
A Short Introduction to Unix I/O Operations (might be helpful in Project 4)

January 2012

Low-Level Input/Output

- ▶ The *stdio* library enables the average user carry out I/Os without worrying about buffering and/or data conversion.
- ▶ The *stdio* is user-friendly set of system calls.
- ▶ Low-level I/O functionality is required
 1. when the amenities that *stdio* are not desirable (for whatever reason) in accessing files/devices or
 2. when interprocess communication occurs with the help of pipes/sockets.

Low-Level I/Os

- ▶ In low-level I/O, file descriptors that identify files, pipes, sockets and devices are small integers
 - ▶ The above is in contrast to what happens in the *stdio* where respective identifiers are pointers.
- ▶ Designated (fixed) file descriptors:
 - 0 : standard input
 - 1 : standard output
 - 2 : standard error (for error diagnostics).
- ▶ The above file descriptors 0, 1, and 2 correspond to pointers to the *stdin*, *stdout* and *stderr* files of the *stdio* library.
- ▶ The file descriptors are parent-“inherited” to any child process that the parent in question creates.

The `open()` system call

- ▶ `int open(char *pathname, int flags [, mode_t mode])`
- ▶ The call opens or creates a file with absolute or relative `pathname` for reading/writing.
- ▶ `flags` designates the way (a number) with which the file can be accessed; values for `flags` may be constructed by a bitwise-inclusive OR of flags from the following set:
 - ▶ `O_RDONLY`: opening for reading only.
 - ▶ `O_WRONLY`: opening for writing only.
 - ▶ `O_RDWR`: opening for both reading and writing.
 - ▶ `O_APPEND`: writing at the end of the file.
 - ▶ `O_CREAT`: create a file if it does not exist already.
 - ▶ `O_TRUNC`: the size of the file will be truncated to 0 if the file exists.

The `open()` system call

- ▶ required: `#include <fcntl.h>`
⇒ `fcntl.h` defines all these (and more) flags.
- ▶ The not-compulsory `mode` parameter is an integer that designates the desired access primitives during the creation of a file (access rights not allowed from the `umask` are not allowed).
- ▶ `open` returns an integer that designates the file created and in case of no success, it returns `-1`.

createfile.c

```
#include <stdio.h> // to have access to printf()
#include <stdlib.h> // to enable exit calls
#include <fcntl.h> // to have access to flags def
#define PERMS 0644 // set access permissions

char *workfile="mytest";

main(){
    int filedes;

    if ((filedes=open(workfile,O_CREAT|O_RDWR,PERMS))===-1){
        perror("creating");
        exit(1);
    }
    else {
        printf("Managed to get to the file successfully\n");
    }
    exit(0);
}
```

Running the executable for *createfile.c*

```
ad@thales:~/Transparencies/Set004/src$ gcc createfile.c
ad@thales:~/Transparencies/Set004/src$ ./a.out
Managed to get to the file successfully
ad@thales:~/Transparencies/Set004/src$ ls -l
total 20
-rwxr-xr-x 1 ad ad 8442 2010-04-06 21:50 a.out
-rw-r--r-- 1 ad ad 375 2010-04-06 21:49 createfile.c
-rw-r--r-- 1 ad ad 506 2010-04-06 16:24 errors_demo.c
-rw-r--r-- 1 ad ad 0 2010-04-06 21:50 mytest
ad@thales:~/Transparencies/Set004/src$ cat > mytest
This is Kon Tsakalozos
ad@thales:~/Transparencies/Set004/src$ ./a.out
Managed to get to the file successfully
ad@thales:~/Transparencies/Set004/src$ ls
a.out createfile.c errors_demo.c mytest
ad@thales:~/Transparencies/Set004/src$ more mytest
This is Kon Tsakalozos
ad@thales:~/Transparencies/Set004/src$
```

