オペレーティングシステム

2022/11/1

海谷 治彦

目次

- 入出力の制御

- 入出力装置
  - DMAとチャネル
- 入出力要求とその制御
  - UNIX流のデバイス管理について
- 入出力の効率化

CPU, メモリ, I/O機器の速度差

圧倒的な速度差がある.

- 例えば, 1回の読み書きするのに,
  - 数ナノ秒 (10^{-9}) CPU(内のレジスタ)
  - 数十ナノ秒 (10^{-8}) メモリ
  - 数十ミリ秒 (10^{-3}) ディスク (SSDだと数倍)

越えられない壁

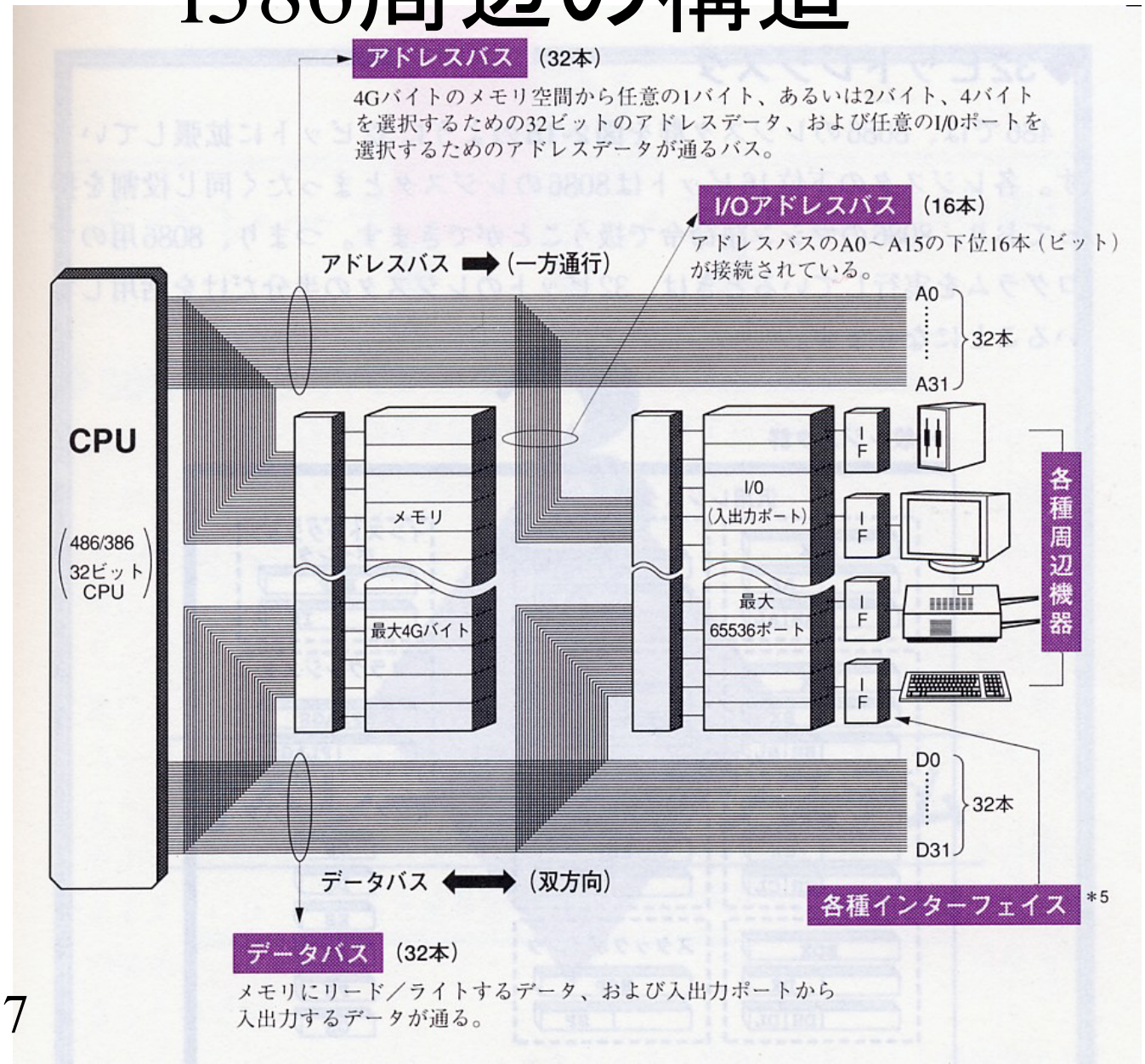
- 上記のように ディスクとメモリは $10^{5\sim6}$ くらいの速度差がある.
  - もちろん人間はもっと遅いよ.

CPUと装置の接続技術

- 入出力制御方式
  - CPUが入出力装置を制御するチップに命令を出す.
- DMA
  - Direct Memory Access
  - CPUを通さずデータをやりとりする.
  - 割り込みがまた重要な役目を果たす.
- 入出力チャネル
  - 大型コンピュータで採用.
  - 今日でも大型コンピュータが使われる所以.
  - これのおかげで大型機は入出力が早い.

再録

i386周辺の構造



文献6 p.57

I/Oポートを使った入出力

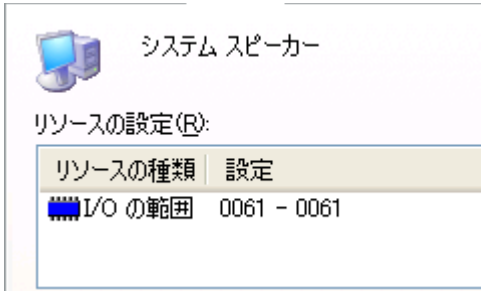
- CPUからは，I/Oポートというメモリのようなモノにデータを置いたり読んだりすることで，機器(ハード)にデータを送ることができる.
  - 途中に入出力制御装置のチップが入っている.
- i386の場合，65536個(2^{16}個)までのアドレスがI/Oポートにふられており，このアドレスで機器を区別する.

