```
:::::::::::::: perror_demo.c ::::::::::::::
#include       <stdio.h>
#include       <errno.h>

/*
 * program   perror_demo.c
 * purpose   show how errno is set by syscalls
 *           and how perror prints the associated message
 *   usage   perror_demo
 *   action  tries to open() a file for a given mode
 *           then, if error, prints errno and message
 */

main()
{
        char    name[100];                  /* filename            */
        int     mode;                       /* second arg to open  */

        printf("Open what file or dir? ");
        gets(name);
        printf("   Open %s for 0,1, or 2? ", name);
        scanf("%d", &mode);
        if ( open( name, mode ) == -1 ){
                printf("Oh No!, errno %d just occurred\n", errno);
                perror( name );                      /* print message      */
        }
        else
                printf("%s opened ok\n", name );
}
:::::::::::::: burrow ::::::::::::::
#!/bin/sh
#
# script   burrow
# purpose  show how to dig deep holes in a file system
# method   mkdir x; cd x; repeat  (like on shampoo bottles)
# warning  sure way to antagonize system managers
# note     only run this on your own system, not on one managed by others
#

        while true
        do
                mkdir deepdir
                cd deepdir
        done
:::::::::::::: ls-i.c ::::::::::::::
#include       <stdio.h>
#include       <sys/types.h>
#include       <sys/stat.h>

main(int ac, char *av[])
{
        while( --ac )
                info( *++av );
}

info( char *s )
{
        struct stat stb;

        if ( stat( s, &stb ) != -1 )
                printf("%s\t%d\t%x", s, stb.st_ino, (int) stb.st_dev);
        putchar('\n');
}
:::::::::::::: rm1.c ::::::::::::::
#include       <stdio.h>


main(int ac, char *av[])
{
        while(--ac){
                if ( unlink( *++av ) == -1 )
                        perror(*av);
                else
                        printf("%s is gone\n", *av);
        }
}
```

```
:::::::::::::: rm2.c ::::::::::::::

#include       <stdio.h>

main()
{
        char    name[100];
        char    confirm[100];

        while ( printf("delete what file? "), gets(name) )
        {
                printf("about to unlink %s, ok? ", name );
                gets(confirm);
                if ( *confirm == 'y' )
                        if( unlink(name) == -1 )
                                perror(name);
                        else
                                printf("%s is gone\n", name);
        }
}
:::::::::::::: mv1.c ::::::::::::::

#include       <stdio.h>
#include       <errno.h>

/*
 * program  mv1.c
 * purpose  show how to use link and unlink to rename a file
 * notes    checks errno to make it more adaptable
 * note     This version could be replaced by rename() but it
 *          works differently when the target exists
 */

main(int ac, char *av[])
{
        extern int errno ;

        if ( link( av[1], av[2]) != -1 )
                unlink( av[1] );

        else if ( errno == EEXIST )           /* name2 already in use */
        {
                if (unlink( av[2] ) == -1 )   /* not any more               */
                {
                        perror(av[2]);
                        exit(1);
                }
                main(ac,av);
        }
}
```

```
:::::::::::::: mv2.c ::::::::::::::

#include       <stdio.h>
#include       <sys/syslimits.h>

/**
 **     mv2   version 1
 **
 **            if last arg is a dir, then move all files into there
 **            if two args and last is a file, then move first to
 **            second.  will clobber second if exists.
 **            Uses isadir .
 **
 **            problems: 1) what if an arg is a dir?
 **                      2) what about paths in source files?
 **/

main( int ac, char *av[] )
{
        if ( ac < 3 ){
                fprintf( stderr, "too few args\n");
                exit(1);
        }

        /*
         *      if last arg is a dir, then move args to there
         */
        if ( isadir( av[ac-1] ) )
                mv_to_newdir(ac-1,av,av[ac-1]);

        /*
         *      last arg is not a dir, so must be two args only
         */
        else if ( ac != 3 ){
                fprintf( stderr, "too many args\n");
                exit(1);
        }
        else mv(av[1], av[2]);
}

mv_to_newdir( int ac, char *av[], char *newdir )
/*
 *      move av[1], av[2], ... av[ac-1] into newdir
 */
{
        char    newpath[PATH_MAX];
        int     pos ;

        for( pos = 1 ; pos<ac; pos++ ){
                sprintf( newpath , "%s/%s", newdir, av[pos]);
                mv( av[pos] , newpath );
        }
}

mv( oldname, newname )
char *oldname, *newname;
{
        if ( link(oldname, newname) == -1 || unlink(oldname) == -1 ){
                fprintf(stderr,"error mv'ing %s to %s\n", oldname, newname);
                exit(1);
        }
}
```

```
:::::::::::::: spwd.c ::::::::::::::::

#include       <stdio.h>
#include       <sys/types.h>
#include       <sys/stat.h>
#include       <dirent.h>

/**
 **     a simplified version of pwd
 **
 **     starts in current directory and recursively
 **     climbs up to root of filesystem, prints top part
 **     then prints current part
 **
 **     uses readdir() to get info about each thing
 **/

ino_t   get_inode();

int
main()
{
        printpathto( get_inode( "." ) );      /* print path to here  */
        putchar('\n');                        /* then add newline    */
        return 0;
}

printpathto( ino_t this_inode )
/*
 *      prints path leading down to an object with this inode
 *      kindof recursive
 */
{
        int     my_inode ;
        char    its_name[BUFSIZ];

        chdir( ".." );                                  /* up one dir  */

        inum_to_name( this_inode , its_name );               /* get its name       */

        my_inode = get_inode( "." );                    /* print head  */
        if ( my_inode != get_inode( ".." ) )            /* of pathname */
                printpathto( my_inode );                /* recursively */
        printf("/%s", its_name );                       /* now print   */
                                                        /* name of this       */
}
```

```
inum_to_name(ino_t inode_to_find , char *namebuf)
/*
 *      looks through current directory for a file with this inode
 *      number and copies its name into namebuf
 */
{
        DIR             *dir_ptr;               /* the directory */
        struct dirent   *direntp;               /* each entry   */

        dir_ptr = opendir( "." );
        if ( dir_ptr == NULL ){
                fprintf(stderr, "cannot open a directory\n");
                exit(1);
        }

        /*
         * search directory for a file with specified inum
         */

        while ( ( direntp = readdir( dir_ptr ) ) != NULL )
                if ( direntp->d_ino == inode_to_find )
                {
                        strcpy( namebuf, direntp->d_name );
                        closedir( dir_ptr );
                        return 0;
                }
        fprintf(stderr, "error looking for inum %d\n", inode_to_find);
        exit(1);
}

ino_t
get_inode( char *fname )
/*
 *      returns inode number of the file
 */
{
        struct stat info;

        if ( stat( fname , &info ) == -1 ){
                fprintf(stderr, "Cannot stat ");
                perror(fname);
                exit(1);
        }
        return info.st_ino;
}
```