オペレーティングシステム2005
デバイス管理 (1)

2005年12月8日

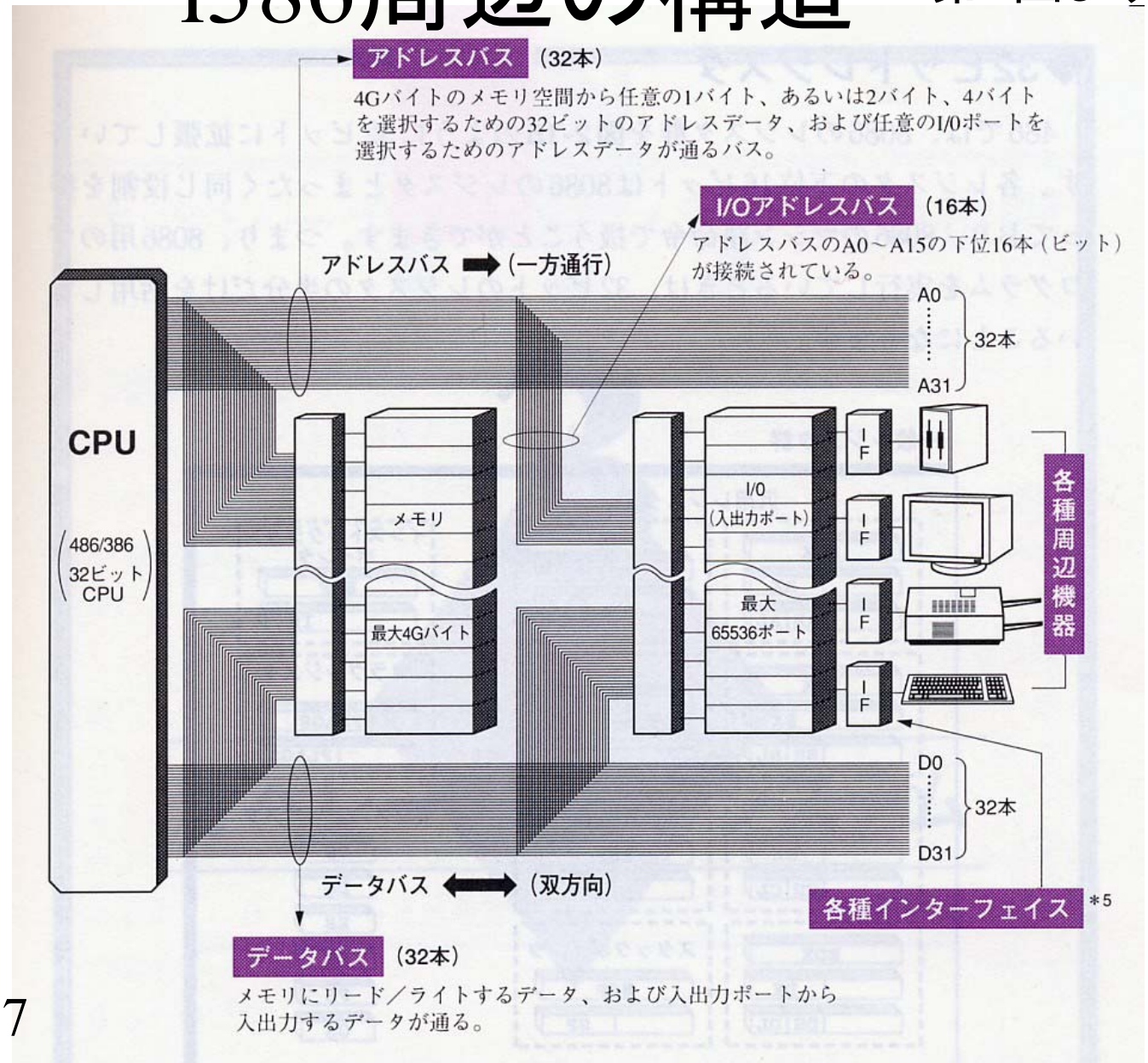
海谷 治彦

目次

- i386におけるデバイス
- Linuxにおけるデバイスの抽象化
- Linuxから見たデバイスの分類
- デバイスドライバ

i386周辺の構造

第3回より再録



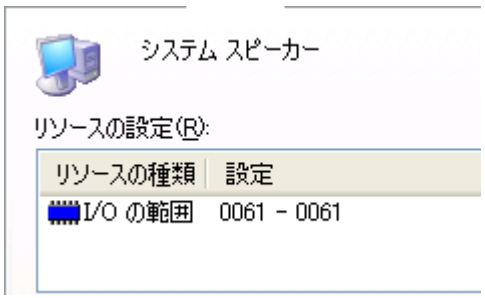
文献6 p.57

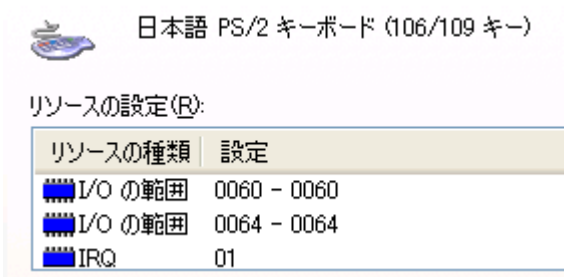
I/Oポートを使った入出力

- CPUからは，I/Oポートというメモリのようなモノにデータを置いたり読んだりすることで，機器(ハード)にデータを送ることができる.
