Unix Files & Directories [1]

Alex Delis
delis -at+ pitt.edu

November 2012

[1] Acknowledgements to Prof. T. Stamatopoulos, M. Avidor, Prof. A. Deligiannakis, S. Evangelatos, Dr. V. Kanitkar and Dr. K. Tsakalozos.

Access Rights

- Every file/catalog belongs to the user that created it.
- Every user belongs to at lest one group (postgrads, undergrads, dep, users, etc)
- Every file is provided with 10 characters
  - \- rwx r-x r-x
    user group others
  - 1st character is either “d” (directory) or “-” (file)
  - Three groups of characters — read/write/execute options

Permission Rules for files/directories

	File	Directory
r	Read or Copy a file	Read content in directory
w	Change or delete a file	Add or delete entries (files) in directory using commands
x	Run executable file	Reference or move to directory (without seeing names of other files)

Hard Links with ln

```
ad@sydney:~/Set001/Samples$ ln bbbb myhardlink
ad@sydney:~/Set001/Samples$ ls -l
total 10356
-rwxr-xr-x 1 ad ad       15 2010-02-18 12:25 aaa
-rwxr-xr-x 2 ad ad     1200 2010-02-18 12:27 bbbb
-rwxr-xr-x 1 ad ad        0 2010-02-18 12:25 delis
-rwxr-xr-x 1 ad ad      183 2010-02-18 12:25 lista
-rwxr-xr-x 2 ad ad     1200 2010-02-18 12:27 myhardlink
-rwxr-xr-x 1 ad ad 10583040 2010-02-18 12:25 Set-01.ppt
-rwxr-xr-x 1 ad ad       72 2010-02-18 12:25 zzz
ad@sydney:~/Set001/Samples$ ls -i bbbb myhardlink
691247 bbbb  691247 myhardlink
ad@sydney:~/Set001/Samples$ cp bbbb eeee
ad@sydney:~/Set001/Samples$ ls -li bbbb myhardlink eeee
691247 -rwxr-xr-x 2 ad ad 1200 2010-02-18 12:27 bbbb
691204 -rwxr-xr-x 1 ad ad 1200 2010-02-18 12:34 eeee
691247 -rwxr-xr-x 2 ad ad 1200 2010-02-18 12:27 myhardlink
ad@sydney:~/Set001/Samples$ diff bbbb myhardlink
ad@sydney:~/Set001/Samples$ rm bbbb
ad@sydney:~/Set001/Samples$ ls -l myhardlink
-rwxr-xr-x 1 ad ad 1200 2010-02-18 12:27 myhardlink
ad@sydney:~/Set001/Samples$ cp myhardlink bbbb
ad@sydney:~/Set001/Samples$ ls -l
total 10360
-rwxr-xr-x 1 ad ad       15 2010-02-18 12:25 aaa
-rwxr-xr-x 1 ad ad     1200 2010-02-18 12:35 bbbb
-rwxr-xr-x 1 ad ad        0 2010-02-18 12:25 delis
-rwxr-xr-x 1 ad ad     1200 2010-02-18 12:34 eeee
-rwxr-xr-x 1 ad ad      183 2010-02-18 12:25 lista
-rwxr-xr-x 1 ad ad     1200 2010-02-18 12:27 myhardlink
-rwxr-xr-x 1 ad ad 10583040 2010-02-18 12:25 Set-01.ppt
```

Soft links with ln

```
ad@sydney:~/Set001/Samples$ ls
aaa  bbbb  delis  eeee  lista  myhardlink  Set-01.ppt  zzz
ad@sydney:~/Set001/Samples$ ln -s bbbb mysoftlink
ad@sydney:~/Set001/Samples$ ls -l mysoftlink
lrwxrwxrwx 1 ad ad 4 2010-02-18 12:43 mysoftlink -> bbbb
ad@sydney:~/Set001/Samples$ file *
aaa:         ASCII text
bbbb:        ASCII text
delis:       empty
eeee:        ASCII text
lista:       ASCII text
myhardlink: ASCII text
mysoftlink: symbolic link to ‘bbbb’
Set-01.ppt: CDF V2 Document, corrupt: Can’t expand
    summary_info
zzz:         ASCII text
ad@sydney:~/Set001/Samples$ rm bbbb
ad@sydney:~/Set001/Samples$ ls
aaa  delis  eeee  lista  myhardlink  mysoftlink  Set-01.ppt
     zzz
ad@sydney:~/Set001/Samples$ more mysoftlink
mysoftlink: No such file or directory
ad@sydney:~/Set001/Samples$
```

