Subject: Practice on Operating System Lecture Practice File
From: IKENOYA Katsutoshi <j05002@ie.u-ryukyu.ac.jp>
Date: Wed, 01 Nov 2006 13:36:04 +0900
To: kono@ie.u-ryukyu.ac.jp

学籍番号 : 055702B

・開発環境
OS : Mac OS X 10.3.9
コンパイラ : gcc version 3.3 20030304 (Apple Computer, Inc. build 1666)

・実行環境
同上

・Makefileの説明
Makefileのある場所でmakeコマンドをすると、Makefileに従って
プログラムを実行する。
Makefileの基本的な書き方は以下の通り。

ターゲット:依存するファイル
<TAB>コマンド

また、複数のコマンドを同時に実行する場合は以下のようにすればよい。

all:ターゲット1 ターゲット2
ターゲット1:依存するファイル
<TAB>コマンド
ターゲット2:依存するファイル
<TAB>コマンド

実際に以下のMakefileを実行してみた。

#ファイルの場所:/net/home/y05/j05002/OS/report1/sample-source

```
all:file-copy  stdio-thru
file-copy:file-copy.c
cc  -o  file-copy  file-copy.c
stdio-thru:stdio-thru.c
cc  -o  stdio-thru  stdio-thru.c
```

実行結果
```
[j05002@sample-source]%  make
cc  -o  file-copy  file-copy.c
cc  -o  stdio-thru  stdio-thru.c
```

課題1 teeプログラム

```
/*
mytee.c -- tee コマンドと似た動きをするコマンド
ソースの場所:  /home/y05/j05002/OS/report1/ex1/mytee.c
プログラム作成に要した時間: 30分
*/
#include  <stdio.h>  /*  stderr  */
#include  <fcntl.h>  /*  open(2)  */
#define  BUFFSIZE  1024

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

```
mytee( filename )
char *filename;
{
省略
to_fd = open( filename,O_WRONLY|O_CREAT|O_TRUNC,0666 );

省略

while( (rcount=read(0,buff,BUFFSIZE)) > 0 )
{

if( write(1,buff,rcount)== -1 )
{
perror("stdout");
exit( 1 );
}

if( write(to_fd,buff,rcount)== -1 )
{
perror( filename );
exit( 1 );
}

}

close( to_fd );

}
```

・実行結果

```
[j05002@ex1]% ./mytee test.txt
begin
begin
test
test
end
end
[j05002@ex1]% cat test.txt
begin
test
end
```

teeコマンドでmakeコマンドの実行結果をファイルに保存し、usrにメールを送る方法は、usrがメールを受け取った事でmakeコマンドの実行終了を知る事ができて便利である。
tailコマンドはファイルの最後の部分を（指定した行数だけ）表示するというコマンドであるが、「-f」オプションを指定することにより、ファイルの最後を監視し、新しいデータ（行）が追加されるたびにそれを表示するという動作をする。これを使ってログ・ファイルの最後を表示させておけば、新しいログ・イベントをリアルタイムに監視することができるという利点がある。
以下の方法の短所は、コマンドの終了時が分からないということである。

```
% make >& make.out &
% tail -f make.out
```

課題2 cat コマンド

```
/*
mycat.c -- cat コマンドと似た動きをするコマンド
ソースの場所: /home/y05/j05002/OS/report1/ex2/mycat.c
プログラム作成に要した時間: 40分
*/
#include <stdio.h>/* stderr */
```

```
#include <fcntl.h>/* open(2) */
#define BUFFSIZE 1024

main( argc, argv )
int argc;
char *argv[];
{
if( argc >= 2 ){
int i;
for ( i = 1; i < argc; i++ ) {
if ( strcmp(argv[i] , "-")== 0) /* 引数に文字列「-」があるかかどうか調べ
る */
mycat2();
else
mycat1(argv[i]);
}
}

else if( argc == 1 ){
mycat2();
exit(0);
}
}

mycat1( argv )
char *argv;
{
int from_fd;
char buff[BUFFSIZE] ;
int rcount1;

from_fd = open( argv, O_RDONLY );
if( from_fd == -1 )/* エラーが起きたとき */
{
perror( "stdout" );/* ファイル名とエラーメッセージを表示し、 */
exit( 1 );/* プロセスを終了する。 */
}

while( (rcount1=read(from_fd,buff,BUFFSIZE)) > 0 )
{
if( write(1,buff,rcount1)== -1 ){
perror( "stdout" );
exit( 1 );
}
}
}

mycat2()
{
int rcount ;
char buff[BUFFSIZE] ;