- i386の場合，65536個(2^{16}個)までのアドレスがI/Oポートにふられており，このアドレスで機器を区別する.

例

システムスピーカは0x0061のI/Oポートに接続されている.
⇒ ここになんかデータをおけばスピーカーが鳴る.

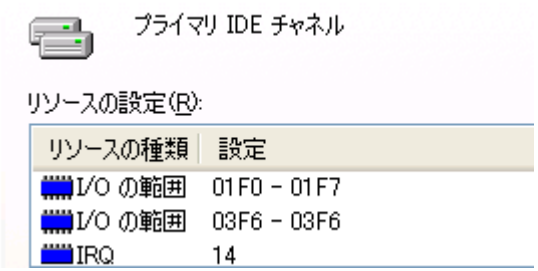


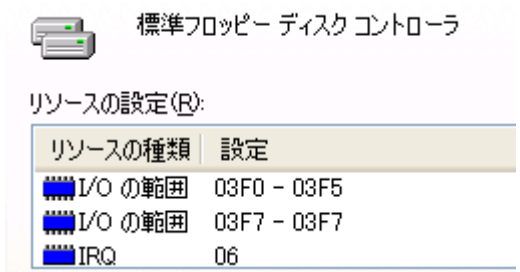


キーボードは0x0060と0x0064のI/Oポートに接続されている.
⇒ これは制御に使われる？(未確認・後述)

例 ディスク関係

いわゆるIDEのディスク
(昨今，もっとも一般的なハードディスク)



