計算機システム

教授　石川　裕
助教　美添
連絡先：ishikawa@is.s.u-tokyo.ac.jp
　　yoshizoe@is.s.u-tokyo.ac.jp
本郷　理学部7号館505、507

計算機システム　第9回

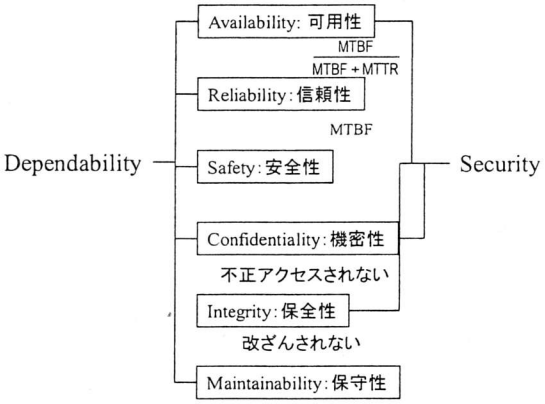
2010/12/20

講義予定

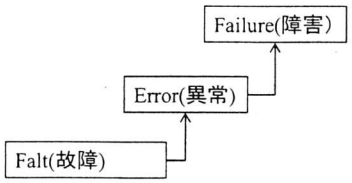
- 10月18日　第 1回　イントロダクション
- 10月25日　第 2回　コンピュータの歴史、コンピュータの構成
- 11月 1日　第 3回　机上の計算機　演算、アドレッシング
- 11月 8日　第 4回　机上の計算機　サブルーチン
- 11月15日　第 5回　番外編：探索アルゴリズム （美添）
- 11月22日　休み (駒場祭)
- 11月29日　第 6回　割り込み
- 12月 6日　第 7回　パイプライン処理＆メモリ階層
- 12月13日　第 8回　仮想メモリ
- 12月20日　第 9回　プログラミング言語、ライブラリ、OS
- 12月24日　第10回　I/O (Disk, Network)
- 1月14日　第11回　評価、ベンチマーク、まとめ、質問
- 1月17日　休講
- 1月24日　試験

ディペンダビリティ

• ディペンダビリティとセキュリティ



• 故障、異常、障害



• 手法

- Fault Prevention（障害予防）
- Fault Tolerance （耐障害）
- Fault Removal（障害除去）
- Fault Forecasting （障害予測）

A. Avizienis et al., "Basic concepts and taxonomy of dependable and secure computing," IEEE transactions on Dependable and secure computing, pp. 11-13, vol.1, no.1, 2001.

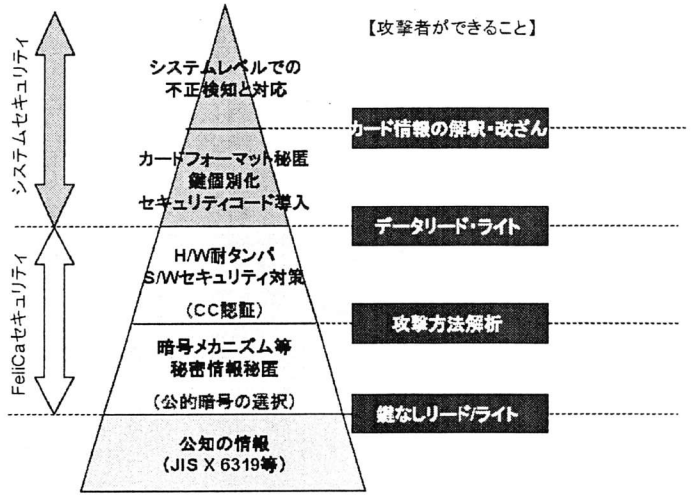
ソニー株式会社
B2Bソリューション事業本部・FeliCa企画開発部門
Chief Distinguished Engineer
要素技術開発部 統括部長 森田 直

