ソフトウェア工学入門

第3回　システムコール

catのプログラムを組みましょう①

早速ですが、次のプログラムをcat.cという名前で作成してください。

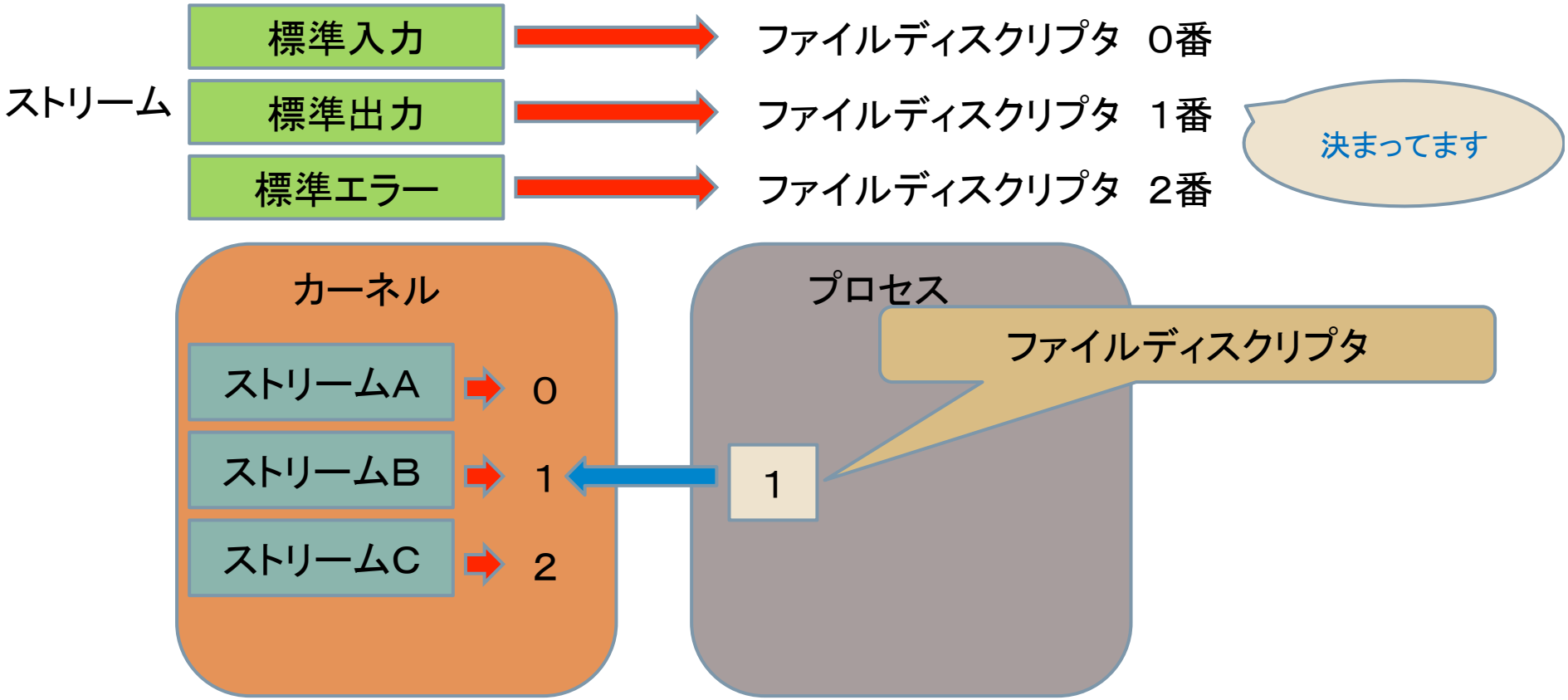
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
static void do_cat(const char *path);
static void die(const char *s);
int
main(int argc, char *argv[])
{
    int i;
    if (argc < 2) {
        fprintf(stderr, "%s: file name not given¥n", argv[0]);
        exit(1);
    }
    for (i = 1; i < argc; i++) {
        do_cat(argv[i]);
    }
    exit(0);
}
```

catのプログラムを組みましょう②