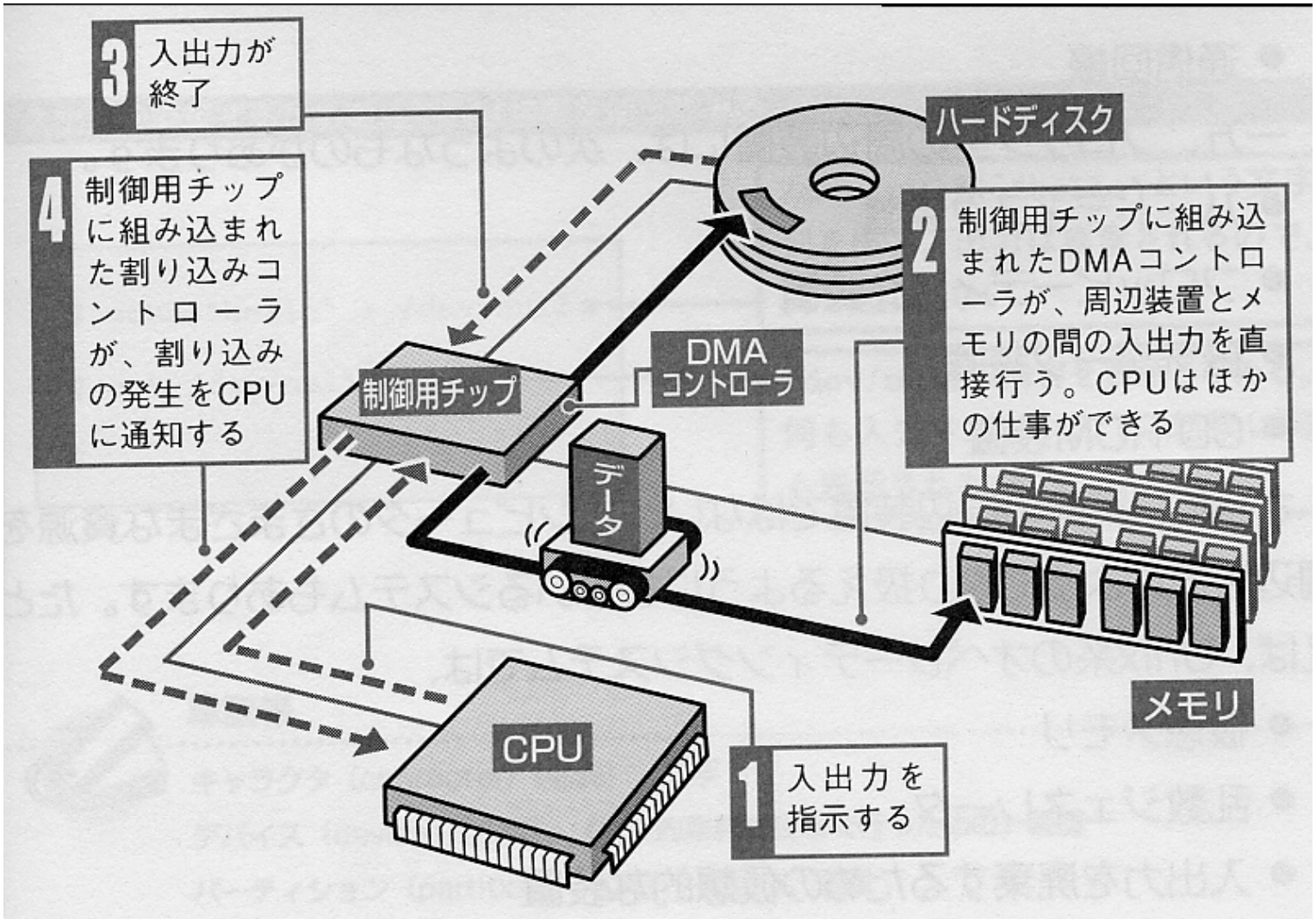


右はフロッピーディスクの制御用のI/Oポート

DMA その必要性

- I/Oポートを使って，CPUが直接データを1個，もしくは数個づつ読んでいては，遅くてラチがあかない.
- そこで，I/Oポートは，「読み出し開始」等の制御情報を送るのに使って，機器とメモリとのデータの行き来はCPUとは独立して行うのがよい.
- Direct Memory Access の略.

