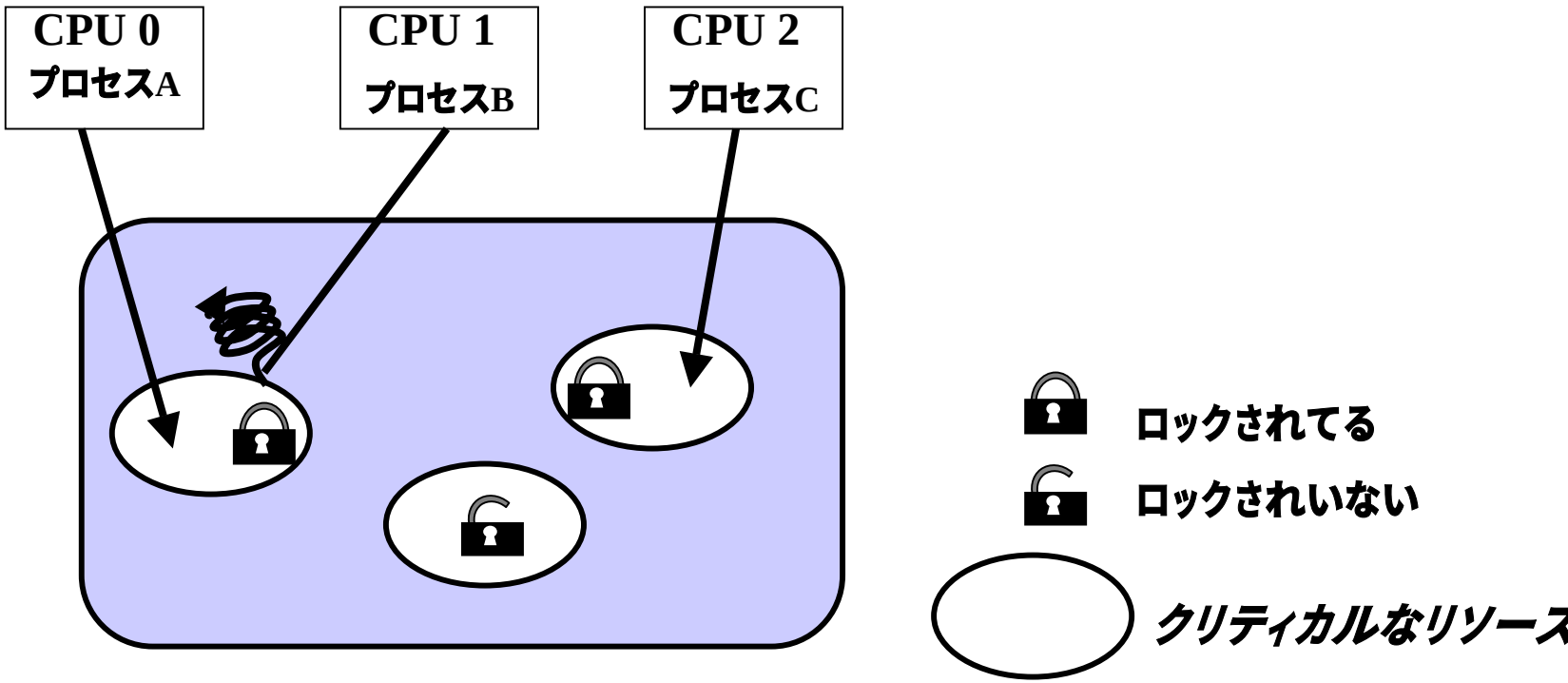
	PowerPC 8xx	PowerPC 74xx	x86 Pentium3	StrongARM	MIPS
SCHED_OTHER 50	136.57	7.14	12.86	18.6	7.14
SCHED_OTHER 100	213.81	12.63	35.89	39.46	30.4
SCHED_OTHER 300	412.71	11.22	58.65	89.13	63.96
SCHED_RR 50	56.32	3.39	6.14	6.29	6.45
SCHED_RR 100	56.28	4.02	6.32	6.29	6.96
SCHED_RR 300	56.31	5.93	6.32	6.31	7.08
SCHED_FIFO 50	56.36	3.17	4.43	6.28	6.23
SCHED_FIFO 100	56.27	4.11	6.32	6.29	6.99
SCHED_FIFO 300	56.29	6.05	6.32	6.31	7.06

リアルタイム性
通常カーネルのプリエンプション処理 (1)