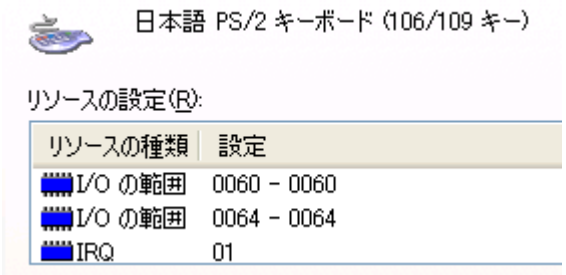
アドレスバスの共用

- アドレスバスは
  - メモリへのアクセスと
  - 入出力機器(I/Oポート)へのアクセスの
  - 両方に使う.
- 共用しているだけで同時には使わない.
  - メモリの場合は32本全てメモリのために使う.
  - I/O機器の場合は16本のみを使う.
- I/Oポートをメモリ上に割り当てるCPUも存在する(した)が, IntelのCPUは分けてる.

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.

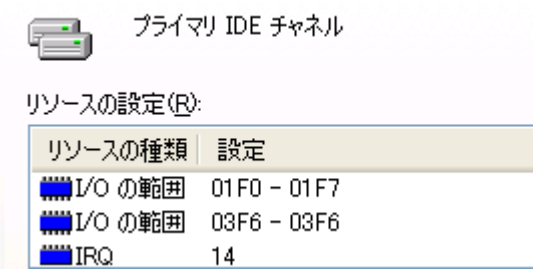


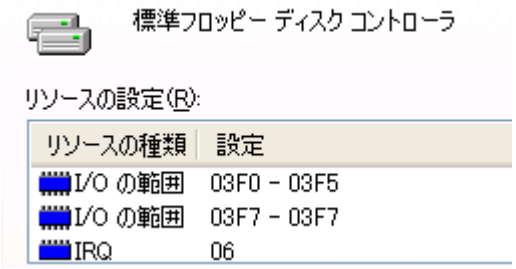


キーボードは0x0060と0x0064のI/Oポートに接続されている.
⇒ これは制御に使われる？(未確認・後述)

例 ディスク関係

いわゆるIDEのディスク
(むかしの一般的なハードディスク)



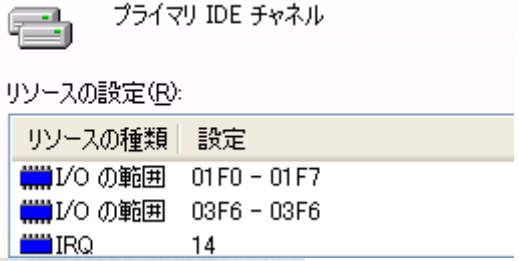


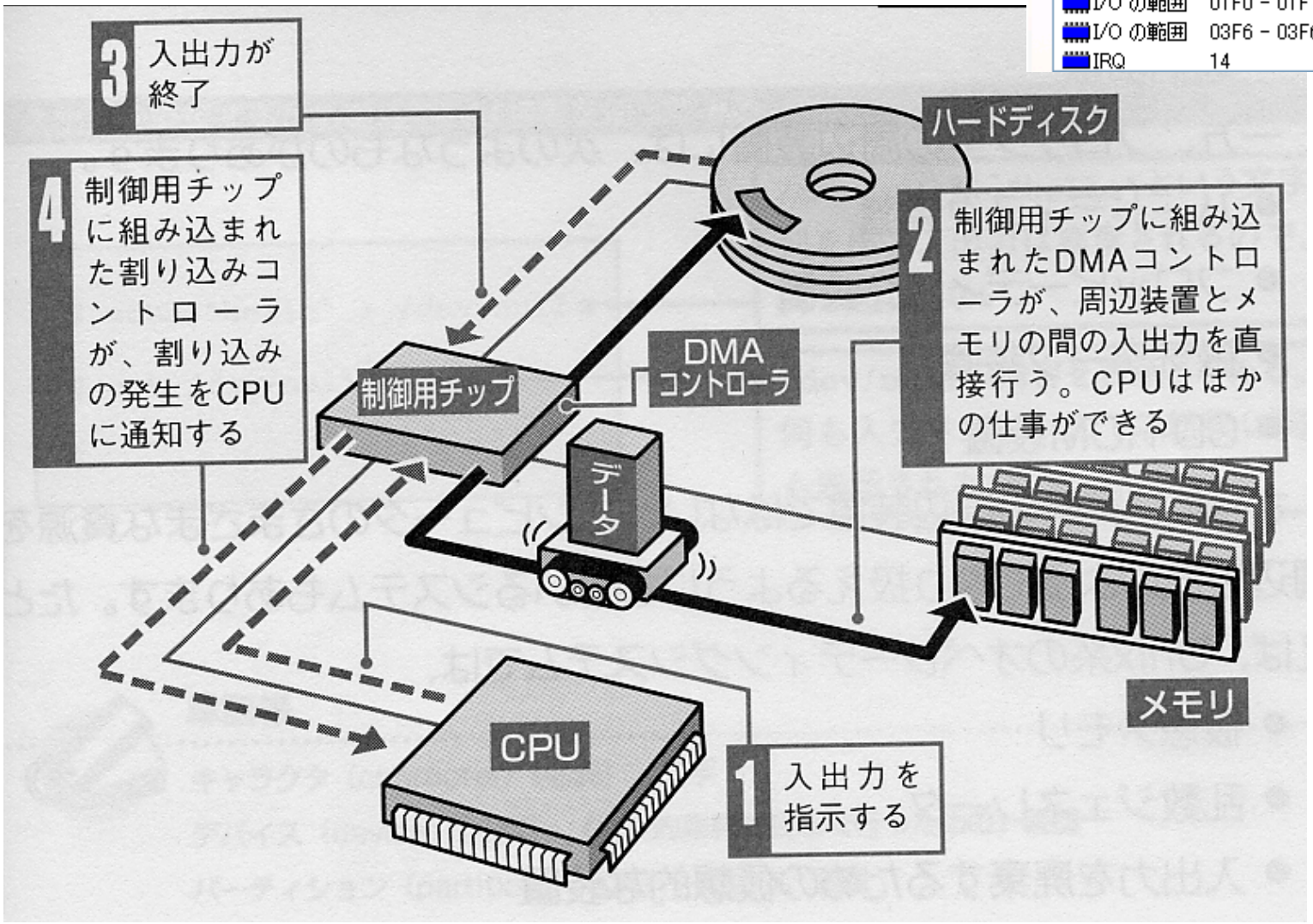
右はフロッピーディスクの制御用のI/Oポート

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，もしくは数個ずつ読んでいては，遅くてラチがあかない.
  - 10^6の速度差を思い出して.
- 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


プライマリ IDE チャネル
リソースの設定(R):
リソースの種類
設定
I/O の範囲
01F0 - 01F7
I/O の範囲
03F6 - 03F6
IRQ
14


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．
- このことも「割り込み」の一種である．
  - システムコール関係のところに出てきた割り込みと同類．

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.