DMAの概念図


3 入出力が終了
4 制御用チップに組み込まれた割り込みコントローラが、割り込みの発生をCPUに通知する
ハードディスク
2 制御用チップに組み込まれたDMAコントローラが、周辺装置とメモリの間の入出力を直接行う。CPUはほかの仕事ができる
制御用チップ
DMA コントローラ
データ
メモリ
CPU
1 入出力を指示する

割り込み その必要性

- 前述のDMAでデータ転送が終わったら，そのことをCPUに通知しないといけない．
- そこで，ハードウェア機器からCPU側に非同期に情報を通知する機構が必要．
- このことを「割り込み」と呼んでいる．

割り込みの定義

- 「プロセッサ(CPU)が実行をする命令を変更するイベント」
- 同期割り込み
  - CPUの制御回路が発生するイベント
  - i386では「例外」と呼ばれる
- 非同期割り込み
  - CPU以外のハードが発生するイベント
  - i386では「割り込み」と呼ばれる.

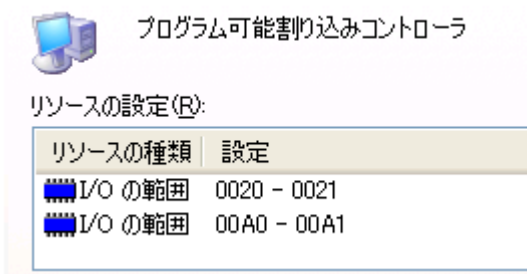
文献5 p.115より

例外・割り込みの分類・特徴

- 例外
  - CPUが異常等を検出した際のイベント (i386では識別子 0～31の一部)
  - ソフトウェア割り込み: 典型的な例は，システムコールによりカーネルコードに実行が移る際.
- 割り込み
  - マスク可能割り込み
    - ソフトウェア的な設定で割り込みを禁止できるもの. (32～47)
  - マスク不能割り込み
    - 禁止できないもの. ハードウェアの故障等に対応する. (i386では識別子の0～31の一部)

IRQとは

- i386のマスク可能割り込みを受け取る信号線.
- それぞれハードウェア機器に関連付けられている.
- それぞれのハードがCPUを使いたい場合に, IRQに対応した割り込みが発生する.
  - 例: キーボードが打たれるとIRQ1番による割り込みがCPUにかかる.
- (前述のように)16個しかない.
  - 左記のI/Oポートにどの割り込みが起きたかが書き込まれている.



Interrupt ReQuest

典型的なIRQ

IRQ番号	用途	システム予約
0	システムタイマー	X
1	キーボード	X
2	IRQ9より呼び出し	X
3	シリアルポート（COM2）	
4	シリアルポート（COM1）	
5	パラレルポート(LPT2)	
6	フロッピーディスクドライブ	
7	パラレルポート(LPT1)	
8	リアルタイムクロック	X
9	未使用（IRQ2へ転送）	
10	未使用	
11	未使用	
12	PS/2マウス	
13	FPU	X
14	プライマリIDE	
15	セカンダリIDE	

