Android におけるデバイスの認識について

2010 年 12 月 4 日
第 4 回 PF 部勉強会

本日の発表内容

自己紹介

Linux でデバイスを認識するには（ **udev** を中心に）

Android でのデバイス認識（ **init.c, devices.c** を中心に）

自己紹介

★氏名： 大野徹

★仕事： しがないサラリーマン（Androidとは基本無関係）
Ap開発が主、SEなどを経て今年は何故か組み込み系の仕事へ
（Androidで勉強したこと役に立ってます）

★ハンドル： @pakuqi
→元々、香菜(xinagcai)というHN使ってましたが、女性に間違えられる事がまれにあったので、タイ語のパクチーに変更

★Blog： 電脳羊（http://xiangcai.at.webry.info/）

★活動：最近名古屋の@magoroku15さんが始められたUST「V7から始めるUNIX講座」に生徒役？としてお手伝い中

Linux でのデバイスの認識

Linuxでデバイスとやりとりするのに必要なもの

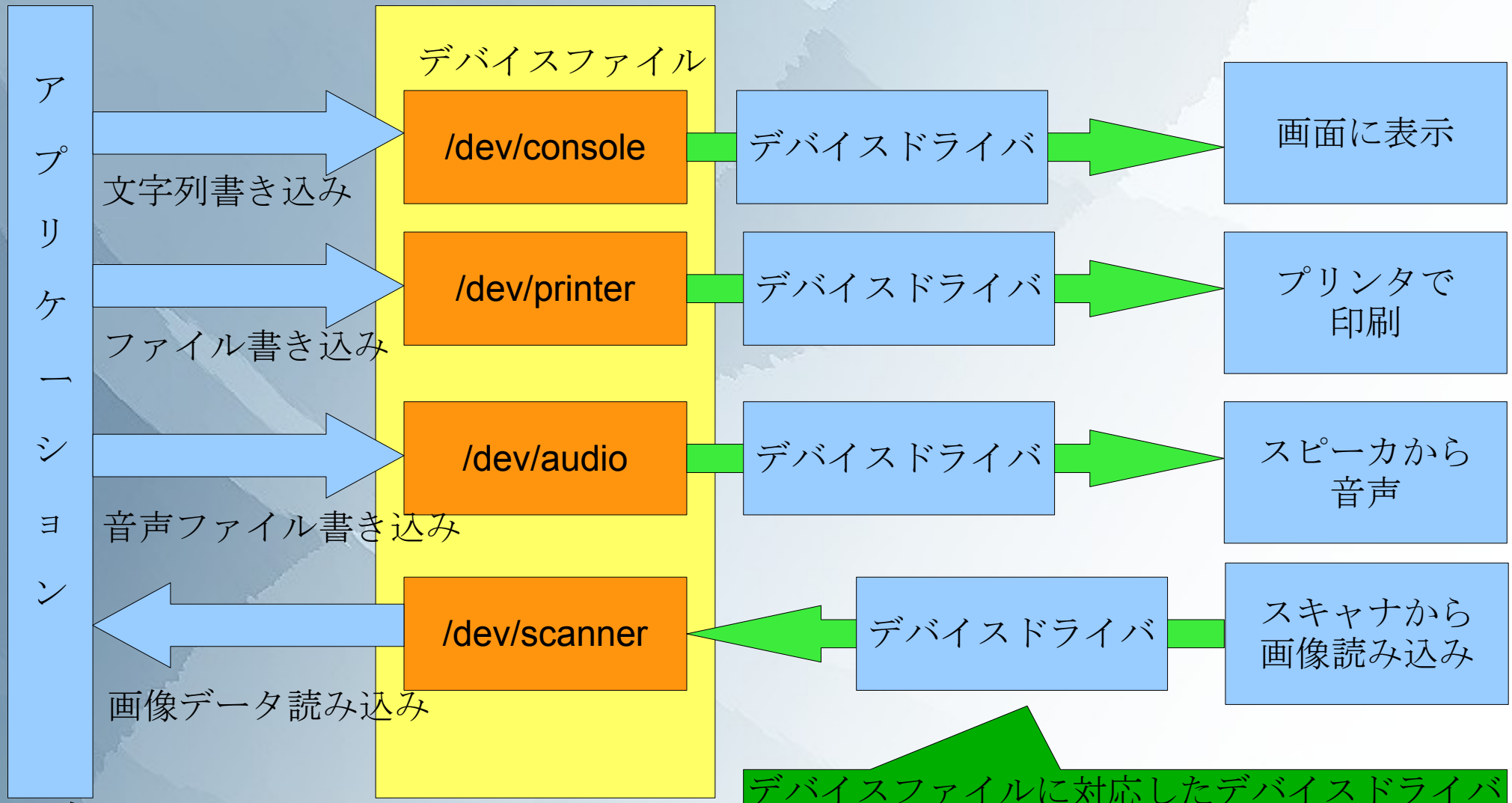
- デバイスドライバ

ハードウエアを制御するためのソフトウエアです。
Windowsなどでもインストールするのでお馴染みかと思います。
Linuxではカーネルに最初から組み込むか、後でモジュールをロードするかします。（後述）

