# ソフトウェア第三
# —デバイスへのアクセス，OSのブート，組込システム—

機械情報工学科　水内郁夫

http://www.jsk.t.u-tokyo.ac.jp/~ikuo/lec/soft3/

(ID:soft3, pass:MechanoI)

2008年11月10日(月)

## 1　デバイスへのアクセス (再掲)

一般にコンピュータに接続されたデバイス (キーボード・プリンタ・ネットワークインタフェース・音声・マウス・通信等々) にアクセスするためのユーザプログラムを書く場合，OSのデバイスドライバを経由する．ユーザプログラムでは，デバイスファイル (UNIXでは/dev/以下にあるファイル) をopenし，read，write，ioctl等のシステムコールを用いてデバイスを利用する．デバイスドライバは，デバイスを操作する具体的な方法を知っていて，ユーザプログラムの呼び出すシステムコールに応じたデバイス操作を行う．ここでは，Linuxのシステムにおいてデバイスへのアクセスがどのように行われるかを例にとって説明する．

### 1.1　カーネル空間とユーザプロセス空間

OSの機能を実行するプログラム (OSそのもの) はカーネル (kernel) と呼ばれる．OSのカーネルの使用するメモリ領域は，一般に物理アドレスで，不動でありページアウトしない (MMUによるアドレス変換が行われない)．ユーザプロセスのメモリ空間は論理アドレスで，MMUによる物理アドレスへの変換があり，ページングも発生する．ユーザプロセスは簡単にはカーネルのメモリ空間にアクセスすることはできない．カーネル空間のメモリにアクセスするための専用の関数 (copy_to_user()等) が用意されている．

### 1.2　システムコール

システムコールは，ユーザプログラム・アプリケーションプログラムが，OSのサービスを呼び出すためのインタフェースである．プログラムからシステムコールが呼び出されると，ユーザプロセスモードからカーネルモードに切り替わる．システムコールは，前回の資料・課題に挙げたような例がある．

また，fopen，printf等の標準ライブラリの関数も，中でシステムコールを呼び出している．システムのライブラリ関数の呼び出しは，システムコールに対しライブラリコールと言う．ライブラリコールの実行は，ユーザ空間で行われるが，その中から呼び出されるシステムコールの実行はカーネル空間で実行される．

## 1.3 デバイスドライバ

デバイスドライバは，ハードウェアとソフトウェアを結ぶインタフェースの役割を担うソフトウェアである．コンピュータに接続される物理デバイスは，それぞれが独自の制御装置を持っている．シリアルポートはたいていUARTというチップで制御されるし，ハードディスクは，IDE(ATA)・SCSI[1]・SATA等のコントローラにより制御されている．個々のデバイスのコントローラは，そのデバイスを操作するためのコントロールレジスタや，そのデバイスの状態を表すステータスレジスタを持っている．デバイスドライバは，バスを介してこれらのコントローラのレジスタにアクセスし，デバイスとのやりとりを行う．

デバイスドライバには，デバイスの操作を行う部分と，ユーザプログラムからのアクセスに対応する部分とがある．

デバイスの操作を行う部分は，デバイス制御装置からの返事 (制御装置のステータスレジスタの変化など) を待たなければならない場合が多い．デバイス制御装置から返事が来たことを検知する方法にはポーリングと割り込みがある．ポーリングは頻繁にデバイスからの返事が来たかを確認するが，デバイスからの返事が来たら割り込みがかかるようにする方がCPUの使用量は少なくてすむ．

Linuxでは，割り込みの際の処理が軽い場合は即座に行う場合もあるが，割り込みの際に実行する処理量が多い場合は，割り込み処理ルーチンの中では処理せず，割り込みがあったことの記録だけされ，実際の処理は後で行われる．後で行う処理として登録する機構をボトムハーフハンドラ (bottom half handler) と言い，定期的にカーネル内で実行される．

デバイスドライバは一般にOS起動時にメモリ上 (カーネル空間) に読み込まれる．デバイスドライバのユーザプログラムからのアクセスに対応する部分は，システムコールを介して実行される．デバイスドライバは，ユーザプログラムがシステムコール `open` を呼んだ時に実行するべき関数をカーネルに登録してある．デバイスドライバの中で，ユーザプログラムからシステムコールを介して呼び出されるルーチンを，Linuxではトップハーフルーチンと言う．主なトップハーフルーチンは，`open`，`close`，`read`，`write`，`ioctl` に対応するものである．

ボトムハーフルーチンは，デバイスドライバの中で，物理的なデバイスにアクセスする役割を担うが，この中では，`inb` や `outb` により物理アドレスをしていして，バスに接続された各デバイスの制御装置のレジスタにアクセスする．

