

2019年度 システムプログラミング (CSC.T344)

1. イントロダクション

林晋平@情報理工学院

講義情報

- 講義名: システムプログラミング
- 科目コード: CSC.T344
- 単位数: 2 (1-1-0)
- 開講クォーター: 1Q
- 授業形態: 講義 / 演習
- 曜日・时限: 火5-6, 金5-6
 - 基本的には 火: 講義 / 金: 演習 (正確には日程参照)

受講対象者

- 受講者数が計算機室の座席数を超えた場合、
受講者制限の可能性あり
 - 情報工学系 3 年生を優先します
- 旧講義（プログラミング第三）の
単位取得済の場合、受講できません

シラバス

講義の概要とねらい

- UNIX をはじめとするオペレーティングシステムが提供するシステムコールおよびそれを利用したプログラミング法を理解する。
- 特に，プロセスとプロセス間通信の概念，ファイル抽象，ソケット通信の基礎を学習し，実際のプログラミング言語によるプログラミングを実践する。

シラバス

授業計画

- 1. イントロダクション
- 2. UNIX環境 演習
- 3/4. 入出力, ファイル抽象 & 演習
- 5/6/7/8. プロセス, フォーク, パイプ & 演習
- 9/10. シグナル, ソケット通信 & 演習
- 11/12. メモリ管理 & 演習
- 13/14. システムプログラミングツール & 演習

説明の都合上, 順序が前後したり, 一部が省略・追加されるなど, 内容の変更の可能性あり.

チーム

- 教員

- 林 晋平 (はやししんpei; 講義メイン担当)
 - 居室: 西8号館E棟906号室
- 佐藤 俊樹 (さとうとしき; 演習メイン担当)
 - 居室: 西8号館E棟401号室

- TA (皆さんのお手伝いを助けてます)

- 宮藤 詩緒 (小池研)
- ZhengQing Li (小池研)
- Dong-Hyun Hwang (小池研)
- 吉岡 拓真 (西崎研)

- 授業に関する連絡先

- sysprog-2019@se.c.titech.ac.jp
 - 教員およびTAに届きます

成績評価ポリシー

- **演習: 60%**
 - ただし基準を超えて課題に取り組んだ学生には60点満点超えの点数を与える
- **期末試験: 40%**

講義スケジュール

1. 4/09(火) イントロダクション
2. 4/12(金) 演習 (UNIX環境)
3. 4/16(火) 入出力
4. 4/19(金) 演習 (入出力)
5. 4/23(火) プロセス, フォーク
6. 4/27(金) 演習 (プロセス1)
7. 5/07(火) パイプ
8. 5/10(金) 演習 (プロセス2)
9. 5/14(火) シグナル, ソケット
10. 5/17(金) 演習 (シグナル, ソケット)
11. 5/21(火) メモリ管理
12. 5/24(金) システムプログラミングツール
13. 5/28(火) 演習 (メモリ管理)
14. 5/31(金) 演習 (システムプログラミングツール)
15. 6/07(金) 期末試験

講義 : W933

演習 : 計算機室312

本講義で仮定する知識/関連講義

- **Cプログラミング**
 - 手続き型プログラミング基礎
 - 手続き型プログラミング発展
- **UNIX の基本的な概念、操作方法**
 - コマンドラインインタフェースの利用
 - 基本的なコマンドの利用方法
 - 情報リテラシ第一・第二
- **計算機の動作原理**
 - i386の基本構造 (アセンブリ言語)

