

# プログラミング応用

<http://bit.ly/ouyou3d>

## オペレーティングシステムと プログラミング (3)

前期 第5週

2018/5/15

# 本日は・・・

- ▶ システムコールとは？
- ▶ C言語のライブラリ関数とシステムコール
- ▶ システムコールの使い方
- ▶ システムコールの種類
- ▶ 各システムコールの説明
- ▶ コマンドライン引数

# 本日は・・・

- ▶ システムコールとは？
- ▶ C言語のライブラリ関数とシステムコール
- ▶ システムコールの使い方
- ▶ システムコールの種類
- ▶ 各システムコールの説明
- ▶ コマンドライン引数

# 復習：OSの内部構造

- ▶ OS = カーネルの集合体
- ▶ 各々のカーネルはOSの基本機能を提供
- ▶ カーネル提供する機能：プロセス管理、空間管理、ファイル管理、割り込み制御、入出力制御、時間管理など
- ▶ アプリケーションソフトウェアは、システムコールを使ってカーネルの機能を呼び出す  
(アプリケーションソフトウェアは基本的にハードウェアを制御する命令を出すことは出来ない)

# システムコールとライブラリ関数

## ▶ システムコール

- ▶ C言語からカーネルが提供する機能を使うときに必ずシステムコールを使うことになっている
- ▶ 代表例：open(), write(), close(), fork()など  
(C言語から関数として使うことができる)

## ▶ ライブラリ関数 (標準ライブラリ関数)

- ▶ システムコールを組み合わせて、まとめた処理を実現する
- ▶ 代表例：printf(), scanf(), fopen(), fclose()など  
(子プロセスの生成など、ライブラリ関数ではできないものもある)

# straceコマンドとは

コマンド実行時に、**どのようなシステムコールが呼び出されているかを確認するためのコマンド**

## 【使い方】

```
$ strace 実行ファイル名
```

## 【使用例】

```
$ strace ./a.out  
(「./a.out」にstraceを付けて実行する)
```

# straceコマンドの例

以下のようなCプログラムを書いてコンパイルして…

【例1】

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World\n");
    return 0;
}
```

straceコマンドを付けて実行してみると…

```
$ strace ./a.out
```

# straceコマンドの例

次のような情報が出力される（一部）

```
mprotect(0x7f3c2dba7000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)        = 0
mprotect(0x7f3c2ddd4000, 4096, PROT_READ) = 0
munmap(0x7f3c2ddbb000, 91726)            = 0
fstat(1, {st_mode=S_IFCHR|0620,
st_rdev=makedev(136, 1), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f3c2ddd1000
write(1, "Hello, World\n", 14Hello, World
)                = 14
exit_group(0)                                = ?
```

mprotect(), fstat(), mmap(), write()など  
上記で出力される関数は全てシステムコール

# 本日は・・・

- ▶ システムコールとは？
- ▶ C言語のライブラリ関数とシステムコール
- ▶ システムコールの使い方
- ▶ システムコールの種類
- ▶ 各システムコールの説明
- ▶ コマンドライン引数

# UNIXのシステムコール

UNIXが用意しているシステムコールは**約300種類**ですが、この授業では代表的な以下の**5個**について学びます。

- ▶ 入出力処理をする  
`write()`, `read()`
- ▶ ファイルを扱う  
`open()`, `close()`, `mv()`

# 本日は・・・

- ▶ システムコールとは？
- ▶ C言語のライブラリ関数とシステムコール
- ▶ システムコールの使い方
- ▶ システムコールの種類
- ▶ 各システムコールの説明
- ▶ コマンドライン引数

# write()

出力処理をするシステムコール

## 【使い方】

```
write(ファイル記述子, バッファ, バッファ長);
```

- ▶ ファイル記述子 … 整数1, 2もしくはファイルディスクリプタ数を指定
  - \* 1 : 標準出力
  - \* 2 : 標準エラー出力
  - \* ファイルディスクリプタ (後で説明)
- ▶ バッファ … 出力する文字列を格納したポインタ変数
- ▶ バッファ長 … 文字列の長さ (sizeofで取得できる)

# write() の例

システムコールwrite()を使ってstrに格納されている  
文字列を出力する

## 【例2】