デバイス制御装置からの割り込み (IRQ: interrupt require) を受けてOSから起動される割り込みハンドラの登録は，デバイスドライバ読み込み時に行われる．

## 1.4 デバイスドライバの危険性

デバイスドライバはカーネル空間で動作するので，システム全体に不具合を及ぼすこともできるし，効率の悪い書き方 (例えば，無駄にビジーウェイトするような書き方) をしてあればCPUを占有し他のプログラムの実行に支障をきたす場合もある．

また，デバイスドライバが利用するメモリ領域も，ページングされないカーネル空間に取られるので，万が一デバドラがメモリリークを起こすような場合，利用できる物理メモリが少なくなっていってしまう．通信デバイスなどのデバイスドライバの場合，送受信データをためておくメモリ領域 (バッファ) が必要になるが，この場合のバッファもページングされないカーネル空間に確保される．

[1] スカジーと読む

デバイスドライバは，ユーザプログラムのようにシステムコールやライブラリコールを使用することができない．動的メモリ確保は `malloc`, `free` でなく，`kmalloc`, `kfree` を用いる．`printf` の代わりは `printk` であり，syslog にも表示される．

## 1.5 DMA

データの流量が多い場合は，割り込みでのデバイスとのやり取りは，割り込みの頻度が高くなりすぎて破綻する．データ流量が多いデバイス (ディスク，高速ネットワーク等) は，DMA(direct memory access) を用いてデバイスとメモリの間でデータを直接やりとりすることで，CPU の使用を抑える．DMA で一定量のデータを転送し終わったタイミングなどで割り込みを用いる．

## 1.6 ローダブルモジュール

デバイスドライバは OS 動作中に動的にロード，アンロードする機構を備えた OS もある．Linux では，このようなデバイスドライバをローダブルモジュールと呼ぶ．デバイスドライバの動的ロードが可能だと，システムリソースの利用効率が向上する．また，カーネルを再構築する必要性が大幅に減る．

Linux で，デバイスドライバをロード，アンロードするコマンドは，それぞれ `insmod`，`rmmod` である．現在カーネルに組み込まれているドライバのリストは，`lsmod` で表示される．`insmod` コマンドによりカーネルに組み込まれる際には，トップハーフルーチン，ボトムハーフルーチンの登録が行われる．

## 1.7 デバイスファイル

UNIX 系の OS では，デバイスへのアクセスはデバイスファイルを介する．Linux ではデバイスファイルは，`mknod` コマンドにより作成できる．例えば，`mknod /dev/hello c 42 0` とすると，メジャー番号 42・マイナー番号 0 のキャラクタデバイス`/dev/hello` を作成する．

基本的には，一つのデバイスファイルは一つのデバイスに結び付けられている．同じ種類のデバイスを複数同時に利用する場合，たいていデバイスファイルは複数になるが，デバイスドライバは同じものが使用されるのが一般的である．

## 1.8 /proc ファイルシステム

Linux では，システムリソースに関する様々な情報を，`/proc/`以下の仮想的なファイルを見ることで知ることができる．`cat /proc/interrupts` とすれば，ハードウェアの割り込み番号とその割り込みを使用するデバイスが一覧表示される．`cat /proc/cpuinfo` とすれば，そのコンピュータの CPU が何なのかがわかる．`ls /proc` として他に観察できるリソースの種類と意味を調べてみよう．

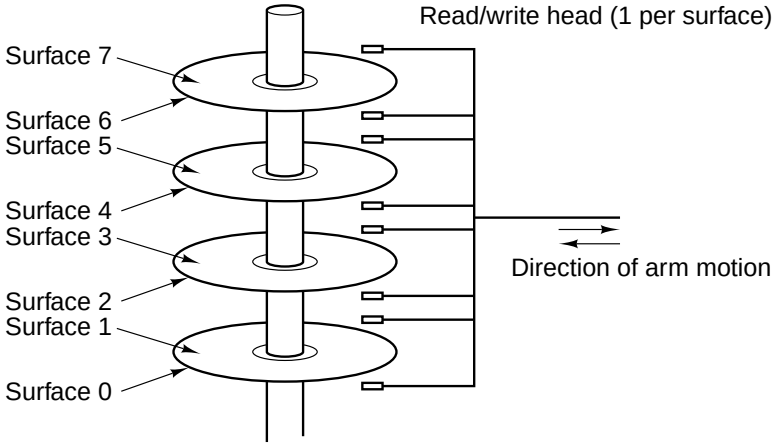


図 1: HDD の構造 (4 枚のディスク (プラッタ)，8 つのヘッド)