```
#define BUFFER_SIZE 2048
static void
do_cat(const char *path)
{
    int fd;
    unsigned char buf[BUFFER_SIZE];
    int n;
    fd = open(path, O_RDONLY);
    if (fd < 0) die(path);
    for (;;) {
        n = read(fd, buf, sizeof buf);
        if (n < 0) die(path);
        if (n == 0) break;
        if (write(STDOUT_FILENO, buf, n) < 0) die(path);
    }
    if (close(fd) < 0) die(path);
}
static void
die(const char *s)
{
    perror(s);
    exit(1);
}
```

ファイルディスクリプタ

ファイルディスクリプタ：　カーネルのストリームをプロセスの持つ整数値で指定



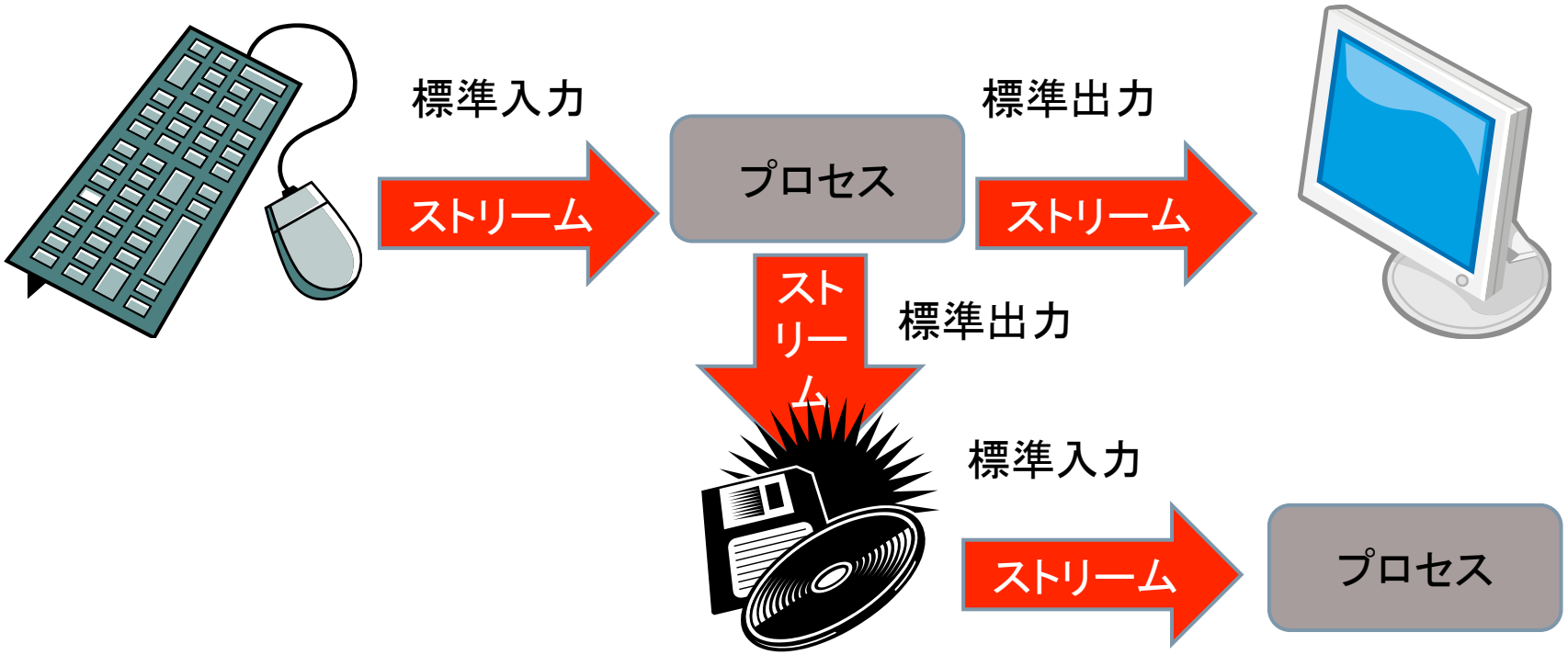
カーネルは見えないけれど、ファイルディスクリプタでストリームが理解できる

標準入力と標準出力

標準入力と標準出力って何で必要？　→　コマンドを様々に組み合わせるため

プロセスを媒介して、様々なコマンドが標準入力→標準出力を繰り返す

制御するのはシェル



ファイルの読み書き

```
#include <unistd.h>

ssize_t read( int fd , void *buf , size_t bufsize );
```

unistd.h ： readを使うためのヘッダファイル
ssize_t ： 符号付き整数
size_t ： 符号なし整数

readの意味

ファイルディスクリプタ fd番のストリームからバイト列を最大でbufsizeまで読み込み bufに格納する
正常に読み込んだ時は戻り値0を返し、エラーの時は -1 を返す