教科書と参考書

- **教科書**

- なし
 - 講義スライドを順次 OCW-i で公開
 - 著作権等の関係で、OCW では未公開

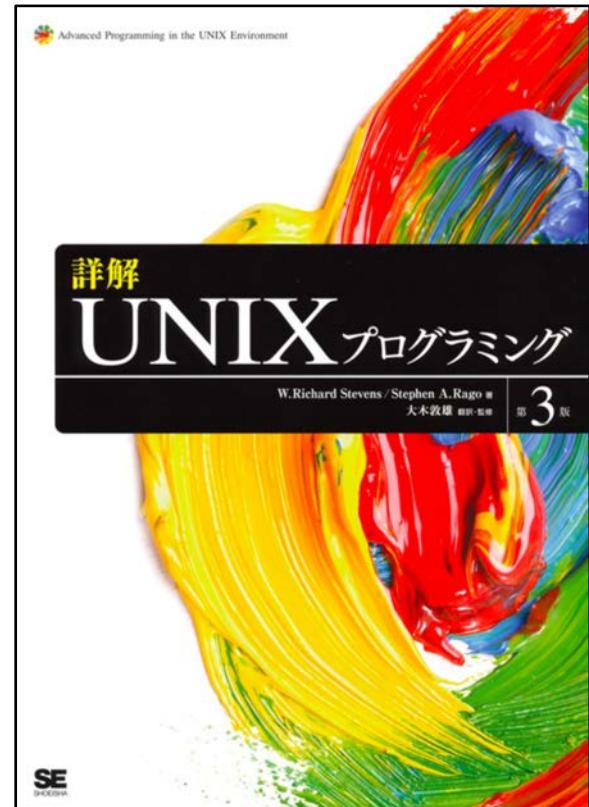
- **参考書**

- 復習や予習等の独習用に購入を薦める書籍
 - 講義や実習の時間内に使うことはありません
 - 購入は必須ではありません

参考書

- **詳解UNIXプログラミング第3版**

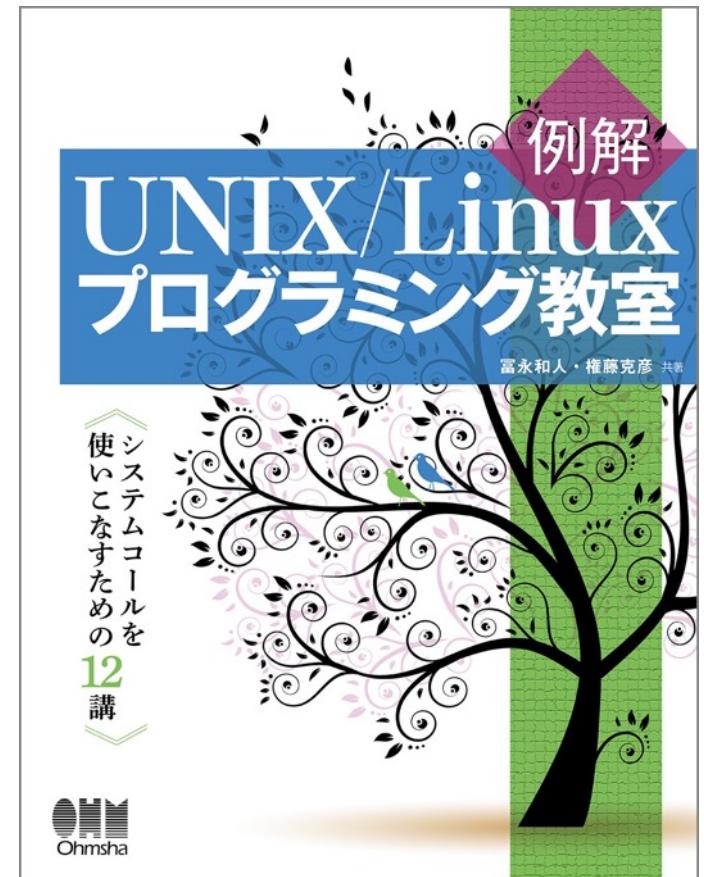
- W.Richard Stevens (原著), 大木敦雄(訳)
- 翔泳社
- ISBN: 978-4798134888
- 図書館にもあるはず



参考書

- **例解UNIX/Linuxプログラミング教室**

- 富永和人, 権藤克彦
- オーム社
- ISBN: 978-4274222108
- 2018年に改版されました
(旧:ピアソン)
- 図書館にもあるはず



参考書

• プログラミング作法

– B.W. Kernighan, Rob Pike (著), 福崎俊博 (訳)

