オペレーティングシステム2005
デバイス管理 (2)

2005年12月15日

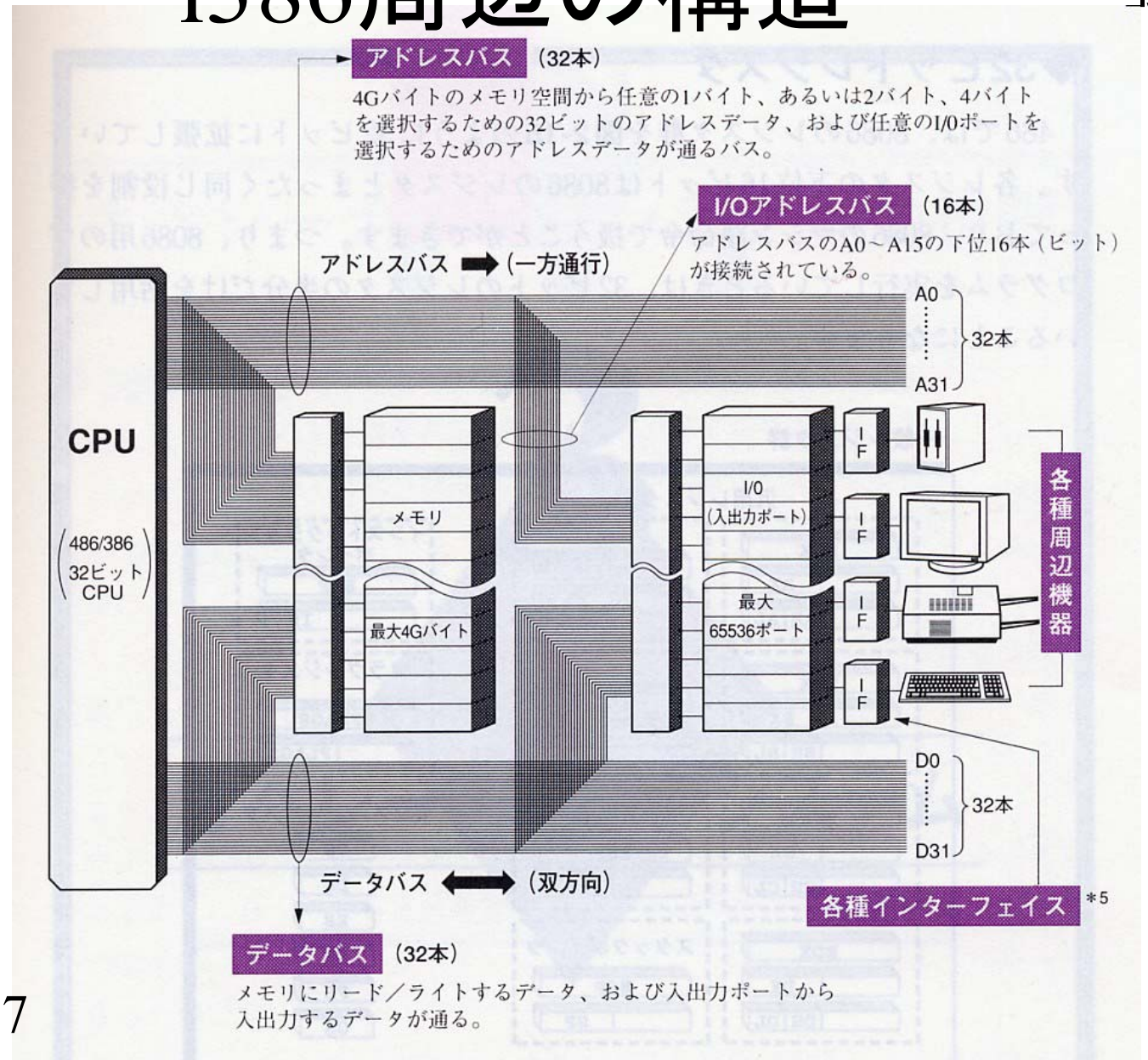
海谷 治彦

目次

- モジュール
- デバイスドライバの例

i386周辺の構造

復習



文献6 p.57

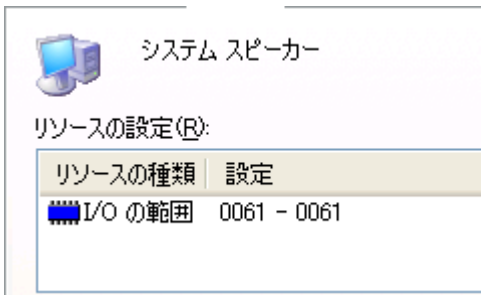
復習

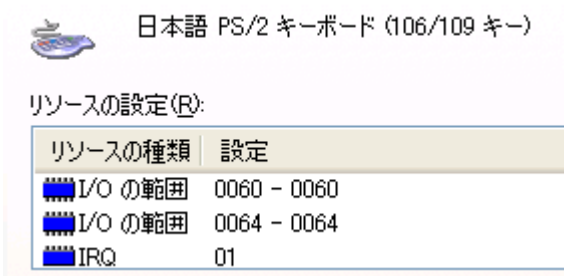
I/Oポートを使った入出力

- 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.

