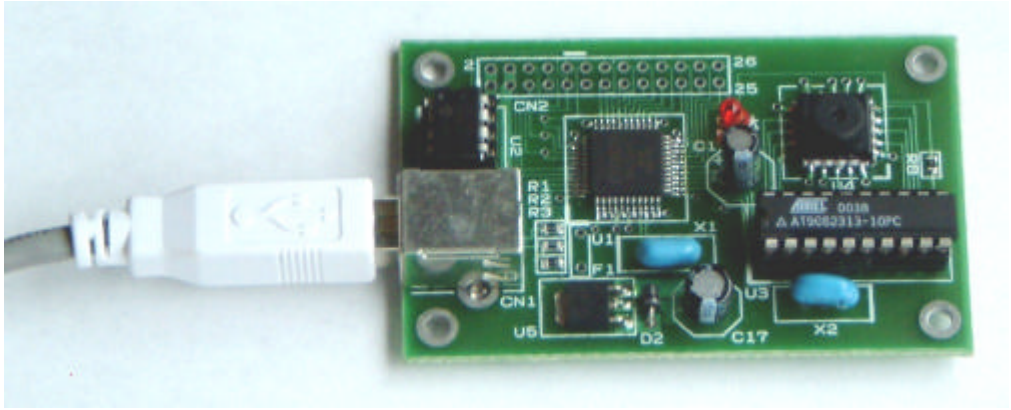
最新Linuxデバイスドライバ開発手法

株式会社デバイスドライバーズ

最新Linuxデバイスドライバ開発手法

- プログラミング・テクニック
- デバッグツール
- パフォーマンス・ツール
- カーネル2.6のトピックス



プログラミングテクニック

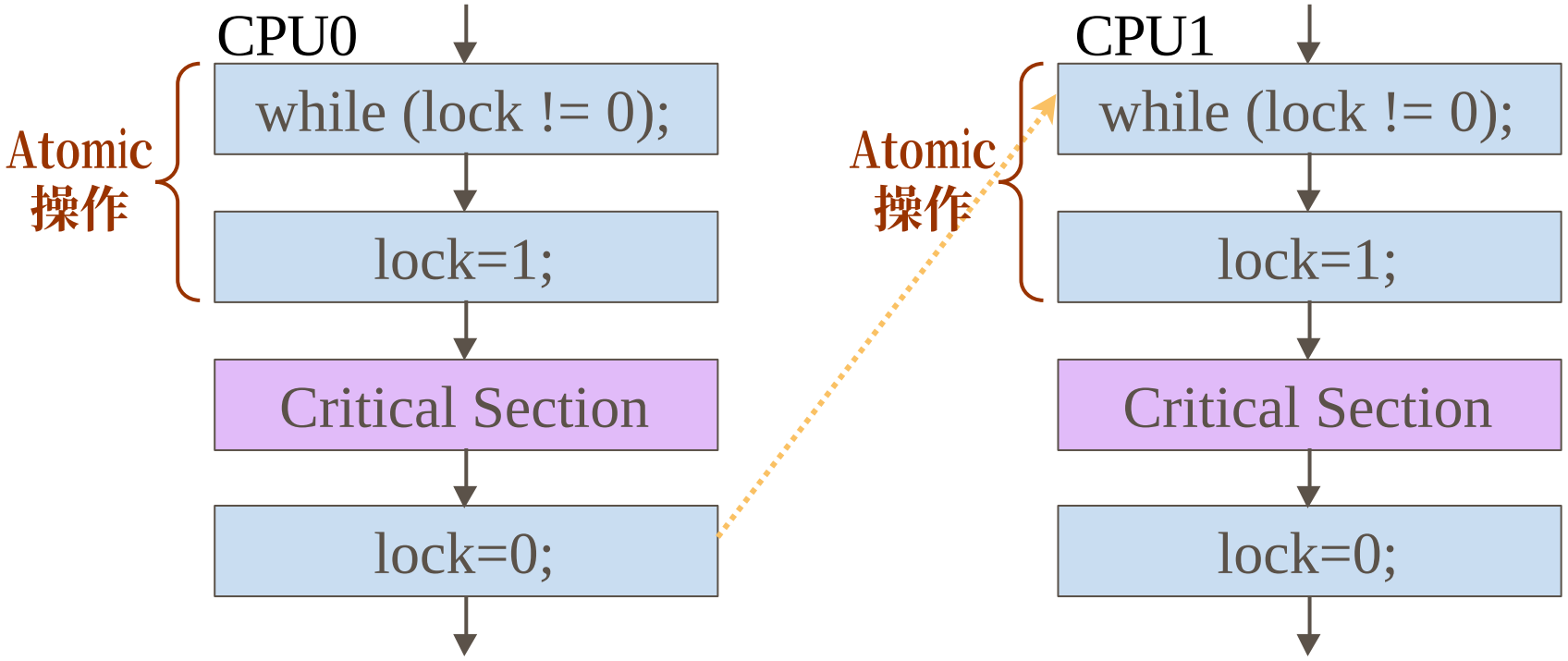
- SpinLockとAtomic操作
  - マルチプロセッサ(SMP)とHyperThreading(HT)
- taskletとtask_queue
- /procファイルインタフェイス