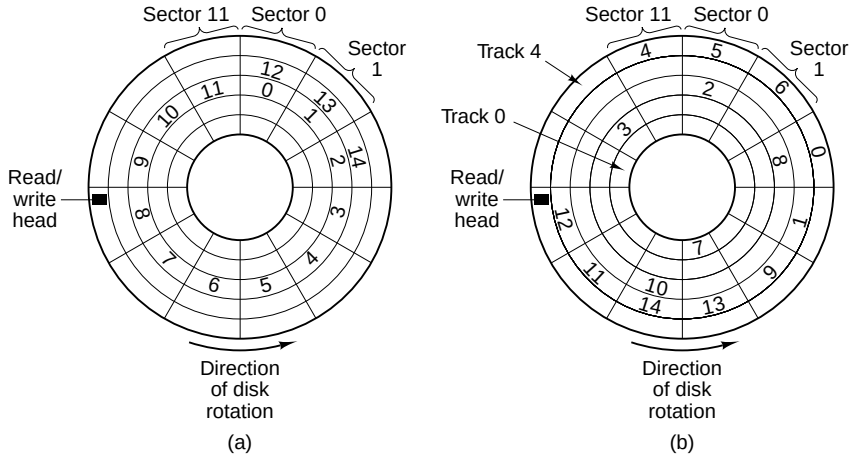


図 2: HDD のディスクの構造 (セクタ (sector)，トラック (track))

# 2 HDD の構成

## 2.1 HDD の物理構成

HDD(hard disk drive) は，1 枚以上のディスク (プラッタ; platter) から成り (図 1)，プラッタの片面は同心円状のトラックに分けられ，各トラックは複数のセクタに分けられる (図 2)．同一半径のトラックの集合を，シリンダと呼ぶ．BIOS を経由して IDE[2] の HDD にアクセスする際は，ヘッド番号，トラック番号，セクタ番号を指定してアクセスする．

IDE では，ヘッド 4 ビット，セクタ 6 ビット，シリンダ (トラック)10 ビットで指定するので，最大で $16 \times 63 \times 1024 \times 512 = 528482304$ バイト (504MB) [3]の HDD までしかアクセスできなかった．

[2]IDE は integrated device electronics の略．なお，10 月の課題にあったキーワードの IDE は，integrated development environment(統合開発環境) の略である．

[3]1 セクタは 512 バイト．セクタ 6 ビットがセクタ数 64 でなく 63 までなのは，0 から数えず 1 から数えているためである．これは初期の BIOS プログラマのミスだろうと言われている．

EIDE(extended IDE) では，LBA(logical block addressing) と呼ばれるアドレッシング方式 (セクタ番号のみで指定する) がサポートされ，504MB の壁が無くなった[4]．EIDE では，HDD 以外に CD ドライブ等も接続できるようになり，接続可能数も 4 になった．4 つの接続は，プライマリマスタ，プライマリスレーブ，セカンダリマスタ，セカンダリスレーブという四つがあり，スレーブはマスタが接続されていないと認識されない．IDE(EIDE, ATA, パラレル方式の ATA[5]) のケーブルは 40 ピンのフラットケーブルである．プライマリかセカンダリで 2 本のケーブルがある．1 本のケーブルにはマスターとスレーブの二つの HDD(や CD ドライブ) を接続できる．HDD にはマスターかスレーブかを設定するジャンパがある．

40 ピンの接続は信号線が並ぶパラレル形式のバスだが，パラレルバスの高速化は信号の干渉のため難しさがある．信号線が 1 本のシリアルバスの方が差動信号[6]によって高速化しやすく，近年は高速なシリアルバスが良く使われる (USB や IEEE1394(=iLink=FireWire) も差動信号を用いる)．パラレルの IDE( ATA) に変わる最近の HDD の高速接続インタフェースとして，信号線を差動信号のシリアルにした SATA(serial ATA) が使われるようになってきている．

## 2.2 RAID

HDD は，回転数に物理的な限界があることや，ヘッドが目当てのセクタに到達するまでの不定時間があることなどから，読み書き速度の限界がある．RAID (redundant array of inexpensive disks) は，HDD を安価に高速・大容量化するための方法として，Patterson[7]らによって提案された．図 3 は，RAID0 から RAID5 を示す．それぞれの特徴・特長を理解しよう (課題 2)．キーワード:ストライピング，冗長性．