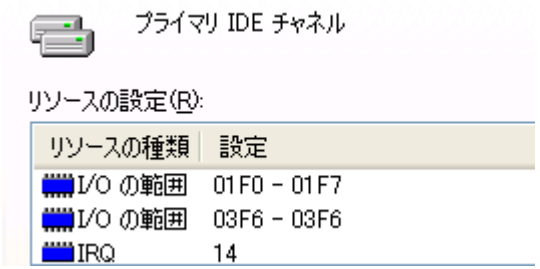
文献5 p.115より

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.
- 386には16個しかない.
  - 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.



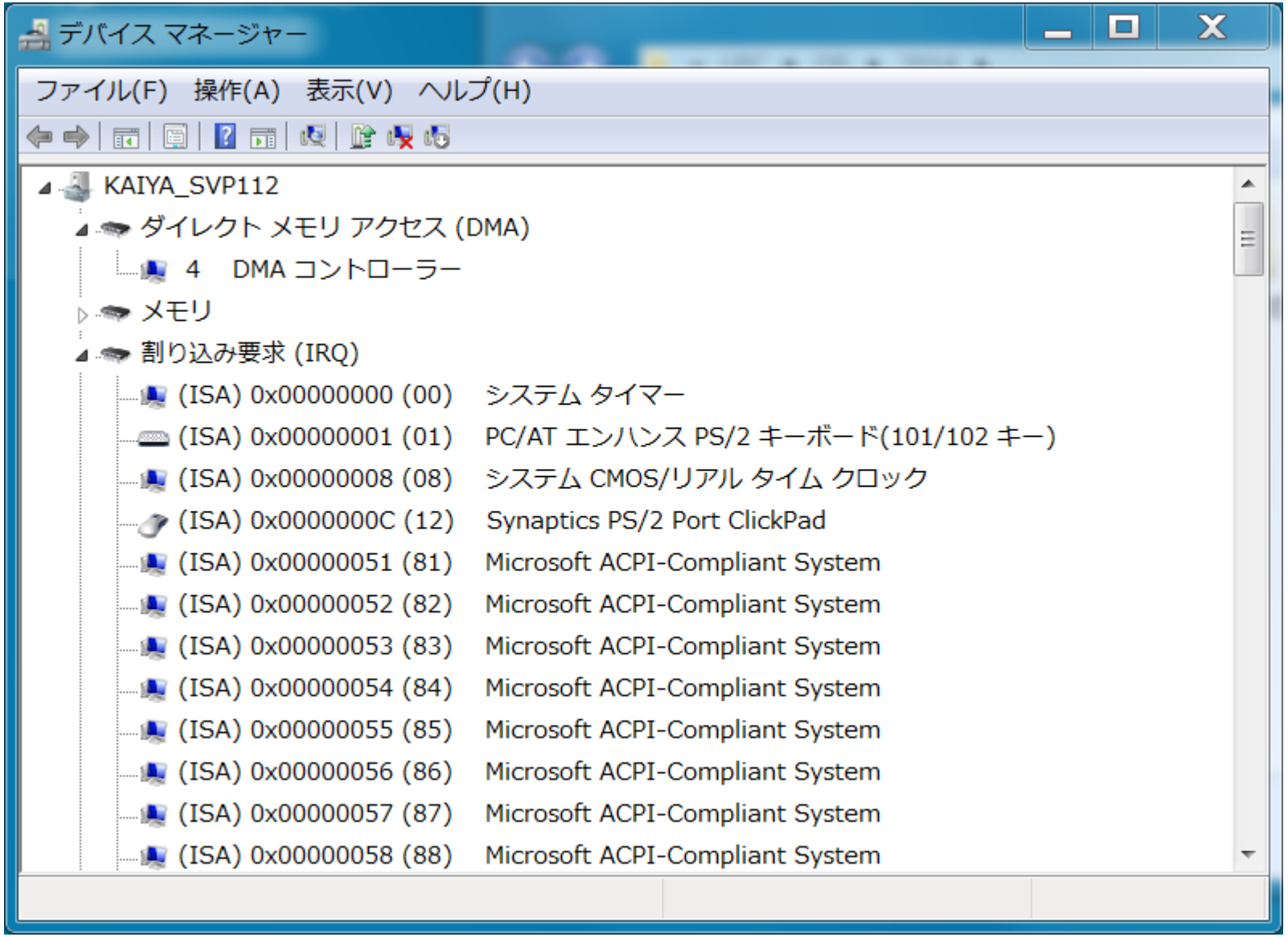
Interrupt ReQuest

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．

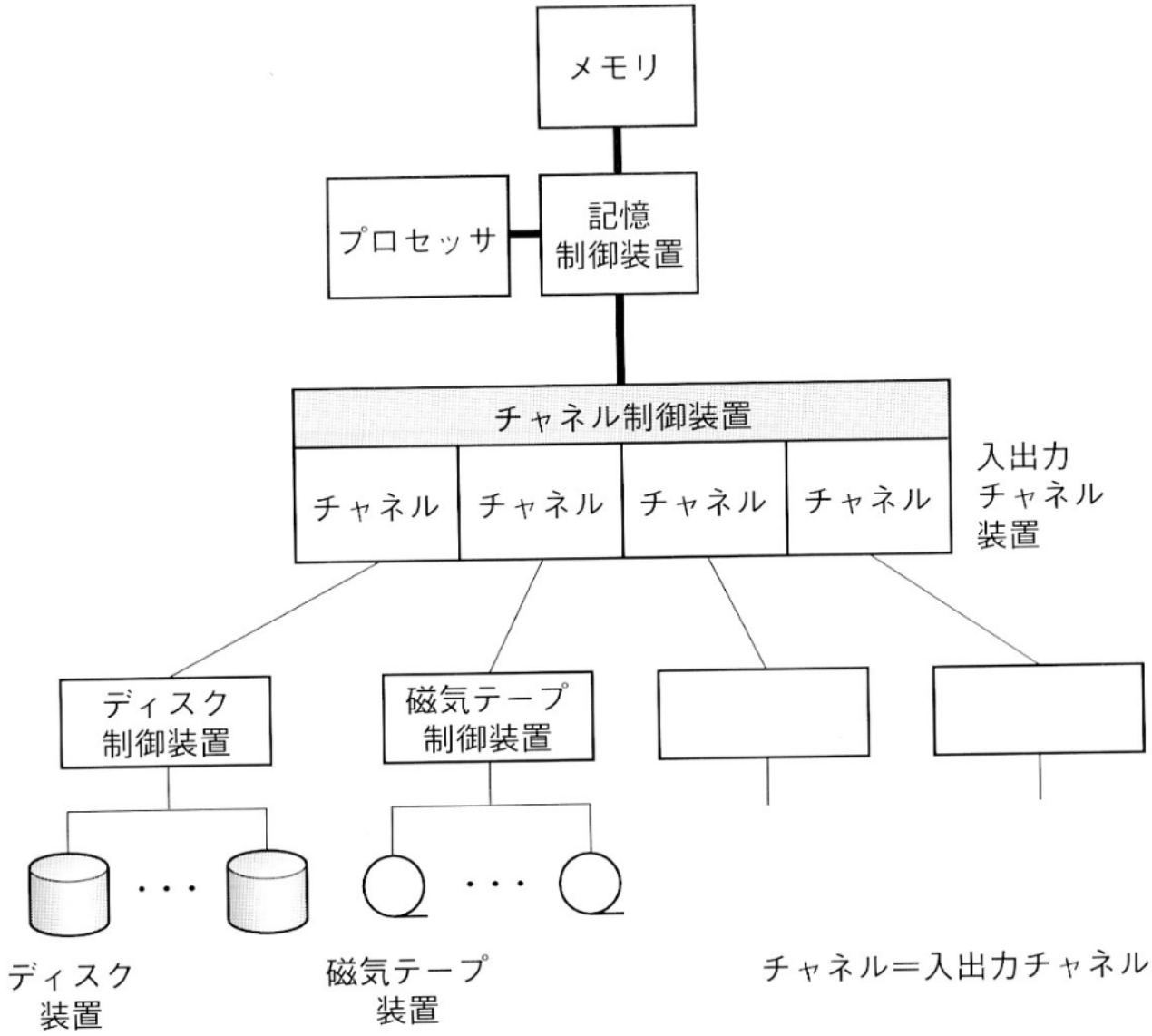
とあるマシン(Core i7)の状況



入出力チャネル

- 大型コンピュータ(メインフレームとも呼ばれる)が並列で高速な入出力ができるための機能.