Comparison between soft/hard links

Hard Links	Soft Links
"Pointer" to the initial file	Copy of the path to the initial file
Does not apply to directories	Applies to directories
Name change of the initial file does not create any problems	Name change in the initial file creates problems
Content changes in initial file are reflected in the link as well	Content changes in initial file are reflected in the link as well
File gets purged when *all* links are deleted	Deletion of initial file affects the link (point to non-existing file)

calls: *dup, dup2*

- *int dup(int oldfd);*
  uses the lowest-numbered unused descriptor for the new descriptor.

- *int dup2(int oldfd, int newfd);*
  makes *newfd* be the copy of *oldfd* - note:

  1. If *oldfd* is not a valid file descriptor, then the call fails, and *newfd* is not closed.
  2. If *oldfd* is a valid file descriptor, and *newfd* has the same value as *oldfd*, then *dup2()* does nothing, and returns *newfd*.

- After a successful return from one of these system calls, the old and new file descriptors may be used *interchangeably*.

Example of *dup* and *dup2*

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

int main(){
  int fd1, fd2, fd3;
  mode_t fdmode = S_IRUSR|S_IWUSR|S_IRGRP| S_IROTH;

  if ( ( fd1=open("dupdup2file", O_WRONLY | O_CREAT | O_TRUNC, fdmode ) ) == -1
       ){
    perror("open");
    exit(1);
  }
  printf("fd1 = %d\n", fd1);
  write(fd1, "What ", 5);
  fd2=dup(fd1);
  printf("fd2 = %d\n", fd2);
  write(fd2, "time", 4);
  close(0);

  fd3=dup(fd1);
  printf("fd3 = %d\n", fd3);
  write(fd3, " is it", 6);
  dup2(fd2, 2);
  write(2,"?\n",2);
  close(fd1); close(fd2); close(fd3);
  return 1;
}
```

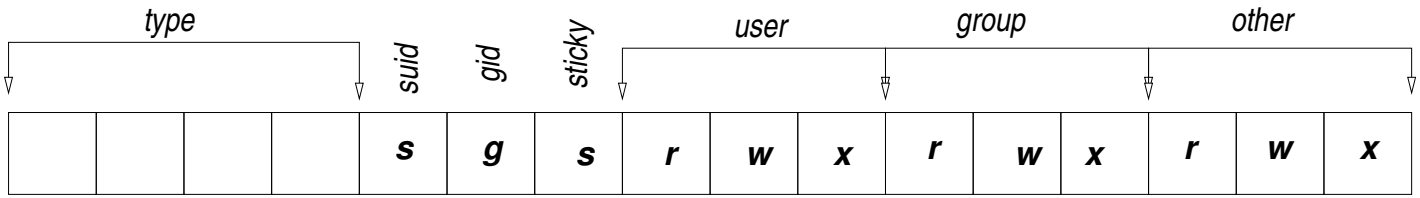
Execution Outcome:

```
ad@thales:~/Transparencies/Set004/src$ ls
anotherfile   count.c       dupdup2file    mytest
a.out         createfile.c  errors_demo.c  readwriteclose.c
buffeffect.c  dupdup2.c     filecontrol.c
ad@thales:~/Transparencies/Set004/src$ ./a.out
fd1 = 3
fd2 = 4
fd3 = 0
ad@thales:~/Transparencies/Set004/src$ ls
anotherfile   count.c       dupdup2file    mytest
a.out         createfile.c  errors_demo.c  readwriteclose.c
buffeffect.c  dupdup2.c     filecontrol.c
ad@thales:~/Transparencies/Set004/src$ cat dupdup2file
What time is it?
ad@thales:~/Transparencies/Set004/src$
```

Accessing *inode* information with *stat()*

- *int stat(char *path, struct stat *buf);*
  *int fstat(int fd, struct stat *buf);*
  returns information about a file; *path* points to the file (or *fd*) and the *buf* structure helps "carry" all derived information.
- such information includes:
  1. *buff→st_dev*: ID of device containing file
  2. *buff→st_ino*: inode number
  3. *buff→st_mode*: the last 9 bits represent the access rights of owner, group, and others. The first 4 bits indicate the type of the node (after a bitwise-AND with the constant S_IFMT, if the outcome is S_IFDIR, the node is a catalog, if outcome is S_IFREG, the mode is a regular file etc.)
  4. *buff→st_nlink*: number of hard links
  5. *buff→st_uid*: user-ID of owner
  6. *buff→st_gid*: group ID of owner
  7. *buff→st_size*: total size, in bytes
  8. *buff→st_atime*: time of last access
  9. *buff→st_mtime*: time of last modification of content
  10. *buff→st_ctime*: time of last status change

st_mode is a 16-bit quantity



1. 4 first bits indicate the type of the file (16 possible values - less than 10 file types are in use now: regular file, dir, block-special, char-special, fifo, symbolic link, socket).
2. the next three bits set the flags: *set-user-ID, set-group-ID* and the *sticky* bits respectively.
3. next three groups of 3 bits a piece indicate the read/write/execute access right for the the groups: *owner, group* and *others*.
4. masking can be used to decipher the permissions each file entiry is given.

stat-ing inodes

- The fields *st_atime*, *st_mtime* and *st_ctime* designate time as number of seconds past since 1/1/1970 of the Coordinated Universal Time (UTC).
- The function *ctime* helps bring the content of the fileds *st_atime*, *st_mtime* and *st_ctime* in a more readable format (that of the *date*). The call is: *char *ctime(time_t *timep);*
- *stat* returns 0 if successful; otherwise, -1
- Header files needed:
  <sys/stat.h> and *<sys/types.h>*

- *int fstat(int fd, struct stat *buf);* is identical to *stat* but it works with *file descriptors*.
- *int lstat(char *path, struct stat *buf);* is identical to *stat*, except that if path is a *symbolic link*, then the link itself is stat-ed, **not** the file that it refers to.

Definitions in <*sys/stat.h*>

```
#define     S_IFMT      0170000     /* type of file*/
#define     S_IFREG     0100000     /* regular */
#define     S_IFDIR     0040000     /* directory */
#define     S_IFBLK     0060000     /* block special */
#define     S_IFCHR     0020000     /* character sspecial */
#define     S_IFIFO     0010000     /* fifo */
#define     S_IFLNK     0120000     /* symbolic link */
#define     S_IFSOCK    0140000     /* socket */
```

Testing for a specific type of a file is easy using code fragments of the following style:

```
if ( (info.st_mode & S_IFMT) == S_IFIFO )
    printf("this is a fifo queue.\n");
```

Accessing information from inode

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

/* D. Savvas */
int main ( int argc , char * argv []) {
  struct stat statbuf ;
  struct tm lt;
  if ( stat ( argv [1] , &statbuf ) == -1) {
    perror ( " Failed to get file status " );
    exit (2) ;
  }
  else{
    char str_access[100];
    char str_modify[100];
    /* convert time_t to struct tm (access time)*/
    localtime_r(&statbuf.st_atime,&lt);
    /* use localtime_r instead of localtime*/
    /* create the string representation of lt */
    strftime(str_access, sizeof(str_access), "%c", &lt);

    /* do the same for modify time */
    localtime_r(&statbuf.st_mtime,&lt);
    strftime(str_modify, sizeof(str_modify), "%c", &lt);
```