↑ K&Rの K



– KADOKAWA

– ISBN: 978-4048930529

– 昨年に再刊行されました
(旧: アスキー)

– 読みやすいCコードを
書きたい方に
– 犬がかわいい



参考書（ゆるふわ枠）

- **まんがでわかるLinux シス管系女子
コマンド&シェルスクリプト入門**
 - 結城洋志 (Piro)
 - 日経BP
 - ISBN: 978-4822224967
 - 入門書：もしあなたが CLI 恐怖症なら
 - 内容は初步的だがユースケース指向の良書



本講義の位置付け

言語の習得 (手続き型プログラミング)
→ OSの界面 (システムプログラミング)
→ OSの内面 (システムソフトウェア)

【情報工学系（学士課程）】

	1年目	2①	2②	2③	2④	3①	3②	3③	3④	4①～②	4③～④④
情報工学基礎	類専門科目	計算基礎論	オートマトンと形式言語	情報論理	データ構造とアルゴリズム				システムソフトウェア		
	微分積分第一	確率論・統計学	論理回路理論	アセンブリ言語		コンピュータ論理設計					
情報工学発展	コンピュータサイエンス第一・第二				人工知能	システムの問題解決と意思決定	システム解析	プロジェクト指向設計	コンパイラ構成		
	情報リテラシー第一・第二				データベース	情報認識	数値計算法	動的システム			
	生命科学基礎第一				システムプログラミング	生命情報解析	コンピューターアーキテクチャ	システム制御			
					コンピュータネットワーク	並列プログラミング					
プログラミング	手続き型プログラミング基礎	手続き型プログラミング発展	関数型プログラミング	オブジェクト指向プログラミング			システム設計演習	システム構築演習			
										先端情報工学	
										情報工学英語プレゼンテーション	

2019年度 システムプログラミング (CSC.T344)