SMP / マルチプロセッサシステム

- なぜマルチプロセッサシステムを使うか
  - 少しでも速い処理をさせたい
  - コスト性能(3.2GHzより2.4GHz×2)
  - Pentium4に付いてくる(HyperThreading)
  - SMP vs. HyperThreading vs. Dual Core
    - 長期的にはHyperThreading→Dual Coreへの流れ
- マルチプロセッサシステムの特徴
  - 多量の重いアプリケーション処理用
  - リソースはふんだんにある場合が多い

SpinLock (1)

- マルチプロセッサで機能する一種の「セマフォ」
- シングルプロセッサでは何もしない



SpinLock (2)

- SpinLockの実装パターン（概略）

カーネル2.4

```
while (lock != 0)  {
        ;


}
```

Intel HT推奨

```
while (lock != 0)  {
  asm("PAUSE");


}
```

カーネル2.6

```
while (lock != 0)  {
  schedule();


}
```

Prescott以後で対応？

Hyper Threading

- 2.4.18以降何が変わったか？
  - カーネルのスケジューリング
  - SpinLockの命令は変わったか？
- SpinLockは万全か？
  - 記述が簡単、後付けでコーディングできる
  - 取り合えず書いておいても弊害はない
  - 必ずループする
    - 常時ダーティなメモリアクセス（遅い：キャッシュされない）
    - Test and Set命令の頻発による負荷

Atomic操作 (1)

- ループせずに排他制御する切り札
  - CPU間で同期を取るために使用できる事が保障されている(Lock命令)
- Atomic操作をセマフォとして使う
  - SMPとUPのコードの同一性
- SpinLockとの使い分け
  - 2.6のプリエンプティブ・カーネルの登場
  - 2.6ではSpinLockがプリエンプションを引き起こす!

Atomic操作 (2)

- プログラム例

```
atomic_t  a;  /* atomic_set(&a, 1); */
if (atomic_read(&a) == 0)
        return BUSY;
if (atomic_dec_and_test(&a)) { /* a == 0 */
         critical();
        atomic_inc(&p->live)
} else {
        atomic_inc(&p->live)
        reurn BUSY;
}
```

Atomic操作で、

Atomic値を1減じてその結果の値が‘0’ならばTrue(1)を返す。

Linuxデバドラ本の解説は間違い!

SMP / HT時代のプログラミング

- SpinLock→Atomic操作への切り替え
  - ループする時間に他の仕事をさせる
  - lockプレフィックス :asm()でオリジナル関数も…
- beforeイメージ ベースのコーディング
- リソースを複数個持たせて同時実行
  - リソースがふんだんにある場合
- 根本的には設計段階から考慮する
- カーネル2.6の採用

taskletとtask_queue

- tasklet
  - 割り込み時の実行をスケジュールされたルーチン
- task_queue(schedule_task)
  - 空き時間での実行をキューイングされたルーチン

tasklet

- BH(ボトムハーフ)としての実装
  - 割り込みコンテキストの中で実行
  - 制限事項：
    - リソースを待てない
    - スリープできない
    - ユーザ空間にアクセスできない
    - スケジューラを起動できない

task_queue

- schedule_task
  - スケジュール後非割り込み処理ですぐ実行される
  - 処理効率は悪くない
- Immediate_task
  - 直後のBHで直ぐに実行される（割り込み中）
  - キューイング処理では一番速い