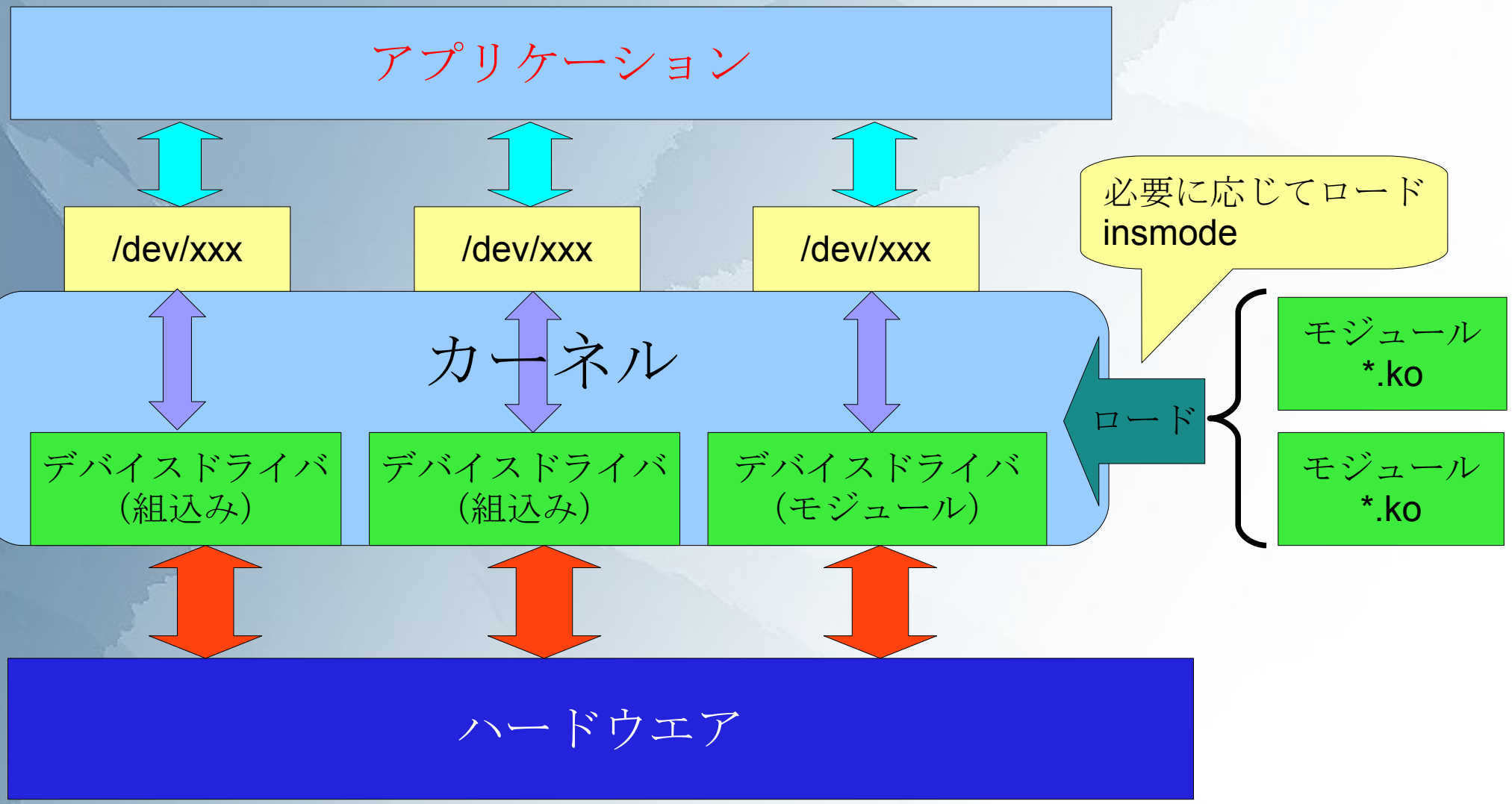
- デバイスファイル

/dev/xxxxというファイルです。
Linuxではデバイスを仮想化しています。
デバイスファイルへの入出力が外部装置への入出力になります。

デバイスの仮想化のイメージ



構成概要図


アプリケーション
/dev/xxx
/dev/xxx
/dev/xxx
必要に応じてロード
insmode
カーネル
ロード
モジュール
*.ko
モジュール
*.ko
デバイスドライバ
(組込み)
デバイスドライバ
(組込み)
デバイスドライバ
(モジュール)
ハードウエア

デバイスファイルをどう生成するか

デバイスドライバは、メーカーが添付したり、サイトからダウンロード出来るようになってたりします。

デバイスファイルはどう生成すれば良いでしょうか？

１、カーネル 2.4 以前

/dev の下に使用する可能性のあるデバイスファイルを予め作成していました。
新たにデバイスを追加した場合に既にデバイスファイルがあれば、そのまま使えます。
もしなければ、メジャー番号（デバイスドライバの識別に使用）、マイナー番号（そのドライバが制御する個々の機器の識別に使う）を調べて手動で作成していました。
→ 不便です。