```
#include <unistd.h>

ssize_t write( int fd , void *buf , size_t bufsize );
```

writeの意味

bufsizeバイト分をbufからファイルディスクリプタfd番のストリームに書き込む
正常に読み込んだ時は書いたバイト数を返し、エラーの時は -1 を返す

ファイルを開く

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl/h>

int open ( const char *path , int flags );
```

openの意味

pathで表わされるファイルにつながるストリームを作成し、そのストリームを指す
ファイルディスクリプタを返す

第二引数はストリームの性質を表わす。主な性質は下の表

フラグ	意味
O_RDONLY	読み込み専用
O_WRONLY	書き込み専用
O_RDWR	読み書き両用

ファイルを閉じる

```
#include <unistd.h>

int close ( int fd );
```

closeの意味

ファイルディスクリプタfdで関連付けられているストリームを片付ける

基本的にはカーネルが片付けてくれるが、プログラムによってはclose()がないと
片付けられない場合もある

catプログラムを実行してみる

ビルドします

```
>gcc  -o  cat  cat.c
>ls
args  args.c  bell  bell.c  cat  cat.c  hello  hello.c
```

実行します

```
>./cat  hello.c  args.c  > out.txt
>less  out.txt
```

何と表示されるでしょうか？

コマンドライン引数で指定されたファイルを連結して出力する

catを分析　①　main

```
int
main(int argc, char *argv[])
{
    int i;
    if (argc < 2) {
        fprintf(stderr, "%s: file name not given¥n", argv[0]);
        exit(1);
    }
    for (i = 1; i < argc; i++) {
        do_cat(argv[i]);
    }
    exit(0);
}
```

コマンドライン引数が渡されているかどうかチェックしています
渡されていなければ、fprint()を実行して、エラーを表示します

全てのコマンドライン引数に対して順番にdo_cat()関数を実行します

catを分析　②　do_cat 内容1

```
static void
do_cat(const char *path)
{
    int fd;
    unsigned char buf[BUFFER_SIZE];
    int n;
    fd = open(path, O_RDONLY);
    if (fd < 0) die(path);
    for (;;) {
        n = read(fd, buf, sizeof buf);
        if (n < 0) die(path);
        if (n == 0) break;
        if (write(STDOUT_FILENO, buf, n) < 0) die(path);
    }
    if (close(fd) < 0) die(path);
}
```

open()を使い、読み込み専用でファイルを開いている
次の行では、ファイルが開けたかどうかを確認している
最後にclose()でファイルを閉じている

catを分析　③　do_cat 内容2

```
static void
do_cat(const char *path)
{
    int fd;
    unsigned char buf[BUFFER_SIZE];
    int n;
    fd = open(path, O_RDONLY);
    if (fd < 0) die(path);
    for (;;) {
        n = read(fd, buf, sizeof buf);
        if (n < 0) die(path);
        if (n == 0) break;
        if (write(STDOUT_FILENO, buf, n) < 0) die(path);
    }
    if (close(fd) < 0) die(path);
}
```

for(;;)は以下の内容を無限に繰り返せという無限ループ
read()でストリームを読み込み、write()で書き込めという操作を永遠に繰り返せという命令
無限ループはn==0つまり、ファイルがなくなったら終わる
STDOUT_FILENO は標準出力という意味
read()で読み込んだ nバイトだけ、標準出力に書き込むという意味

catを分析　④　die

```
static void
die(const char *s)
{
    perror(s);
    exit(1);
}
```

予想外の事態が発生した時に標準エラーを出力し、終了する

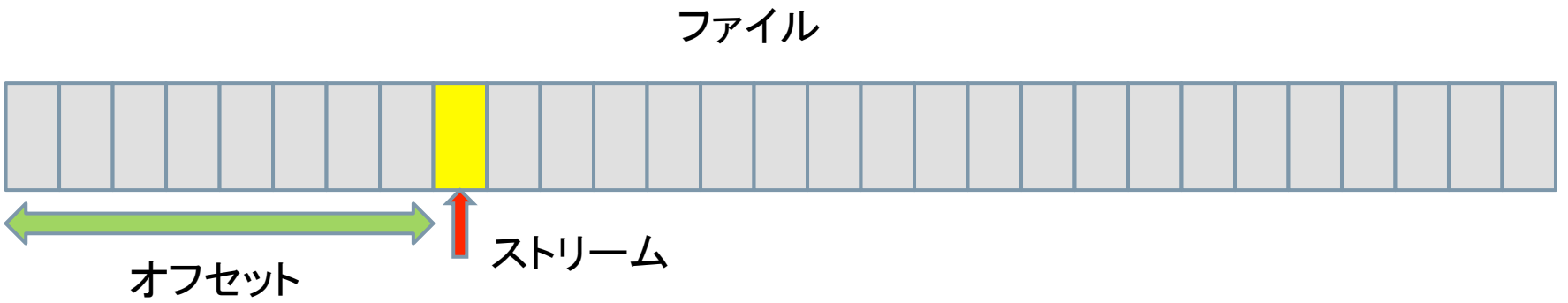
```
#include <stdio.h>

void perror( const char *s );
```