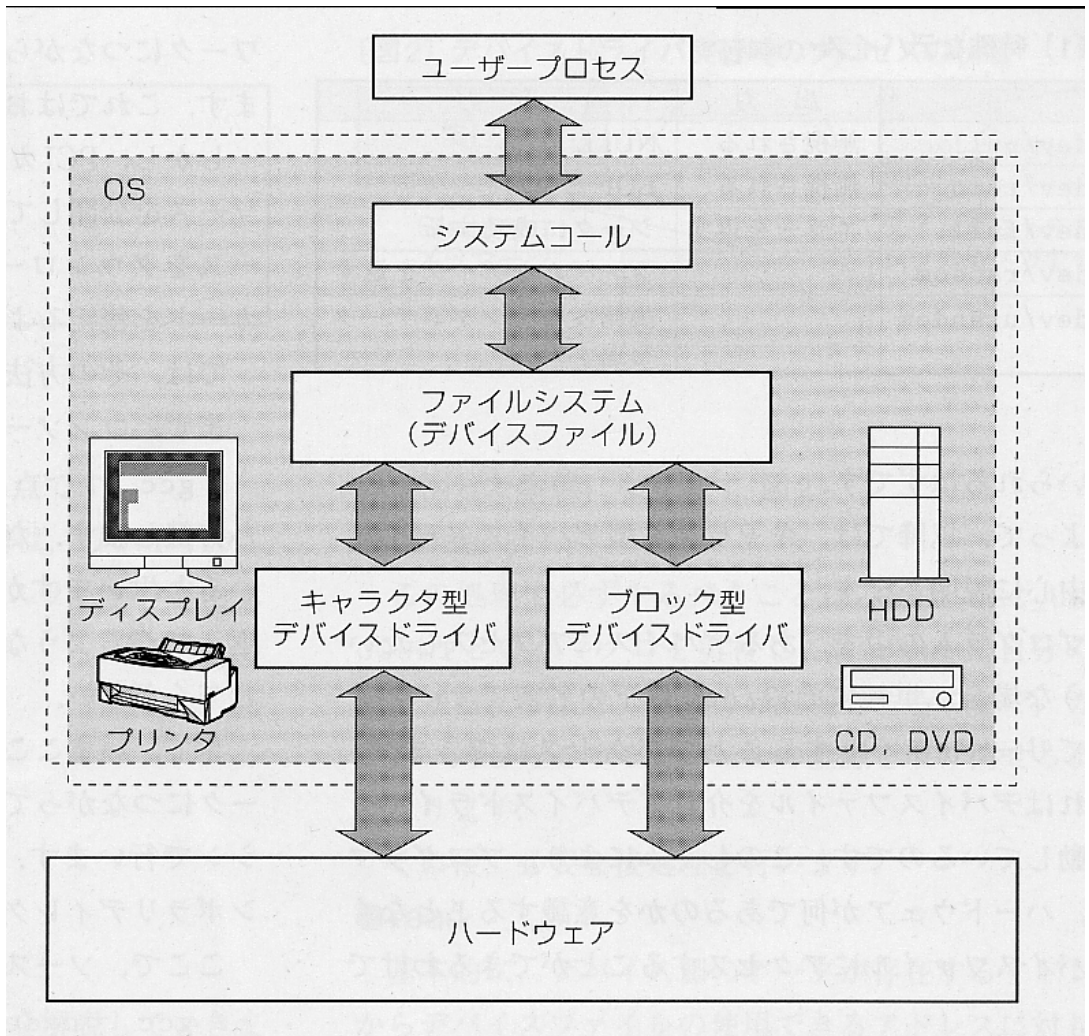




キーボードは0x0060と0x0064のI/Oポートに接続されている.
⇒ これは制御に使われる？(未確認・後述)

