Linux システムコール演習 4

岡野浩三

2022 年 5 月 24 日

ここではファイリシステム周りのシステムコールおよび標準ライブラリを扱う.

1 pwd

まずは pwd と同等のプログラムを見てみよう. カレントディレクトリの情報を得るためのシステムコールは
getcwd()
である.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>

int main(int argc, char *argv[]) {
    char buf[PATH_MAX];

    if (!getcwd(buf, PATH_MAX)) {
        perror("getcwd");
        exit(1);
    }
    puts(buf);
    exit(0);
}
```

おそらく, 難しい点はないと思われるので解説は略する.

2 ls

次に ls コマンドの簡易版として引数に指定されたディレクトリ下にあるファイルまたはディレクトリ名を 1 段階のみ表示するプログラムを考える.

ディレクトリ自体も OS からみればファイルとして扱われるのでディレクトリの open はシステムコール open でも可能である．ただしディレクトリの中身を意味のあるデータとして扱うためには直接 open するよりは opendir という標準ライブラリ関数を用いるほうが便利である．opendir が成功すると DIR 型の構造体へのポインタが得られる．DIR 型は FILE 型と対応していると理解するとわかるだろう．DIR 型のポインタに対して，具体的にディレクトリの各エントリを得るための標準ライブラリ関数が readdir である．readdir はあればエントリを表す構造体 struct dirent へのポインタを順次返す．エントリがなくなれば 0 を返す．以上に基づいて作成したプログラム例を以下に示す．

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <dirent.h>

void do_ls(char *path) {
    DIR *d;
    struct dirent *ent;

    d = opendir(path);
    if (!d) {
        perror(path);
        exit(1);
    }
    while (ent = readdir(d)) {
        printf("%s\n", ent->d_name);
    }
    closedir(d);
}

int main(int argc, char *argv[]) {
    int i;

    if (argc < 2) {
        fprintf(stderr, "%s: no arguments\n", argv[0]);
        exit(1);
    }
    for (i = 1; i < argc; i++) {
        do_ls(argv[i]);
    }
    exit(0);
```

```
}
```

3 ls -l

次に ls コマンドの-l オプションの簡易版を考えよう．-l オプションは引数で指定されたファイルまたはディレクトリの詳細情報を表示する．詳細情報を得るには stat システムコールを用いる．ここでは stat システムコールとほぼ同等の動きをする lstat システムコールを用いる．lstat は成功すると struct stat の構造体を，第二引数を通じて返す.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

char* filetype(mode_t mode) {
    if (S_ISREG(mode)) return "file";
    if (S_ISDIR(mode)) return "directory";
    if (S_ISCHR(mode)) return "chardev";
    if (S_ISBLK(mode)) return "blockdev";
    if (S_ISFIFO(mode)) return "fifo";
    if (S_ISLNK(mode)) return "symlink";
    if (S_ISSOCK(mode)) return "socket";
    return "unknown";
}

int main(int argc, char *argv[]) {
    struct stat st;

    if (argc != 2) {
        fprintf(stderr, "wrong argument\n");
        exit(1);
    }
    if (lstat(argv[1], &st) < 0) {
        perror(argv[1]);
        exit(1);
    }
    printf("type\t%o (%s)\n", (st.st_mode & S_IFMT), filetype(st.st_mode));
    printf("mode\t%o\n", st.st_mode & ~S_IFMT);
```

```
    printf("dev\t%llu\n", (unsigned long long)st.st_dev);
    printf("i node number\t%lu\n", (unsigned long)st.st_ino);
    printf("rdev\t%llu\n", (unsigned long long)st.st_rdev);
    printf("nlink\t%ld\n", st.st_nlink);
    printf("uid\t%d\n", st.st_uid);
    printf("gid\t%d\n", st.st_gid);
    printf("size\t%ld\n", st.st_size);
    printf("blksize\t%lu\n", (unsigned long)st.st_blksize);
    printf("blocks\t%lu\n", (unsigned long)st.st_blocks);
    printf("atime\t%s", ctime(&st.st_atime));
    printf("mtime\t%s", ctime(&st.st_mtime));
    printf("ctime\t%s", ctime(&st.st_ctime));
    exit(0);
}
```

i-node 番号については講義で触れると思う．atime, mtime, ctime はそれぞれ，最近にアクセスされた時刻，最近に属性修正された時刻，最近に修正時刻を意味する．

演習 1
ここまでのプログラムを実際に動かしてみよ．

4 mv

次に mv コマンドの簡易版を考えよう．このコマンドを実現するのに必要なシステムコールは mv ではなく rename である．

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "%s: wrong arguments\n", argv[0]);
        exit(1);
    }
    if (rename(argv[1], argv[2]) < 0) {
        perror(argv[1]);
        exit(1);
    }
    exit(0);
}
```