```
#include <stdio.h>
#include <unistd.h> //write()を使うためにインクルードする

int main(void)
{
    char str[] = "Good Morning"; //strに文字列を格納する
    write(0, str, sizeof(str)); //strの文字列を出力する
    return 0;
}
```

# open()

ファイルを開くシステムコール (fopen()とほぼ同じ)

## 【使い方】

```
open("ファイルへのパス", モード);
```

- ▶ ファイルへのパス … 開くファイルのファイル名
- ▶ モード … fopenと同様なモードを指定できる
  - \* O\_RDONLY (読み込み専用)
  - \* O\_WRONLY (書き込み専用)
  - \* O\_RDWR (読み書き可能)

# close()

ファイルを閉じるシステムコール (fclose()とほぼ同じ)

## 【使い方】

```
close(ファイルディスクリプタ);
```

# ファイルディスクリプタ

- ▶ `open()`の戻り値となる**整数値**で、ファイルを一意に特定するための識別子となる
- ▶ 戻り値は、**3以降の整数値**となる
- ▶ 0～2は以下のように**予約済み**となっている
  - \*0：標準入力
  - \*1：標準出力
  - \*2：標準エラー出力

# open(), close() の例

オープンしたファイルのファイルディスクリプタを表示して、ファイルをクローズする

## 【例3】

```
#include <stdio.h>
#include <fcntl.h> //open()を使うためにインクルードする
#include <unistd.h> //close()を使うためにインクルードする

int main(void)
{
    int file1 = open("sample1.txt", O_RDONLY);
    int file2 = open("sample2.txt", O_RDONLY);
    printf("file1のディスクリプタ: %d\n", file1);
    printf("file2のディスクリプタ: %d\n", file2);

    close(file1);
    close(file2);
    return 0;
}
```

# read()

オープンしたファイルの中身を読み込みバッファに格納

## 【使い方】

```
read(ファイルディスクリプタ, バッファ, バッファ長);
```

- ▶ バッファ … 読み込んだ文字列を格納する配列  
(例えば「char buffer[256];」などと配列を宣言しておく)
- ▶ バッファ長 … バッファの長さ (「sizeof(buffer)」で取得できる)

# read()の例

sample1.txtの中身を標準出力に出力する

## 【例4】

```
#include <stdio.h>
#include <fcntl.h> //open()を使うためにインクルードする
#include <unistd.h> //read()を使うためにインクルードする

int main(void)
{
    char buffer[256];
    int file1 = open("sample1.txt", O_RDONLY);
    read(file1, buffer, sizeof(buffer));
    printf("%s", buffer);
    close(file1);
    return 0;
}
```

# 本日は・・・

- ▶ システムコールとは？
- ▶ C言語のライブラリ関数とシステムコール
- ▶ システムコールの使い方
- ▶ システムコールの種類
- ▶ 各システムコールの説明
- ▶ コマンドライン引数

# コマンドライン引数

catやgrepなどのUNIXコマンドもC言語で実装されている

【例えばこんな風に実行する場合】

```
$ cat sample1.txt
```

コマンド

コマンドライン引数

# コマンドライン引数について

プログラム実行時（つまり「a.out」を実行する時）に、ユーザがプログラムに与えることができる引数

これまでmain()で使っていた仮引数voidの代わりに、  
仮引数argcとargvを使う (p.421～)

```
1: #include <stdio.h>
2:
3: int main(int argc, char *argv[ ])
4: {
5:     int i;
6:
7:     printf("コマンドライン引数の個数: %d\n", argc);
8:     for(i=0; i<argc; i++) {
9:         printf("argv[%d]: %s\n", i, argv[i]);
10:    }
11:
12:    return 0;
13: }
```

argcが文字列の配列となり、argvが配列  
の要素数となって、プログラムが開始する

# コマンドライン引数の例

「./a.out」の後に、文字列を空白で区切って指定する

端末（ターミナル）