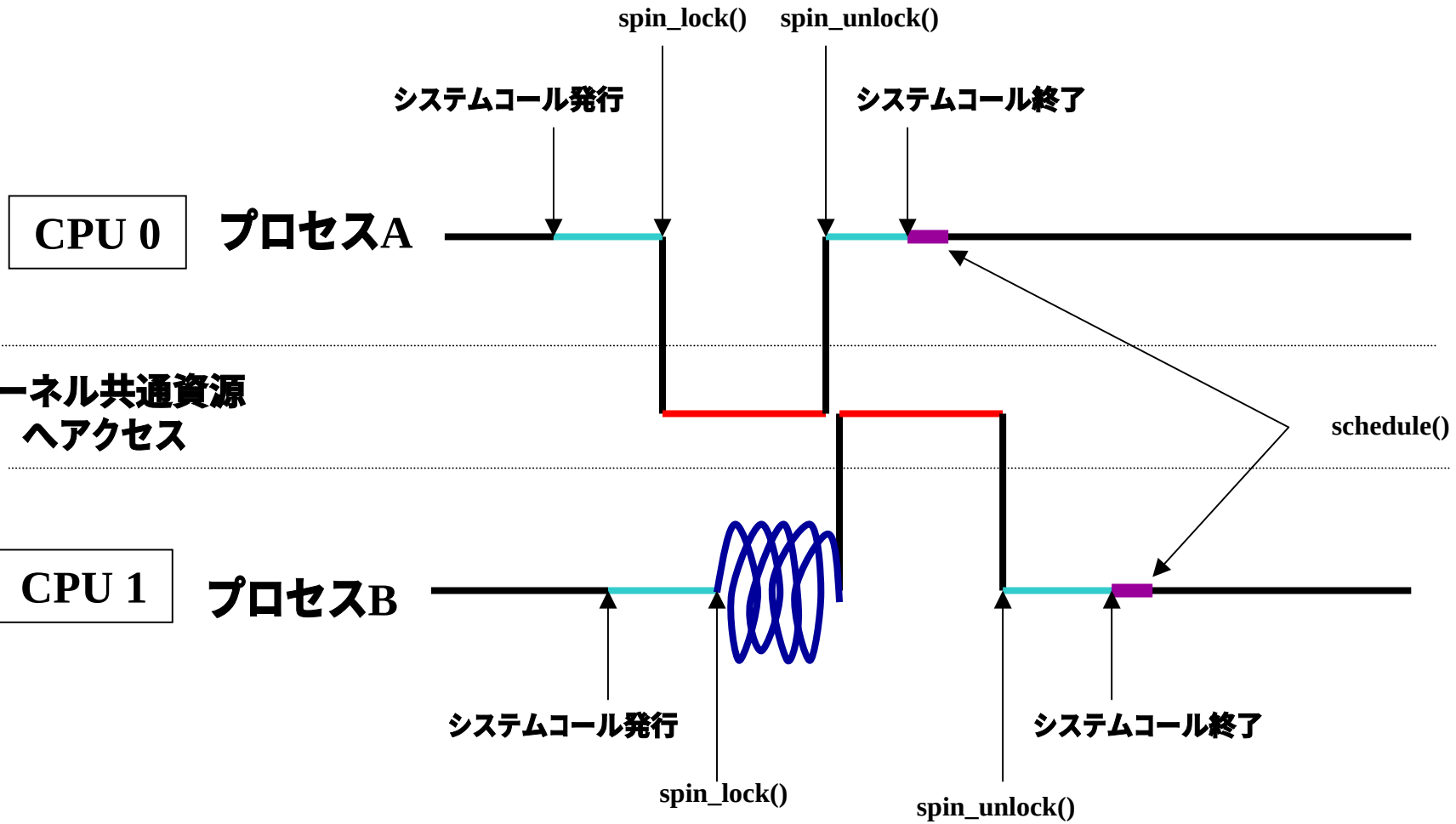
- Linuxカーネルはプリエンプティブルではない
  - プロセスがカーネルモード (システムコール中) で動作している時には他のプロセスにCPUを割り当てることは無い
- 再スケジューリングされるタイミング
  - システムコール終了時
  - 割込みハンドラ終了時
  - アイドル処理

リアルタイム性
SMP版Linuxカーネル スピンロック (1)

- カーネル内部情報のアクセスに排他制御が必要
- 他のプロセスがロックした資源を待つ場合、スピンする



リアルタイム性
SMP版Linuxカーネル スピンロック (2)