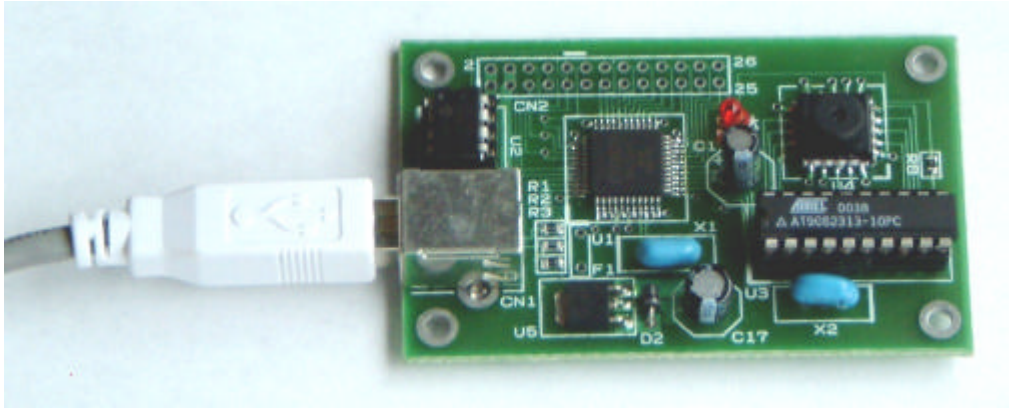
/proc ファイルシステム

- /proc/以下のパラメータ・ファイル
  - 自由に 無秩序に」使える
- ソケットや/devデバイスノードの代わり
  - システムコール・インタフェイス
  - サンプル・コード（デモ）

- 補足 :カーネル2.6以降はsysfsと棲み分け

最新Linuxデバイスドライバ開発手法

- プログラミング・テクニック
- デバッグツール
- パフォーマンス・ツール
- カーネル2.6のトピックス



デバッグツール

- kdb –静的デバッグ
- kgdb – リモートデバッグ
- JTAGデバッガ – ハードウェアデバッグ

kdb (Built-In kernel debugger)

- http://oss.sgi.com/projects/kdb/
- パッチ組込みカーネルデバッガ
- アセンブラレベルのデバッグ
  - gcc -S オプションでアセンブラリストを出力する工夫
- マニュアルやドキュメントが整備されていて紹介サイトが多い
- シリアル・コンソールの工夫！

kdb+シリアル・コンソール

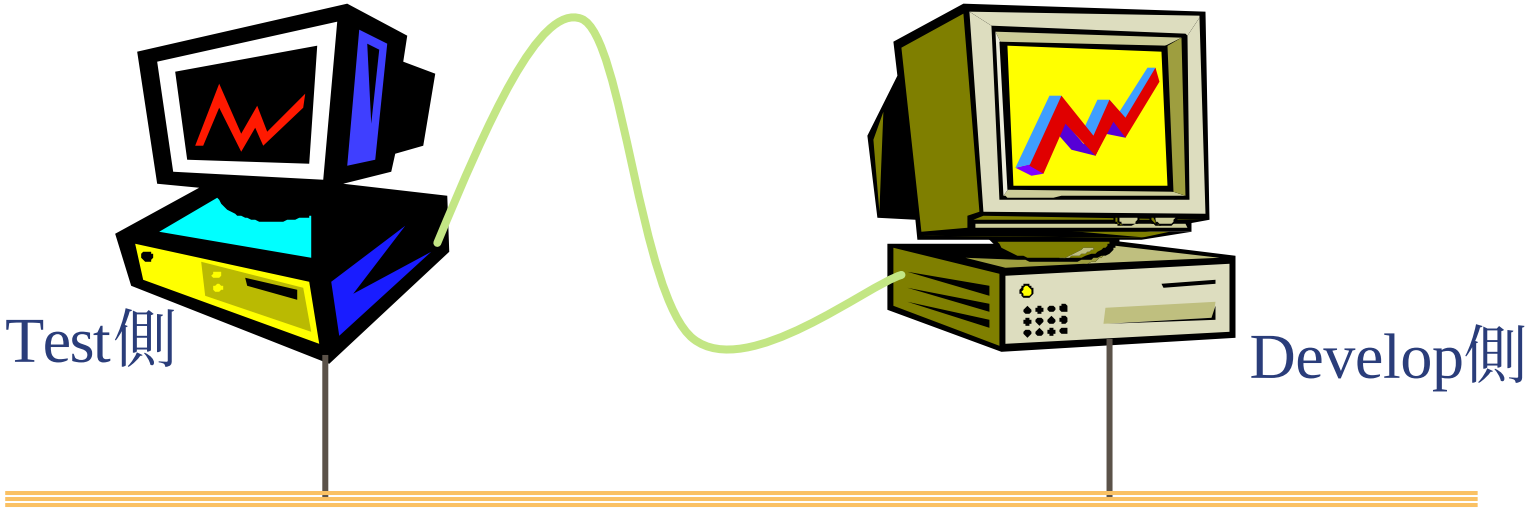
- 別マシンからリモート接続する意義
  - マルチ画面
  - カットアンドペースト
  - デバッグ履歴の保存
  - Kdbのコンソール・キーボード操作の安定