  - コレのおかげで大型機は今でも入出力に関してPCより優位.
- 入出力専用の下請けコンピュータを多数従えているイメージ (次項)
- 入出力を高速に行うため，専用メモリ上にデータを先読みしたり，処理中のデータを専用メモリ上にのみ保持したりする.
  - キャシュと呼ばれる.
  - キャッシュは今時はPCでも普通に使われている.

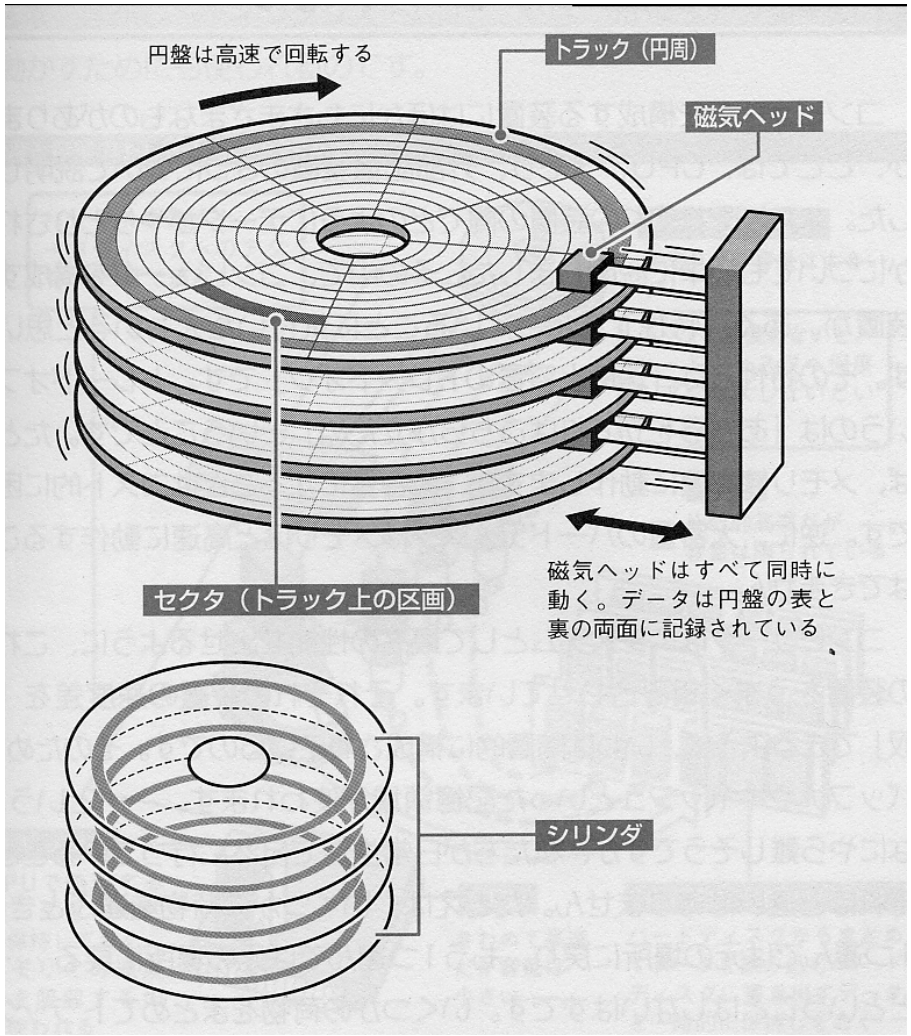
チャネル


メモリ
プロセッサ
記憶
制御装置
チャネル制御装置
チャネル
チャネル
チャネル
チャネル
入出力
チャネル
装置
ディスク
制御装置
磁気テープ
制御装置
ディスク
装置
磁気テープ
装置
チャネル＝入出力チャネル

入出力装置の実イメージ

- 代表的な記憶装置としてのハードディスクについての実イメージを以降に示す.
- SSDやUSBキー等の，実際には円盤が回っているわけでない装置も，置換性から，同じように扱われる.