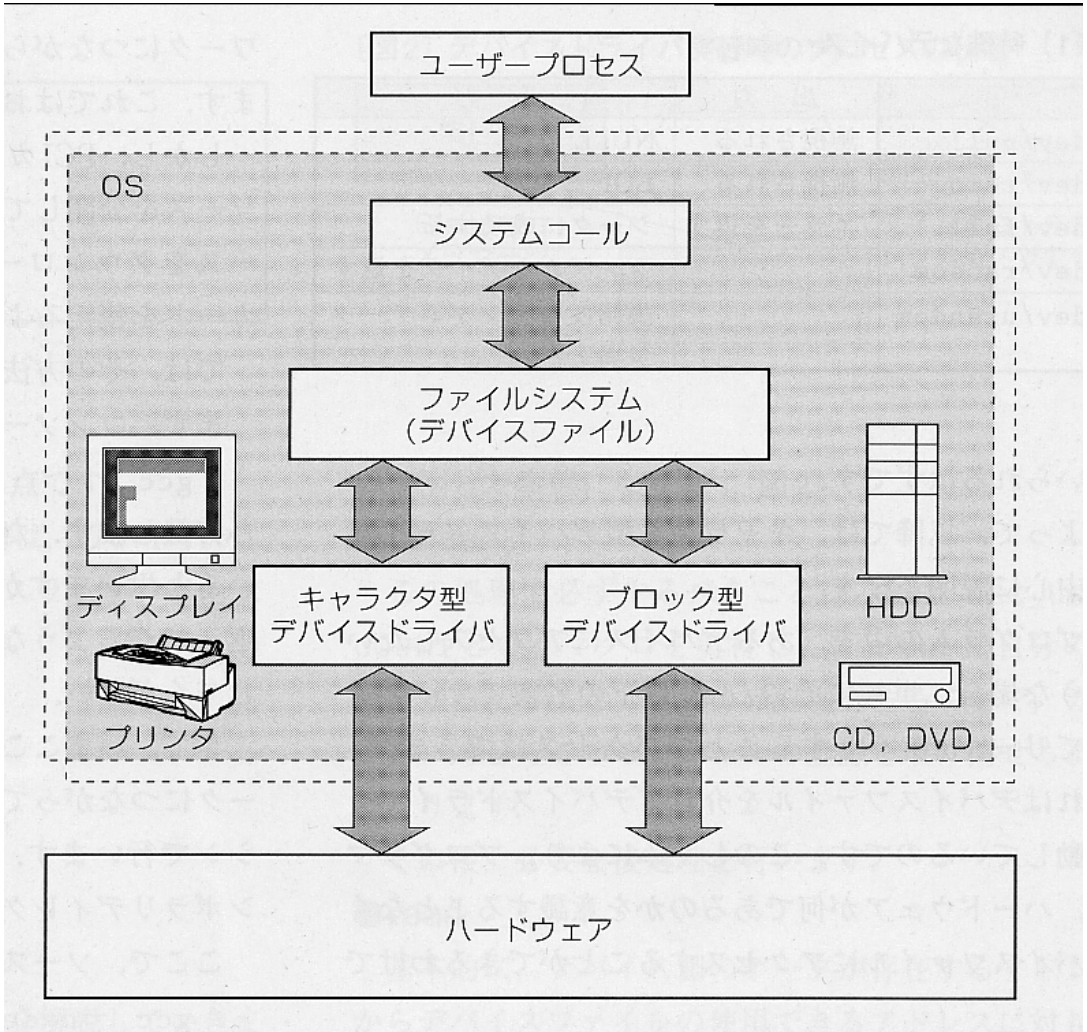
上記は典型例であり，特定の機器を外して，他の機器を割り当てることも少なく無い．

デバイスの抽象化 動機

- 前述のように，ポートだのアドレスだの一々叩いて機器をいじっていたのでは大変.
- 機器によって，実際，操作法は違うのだが，新しい機器(たとえばUSB2のディスクとか)が増える毎にプログラムを作り直さないといけないとラチがあかない.

⇒ 機器をほぼ全てファイルに見立て，ファイルの開閉，読書として機器をいじろう！

デバイスの抽象化


ユーザープロセス
OS
システムコール
ファイルシステム
(デバイスファイル)
ディスプレイ
キャラクタ型
デバイスドライバ
ブロック型
デバイスドライバ
HDD
プリンタ
CD, DVD
ハードウェア