Setting *modes* with symbolic names

| | | |
|---------|-------|--|
| S_IRWXU | 00700 | owner has read, write and execute permission |
| S_IRUSR | 00400 | owner has read permission |
| S_IWUSR | 00200 | owner has write permission |
| S_IXUSR | 00100 | owner has execute permission |
| S_IRWXG | 00070 | group has read, write and execute permission |
| S_IRGRP | 00040 | group has read permission |
| S_IWGRP | 00020 | group has write permission |
| S_IXGRP | 00010 | group has execute permission |
| S_IRWXO | 00007 | others have read, write and execute permission |
| S_IROTH | 00004 | others have read permission |
| S_IWOTH | 00002 | others have write permission |
| S_IXOTH | 00001 | others have execute permission |

Working with access modes

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *filename = "/tmp/file";
...
fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
...
```

1. If the call to *open()* is successful, the file is opened for reading/writing by the user.
2. Those in the “group” and “others” can read the file.

The *creat()* call

- ▶ The *creat* is an alternative way to create a file (instead of using *open()*).
- ▶ *int creat(char *pathname, mode_t mode);*
- ▶ *pathname* is any Unix pathname giving the target location in which the file is to be created.
- ▶ *mode* helps set up the access rights.
- ▶ *creat* will always truncate (an existing file before returning its file descriptor).

```
filedes = create("/tmp/tsak", 0644);
```

is equivalent to:

```
filedes = open("/tmp/tsak", O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

The `read()` call

- ▶ `ssize_t read(int filedes, char *buffer, size_t n)`
- ▶ Reads at most n bytes from a file, device, end-point of a pipe, socket that is designated by `filedes` and place the bytes on `buffer`.
- ▶ The call returns the number of bytes successfully read, 0 if we are past the last byte-already read, and -1 if a problem occurs.
- When do we read less bytes?
 1. The file has less characters left to be read.
 2. The operation is “interrupted” by a signal.
 3. Reading on pipe/socket takes place and a character becomes available (in which case a while-loop is needed to read all characters).

Using the *read()* call (count.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFSIZE 27

main(){
    char buffer[BUFSIZE]; int  filedes; ssize_t nread; long total=0;

    if ((filedes=open("anotherfile", O_RDONLY))== -1){
        printf("error in opening anotherfile \n");
        exit(1);
    }

    while ( (nread=read(filedes ,buffer ,BUFSIZE)) > 0 )
        total += nread;
    printf("Total char in anotherfile %ld \n",total);
    exit(0);
}
```

Running the executable:

```
ad@thales:~/Transparencies/Set004/src$ ./a.out
Total char in anotherfile 936
ad@thales:~/Transparencies/Set004/src$
```

- What happens if *char *buffer=NULL;* is used instead of *char buffer[BUFSIZE];* ??

The `write()` and `close()` system calls

- ▶ `ssize_t write(int fildes, char *buffer, size_t n);`
- ▶ The call writes at most `ls n` bytes of content from the `buffer` to the file that is described by `fildes`.
- ▶ `write` returns the number of bytes successfully written out to the file or `-1` in case of failure.
- ▶ use the `write` call with: `#include <unistd.h>`

- ▶ `int close(int fildes);`
- ▶ releases the file descriptor `fildes`; returns `0` in case of successful release and `-1` otherwise.
- ▶ use the `close` call with: `#include <unistd.h>`

Working with *open read, write* and *close* calls

Write a program that appends the content of a file at the very end of the content of another file.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#define BUFFSIZE 1024

int main(int argc, char *argv[]){
    int n, from, to; char buf[BUFFSIZE];
    mode_t fdmode = S_IRUSR|S_IWUSR|S_IRGRP| S_IROTH;

    if (argc!=3) {
        write(2,"Usage: ", 7); write(2, argv[0], strlen(argv[0]));
        write(2," from-file to-file\n", 19); exit(1); }

    if ( ( from=open(argv[1], O_RDONLY)) < 0 ){
        perror("open"); exit(1); }

    if ( (to=open(argv[2], O_WRONLY|O_CREAT|O_APPEND , fdmode)) < 0 ){
        perror("open"); exit(1); }

    while ( (n=read(from, buf, sizeof(buf))) > 0 )
        write(to,buf,n);
    close(from); close(to); return(1);
}
```

