

システムプログラミング II(System Programming II)					
本科	選択・必修	開設時期	単位数	授業形態	担 当
情報電子	必修	4 年前	1	講義	重村哲至 山本孝子
【授業の概要】 3 年の「システムプログラミング I」に続く科目である。3 年に引き続き、UNIX のシステムコールを使用したプログラミングと、その背景にあるオペレーティングシステムの仕組みについて授業する。					
【授業の進め方】 講義と演習を繰り返し講義した内容はプログラムを作成して動作確認をしながら進める。十分な演習時間は確保しないので、時間外の演習（プログラムの完成）が必須である。プログラムは次回の授業までに提出すること。					
【授業の概要】	【授業項目】			【内 容】	
1 回	高水準 I/O、エラー処理			FILE 型 , stdin, stdout, stderr, perror, fopen, fclose 等について学ぶ。[演習：cp、cmp コマンドを作成する。]	
2 回	システムコールと低水準 I/O			システムコールとは何か、また、open, read, write, close をシステムコールの例として学ぶ。[演習：cp コマンドの作成と性能比較]	
3 回	UNIX ファイルシステム			ファイルのモード、ディレクトリ、ハードリンク、シンボリックリンクについて学ぶ。[演習：ファイルシステムの操作と観察]	
4 回	ファイル操作			mkdir, rmdir, symlink, link, unlink, rename, chmod, chdir システムコールについて学ぶ。[演習：これらのシステムコールを使用する幾つかのコマンドを作成する。]	
5 回	プロセス			プロセスとは何か解説した後、プロセス関連のコマンドについて学ぶ。[演習：ps, kill コマンドを用いて時計プログラムを一時停止させたりする。]	
6 回	非同期処理 (1)			UNIX のシグナル機構の概要を学んだ後、signal, kill システムコールを紹介する。[演習：^ C を押すと途中結果を表示するプログラム]	
7 回	非同期処理 (2)			alarm, pause, sleep システムコールを使用したタイムアウトのテクニックを紹介する。[演習：1 秒毎に途中結果を表示するプログラム]	
8 回	中間試験			ここまでに出てきたシステムコールや関数の役割り使用方法等について試験する。	
9 回	環境変数			UNIX の環境変数の役割、メカニズムに付いて紹介する。[演習：環境変数の変更と、その結果の確認]	
10 回	プログラムからの環境変数利用			環境変数をプログラムから参照する方法を学習する。[演習：printenv コマンドを作成する。]	
11 回	プロセスの生成とプログラムの実行			fork, exec, exit システムコールを紹介する。	
12 回	演習			[演習：env プログラムを作成する。]	
13 回	シェルプログラム (1)			C 言語で記述された簡単なシェルプログラムのソースコードについて解説する。[演習：シェルプログラムの実行]	
14 回	シェルプログラム (2)			組み込みコマンドの必要性を解説する。[演習：setenv, cd 等の組み込みコマンドを追加する。]	
	期末試験			環境変数、プロセス、リダイレクト、シェルについて、役割り使用方法等を試験する。	
15 回	解答返却など			試験の解答を解説する。	
【到達目標】		UNIX のシステムコールを使用したプログラムが作成できるようになる。また、背景にあるファイルシステムやプロセスの構造とシステムコールの関連が理解できる。			
【徳山高専学習・教育目標】			B1	【J A B E E 基準】	1(2)d-1
【評価法】		演習・宿題で作成したプログラム・レポート [20%]、2 回の試験の平均点 [80%] の総合評価とする。			
【テキスト】		教科書：カーニハン他、「プログラミング言語 C」(共立) 参考書：David A curry、「UNIX C プログラミング」(アスキー)			
【関連科目】		本科：システムプログラミング I(3 年)、オペレーティングシステム I、II(4,5 年)			

今回は、ファイルの読み書きについて学ぶ。教科書の P.194 ～ P.202 の内容が対応する。3年次に作ったプログラムは `stdin` から入力 (`getchar()`) し、`stdout` へ出力 (`putchar()`、`printf()`) するものだった。今回は、任意のファイルを読み書きするプログラムを作る。