デバイスファイルの例

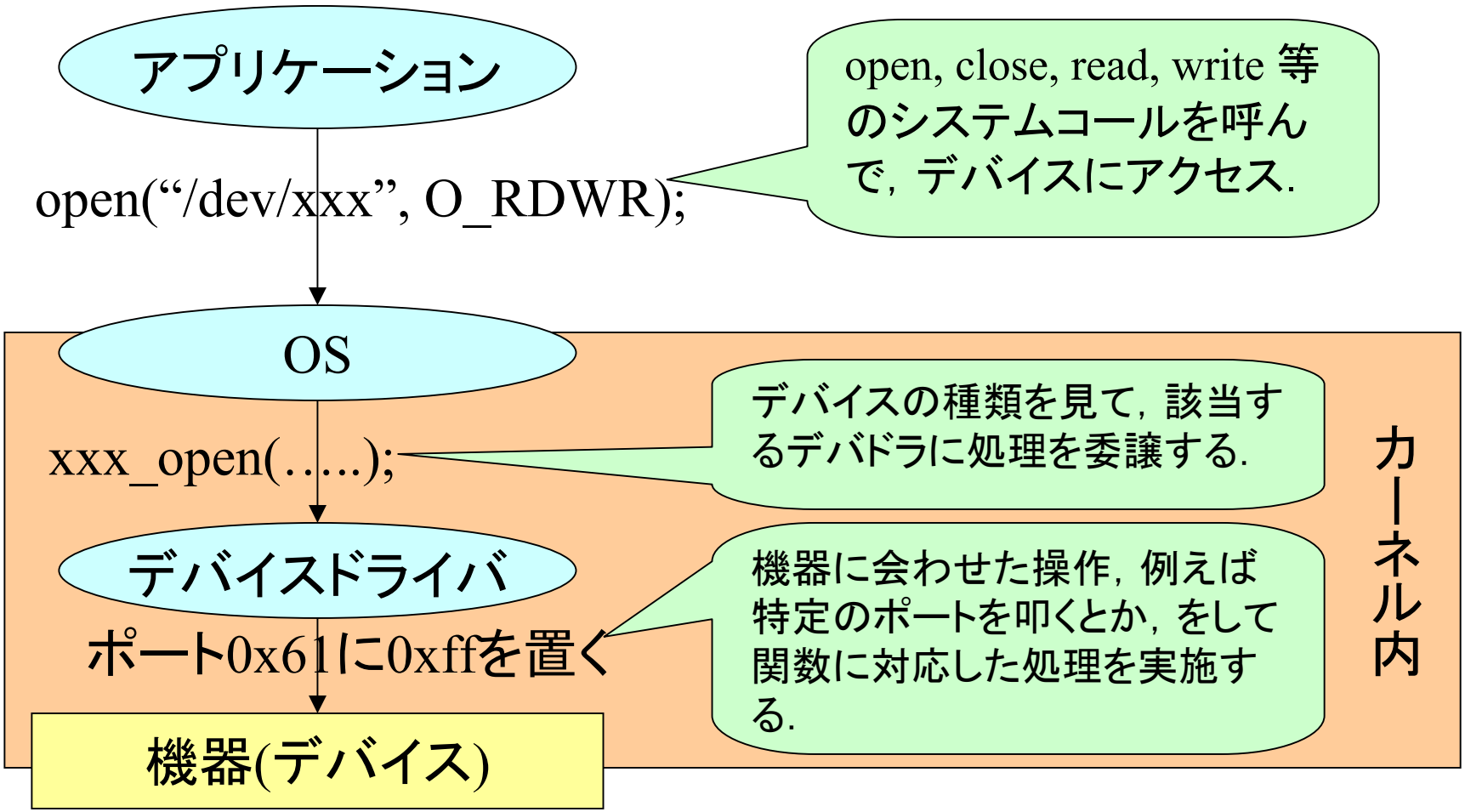
/dev/ ディレクトリの下にある.

- `/dev/hda` IDEハードディスクその一
- `/dev/hda5` hda のパーティションその5
- `/dev/fd0` フロッピーディスク
- `/dev/psaux` PS/2マウス

デバイスドライバとは？

- 機器をファイルのように，open, close, read, write 等の処理が扱えるような仲介をしてくれる関数群.
- 通常はカーネルに組み込まれる.
  - 機器に直接アクセスできるのはカーネルだけのため.
- 新しいハードウェア機器が開発される毎にデバイスドライバもそれにあわせて作らなければならない.
  - しかし，名の通った機器(USBとかIEEE1294とか)のデバドラは大抵，すでに誰かが作成済である.
  - しかも，Linuxの世界では大抵それが無償で使える.