while( (rcount=read(0,buff,BUFFSIZE)) > 0 ){

if( write(1,buff,rcount)== -1 ){
perror("stdout");
exit( 1 );
}
}
}
```

・実行結果

```
[j05002@ex2]% cat test.txt
begin
```

```
[j05002@ex2]%  cat  text.txt
end
[j05002@ex2]%  ./mycat  test.txt  -  text.txt
begin
test
test
test2
test2
end
```

・プログラムの説明
mycat.cではまず、引数の数を調べ、引数が 1 個以上の場合は
strcmp() ライブラリ関数を用いて引数の中に"-"が入っているか
いないかを引数1個ずつ順番に調べ、無ければmycat1()を
呼び出し、入っていればmycat2()を呼び出す。
mycat1()は引数として渡されたファイルの内容を標準出力に
出力する関数で、mycat2()は標準入力から入力された文字列
を標準出力に出力する関数である。
引数が無い場合はmycat2()を呼び出す。

課題3 ls -l コマンドの仕組み

```
/*
myls-l.c -- ls -l filename と似た動きをするプログラム
ソースの場所:  /home/y05/j05002/OS/report1/ex3/myls-l.c
プログラム作成に要した時間: 40分
*/
#include  <sys/types.h>
#include  <sys/stat.h>
#include  <stdio.h>
#include  <pwd.h>

main( argc, argv )
int  argc;
char  *argv[];
{
省略
ls_l( argv[1] ); /* 引数はファイル */
}

ls_l( path )
char  *path;
{
省略

/* ファイルを識別 */
switch ( buf.st_mode & S_IFMT ) {

case  S_IFREG:
printf("-");
break;

case  S_IFDIR:
printf("d");
break;

default:
printf("  ");
break;

}

for(i = 0; i <= 8; i++) { /* st_modeを英文字に変換 */
if((buf.st_mode  &  mode[i])  !=  mode[i]){
```

```
m_print[i]='-';
}
}

printf("%s",m_print); /* 英文字に変換したst_modeを表示 */
printf(" ");
printf("%3d ",buf.st_nlink); /* リンク数 */
uid_print(buf.st_uid); /* ファイルの所有者 */
printf("%7d ",(int)buf.st_size); /* ファイルの大きさ */
ltime = localtime(&buf.st_mtime); /* 取得した時間を日本時間に変換 */
printf("%s ",mon[ltime->tm_mon]); /* 月を英語表記に変換 */
printf("%2d ",ltime->tm_mday); /* 日付 */
printf("%2d:",ltime->tm_hour); /* 時間 */
printf("%2d ",ltime->tm_min); /* 分 */
printf("%s¥n",path);
}

/*
struct stat の st_uid フィールドには、uid が入っている。これを数値とし
てではなく、ログイン名として表示するためには、次の関数 uid_print() を
使うとよい。引数は、uid_t 型（unsigned short）である。
*/

uid_print( uid )
uid_t uid ; /* unsigned short, in <sys/types.h > */
{
省略
}
```

・実行結果

```
[j05002@ex3]% ls -l myls-l.c
-rw-r--r-- 1 j05002 j05002 2125 Oct 30 17:23 myls-l.c
[j05002@ex3]% ./myls-l myls-l.c
-rw-r--r-- 1 j05002 2125 Oct 30 17:23 myls-l.c
```

http://www.ie.u-ryukyu.ac.jp/~kono/lecture/os/ex/file/src/stat.c
にあるファイルの属性を調べ、画面に出力するプログラムである
stat.c を用いてls -l で表示される時刻はどの時刻なのかを確かめてみた。

```
[j05002@sample-source]% ./a.out text.txt
省略
atime: Mon Oct 30 12:58:47 2006
mtime: Sun Oct 29 13:34:52 2006
ctime: Mon Oct 30 09:58:32 2006
省略
[j05002@sample-source]% ls -l text.txt
-rw-r--r-- 1 j05002 j05002 15 Oct 29 13:34 text.txt
```

したがって ls -l はmtime、すなわち最終更新時刻を表示している
ことが分かる。
また、ls -l ではなく ls -lu とすると最終アクセス時刻を表示する。
ls -lc とするとファイルの作成時刻を表示する。

atime・mtime は touch コマンドを用いて時刻を変更することができる。
touch -a で最終アクセス時刻を変更でき、 touch -m で最終変更時刻を
変更できる。
ctimeはchmodコマンドを使うと現在の時刻に変更することができる。
結果は以下の通り。

```
[j05002@sample-source]% ./a.out text.txt
省略
atime: Tue Oct 31 14:42:50 2006
mtime: Tue Oct 31 14:42:50 2006
```