解説は不要と思われる．

5 rm

次に rm コマンドの簡易版を考えよう．このコマンドを実現するのに必要なシステムコールは rm ではなく unlink である．

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {

    if (argc < 2) {
        fprintf(stderr, "%s: no arguments\n", argv[0]);
        exit(1);
    }
    int i;
    for (i = 1; i < argc; i++) {
        if (unlink(argv[i]) < 0) {
            perror(argv[i]);
            exit(1);
        }
    }
    exit(0);
}
```

6 mkdir

次に mkdir コマンドの簡易版を考えよう．このコマンドを実現するのに必要なシステムコールは mkdir である．第二引数にこのディレクトリに対するパーミッションも与える．

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
```

```
    if (argc < 2) {
        fprintf(stderr, "%s: no arguments\n", argv[0]);
        exit(1);
    }
    int i;
    for (i = 1; i < argc; i++) {
        if (mkdir(argv[i], 0777) < 0) {
            perror(argv[i]);
            exit(1);
        }
    }
    exit(0);
}
```

注意として，このパーミッションはそのまま使われることはない．実際には umask の値の否定との論理積が取られた値が使われる．デフォルトの umask の値は 022 である．したがって上述のプログラムの場合 777 & 022 が計算されて 755 の結果になる．umask の値の変更はシステムコール umask が用いられる．なお open のときにも umask の値との計算がされる．

7 rmdir

次に rmdir コマンドの簡易版を考えよう．このコマンドを実現するのに必要なシステムコールは rmdir である．なお，ファイル等のエントリが存在するディレクトリを rmdir することはできない．その用途にはコマンドでは rm -r で-r オプションの指定が必要であり，またこの-r オプションは内部で各ファイルごとあるいはディレクトリごと個別に unlink, rmdir を呼び出している．

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "%s: no arguments\n", argv[0]);
        exit(1);
    }
    int i;
    for (i = 1; i < argc; i++) {
        if (rmdir(argv[i]) < 0) {
            perror(argv[i]);
            exit(1);
```

```
        }
    }
    exit(0);
}
```

演習 2
ここまでのプログラムを実際に動かしてみよ.

8 ln

1 つのファイル実体に対して，複数の名前を与えることができる．それを行うのが ln コマンドであった．ln と cp の違いは例えば以下の操作列で確かめることができる.

```
$ echo 123 > test
$ cat test
123
$cp test test0
$ln test test1
$cat test0
123
$cat test1
123
$echo 456 >>test
$cat test
123
456
$cat test0
123
$cat test1
123
456
```

注意: 上記の実行は LINUX のファイルシステム上で行うこと．この実行系列では test test1 が同一のファイル実体を持った別々のファイル名のエントリを持っていることを意味している．このような処理をハードリンクと呼ぶ.

ln コマンドの簡易版を考えよう．ハードリンクのためのシステムコールは link である.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
int
main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "%s: wrong arguments\n", argv[0]);
        exit(1);
    }
    if (link(argv[1], argv[2]) < 0) {
        perror(argv[1]);
        exit(1);
    }
    exit(0);
}
```

9 ln -s

ハードリンクはLinuxの同一ファイルシステム内でしか作成できない．また，ディレクトリに対するハードリンクも作成できない．

このような制約をはずしてリンクが作成できるのがシンボリックリンクあるいはソフトリンクというものである．これはWindowsのショートカットファイルと同じようなもので，その新しいエントリのファイルが作成され，そのファイルの中身としてファイル実体へのパス名が記録される．シンボリックリンクを作成するシステムコールはsymlinkである．