Execution Outcome:

```
ad@thales:~/Transparencies/Set004/src$ ls
anotherfile  count.c      dupdup2file  mytest
              writeafterend.c
a.out        createfile.c errors_demo.c mytest1
buffeffect.c dupdup2.c    filecontrol.c readwriteclose.c
ad@thales:~/Transparencies/Set004/src$ more mytest
This is Konstantinos Tsakalozos
ad@thales:~/Transparencies/Set004/src$ more mytest1
that I use to show something silly
use to show something silly
to show something silly
ad@thales:~/Transparencies/Set004/src$ ./a.out
Usage: ./a.out from-file to-file
ad@thales:~/Transparencies/Set004/src$ ./a.out mytest
mytest1
ad@thales:~/Transparencies/Set004/src$ cat mytest1
that I use to show something silly
use to show something silly
to show something silly
This is Konstantinos Tsakalozos
ad@thales:~/Transparencies/Set004/src$
```

Using *open read, write* and *close* calls

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

int main(){
    int fd, bytes, bytes1, bytes2;
    char buf[50];

    mode_t fdmode = S_IRUSR|S_IWUSR;

    if ( ( fd=open("t", O_WRONLY | O_CREAT, fdmode ) ) == -1 ){
        perror("open");
        exit(1);
    }

    bytes1 = write(fd, "First write. ", 13);
    printf("%d bytes were written. \n", bytes1);
    close(fd);

    if ( (fd=open("t", O_WRONLY | O_APPEND)) == -1 ){
        perror("open");
        exit(1);
    }

    bytes2 = write(fd, "Second Write. \n", 14);
    printf("%d bytes were written. \n", bytes2);
    close(fd);
}
```

```
if ( (fd=open("t", O_RDONLY)) == -1 ){
    perror("open");
    exit(1);
}

bytes=read(fd, buf, bytes1+bytes2);
printf("%d bytes were read \n",bytes);
close(fd);

buf[bytes]='\0';
printf("%s\n",buf);
return(1);
}
```

Running the program..

```
ad@thales:~/Transparencies/Set004/src$ ls
anotherfile  count.c          errors_demo.c  readwriteclose.c
a.out       createfile.c    mytest
ad@thales:~/Transparencies/Set004/src$ ./a.out
13 bytes were written.
14 bytes were written.
27 bytes were read
First write. Second Write.
ad@thales:~/Transparencies/Set004/src$ ls
anotherfile  count.c          errors_demo.c  readwriteclose.c
a.out       createfile.c    mytest         t
ad@thales:~/Transparencies/Set004/src$
```

Copying a file with variable buffer size

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

#define SIZE          30
#define PERM          0644

int mycopyfile(char *name1, char *name2, int BUFFSIZE){
    int infile, outfile;
    ssize_t nread;
    char buffer[BUFFSIZE];

    if ( (infile=open(name1,O_RDONLY)) == -1 )
        return(-1);

    if ( (outfile=open(name2, O_WRONLY|O_CREAT|O_TRUNC, PERM)) == -1){
        close(infile);
        return(-2);
    }

    while ( (nread=read(infile, buffer, BUFFSIZE) ) > 0 ){
        if ( write(outfile,buffer,nread) < nread ){
            close(infile); close(outfile); return(-3);
        }
    }
    close(infile); close(outfile);
}
```

Copying a file with variable buffer size

```
        if (nread == -1 ) return(-4);
        else      return(0);
}

int main(int argc, char *argv []){
    int      status=0;

    status=mycopyfile (argv [1], argv [2], atoi (argv [3]));
    exit (status);
}
```