1. イントロダクション

林晋平@情報理工学院

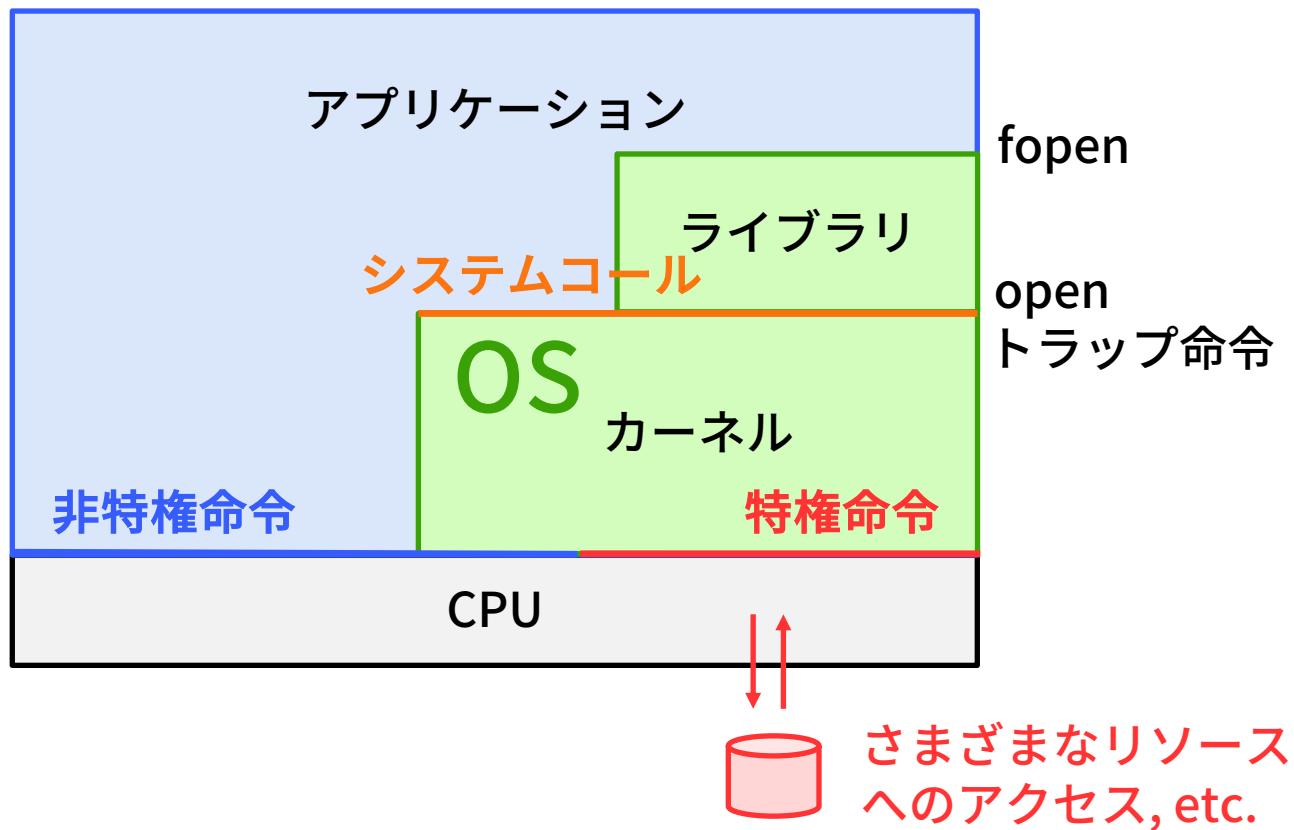
シラバス

講義の概要とねらい

- UNIX をはじめとするオペレーティングシステムが提供するシステムコールおよびそれを利用したプログラミング法を理解する。
- 特に，プロセスとプロセス間通信の概念，ファイル抽象，ソケット通信の基礎を学習し，実際のプログラミング言語によるプログラミングを実践する。

システムコール

- OSが提供するサービスの関数インターフェース
 - OS内部の機能を関数として呼び出せるようにしたもの



Mac OS X のシステムコール

#define SYS_syscall	0
#define SYS_exit	1
#define SYS_fork	2
#define SYS_read	3
#define SYS_write	4
#define SYS_open	5
#define SYS_close	6
#define SYS_wait4	7
	/* 8 old creat */
#define SYS_link	9
#define SYS_unlink	10
	/* 11 old execv */
#define SYS_chdir	12

システムコールはなぜ必要?

- OSが便利な機能を提供するため
 - ハードウェアを直接使うのは難しい
 - 例: HDDへのファイルの書き込み
 - 簡単に使えるインターフェースを提供
 - 複数のプロセスで資源を共有できるよう調整
- OSが機能を安全に提供するため
 - 例: 他人のファイルを読めてはいけない
 - 決められたシステムコールの呼び出し以外からのアクセスを許さないことにより実現

例: ファイルアクセス

- OSが便利な機能を提供するため
 - open, read, write, ……
 - を使えば、HDDに書き込むためのコードをアプリケーションプログラムに書く必要がない
- OSが機能を安全に提供するため
 - open, read, write, ……
 - は、ファイルの所有権をチェックする
 - 以外の方法によるファイルアクセスを許さない

システムコール：具体的に

```
#include <fcntl.h>
int open(const char *path, int flags);
int open(const char *path, int flags, mode_t mode);
```

```
#include <fcntl.h>
int fd = open("path/to/file.txt", O_RDONLY);
```

- **一見、ふつうのC言語の関数**
 - Cプログラマにとってわかりやすいインターフェースを提供している
 - UNIXシステムプログラミングにCが使われる理由の一つ
- **実際はカーネルへの処理の依頼**
 - ユーザプロセス自体は処理をしない



open (2): ファイルを開く

```
#include <fcntl.h>
int open(const char *path, int flags);
int open(const char *path, int flags, mode_t mode);
```