C 言語では、ファイルをバイトの1次元配列 (バイトストリーム) と考える。`stdin` や `stdout`、`stderr` もバイトストリームだったので、これらと同様の手順でファイルも扱うことができる。

1. ファイルのオープン (`fopen()` 関数)

`stdin` や `stdout`、`stderr` は、特別に予めオープンされたファイルである。普通のファイルはプログラム内で明示的にオープンする必要がある。`fopen()` がファイルをオープンする関数である。

```
#include <stdio.h>
FILE *fopen(char *fname, char *mode);

例: // バイナリファイルをリードオープン
    FILE *fp;           // ファイルポインタ
    fp = fopen("a.dat", "rb");
    if (fp==NULL) {     // エラーチェック
```

`fname`: ファイル名

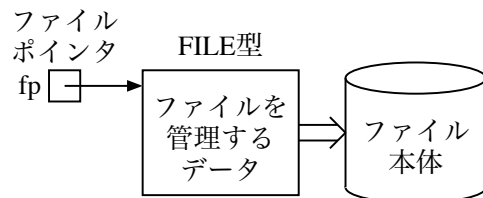
`mode`:

"r": 読み (read)

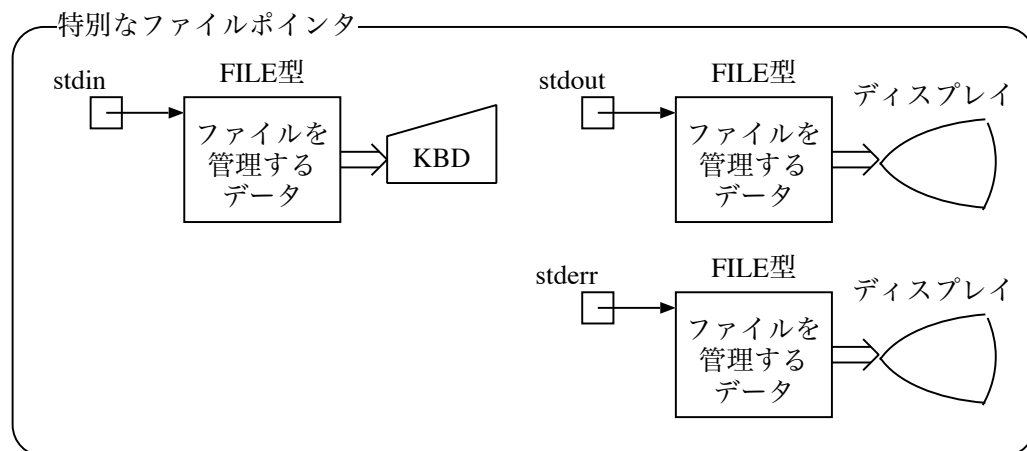
"w": 書き (write) + "b": バイナリ

"a": 追加 (append)

返り値: `FILE` 構造体を指すポインタ



`fopen` が `FILE` 型構造体領域を作って初期化する。勿論、ファイル本体のオープンもする。



2. ファイル入出力

(a) 1文字入力 (`getc()` 関数 — `getchar()` のファイル版 —)

```
#include <stdio.h>
int getc(FILE *fp);
// getchar() は次のように定義できる
// #define getchar() getc(stdin)
```

`getc` は、ファイルから読んだ1バイトの値 (0~255) を返す。EOF になると -1 を返す。(`getchar()` と同様)

```
例: ファイルから1バイト読む
FILE *fp = fopen("a.txt", "r");
int ch;           // int 型!!
ch = getc(fp);
```

左の例は、`a.txt` ファイルの最初の1バイトを変数 `ch` に読み込む。

(b) 1文字出力 (putc() 関数 — putchar() のファイル版 —)

```
#include <stdio.h>
int putc(int c, FILE *fp);
// putchar() は次のように定義できる
// #define putchar() putc(stdout)
```