コントロール+a
キーで呼び出し

- 補足：カーネル2.6ではUSBドングル接続が使用可能

kdb+シリアル・コンソール のデモ環境

- 2台のPCをシリアルケーブルで115200bps接続
- Develop側で端末エミュレーションソフトを起動
- LANでも接続



Kdb＋シリアルコンソールの設定

- /boot/grub/grub.conf

```
default=0
timeout=10
 serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console
#splashimage=(hd0,1)/boot/grub/splash.xpm.gz
title Red Hat Linux (2.4.20+KDB on SerialConsole) from kernel.org
  root (hd0,1)
  kernel /boot/vmlinuz-2.4.20-kdb ro root=LABEL=/ console=ttyS0,115200n8r
  initrd /boot/initrd-2.4.20-kdb.img
```

そのほかに/etc/inittab, /etc/securetty, /etc/sysctl.confの修正が必要

kdbの代表的なコマンド

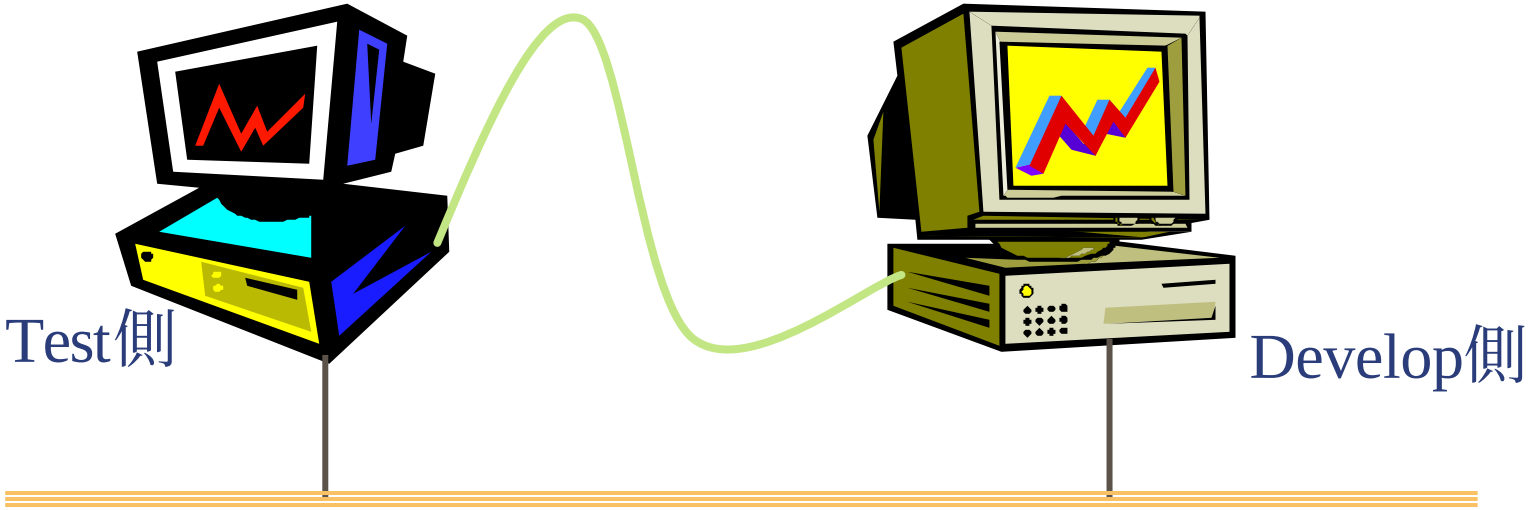
Command	Usage	Description
md	<vaddr>	Display Memory Contents
mdr	<vaddr> <bytes>	Display Raw Memory
mds	<vaddr>	Display Memory Symbolically
mm	<vaddr> <contents>	Modify Memory Contents
id	<vaddr>	Display Instructions
go	[<vaddr>]	Continue Execution
rd		Display Registers
rm	<reg> <contents>	Modify Registers
ef	<vaddr>	Display exception frame
bt	[<vaddr>]	Stack traceback
btp	<pid>	Display stack for process <pid>
bta		Display stack all processes
ps		Display active task list
sections		List kernel and module sections
lsmod		List loaded kernel modules
rmmod	<modname>	Remove a kernel module
bp	[<vaddr>]	Set/Display breakpoints
bl	[<vaddr>]	Display breakpoints
bpa	[<vaddr>]	Set/Display global breakpoints
bc	<bpnum>	Clear Breakpoint
ss	[<#steps>]	Single Step
ssb		Single step to branch/call

kgdb (linux kernel source level debugger)