SONY

出典：www.dvlsi.jst.go.jp/topics/09ws/09DVLSIWS%20morita.pdf

FeliCaのセキュリティ階層構造

FeliCa



出典：www.dvlsi.jst.go.jp/topics/09ws/09DVLSIWS%20morita.pdf

攻撃へのセキュリティ対策

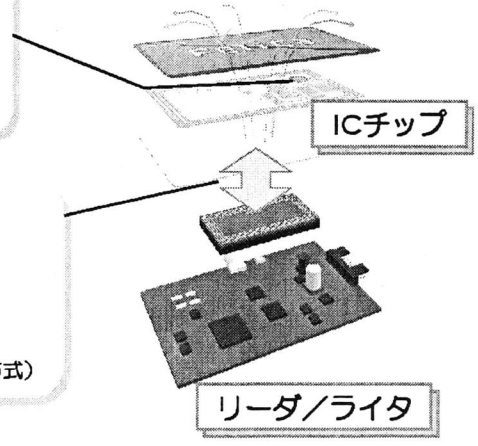
FeliCa

ハードウエア的な攻撃
- チップ解析
- 異常な電圧での動作　など

→ 耐タンパ技術

ソフトウエア的な攻撃
- 盗聴
- 不正なデータ改ざん
- コマンドの繰り返し　など

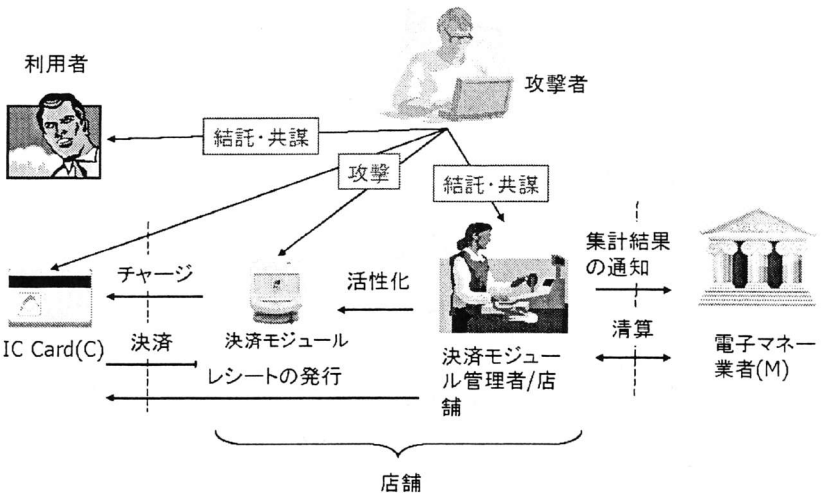
→暗号通信技術
（相互認証と暗号通信、鍵運用方式）



出典：www.dvlsi.jst.go.jp/topics/09ws/09DVLSIWS%20morita.pdf

想定するエンティティ

FeliCa



Rev.1.0 December 2009

41

出典：www.dvlsi.jst.go.jp/topics/09ws/09DVLSIWS%20morita.pdf

プログラミング言語

- コンパイラ型言語
  - Fortran, C/C++, …
- インタープリタ型言語
  - Java, Ruby, Perl, PHP, …
- 言語拡張
  - OpenMP, CUDA, HPF

ポインタとレジスタ間接

- C言語では、ポインタという概念がある
  - データを格納したメモリアドレス
- 機械語におけるレジスタ間接=ポインタ
  - レジスタの内容をメモリアドレスとする

C言語におけるポインタ

```
int     iarray[10];
short   sarray[10];
main()
{
    int    *ip;
    short  *sp;
    int    i;

    ip = iarray;
    for (i = 0; i < 10; i++) {
       *ip++ = i;
    }
    sp = sarray;
    for (i = 0; i < 10; i++) {
       *sp++ = i;
    }
}
```

int配列の定義

short配列の定義

int型を持つメモリ領域へのポインタ

short型を持つメモリ領域へのポインタ

iarrayのアドレスをipに代入

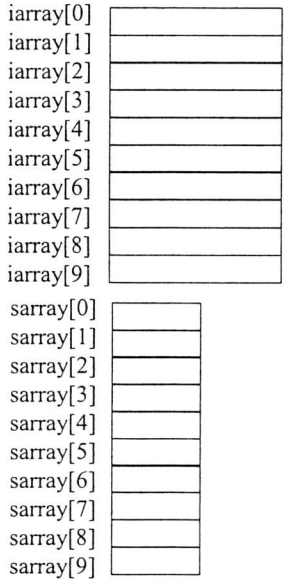
ipが指すメモリ領域
(ipの内容をメモリアドレスとして)
にiの値を格納し、ipの内容を+1する。
ここで+1とは、int型のサイズバイト分。
Sparcの場合、4バイト。

