オペレーティングシステムの機能を使ってみよう
第２章 ファイル入出力システムコール

ファイル入出力システムコール

- ファイルの読み書きを行うシステムコールを勉強する.
- システムコールを直接に使用したプログラムを作成してみる.
- プログラムの作成には C 言語を用いる.
- システムコールを直接に使用する入出力を**低水準入出力**と呼ぶ.
  これまで使用してきたものは**高水準入出力**と呼ぶ.

高水準入出力と低水準入出力

- C 言語の入門で勉強した入出力関数は**高水準入出力**関数.
- システムコールを直接使用する入出力は**低水準入出力**.
- 高水準入出力関数は内部でシステムコールを利用.
- 高機能・豊富な高水準と，シンプルな低水準.

高水準入出力関数	対応するシステムコール
`fopen()`	`open` システムコール
`printf()`	`write` システムコール
`putchar()`	`write` システムコール
`fputs()`	`write` システムコール
`fputc()`	`write` システムコール
...	...
`scanf()`	`read` システムコール
`getchar()`	`read` システムコール
`fgets()`	`read` システムコール
`fgetc()`	`read` システムコール
...	...
`fclose()`	`close` システムコール

open システムコール（書式１）

- ファイルを開くシステムコール
- fopen() 関数が使用している

書式　（オープンするだけの場合）

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

解説　（書式１，２共通）

- fcntl.h をインクルードする必要がある．
- open システムコールは正常時には**ファイルディスクリプタ**（３以上の番号）を返す．[1]．
- エラーが発生した時は-1 を返す．
- エラー原因は perror() 関数で表示できる．

[1] stdin が 0，stdout が 1，stderr が 2 なので 3 以降になる．

引数

- path はオープンまたは作成するファイルのパス（名前）
- oflag はオープンの方法を表の記号定数を「|」で接続して書く．（「|」は，C 言語のビット毎の論理和演算子）

以下の一つ	と	以下のいくつか
O_RDONLY（読み出し用）		O_APPEND（追記）
O_WRONLY（書き込み用）	+	O_CREAT（作成）
O_RDWR（読み書き両用）		O_TRUNC（切詰め）
		...

使用例

```
#include <fcntl.h>
...
int fdr, fdw, fda;
fdr=open("r.txt", O_RDONLY);                  // 読み出し用にオープン
fdw=open("w.txt", O_WRONLY);                  // 書き込み用にオープン
fda=open("a.txt", O_WRONLY|O_APPEND);         // 追記用にオープン

if (fdw<0) {                                  // エラーチェック
  perror("w.txt");                            //   原因の表示
  exit(1);                                    //   エラー終了
}
```

open システムコール（書式２）

ファイルが存在しない時はファイルを自動的に作った上で開く．

書式　（ファイル作成もする場合）

oflag に O_CREAT を含む場合は，該当ファイルが存在しないなら新規作成してからオープンする．新規作成するファイルの**保護モード**を mode で指定する．

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

- mode_t は，16bit の整数型（16bit int）である．
- mode は，作成されるファイルの保護モードである．
- mode は，８進数で記述することが多い．

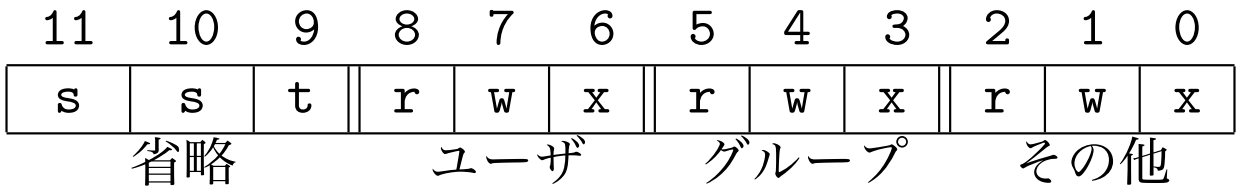
0: ---	2: -w-	4: r--	6: rw-
1: --x	3: -wx	5: r-x	7: rwx

使用例

```
fd=open("a.txt", O_WRONLY|O_CREAT, 0644);  // モードは rw-r--r--
```

ファイルの保護モード

open システムコールの第 3 引数（mode）は次のような 12bit の値である.



- 最初の 3bit の意味は難しいのでここでは説明を省略する.
- 他のビットは rwx のどれかである. rwx の意味は次の通り.

r	:	***r**ead* 可（読み出し可能）
w	:	***w**rite* 可（書き込み可能）
x	:	***x**ecute* 可（実行可能）

例えば, 第 8 ビットが 1 だった. ＝＞
ユーザ（ファイルの所有者）が read（読み出し）可能の意味.