- http://kgdb.sourceforge.net/
- Cソースコードレベル・デバッガ
  - カーネル全体と、テストモジュールをframe pointer optionで再コンパイルする必要がある
    - 例）
      gcc –g –fomit-frame-pointer -DKERNEL …
- 2台のPCをシリアルケーブルとLANで接続
  - Develop側ではgdbをフロントエンドとして起動
  - デバッグする同じカーネルを双方に入れておく
  - Develop側からroot権限でrshを実行できるようにする
    - **実は結構大変な作業。デバッグを行う実験ネットは分離する必要。**

kgdb のデモ環境

- 2台のPCをシリアルケーブルで115200bps接続
- Develop側でgdbをリモート・モードで起動
- LANでも接続



kgdbの設定

- カーネルパッチを当てる
- リモートからroot権限でrshが使える
  - 最近のディストリビューションでは難しい場合も
- 設定ファイル
  - rsh-serverのインストール
  - /etc/pam.d以下
  - /etc/xinetd.d以下
  - /etc/hosts.equiv

gdbの代表的なコマンド

Ctrl-C	ブレーク（中断）
q(uit)	終了
r(un) [param]	アプリケーションの実行
l(ist) [func]	ソースの表示
p(rint)	変数の表示
x [/形式 アドレス]	メモリの表示
h(elp)	ヘルプ
bt, whe(re)	バックトレース表示
b(reak) [行番号 | 関数]	ブレークポイントの設定
i(nfo) b(reak)	ブレークポイントの表示
d(elete) [ブレーク番号]	ブレークポイントの削除
n(ext)	関数の中に入らない（同じレベル）
s(tep)	関数の中に入る
f(inish)	関数の終りまで実行
c(ontinue)	継続・実行

補足 :kdbとkgdbの使い分け

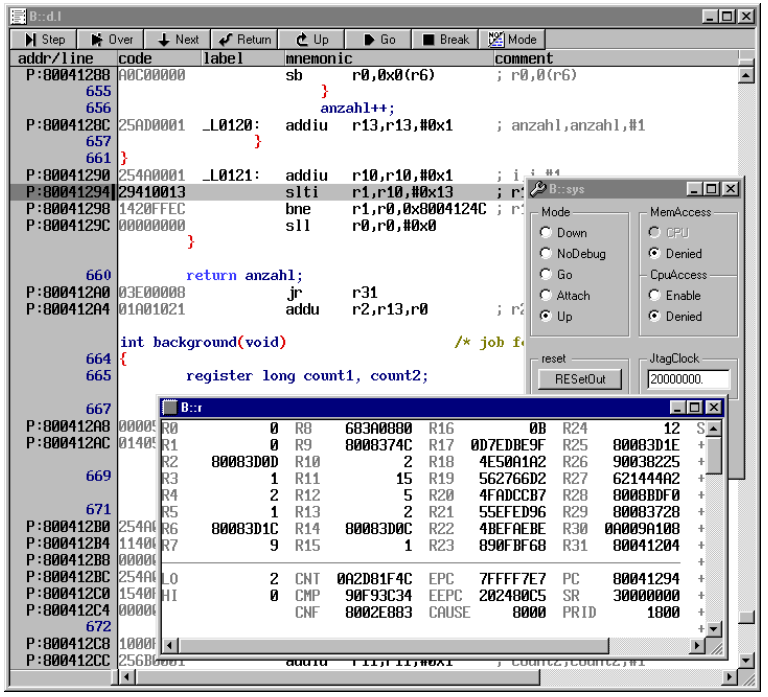
- http://kgdb.sourceforge.net/whichdebugger.html
- kdb
  - Kernelモジュール、ドライバのデバッグ用
  - アプリケーションのデバッグにも使える
  - 独自インタフェイスでアセンブラ・レベル
- kgdb
  - 大規模Kernelモジュール、ドライバの開発用
  - アプリケーションのデバッグとは別環境
  - 強力なgdbフロントエンドとソースコード・デバッグ

JTAGデバッガ

- JTAG経由でCPU内蔵のデバッグ機能を使用
  - 対応CPUが限定される
- サポート・ソフトウェアによる豊富な機能
  - ソースコード・デバッグ
  - ブレークポイント
  - トーレスバック
- gccとLinux MMUのサポート
  - カーネルからアプリケーションまでソースデバッグ