perrorの意味

エラーの起こりそうな文字列 s を渡し、エラーが発生したら標準エラーを出力する

ファイルオフセット



同じファイルディスクリプタに対して何度もread()を行うと前の続きが返ってくる

ファイルオフセット　：　ストリームが繋がっている位置情報

ファイルオフセットを操作するシステムコール　lseek()

lseek

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek( int fd , off_t offset , int whence );
```

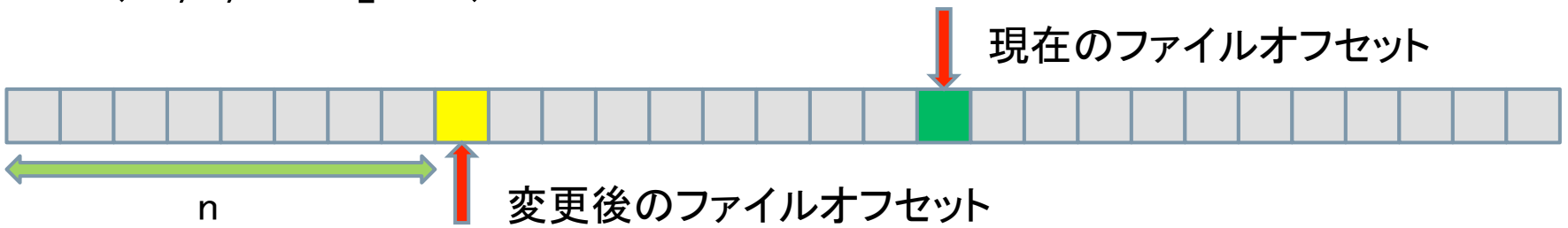
lseekの意味

ファイルディスクリプタfd内部にあるオフセットを指定した位置offsetに移動する
移動方法は3種類あり、フラグwhenceで指定する

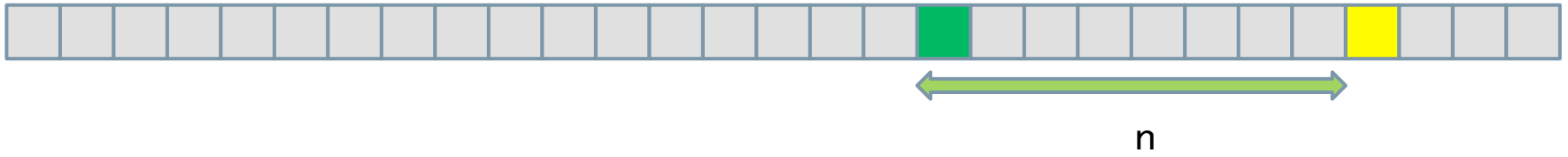
フラグ	移動先
SEEK_SET	offsetに移動する
SEEK_CUR	現在のファイルオフセット+offsetに移動する
SEEK_END	ファイル末尾+offsetに移動する

lseekの操作パターン3種

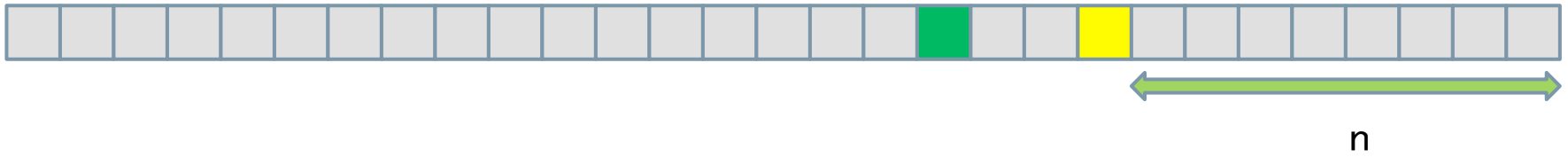
lseek(fd, n, SEEK_SET)



lseek(fd, n, SEEK_CUR)



lseek(fd, n, SEEK_END)



本日の練習問題(C言語の再復習:メモリ確保)

次のプログラムはコマンドライン引数で与えた数字を配列dataに格納します。

配列の大きさは静的に3個と決まっており、4個以上の数字は格納できません。
mallocを使用して、この制限を取り払い、4個以上のデータでも扱えるようにしなさい。

必要な道具　atoi関数　:　文字列をint型の整数に変換します

```
include <stdio.h>
#include <stdlib.h>
#define  N 3
int main (int argc, char *argv [])
{
    int i, data [N];
    if (argc - 1 > N)
            exit (1);
    for (i = 1; i < argc; i++) {
            data [i - 1] = atoi (argv [i]);
    }
    for (i = 1; i < argc; i++) {
            printf ("%ld¥n", data [i - 1]);
    }
}
```

```
#include <stdlib.h>
int atoi( const char *str );
```

戻り値:int型に変換した数値.
変換不能文字は0を返す.