

# Linuxにおけるデバイスドライバの 開発とデバッグの実際



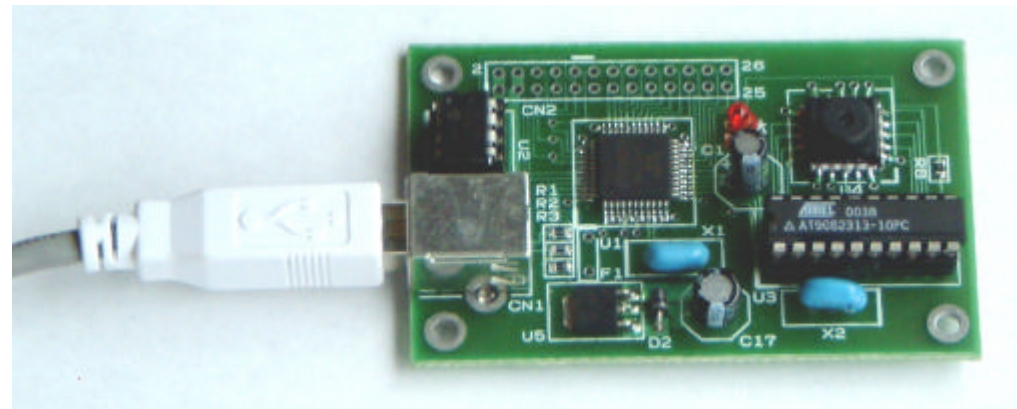
株式会社デバイスドライバーズ  
hidaka@devdrv.com

# はじめに

- デバイスドライバとは？
- なぜLinuxなのか？

# Linuxデバイスドライバ開発

- 準備と心構え
- カーネルとデバイスドライバ
- デバイスドライバ開発
- デバッグツール
- 実機デモ



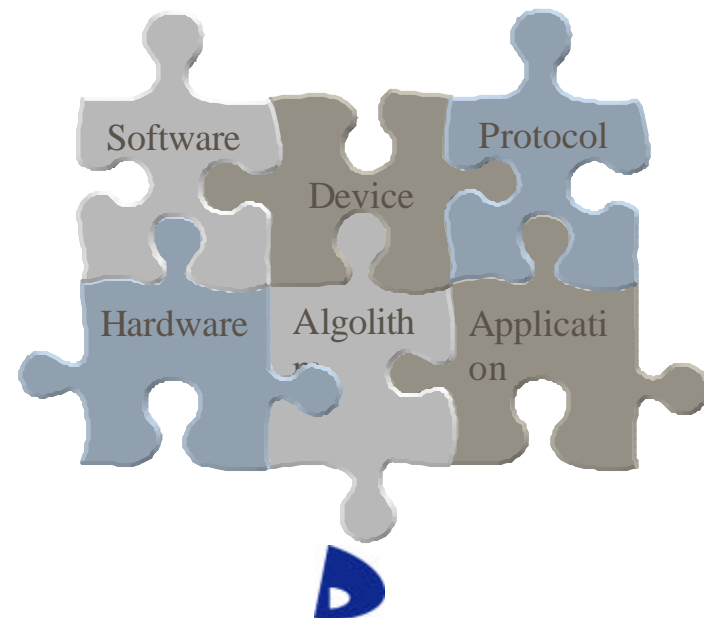
# 確認 :スキルと環境

## ■ 開発に必要なスキル

- ハードウェアの知識
- ソフトウェアの知識
- + アルファ (度胸、経験、根性、発想、人脈、英語力)

## ■ 開発に必要なもの

- 開発環境
- ターゲット環境
- テスト用アプリケーション



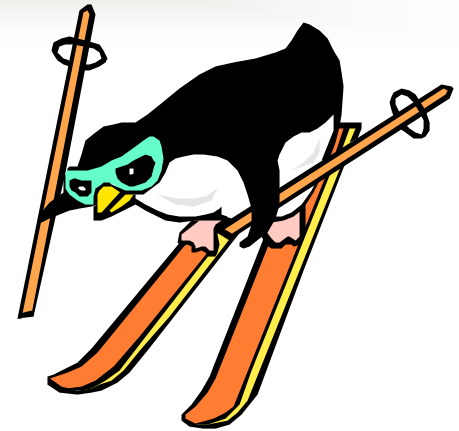
# 準備 : マシン環境

- 新しくて速いマシンを用意
  - 速いCPU
  - 速いディスク
  - 128MB以上のメモリ
- 標準でサポートのビデオカードとNIC
  - ATI Radeon
  - GeForce2 MX400
  - Matrox G400
  - オンボード・ビデオには注意！



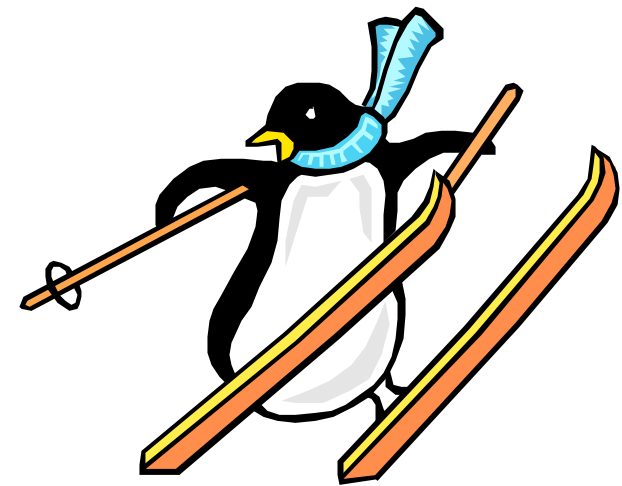
# 準備 :OS環境

- カーネル2.4系
  - Redhat 7.1以降
  - Turbo Linux, Vine, Debian, Slackware,
- kernel.org の純粹カーネル
  - 2.4.10以降がおすすめ
- 必要なオプションのインストール
  - カーネル開発環境
  - ネットワーク環境の動作