C言語における配列

```
int     iarray[10];
short   sarray[10];
main()
{
    int     i;

    for (i = 0; i < 10; i++) {
        iarray[i] = i;
    }
    for (i = 0; i < 10; i++) {
        sarray[i] = i;
    }
}
```

int配列の定義

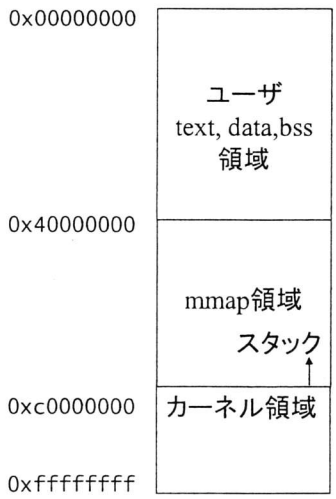
short配列の定義



メモリ領域

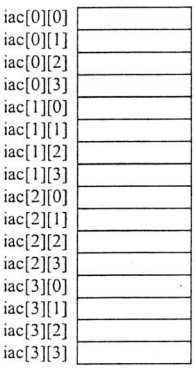
- グローバル変数
  - data (初期化されている領域)
  - Bss(初期化されていない領域)
- ローカル変数
  - スタック上に取られる
- 動的にメモリを確保した場合
  - mmap領域に取られる

```
static int   ia[10];                          /* bss */
static int   ib[10] = { 0,1,2,3,4,5,6,7,8,9 }; /* data */

main()
{
   void     *vp;
   printf("ia = %p¥n", ia);
   printf("ib = %p¥n", ib);
   vp = malloc(1024*1024);
   printf("vp = %p¥n", vp);
}
```

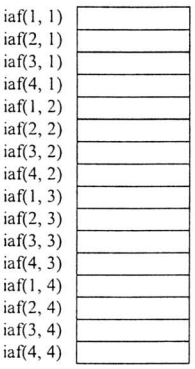


C言語vsFortran言語　2次元配列

```
int     iac [4][4];

for (i = 0; i < 4; i++) {
      for (j = 0; j < 4; j++) {
            iac[i][j] = i*j;
      }
}
```




```
integer     iaf (4, 4);

    do 20 j = 1, 4
        do 10 i = 1, 4
            iaf(i,j) = i*j
10      continue
20 continue
```



ライブラリ

- ライブラリとは？
  - ある機能をサブルーチンとして部品化し、いろいろなアプリケーションプログラムに利用できるようにしたもの
- プログラミング言語が提供している標準ライブラリ
- 数値計算ライブラリ
- グラフィックスライブラリ

オペレーティングシステムとは？

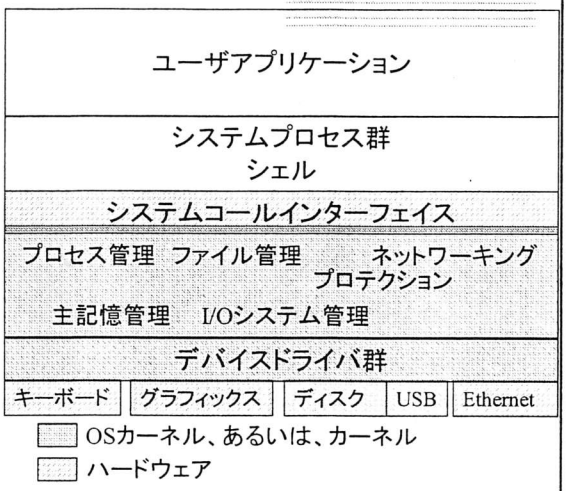
- ユーザとハードウエアの間で動くプログラム
- オペレーティングシステム(以降OS)の目的は、
  - ユーザプログラムを動かす
  - コンピュータを使いやすくする
  - 抽象化と仮想化
- コンピュータのハードウェアを効率よく使用する

OSの役目

- 仮想化＆抽象化
  - ハードウエア資源をそれ以上に見せる
    - 仮想メモリ
  - 機器の細かい操作の違いを隠蔽し、統一インターフェイスを提供する
    - open, read, write, close
- 資源管理
  - メモリ
  - ディスク
  - プロセッサ
  - ネットワーク等のI/O