リアルタイム性
プリエンプタブルカーネルの実装 (1)

- Linux 2.4 SMPカーネルのスピンロックと同様の原理で実装
- カーネルリソースをスピンロックで保護
- spin_unlock時に必要があれば再スケジューリング

```
spin_lock(lock) {
   ctx_sw_off();
   _raw_spin_lock(lock):
}
```

```
spin_unlock(lock) {
   _raw_spin_ulock(lock);
   ctx_sw_on();
}
```

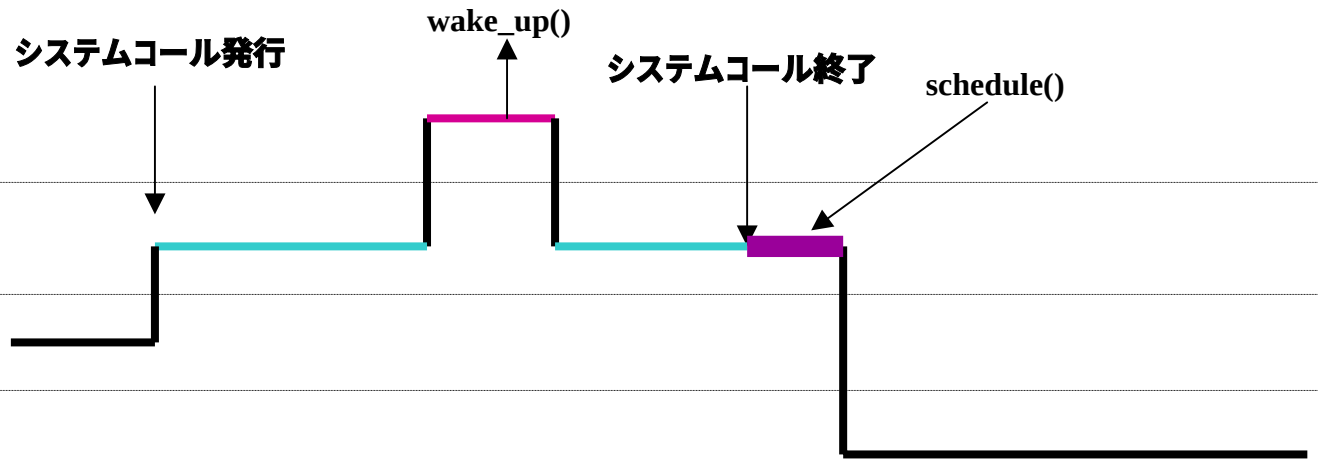
```
ctx_sw_off () {
   atomic_inc(cur->preempt_count);
}
```

```
ctx_sw_on () {
   if (atomic_dec_and_test(cur->preempt_count)
            && cur->need_resched)
            preempt_schedule();
}
```

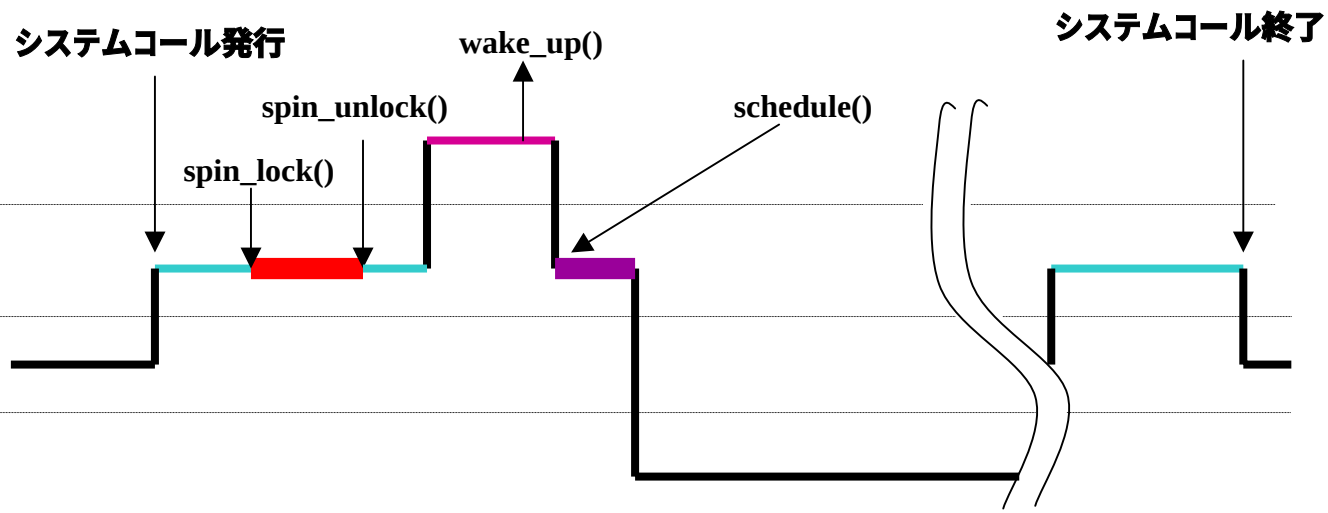
リアルタイム性
プリエンプタブルカーネルの実装 (2)

