システムプログラム概論

入出力（I/O）制御

第6講: 平成20年10月22日 (水) 2限 S1教室

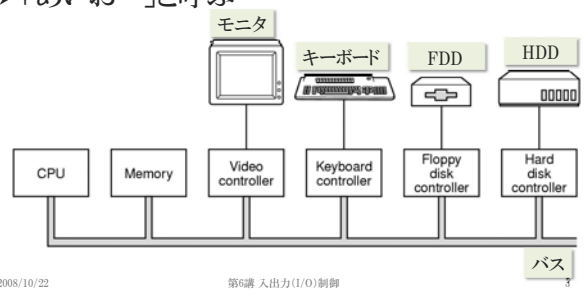
中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

今日の講義概要

- 入出力デバイスのハードウェア
- 入出力デバイスの制御
- 入出力デバイスのソフトウェア

入出力(I/O)デバイス

- I/O = Input/Output = 入出力
- 「あいおー」と呼ぶ



I/O デバイスのハードウェア

I/O デバイスの種類

- **ブロックデバイス**
  - 固定サイズのブロック単位で読書
  - ランダムアクセスが可能

  例)ディスク
- **キャラクタデバイス**
  - 文字列(バイトストリーム)として読み書き
  - シーケンシャルアクセス

  例)プリンタ, ネットワーク I/F

参考

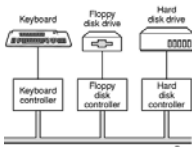
PCI Express	2.5Gbits/sec
10G Ethernet	10Gbits/sec

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

様々なデバイス，バス，ネットワーク

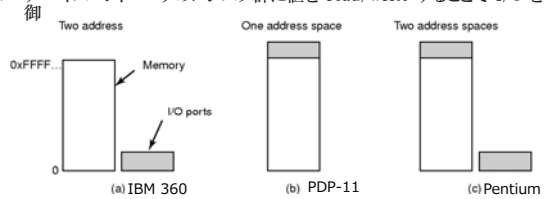
デバイスコントローラ

- I/Oデバイスの構成
  - 機械的な部品(デバイス自身)と電子的な部品(コントローラ)から構成
- **デバイスコントローラ**
  - 複数のデバイスを同時にコントロール可能
  - 例) IDE コントローラは 2 つ，SCSI コントローラは 7 つのデバイスを同時に制御できる
- デバイスコントローラの仕事
  - ビット列(bit stream)をデータの固まり(ブロック)に変換する(コントローラ内にバッファリングする)
  - 読み取ったブロックに対し必要に応じてエラー訂正を行う
  - ブロックのデータを主記憶にコピーする



Memory-Mapped I/O

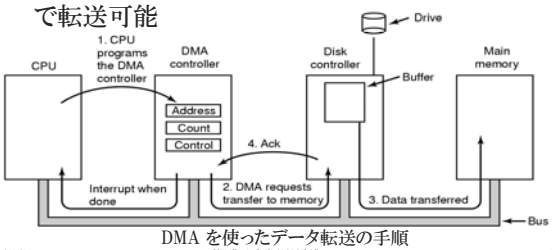
- デバイスコントローラのレジスタ群に値を read/write することで I/O を制御



- レジスタへのアクセスの方式は次の 3 つがある
  (a) メモリアドレス空間とは別に I/O のアドレス空間を用意
  (b) Memory-mapped I/O(I/O 空間をメモリ空間の一部にマップ)
  (c) Hybrid(上記 2 つの混合)
- それぞれ一長一短あるが，(c) が主流

ダイレクトメモリアクセス(DMA)

- I/O デバイスとのデータのやりとりに CPU を使うのはもったいない
  - DMA: CPU を介さず I/O デバイスと主記憶の間で転送可能



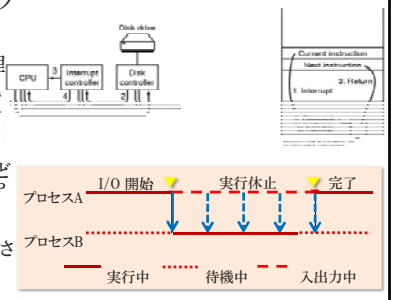
DMA を使ったデータ転送の手順

入出力の完了を知る方法

- ポーリング(polling)と割込み(interrupt)
  - ポーリングでは，CPU が一定間隔毎に入力を確認する
  - 割込みでは処理の終了を割込みによって知らせる
  - ブロックデバイスはポーリング，キャラクタデバイスでは割込みが使われる