## 2.3 HDD の論理構成

HDD の論理的構造 (ソフトウェアからの見え方) は，一列の非常に長いアドレス空間の記憶装置である．図 4 は，HDD の論理構造である．先頭には 512 バイトの MBR(master boot record) があり (図 5)，残りの空間には，空き領域の他に，0〜4 つの基本パーティション (primary partition) 又は 0〜4 つの基本パーティションと 1 つの拡張パーティション (extended partition) がある．拡張パーティションは内部に任意の数の論理パーティション (logical partition) を作ることができる．論理パーティション 1 つにつき，1 つの EPBR(extended partition boot record; 拡張パーティションブートレコード) がある．

MBR には小さなプログラム領域とパーティションテーブルがある．パーティションテーブルのエントリが 4 つなので，基本パーティションと拡張パーティションを合わせて 4 つまでしか作ることができない．EPBR は MBR と同様の構成だが，パーティションテーブルはエントリ 2 が次の論理パーティションの情報を持ち，エントリ 3 と 4 は使用されない．

---

[4] LBA に対し従来式を CHS(cylinder, head, sector) とも言う．

[5] これらの違いを各自調べてみよう．ATAPI, BigDrive, ATA-3, ATA/ATAPI-6 等色々ある．

[6] 差動信号による通信では，2 本の線に逆位相の信号を通す．この 2 本の差を取ることで，両信号線に同時に付加されるノイズ (コモンモードノイズ) の影響をほとんど受けずに通信できる．

[7] 初回 (10 月 6 日) に配付した資料に，参考書として，D.Patterson and J.Hennessy: Computer Organization & Design, 2nd Ed. (邦訳: コンピュータの構造と設計 第 2 版（上）（下），日経 BP) を示したが，この本の作者が RAID の提唱者である．さらに RISC(/CISC) の概念の提唱者でもある．