Accessing information from inode

```
    printf("\n>> Correct Execution \n File: %s \n accessed : %s \n modified : %s
          " , argv [1], str_access, str_modify);

    /* This code won't work, because ctime uses a static variable
        where it stores the result */
    printf("\n\n>> Wrong Execution \n File: %s \n accessed : %s modified : %s" ,
          argv [1] , ctime(&statbuf.st_atime) , ctime(&statbuf.st_mtime)) ;
    /* It will compute the results of ctime(s) calls. and use the pointers to
         print */
    printf("\n-->> Pointer 1: %p, Pointer 2: %p\n",ctime(&statbuf.st_atime) ,
         ctime(&statbuf.st_mtime));
/* However, these two pointers are the same */

    /* This code works well, because we use two printf calls */
    printf("\n>> Correct Exeution but to be Avoided: \n");
    printf(" File: %s \n",argv[1]);
    printf(" accessed : %s",ctime(&statbuf.st_atime));
    printf(" modified : %s \n",ctime(&statbuf.st_mtime));
    /* However, POSIX.1-2008 marks asctime(),
                  asctime_r(), ctime(), and ctime_r() as obsolete,
                  recommending the use
                  of strftime(3) instead.*/
  }
  return (1) ;
}
```

Accessing information from inode

```
Running the program..
\begin{lstlisting}[basicstyle=\scriptsize\ttfamily,
    language=sh]
ad@thales:~/Transparencies/Set004/src$ ./a.out samplestat.c

>> Correct Execution
 File: samplestat.c
 accessed : Tue Feb 16 14:22:20 2016
 modified : Tue Feb 16 14:21:45 2016

>> Wrong Execution
 File: samplestat.c
 accessed : Tue Feb 16 14:22:20 2016
 modified : Tue Feb 16 14:22:20 2016

-->> Pointer 1: 0x7fbbdf96fda0, Pointer 2: 0x7fbbdf96fda0

>> Correct Exeution but to be Avoided:
 File: samplestat.c
 accessed : Tue Feb 16 14:22:20 2016
 modified : Tue Feb 16 14:21:45 2016

ad@thales:~/Transparencies/Set004/src$
```

Accessing Catalog Content

- The catalog content (ie, pairs of *inodes* and node names) can be accessed with the help of the calls: *opendir*, *readdir* and *closedir*.
- Accessing of a catalog happens via a pointer *DIR ** (similar to the *FILE ** pointer that is used by the *stdio*).
- Every item in the catalog is weaved around a structure called *struct dirent* that includes the following two elements:
  1. *d_ino*: inode number;
  2. *d_name[]*: a character string giving the filename (null terminated)
- Using these calls, it is not feasible to change the content of the directory or its structure.
- Required header files: <*sys/types.h*> and <*dirent.h*>

calls: *opendir, readdir, closedir*

- *DIR *opendir(char *name)*:
  1. Opens up the catalog termed *name* and returns a pointer type *DIR* for accessing the catalog.
  2. If there is a mistake, the call returns NULL

- *struct dirent *readdir(DIR *dirp);*
  1. the call returns a pointer to a *dirent* structure representing the next directory entry in the directory pointed to by *dirp*
  2. if for the current entry, the field *d_ino* is 0, the respective entry has been deleted.
  3. returns NULL if there are no more entries to be read.

- *int closedir(DIR *dirp);*
  1. closes the directory associated with *dirp*
  2. function returns 0 on success. On error, -1 is returned, and *errno* is set appropriately.

Example

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

void do_ls(char dirname[]){
DIR *dir_ptr;
struct  dirent *direntp;

if ( ( dir_ptr = opendir( dirname ) ) == NULL )
    fprintf(stderr, "cannot open %s \n",dirname);
else{
    while ( ( direntp=readdir(dir_ptr) ) != NULL )
        printf("%s\n", direntp->d_name) ;
     closedir(dir_ptr);
     }
}

int main(int argc, char *argv[]) {
if (argc == 1 ) do_ls(".");
else while ( --argc ){
        printf("%s: \n", *++argv ) ;
        do_ls(*argv);
    }
}
```

Execution Outcome