# 準備 : ツール類の使いこなし

- OSのインストール
- Shell, コマンド, ツール
  - bash, ln, tar, gzip, patch, diff, rpm, linuxconf, ftp
- エディタ, X-Window
  - vi, ...
- スクリプト言語
  - sh, Perl, ...
- C言語プログラミング
  - Makefileとmake
  - gccのオプション、ライブラリ



# インターネットの活用

- 情報の検索術
  - 検索エンジン
  - オープンソースの活用
- コミュニティの活用術
  - メール、BBS、人脈
- 情報の発信術
  - ホームページの作成





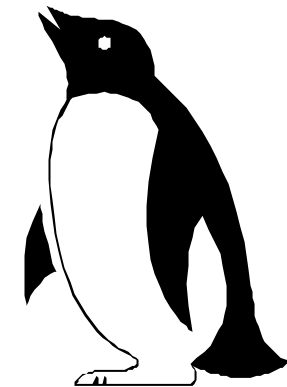
# 情報サイト(英語)

- <http://kernel.org/>
- The Linux Document Project
  - <http://www.tldp.org/>
- Linux usb
  - <http://www.linux-usb.org/>
- <http://examples.oreilly.com/linuxdrive2/>
- The Open Source Development Network
  - <http://www.osdn.com/>
  - <http://sourceforge.net/>
  - <http://freshmeat.net/>



# 情報サイト(日本語)

- 日本のLinux情報
  - <http://www.linux.or.jp/>
- OSDN Japan
  - <http://osdn.jp/>
- Linux at IBM
  - <http://www-6.ibm.com/jp/linux/>
- Change Log
  - <http://www.changelog.net/>
- ASCII Linux
  - <http://linux.ascii24.com/>



# さらに上を目指すためのヒント

- アプリケーション開発経験
- ハードウェアとデバイスの知識
- 各種計測器の使いこなし
- ネットワークとセキュリティ
- GPL、Open Source、FreeSoftの理解と区別



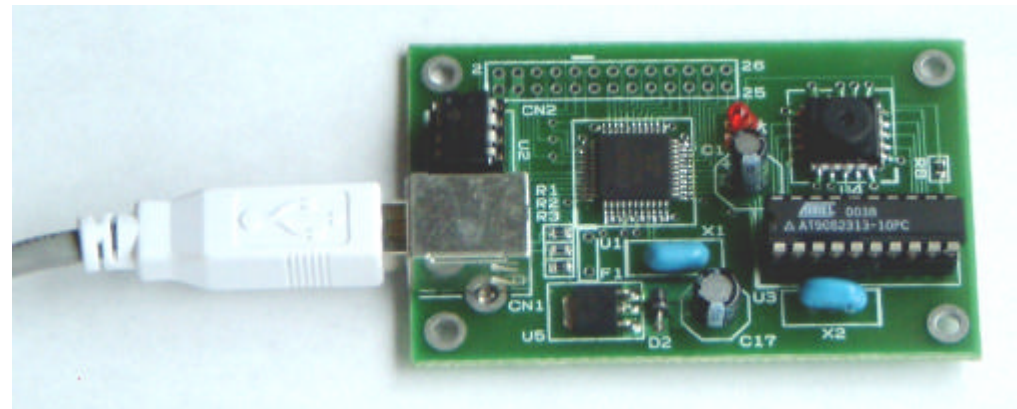
# まとめ 準備と心構え

- 開発効率を向上させるために
  - 最新のハードウェアとソフトウェア
  - 標準品のデバイス
- スキルを向上させるために
  - OSインストールやカーネルコンパイル
  - 日頃からUnix / Linuxに親しむ
    - Unix系のコマンドと操作
  - プログラミング以外の開発作業に慣れる
- 情報を収集、活用するために
  - インターネットの活用