MONTAVISTA SOFTWARE

通常のカーネル



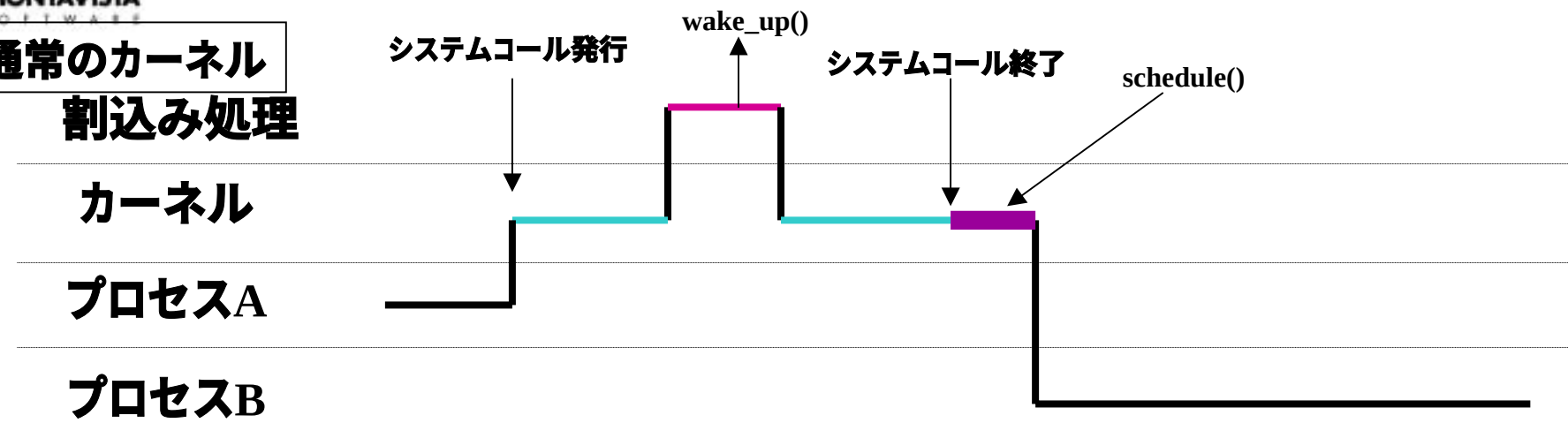
プリエンプタブル カーネル



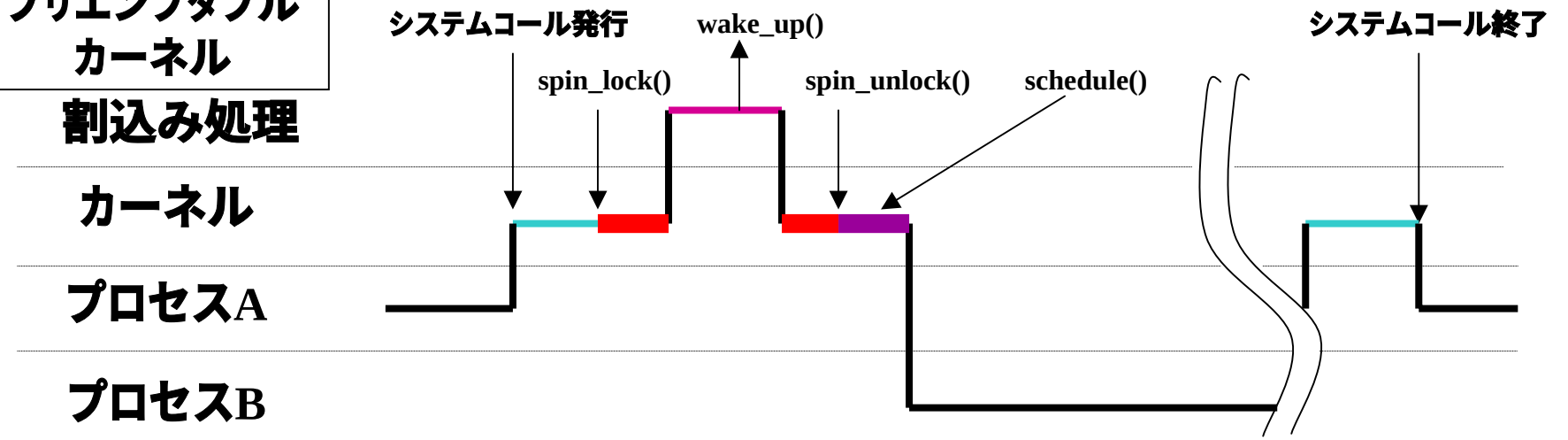
リアルタイム性
プリエンプタブルカーネルの実装 (3)