ディスクの物理構造



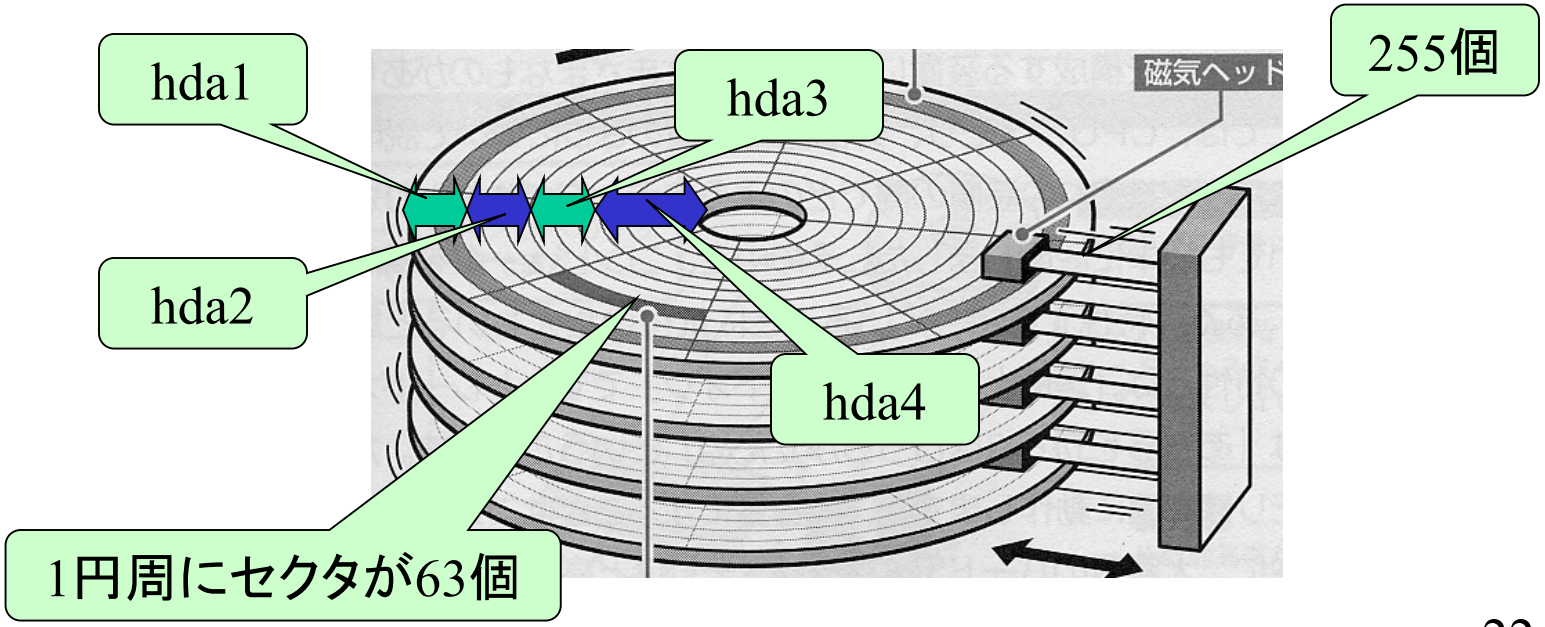
「計算機システム基礎」でも似たような図をみたのではないでしょうか？

とあるディスクの実情報

```
ディスク /dev/hda: ヘッド 255, セクタ 63, シリンダ 2434
ユニット = シリンダ数 of 16065 * 512 バイト

  デバイス     始点  終点     ブロック      ID  システム
/dev/hda1       1   255    2048256    83  Linux
/dev/hda2     256   321     530145    82  Linux スワップ
/dev/hda3     322   576    2048287+   83  Linux
/dev/hda4     577  2434   14924385    83  Linux
```

4つのパーティション



ディスクの回転数

- 以下が一般的.
- 5400 rpm もしくは
- 7200 rpm
- rpm = round per minute = rotation per minute = 一分当たりの回転数
- よって，7200 rpm の場合，
  - 1秒間に120回，回転する. わりと早い.