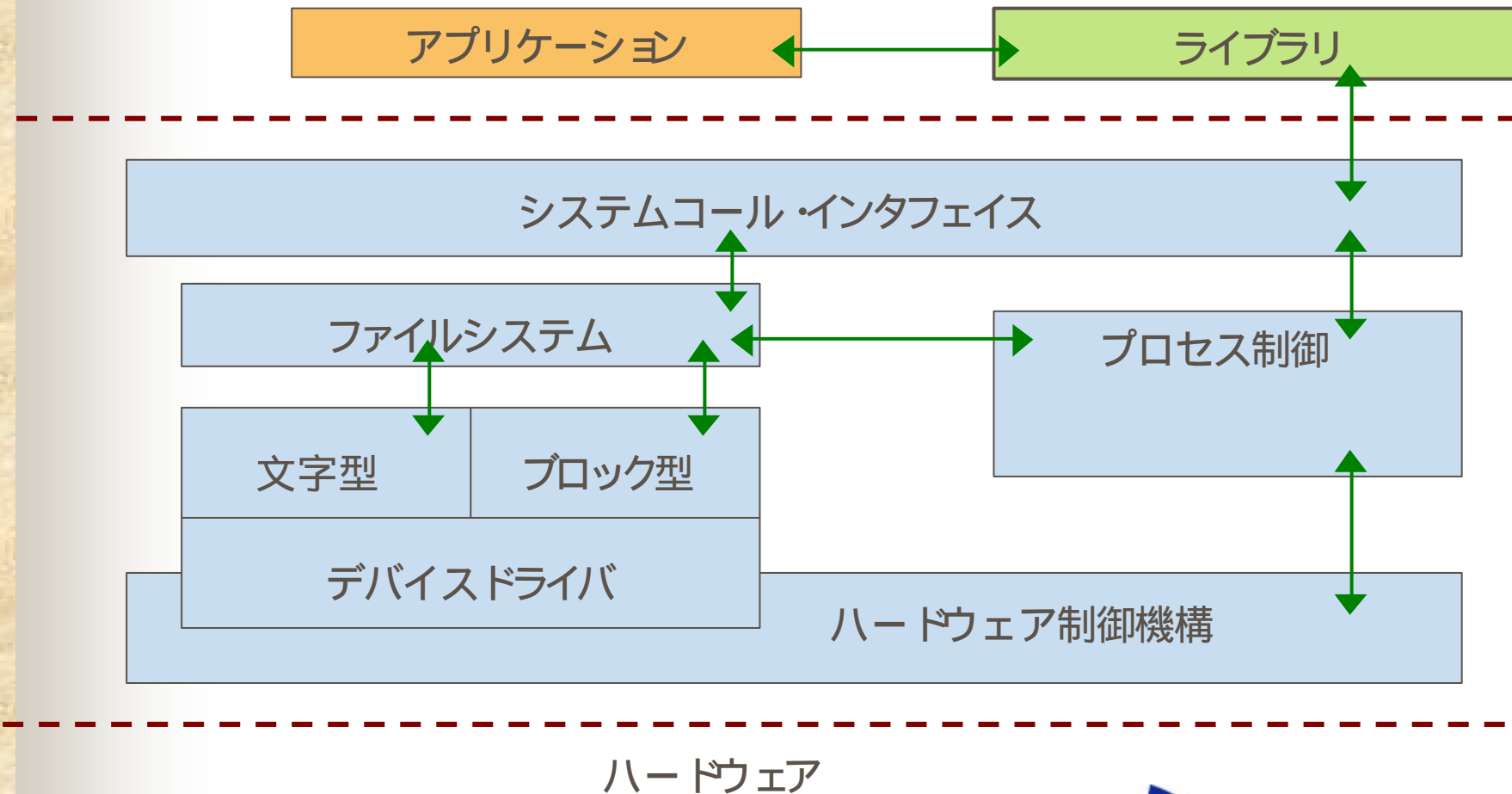


# Linuxデバイスドライバ開発

- 準備と心構え
- カーネルとデバイスドライバ
- デバイスドライバ開発
- デバッグツール
- 実機デモ

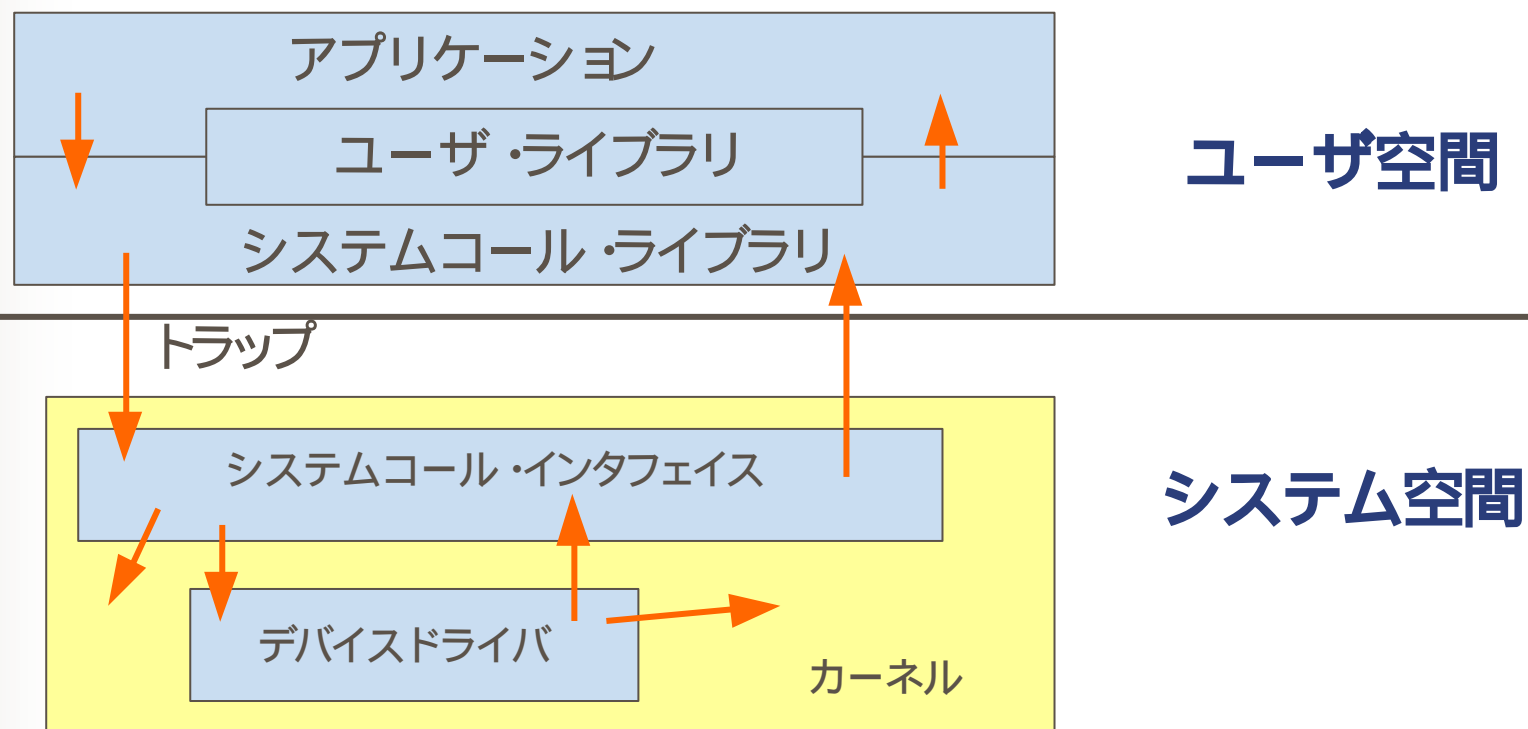


# Linuxカーネルの構造



# 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



# 入出力とシステムコール

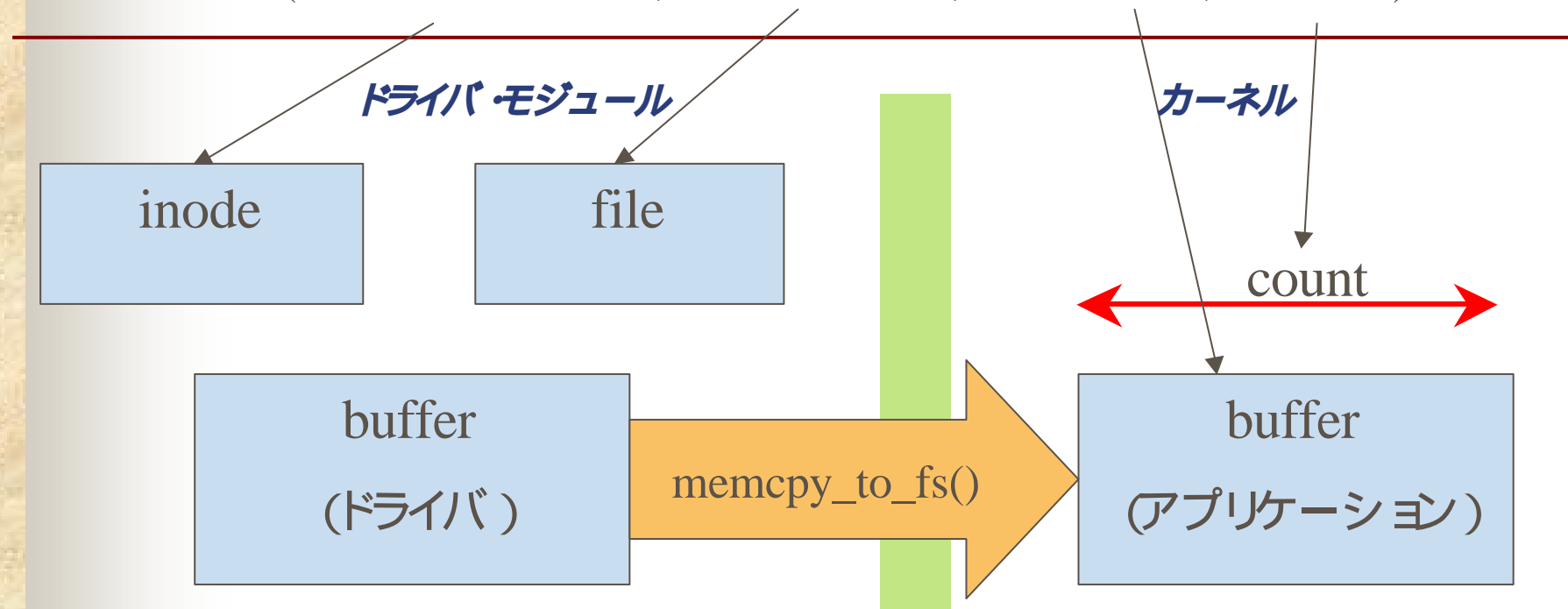
- 入出力 = ファイル、デバイス用
- Linuxのシステムコール
  - メソッド(method)とよび
  - open, read, write, release, ioctl, lseek, ...
- デバイスドライバ
  - メソッドの実体を提供するのがドライバ



