BSM-308 System Programming

Sakarya University-Computer Engineering

Contents

a) Stat and Opendir / Readdir / Closedir

b) Prsize : recursive directory traversal

c) umask and time

- http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Stat/lecture.html
- http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Prsize/lecture.html
- https://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Umask-And-Others/

Stat

- Stat is a system call you can use to get information about files - information contained in the files' inodes .
- I'll go over a simple motivational example here.
- Let's say you don't have a stat system call, and you want to write a program that for each argument that is a file, lists the size and name of the file.
- Something like this (src /ls1.c) will work:

```c
/* This program lists each program on its command line together with
   its size.  It does this by opening each file, lseeking to the
   end, and printing the value of the file pointer. */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int i;
  int fd;
  off_t size;

  for (i = 1; i < argc; i++) {
    fd = open(argv[i], O_RDONLY);
    if (fd < 0) {
      printf("Couldn't open %s\n", argv[i]);
    } else {
      size = lseek(fd, (off_t) 0, SEEK_END);
      printf("%10lld %s\n", size, argv[i]);
      close(fd);
    }
  }
  return 0;
}
```

Stat

- What is done is to try to open each file and then search all the way to the end of the file to figure out its size.

```
unalc@unalc:~/Desktop/cs360/Stat$ ls -l txt/input*.txt
-rw-rw-r-- 1 unalc unalc 185 May   5  2022 txt/input1.txt
-rw-rw-r-- 1 unalc unalc 179 May   5  2022 txt/input2.txt
unalc@unalc:~/Desktop/cs360/Stat$ bin/ls1 txt/input*.txt
       185 txt/input1.txt
       179 txt/input2.txt
```

- There's a problem here though:

```
unalc@unalc:~/Desktop/cs360/Stat$ rm -rf txt/myfile.txt
unalc@unalc:~/Desktop/cs360/Stat$ make txt/myfile.txt
echo "Hi" > txt/myfile.txt
chmod 0 txt/myfile.txt
unalc@unalc:~/Desktop/cs360/Stat$ ls -l txt/myfile.txt
---------- 1 unalc unalc 3 Ara  12 00:54 txt/myfile.txt
unalc@unalc:~/Desktop/cs360/Stat$ bin/ls1 txt/myfile.txt
Couldn't open txt/myfile.txt
```

Stat

- ls1 could not open the file "txt/myfile.txt" and print its size.
- This is unfortunate, but it also points to why we need the "stat" function -- there are things about a file that would be nice to know even if we weren't allowed to access the file itself.
- To reiterate, the stat system call gives you information about the inode of a file .
- The user can do this as long as they have permission to access the directory containing the file.
- Read the man page for statistics .

Stat

- The stat structure is defined in /usr/include/sys/stat.h and is roughly like this :

```
struct stat {
           mode_t    st_mode;     /* File mode (see mknod(2)) */
           ino_t     st_ino;      /* Inode number */
           dev_t     st_dev;      /* ID of device containing */
                                  /* a directory entry for this file */
           dev_t     st_rdev;     /* ID of device */
                                  /* This entry is defined only for */
                                  /* char special or block special files */
           nlink_t   st_nlink;    /* Number of links */
           uid_t     st_uid;      /* User ID of the file's owner */
           gid_t     st_gid;      /* Group ID of the file's group */
           off_t     st_size;     /* File size in bytes */
           time_t    st_atime;    /* Time of last access */
           time_t    st_mtime;    /* Time of last data modification */
           time_t    st_ctime;    /* Time of last file status change */
                                  /* Times measured in seconds since */
           long      st_blksize;  /* Preferred I/O block size */
           long      st_blocks;   /* Number of 512 byte blocks allocated*/
};
```

Stat

- Confusing types are mostly ints , longs and shorts . i.e. from /usr/include/sys/types.h :

```
typedef unsigned long   ino_t;
typedef short           dev_t;
typedef long            off_t;
typedef unsigned short  uid_t;
typedef unsigned short  gid_t;
```

- And from /usr/include/sys/stdtypes.h :

```
typedef unsigned short  mode_t;         /* file mode bits */
typedef short           nlink_t;        /* links to a file */
typedef long            time_t;         /* value = secs since epoch */
```

Stat

- After reading this man page, it should be simple to modify ls1.c to work properly using stat instead of open / lseek .
- In this src /ls2.c:

```
UNIX> bin/ls2 txt/*
       185 txt/input1.txt
       179 txt/input2.txt
         3 txt/myfile.txt
UNIX>
```

```c
/* This is a program which lists files and their sizes to standard output.
   The files are specified on the command line arguments.  It uses stat to
   see if the files exist, and to determine the files' sizes.  */

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

int main(int argc, char **argv)
{
  int i;
  struct stat buf;
  int exists;

  for (i = 1; i < argc; i++) {
    exists = stat(argv[i], &buf);
    if (exists < 0) {
      fprintf(stderr, "%s not found\n", argv[i]);
    } else {
      printf("%10lld %s\n", buf.st_size, argv[i]);
    }
  }
  return 0;
}
```

Stat()-Times

- As history, st_atime , st_mtime , and st_ctime record file access, modification, and creation times in seconds.
- As operating systems progressed, these were replaced with higher precision times.
- Fortunately, there are macros defined in stat.h to give you access to second-precision times, usually with st_atime , st_mtime and st_ctime .
- This is nice because if you only care about second-precision, your code will remain portable.