```
ctime: Tue Oct 31 14:42:50 2006
省略
[j05002@sample-source]% touch -a text.txt
[j05002@sample-source]% ls -lu text.txt
-rw-r--r-- 1 j05002 j05002 15 Oct 31 15:02 text.txt
[j05002@sample-source]% touch -m text.txt
[j05002@sample-source]% ls -l text.txt
-rw-r--r-- 1 j05002 j05002 15 Oct 31 15:03 text.txt
[j05002@sample-source]% chmod 744 text.txt
[j05002@sample-source]% ls -lc text.txt
-rwxr--r-- 1 j05002 j05002 15 Oct 31 15:05 text.txt*
```

課題4 シンボリック・リンクの内容の表示

```
/*
myls-l2.c -- ls -l filename と似た動きをするプログラム(シンボリックリン
クに対応)
ソースの場所: /home/y05/j05002/OS/report1/ex4/myls-l2.c
プログラム作成に要した時間: 3時間
*/
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <pwd.h>

main( argc, argv )
int argc;
char *argv[];
{
省略
ls_l( argv[1] ); /* 引数はファイル */
}

ls_l( path )
char *path;
{
省略
char line[256];

if ( lstat ( path,&buf ) == -1 ) { /* ファイルの属性取得 */
perror( path );
exit(1);
}

/* ファイルを識別 */
switch ( buf.st_mode & S_IFMT ) {
case S_IFREG:
printf("-");
break;

case S_IFDIR:
printf("d");
break;

case S_IFLNK: /* シンボリックリンクを識別 */
printf("l");
flag = 1;
break;

default:
printf(" ");
break;
}
```

```
for(i = 0; i <= 8; i++) { /* st_modeを英文字に変換 */
if((buf.st_mode & mode[i]) != mode[i]){
m_print[i]='-';
}

}



printf("%s",m_print); /* 英文字に変換したst_modeを表示 */
省略
printf("%2d ",ltime->tm_min); /* 分 */

if (flag == 1) { /* もし引数がシンボリックリンクなら */
printf("%s",path);
printf(" -> ");
j = readlink(path, line, 256);
line[j] = '¥0';
printf("%s ¥n",line);
}
else{
printf("%s ¥n",path);
}
}

/*
struct stat の st_uid フィールドには、uid が入っている。これを数値とし
てではなく、ログイン名として表示するためには、次の関数 uid_print() を
使うとよい。引数は、uid_t 型（unsigned short）である。
*/

uid_print( uid )
uid_t uid; /* unsigned short, in <sys/types.h > */
{
省略
}
```

・実行結果

```
[j05002@ex4]% ls -l link1
lrwxr-xr-x 1 j05002 j05002 13 Oct 29 15:15 link1 -> myls-l_link.c
[j05002@ex4]% ./myls-l2 link1
lrwxr-xr-x 1 j05002 13 Oct 29 15:15 link1 -> myls-l_link.c
```

このプログラムは課題3のプログラムで用いたstat()をlstat()に変更し、
ファイルの識別にシンボリックリンクを加え、シンボリックリンクの出力
の仕方などを追加しただけで、他はあまり変わっていない。
stat()はpathで指定されたファイルの状態を取得するが、lstat()はpathが
シンボリックリンクの場合、リンクが参照しているファイルではなく、
リンク自信の状態を取得する点が異なる。

課題5

```
/*
myls-l3.c -- ls -l filename と似た動きをするプログラム(よりls -lに近くし
たもの)
ソースの場所: /home/y05/j05002/OS/report1/ex5/myls-l3.c
プログラム作成に要した時間: 3時間
*/
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <pwd.h>
```

```
#include <fcntl.h> /* open(2) */
#include <sys/dirent.h> /* getdirentries(2) */
#define BUFFSIZE 1024

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

if ( lstat ( argv[1],&fir ) == -1 ) { /* 属性を取得 */
perror( argv[1] );
exit(1);
}

if ( S_ISDIR(fir.st_mode) ) { /* もしargv[1]がディレクトリなら */
fd = open( argv[1],O_RDONLY ); /* 読み込む */
if( fd == -1 ) {
perror( argv[1] );
exit( 1 );
}

/*
ここでは、malloc() を用いて BUFFSIZE 分のメモリを確保する。
*/
buff = malloc( BUFFSIZE );
if( buff == 0 ) {
perror("memory");
exit( 1 );
}