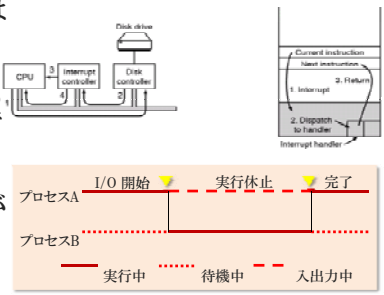
ポーリングとは？

- デバイスコントローラに入出力動作の完了を示すフラグを用意
- CPU は入出力操作と並列で他のプロセスを処理
- 周期的にこのフラグを見に行き，動作完了をチェックする
  - 周期が短いと
    → CPU 時間のほとんどがフラグチェックに費やされてしまう
  - 周期が長いと
    → 入出力装置が放置される時間が長くなる

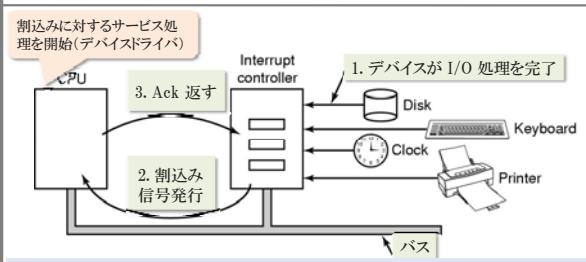


割込みとは？

- I/O デバイスのデータ転送速度は比較的遅い
- デバイスからのデータが到着するまでの間，CPU では別のプロセスの処理を行いたい
- まとまったデータが到着後，CPU で行っている処理を割込んで処理



割込み処理の手順



- 各 I/O デバイスはバス上の割込み用信号線を使用し割込みを通知
- コントローラは IRQ(Interrupt ReQuest)番号をバス経由で CPU に通知
- CPU は interrupt vector に登録された，割込み処理ルーチン(Interrupt service procedure)を実行し，処理完了後 Ack を返す

割込み発生時の CPU の処理

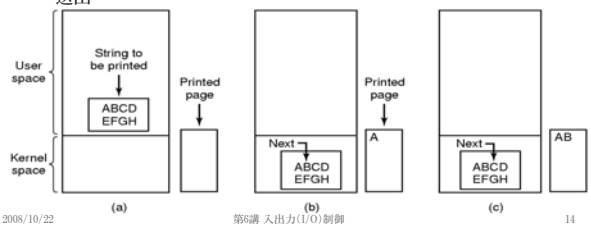
- 割込みが発生
  1. CPU が割込み発生を検出
  2. CPU がカーネルモードへ自動的に移行
  3. OS 内の割込み処理手続きへジャンプ
  4. OS 内手続きは，現在実行中のプロセス(レジスタ等)を退避
  5. 割り込みの種類を判定して処理を行う
  6. OS が，CPUモードをユーザモードに変更
  7. プロセスの処理に戻る
- 割込み処理への割込み
  - 追加割込み禁止
    - 割込み処理中は，他の割り込みを一切受け付けない
  - 選択的割込み許可
    - 現在処理中のものより優先度の高いものだけ受付

I/O の処理方式

- Programmed I/O(PIO, プログラム I/O)
- Interrupt-driven I/O(割り込み駆動型)
- I/O using DMA(DMA)

Programmed I/O (1)

- 各デバイスとメインメモリの間のデータ転送を CPU が管理する転送方式
  - ユーザプロセスから文字列 "ABCDEDGH" を印刷する場合の手順
    - **open** システムコールを呼ぶ → プリンタが利用可能になるまでブロック
    - **print** システムコールを呼ぶ → OS がカーネルのバッファにコピー
  - プリンタの状況をチェックしつつ OS が一文字ずつプリンタにデータを送出



Programmed I/O (2)

busy waiting

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
    while (*printer_status_reg != READY) ; /* loop until ready */
    *printer_data_register = p[i];         /* output one character */
}
return_to_user();
```

print システムコール呼び出し時に OS が実行するコード

- マルチタスク OS では非効率

割込み駆動型 I/O

一文字印刷が完了するたびに割込みにより呼び出される

```
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();
```

他のプロセスに CPU が渡される

print システムコール (a)呼出し時に実行されるコード

```
if (count == 0) {
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

割込み処理ルーチン (b)