例: ファイルに文字「a」を書き込む
FILE *fp = fopen("a.txt", "w");
putc('a', fp);

(c) 1行入力 (fgets() 関数)

```
#include <stdio.h>
char *fgets(char *buf,
            int size, FILE *fp);
// fgets は EOF で NULL を返す
```

buf: 1行入力バッファ
size: バッファの大きさ
fp: ファイルポインタ

例: ファイルから1行読む
FILE *fp = fopen("a.txt", "r");
char buf[100];
fgets(buf, 100, fp);

左の例は、a.txt ファイルの最初の1行を buf に入力する。buf の最後には、"\n\0" が格納される。

(d) 1行出力 (fputs() 関数 — puts() のファイル版 —)

```
#include <stdio.h>
int fputs(char *buf, FILE *fp);
// エラー時に EOF を返す
// buf は '\0' で終わる文字列
```

例: ファイルに文字列「abc\n」を書き込む
FILE *fp = fopen("a.txt", "w");
fputs("abc\n", fp);

(e) 書式付き出力 (fprintf() 関数 — printf() のファイル版 —)

```
#include <stdio.h>
int fprintf(FILE *fp,
            char *format, ...);
```

例: ファイルにデータを書き込む
FILE *fp = fopen("a.txt", "w");
fprintf(fp, "%d,%d\n", x, y);

3. ファイルのクローズ (fclose() 関数)

```
#include <stdio.h>
int fclose(FILE *fp);
```

4. エラーメッセージの出力 (perror() 関数)

直近に発生したエラーを説明するメッセージを stderr に出力する。

```
#include <stdio.h>
void perror(char *msg); // msg はエラーメッセージの先頭に付加する文字列

例: ファイルオープンに失敗のエラー処理
fp = fopen("a.txt", "r");
if (fp==NULL) {
    perror("a.txt"); // "a.txt: No such file or directory" などと表示
    exit(1);
}
```

5. プログラム例

```
// mycat.c : 複数のファイルを stdout へ連続出力
#include <stdio.h>
#include <stdlib.h> // exit のため

// ifp から ofp にコピー
void fileCopy(FILE *ifp, FILE *ofp) {
    int c;
    while ((c=getc(ifp))!=EOF)
        putc(c, ofp);
}

int main(int argc, char *argv[]) {
    if (argc==1) // ファイルが指定されていない
        fileCopy(stdin, stdout); // stdin から stdout へコピー
    else // ファイルが指定されている
        for (int i=1; i<argc; i++) { // 全てのファイルについて
            FILE *fp = fopen(argv[i], "r"); // オープンして
            if (fp==NULL) { // エラーチェックして
                perror(argv[i]);
                exit(1);
            }
            fileCopy(fp, stdout); // ファイルから stdout へコピー
            fclose(fp); // 忘れずクローズ
        }
    return 0;
}
```

実行例：

```
$ mycat <--- ファイルが指定されていない場合
aaaa <--- 入力
aaaa <--- 出力
bbbb <--- 入力
bbbb <--- 出力
^D <--- EOF

$ mycat a.txt <--- a.txt の内容を表示
hello

$ mycat b.txt <--- b.txt の内容を表示
world

$ mycat a.txt b.txt
hello <--- a.txt の内容を表示
world <--- b.txt の内容を表示
$
```

6. 演習（#1） め切 4月11日

(a) mycp コマンド

ファイルをコピーするプログラム mycp を作りなさい。mycp は、cp コマンドの簡易版です。次のように使います。エラー処理もすること。

```
$ mycp file1 file2 <--- file1 の内容を file2 にコピー
```

(b) mydiff コマンド

二つのファイルを比較して、違っている最初の行と行番号を印刷するプログラムを書きなさい。処理できる1行の最大の長さは自分で決めていい。次のように使います。エラー処理もすること。

```
$ mydiff a.txt b.txt <--- a.txt と b.txt を比較
2 行目
> world
< World
$
```