```
    if ( (dp = opendir(fullpath)) == NULL)
        return(func(fullpath, &statbuf, FTW_DNR));
                                    /* can't read directory */

    while ( (dirp = readdir(dp)) != NULL) {
        if (strcmp(dirp->d_name, ".") == 0  ||
            strcmp(dirp->d_name, "..") == 0)
                continue;           /* ignore dot and dot-dot */

        strcpy(ptr, dirp->d_name);  /* append name after slash */

        if ( (ret = dopath(func)) != 0)     /* recursive */
            break;  /* time to leave */
    }
    ptr[-1] = 0;    /* erase everything from slash onwards */

    if (closedir(dp) < 0)
        err_ret("can't close directory %s", fullpath);

    return(ret);
}

static int
myfunc(const char *pathname, const struct stat *statptr, int type)
{
    switch (type) {
    case FTW_F:
        switch (statptr->st_mode & S_IFMT) {
        case S_IFREG:   nreg++;     break;
        case S_IFBLK:   nblk++;     break;
        case S_IFCHR:   nchr++;     break;
        case S_IFIFO:   nfifo++;    break;
        case S_IFLNK:   nslink++;   break;
        case S_IFSOCK:  nsock++;    break;
        case S_IFDIR:
            err_dump("for S_IFDIR for %s", pathname);
                        /* directories should have type = FTW_D */
        }
        break;
    case FTW_D:
        ndir++;
        break;
    case FTW_DNR:
        err_ret("can't read directory %s", pathname);
        break;
    case FTW_NS:
        err_ret("stat error for %s", pathname);
        break;
    default:
        err_dump("unknown type %d for pathname %s", type, pathname);
    }

    return(0);
}
```

このプログラムは必要以上に一般化してあるが、実際の ftw 関数を例示するためである。例えば、関数 myfunc は常に 0 を返すが、呼び出す側では 0 以外の値が戻っても処理できる。□

ファイルシステムを辿ることについてより詳しくは [Fowler, Korn and Vo 1989] を参照してほしい。UNIX の多くの標準コマンド (find、ls、tar など) で用いられている技法についてもこれを参照してほしい。4.3+BSD には、ディレクトリを辿るための新たな関数がある。マニュアルページ fts(3) を参照してほしい。

4.22 chdir、fchdir、getcwd関数

すべてのプロセスには、カレント作業ディレクトリがある。このディレクトリは、すべての相対パス名 (スラッシュで始まらないすべてのパス名) を探す出発点である。ユーザが UNIX システムにログインすると、/etc/passwd ファイルの 6 番目のフィールドで指定されるディレクトリ、つまり、ユーザのホームディレクトリがカレント作業ディレクトリとなる。カレント作業ディレクトリはプロセスの属性の 1 つであり、ホームディレクトリはログイン名の属性の 1 つである。プロセスのカレント作業ディレクトリは、chdir か fchdir 関数を呼び出して変更する。

```
#include <unistd.h>

int chdir(const char *pathname);

int fchdir(int filedes);
```

2 つの関数の戻り値: 成功ならば 0、エラーならば-1

新しいカレント作業ディレクトリを、パス名 *pathname* かオープンしたファイル記述子 *filedes* で指定できる。

fchdir 関数は POSIX.1 にはない。これは SVR4 と 4.3+BSD で使える拡張機能である。

●プログラム例●

カレント作業ディレクトリはプロセスの属性の 1 つであるため、chdir を実行するプロセスを起動したプロセスには影響しない。(プロセス間の関係について詳しくは第 8 章で述べる。) つまり、プログラム 4.8 は予想した動作をしない。

▼プログラム 4.8　chdir の例

```
#include    "ourhdr.h"

int
main(void)
{
    if (chdir("/tmp") < 0)
        err_sys("chdir failed");

    printf("chdir to /tmp succeeded\n");
    exit(0);
}
```

プログラム 4.8 をコンパイルして実行形式を mycd とする。結果はつぎのようになる。

```
$ pwd
/usr/lib
$ mycd
chdir to /tmp succeeded
$ pwd
/usr/lib
```

mycd プログラムを実行したシェルのカレント作業ディレクトリは変わっていない。シェルが直接 chdir 関数を呼ぶ必要があり、そのため cd コマンドはシェルに組み込まれているのである。□

カーネルはカレント作業ディレクトリに関する知識を管理する必要があるので、プログラムからその値を取得できるはずである。残念ながら、カーネルが各プロセスに関して管理するものは、カレント作業ディレクトリの i ノード番号と装置の識別子だけであり、ディレクトリの完全パス名は管理しない。

ここで必要となる関数は、カレント作業ディレクトリ (ドット) から始めて、(1 段階ずつドットを用いて) ディレクトリ階層を遡るものである。各ディレクトリにおいてディレクトリ項目を読み、遡ってきた直下のディレクトリの i ノードに対応する名前を探す。この手順をルートまで繰り返せば、カレント作業ディレクトリの絶対パス名が得られる。幸運にもこれを行う関数は与えられている。

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

戻り値：成功ならば *buf*、エラーならば NULL

この関数には、バッファのアドレス *buf* とそのサイズ *size* を与える必要がある。絶対パス名と終端の null バイトを保持するに十分な大きさのバッファを与えないと、エラーとなる。(2.5.7 節のパス名用の最長の領域を割り付ける議論を思い出してほしい。)

実装によっては、第 1 引数 *buf* に NULL を指定できる getcwd もある。この場合、動的に *size* バイトを割り付けるために malloc を呼び出す。これは POSIX.1 や XPG3 には含まれていないので、避けるべきである。

●プログラム例●

プログラム 4.9 は指定されたディレクトリに移り、getcwd を呼び出して作業ディレクトリ名を出力する。

▼プログラム 4.9　getcwd 関数の例

```
#include    "ourhdr.h"

int
main(void)
{
    char    *ptr;
    int     size;

    if (chdir("/usr/spool/uucppublic") < 0)
        err_sys("chdir failed");

    ptr = path_alloc(&size);    /* our own function */
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd failed");

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

プログラムを実行するとつぎのようになる。

```
$ a.out
cwd = /var/spool/uucppublic
$ ls -l /usr/spool
lrwxrwxrwx  1 root  12 Jan 31 07:57 /usr/spool -> ../var/spool
```

(図 4.10 から予想されるように) chdir はシンボリックリンクを辿るが、getcwd がディレクトリ階層木を遡るときに /var/spool に出合って、シンボリックリンク /usr/spool がこれを指すかどうかは分からない。これはシンボリックリンクの特性である。□

4.23 特殊装置ファイル

st_dev と st_rdev の 2 つのメンバーは、混同しがちである。11.9 節で ttyname 関数を書く場合に、これらのメンバーが必要であるが、規則は簡単である。