今回は、ファイル操作のシステムコールについて学ぶ。つまり、前回コマンド操作で行ったリンクの作成等をプログラム中から行う方法を学ぶ。

1. ファイルの削除（unlink システムコール）
   rm コマンドは、このシステムコールを利用している。「ファイルの削除」は正確にはリンク（名前）の削除の意味である。ファイルは一つもリンクを持たなくなった時に削除される。

書式：
```
#include <unistd.h>
int unlink(char *path);
  path：削除するリンク（ファイル）
  返り値：正常=0、エラー=-1
```

使用例：
```
// ファイルの削除
if (unlink("a.txt")<0) {   // "a.txt" 削除
  perror("a.txt");         // エラー原因表示
  exit(1);                 // エラー終了
}
```

2. ディレクトリの作成（mkdir システムコール）
   mkdir コマンドは、このシステムコールを利用している。第２引数で新ディレクトリのモード (rwx) を指定する。

書式：
```
#include <sys/stat.h>
int mkdir(char *path, int mode);
  path：新規作成するディレクトリ
  mode：新しいディレクトリの rwxrwxrwx
  返り値：正常=0、エラー=-1
```

使用例：
```
// ディレクトリの作成
if (mkdir("newdir", 0755)<0) {   // "newdir" を rwxr-xr-x で作成
  perror("newdir");              // エラー原因表示
  exit(1);                       // エラー終了
}
```

3. ディレクトリの削除（rmdir システムコール）
   rmdir コマンドは、このシステムコールを利用している。空ではないディレクトリを削除することはできない。

書式：
```
#include <unistd.h>
int rmdir(char *path);
  path：削除するディレクトリ
  返り値：正常=0、エラー=-1
```

使用例：
```
// ディレクトリの削除
if (rmdir("newdir")<0) {  // "newdir" 削除
  perror("newdir");       // エラー原因表示
  exit(1);                // エラー終了
}
```

4. リンクの作成（link システムコール）

   ln コマンドは、ハードリンクを作るとき、このシステムコールを利用している。

書式：

```
#include <unistd.h>
int link(char *oldpath, char *newpath);
  oldpath：もとの名前
  newpath：新しい名前
  返り値：正常=0、エラー=-1
```

使用例：

```
// ハードリンクの作成
if (link("a.txt", "b.txt")<0) { // リンク"b.txt"を作る
  perror("link");               // "a.txt"と"b.txt"のどちらが原因か不明なので
  exit(1);                      // エラー終了
}
```

ファイルの移動（ファイル名の変更）に応用できる。

```
unlink("b.txt");                // 念のため"b.txt"を消す(エラーは無視)
if (link("a.txt", "b.txt")<0) { // リンク"b.txt"を作る
  ... エラー処理 ...
if (unlink("a.txt")<0) {        // リンク"a.txt"を消す。
  ... エラー処理 ...
```

5. シンボリックリンクの作成（symlink システムコール）

   ln コマンドは、シンボリックリンクを作るとき（-s オプション使用時)、このシステムコールを利用している。

書式：

```
#include <unistd.h>
int symlink(char *path, char *newpath);
  path：シンボリックリンクに書き込む内容
  newpath：新しい名前（シンボリックリンクの名前）
  返り値：正常=0、エラー=-1
```

使用例：

```
// シンボリックリンクの作成
if (symlink("a.txt", "b.txt")<0) { // リンク"b.txt"を作る
  perror("b.txt");                 // エラー原因は必ず"b.txt"
  exit(1);                         // エラー終了
}
```

6. ファイルの移動（ファイル名の変更）（rename システムコール）

   mv コマンドは、このシステムコールを利用している。

書式：

```
#include <stdio.h>
int rename(char *from, char *to);
  from：もとのファイル名（パス）
  to：移動後のファイル名（パス）
  返り値：正常=0、エラー=-1
```

使用例：

```
// ファイルの移動
if (rename("a.txt", "b.txt")<0) { // "a.txt" を "b.txt" に変更
  perror("rename");               // エラー原因がどっちのパスか不明
  exit(1);                        // エラー終了
}
```

7. ファイルモードの変更（chmod、lchmod システムコール）
   chmod コマンドは、このシステムコールを利用している。

書式：
```
#include <sys/stat.h>
int chmod(char *path, int mode);
int lchmod(char *path, int mode);
  path：ファイル名（パス）
  mode：モード
  返り値 :正常=0、エラー=-1
 （lchmod はシンボリックリンクを辿らない）
```

使用例：
```
// ファイルモードの変更
if (chmod("a.txt", 0644)<0) { // ファイル"a.txt"のモードを"rw-r--r--"に変更
  perror("a.txt");            // エラー原因を表示
  exit(1);                    // エラー終了
}
```

8. シンボリックリンクの内容を読む（readlink システムコール）
   ls コマンドは、このシステムコールを利用している。

書式：
```
#include <unistd.h>
int readlink(char *path, char *buf, int size);
  path：シンボリックリンクのパス
  buf：内容を読み出す領域（バッファ）
  size：領域（バッファ）のバイト単位のサイズ
  返り値 :読みだした文字数、エラー=-1
```

使用例：
```
// シンボリックリンクの読み出し
char *name = "b.txt";
char buf[100];
int n = readlink(name, buf, 99); // シンボリックリンクの内容を buf に読み出す
if (n<0) {                       // エラーチェック
  perror(name);                  // エラー原因を表示
  exit(1);                       // エラー終了
}
buf[n]='\0';                     // C 言語型の文字列として完成させる
printf("%s␣->␣%s\n", name, buf); // ls -s 表示をまねて出力
```

9. 宿題

   (a) 簡易版 UNIX コマンドの作成

   UNIX コマンド、rm、mkdir、rmdir、ln、mv の中から３つ以上を選択する。各コマンドについて上記のシステムコールを使用して簡易版を作ってみる。但し、使用方法のエラーチェック、システムコールのエラーチェックは行うこと。

   また、自分が作った簡易版と本物の機能の違いを調べる。

   ($ man 1 rm 等でマニュアルを参照できる)

   (b) rename システムコールの必要性

   UNIX は Simple is best の思想で作られてきた。そのため、専用の機能を用意しなくても既存の機能を組み合わせて実現できる場合は専用機能の追加はしないことが多い。rename システムコールの機能は link、unlink システムコールの組み合わせでも実現できそうだが rename システムコールが追加された。なぜ追加されたか考えなさい。