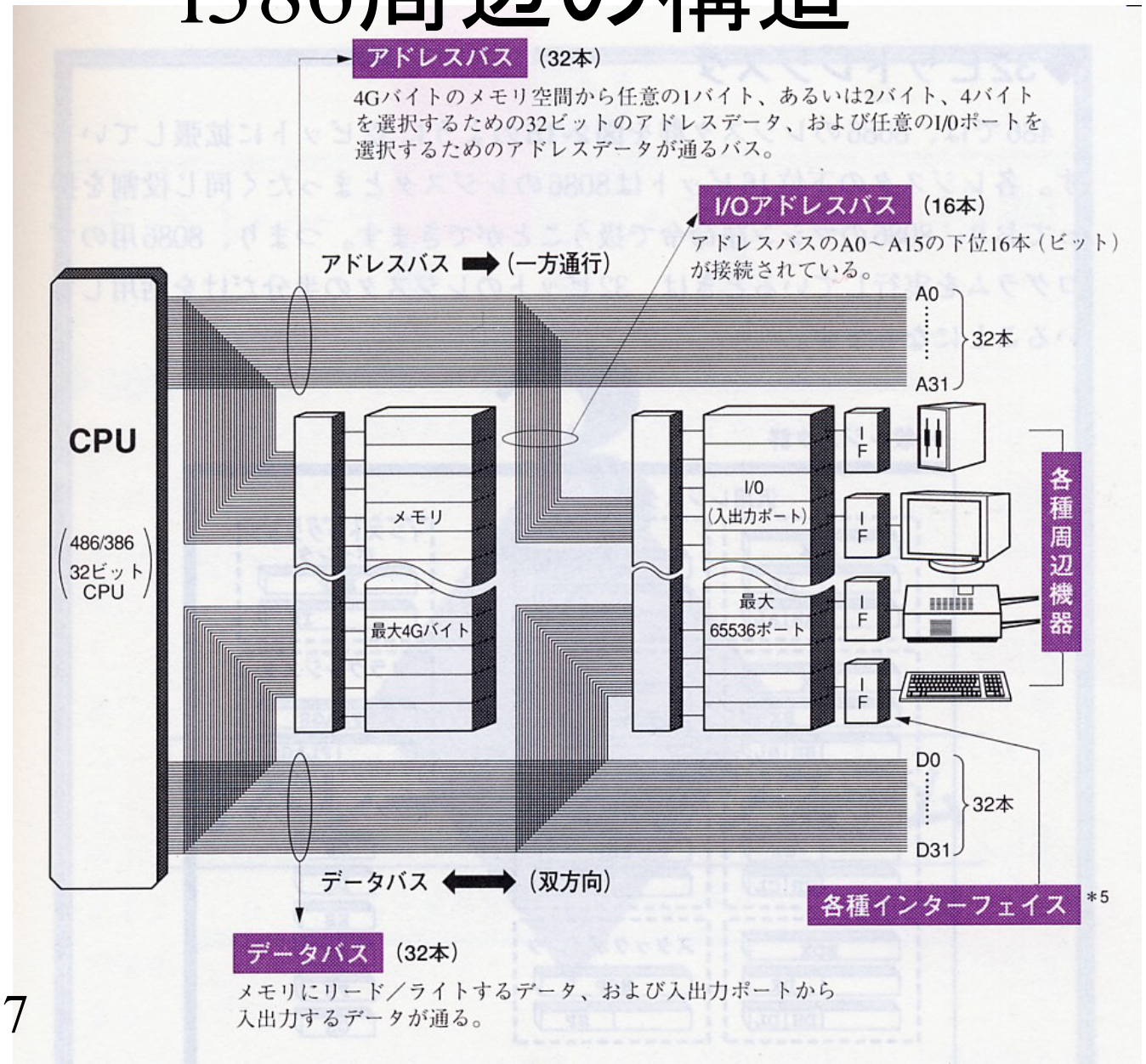
とあるUSBメモリの情報

Disk /dev/sdc: 2013 MB, 2013265920 bytes
16 heads, 15 sectors/track, 16384 cylinders
Units = cylinders of 240 * 512 = 122880 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xed044527

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc1		33	16384	1962212	6	FAT16

再録

i386周辺の構造



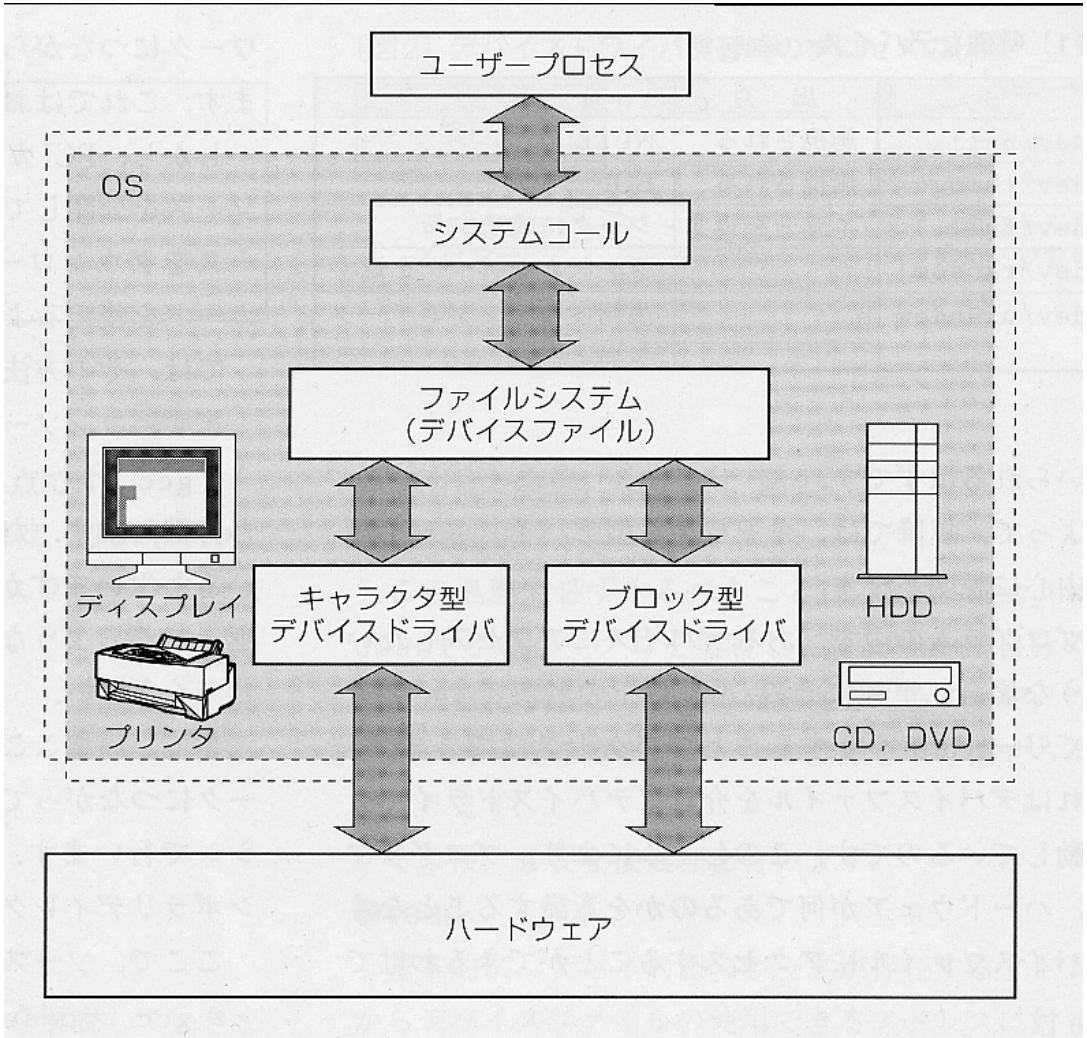
文献6 p.57

Linuxでのデバイスの抽象化 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プログラマが大変.
- 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USB3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