# デバイスドライバの仕事

## ■ readのコードの例

```
int read (struct inode *inode, struct file *file, char *buffer, int count)
```



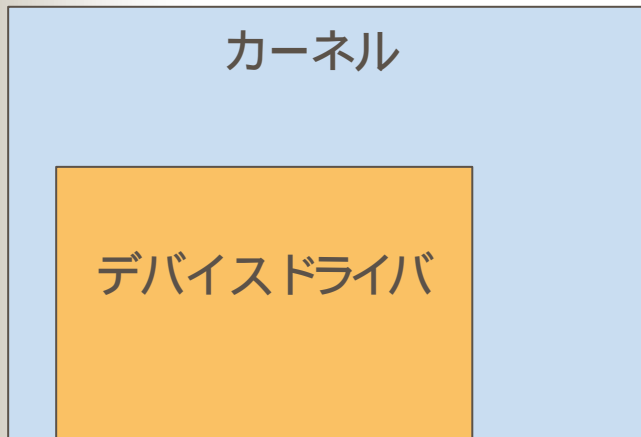
# メジャー番号とマイナー番号

- /dev/ディレクトリの下の特権ファイル (ノード)
  - デバイスノード (= mknod という特別なコマンドで作成)
- メジャー番号がドライバとデバイスを結びつける
  - デバイスドライバの種類 = メジャー番号で決定
- マイナー番号が個々の同一型デバイスを区別
  - デバイスドライバにはマイナー番号がそのまま渡される
- メジャー番号 & マイナー番号 = デバイスとドライバの接続点
- 動的な割り当てと静的な割り当て (固定割り当て)がある