復習

デバイスの抽象化

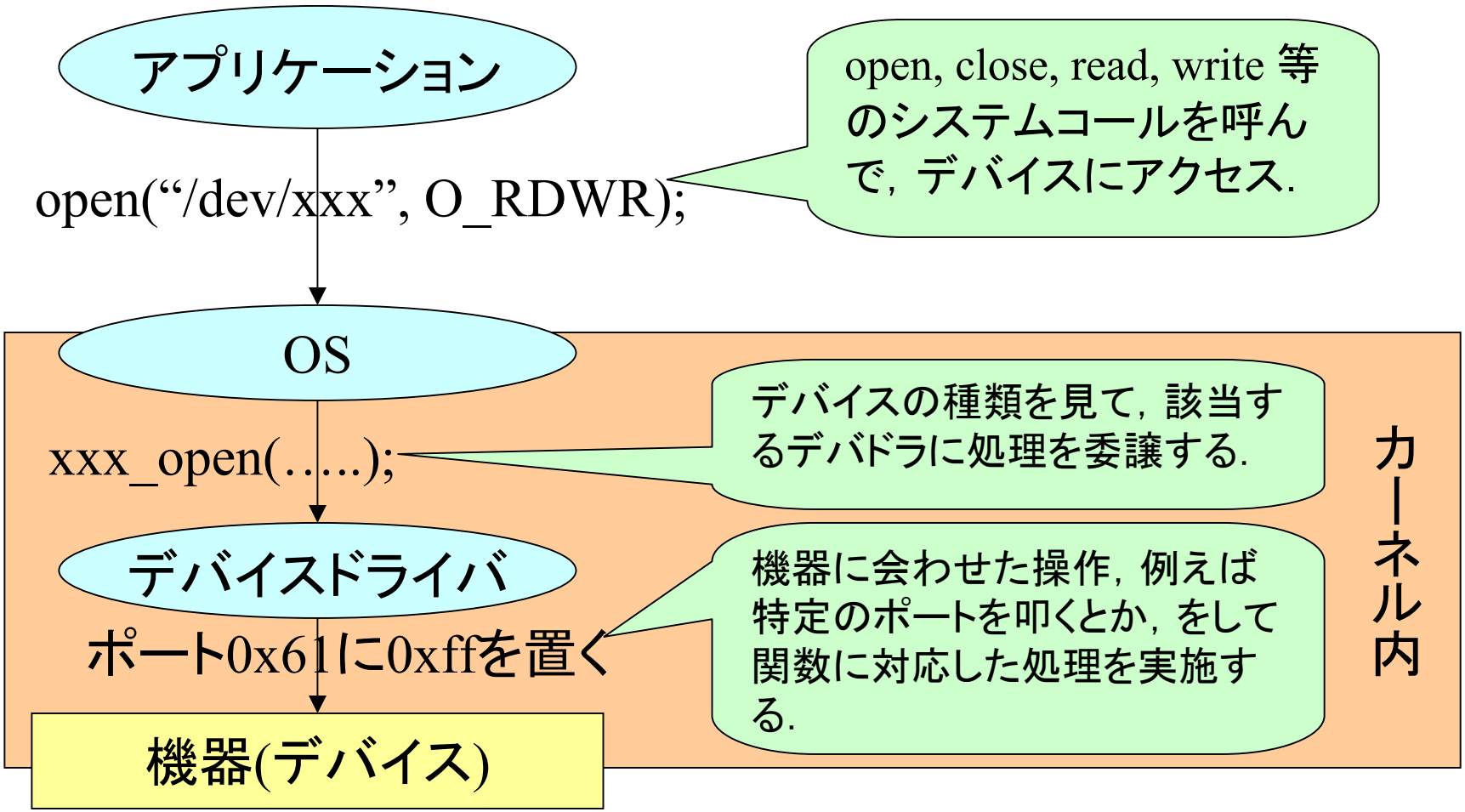

ユーザープロセス
OS
システムコール
ファイルシステム
(デバイスファイル)
ディスプレイ
プリンタ
キャラクタ型
デバイスドライバ
ブロック型
デバイスドライバ
HDD
CD, DVD
ハードウェア

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

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.
カーネル内
機器(デバイス)

モジュールとは？

- 実行中にカーネルに対して組み込みもしくは削除可能なオブジェクトコード (なんとか.o) である.
  - 要は動作中のプログラムに関数を途中で追加とか削除とかできてしまう.
- OSの動作最中に機能・削除ができて便利.
  - 新しいデイバスドライバを途中で追加できる.
- OSをカスタマイズするのも楽になる.
  - OS全体をコンパイルしなおすのはやっぱり時間や手間がかかる.

モジュールの作り方の例

- C言語で作る.
  - カーネルの一部なので標準関数が使えない.
  - オブジェクトコードだけを作るので, `-c` オプションをコンパイラに与える. (例えば `gcc -c` なんとか`.c`)
  - オプション `-O` も必要.
- 特定のヘッダーやマクロを定義する.
- 特定の関数を最低限定義しなければならない.

```
int init_module(void)
void cleanup_module(void)
```

- 詳細は実例にて.

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.

