システムプログラミング
ソフトウエアの構成と関係、開発準備

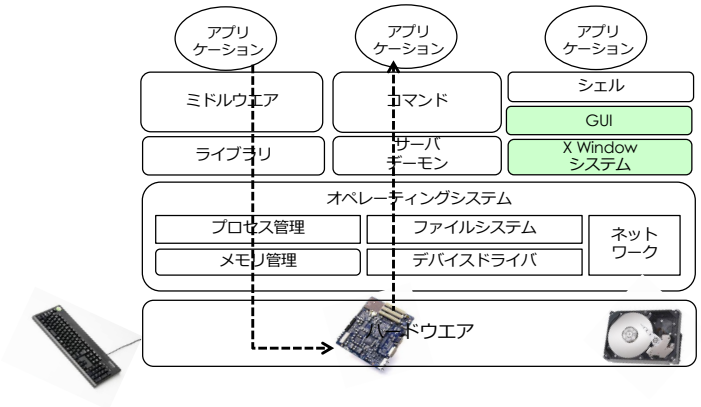
芝浦工業大学 情報工学科
菅谷みどり

システムプログラミング

- **CPU(演算子）やメモリ（ポインタ、配列）の基本的な機能だけではなく、OS が提供する様々な機能を使ったプログラミング**
  - コマンド、プロセス管理
  - 入出力ストリーム
  - ファイルシステム
  - ネットワーク通信
  - CG
  - デバイス制御
- **システム・プログラミングは,オペレーティング・システムが提供するシステム関数 (API) のライブラリを利用します。**
  - 難しいところは,このライブラリ関数がオペレーティング・シ ステムに依存する点です。

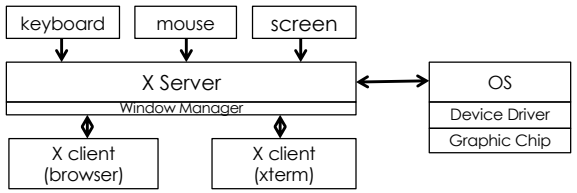
システムの構成要素

- **主に UNIX 系オペレーティングシステム**
  - 様々なプログラムから構成される



X Window System & GUI (Graphical User Interface)

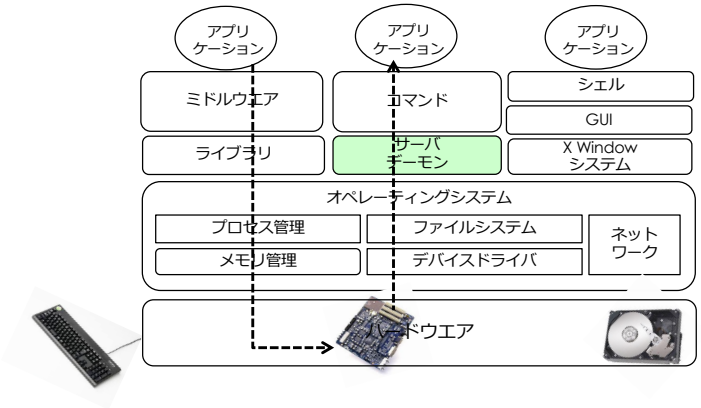
- **UNIX**
  - Xウィンドウシステムとウィンドウマネージャという独立したプログラムとして提供されている.
- **Xウィンドウシステム**
  - ビットマップディスプレイ上にウィンドウを表示するための基本的な機能を提供する



  - ユーザがウィンドウを操作するためのGUIは，GNOMEやKDEといったプログラム群（デスクトップ環境とも呼ばれる）により提供される.
    - **GNOME : http://www.gnome.org/**
    - **KDE : http://www.gnome.org/**