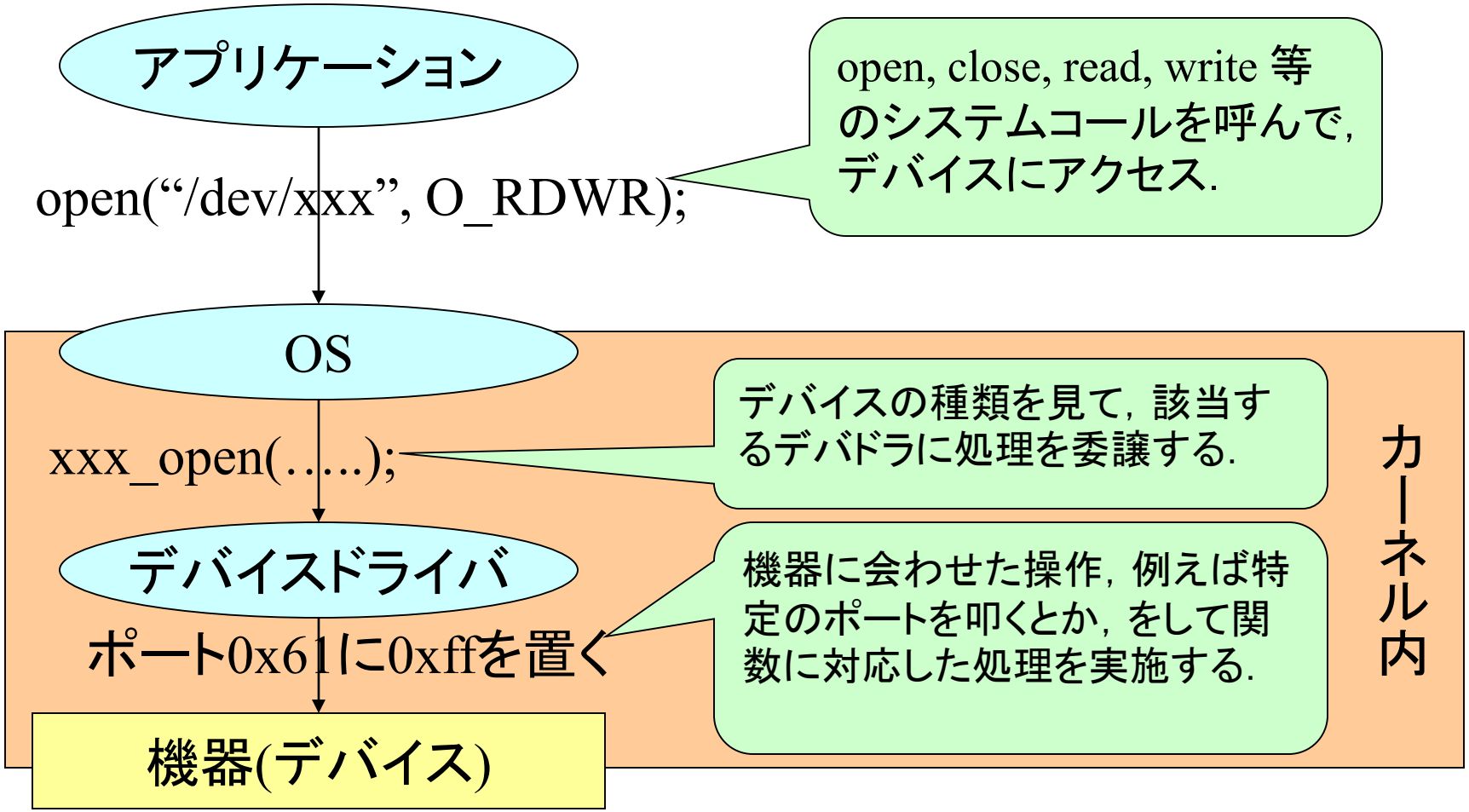
/dev/ ディレクトリの下にある.

- /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 ※ 最近は無い

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IEEE1394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.
- 一般にブロック型のほうが複雑.
  - 後述のブロッキングやキャッシュ等を行うため.