モジュールの追加・削除の仕方

- カーネルへの追加の方法
  - /sbin/insmod なんとか.o
- 削除の方法
  - /sbin/rmmod モジュール名
  - なんとか.o でないので注意
- 追加されているモジュールの表示
  - /sbin/lsmod コマンド, もしくは,
  - /proc/modules ファイルの中身を見る.

デバイスの作成

- モジュールをカーネルに登録しただけでは，あるデバイスを処理するコードが追加されただけである．
- そのモジュールを使ってアクセスするデバイスファイルを作成する必要がある．
- コマンドは以下

  mknod /dev/なんとか c メジャー番号 マイナー番号

詳細は例題にて．

実例 (beepdriver)とデモ

- open, close を使って，PCのブザーを鳴らしたり，とめたりするダケのデバイスドライバ.
- 一応，ブザーというデバイスを制御している.
- マシンに大きく依存する.
- ソース自体もカーネルバージョンに大きく依存. 2.4カーネルの一部でテストした.
  - ノートパソコンではテストしてない. (該当マシンをもってない)
- 参考文献 4の3章にある例題を簡素化したもの.
- ソースがおいてあるWebページにはアクセス制限あり.

リンケージについて

付録的な話

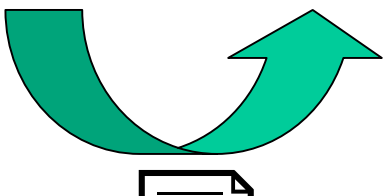
Cというか一般的なUNIX, Linuxのバイナリ

Cプログラムの開発の流れ

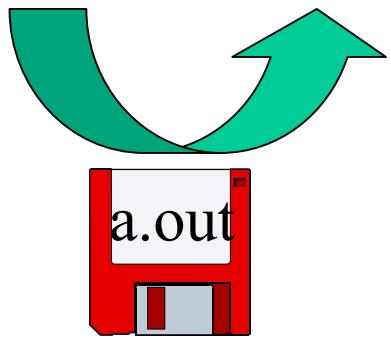
プログラミング
(手作業)

コンパイル

実行



ソースコード
(原始プログラム
hoge.c 等)



ロードモジュール
(実行可能プログラム
a.out 等)

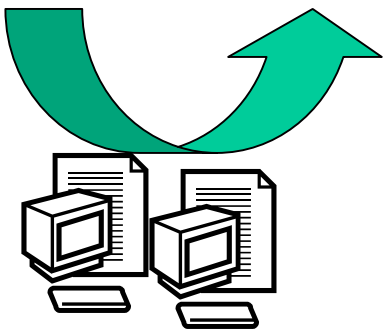
分割コンパイル

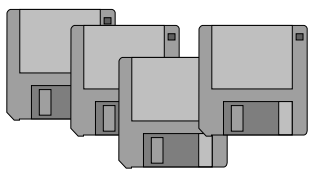
プログラミング
(手作業)

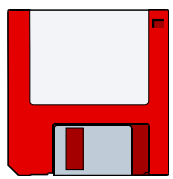
コンパイル

リンケージ
・エディット

実行







ソースコード
(原始プログラム
hoge.c 等)

オブジェクトコード
（目的ファイル，
マシン語，
hoge.o など)

ロードモジュール
(実行可能プログラム
a.out 等)

プログラム実行時に注目する点

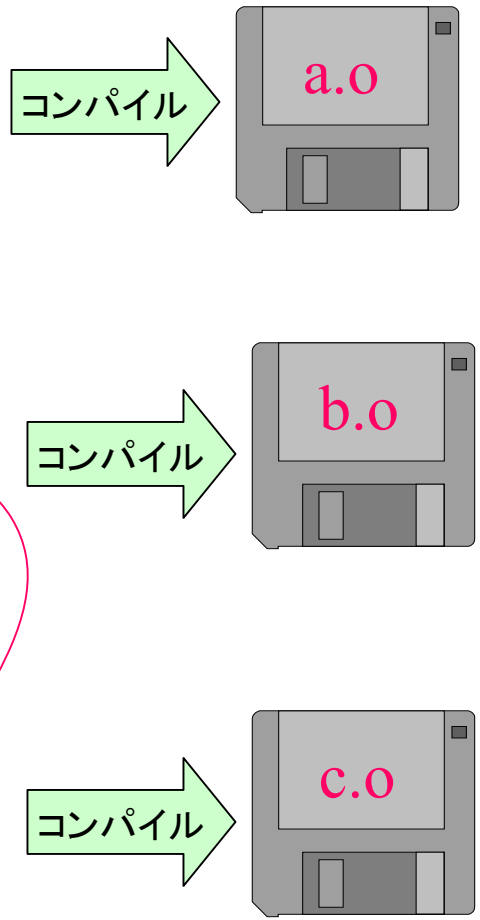
- オブジェクトコード(マシン語)の中身の構造
- ばらばらのファイルに分かれたマシン語が相互利用できる仕組み(リンケージ)
- マシン語がマシンに読み込まれる仕組み(ロード)
- 読み込まれてからの振る舞い