/* ディレクトリ内のファイル名を取得して「ls_l()」に渡す */

while((rcount=getdirentries(fd, buff, BUFFSIZE, &pointer))>0) {
for( p = (struct dirent *)buff ; (char *)p < &buff[rcount] ;
p=(struct dirent *) ((int)p+(p->d_reclen)) ) {
ls_l(p->d_name, argv[1]);
}
}
close( fd );
free( buff );
}
else { /* argv[1]がディレクトリでなければ */
ls_l( argv[1],"./"); /* argv[1]がファイル名なのでカレントディレクトリを
渡す */
}

}

ls_l( path , dir )
char *path;
char *dir;
{
省略

if ( lstat ( path,&buf ) == -1 ) { /* ファイルの属性取得 */
perror( path );
exit(1);

}

chdir(dir); /* 指定したディレクトリにチェンジディレクトリ */
if ( lstat ( path,&buf ) == -1 ) {
perror( path );
```

```
exit(1);
}

/* ファイルを識別 */

switch ( buf.st_mode & S_IFMT ) {

省略

}

      省略

if (flag == 1){ /* もし引数がシンボリックリンクなら */
省略
}
else{
省略
}
}

/*
struct stat の st_uid フィールドには、uid が入っている。これを数値とし
てではなく、ログイン名として表示するためには、次の関数 uid_print() を
使うとよい。引数は、uid_t 型（unsigned short）である。
*/

uid_print( uid )
uid_t uid; /* unsigned short, in <sys/types.h > */
{
省略
}
```

・実行結果

```
[j05002@ex5]% ls -l ./
total 28
-rwxr-xr-x 1 j05002 j05002 16088 Oct 30 17:40 myls-l3*
-rw-r--r-- 1 j05002 j05002 3696 Oct 30 17:37 myls-l3.c
lrwxr-xr-x 1 j05002 j05002 36 Oct 26 20:26 s-link ->
/Users/j05002/OS/report1/ex1/mytee.c
[j05002@ex5]% ./myls-l3 ./
drwxr-xr-x 6 j05002 204 Oct 30 17:39 .
drwxr-xr-x 11 j05002 374 Oct 30 18:15 ..
-rw-r--r-- 1 j05002 6148 Oct 30 18:16 .DS_Store
-rwxr-xr-x 1 j05002 16088 Oct 30 17:40 myls-l3
-rw-r--r-- 1 j05002 3696 Oct 30 17:37 myls-l3.c
lrwxr-xr-x 1 j05002 36 Oct 26 20:26 s-link ->
/Users/j05002/OS/report1/ex1/mytee.c
```

このプログラムは課題4のプログラムにopen(), getdirentries()システム・コール
を加えて、より"ls -l"の表示に近づけたもの。
open()でディレクトリを開いて読み込み、getdirentries()でディレクトリのエ
ントリを取得し、その一つ一つを関数ls_l()に渡してファイルの種類などを調べ
ている。

課題6 ディレクトリの内容の検索

```
/*
dir-lookup.c -- ディレクトリの内容を検索するプログラム
ソースの場所： /home/y05/j05002/OS/report1/ex6/dir-lookup.c
プログラム作成に要した時間: 1時間40分
*/
#include <stdio.h> /* stderr */
#include <fcntl.h> /* open(2) */
```

```
#include  <sys/stat.h>
#include  <sys/types.h>  /*  getdents(2)  */
#include  <sys/dirent.h>  /*  getdents(2)  */
#include  <stdlib.h>
#define  BUFFSIZE  1024

extern  char  *malloc();

main(  argc,  argv  )
int  argc;
char  *argv[];
{
省略
dir_lookup(  argv[1],  argv[2]  );
}

dir_lookup(  dirname,  filename  )
char  *dirname;
char  *filename;
{
省略

if  (  lstat  (  dirname,  &dir  )  ==  -1  )  {  /*  argv[1]の属性を取得  */
perror(  dirname  );
exit(1);
}

if  (  S_ISDIR(dir.st_mode)  )  {
flag  =  1;
}

else  {  /*  ディレクトリでなければエラーメッセージ  */
fprintf(  stderr,  "%s  is  not  directory  ¥n",dirname);
}


chdir(dirname);
if  (  lstat  (  filename,  &fil  )  ==  -1  )  {  /*  argv[2]の属性を取得  */
perror(  filename  );
exit(1);
}

if  (  !  S_ISREG(fil.st_mode)  )  {  /*  ファイル名じゃなければエラーメッセージ  */
fprintf(  stderr,  "%s  is  not  filename  ¥n",  filename);
exit(1);
}


if  (  flag  ==  1  )  {
chdir("../");
fd  =  open(  dirname,O_RDONLY  );
if(  fd  ==  -1  )  {
perror(  dirname  );
exit(  1  );
}

buff  =  malloc(  BUFFSIZE  );
if(  buff  ==  0  )  {
perror("memory");
exit(  1  );
}

while((rcount=getdirentries(fd,  buff,  BUFFSIZE,  &pointer))>0)  {
for(  p  =  (struct  dirent  *)buff  ;  (char  *)p  <  &buff[rcount]  ;
```