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "%s: wrong number of arguments\n", argv[0]);
        exit(1);
    }
    if (symlink(argv[1], argv[2]) < 0) {
        perror(argv[1]);
        exit(1);
    }
    exit(0);
}
```

演習 3
ここまでのプログラムを実際に動かしてみよ.

10 ls -R

最後に ls -R の簡易版のプログラムを見てみる．システムコールとしては目新しいところはない．-R オプションを実現するにはディレクトリが見つかれば，再帰的にその先を表示すればよい．ただし，いくつか注意点があり，. や .. のエントリは再帰訪問してはいけないし，ルートディレクトリも特別扱いが必要である．またシンボリックリンクがあれば，その先を訪問するのも問題になる可能性がある．これらに注意したプログラム例が以下である．

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>

struct strbuf {
    char *ptr;
    size_t len;
};

void traverse(struct strbuf *buf);
void traverseSub(struct strbuf *buf, int first);
struct strbuf *strbuf_new(void);
void strbuf_realloc(struct strbuf *buf, size_t size);

void print_error(char *s) {
    fprintf(stderr, "%s: %s\n", s, strerror(errno));
}

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

int main(int argc, char *argv[]) {
```

```
    struct strbuf *pathbuf;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s dir\n", argv[0]);
        exit(1);
    }
    pathbuf = strbuf_new();
    strbuf_realloc(pathbuf, strlen(argv[1]));
    strcpy(pathbuf->ptr, argv[1]);
    traverse(pathbuf);
    exit(0);
}

void traverse(struct strbuf *pathbuf) {
    traverseSub(pathbuf, 1);
}

void traverseSub(struct strbuf *pathbuf, int first) {
    DIR *d;
    struct dirent *ent;
    struct stat st;

    d = opendir(pathbuf->ptr);
    if (!d) {
        switch (errno) {
        case ENOTDIR:
            return;
        case ENOENT:
            if (first) {
                die(pathbuf->ptr);
            } else {
                return;
            }
        case EACCES:    /* permission error. just print and continue */
            puts(pathbuf->ptr);
            print_error(pathbuf->ptr);
            return;
        default:
            perror(pathbuf->ptr);
```

```
            exit(1);
        }
    }
    puts(pathbuf->ptr);
    while (ent = readdir(d)) {
        if (strcmp(ent->d_name, ".") == 0)
            continue;
        if (strcmp(ent->d_name, "..") == 0)
            continue;
        strbuf_realloc(pathbuf, pathbuf->len + 1 + strlen(ent->d_name) + 1);

        /* special handling for the root directory */
        if (strcmp(pathbuf->ptr, "/") != 0) {
            strcat(pathbuf->ptr, "/");
        }
        strcat(pathbuf->ptr, ent->d_name);
        if (lstat(pathbuf->ptr, &st) < 0) {
            switch (errno) {
            case ENOENT:
                break;
            case EACCES:
                print_error(pathbuf->ptr);
                break;
            default:
                die(pathbuf->ptr);
            }
        }
        else {
            if (S_ISDIR(st.st_mode) && !S_ISLNK(st.st_mode)) {
                traverseSub(pathbuf, 0);
            }
            else {
                puts(pathbuf->ptr);
            }
        }
        *strrchr(pathbuf->ptr, '/') = '\0';
    }
    closedir(d);
}
```

```
#define INITLEN 1024

struct strbuf * strbuf_new(void) {
    struct strbuf *buf;

    buf = (struct strbuf*)malloc(sizeof(struct strbuf));
    if (!buf) {
        die("malloc(3)");
    }
    buf->ptr = malloc(INITLEN);
    if (!buf->ptr) {
        die("malloc(3)");
    }
    buf->len = INITLEN;
    return buf;
}

void strbuf_realloc(struct strbuf *buf, size_t len) {
    char *tmp;

    if (buf->len > len)
        return;
    tmp = realloc(buf->ptr, len);
    if (!tmp) {
        die("realloc(3)");
    }
    buf->ptr = tmp;
    buf->len = len;
}
```

問 1

上記の *ls -R* プログラムの動作詳細に説明せよ.

演習 4

例えば */usr/local/bin* を引数で渡されたとき，*/usr/local/bin/*下のすべてのサブディレクトリ（孫ディレクトリ以下も含む）やすべてのファイルの各種情報（-l オプションで得られる情報と同等程度の情報）を出力するプログラムを「再帰」を活用して作成し，プログラム内容，動作内容などを報告せよ.

できない場合は，独自で仕様を作り，上の *ls -R* プログラムを変更したプログラムを作成し，プログラム内容，動作内容などを報告せよ．

参考文献

[1] 青木峰郎:ふつうの Linux プログラミング Linux の仕組みから学べる gcc プログラミングの王道, ソフトバンククリエイティブ, ISBN-13: 978-4797328356, 2005.