- *path*: ファイル名
- *flags*: 操作フラグの OR
 - O_RDONLY 読み専用
 - O_WRONLY 書き専用
 - O_RDWR 読み書き両用
 - O_APPEND 追記モード
 - O_CREAT ファイルが無ければ作る
 - O_TRUNC ファイルがあれば空にしてから開く
 - O_EXCL O_CREAT 時にファイルがすでにあれば失敗
 - O_NONBLOCK ブロックしない
- *mode*: 操作許可ビット
- 戻り値: ファイル記述子

\$ man 2 open

OPEN(2)

BSD System Calls Manual

OPEN(2)

NAME

open, openat -- open or create a file for reading or writing

SYNOPSIS

```
#include <fcntl.h>
```

```
int  
open(const char *path, int oflag, ...);
```

```
int  
openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The file name specified by path is opened for reading and/or writing, as specified by the argument oflag; the file descriptor is returned to the calling process.

The oflag argument may indicate that the file is to be created if it does not exist (by specifying the O_CREAT flag). In this case, open() and openat() require an additional argument mode_t mode; the file is created with mode mode as described in chmod(2) and modified by the process' umask value (see umask(2)).

The openat() function is equivalent to the open() function except in the case where the path argument is the null pointer. In this case, the file

システムコール呼出し

- 普通の関数呼び出しではない
- ただし，Mac OS X では，標準ライブラリ中のラッパー関数の呼び出しになってしまっている

man に書かれた
getpid(2) のインターフェース

```
#include <unistd.h> /* getpid */  
pid_t getpid(void);
```

を呼ぶ関数を書いてみた

```
int lib_getpid(void) {  
    return getpid();  
}
```

残念ながら普通の呼び出しになってしまう

```
_lib_getpid:  
    pushl  %ebp  
    movl  %esp, %ebp  
    subl  $8, %esp  
    calll _getpid  
    addl  $8, %esp  
    popl  %ebp  
    retl
```

システムコール呼出し：概念的には

man に書かれた
getpid(2) のインターフェース

```
#include <unistd.h> /* getpid */  
pid_t getpid(void);
```

システムコール番号は
syscall.h に書いてある

```
/* /usr/include/sys/syscall.h */  
#define SYS_getpid 20
```

システムコール呼び出しあはこんな感じ