# ロードダブル・モジュール

## スタティックリンクのドライバ

アプリケーション

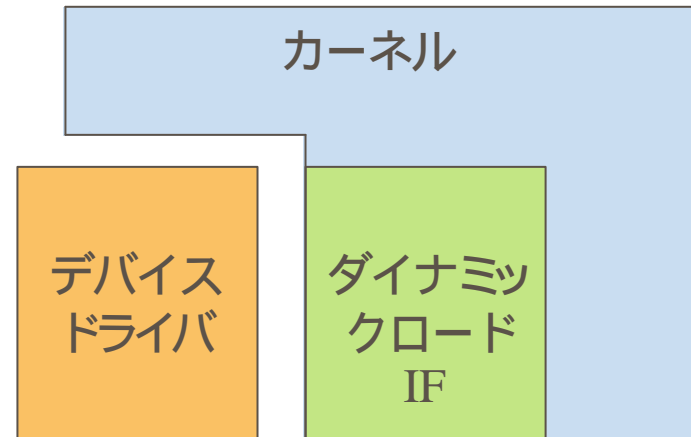


ハードウェア



## ロードダブルモジュールのドライバ

アプリケーション



ハードウェア



# ロードダブル・モジュールの操作

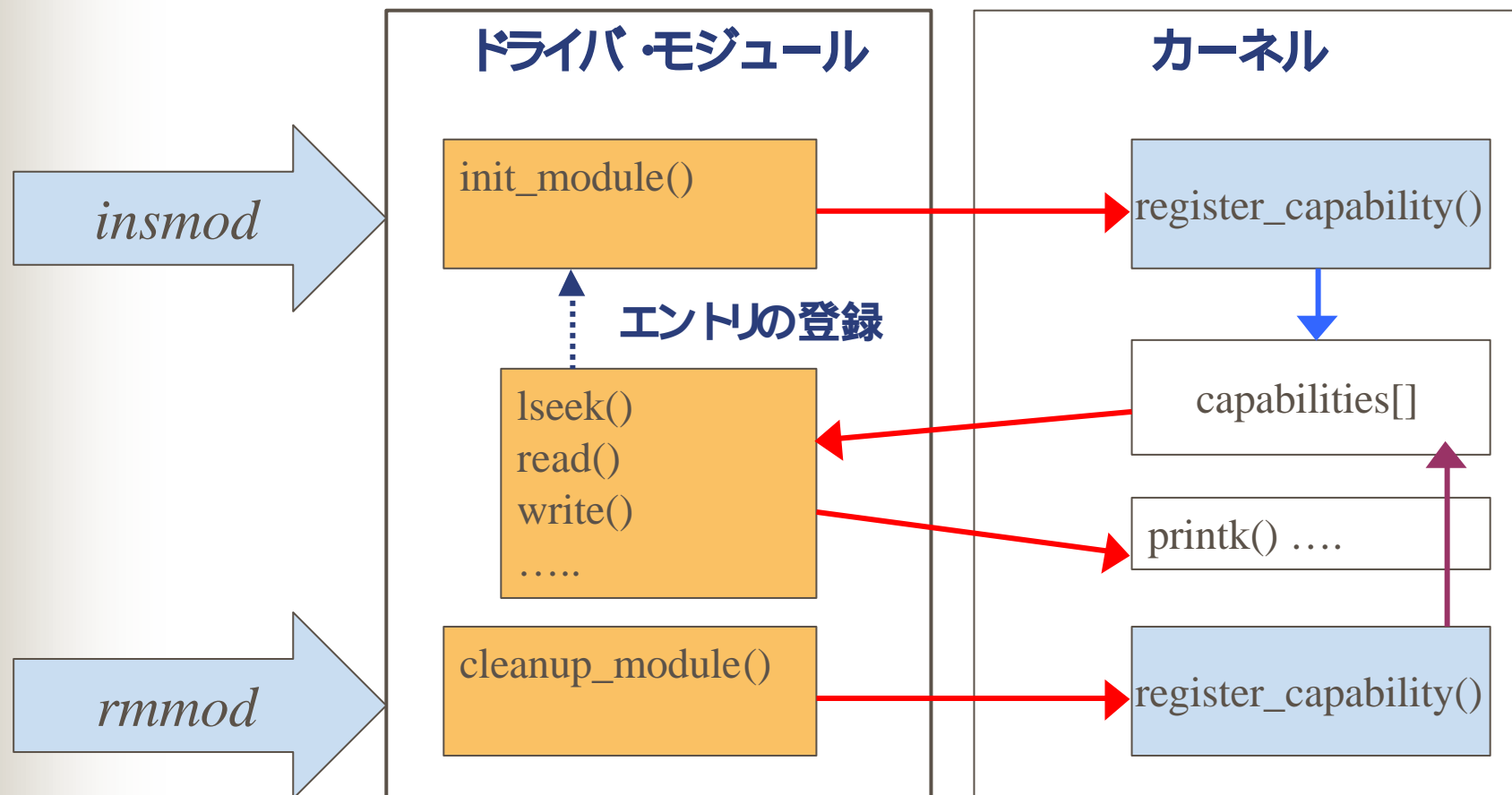
## ■ 自動的なロード

- `/lib/modules/2.4.??-*/kernel/` 以下
- `/etc/modules.conf` の記述

## ■ 手作業でのロード

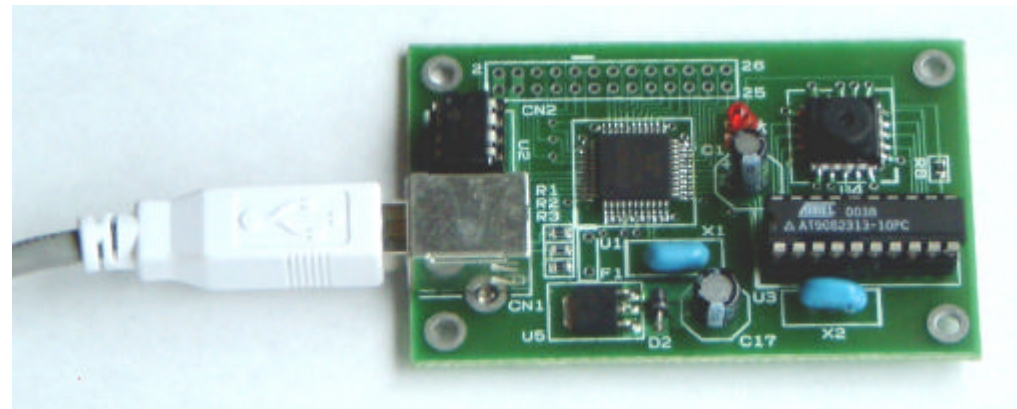
- `insmod aaa.o`
- `rmmmod aaa`
- `modprobe aaa`
- `lsmod`
- `depmod -a`