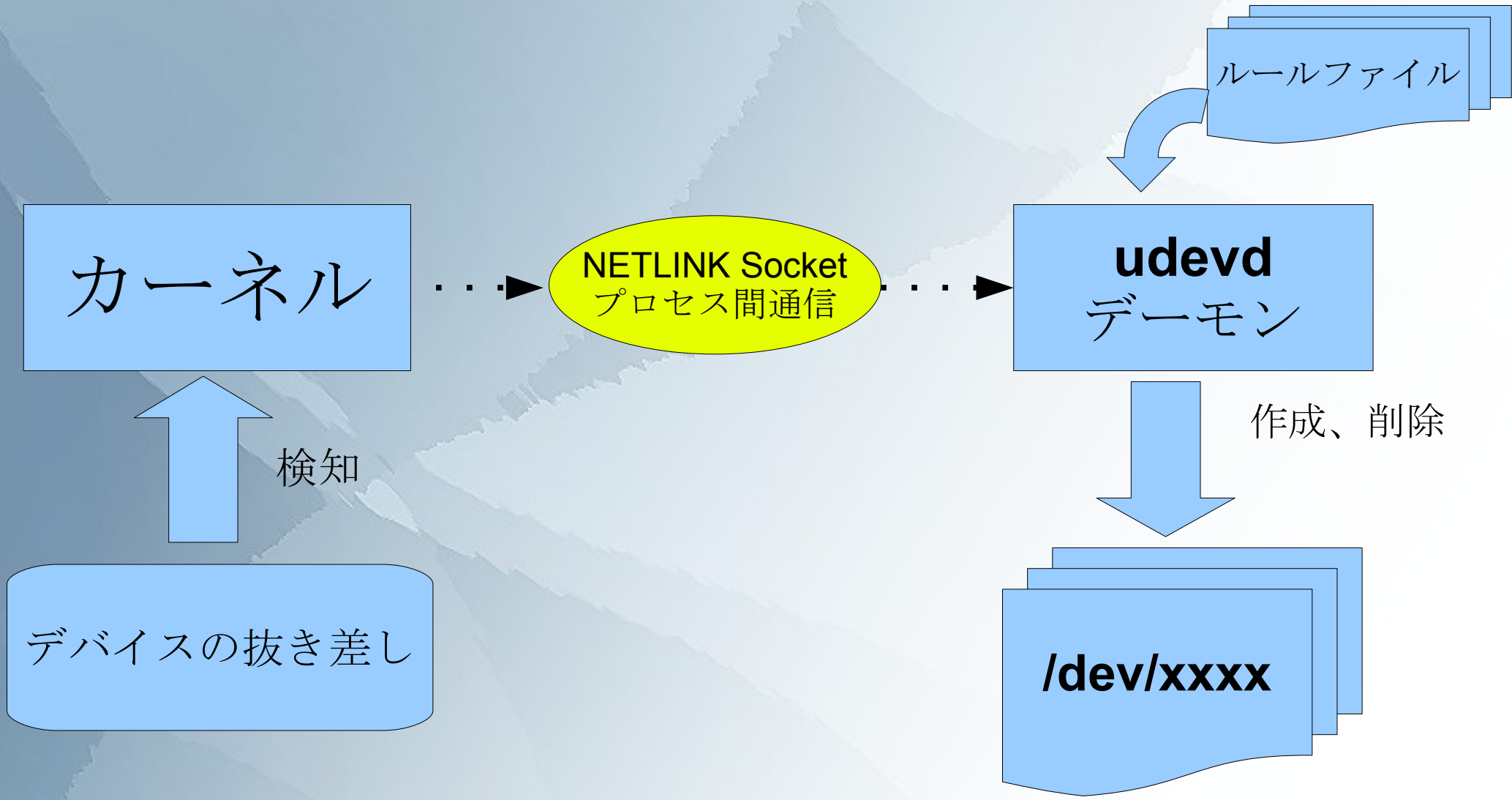
UDEV の仕組み

２、カーネル 2.6 以降

カーネルがデバイスの追加、削除を検知して、 NETLINK Socet を通じてユーザに通知してくれるようになりました。
udev がその通知を受けてデバイスファイルを作成したり削除したりしますので、予め作成や手動での作成が不要になりました。
そのためカーネル 2.6 以降では実在するデバイスのデバイスファイルだけが存在します。
これらを行う udevd というデーモンが存在します。
ubuntu10.04 で確かめてみると

```
$ ps aux | grep udev
root     17187  0.0  0.0  17092   940 ?        S<s  15:14   0:00 udevd
--daemon
root     26605  0.0  0.0  17088   908 ?        S<   21:00   0:00 udevd
--daemon
root     26606  0.0  0.0  17088   904 ?        S<   21:00   0:00 udevd
--daemon
```

UDEV の仕組み


ルールファイル
カーネル
NETLINK Socket
プロセス間通信
udevd
デーモン
作成、削除
検知
デバイスの抜き差し
/dev/xxxx

ルールファイルの作成

udev が自動でデバイスファイルの作成、削除をやってくれるのは良いのですが udev も何も情報がないところからデバイスファイルの作成・削除は出来ません。

ルールを書いて udev に教える必要があります。
/etc/udev/rules.d ディレクトリにルールファイルを作成します。

といっても Ubuntu で Android アプリ開発してる人はルールファイル書いたことがあるはずです。
http://developer.android.com/guide/developing/device.html

スマートフォンを認識するのに、 /etc/udev/rules.d/51-android.rules に
SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
と記述します。