```
/* 模倣コード */  
int getpid(void) {  
    int ret;  
    asm volatile  
        ("int $0x80"  
    出力 :"=a"(ret)  
    入力 :"0"(SYS_getpid));  
    return ret;    20  
}
```

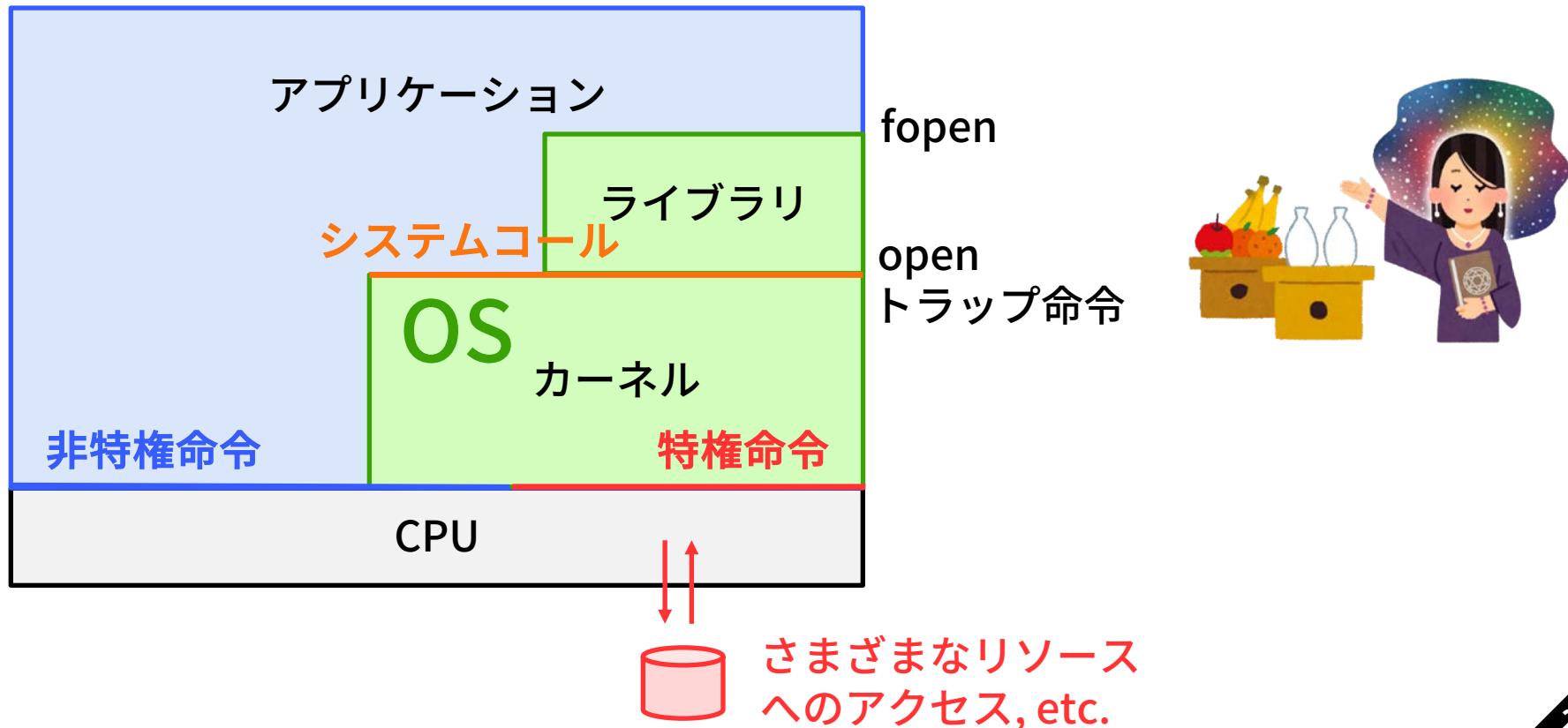
つまりこういうアセンブリで動く

```
pushl %ebp  
movl %esp, %ebp  
pushl %eax  
movl $20, %eax  
int $128  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax  
addl $4, %esp  
popl %ebp  
retl
```

システムコール番号 (20) を eax レジスタに添えて int 命令を実行する。

システムコール

- OSが提供するサービスの関数インターフェース
 - OS内部の機能を関数として呼び出せるようにしたもの



システムコールの調べ方

- **man コマンドで読めるマニュアルを読む**
 - `man 2 システムコール名` を実行
 - NAME を読んで概要を理解
 - SYNOPSIS (大意) を読んで
 - `include` すべきヘッダファイルを理解
 - 関数プロトタイプを理解
 - DESCRIPTION を読んで詳細を理解
 - 引数の意味を理解
 - RETURN VALUES を読んで返り値の意味を理解
 - ERRORS を読んでエラー条件とコードを理解
- **英語を怖がらない**

man (1): マニュアルの表示

```
% man [options...] [section] name
```

[…]: 省略可能

- *section*: ページを探すセクションを指定。
別セクションに同名があるときは指定した方が良い
 - 1 コマンド
 - 2 システムコール
 - 3 ライブラリ関数
 - 4 スペシャルファイル
 - 5 ファイルフォーマット
 -
- *name*: マニュアルページ名
 - コマンド名, システムコール名, ライブラリ関数名など
 - セクション名を添えて man(1) のように対象を呼んだりする

\$ man 2 open ← 検索開始

OPEN(2)

BSD System Calls Manual

OPEN(2)

NAME

open, openat -- open or create a file for reading or writing

↑ 読み書き用にファイルを開く/作成する

SYNOPSIS

#include <fcntl.h> ← fcntl.h が必要

int

open(const char *path, int oflag, ...); ← 関数プロトタイプ

int

openat(int fd, const char *path, int oflag, ...);

DESCRIPTION

↓ 引数 path はファイル名

The file name specified by path is opened for reading and/or writing, as specified by the argument oflag; the file descriptor is returned to the calling process.

↑ 開き方は oflag で指定する

RETURN VALUES

↓ 成功時にはファイル記述子を表す非負整数を返す

If successful, open() returns a non-negative integer, termed a file descriptor. It returns -1 on failure, and sets errno to indicate the error.

↑ 失敗時は -1 を返し, エラー詳細を errno をセットする

.....

セクションの指定が必要な場合