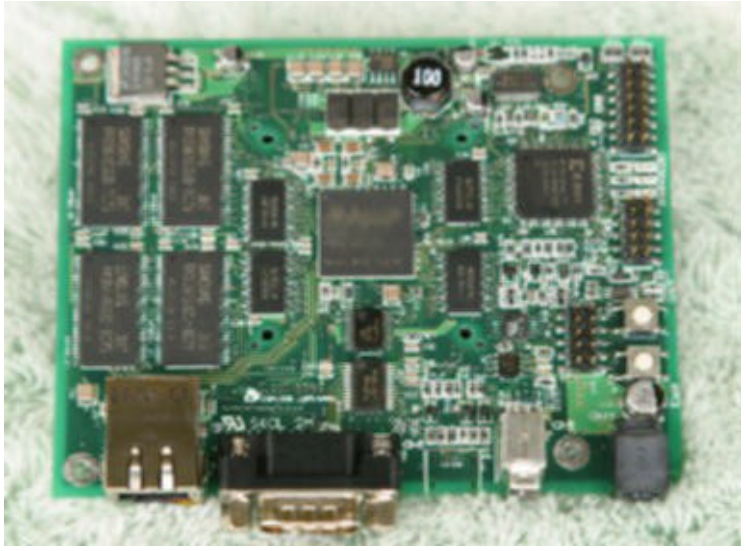
JTAGデバッガ 続き)

- 独ロータバッハ社製JTAG Debuggerシリーズ
- 対応CPU
  - ARM, MIPS, PPC, SH, H8, i186, 各種DSP, FPGA
- 提供機能
  - ハードウェアデバッグ
  - ソフトウェア・シミュレータ
  - ROMエミュレータ



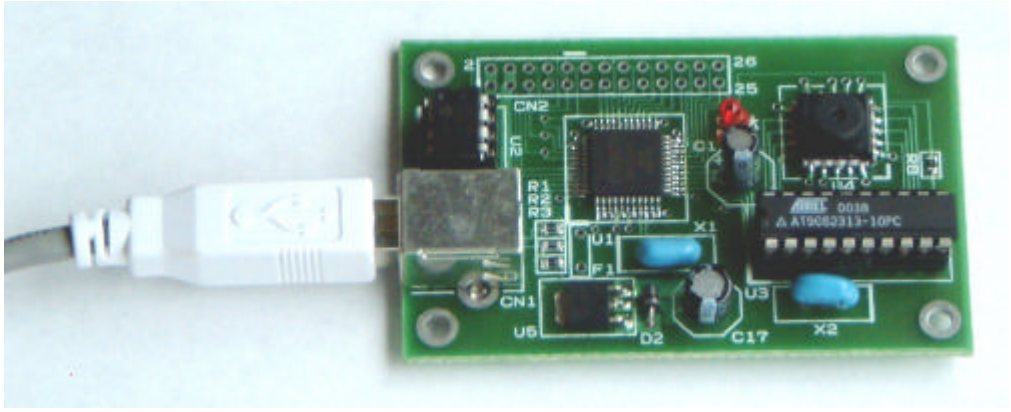
JTAGデバッグ •デモ

- Device Drivers社製 組み込みLinuxボード
  - AMD Au1100 400MHz (MIPS32アーキテクチャ)
  - 128MB RAM(最大256MB), 8MB Flush ROM
  - CF Type2 × 2, UART × 3, IrDA
  - LAN, USB Host/Target
  - 内蔵LCDコントローラ
  - GPIO
  - 名刺 2枚サイズ
  - 低消費電力



最新Linuxデバイスドライバ開発手法

- プログラミング・テクニック
- デバッグツール
- パフォーマンス・ツール
- カーネル2.6のトピックス



パフォーマンスツール

- Kernprof (Kernel Profiling)
  - カーネル・プロファイリング
- Lockmeter (Linux kernel lock-metering)
  - SpinLockメータ
- PerfCtr(memory profiling tool)
  - カーネル・プロファイラ
  - Hardmeter=メモリ・プロファイリング
- Oprofile
  - システムワイド・プロファイラ

Kernprof

- http://oss.sgi.com/projects/kernprof/
- カーネル内ルーチンのプロファイラ
  - カーネルパッチ＋kernprofコマンド
- 出力結果(gmon.out)をgprofで処理可能
- i386, ia64をサポート
  - 他のアーキテクチャ用は個別に移植されている
    - PowerPC, ARM
  - 現在は2.4.20用が最新版
    - カーネル2.6対応作業中

Lockmeter

- http://oss.sgi.com/projects/lockmeter/
- Kernel Spinlock Metering for Linux
  - SpinLockの利用状況のレポート
- カーネルパッチ＋lockstatアプリケーション
- i386, ia64, Alpha, Sparc64, mips64をサポート
  - カーネル2.6対応作業中
  - 公開版では2.4.18 / 2.5.17が最新版となり
    メンテナンスされてない→実験研究用