Stat-Times

- For example, the src / mtime.c program prints the modification times of files given on the command line in seconds. It uses st_mtime, a macro defined in src / stat.h on most machines :

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

int main(int argc, char **argv)
{
  int i;
  struct stat buf;

  for (i = 1; i < argc; i++) {
    if (stat(argv[i], &buf) != 0) {
      printf("Couldn't stat %s\n", argv[i]);
    } else {
      printf("%s %ld\n", argv[i], buf.st_mtime);
    }
  }
  return 0;
}
```

```
UNIX> echo "Fred" > fred.txt    # Create fred.txt
UNIX> bin/mtime fred.txt        # Show its modification time.
fred.txt 1645028284
UNIX> touch fred.txt            # I waited 5 seconds before doing this
UNIX> bin/mtime fred.txt        # This is reflected in its modification time.
fred.txt 1645028289
UNIX> rm fred.txt
UNIX>
```

Stat

- Next, we want our " ls " to work like a real " ls " -- accepting no arguments and listing all files in the current directory.
- For this we need to use the " opendir / readdir / writedir " calls.
- Note that these are C library calls and not system calls.
- This means they tell you to open/close/read/write and interpret the format of the directory files.
- " struct The structure " dir " is defined in /usr/include/sys/dir.h :

```
struct  dirent {
        off_t           d_off;          /* offset of next disk dir entry */
        unsigned long   d_fileno;       /* file number of entry */
        unsigned short  d_reclen;       /* length of this record */
        char            *d_name;        /* name */
};
```

Stat

- src /ls3.c sets ls2.c to read from the current directory (".") and print all files and their sizes:

```
UNIX> ( cd txt ; ../bin/ls3 )
. 192
.. 320
input1.txt 185
input2.txt 179
.keep 0
myfile.txt 3
UNIX>
```

```
int main(int argc, char **argv)
{
  struct stat buf;
  int exists;
  DIR *d;
  struct dirent *de;

  d = opendir(".");
  if (d == NULL) {
    fprintf(stderr, "Couldn't open \".\"\n");
    exit(1);
  }

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    exists = stat(de->d_name, &buf);
    if (exists < 0) {
      fprintf(stderr, "%s not found\n", de->d_name);
    } else {
      printf("%s %lld\n", de->d_name, buf.st_size);
    }
  }
  closedir(d);
  return 0;
}
```

Stat

- Now, when you run ls3 notice two things –
- First, the output is unformatted.
- Second, the files are not sorted.
- This is because readdir () makes no guarantees about the ordering of files in a directory.
- The next two programs solve each of these problems.
- First, formatting the output. What we want to see looks like this:

```
.                192
..               352
input1.txt       185
input2.txt       179
.keep              0
myfile.txt         3
```

Stat

- To do this, we need to know how long the longest file name is.
- We need to know this before printing any filename.
- So, what we do is read all directory entries into a linked list and calculate the maximum length along the way.
- After doing this we loop through the list and print the output in a nice format.
- the printf () statement and read the man page on printf () to understand why it works .

Stat

This is src /ls4.c:

```
int main(int argc, char **argv)
{
  struct stat buf;
  int exists;
  DIR *d;
  struct dirent *de;
  Dllist files, tmp;
  int maxlen;

  d = opendir(".");
  if (d == NULL) {
    fprintf(stderr, "Couldn't open \".\"\n");
    exit(1);
  }

  maxlen = 0;
  files = new_dllist();
```

Stat

```
  for (de = readdir(d); de != NULL; de = readdir(d)) {   /* List all fo the files and store in a linked list. */
    dll_append(files, new_jval_s(strdup(de->d_name)));
    if (strlen(de->d_name) > maxlen) maxlen = strlen(de->d_name);   /* Maintain the size of the longers filename. */
  }
  closedir(d);