```
p=(struct dirent *) ((int)p+(p->d_reclen)) ) {
if ( strcmp(filename, p->d_name ) == 0 ) {
printf("ino : %d ",fil.st_ino);
printf("path: %s¥n",p->d_name);
}

}
}

close( fd );
free ( buff );

}
}
```

・実行結果

```
[j05002@ex6]% ./dir-lookup test test.txt
ino : 2215981 path: test.txt
[j05002@ex6]% ls -i ./test/test.txt
2215981 ./test/test.txt
```

このプログラムは、
http://www.ie.u-ryukyu.ac.jp/~kono/lecture/os/ex/file/src/dir-list.c
にある、ディレクトリの内容を表示させるプログラムdir-list.cを参考
にし、getdirentries() システム・コールを使うところで、strcmp()
ライブラリ関数を使い、ディレクトリの中にプログラムを実行する
ときに指定したファイル名と一致する文字列が存在するかを調べて
一致するものがあればそのiノード番号とファイル名を表示させるように
作成した。

課題7 複数のディレクトリに渡るディレクトリ内容の検索

```
/*
dir-lookup2.c -- 与えられたファイル名から、そのiノード番号を調べるプログラム
ソースの場所: /home/y05/j05002/OS/report1/ex7/dir-lookup2.c
プログラム作成に要した時間: 2時間30分
*/
#include <stdio.h>/* stderr */
#include <fcntl.h>/* open(2) */
#include <sys/stat.h>
#include <sys/types.h>/* getdents(2) */
#include <sys/dirent.h>/* getdents(2) */
#include <stdlib.h>
#define BUFFSIZE 1024

extern char *malloc();

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

int i = 0, j = 0, k = 0;
char name[15][255]; /* argv を格納する配列 */
```

省略

```
if ( argv[1][0] == '/' ) {
chdir( "/" );
i ++;
}

for( i ; i <= strlen(argv[1])-1; i++ ) {
```

```
if ( argv[1][i] == '/' ) {
j++;
k = 0;
chdir( name[j] );
}
else {
name[j][k] = argv[1][i];
k++;
}
}
k = 0;
if ( argv[1][i-1] == '/' ) { /* もし argv[1] の最後の１文字が「/」なら */
k = j-2 ;
}
else {
k = j -1 ;
}

for( j = 0; j <= k; j++ ) {
if( chdir( name[j] ) == -1) {
perror(name[j]);
exit(1);
}
}

dir_lookup( name[j-1], name[j] );

}

dir_lookup( dirname, filename )
char *dirname;
char *filename;
{
struct stat dir;
struct stat fil;
int fd ;
struct dirent *p ;
char *buff ;
int rcount ;
long pointer;
int flag = 0;

chdir( "../" );
if ( lstat ( dirname, &dir ) == -1 ) {
perror( dirname );
exit(1);
}

if ( S_ISDIR(dir.st_mode) ) {
flag = 1;
}
```

省略

```
if ( flag == 1 ) {
chdir("../");
fd = open( dirname,O_RDONLY );
if( fd == -1 ) {
perror( dirname );
exit( 1 );
}
```

省略

```
close( fd );
free ( buff );

}
}
```

・実行結果

```
[j05002@ex7]%  ./dir-lookup2  /Users/j05002/OS/report1/ex7/dir-lookup2.c
ino  :  2217134  path:  dir-lookup2.c
[j05002@ex7]%  ls  -i  /Users/j05002/OS/report1/ex7/dir-lookup2.c
2217134  /Users/j05002/OS/report1/ex7/dir-lookup2.c
```

・プログラムの説明

引数として/Users/j05002/OS/report1/ex7/dir-lookup2.c を与えて
実行すると、まず引数の最初が"/"なのでルートへcdする。
その後、"/"がくるまで一文字ずつ配列name[j][k]へ格納していく。
最後まで格納したとき、配列nameは次のようになっている

```
name[0] = User
name[1] = j05002
name[2] = OS
name[3] = report1
name[4] = ex7
name[5] = dir-lookup.c
```

ここでjの値は5となっているので、dirnameとしてname[j-1], filename
としてname[j]を関数dir_lookup()に渡してやれば、課題6と同じ状況になるので
iノード番号を調べることができる。