システムの構成要素

- **主に UNIX 系オペレーティングシステム**
  - 様々なプログラムから構成される



サーバ、デーモン

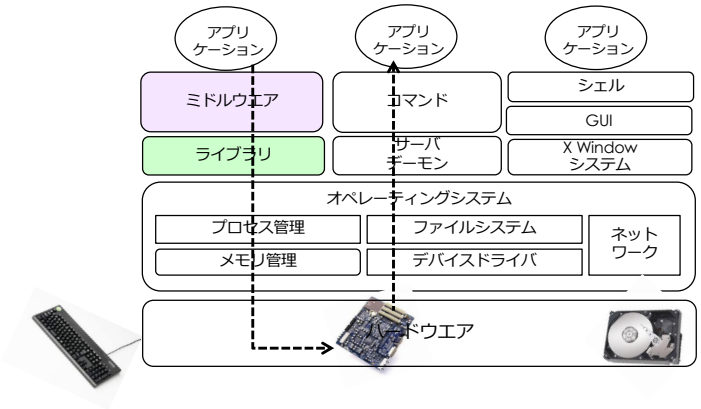
- **デーモン**
  - UNIXでは，バックグラウンドで動作し様々なサービスを提供する裏方で働くプログラムのことをデーモン（daemon）と呼ぶ
    - 最近ではサーバと呼ぶことも多い．
- **デーモンの例**
  - メイルの配信をするプログラム
  - プリンタへの出力要求を仲介するプログラム
  - リモートログインやリモートファイルコピーなどのネットワーク機能を提供するプログラムなどがある．

Top コマンドでデーモンを確認してみよう

```
$ top
```


システムの構成要素

- **主に UNIX 系オペレーティングシステム**
  - 様々なプログラムから構成される

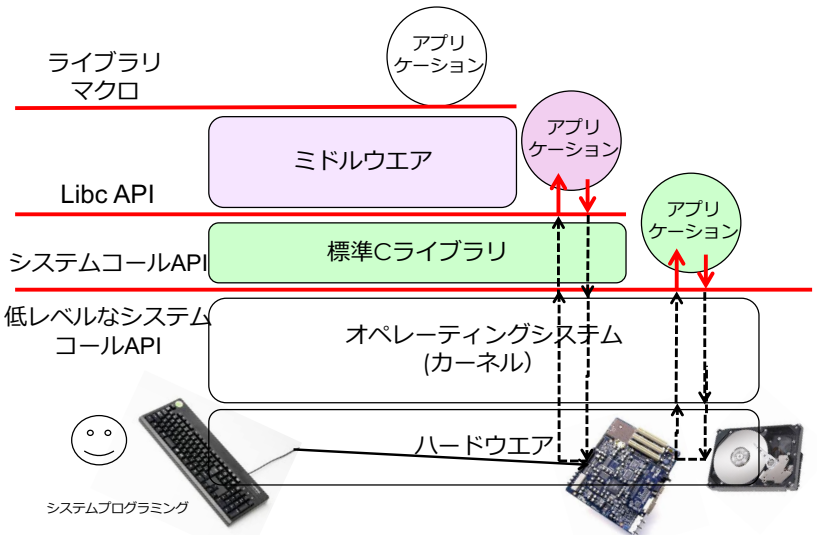


システムコール，ライブラリ，ミドルウェア

- **UNIXでプログラミングをするとは**
  - システムコール，ライブラリ，ミドルウェアを使用してプログラムを作成する．
- **システムコール**
  - OSカーネルの機能を直接呼び出すためのインタフェース．
    - UNIXのシステムコールは，できるだけシンプルになるように設計されている．
- **ライブラリとミドルウェア**
  - プログラムの部品となる関数の集合．
- **ライブラリとミドルウェアの違い**
  - ライブラリ:様々な目的のプログラムで共通の機能を提供するもの
  - ミドルウェア: ライブラリより特定のプログラム（例えばGUI）の共通部品となるものである．

API の階層

APIを通して，プログラムが，OS，OSの管理するデバイスを通して，
外の世界とつながり，プログラムと人間のインタラクションが可能になる



API (Application Programming Interfaces)