PerfCtr とhardmeter

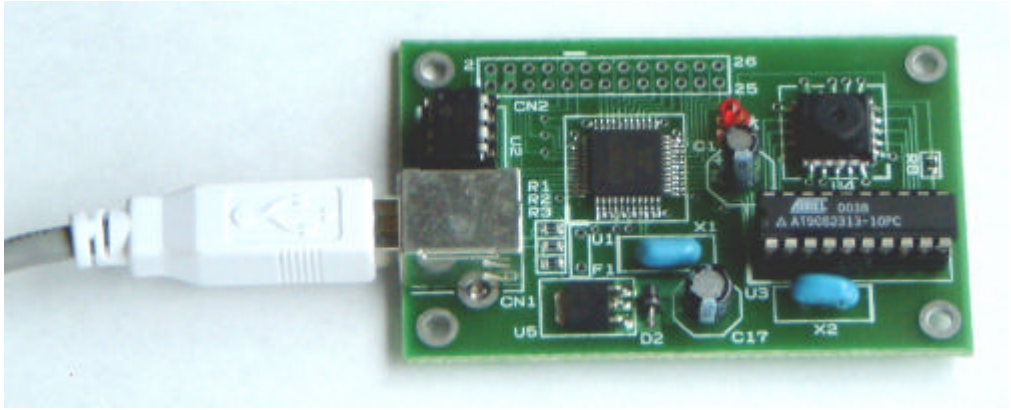
- http://user.it.uu.se/~mikpe/linux/perfctr/
- PerfCtr – カーネルパッチ形式のモニタドライバ
  - x86ファミリ向け(P6-Xeon, K7, K8, Geode, C3)
  - ローダブル・モジュールまたは組み込み形式
  - アプリケーション・ライブラリの提供と多数の派生プロジェクト
- hardmeter, xhardmeter – パッチ＋アプリケーション
  - http://sourceforge.jp/projects/hardmeter/
  - Pentium!!!以降のCPUのパフォーマンス・レジスタを使用
  - メモリ・プロファイリング
    - キャッシュメモリの利用状況をレポート
  - IPA2002年度「未踏ソフトウェア創造事業」の成果

Oprofiler

- http://oprofile.sourceforge.net/
- システムワイド・プロファイラ
  - アプリケーションデーモン＋ローダブルモジュール
  - カーネル・パッチは不要！
    - モジュールはカーネル2.6からサポートされている
- i386, ia64, Alpha, Athlon, Hammerをサポート
  - 豊富な実行時オプション
  - 独自のGUI(Qt対応）
  - Gprof互換のフォーマッタ(opgprof)

最新Linuxデバイスドライバ開発手法

- プログラミング・テクニック
- デバッグツール
- パフォーマンス・ツール
- カーネル2.6のトピックス



sysfsとkobject

- 新しいドライバ・モデルの導入
  - /sys以下に論理接続・物理接続
  - PnP, hotplugとの連携
- 新しいモジュール・フォーマット
  - **.o → .ko**
  - 複雑になったコンパイル・リンク手順
    - modpostの登場
    - Russelの新しいmodule-init-tools