ベンダー **ID**

Android でのデバイスの認識

Android では？

ここから、 Android に関するお話です。
Android も Linux カーネルを使っているので、ほぼ同じです。

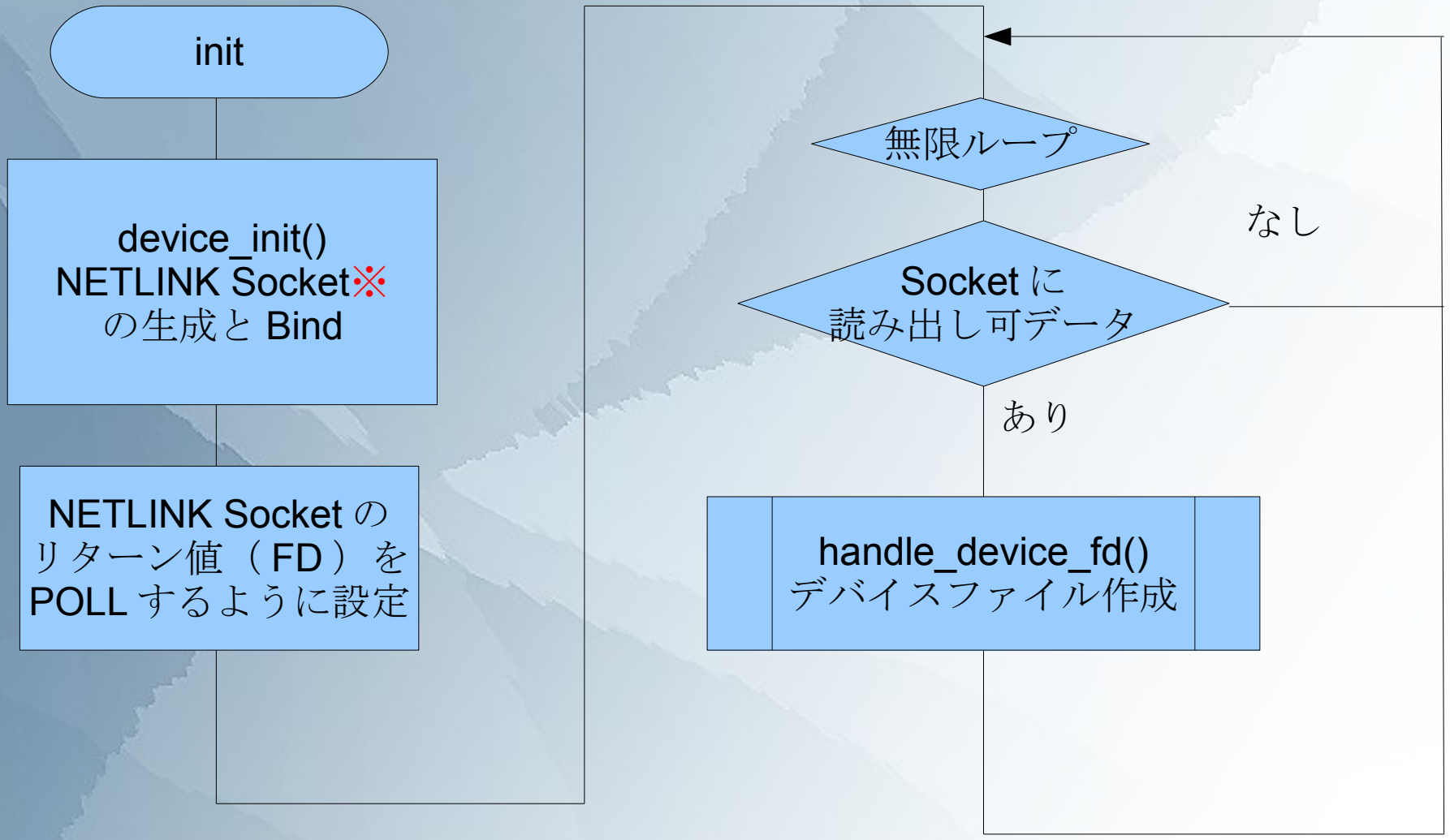
１、デバイスドライバ
Linux と同じようにカーネルに組み込むかモジュールをロードします。
モジュールをロードする時は init.rc に記載します。

２、デバイスファイルの作成
手を入れています。
まず、 **udevd** がありません。

前回、 **Init.c** の中をざっと見たのですが **Init** の中で処理しています。
今回は、デバイスファイルの作成を中心に再度見直したいと思います。
init.c と devices.c を見ていきます。