MONTAVISTA SOFTWARE

通常のカーネル

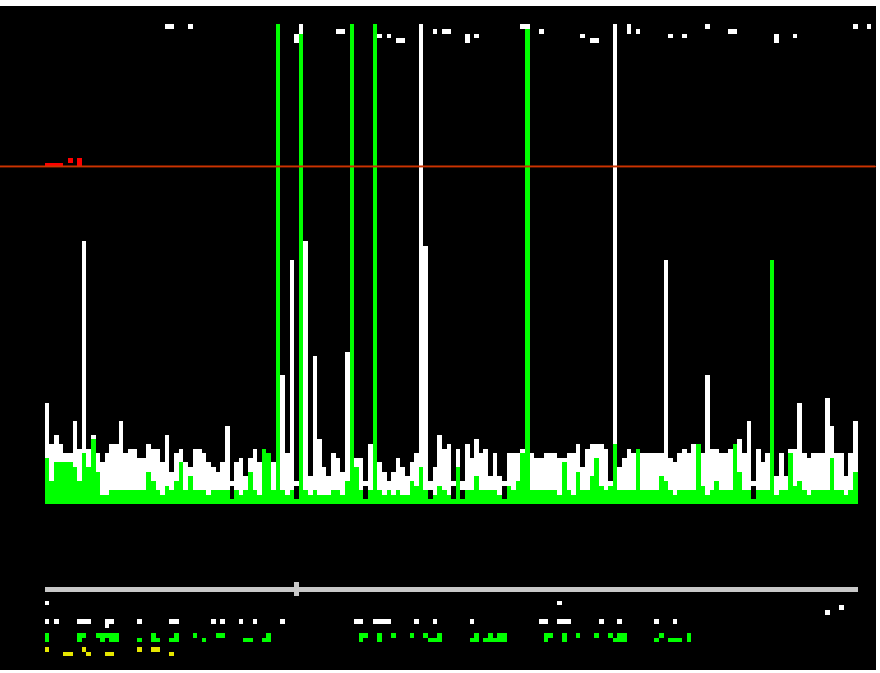


プリエンプタブル
カーネル

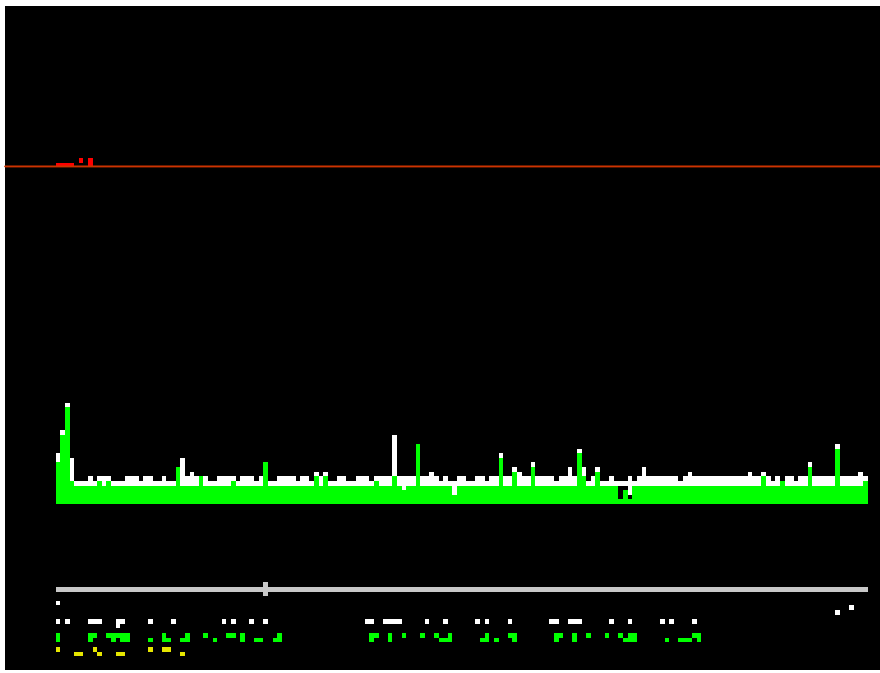


MontaVista プリエンプティブ カーネル
パフォーマンス

- 負荷をかけた状態でのオーディオ再生
  - レスポンスタイムのワーストケース



標準カーネル



プリエンプティブ カーネル

起動時間、電源OFF

- システム起動時間
  - メッセージ抑制、タイムアウトなどを工夫することで短縮が可能
  - より高速性が必要な場合、サスペンド／リジュームなどを検討
- 電源ＯＦＦ時のファイルシステムのダメージ
  - ジャーナリング・ファイルシステム（JFFS / JFFS2）
  - cramfs
  - RAMディスク, initrdの利用

ファイシステムの保護

- ジャーナリング・ファイルシステム
  - Reiserfs、Ext3 file system、JFS、XFS、JFFS/JFFS2
  - ライトアクセス中のデータは保証されない
  - ファイル・キャッシュの影響
  - 整合性の確認が不要
  - JFFS/JFFS2はFlashROMに対応
- RAM/ROMファイルシステム
  - Compressed ROM file system、Simple RAM-base file system、ROM file system
  - RAMファイルシステムは電源断で消滅
  - ROMファイルシステムは読込み専用

ライセンス／GPL（1）

- GNU Public License
  - Free Software Foundationより発行されたライセンス
- 概要
  - オリジナルのコードは著作者に属する
  - ソフトウエアの利用条件として自由であることを利用者へ要求する
  - 下流の開発者へGPLは継承する
- コピーレフト
  - プログラムの使用、複写、修正、再配布が「自由」
- 原文
  - http://www.fsf.org/copyleft/gpl.html
- Emblix 法務小委員会で協議中

ライセンス／GPL（2）