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
```

抜粋

```
Block devices:
  3 ide0
```

ブロック or キャラクタの区別

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.

5.3 入出力の効率化技術

- ブロッキング
  - データをある程度のまとまり(4KB等)毎に入出力を行う.
  - 前述のブロックデバイス(ディスク等)はブロッキングを行うため，こう呼ばれる.
- バッファリング
  - 入出力装置を直接読み書きするのではなく，メモリにある程度蓄えてから読み書きする方法.
- キャッシング
  - 入出力装置の中身を予めメモリに読みだしておき，そこに対して処理を行う方法.

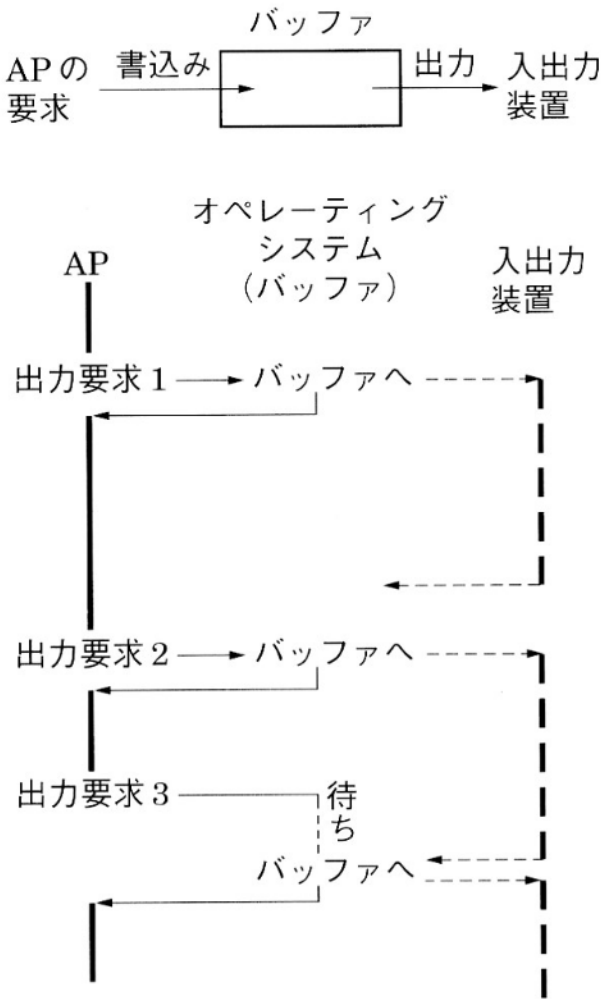
ブロッキング

- 前述のようにデータをある程度のまとまり(例えば4KB等)にまとめて入出力を行い，入出力回数を減らす技術.
- HDの場合，セタクタ単位でのデータの読み書きが最小単位だが，通常，ブロックはセクタより大きい.
  - 本スライドの例では1セクタ=512B
  - 8セクタを1ブロックとしている.
- 連続的な大規模データを扱う場合，そのままでも効率的だが，飛びとびの場合，後述のキャッシュと組み合わせないと具合が悪い.

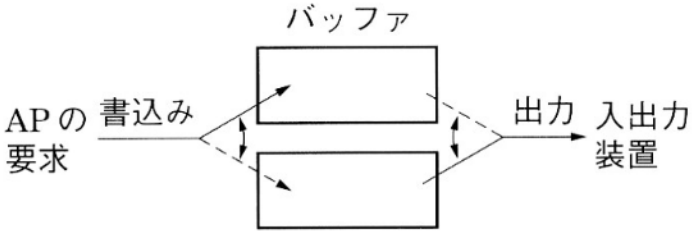
バッファリング

- (以前にも触れたが，)データをある程度の量蓄えるための緩衝領域，通常，メモリ上にとる.
- 前述のようにメモリは10^6くらいディスクより速いので，逐次的にディスクに書き込むより無茶苦茶はやくなる.
- 今時，色々なレベルでバッファリングが行われ，処理の高速化に寄与している.
  - 教科書のようにOS内の入出力処理において.
  - APIにおいてもバッファを採用しているものがある. fgets, fread 等

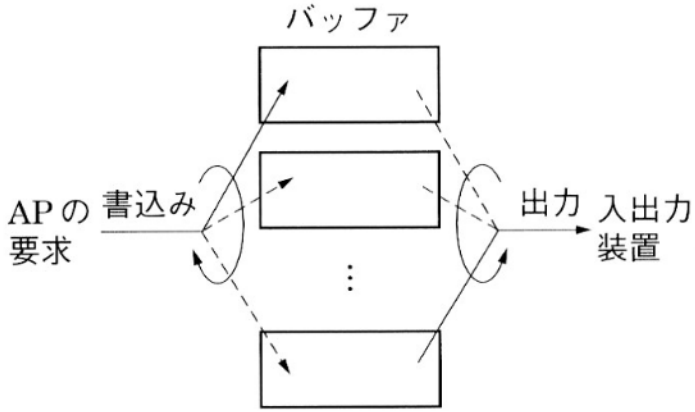
I/Oにおけるバッファの方式



（a） 1面バッファ



（b） 2面バッファ

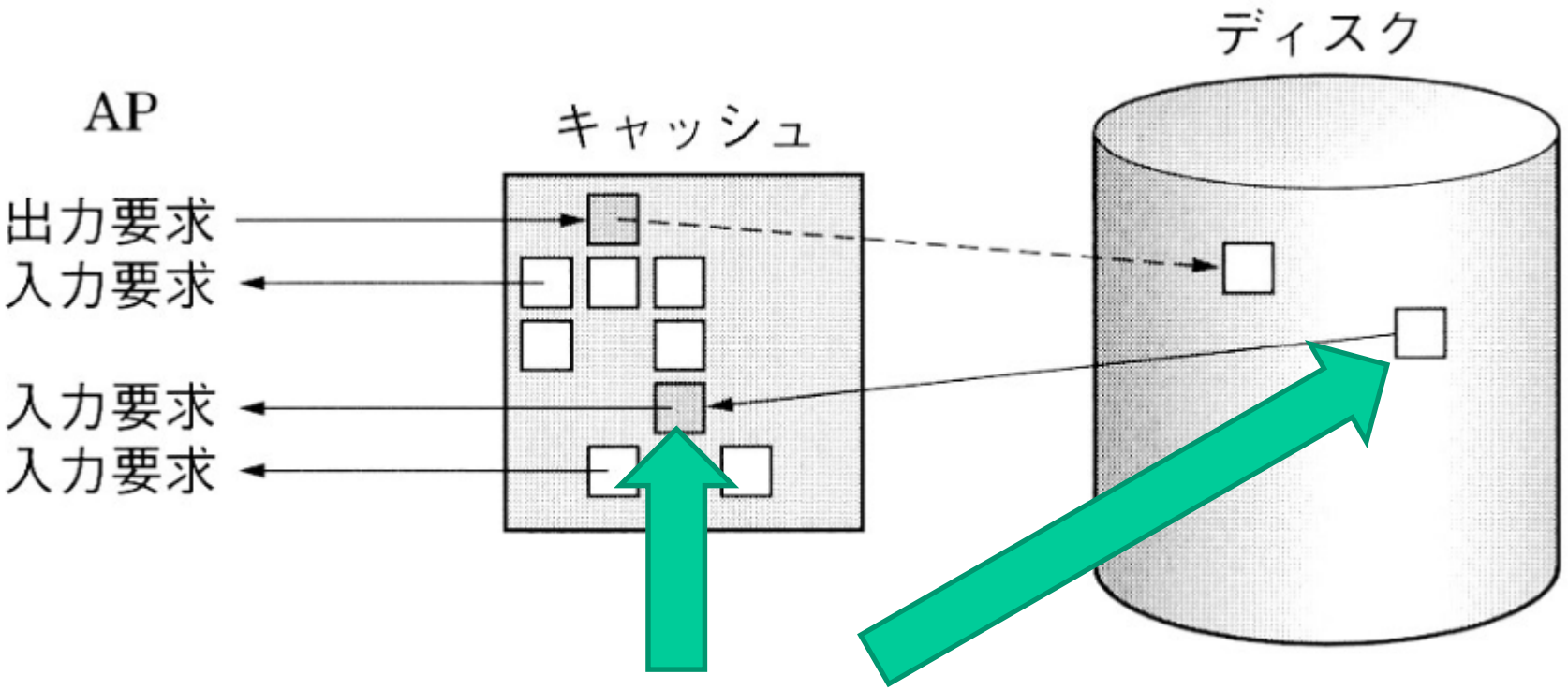


（c） 多面バッファ（バッファプール）

キャッシング

- バッファと似てるが，あたかもディスク等にあるデータの実体にアクセスしているようにアプリからは見えるが，実際はメモリ上のコピーにアクセスする仕組み．
- 当然，実体へのアクセスより速い．
- 効率化のため教科書の「遅延書き出し」を行うのが普通．
- 実体とキャッシュとの同期に失敗するとデータが失われる (涙)
- 今時は，OS以外のレベルでもキャッシュは行われる，例えばブラウザのキャッシュ等．

キャッシュのイメージ



最終的に同期をとらないと大変なことに・・・

本日は以上

アンケートのほう，
よろしくご提出ください