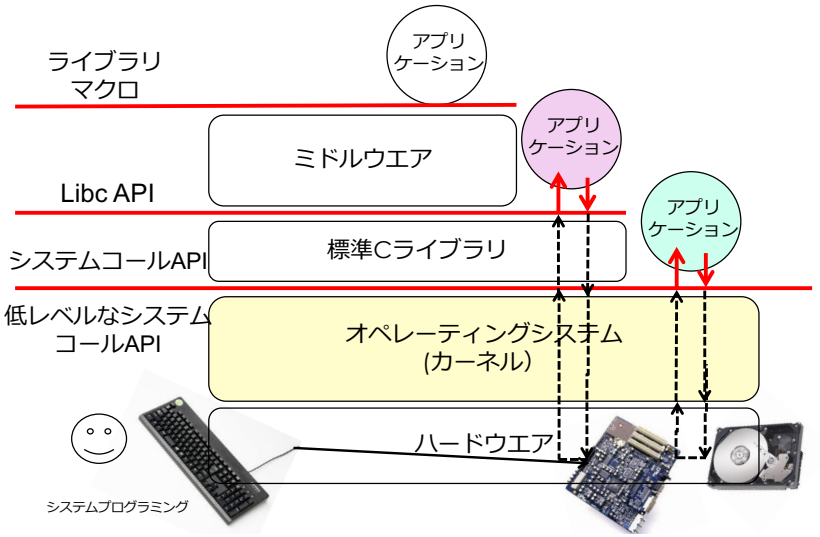
- **API とは**
  - ソフトウエアコンポーネントが、互いにやり取りするために利用するインターフェイス(関数）の仕様
    - 低レベルな（機械よりのプログラム言語を使う）ソフトウエアと、高レベルな（人間よりのプログラム言語を使う）ソフトウエア間の関係をより抽象化するための方法
    - 主に、ファイル制御、ウィンドウ制御、画像処理、文字制御などのための関数として提供されることが多い
- **e.g.**
  - POSIX : Portable Operating System Interface
    - 各種 UNIX をはじめとする OS 実装に共通の API を定めた国際規格
    - 移植性の高いアプリケーションの開発を容易にする
    - ANSI/ISO C
      - C言語のシステムコールとライブラリ関数を規定
  - Windows API
    - ベンダーによる文書

API の取り扱い

- **API の種類**
  - OS が提供するシステムコール
  - システムコールを便利に使えるようにするための、ライブラリやマクロ
- **API がなぜ必要なのか？**
  - API が無かったら？
    - ファイルをオープンして文字を入力して閉じるという操作
    - ➔ 全て、ハードウエアと直接（アセンブラ、機械語）で書く必要がある
  - API を使いこなすことで
    - ハードウエアの機能を分かりやすく、簡潔な記述で動かすことができる
- **本講義では**
  - Linux を用いて、POSIX (Portable Operating System Interface for UNIX) API を利用したプログラムを作成する

API の階層

APIを通して，プログラムが，OS，OSの管理するデバイスを通して，
外の世界とつながり，プログラムと人間のインタラクションが可能になる



何の API なのか？

- **どうやって確認するのか？**
  - Man コマンド
    - Manual (man) :
    - $ man man

$ man putchar

$ man getchar

- **標準C ライブラリ(libc) API**

Man で確認してみよう

```
doly@Midori-no-MacBook-Pro: ~ — less — 90×31

PUTC(3)                 BSD Library Functions Manual                 PUTC(3)

NAME
     fputc, putc, putc_unlocked, putchar, putchar_unlocked, putw -- output a
     character or word to a stream

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int
     fputc(int c, FILE *stream);
```

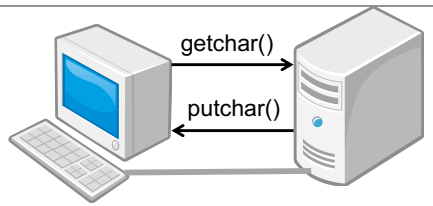

ライブラリ API を使ったプログラミング

- **OS が提供する抽象概念**
  - ファイル、プロセス、シグナル
    - ➔ ユーザからみた場合のコンピュータ（ハード資源）

```
#include <stdio.h>

int main (void)
{
        int c;
        while (( c = getchar()) != EOF)
                putchar(c);
}
```

どれが API ?