相互利用とは？

```
// a.c
int foo(){
    .....
}
```

```
// b.c
int bar(){
    x=foo();
}
```

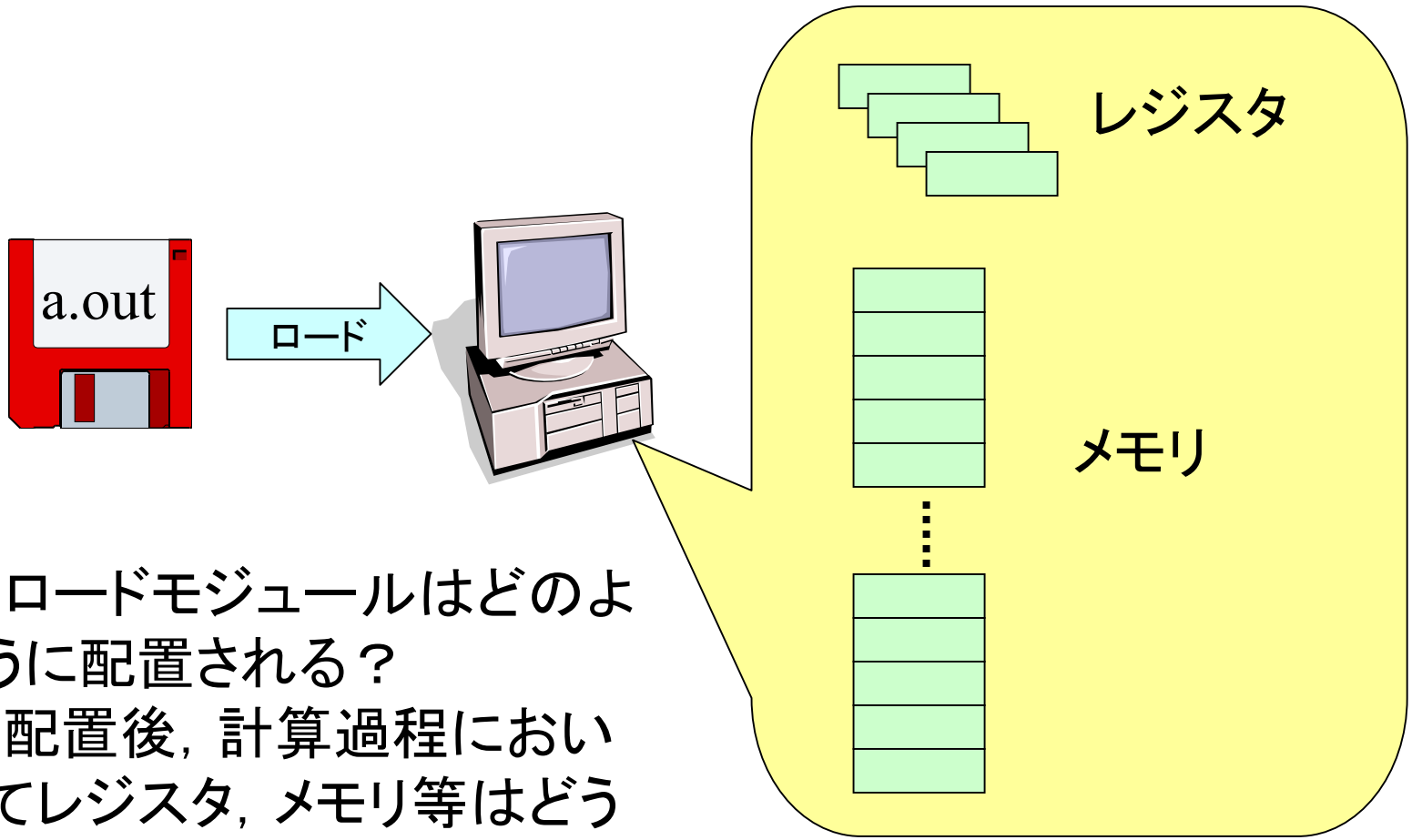
```
// c.c
main(){
    x=bar();
}
```



a.o から
foo() をどう探す？

b.o から
bar() をどう探す？

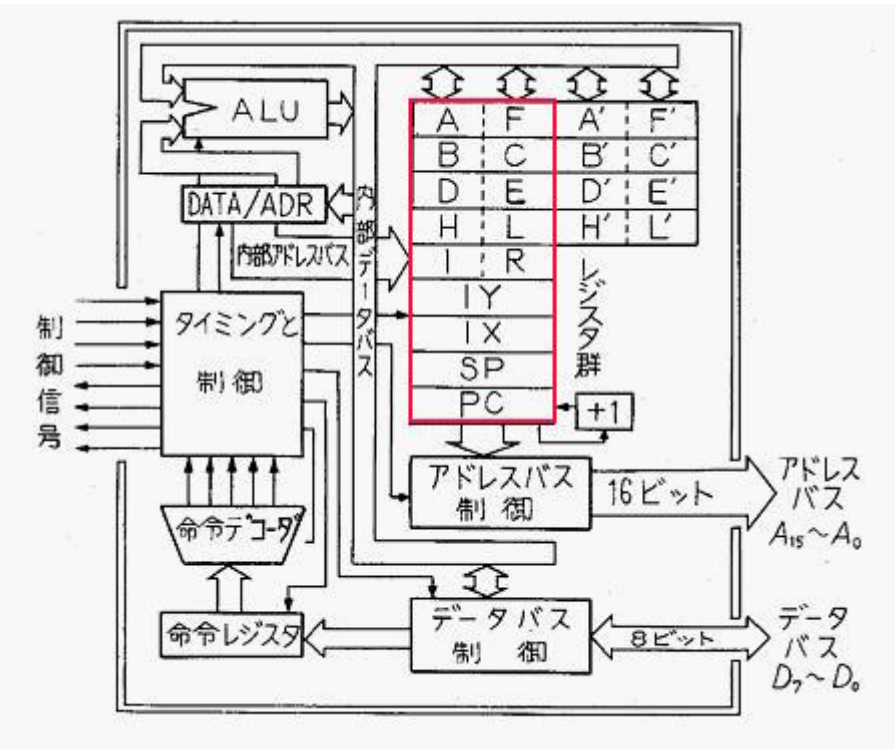
マシン語に読み込まれる

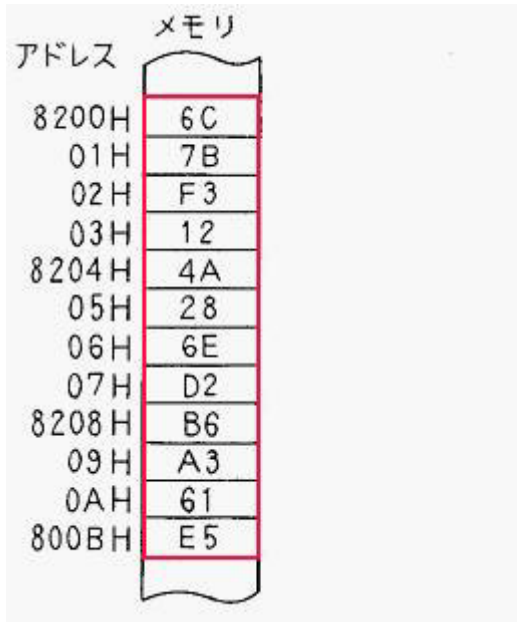


- ロードモジュールはどのように配置される？
- 配置後，計算過程においてレジスタ，メモリ等はどう変化する？

ノイマンマシンの基本構造

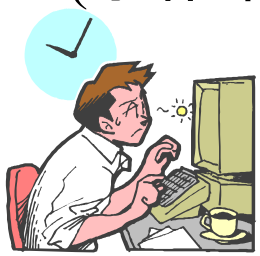
レジスタと呼ばれる数個の変数と，メモリと呼ばれるたった1つの配列しか使えないプログラミング環境.





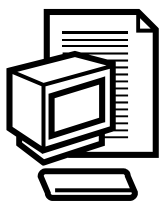
Javaプログラム開発の流れ

プログラミング
(手作業)



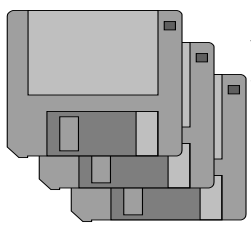
コンパイル
javac hoge.java

実行
java hoge



hoge.java

ソースコード
(原始プログラム
hoge.java 等)



hoge.class

クラスファイル
(**hoge.class** 等)