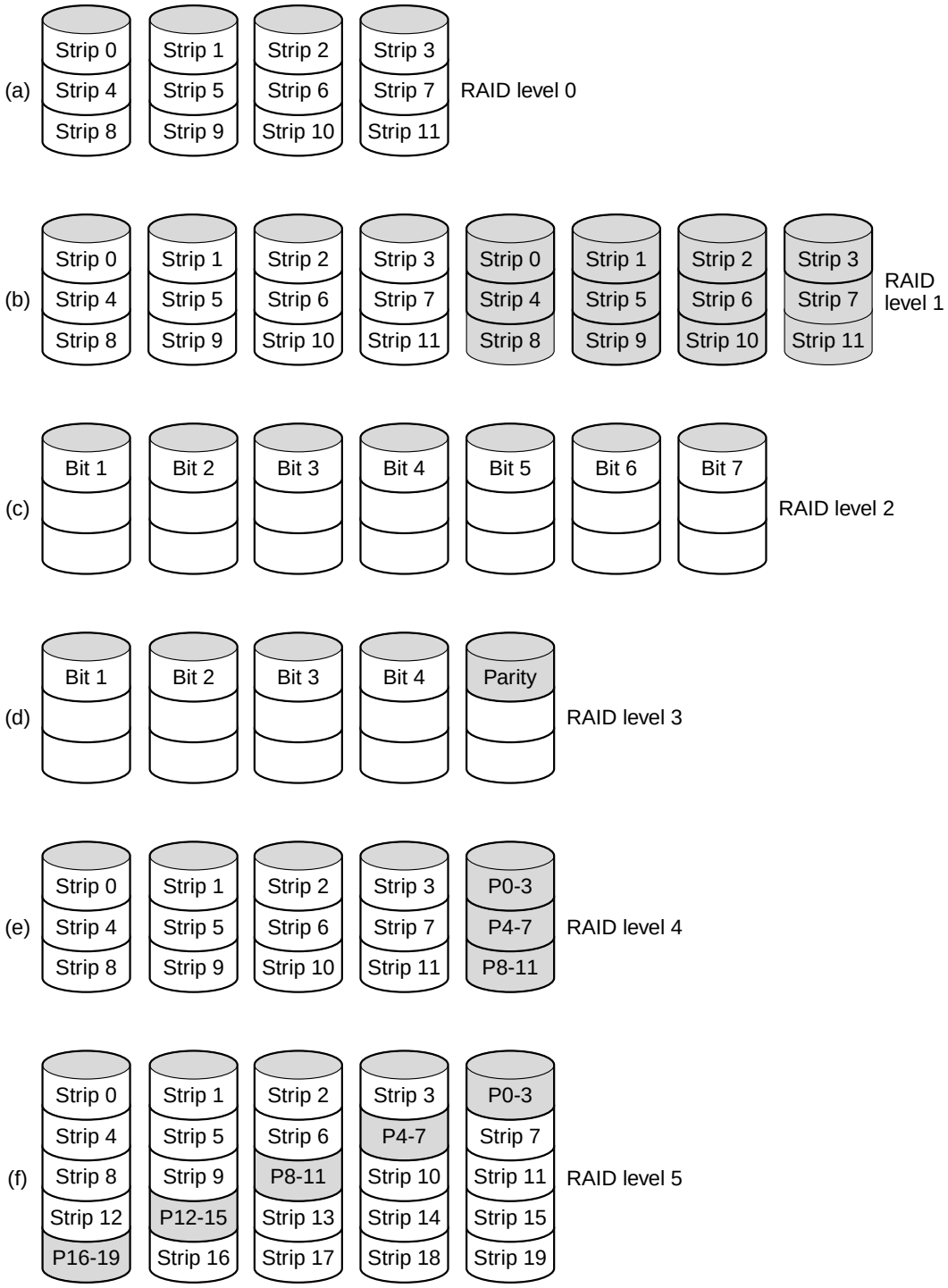


図 3: RAID0 から RAID5．バックアップとパリティは影つきで示してある．

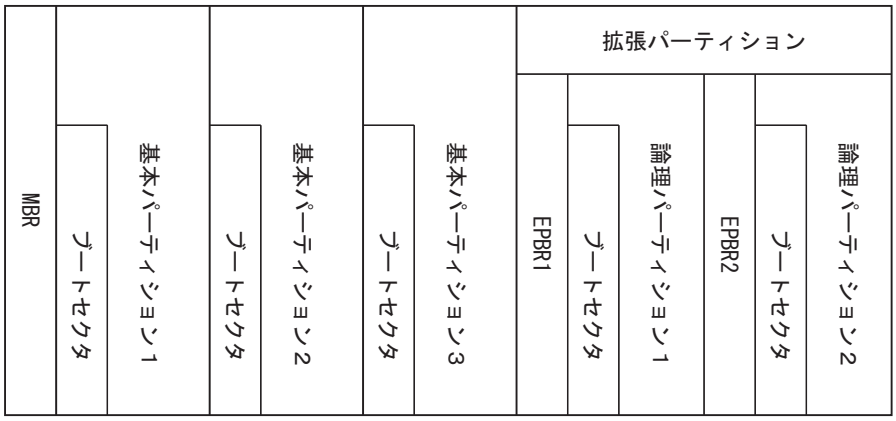


図 4: HDD の論理構造

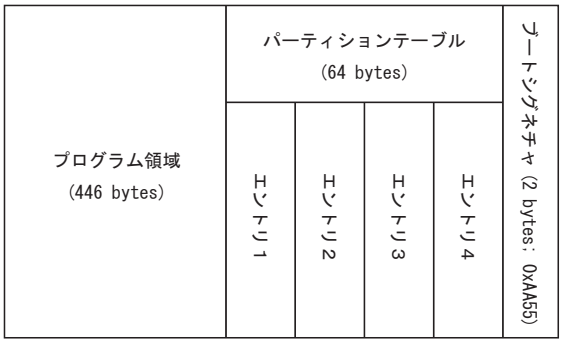


図 5: MBR: master boot record

# 3 OS が起動するまで

## 3.1 PC の OS の HDD からの起動

一般の PC(IBM PC/AT 互換機)[8]の電源投入後，OS が起動するまでの流れを確認しよう．

1. BIOS (basic input output system) の読み込み・実行
   マザーボード上の ROM から BIOS を読み込む．BIOS はキーボード・画面表示・ディスクドライブ等を扱う最小限のプログラム．OS はこれらのデバイスにアクセスする時に BIOS を経由する場合としない場合がある (最近は BIOS を介さずにデバイスにアクセスする場合が多い)．

2. ブートデバイスの決定
   どのデバイスから OS をロードするかを決める．候補は，フロッピー，光学ドライブ (CD,DVD)，ネットワーク，HDD 等である．優先順位は，BIOS の設定で変更できる．
   ブートデバイスが決まると，BIOS の機能を使って，そのデバイスから OS 等のプログラムをメモリ上に読み込み，実行を開始する．

[8]PC/AT 互換機とは，IBM が 1984 年に出したパソコンと互換性のあるパソコンの総称．パソコン規格のデファクトスタンダード (de facto standard; 事実上の標準; 業界標準) である．

3. MBR (master boot record)
   HDD がブートデバイスとして決定された場合，まず MBR をメモリにロードし，実行を開始する．MBR は，HDD の先頭から 512 バイトの領域で，プログラム領域[9]，パーティションテーブル，ブートシグネチャ，の三つが入っている．ブートシグネチャは 2 バイトで 0xAA55 という値である．パーティションテーブルは 16 バイト× 4．IPL は残りの 446 バイトに入れられた，小さなプログラムである (図 4).