```
ad@thales:~/Transparencies/Set004/src$ ./a.out
.
count.c
dupdup2.c
mytest
a.out
createfile.c
samplestat.c
writeafterend.c
readwriteclose.c
filecontrol.c
openreadclosedir.c
buffeffect.c
mytest1
dupdup2file
errors_demo.c
anotherfile
..
ad@thales:~/Transparencies/Set004/src$
```

Creating a program that behaves as *ls -la*

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

char *modes[]={"---","--x","-w-","-wx","r--","r-x","rw-","rwx"};
              // eight distinct modes

void list(char *);
void printout(char *);

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

if (argc<2) { list("."); exit(0);}

while(--argc){
  if (stat(*++argv, &mybuf) < 0) {
    perror(*argv); continue;
    }

  if ((mybuf.st_mode & S_IFMT) == S_IFDIR )
        list(*argv);       // directory encountered
  else  printout(*argv);   // file encountered
  }
}
```

Creating a program that behaves as *ls -la*

```
void list(char *name){
DIR     *dp;
struct dirent *dir;
char    *newname;

    if ((dp=opendir(name))== NULL ) {
        perror("opendir"); return;
        }
    while ((dir = readdir(dp)) != NULL ) {
        if (dir->d_ino == 0 ) continue;
        newname=(char *)malloc(strlen(name)+strlen(dir->d_name)+2);
        strcpy(newname,name);
        strcat(newname,"/");
        strcat(newname,dir->d_name);
        printout(newname);
        free(newname); newname=NULL;
        }
    close(dp);
}
```

Creating a program that behaves as *ls -la*

```
void printout(char *name){
struct stat     mybuf;
char        type, perms[10];
int         i,j;

    stat(name, &mybuf);
    switch (mybuf.st_mode & S_IFMT){
    case S_IFREG: type = '-'; break;
    case S_IFDIR: type = 'd'; break;
    default:      type = '?'; break;
    }

    *perms='\0';

    for(i=2; i>=0; i--){
        j = (mybuf.st_mode >> (i*3)) & 07;
        strcat(perms,modes[j]);
        }

    printf("%c%s%3d %5d/%-5d %7d %.12s %s \n",\
        type, perms, mybuf.st_nlink, mybuf.st_uid, mybuf.st_gid, \
        mybuf.st_size, ctime(&mybuf.st_mtime)+4, name);
}
```

```
ad@thales:~/Transparencies/Set004/src$ ./a.out mydir
drwxr-xr-x 10  1000/1000     4096 Apr 11 01:05 mydir/.
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/e
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/g
-rw-r--r--  1  1000/1000      368 Apr 11 01:05 mydir/i
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/f
-rw-r--r--  1  1000/1000      750 Apr 11 01:05 mydir/j
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/b
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/h
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/c
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/d
drwx------  3  1000/1000     4096 Apr 11 01:04 mydir/..
-rw-r--r--  1  1000/1000       12 Apr 11 01:05 mydir/k
drwxr-xr-x  2  1000/1000     4096 Apr 10 01:24 mydir/a
ad@thales:~/Transparencies/Set004/src$
```