- 印字速度が 100 文字/秒の時, 1 文字あたり 10 ミリ秒の待ち時間
- 待ち時間を他のプロセスの実行に有効利用できる

DMA を使った I/O

DMA の処理完了時に割込みが起こり, 呼び出される

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();
```

(a) **print** システムコール呼び出し時に実行されるコード

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

(b) 割込み処理ルーチン

- バッファ内の文字列全部の印刷を(CPU から見て)1 回の割込みで実現できるため, 効率が良い
- 実際には, CPU の代わりに DMA が Programmed I/O 方式で印字

I/O ソフトウェアの目的 (1)

- デバイス非依存性
  - 任意のデバイスにアクセスできるようプログラムが書ける

  例) sort < input > output(FDD, HDD, CD-ROM など指定可)
- 一様な名前付け
  - ファイルやデバイスの名前を, デバイスの種類に依存せずにつけれる
- エラー処理
  - ハードウェアに近い側で(エラー発生時: デバイス → ドライバの順に)解決

I/O ソフトウェアの目的 (2)

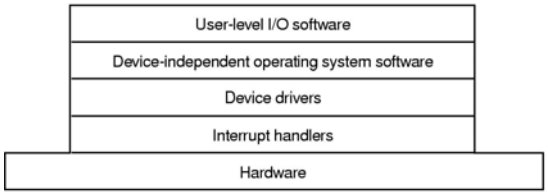
- 同期転送 vs. 非同期転送
  - 呼び出し側をブロックするタイプ　vs. 割込みで駆動するタイプ
  - ほとんどの I/O は非同期・ただし，プログラムを書く時は同期の方が楽
- バッファリング
  - デバイスから転送されるデータは，最終的な場所に直接保存できない

  例）パケット：受信してポート番号を見る → どのプロセスに渡すか分かる
- 共有型デバイス vs. 占有型デバイス
  - ディスクは共有型：多くのユーザから同時にアクセス可能
  - テープドライブは占有型：同じテープに多くのユーザから書き込み要求があったらうまく動作しない → スプーリング

I/O ソフトウェアの階層

- I/O ソフトウェアシステムの階層
  - 隣接した層の間には，利用可能な機能，インターフェースが明確に定義

  → 新たなデバイスへの対応が容易



割込みハンドラ

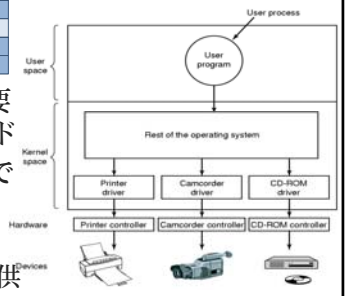
ユーザプログラム
デバイス非依存ソフトウェア
デバイスドライバ
割込みハンドラ
ハードウェア

- 割込みハンドラの仕事
  - 割込み種類の判別
  - 割込み処理ルーチン（デバイスドライバ）への制御の移行
- 最も単純な場合の制御
  - I/O を開始し，処理完了による割込みが起こるまでドライバをブロック
  - 入出力処理後，ドライバでの処理を再開（unblock）
- 実際にはもう少し複雑

デバイスドライバ

ユーザプログラム
デバイス非依存ソフトウェア
デバイスドライバ
割込みハンドラ
ハードウェア

- デバイスの制御に必要な専用プログラムコード
- OS がカーネルモードで実行
- 抽象度の高い read/write 機能を提供



デバイス非依存ソフトウェア

ユーザプログラム
デバイス非依存ソフトウェア
デバイスドライバ
割込みハンドラ
ハードウェア

- デバイス非依存ソフトウェアが提供する機能
  - デバイスドライバに対する一様なインターフェース
  - バッファリング
  - エラーの報告
  - 占有型デバイスの確保と解放
  - デバイス非依存なブロックサイズ
    - デバイス毎に物理的なブロックサイズは異なる
    - 物理的な違いを吸収し，全てのデバイスを同一ブロックサイズで読み書きできる機能を提供する

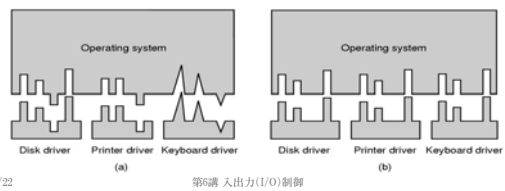
ディスクのセクタ等

一様なインターフェース

(a) 標準ドライバインターフェースがない場合
- デバイスごとにインターフェースが異なるとプログラムが大変

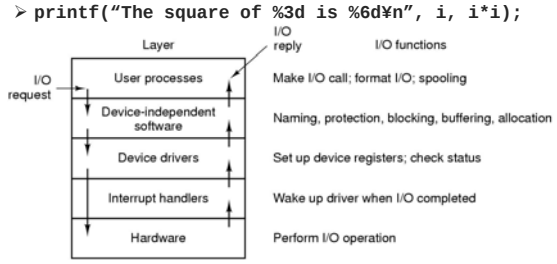
例）デバイスごとに操作が違う（open/close, connect/disconnect 等）

(b) 標準ドライバインターフェースがある場合
- ドライバ設計者，利用者のどちらも，インターフェースが同じなので，プログラムが楽



ユーザプログラムにおける I/O 処理

- I/O ライブラリ関数の例
  - `count = write(fd, buffer, nbytes);`
  - `printf("The square of %3d is %6d¥n", i, i*i);`



I/O system の階層と各層の主な機能

入出力制御のまとめ

- I/O デバイスのハードウェア
- 割込み制御，入出力制御
- I/O デバイスのソフトウェア