アプリからデバスまで


アプリケーション
open("/dev/xxx", O_RDWR);
open, close, read, write 等のシステムコールを呼んで，デバイスにアクセス.
OS
xxx_open(……);
デバイスの種類を見て，該当するデバドラに処理を委譲する.
デバイスドライバ
ポート0x61に0xffを置く
機器に会わせた操作，例えば特定のポートを叩くとか，をして関数に対応した処理を実施する.
機器(デバイス)
カーネル内

実際に利用可能なデバドラを見る

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  7 vcs
 10 misc
 29 fb
 36 netlink
128 ptm
136 pts
162 raw
180 usb
254 wildio
```

/proc/devices ファイルに列挙されている.
それぞれがある種類のデバイス用(共用もある.)
先頭にある番号で識別されている.
2種類の種別がある.

```
Block devices:
  1 ramdisk
  2 fd
  3 ide0
  9 md
 22 ide1
```

デバイスの種類

- キャラクタ型
  - 一度のI/Oで任意のデータが転送可能な機器.
  - キーボードとかネットワークとか.
  - 通常，順次アクセスしかできない.
- ブロック型
  - 一度のI/Oで固定長のデータブロックを転送可能な機器.
  - ディスク一般.
  - ブロック内でランダムアクセスが可能.

デバイスの種類に関する補足

- キャラ，ブロックに分けるのは歴史的背景からという話もある.
- それぞれに適用できるシステムコールの種類は異なる.
- もう一つの型「ネットワーク型」というのがあるが本講義では扱わない.
- 一般にブロック型のほうが複雑．本講義では以降，基本的にキャラクタ型のみを扱う.

個々のデバイスの認識

- 前述のように，個々のデバイスファイル /dev/なんとか は，OS内からは個々のデバイスそのものである.
- それぞれのデバイスが，何型でどのドライバを使っているかは，ls –l 等でデバイスファイルの属性を見れば，すぐにわかる.

例

```
# cat /proc/devices
Character devices:
 10 misc
Block devices:
   3 ide0
```

抜粋

3番のデバイスドライバを使用,
このドライバは共用されている.
メジャー番号

```
# ls -l /dev/hda1
brw-rw----    1 root disk   3,    1 Oct   5   2000 /dev/hda1
# ls -l /dev/hdb1
brw-rw----    1 root disk   3,   65 Oct   5   2000 /dev/hdb1
# ls -l /dev/psaux
crw-rw----    1 root root  10,    1 Oct   5   2000 /dev/psaux
```

10番のデバイスドライバを使用

同じドライバを使う異なるデバイスは番号をつけて管理する.
マイナー番号

通常ファイルではサイズを表示

```
-rw-r--r-- 1 root root   52 Apr 11   2002 /etc/resolv.conf
```

ファイルに対するシステムコール

- open ファイルを開ける
- close 閉じる
- read ファイルからデータを読む.
- write ファイルにデータを書き込む.
- ioctl デバイスの属性を変更したり，制御したりするのに使用する.
  - 例えば，通信ポートのスピードを変更するなど.

これらはアプリケーションが利用する関数群である.

カーネルに特定機器を登録する

要はデバイスドライバを作るということである.

- ファイル管理の所で話した struct file_operations のメンバーに相当する関数がデバイスドライバの実体である.
  - 全てのメンバーを定義する必要は無い.
- 上記の構造体を使って，新たに作成したドライバをカーネルに組み込む関数がLinuxには準備されている.
  - モジュールの登録

構造体 file_operations

```
struct file_operations { // Kernel 2.4版
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    ・・・ 以下省略
};
```

include/linux/fs.h
の810行目くらい

全てのファイルシステムに全ての関数が定義されているわけではない.