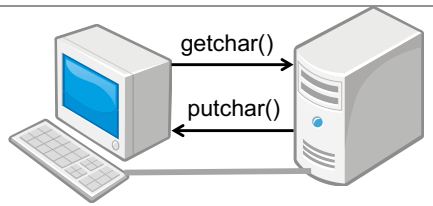
ライブラリ API を使ったプログラミング

- **ファイル、プロセス、シグナル**
  - ➔ ユーザからみた場合のコンピュータ（ハード資源）

```
#include <stdio.h>

int main (void)
{
        int c;
        while (( c = getchar()) != EOF)
                putchar(c);
}
```

どれが API ?



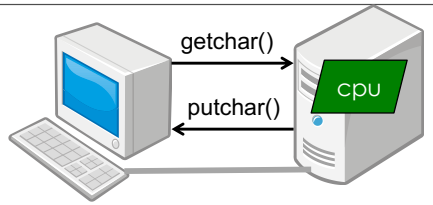
ライブラリ API を使ったプログラミング

- **ファイル、プロセス、シグナル**
  - ➔ ユーザからみた場合のコンピュータ（ハード資源）

```
#include <stdio.h>

int main (void)
{
        int c;
        while (( c = getchar()) != EOF)
                putchar(c);
}
```

どれが API ?

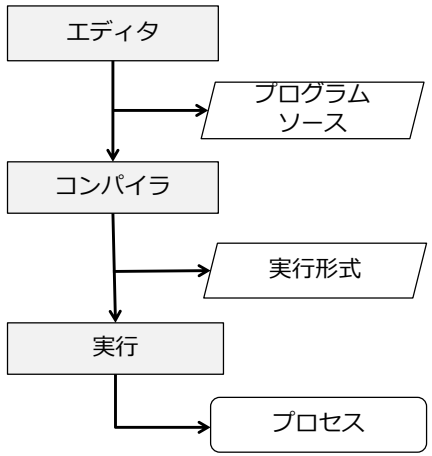


<補足> プログラムのコンパイル

- **ターミナルから、プログラムをコンパイルする**
  - コンパイラ (gcc) <オプション> 出力イメージ ソース
- **$ gcc –o prog prog.c**
- **Cプログラムのヘッダ**
  - #include <stdio.h>
- **ヘッダファイルの観察**
  - $ egrep getc /usr/include/stdio.h
- **プリプロセスのみ**
  - $ gcc –E getchar.c > getchar.i
  - $ less getchar.i
  - $ tail getchar.i
- **シンボルリストの確認**
  - $ nm getchar.o
  - $ nm getchar

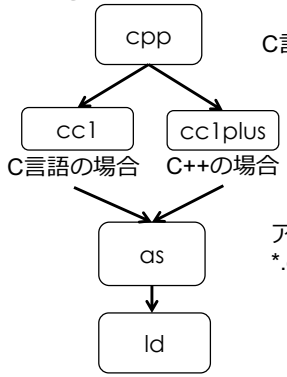
<補足> コンパイルとリンク

- **プログラム作成から実行まで**



(g)cc

- **(g) cc コマンド**
  - (g)cc それ自体は実はコンパイルといった処理をしない． コンパイルに必要な処理をしてくれるコマンドを呼び出すだけである．



C言語のプリプロセッサ, #include や #defineの展開

コンパイラ本体（アセンブラコードを出力）

アセンブラ。リンク前のオブジェクトファイル *.o を出力する。

リンカ。*.o やライブラリをリンクして、実行バイナリを作成する

- **Glibc**
  - http://ftp.gnu.org/gnu/libc/

$ glibc

  - $ find . | grep getchar

Segmentation Fault 確認

次のANSI C言語のコードはメモリ保護機能を持つプラットフォーム上でセグメンテーション違反を作り出す例

const char *s = "hello world";