```
$ ./a.out myfile1.txt myfile2.txt
```

argcは配列の要素数が入っている

実行

プログラム

```
"./a.out" "myfile1.txt" "myfile2.txt"
```

3

argc

argv[0]

argv[1]

argv[2]

「./a.out」も含めて、argvがこれらの文字列を参照している

```
int main(int argc, char *argv[])
{
    ...
}
```

# 【練習5-1】

例1のプログラムを入力して、straceを付けて実行し、  
実行結果を確認してみましょう。

# 【練習5-2】

例2のwrite()のサンプルプログラムを入力して、実行結果を確認してみましょう。

# 【練習5-3】

例4のread()のサンプルプログラムを入力して、実行結果を確認してみましょう。

ただし、ファイル「sample1.txt」は予め用意しておく必要があります。（例えば、以下のようにechoとリダイレクトを組み合わせて）

```
$ echo "Good Morning!" > sample1.txt
```

# 【課題5-1】

標準ライブラリ関数（システムコールでない）を用いて「sample1.txt」の内容を「sample2.txt」にコピーするプログラムを作成して下さい。（次のスライドのプログラムを参考にして作れます。）

# 【課題5-1】

以下にfgetc()とfputc()を追加すると完成します

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    FILE *fp1, *fp2;
    char c;
    if ((fp1 = fopen(argv[1], "r")) == NULL) {
        printf("コピー元のファイルが開けません。\\n");
        return 1;
    }
    if ((fp2 = fopen(argv[2], "w")) == NULL) {
        printf("コピー先のファイルが開けません。\\n");
        return 1;
    }
    while ((c = /* ここでfgetc()を使う */ ) != EOF) {
        /* ここでfputc()を使う */;
    }
    fclose(fp2);
    fclose(fp1);
    return 0;
}
```

# 【課題5-2】

課題5-1で作成したプログラムを、 straceを付けて実行して結果を確認して下さい。（コピー元のファイルは予め用意しておく必要があります。）

実行結果の出力の中に、 **システムコールopen(), read(), write(), close()**が含まれていることを確認してください。

# 【課題5-3①】

～システムコールによるファイル移動コマンドmvの実装～

コマンドライン引数として2つファイル名を受け取り  
printf()で表示するプログラムを作成して下さい。

# 【課題5-3②】

課題5-3①のプログラムに次の処理を追加して下さい。

- ▶ コマンドライン引数の第1引数として受け取ったファイルを、**読み込み専用**でopenするシステムコールを追加
- ▶ 第2引数として受け取ったファイルを、**書き込み可能**で、かつ、ファイルがすでに存在する場合は空にするシステムコールを追加

「読み書き可能でかつ、ファイルが存在するならば空にする」には以下のように書く

```
open(ファイル名,O_WRONLY|O_CREAT| O_TRUNC,0666);
```

# 【課題5-3③】

課題5-3②のプログラムに次の処理を追加して下さい。

- ▶ 第1引数で受け取ったファイルから文字列を読み込む  
(read()を使う)
- ▶ 第2引数で受け取ったファイルへ文字列を書き込む  
(write()を使う)

# 【課題5-3④】

課題5-3③のプログラムに次の処理を追加して下さい。

- ▶ 第1引数で受け取ったファイル（つまり、`argv[1]`を）を削除する（システムコール`remove()`を使う）

例えば、ファイル「`test.txt`」というファイルを削除するのは以下のように書く

```
remove("test.txt");
```

# 【課題5-3⑤】

課題5-3④のプログラムに次の処理を追加して下さい。

- ▶ 第1引数で受け取ったファイルをクローズする
- ▶ 第2引数で受け取ったファイルをクローズする

# 【課題5-4】

課題5-3で作成したプログラムを実行して、ファイル移動ができるかの動作を確認して下さい。

(例えば、予めファイル「src.txt」を用意しておいて、その中身を「dst.txt」に移動できるかどうか実行してみる。)