```
$ man open
```

OPEN(1)

BSD General Commands Manual

OPEN(1)

NAME

open -- open files and directories

SYNOPSIS

```
open [-e] [-t] [-f] [-F] [-W] [-R] [-n] [-g] [-h] [-s sdk]
      [-b bundle_identifier] [-a application] file ... [--args arg1 ...]
```

DESCRIPTION

The open command opens a file (or a directory or URL), just as if you had double-clicked the file's icon.

```
$ man 2 open
```

OPEN(2)

BSD System Calls Manual

OPEN(2)

NAME

open, openat -- open or create a file for reading or writing

SYNOPSIS

```
#include <fcntl.h>
```

.....

man の詳細も man でわかる

```
$ man man
```

```
man(1) man(1)
```

NAME

man - format and display the on-line manual pages

SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
[-M pathlist] [-P pager] [-B browser] [-H htmlpager] [-S section_list]
[section] name ...
```

DESCRIPTION

man formats and displays the on-line manual pages. If you specify section, man only looks in that section of the manual. name is normally the name of the manual page, which is typically the name of a command,

function, or file. However, if name contains a slash (/) then man interprets it as a file specification, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.

ググってもいいけど……

- ググって出てきた情報，本当に正しい?
 - 正しいにしても，別の OS の話だったり
- man を読んで，意味のわからないところを補足する形でググるとよい
 - まず一次情報にあたるクセをつけよう

システムコールの理解はOSのしくみの理解を助ける

- これまでに使ってきたライブラリ関数と対応するシステムコール
 - ライブラリ関数 `fopen`, `fclose`, `printf`, …
 - システムコール `open`, `close`, `write`, …
- 違いは?
 - ライブラリ関数も内部的にシステムコールを実行
 - ライブラリ関数は**バッファを利用**（詳細は後日）
 - ので（多くの場合）速い
 - システムコールを行う回数を抑えているため
(システムコールは必ずしも速くない)
 - ので注意が必要
 - `printf` したからといって、本当に書かれているとは限らない

講義のねらい (+α)

- **UNIX が提供するシステムコールを
利用法の側面から理解することにより, ……**
 - システムプログラミングができるようになる
 - 発展的なCプログラミングの訓練
 - UNIX のしくみを理解する
 - より深い内部構造の理解（システムソフトウェア）
 - UNIXらしさに親しむ
 - UNIX のインターフェースは POSIX 規格で標準化されている
プログラマの常識
 - 多くのプログラミングの概念がこれに基づく
 - ソースコード理解にも役立つ
 - 変に怖がらないようになる

講義スケジュール

1. 4/09(火) イントロダクション
2. 4/12(金) 演習 (UNIX環境)
3. 4/16(火) 入出力
4. 4/19(金) 演習 (入出力)
5. 4/23(火) プロセス, フォーク
6. 4/27(金) 演習 (プロセス1)
7. 5/07(火) パイプ
8. 5/10(金) 演習 (プロセス2)
9. 5/14(火) シグナル, ソケット
10. 5/17(金) 演習 (シグナル, ソケット)
11. 5/21(火) メモリ管理
12. 5/24(金) システムプログラミングツール
13. 5/28(火) 演習 (メモリ管理)
14. 5/31(金) 演習 (システムプログラミングツール)
15. 6/07(金) 期末試験

講義 : W933

演習 : 計算機室312