***s = 'H';**

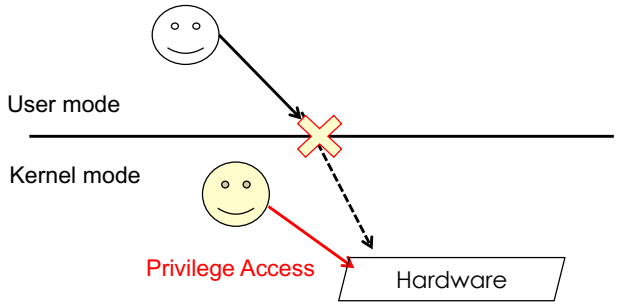
const （定数）は，書き換え不可状態で初期化

gcc –g –o segfault segfault.c

Backtrace してみよう

UNIX カーネル（オペレーティングシステム）

- **特権**
  - UNIX環境で特権モードで動作するプログラムはカーネルだけである． その他のプログラムは，ユーザモードというハードウェアへのアクセスは制御された環境で動作する．



  - Segmentation Fault で確認してみよう

カーネルのメモリ保護機能

- **Segmentation Fault**
  - ソフトウエア実行時のエラー条件
  - アクセスが許可されていないメモリ上の位置、許可されていない方法（Read only の位置への書き込み、OS 領域への書き込み）でメモリ上の位置にアクセスする際におこる
- **不正なメモリアクセス**
  - UNIX 系 OS => SIGSEGV シグナルを送信
  - Windows => STATUS_ACCESS_VIOLATION 例外
- **実際に確認してみよう**
  - **Segfault.c**
    - メモリ保護機能を持つプラットフォーム上で segmentation fault 違反
    - gcc segfault.c –g –o segfault
    - gdb ./segfault

segfault

Mac

$ gcc -o segfault segfault.c

segfault.c:5:5: error: read-only variable is not assignable

```
    *s = 'H';
    ~~ ^
```

1 error generated.

Apple LLVM version 7.0.2 (clang-700.1.81)

Target: x86_64-apple-darwin17.7.0

Thread model: posix

Linux

[doly@yli002 03_Samples]$ gcc segfault.c -o segfault

[doly@yli002 03_Samples]$ ./segfault

セグメントエラー

```
$ gdb segfault
GNU gdb 6.7.1
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
This GDB was configured as "x86_64-vine-linux"...
(no debugging symbols found)
Using host libthread_db library "/lib64/libthread_db.so.1".
(gdb) run
Starting program: /home/sit/doly/SysPro/2014_SysPro/03_Samples/segfault
(no debugging symbols found)
(no debugging symbols found)
(no debugging symbols found)

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400438 in main ()
```


ライブラリとOS のシステムコール

- **システムコールのリスト**
  - /user/include/asm/unistd.h
- **実行時の動作**
  - 実行時にライブラリ関数はそれを使用するプログラムの一部となる
  - カーネルは完全に独立したプログラムとして存在する
    - プロセスはシステムコールを通してのみカーネルの機能を使うことができる.
- **プログラムからの入出力**
  - 最後はシステムコールを通して実現：
    - **open, read, write, close** ： システムコール
    - **fopen, fread, fwrite, fclose** ：ライブラリ関数
      - 外の世界（入出力機器）とつながっているのはカーネルだけなので，scanf, printf は read, write と最終的にはシステムコールを呼ばないと入出力は行えない.

ライブラリ、システムコール

- **ライブラリだけで実現される機能もある**
  - 文字数を数える，文字列を比較する
    - E.g. strcmp
    - ➔ 入出力を伴わない，しかしプログラムでよく使われる部品となるような機能を，ライブラリはいろいろ提供している.
  - ※ライブラリをうまく使うことで，効率よくプログラミングができるようになる

マニュアルでの確認方法

- **システムコール, ライブラリ**
  - Cプログラムから呼び出す場合はどちらも関数呼び出しの形態で使用できるため，同じに見える
- **UNIX マニュアル**
  - システムコールは2章
  - ライブラリは3章
- **man コマンドで確認してみよう**
  - $ man strcmp
  - $ man read