ファイルのモードやユーザ（所有者），グループは次のようにして確認できる．

```
$ ls -l a.txt
-rw-r--r--  1 sigemura  staff  0 Apr 11 05:53 a.txt
$
```

実行結果から a.txt ファイルについて以下のことが分かる．

- モードの下位 9 ビットが 110100100 である．
- 所有者は sigemura である．
- グループは staff である．

以上を総合すると a.txt ファイルについて以下のことが分かる．

- sigemura が読み書きができる．
- staff グループに属するユーザは読むことだけできる．
- その他のユーザも読むことだけできる．

read システムコール（1）

- 読み出し用にオープン済みのファイルからデータを読む.
- ファイルの先頭から順に読み出す.
  （**シーケンシャルアクセス（順アクセス）**）

書式　（詳しくは `man 2 read` で調べる.）

```
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);
```

解説

- `unistd.h` をインクルードする必要がある.
- `ssize_t` は 64bit int 型.
- 正常時には読んだデータのバイト数（正の値）を返す.
- EOF では `0` を返す.
- エラーが発生した時は`-1` を返す.
- エラーの原因は `perror()` 関数で表示できる.

read システムコール（2）

引数

- fildes はオープン済みのファイルディスクリプタ
- buf はデータを読み出すバッファ領域を指すポインタ
- nbyte はバッファ領域の大きさ（バイト単位）

使用例 1

- fd は open システムコールでオープン済みと仮定
- buf はバッファ用の char 型の大きさ 100 の配列
- char 型は 1 バイトなので，配列全体で 100 バイト
- 3 回の read によりファイルの先頭から順に 100 バイトずつ読む

```
char buf[100];
n = read(fd, buf, 100); // 1 回目
n = read(fd, buf, 100); // 2 回目
n = read(fd, buf, 100); // 3 回目
```

read システムコール（3）

使用例 2

- ループでファイルの先頭から順にデータを読み出す例
- n の値が 0 以下になったら EOF かエラー
- EOF かエラーになったらループを終了

```
while ((n=read(fd, buf, 100)) > 0) { // 読む
    ... 読んだ n バイトのデータを処理する ...
}
```

write システムコール

- 書き込み用にオープン済みのファイルへデータを書き込む.
- ファイルの先頭から順にデータを書き込む.
  (**シーケンシャルアクセス**)
- ファイルの最後に達するまでは元々あったデータを上書きする.
- ファイルの最後に書き込むとファイル長が延長される.

書式 (詳しくは `man 2 write` で調べる.)

```
#include <unistd.h>
ssize_t write(int fildes, void *buf, size_t nbyte);
```

解説

- ファイルに実際に書き込んだデータのバイト数を返す.
- 返された値が `nbyte` と一致しない場合はエラー?

使用例 ファイルに `abc` の 3 バイトを書き込む.

```
char *a = "abc";
n = write(fd, a, 3);          // n が 3 以外ならエラーが疑われる
```

lseek システムコール（1）

- オープン済みファイルの読み書き位置を移動する．
- lseek システムコールと組み合わせることで，read，write システムコールを用いたファイルの**ランダムアクセス（直接アクセス）**が可能になる．

書式　（詳しくは `man 2 lseek` で調べる．）

```
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

解説

- `fildes` はオープン済みのファイルディスクリプタ
- `off_t` 型は 64bit `int` 型
- ファイルの現在の読み書き位置を `offset` に移動
- `offset` の意味は `whence` によって変化する．
- 正常時は新しい読み書き位置が返される．
- エラーが発生した時は`-1` を返す．
- エラーの原因は `perror()` 関数で表示できる．

lseek システムコール（2）

whence の意味

whence	意　味
SEEK_SET	offset はファイルの先頭からのバイト数
SEEK_CUR	offset は現在の読み書き位置からのバイト数
SEEK_END	offset はファイルの最後からのバイト数

使用例 fd はオープン済みのファイルディスクリプタとする．

```
lseek(fd,  200, SEEK_SET);     // 先頭から 200 バイトに移動する．
lseek(fd, -100, SEEK_CUR);    // 現在地から 100 バイト後ろに移動する．
lseek(fd,  -10, SEEK_END);     // 最後から 10 バイト後ろに移動する．
```

close システムコール

ファイルを閉じる.

書式 （詳しくは man 2 close で調べる.）

```
#include <unistd.h>
int close(int fildes);
```

解説

- オープン済みのファイルを閉じる.
- 引数はオープン済みのファイルディスクリプタ
- 多数のファイルを開くプログラムでは不要になったものをクローズしないと，同時に開くことができるファイル数の上限を超えることがある.

使用例 fd はオープン済みのファイルディスクリプタとする.

```
close(fd);
```

課題 No.1

- mycp プログラムを作る.
- 但し，open，read，write，close システムコールを用いる.
- バッファサイズより大きなファイルのコピーもできること.
- ファイルサイズがバッファサイズの整数倍とは限らない.
- open システムコールエラーチェックは必須.
- エラーメッセージは perror() 関数で表示する.

