

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 |
|------------------------|----------------------------|
| <code>fopen()</code> | <code>open</code> システムコール |
| <code>printf()</code> | <code>write</code> システムコール |
| <code>putchar()</code> | <code>write</code> システムコール |
| <code>fputs()</code> | <code>write</code> システムコール |
| <code>fputc()</code> | <code>write</code> システムコール |
| ... | ... |
| <code>scanf()</code> | <code>read</code> システムコール |
| <code>getchar()</code> | <code>read</code> システムコール |
| <code>fgets()</code> | <code>read</code> システムコール |
| <code>fgetc()</code> | <code>read</code> システムコール |
| ... | ... |
| <code>fclose()</code> | <code>close</code> 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() 関数を使用している

書式（オープンする場合）

```
#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。

¹stdin 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.

| | | | | | | | | | | | |
|----|----|---|-----|---|---|------|---|---|-----|---|---|
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| s | s | t | r | w | x | r | w | x | r | w | x |
| 省略 | | | ユーザ | | | グループ | | | その他 | | |

- 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` : **read** 可 (読み出し可能)

`w` : **write** 可 (書き込み可能)

`x` : **execute** 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 fd, 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

- `fildev` 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ヘータを書き込む。
- 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/urandom 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
```