プログラムの開発環境

- **UNIX オンラインマニュアルの読み方**

```
$ man システムコール名
$ man  ライブラリ関数名
$ man コマンド名
$ man –k キーワード
```

- **マニュアルの章立て**
  - 1章　コマンド
  - 2章　システムコール
  - 3章　ライブラリ関数
  - 4章　デバイスファイル
  - 5章　ファイル形式
  - 6章　ゲーム
  - 7章　その他
  - 8章　管理用コマンド

```
$ man printf
$ man 2 intro
```


演習1: mypower コマンドを開発しよう

- **課題）次の仕様で動作する mypower コマンドを作成する**
  - (1) コマンドの第一引数の数値を x とする
  - (2) y >= 1 の時、累乗 x^y を小数点表記で表示する
  - (3) y < 1 の時、累乗根 $\sqrt[y]{x}$を小数点表記で表示する
  - (4) 第 2 引数は省略可能で、省略した場合は y = 2.0 とする
  - (5) (1) から (4) の挙動に反する場合には、エラーとする
- **条件**
  - pow(), strtod()などの標準Cライブラリを用いて良い

コマンド実行例

- 【例】コマンド実行例

```
$ power 3 3
27.000000
```

- 【例】第 2 引数は省略可能 (二乗となる)

```
$ power 3
9.000000
```

- 【例】コマンド実行例 (20.5 は,平方根)

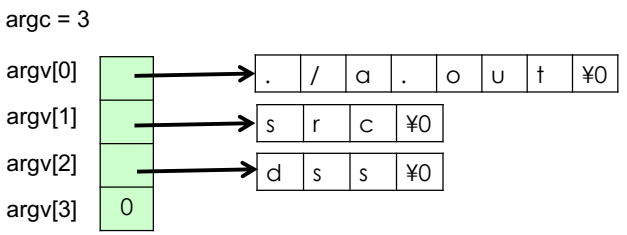
```
$ power 2 0.5
1.414214
```

- 【例】正しくないコマンド引数の場合

```
$ power mac
        Usage: power num [num]
```


main 関数への引数

- **main 関数への引数の意味**
  - argc (argument count)
    - int 型、コマンド行の文字列の個数（コマンド名＋引数）
  - argv (argument vector)
    - char * 型へのポインタを格納する配列
    - コマンド行の文字列それぞれへのポインタを格納した配列
  - 慣例的にこれらの名前が利用されている



コマンド引数プログラム

```
#include <stdio.h>
int main (int argc, const char **argv)
{
  int i;
       for (i = 0; i < argc; i++) {
             printf("argv[%d]: %s¥n", i, argv[i]);
       }
       return 0;
}
```

コンパイル、実行
$gcc –o com command.c
$./comand -l --helo file text
argv[0]: ./com
argv[1]: -l
argv[2]:…

演習2：mystrcmp コマンドを開発しよう

- **課題：次の仕様で動作する mystrcmp コマンドを作成する**
- **仕様**
  - (1) コマンドの第１引数(s1)、第２引数(s2) 2つの文字列を読み込む
  - (2) 一致しているかをチェックし、結果を出力する
    - 一致の場合には 0,
    - 不一致の場合下記の値を返す
      - s1 < s2 : 0 より小さい値
      - s1 > s2 : 0より大きい値
  - (3) 上記の挙動に反する場合には、エラーとする
- **条件**
  - strcmp() は用いない
- **ヒント**
  - 文字列の比較方法（文字列長に固定的な上限を設けない方が良い)
  - 文字型，文字列型：char と * char, const char, const *char の区別を再確認する
  - 提出前にテストを行おう

課題の提出方法

- **授業中に指示があります**

参考書

[1] Dustin Boswell、Trevor Foucher著, 角征典訳, リーダブルコード―より良いコードを書くためのシンプルで実践的なテクニック(Theory of practice), オライリー・ジャパン.

[2] 大圖 衛玄, ゲームプログラマのためのコーディング技術, 2015.

[3] Brian Kernighan, Jon Bentley, その他, ビューティフルコード (THEORY/IN/PRACTICE), 2008.