# ダイナミック・ローディング



# Linuxデバイスドライバ開発

- 準備と心構え
- カーネルとデバイスドライバ
- デバイスドライバ開発
- デバッグツール
- 実機デモ



# バージョン名

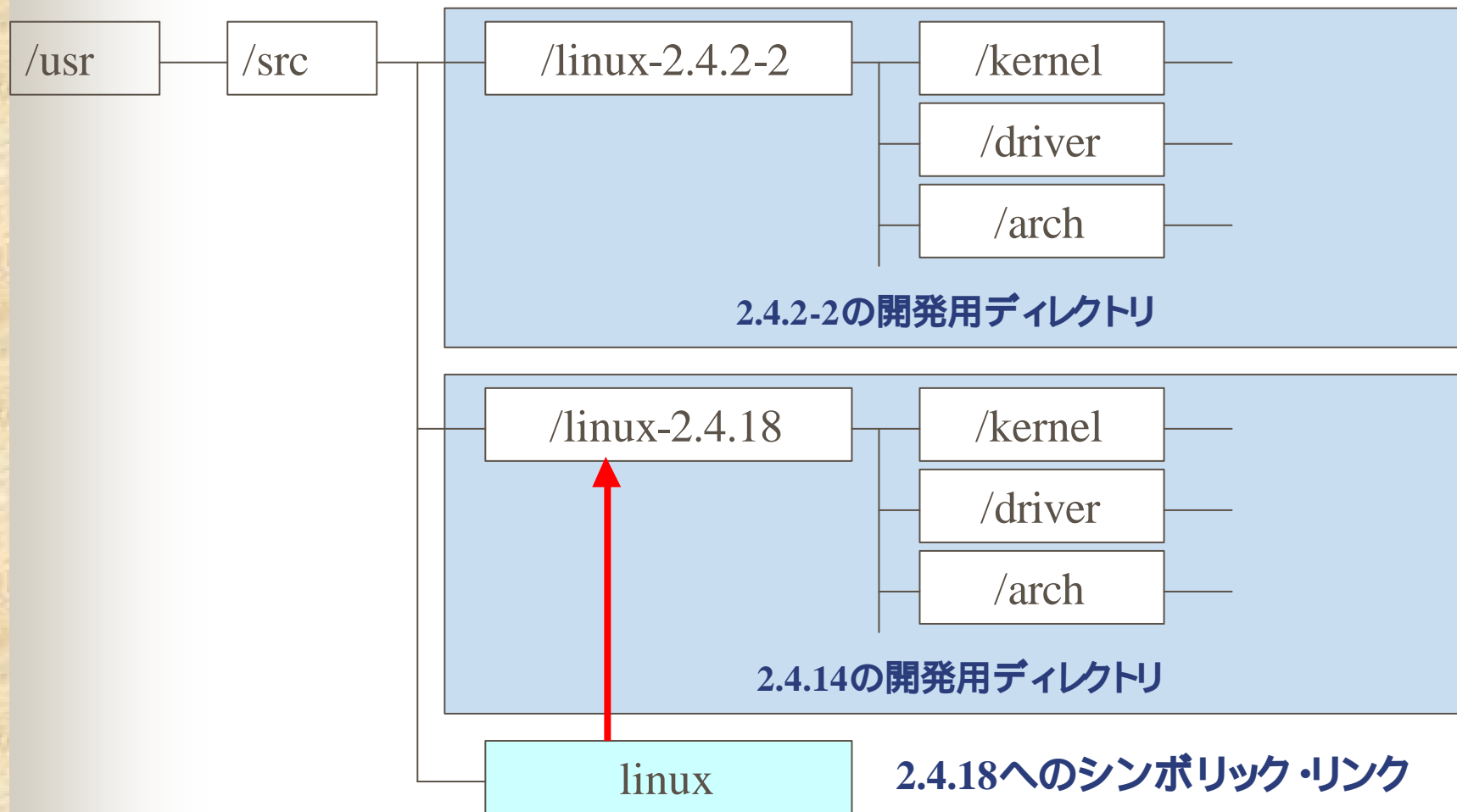
## ■ Makefileの記述例

```
VERSION=2  
PATCH-LEVEL=4  
SUB_LEVEL=14  
EXTRAVERSION=-2
```

## ■ バージョン名管理

ディレクトリ名	ファイル名	Makefile/modules	LILOのラベル
linux-2.4.2-2	vmlinuz-2.4.2-2	2.4.2-2	linux
linux-2.4.18	vmlinuz-2.4.18	2.4.18	linux-18
linux-2.4.18ikd	vmlinuz-2.4.18ikd	2.4.18ikd	ikd-linux

# カーネル開発用ディレクトリ





# カーネル・コンパイル

## ■ 手順とコマンド

- トップディレクトリの必要ファイル確認
- `make mrproper` パラメータや設定ファイルの初期化
- `make *config` コンフィグファイルの作成 & 修正
- `make dep` 依存情報とシンボル情報の生成
- `make clean` 古いオブジェクトの削除
- `make bzImage` カーネルのコンパイル
- `make modules` モジュールのコンパイル
- `make modules_install` モジュールのインストール
- カーネルのコピーとブート設定