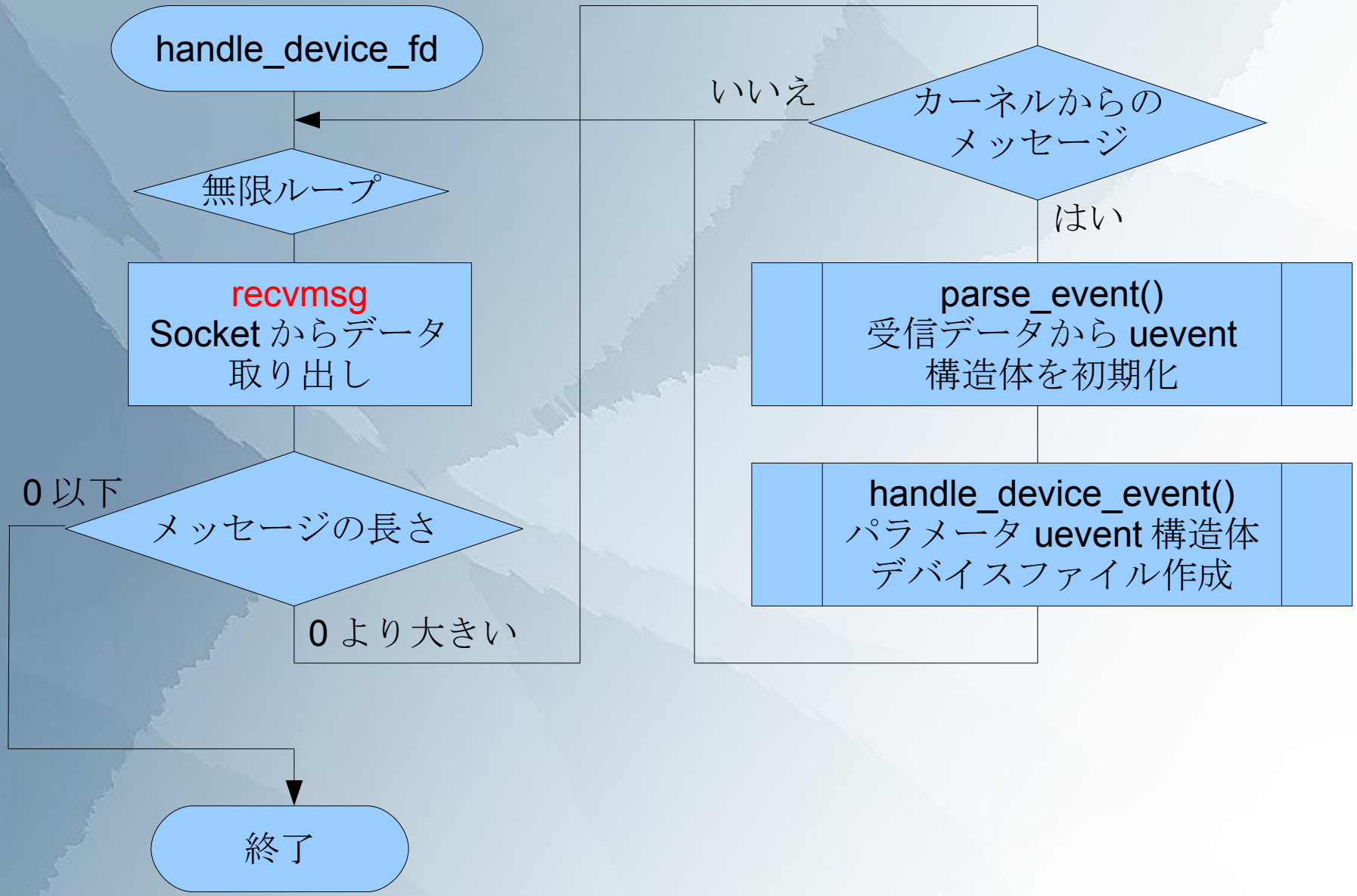
Init.c デバイス周りのフロー図

関連するところのみピックアップ

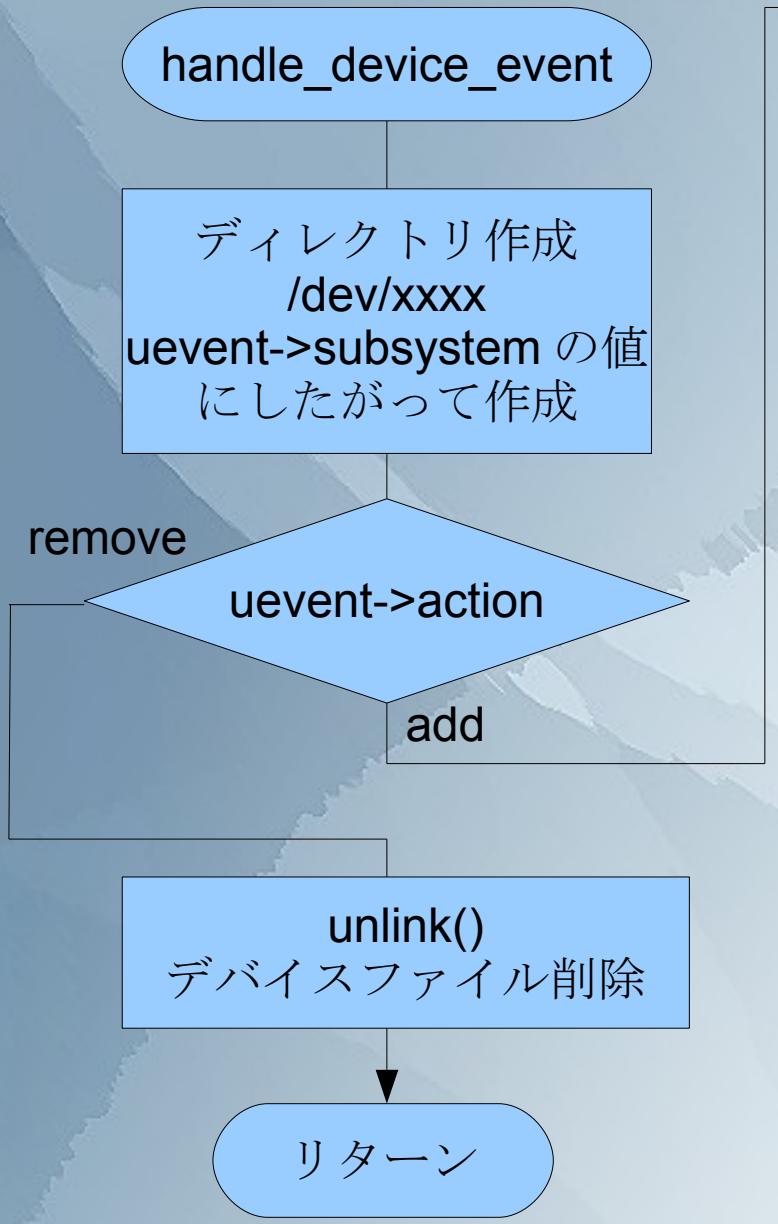


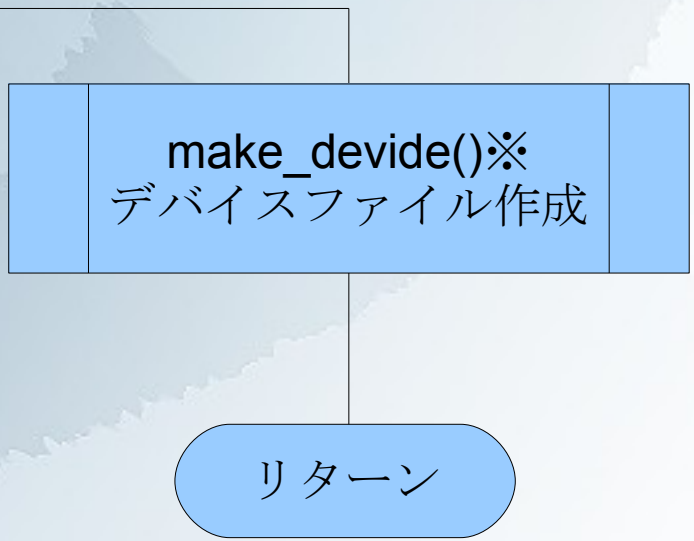
※ カーネルモジュールとユーザー空間のプロセス間で
情報をやりとり するために用いられる

Init.c デバイス周りのフロー図



Init.c デバイス周りのフロー図





※
この関関数では、 mknod コマンドでデバイスファイルの作成と chown コマンドでオーナーを変更しています
→ カーネル 2.4 以前で手動でやっていた処理