JavaとC(等)との違い

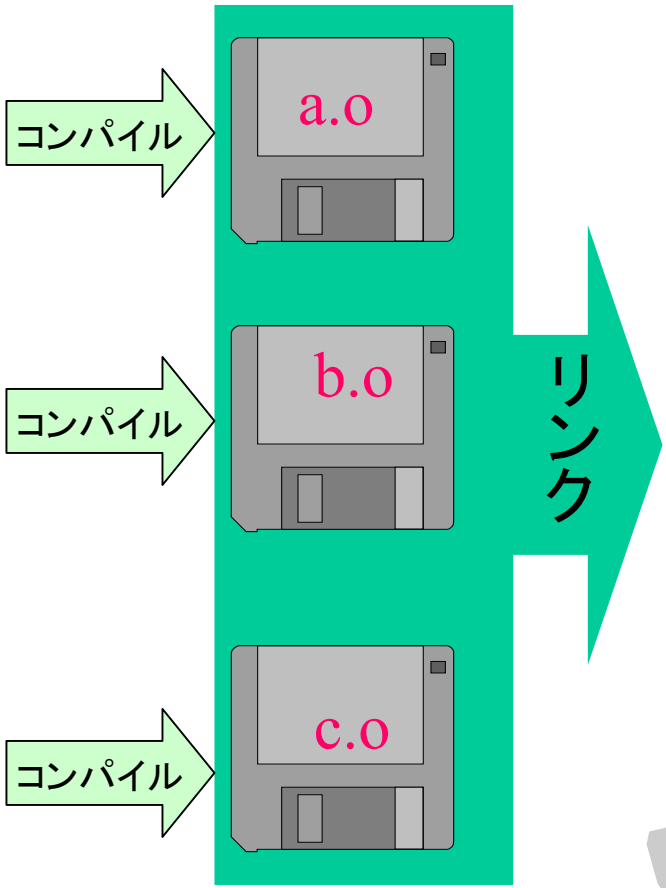
- リンケージはどこでやっているのか？
  - ロードした後，マシン内でやっている.
- ロードはどうやってやっているのか？
  - 必要なクラス(マシン語)のみロードする.

Cのロード

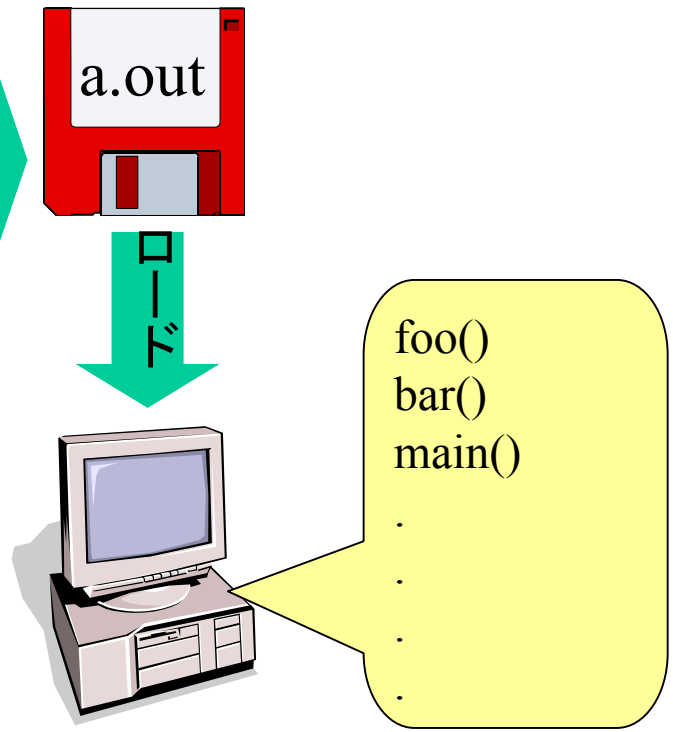
```
// a.c
int foo(){
   .....
}
```

```
// b.c
int bar(){
   ........
}
```

```
// c.c
main(){
 if(x>10)
   foo();
 else
   bar();
}
```



基本的にリンクされた関数は全てマシン内に読み込まれる.



Javaのロード

```
// foo.java
class foo{
   .....
}
```

コンパイル → foo.class

```
// bar.java
class bar{
   .......
}
```

コンパイル → bar.class

```
// go.java
class go{
... main(...){
 if(x>10)
   new foo();
 else
   new bar();
} ....
```