システム構成要素

- OSカーネル(あるいは単にカーネル)
  - メモリに常駐するプログラムで、CPU特権モードで実行される。
    - プロセス管理、メモリ管理、ファイルシステム、ネットワーク管理、その他I/O管理
- システムプロセス
  - システム機能を実現しているプロセス
- シェル
  - OS機能を実行するためのユーザーインターフェースを提供
- ウインドウシステム
- コマンド群



- カーネルはCPUの特権モードで動いている（後述）
- システムプロセスはCPUのユーザモードで動いているが、特別な権限を持つ

システムコールと特権モード

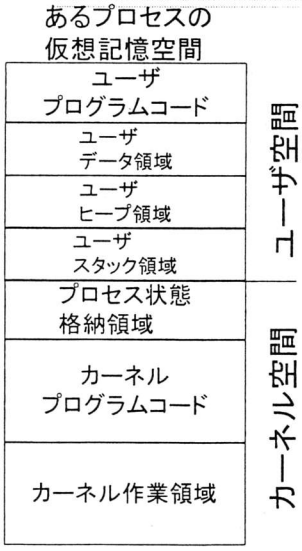
- ユーザモードから特権モードに移行
- 特権モードとは？
  - 危険な操作は特権モードだけで可能にする
    - 入出力デバイスの操作(入出力デバイスは、任意のメモリ領域にアクセス)
    - タイマ、MMU(Memory Management Unit)など共通システム資源の操作
  - 特権モードは、スーパバイザモードとも呼ばれる
  - 特権モードでないモードは、ユーザモード

システムコールの種類

- プロセス制御
  - fork, exec, exit, wait
- メモリ管理
  - break, mmap, swapon, swapoff
- ファイル管理
  - open, read/write, lseek, close, link, mkdir
- デバイス管理
  - ioctl
- 通信
  - socket
  - signal

プロセス

- プロセス
  - プログラムを実行する単位
    - 構成要素
      - プログラムコード
      - データ領域
      - プロセスの状態
        - » プロセッサの状態
- 沢山のプロセスを同時に実行
  - 実際には、ある時間間隔でプロセスを実行・中断し他のプロセスを再開・中断、を繰り返す
- プロセスごとに仮想空間を持っている
  - あるプロセスから他のプロセスのメモリにはアクセスできない
  - プログラムにバグがあり、プロセスが異常終了しても、他のプロセスに影響することはない
- プロセスの種類
  - ユーザプロセス
  - システムプロセス

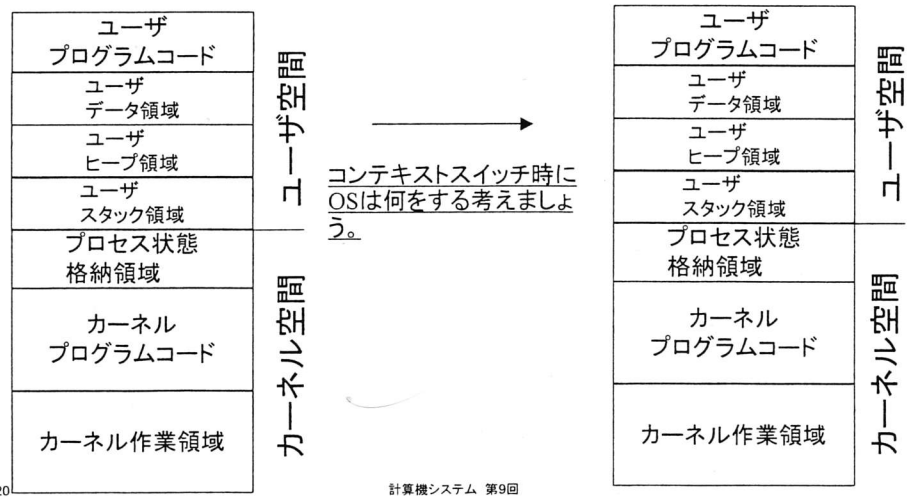


プロセスの種類

- 権限による分類
  - ユーザプロセス
    - 一般ユーザ
  - スーパーユーザプロセス
    - 特権機能が使える
      - 例：任意のユーザプロセスの停止、システム停止
- 形態による分類
  - システムプロセス、デーモンプロセス
    - オペレーティングシステムサービスを提供しているプロセスでスーパーユーザモード（CPUの特権モードとは違う）で動いているプロセスの総称
    - システムプロセスでユーザからは見えないところで動いているプロセスをデーモンプロセスとも呼ばれる
  - サーバプロセス
    - クライアント・サーバモデルに基づいたサービスを提供しているプロセスをサーバプロセスと呼ぶ
    - httpデーモン、httpサーバ、httpシステムプロセス