4. OS のロード・起動
   MBR のプログラム領域に入っている IPL は，そのまま OS をロードし起動するタイプのものもあるが，IPL がさらにブートローダと呼ばれるプログラムをブートセクタ等から読み込み，ブートローダが OS をロードし起動する場合もある．

MBR に入っているブートローダ (IPL) には，Windows 用のブートストラップローダや，Linux 向けの LILO，汎用の GRUB(grand unified bootloader) 等がある．Windows を起動する PC では，MBR にブートストラップローダと呼ばれるプログラムが入っていて，ブートストラップローダが NTLoader というブートローダを起動し，NTLoader が OS(WindowsNT 系 (NT,2000,XP)) をロードし起動する．

Linux を起動する PC では，MBR に LILO か GRUB が入っている．LILO はファイルシステムを認識しないため，`/sbin/lilo` の実行時に，OS 本体が HDD のどのセクタから始まる領域に物理的に入っているかを調べ，記録して，MBR に LILO と一緒に書き込む．`/sbin/lilo` を行った後に，カーネルのファイル (`/boot/vmlinuz` 等) をコピーするなどして HDD の別の場所に移すと，記憶した物理セクタが間違っている形になり，そのカーネルを起動できなくなる．そのような場合には，もう一度`/sbin/lilo` を行う．

Linux では，近年はブートローダとして GRUB がよく使われるようになっている．GRUB は，ファイルシステムを理解できるので，HDD の物理的なアドレスの代わりに，ディレクトリ名.. ファイル名の形で，起動するカーネルを指定できる．複数の中から起動するカーネルを選択するような場合も，/boot/grub/menu.lst を参照してメニューを用意してくれる．万が一カーネルのパス・ファイル名を忘れてしまっても，簡単なシェルも持っているので，ファイルを探して指定することもできる．

## 3.2 自分の PC の MBR を見る

自分の使っている PC の MBR がどうなっているのかを見てみよう．下のようにすると，HDD の先頭の 1024 バイトを見ることができる．ブートシグネチャ(0xAA55) はちゃんとあるだろうか？

```
Linux の場合：
% sudo dd if=/dev/hda | head --bytes=1024 > hdd-head
% hexdump hdd-head | less
Cygwin の場合：
% dd if=/dev/sda | head --bytes=1024 > hdd-head
% dump hdd-head | less
```

[9] ブートローダが入っている．ブートローダは，IPL(initial program loader)，ブートストラップローダ，マスターブートローダ等とも呼ばれる．

### 3.3 SH-Linux

最近は SH のボードマイコンにも，Linux が搭載されている．SH-Linux のブートローダ (IPL) は，ボード上のフラッシュメモリに保存されている．この IPL は，新部裕氏が作成した sh-ipl+g を，独自に拡張したものが用いられており，モニタプログラムをロード・起動するか，Linux をロード・起動するかを選択できるようになっている．

SH-Linux は，コンパクトフラッシュからロードされるので，SH-Linux のカーネルを入れ替える場合はノート PC 等にコンパクトフラッシュのカードを挿して `mount` し，カーネルを含むシステムをコピーし，`lilo` を実行することで，インストールできる．

では，SH ボードマイコンの IPL はどのようにインストールするか？(アドバンスト課題)

## 4 実世界とのインタフェース

コンピュータシステムに接続されたデバイスや，独立して動くロボット制御器，エンジン制御器等の様々な組込システムは，実世界とのインタフェースを持つシステムであるとも言える[10]．実世界とのやり取りに特有な様々な要素がある．

### 4.1 外界からの入力

センサは外界の状態をモニタし，コンピュータが扱える形に変換する．センサの取得する情報は一般に多次元の状態量であり，各次元ごとの値は連続的な値をとる．コンピュータはデジタルという不連続な状態量からなるシステムなので，センサの情報をデジタル化することでコンピュータへの入力となる．AD 変換器は連続量を多値の離散量に変換する．スイッチは連続量を二値の離散量に変換する．

AD 変換器には，積分型 (放電時間を計る)，カウンタ制御式 (カウンタで小さい値から順次比較してゆく)，逐次比較型 (二分木探索式に比較してゆく) 等，様々な方法がある．また，変換する前に入力の電圧をホールドする回路 (サンプルホールド回路; コンデンサに貯める) も必要である．