# デバイスドライバのコンパイル

- モジュール化
  - make \*configで‘m’を選択
- make modules 以降の作業
- KERNELオプションとMODULEオプション
  - コンパイルスクリプトの例

```
gcc -D__KERNEL__ -I/usr/src/linux-2.4.18/include -Wall -Wstrict-prototypes ¥  
-Wno-trigraphs -O2 -fomit-frame-pointer -fno-strict-aliasing -fno-common ¥  
-pipe -mpreferred-stack-boundary=2 -march=i686 -DMODULE -DMODVERSIONS ¥  
-include /usr/src/linux/include/linux/modversions.h ¥  
-DKBUILD_BASENAME=serial -DEXPORT_SYMTAB -c serial.c
```

# デバイスドライバのロードとテスト

## ■ デバイスノードの作成

- mknod

## ■ コマンド

- insmod          モジュールのロード
- rmmod          モジュールのアンロード
- lsmod          モジュール・ディレクトリの表示
- depmod          モジュール依存関係を作成
- modprobe        モジュールの依存関係を配慮したロード

# デバッグ環境

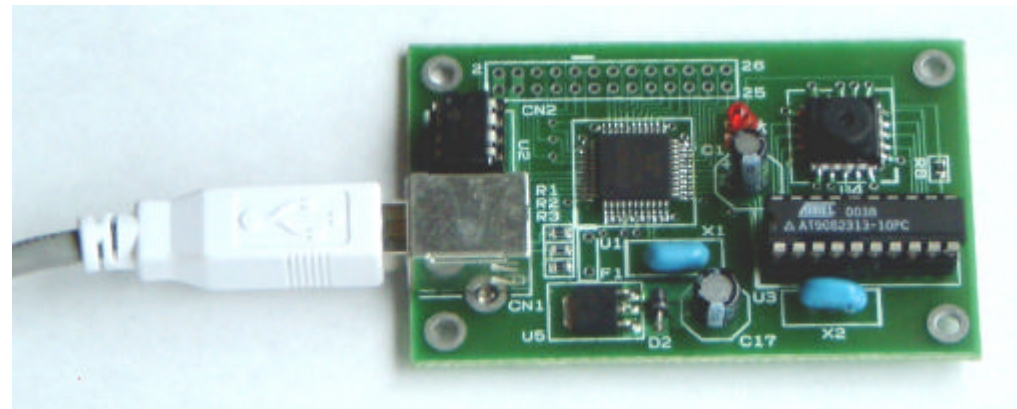
- ターゲットの準備
  - 実機デバイス、類似デバイス
  - シミュレータ、エミュレータ
- アプリケーション
  - エンドユーザ向けアプリケーション
  - デバッグ用アプリケーション
  - テスト用アプリケーション

# 補足情報

- カーネルとサービス・コール (ドライバ・インタフェース) は頻繁に変わる
  - 動作環境の各バージョン確認が必須 (上位互換性が無い)
  - 正確な情報の収集と解析が必要
- ドライバの書き方はUnixと互換性は無い
  - 特にネットワーク、Socketインタフェース

# Linuxデバイスドライバ開発

- 準備と心構え
- カーネルとデバイスドライバ
- デバイスドライバ開発
- デバッグツール
- 実機デモ



# デバッグツール

- gdb – 汎用デバッガ、リモートデバッグ
- printk() – メッセージ
- SysRq – マジック・キー
- /proc ファイルシステム
- kdb – 静的デバッグ
- kdbg – リモートデバッグ
- IKD – 静的オンタイムデータ収集

# printk()

- 標準でも利用されるメッセージ出力
  - float, doubleなどの浮動小数点は表示できない
- 使い方
  - `printk(KERN_ERR "Help = %d¥n", help);`
  - メッセージ・レベル

```
#define KERN_EMERG      "<0>" /* system is unusable */
#define KERN_ALERT     "<1>" /* action must be taken immediately */
#define KERN_CRIT      "<2>" /* critical conditions */
#define KERN_ERR       "<3>" /* error conditions */
#define KERN_WARNING   "<4>" /* warning conditions */
#define KERN_NOTICE    "<5>" /* normal but significant condition */
#define KERN_INFO      "<6>" /* informational */
#define KERN_DEBUG     "<7>" /* debug-level messages */
```



# SysRq

- ‘神秘の’ ‘魔法の’ SysRqキー
- 緊急時やシステム制御のコマンド実行
- カーネルに標準で組み込み済
  - Kernel 2.1.x 以降で利用可能
    - ただしEnableしておけば...
- kdb等でキーがハングしたときに有効