プロセス＆プロセス実行切り替え

- コマンド実行毎にプロセスが作られる。
- プロセス実行の切り替え。一般にプロセスコンテキストスイッチ、コンテキストスイッチと呼ばれる

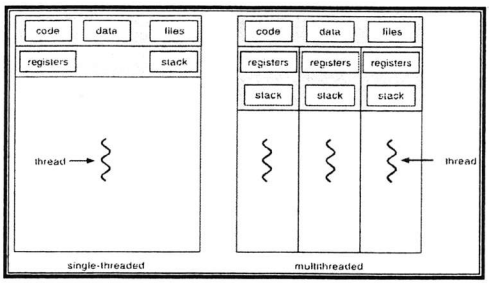


プロセススケジューリング

- 沢山のプロセスの実行のタイミングを制御する
  - リアルタイム性の実現 ⇒ ディスク、音、画像など
  - 公平性の実現
  - 資源の有効利用

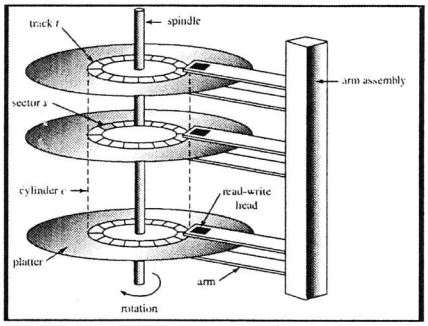
スレッド

- Thread of Control
  - プロセス内での一連の実行の流れ
  - 単にスレッドと呼ばれている
- プロセス内に複数スレッドを動かすような実行モデルをマルチスレッドモデルと呼ぶ



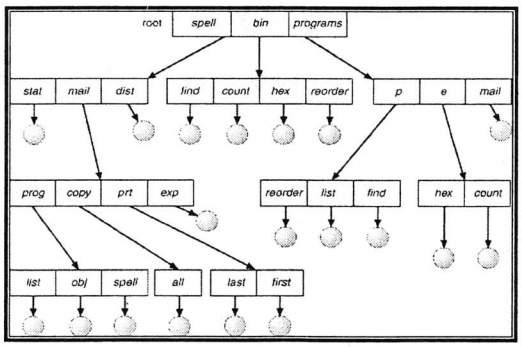
ファイルシステム

- 物理的な入出力では、
  - DISK上のブロックを、CPUのメモリ上に対応つけ
    - SEEK
    - READ
    - WRITE　　の3基本動作
  - ユーザに使い難い
- ファイルシステム
  - ファイルの検索（ファイル名、木構造やグラフ構造、内容検索など）
  - ファイルの属性とファイル処理との結合



木構造ディレクトリ

- 効率良い検索
- グループ化機能
- アクセス制御
- カレントディレクトリ (ワーキングディレクトリ)の概念
  - **cd** /spell/mail/prog
  - **more** list
  - mkdir tmp
  - rmdir tmp

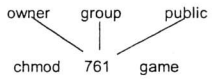


Unix系におけるファイルアクセス制御

- アクセスモード
  - read, write, executeを3bitsで表現する
- 3カテゴリ

			RWX
a) **owner access**	例:7	⇒	1 1 1
b) **group access**	例:6	⇒	RWX 1 1 0
c) **public access**	例:1	⇒	RWX 0 0 1