Running the program for various size buffers..

```
ad@thales :~/Transparencies/Set004/src$ time ./a.out /tmp/stuff.ppt /tmp/alex1
8192
real    0m0.012s user    0m0.000s sys     0m0.012s
ad@thales :~/Transparencies/Set004/src$ time ./a.out /tmp/stuff.ppt /tmp/alex1
4096
real    0m0.010s user    0m0.000s sys     0m0.008s
ad@thales :~/Transparencies/Set004/src$ time ./a.out /tmp/stuff.ppt /tmp/alex1
256
real    0m0.071s user    0m0.000s sys     0m0.072s
ad@thales :~/Transparencies/Set004/src$ time ./a.out /tmp/stuff.ppt /tmp/alex1 32
real    0m0.454s user    0m0.012s sys     0m0.444s
ad@thales :~/Transparencies/Set004/src$ time ./a.out /tmp/stuff.ppt /tmp/alex1 1
real    0m13.738s user    0m0.428s sys     0m13.305s
ad@thales :~/Transparencies/Set004/src$
```

Accessing inode information with *stat()*

- ▶ *int stat(char *path, struct stat *buf);*
returns information about a file; *path* points to the file and the *buf* structure helps “carry” all derived information.
- ▶ such information includes:
 1. *buf*→*st_dev*: ID of device containing file
 2. *buf*→*st_ino*: inode number
 3. *buf*→*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. *buf*→*st_nlink*: number of hard links
 5. *buf*→*st_uid*: user-ID of owner
 6. *buf*→*st_gid*: group ID of owner
 7. *buf*→*st_size*: total size, in bytes
 8. *buf*→*st_atime*: time of last access
 9. *buf*→*st_mtime*: time of last modification of content
 10. *buf*→*st_ctime*: time of last status change

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 fields *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.

Accessing information from inode

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

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

    if (stat(argv[1], &statbuf) == -1){
        perror("Failed to get file status");
        exit(2);
    }
    else
        printf("%s accessed: %s modified: %s",argv[1],
            ctime(&statbuf.st_atime), ctime(&statbuf.st_mtime));

    return(1);
}
```

Running the program..

```
ad@thales:~/Transparencies/Set004/src$ ./a.out samplestat.c
samplestat.c accessed: Sat Apr 10 00:04:08 2010
modified: Sat Apr 10 00:04:08 2010
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$
```

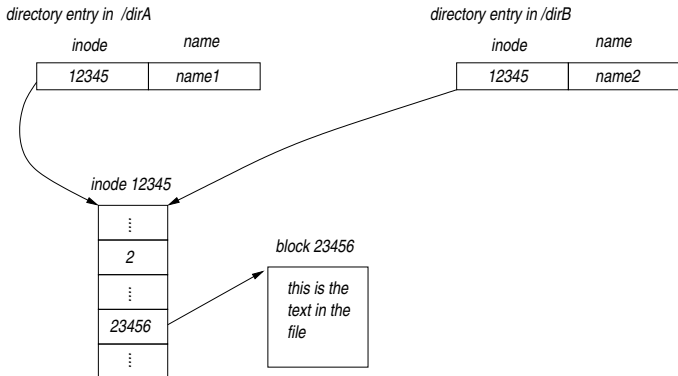
link and *unlink*

- ▶ *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 )
    perror("Failed to make a new hard link in /dirB");
....
```



readlink

- ▶ *ssize_t readlink(const char *path, char *buf, size_t bufsiz)*
- ▶ `readlink()` places the contents of the symbolic link path in the buffer `buf`, which has size `bufsiz`. `readlink()` does not append a null byte to `buf`. It will truncate the contents (to a length of `bufsiz` characters), in case the buffer is too small to hold all of the contents.
- ▶ On success, `readlink()` returns the number of bytes placed in `buf`. On error, `-1` is returned and `errno` is set to indicate the error.

readlink

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

#define BUFFERSIZE 128

int main(void)
{
    char link_val[BUFFERSIZE];
    if (readlink("./symlink", link_val, BUFFERSIZE) ==
        -1)
        perror("Failed to read value of symbolic
                link of file ./symlink");
    else
        printf("Value of symbolic link of file ./
                symlink : %s\n", link_val);

    return 0;
}
```

Execution Outcome

```
$ ln -s file1 symlink
$ ls
file1  symlink
$ ./readlink
Value of symbolic link of file ./symlink : file1

$ mv symlink symlink1
$ ./readlink
Failed to read value of symbolic link of file ./symlink: No
such file or directory
```