# /proc ファイルシステム

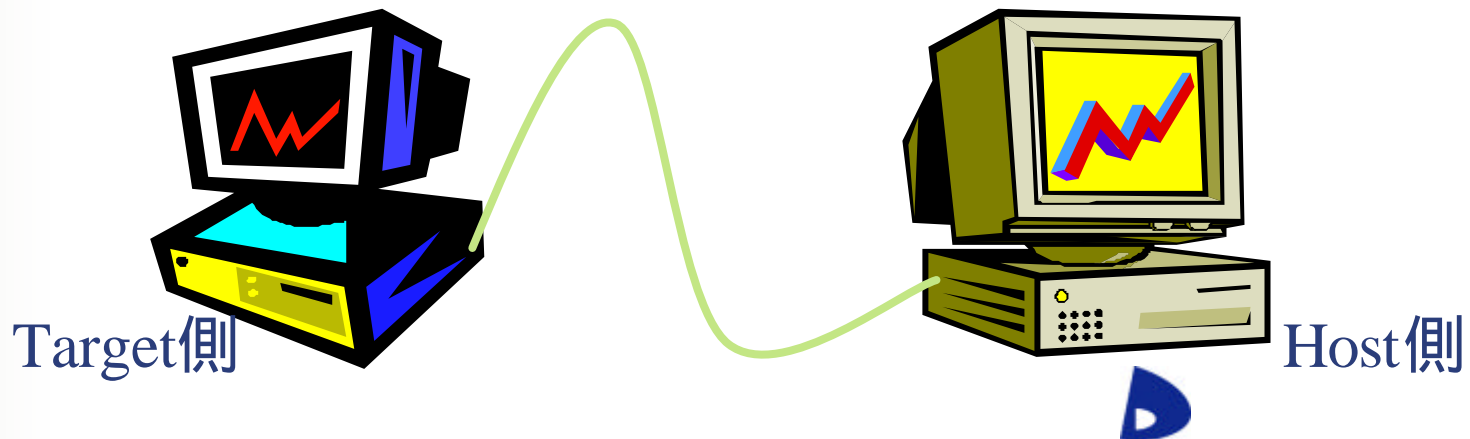
- /proc/ 下の特別なテキスト・ファイル
  - /proc/devices (デバイス割り当てメジャー番号の表示)
  - /proc/ioprots (物理IOポートの割り当て表示)
  - /proc/interrupts (物理IRQの割り当て表示)
- ドライバとユーザのインタフェース
- サンプル・ソース

# 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
- 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
  
- 実機デバッグのデモ

# kgdb (linux kernel source level debugger )

- <http://kgdb.sourceforge.net/>
- gdbベースで2台のPCをシリアルケーブルで接続
- Cソースコードレベル・デバッガ
  - カーネル全体をframe pointer optionで再コンパイルする必要がある



# IKD (Integrated Kernel Debugging Facilities)

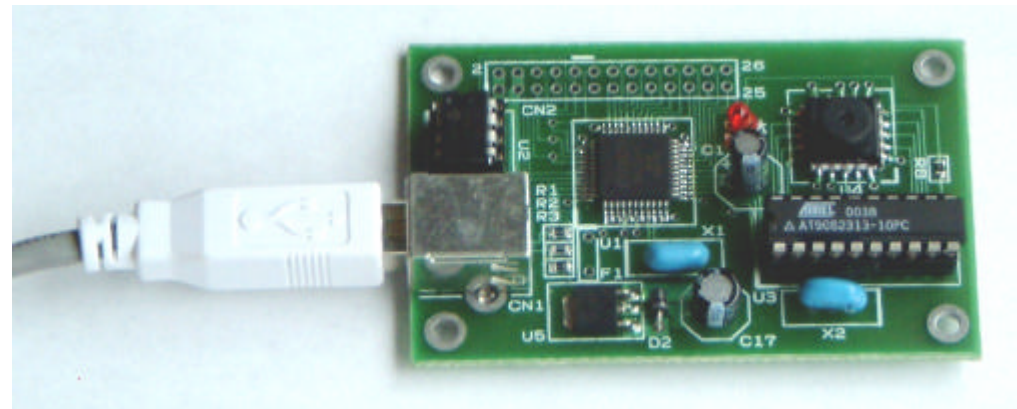
- <http://www.kernel.org/pub/linux/kernel/people/andrea/ikd/>
- パッチ組込みカーネルデバugg  
  - 2.2.12-ikd5 以降のIKD にはkdbが入っている
  - データテーブルにgdbでアクセス可能
  - TraceやMemleak機能等もある
- カーネル本体のデバugg用

# その他のデバッグ・ツール

- lhcd (Linux Kernel Crash Dumps)
  - <http://lkcd.sourceforge.net/>
- lockmeter (Kernel spinlock metering for Linux)
  - <http://oss.sgi.com/projects/lockmeter/>
- KMSGDUMP (Linux Kernel Messages Dump Tool)
  - <http://www-miaif.lip6.fr/willy/pub/linux-patches/>

# Linuxデバイスドライバ開発

- 準備と心構え
- カーネルとデバイスドライバ
- デバイスドライバ開発
- デバッグツール
- 実機デモ



# 実機でのデバッグ例

## ■ デモ用ターゲット

- USBキャプチャ・カメラ EE-260
  - <http://e-kit.jp/> で販売
- OmniVision OV511 + OmniVision OV7620
- [http://www.ovt.com/pdfs/ds\\_511P.pdf](http://www.ovt.com/pdfs/ds_511P.pdf)

## ■ アプリケーション

- W3CAM
  - <http://mpx.freeshell.org/>
- デバッグ用アプリケーション





# デバッグ用アプリケーション例(1)

- Webカメラcgi
  - vidcat
  - w3cam.cgi
    - netscape等のブラウザで確認
- テスト用shell

```
#!/bin/sh -f  
while [ 1 ]; do (./vidcat > /dev/null ; echo capt); done
```

# デバッグ用アプリケーション例(2)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <linux/types.h>
static char *device = "/dev/video";
main()
{
    char b[256];
    int i, fd = 0;
    while(1) {
        b[0] = 0;
        printf("Open, Close, Quit ?> ");
        scanf("%s", &b[0]);
        switch(b[0]) {
            case 'o': case 'O':
                if ((i = open(device, O_RDWR)) < 0) {
                    printf("open error = %d\n", i);
                    break;
                }
                fd = i;
                printf("Opened = %d\n", fd);
                break;
```

```
            case 'c': case 'C':
                if (fd == 3) {
                    printf("not opened = %d\n", fd);
                    break;
                }
                else if ((i = close(fd)) < 0) {
                    printf("close error = %d\n", i);
                    break;
                }
                printf("Closed = %d\n", fd);
                fd = 0;
                break;
            case 'q': case 'Q':
                printf("will quit...\n");
                exit(0);
                break;
            default:
                printf("Invalid command = %s\n", b);
                break;
        }
    }
}
```

# 補足 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
rmmmod	<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