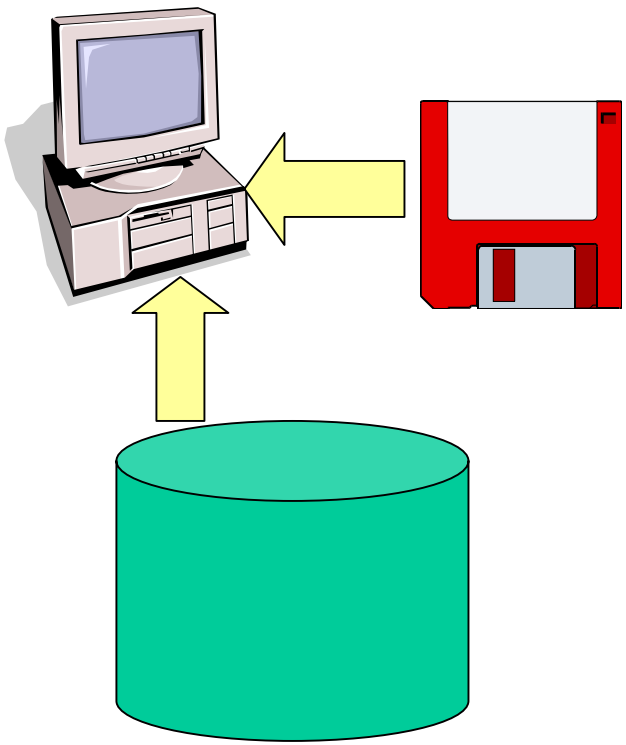
コンパイル → go.class

最初のロード
java go

例えば,
go.class
foo.class
.
.
.

- 事前にリンクしない.
- 必要に応じて必要なクラスを追加ロード.
- (分岐等の場合，一方しかロードされない．)

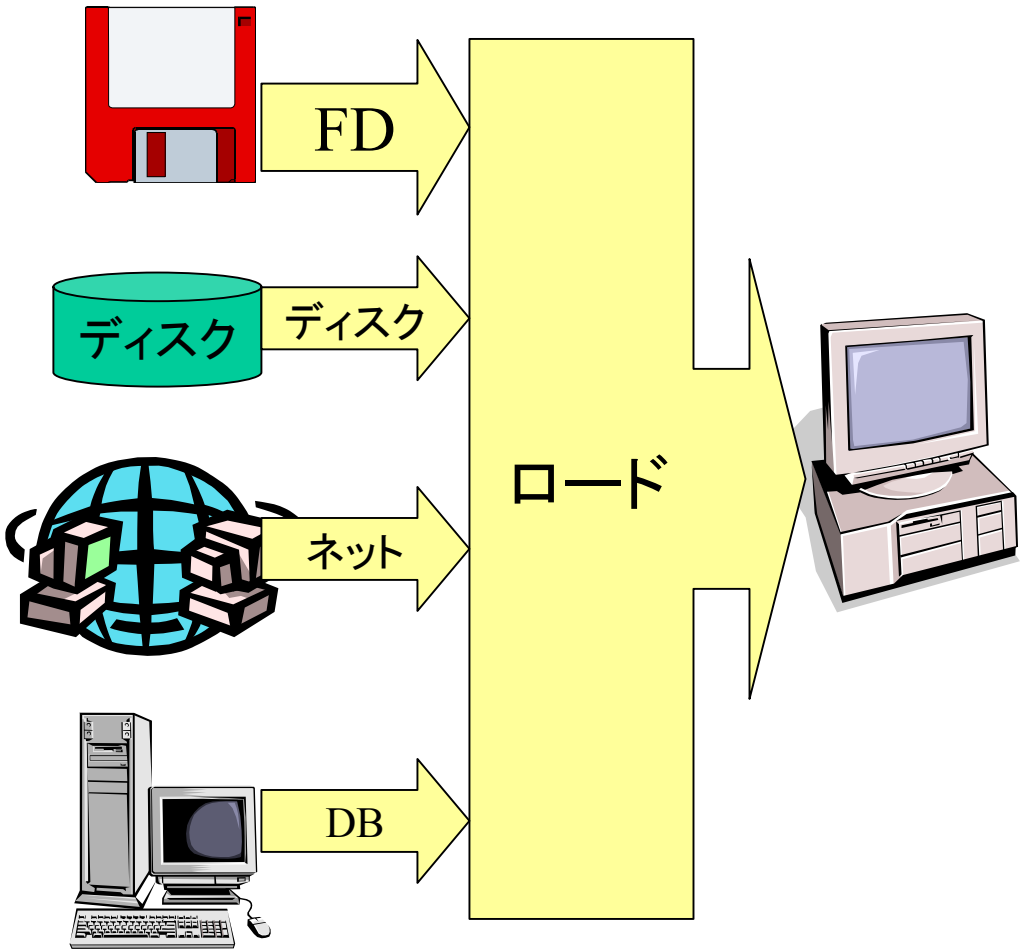
Cの場合のロード元



- 基本的には自身の記憶デバイス(ディスク，FD，CDROM等)からしかロードできない.

- 1つのプログラムにおいて，多様なロード先を持つことは難しい. (というか原則，ロードは1回)

Javaの場合のロード



- 多様なソースからクラスをロードできる.
- 1つのプログラムが複数のソースを持てる.

insmod/rmmodについて

- 実行中のロードモジュール(カーネル)に関数を後から追加している.
- とはいえ，構造体 file_operations準拠のものでないと，うまくくっつかない.
  - 基本的には関数へのポインタを使って，あとからくっついた関数を探すため.