ワンチップマイコンには AD 変換器が複数内蔵されたものもあるが，AD 変換機の回路は 1 つで，アナログマルチプレクサの回路を内蔵し複数チャンネルのアナログ信号を読み取れるようになっている場合が多い．マイコンでは，AD 変換した結果を保持するレジスタがあり，そのレジスタの値を読み出すことで，デジタル化されたデータが利用できる．

タッチスイッチのように入力を 1/0 で判定する装置の場合は，スイッチの状態が ON か OFF かの二値になるが，実際はスイッチが切り替わる瞬間に ON と OFF を高速で繰り返すことがある．このようなノイズなどを取り除くことも外界からの入力の処理に必要になる．

### 4.2 外界への出力

逆に，コンピュータのデジタル値を外界に出力するには，DA 変換器，デジタル IO，等を用いる．ここでも実世界特有の注意が色々とある．例えば，マイコンのデジタル IO(外部入出力ピン)

[10] ロボットを実世界情報システムと呼ぶことがあるが，このようなシステムでは，センサを通して実世界からの情報を入力し，アクチュエータを通して実世界に働きかける (情報を出力する)．その入出力の間には，生の実世界情報から様々な抽象度の情報を抽出し，それを元に判断し記憶し行動を選択し，その行動を生のモータ情報に具体化し，実行するという作業がある，人間は有機体からなる実世界情報入出力システムともみなせ，知能ロボットの究極の目標の一つは，人間の内部処理に近づくことでもある．

は，電圧は希望の電圧が出るが，電流は非常に少ない．そのままモータを回したりするような大電流を流すことはできないので，トランジスタ等の増幅装置 (モータドライバ) を使って大電流を流してモータを回す．

モータの回転数や電灯の明るさを調節する場合には，DA 変換されたアナログ信号や，PWM(pulse width modulation) が用いられる．モータの回転方向を切り替えるためには H ブリッジ等の回路が用いられる．

ラジコンサーボモジュールは，三本の線に，パワー，GND，信号を接続する．ラジコンサーボの信号線には，特殊なデューティ比の PWM 信号が与えられ，デューティ比に応じた出力角度になるように制御される．

# 5 組込システム

組込システム (embedded system) とは，各種の機器に組み込まれてその制御を行うコンピュータシステムである (曖昧な定義だが)．コンピュータに接続されるデバイスを制御するコントローラも組込システムである．世界のプロセッサの 90% 以上は組込システムに使われていると言われる．組込システムの例は以下のように挙げるときりがない: AV 機器，家電，情報・娯楽，PC 周辺機器，OA 機器，通信機器，運輸機器，工業・その他．

## 5.1 組込システムの特徴

組込システムには，省消費電力に対する強い要求がある．電池で駆動するシステムも多いため，電池駆動時間や省エネに対する要求が強い．超低消費電力モードを持つマイコン[11]などもある．携帯機器など，小型の機器も多い．また，低価格化に対する強い要求がある．これは製品の競争力のためである．そのために，必要最小限のスペックを見積もり，搭載するコンピュータを選定する．さらに，これらの要求 (低消費電力・小型化・低価格化) を満たすために，部品点数の削減が進み多機能ワンチップ化が進んでおり，近年では多チップ 1 パッケージ化 (SiP: system in package) も行われるようになっている．

組込システムに利用されるコンピュータは，周波数:数 MHz〜数百 MHz，メモリ:128 バイト程度〜数十 M バイト，電力:数 mW〜数 W というような範囲で，比較的広範囲である．

また，ROM やフラッシュメモリ等の不揮発性メモリが使用されている．様々なモジュールを内蔵している場合が多く，AD/DA，RAM，シリアル，USB 等の通信用インタフェース，DSP(digital signal processor) 搭載など，様々な機能を持つプロセッサがある．

組込システムのプロセッサは，このようにバリエーションが非常に広い．個別の機器に必要な性能・機能を備えることを想定し，汎用性より個別対応のプロセッサが非常に多いためである．

## 5.2 組込システムのソフトウェアの特徴

組込システムのソフトウェアは，コードサイズは比較的小さい．個々の処理は比較的単純．割り込み（イベント）応答速度が重要．開発期間短い．使用期間長い．バージョンアップは難しい．CPU 能力の限界付近で動かすようにする．個別の製品・対象に特化したプログラムになりやすい．等の特徴がある．

[11] マイコンというと，PC 等に使用されるプロセッサより小規模の，組込用プロセッサをさす場合が多い．

## 5.3 組込システムのソフトウェア開発

組込システムのソフトウェア開発は，ごくわずかな計算機資源の中で行われなければならないケースが多かったが，最近は組み込みプロセッサの性能も上がり，ソフトウェア技術も近代的になってきている．