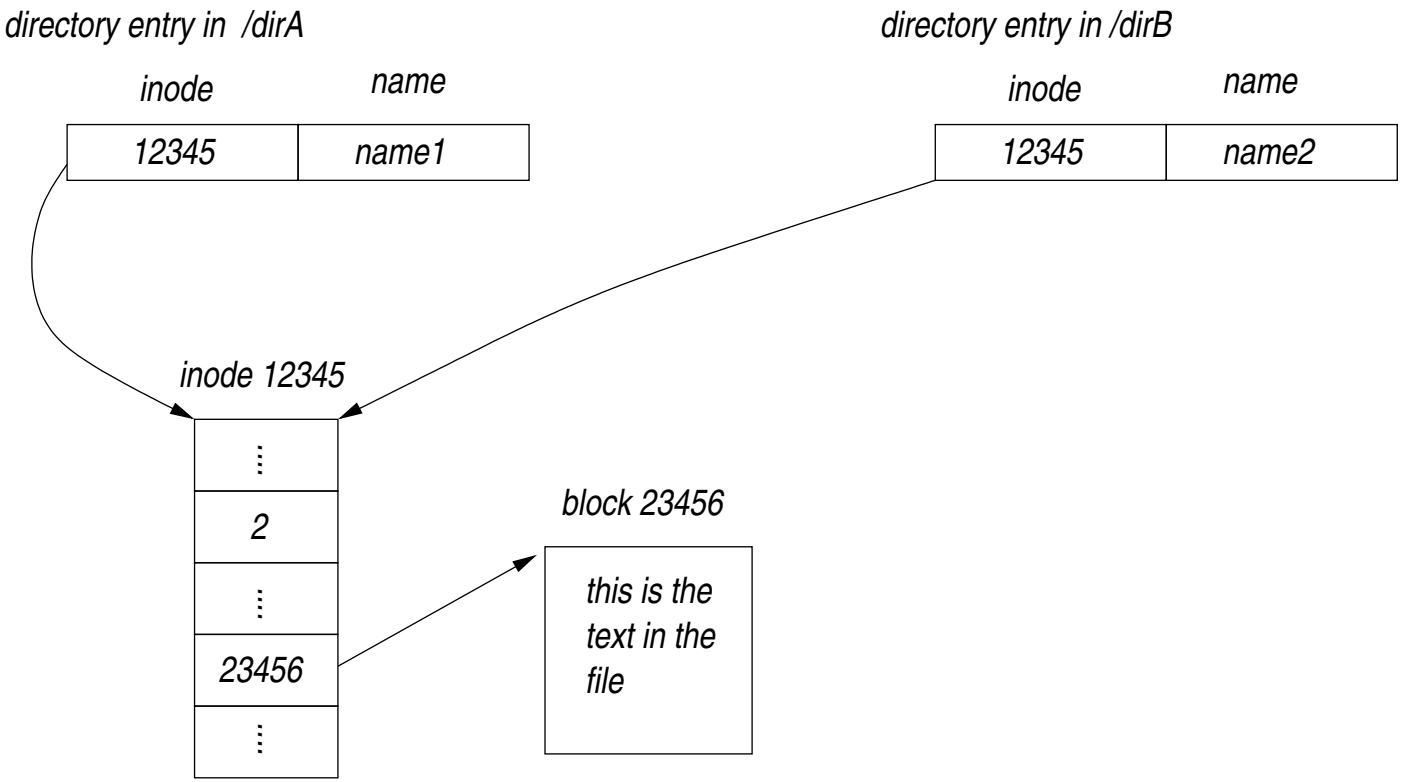
link and *unlink*

- *int unlink(char *pathname)*
- Deletes a name from the file system; if that name is the last link to a file and no other process have the file open, the file is deleted and its space is made available.

- *int link(char *oldpath, char *newpath)*
- It creates an new hard link to an existing file. if *newpath* exists, it will not be overwritten.
- The created link essentially connects the inode of the *oldpath* with the name of the *newpath*.

Example on *link*

```
#include <stdio.h>
#include <unistd.h>
....
if ( link("/dirA/name1","/dirB/name2")== -1 )
    prerror("Failed to make a new hard link in /dirB");
....
```



chmod, rename calls

- *int chmod(char *path, mode_t mode)*
  int fchmod(int fd, mode_t mode)
- Change the permissions (on files with *path* name or having an *fd* descriptor) according to what *mode* designates.
- On success, 0 is returned; otherwise -1

- *int rename(const char *oldpath, const char *newpath)*
- Renames a file, moving it between directories (indicated with the help of *oldpath* and *newpath*) if required.
- On success, 0 is returned; otherwise -1

symlink and *readlink* calls

- *int symlink(const char *oldpath, const char *newpath)*
- Creates a symbolic link named *newpath* that contains the string *oldpath*.
- A symbolic link (or soft link) may point to an existing file or to a nonexistent one; the latter is known as a **dangling link**.
- On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

- *ssize_t readlink (char *path, char *buf, size_t bufsiz)*
- Places the content of the symbolic link *path* in the buffer *buf* that has size *bufsiz*.
- On success, *readlink* returns the number of bytes placed in *buf*; otherwise, -1.