```
$ dd if=/dev/random of=srcfile bs=1024 count=10  # ランダムな内容の
10+0 records in                                  #   10KiB のファイルを作成する
10+0 records out
10240 bytes transferred in 0.001528 secs (6701462 bytes/sec)
$ mycp srcfile destfile                          # mycp プログラムを実行する
$ cmp srcfile destfile                           # コピー元とコピー先ファイルを比較する
$                                                # 内容が同じなら何も表示されない
```

課題 No.1 の解答例（1/5）

リスト 1: プログラム例（1/3）

```
#include <stdio.h>                      // perror のため
#include <stdlib.h>                     // exit のため
#include <fcntl.h>                      // open のため
#include <unistd.h>                     // read,write,close のため

//#define  BSIZ 1                       // !!バッファサイズ:変化させ性能を調べる!!
#define  BSIZ 1024                      // !!バッファサイズ:変化させ性能を調べる!!

void err_exit(char *s) {                // システムコールでエラー発生時に使用
  perror( s );                          //  エラーメッセージを出力して
  exit(1);                              //  エラー終了
}
```

- インクルードファイル
- バッファサイズの定義
- エラー終了関数

課題 No.1 の解答例（2/5）

リスト 2: プログラム例（2/3）

```
int main(int argc, char *argv[]) {
  int fd1, fd2;                          // ファイルディスクリプタ
  ssize_t len;                           // 実際に読んだバイト数
  char buf[BSIZ];                        // バッファ

  // ユーザの使い方エラーのチェック
  if (argc!=3) {
    fprintf(stderr, "Usage␣:␣%s␣<srcfile>␣<dstfile>\n", argv[0]);
    exit(1);
  }
```

- main() 関数
- 変数の宣言，バッファの宣言
- コマンドライン引数のチェック

課題 No.1 の解答例（3/5）

リスト 3: プログラム例（3/3）

```
  // 読み込み用にファイルオープン
  fd1 = open(argv[1], O_RDONLY);
  if (fd1<0) err_exit( argv[1] );  // オープンエラーのチェック

  // 書き込み用にファイルオープン
  fd2 = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC,0644);
  if (fd2<0) err_exit( argv[2] );  // オープンエラーのチェック

  // ファイルの書き写し
  while ((len=read(fd1, buf,BSIZ))>0) {
    write(fd2,buf,len);
  }

  close(fd1);
  close(fd2);
  return 0;                              // 正常終了
}
```

- open() のフラグに注目！！
- write() には len を渡すこと.

課題 No.1 の解答例（4/5）

リスト 4: 実行例（1/2）

```
$ mycp2                                    <-- コマンド行引数がない場合
Usage: mycp2 <srcfile> <dstfile>
$ mycp2 a.txt                              <-- コマンド行引数が不足の場合
Usage: mycp2 <srcfile> <dstfile>
$ mycp2 a.txt b.txt c.txt                  <-- コマンド行引数が過剰な場合
Usage: mycp2 <srcfile> <dstfile>
$ mycp2 z.txt a.txt                        <-- コピー元が存在しない場合
z.txt: No such file or directory
$ mycp2 a.txt /a.txt                       <-- コピー先が書き込み禁止の場合
/a.txt: Permission denied
$ echo aaa bbb > a.txt                     <-- a.txt を作って
$ mycp a.txt b.txt                         <-- b.txt にコピーしてみる
$ cat b.txt                                <-- b.txt の内容を確認
aaa bbb
$ echo ccc ddd > c.txt                     <-- c.txt を作って
```

- コマンドライン引数のエラーチェックの動作確認
- open に関する動作確認
- その他

課題 No.1 の解答例 (5/5)

リスト 5: 実行例（2/2）

```
$ mycp c.txt b.txt                                    <-- b.txt に上書きしてみる
$ cmp c.txt b.txt                                     <-- b.txt の内容を確認
$ dd if=/dev/random of=srcfile bs=1024 count=10       <-- 10KiB の長いファイルを作る
10+0 records in
10+0 records out
10240 bytes transferred in 0.001695 secs (6041591 bytes/sec)
$ rm destfile
$ rm destfile
rm: destfile: No such file or directory               <-- destfile が存在しない場合
$ mycp2 srcfile destfile
$ cmp srcfile destfile                                <-- 正しくコピーできている
$ dd if=/dev/random of=srcfile bs=1023 count=10       <-- 10KiB より少し短いファイル
10+0 records in
10+0 records out
10230 bytes transferred in 0.003218 secs (3179057 bytes/sec)
$ mycp2 srcfile destfile                              <-- destfile が短くなる場合
$ cmp srcfile destfile                                <-- 正しくコピーできている
```

- read/write の繰り返しに関する動作確認
- ファイル長がバッファ長の倍数でない場合の動作確認
- ファイルが短くなる場合の動作確認