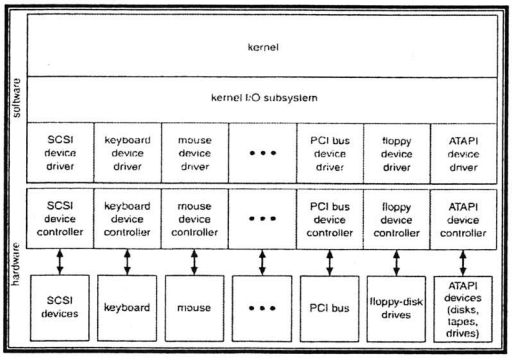
- アクセスモードを変更する
  - chmod



- グループを作成する
  - /etc/group
- ファイルのグループ名を変える
  - chgrp group-name game

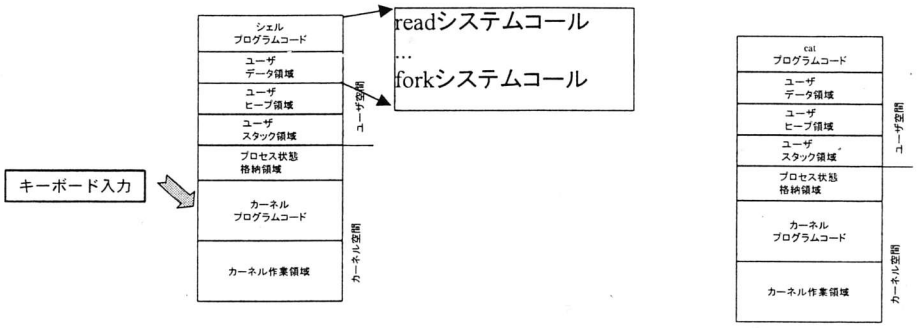
Application I/O Interface

- I/O システムコールによるデバイスの抽象化
  - いろんなデバイスがあるが、デバイスの特徴にあわせてユーザに対して統一したインターフェイスを提供する
    - Blockデバイス
    - Characterデバイス
    - Networkデバイス
- デバイスはいろんな特徴をもつ
  - Character-stream or block
    - 1バイト単位かブロック単位か
  - Sequential or random-access
    - 順次？ランダム？
  - Sharable or dedicated
  - Speed of operation
    - キーボード入力程度のスピード？
    - ギガビットネットワークのスピード？
  - read-write, read only,
  - or write only



コマンド実行の様子

- シェルの入力待ち ⇒ キーボード入力 ⇒ デバイスドライバ ⇒ システムコール ⇒ プロセス生成 ⇒ プロセス実行 ・・・・・・
- strace コマンド ： システムコールを追跡



cat コマンドの実行(csh on Linuxの場合)

```
% strace cat afile
execve("/bin/cat", ["cat", "afile"], [/* 62 vars */]) = 0
brk(0)                          = 0x804b364
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=130715, ...}) = 0
old_mmap(NULL, 130715, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40015000
close(3)                        = 0
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "¥177ELF¥1¥1¥1¥0¥0¥0¥0¥0¥0¥0¥0¥3¥0¥3¥0¥1¥0¥0¥0`¥204¥1"..., 1024) = 1024
...
read(3, "", 4096)                       = 0
close(3)                        = 0
munmap(0x40016000, 4096)                = 0
...
open("/usr/share/locale/ja_JP.eucJP/LC_CTYPE", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=474552, ...}) = 0
old_mmap(NULL, 474552, PROT_READ, MAP_PRIVATE, 3, 0) = 0x401c3000
close(3)                        = 0
...
fstat64(1, {st_mode=S_IFREG|0644, st_size=6849, ...}) = 0
open("afile", O_RDONLY|O_LARGEFILE)     = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=5, ...}) = 0
brk(0x8064000)                  = 0x8064000
read(3, "abcd¥n", 4096)                 = 5
write(1, "abcd¥n", 5asda
```

使用されたシステムコール

- fork　　プロセスの生成
  - » リスト中には現れていない
- execve　　実行可能ファイルを開き、プロセスを作り実行する
- open　　ファイルを開く
- close　　ファイルを閉じる
- old_mmap　　ファイルをメモリにマップする
- munmap　　メモリマップを解除する
- fstat64　　オープンしているファイルの状態を取り出す
- brk　　使用可能メモリ領域を増やす
- read　　オープンしているファイルを読む
- write　　オープンしているファイルに書く
- exit　　プロセスを終了する

cdを実行してみよう

- strace cd
- strace: cd: command not found
  と出てくるでしょう。これはなぜか？