hotplug

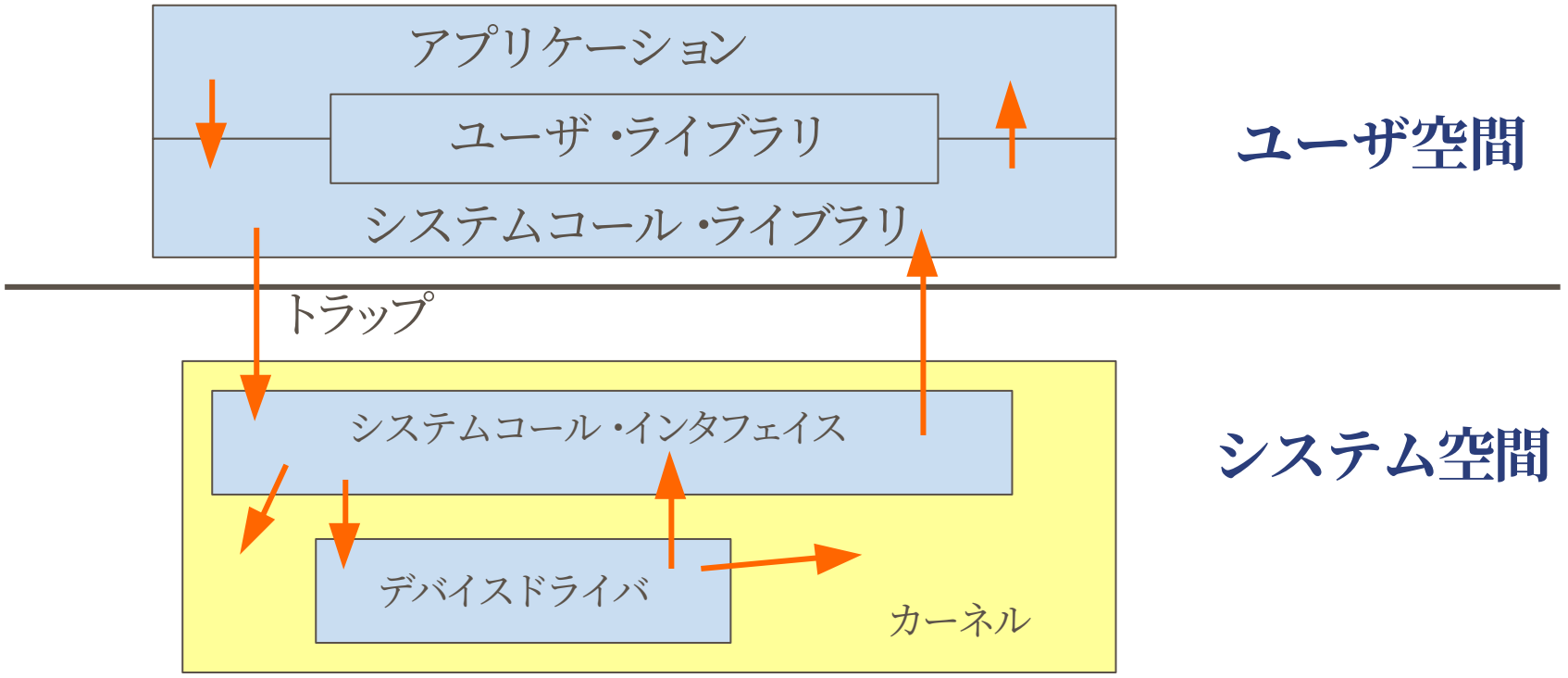
- PnPとhotplugの違い
  - PnP, hotplugできないデバイスが例外扱いになる
- hotplugは広範囲なサポートへ
  - 現在：
    - USB, IEEE1394, Network, PCI(CardBus), IBM Channel
  - 開発中：
    - Dock, Input, PCI(ext.), SCSI, Memory, CPU, IDE
- devfs → udevへの交代
  - devfsはudevに置き換えられていずれ無くなる

パワーマネジメント

- ACPIの本格的サポート
  - S1, S3, S4をサポート
    - - On - Standby - Suspend - Hibernate - Emergency
  - Usbほか、関連デバイスドライバの書き換え
- hotplugとの連携
- sysfsとの連携
  - 完全なパワーマネジメント実現の目的

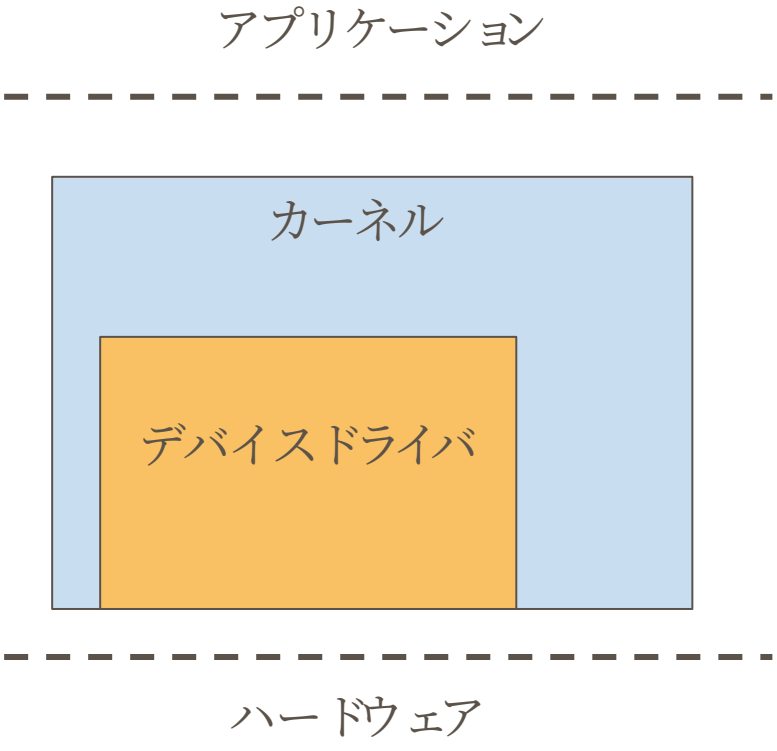
補足資料 :システムコール

- デバイス入出力システムコール処理の流れ



補足資料 :ローダブル •モジュール

スタティックリンクの ドライバ



ローダブルモジュールの ドライバ