- 非力なプロセッサ・わずかなメモリ ⇒ 昔の PC 並みの性能のプロセッサ性能・メモリ容量
- アセンブラ多用．C 言語+割り込み ⇒ オブジェクト指向言語・再利用可能なソフトウェア
- 貧弱なデバッグ環境 ⇒ ICE (In Circuit Emulator)
- 職人芸 ⇒ OS の導入

## 5.4 組込 OS

組込機器用の OS は，必要な機能を最小限に絞った OS である場合が多い．組み込み用に特化した OS[12] は，豊富なコンフィギュレーション機能を持ち，必要とされる機能のみを組み込んだ OS を使用機器毎に構築する．最小構成〜最大構成まで，多様なコンフィギュレーションは，カーネルサイズ，必要メモリ，ライブラリ選択，デバイスドライバ選択，ファイルシステムのサイズ，等々様々な OS 要素をコンフィギュレーション可能である．

過去の組込プログラミングのスタイルは，アセンブリ言語や，割り込みを利用した複数タスクの並列実行など，職人芸的に書かれてきたプログラムが多かった．組込プログラムの開発効率・生産性を向上し，ソフトウェアの可読性・保守性・拡張性を向上するためには，組込 OS が大きく役に立った．組込システムは，通常のコンピュータシステムと比較して，コンピュータ資源 (CPU やメモリ) のうち OS のために利用できる割合が少ないため，オーバーヘッドが小さくカーネルサイズも最小限の物が求められる．

## 5.5 $\mu$ITRON

TRON プロジェクト[13]は，1984 年から続くリアルタイム OS のプロジェクトである．オープンアーキテクチャで,「弱い標準化」という概念が特徴である．TRON プロジェクトの OS は，実装されたソフトウェアとしてではなく，OS の仕様を策定するという形で公開されている．その仕様書を見て，第三者が自由に実装を行っている．商用の OS として販売されているものもあれば，フリーで公開されているものもある．また，自社内で使用するために実装し外部には公開していない実装が多いとも言われる．

$\mu$ITRON は，TRON プロジェクトのうちの，組込機器用リアルタイム OS であり，日本の産業機器・組込機器の非常に広範囲にわたって，利用されている OS である.「弱い標準化」という概念は，具体的なハードウェアやソフトウェアは規定せず，インタフェースのみを規定することで，過度の抽象化を避け OS のオーバヘッドを極力減らす，ハードウェア性能・性質に応じた最適な実装方法を取れるということを目指した概念である．

[12] 組込 OS の例 (クリックで URL が開く): uClinux，eCos，ITRON(Industrial TRON)
[13] `http://www.tron.org/`(TRON プロジェクト), `http://www.assoc.tron.org/jpn/document.html`(ドキュメント)

# 6 課題

課題は，11 月 17 日 (月) 朝までに ikuo-soft3-08@jsk.t.u-tokyo.ac.jp 宛てに提出する．メールのサブジェクトは,「`soft3-kadai:20081110:xxxxx`」(xxxxx は学生証番号) とする．その他，質問・要望・意見などがあれば，遠慮なく ikuo@jsk.t.u-tokyo.ac.jp にメールをください．

## 6.1 課題 1

`/proc` ファイルシステムにアクセスし，そのうちのファイルを 4 つ以上選択し，出力結果を見てそれぞれのファイルの意味を述べよ．Linux で行うのが良いが，Cygwin でも可．

## 6.2 課題 2

RAID のレベル (RAID5 等) のそれぞれの特徴・特長を，違いがわかる形で簡単に説明せよ．RAID5 では，ディスクが 1 台故障しても，情報を失わずに済む．パリティから元の情報を復元する手順を具体的に説明せよ．

## 6.3 アドバンスト課題

SH ボードマイコンの IPL はどのようにインストールするかを説明せよ．

## 6.4 オプショナル課題 1

コンピュータに接続される入出力デバイスを一つ以上挙げ，そのデバイスの物理的電気的な挙動・操作とコンピュータ内部のプログラムの動きとの関係を簡単に説明せよ．
入出力デバイスの例：USB マウス，ネットワークカード，テレビのリモコン

## 6.5 オプショナル課題 2

タッチスイッチのように入力を 1/0 で判定する装置の場合は，スイッチの状態が ON か OFF かの二値になるが，実際はスイッチが切り替わる短い時間に ON と OFF を高速で繰り返す状態が発生することがある．これを何というか．また，これを防ぐ (除去する) ための回路の例を挙げよ．