NETLINK Socket 詳細

```
struct sockaddr_nl addr;
addr.nl_family = AF_NETLINK;
addr.nl_pid = getpid();          //init プロセスのプロセス ID
addr.nl_groups = 0xffffffff;     // 全てのイベントを通知

s = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_KOBJECT_UEVENT);

if(bind(s, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
```

Socket のパラメータ
PF_NETLINK ：カーネル・ユーザ・デバイス
SOCK_DGRAM ：データグラム (信頼性無し、固定最大長メッセージ) コネクションレス型。
NETLINK_KOBJECT_UEVENT ：ユーザー空間へのカーネルメッセージ

sockaddr_nl 構造体
nl_family ： AF_NETLINK を指定
nl_pid ： Socket を生成したプロセスのプロセス ID
nl_groups ： netlink group number のビットマスク。 32 種類の multicast groups がある。
→ 全ビットを立てているので全てのイベントを受信することになる。
multicast groups は、
include/linux/rtnetlink.h に RTMGRP_XXX という名前で定義している。

poll による Socket の監視

Init は初期化が終わると最後で無限ループに入ります。
無限ループの中で、作成した NETLINK socket に対して読み込むことの出来るデータがないかのチェックをしています。

```
ufds[0].fd = device_fd;          //NETLINK Socket の FD をチェックする
ufds[0].events = POLLIN;     // 読み出し可能なデータがあるかをチェックする

// 無限ループ
For(;;) {
    nr = poll(ufds, fd_count, timeout);
    if (nr <= 0)
        continue;

    if (ufds[0].revents == POLLIN) // 読み出し可能なデータがあれば関数呼び出し
        handle_device_fd(device_fd);

}
```

メッセージの判別

NETLINK socket からメッセージを recvmsg で取り出して、カーネルから送られた netlink mutlicat message かの判定を行っています。

```
struct sockaddr_nl snl;
struct msghdr hdr = {&snl, sizeof(snl), &iov, 1, cred_msg, sizeof(cred_msg), 0};

ssize_t n = recvmsg(fd, &hdr, 0);
if (n <= 0) {
    break;
}

if ((snl.nl_groups != 1) || (snl.nl_pid != 0)) {
    /* ignoring non-kernel netlink multicast message */
    continue;
}
```

送信元のプロセスのプロセス ID が 0 （カーネル）かチェック

Netlink multicast message か判定※

※ 思いっきりマジックナンバーで書いてますが、 RTMGRP_LINK かをチェックしていると思われます。

```
#define RTMGRP_LINK     1
```

まとめと課題

Android でのデバイスの認識のためには

１、デバイスドライバ
従来の Linux でカーネルに組み込んだり、 insmode でモジュールを後からロードしたりと同じ。
モジュールをロードするのであれば、 init.rc に記載します。

２、デバイスファイルの作成、削除
Linux の udev に相当することを init の無限ループの中でやっていることが分かりました。 Android には udevd は常駐していない。

NETLINK Socket へのメッセージ送信を POLL でチェックしてカーネルから送信された Netlink multicast message であればメッセージを解析してデバイスファイルの作成（ mknod ）、削除 (unlink) を実行します。

課題

Linux では udev に対してルールファイルを準備することで、追加デバイスに対して自動的にデバイスファイルを作成するようになっていましたが、 Android はどうなんだろう？
疑問には思ったのですが、調べ切れませんでした。

そもそもカーネルってどうやってハードウエアの抜き差しを検知しているんだろう？

また時間が取れれば調べてみたいと思います。

参考文献

UNIX USER 2004 年 4 月号「Linux はハードウェアをどう認識するのか？」より転載
http://www.itmedia.co.jp/enterprise/0405/18/epn06.html

いますぐ実践 ! Linux システム管理 No114 udev の仕組みを理解する
http://www.usupi.org/sysad/114.html

udev のルールの書き方
http://mux03.panda64.net/docs/udevrules_ja.html#basics

NETLINK ソケットに関して
http://kazmax.zpp.jp/cmd/n/netlink.7.html
http://www.gadgety.net/shin/tips/unix/ipc/socket.html

POLL に関して
http://www.linuxcertif.com/man/2/poll/97904/