- プログラムの複製を販売可能
- 改変されたソースコードの扱い
  - 改変を加えて、公開せずに個人的に使うのは自由
  - バイナリ(とオファー)を受け取った人はソースコードを要求できる
- 機密保持契約の下で、改変を許可があるまで公開しないことに同意することは可能

- GPLのモジュールに結合されたモジュール
  - GPLが適用される
  - モジュールを結合するとは複数のモジュールを接続して単一のより大規模なプログラムを形成することを意味する
- 2つのプログラムを単に同じ媒体に隣り合わせに置くのはモジュールの結合では無い

ライセンス／GPL（3）

- 2つのプログラムが独立を保った方法でコミュニケートし、単一のプログラムとなっていなければGPLで保護されたソフトウエアと独占的なシステムを一緒に頒布可能
- GPLはインターネットを使って頒布することを要件としていない
- GPLは誰か特定の人に頒布しなければならないということを要求していない

組込みLinuxの課題

- 組込み開発者のLinux知識の不足
- リアルタイム性の改善
- どのLinuxを使用するか？
- 開発ツールの不足（ICE、J-TAGなど）
- オープンソースプログラムの品質保証
- ライセンス問題

組込みLinux選択の基準

- 100% Pure Linux
  - 100% Pure Linux or 独自のコンポーネントAPI
  - 最終製品にはLinuxではないRTOSにスイッチ？
- オープンソース
  - バイナリで提供されているもの完全にオープンソース？
- ロイヤリティ
  - ロイヤリティフリー or ロイヤリティモデル
- ベンダが組込みをよく理解している
  - デスクトップ環境と組込みシステムは大きく異なる。
- Time-to-Marketの加速！

MONTAVISTA SOFTWARE

- **お問い合わせ先**

モンタビスタ ソフトウエア ジャパン 株式会社

〒150-0011 東京都渋谷区東3-16-3 ｴﾌ･ﾆｯｾｲ恵比寿ﾋﾞﾙ6F

電話　　：03-5469-8840
ファックス：03-5469-8841
Web　　：http://www.montavista.co.jp
e-mail：info@montavista.co.jp