  dll_traverse(tmp, files) {          /* Now traverse the list and call stat() on each file to determine its size. */
    exists = stat(tmp->val.s, &buf);
    if (exists < 0) {
      fprintf(stderr, "%s not found\n", tmp->val.s);
    } else {
      printf("%*s %10lld\n", -maxlen, tmp->val.s, buf.st_size);
    }
  }
  return 0;
}
```

Stat

- Why did we use strdup instead of de-> d_name in the dll_append () call ?
- The man page doesn't tell you anything about how the structure returned by readdir () is allocated.
- the value returned by readdir() is normal until you next call readdir () or closedir ().
- If readdir () returns " struct If we knew that we had malloced space for resist " and that the space was not empty until the user called free(), we could easily put de-> d_name into our dlist ,
- However, since there is no such assurance from the man page, we need to call strdup ().

Stat

- For example, opendir / readdir / closedir can be implemented like this:

1. opendir () opens the directory file and creates a " struct resist " does mallocs .
2. readdir () this " struct reads the next directory entry in resist " and returns a pointer to it.,
3. closedir () closes the directory file and " struct resist " is released.

- why such an implementation requires us to call strdup () in the dll_append () statement .
- de-> d_name here then we will have all kinds of memory related problems.
- This is a subtle but important point.

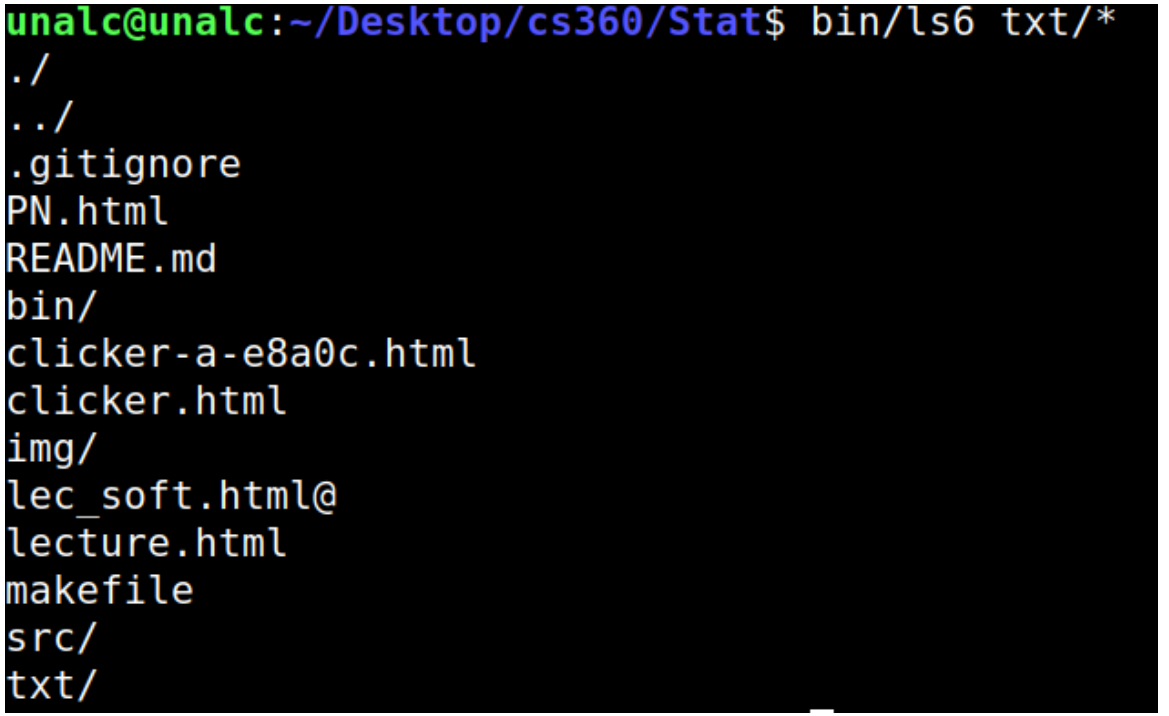
Stat

- Now, to print the sorted index files, red-black the entries instead of a dlllist . Just add it to trees . The code is in src /ls5.c . This is a very simple change.
- Next, we want to get rid of the %10d in the printf statement.
- So we want there to be a space between the last column of file names and the first column of file sizes.
- this by finding the maximum size of the file while traversing the directory and using this in the printf statement. This takes a sprintf () and a strlen () -- see src /ls5a.c.

```
File  Edit  View  Terminal  Tabs  Help
unalc@unalc:~/Desktop/cs360/Stat$ bin/ls5 txt/*
.                          4096
..                         4096
.gitignore                   51
PN.html                     275
README.md                   344
bin                        4096
clicker-a-e8a0c.html       3100
clicker.html               1963
img                        4096
lec_soft.html             17393
lecture.html              17393
makefile                   1042
src                        4096
txt                        4096
unalc@unalc:~/Desktop/cs360/Stat$ bin/ls5a txt/*
.                      4096
..                     4096
.gitignore               51
PN.html                 275
README.md               344
bin                    4096
clicker-a-e8a0c.html   3100
clicker.html           1963
img                    4096
lec_soft.html         17393
lecture.html          17393
makefile               1042
src                    4096
txt                    4096
```

Stat

- Finally, src /ls6.c performs the same function as " ls -F ".
- That is, it prints directories with a trailing "/", symbolic (soft) links with a "@", and executables with a "*".
- You can call this " struct We can do this by commenting the " st_mode " field of the buf .
- Review the code, because it will be very useful when writing jtar



Measuring directory size using stat system call

Recursive directory traversal

This lecture covers the writing of a command **prsize**

What **prsize** does is return the number of bytes taken up by all files reachable from the current directory (excluding soft links).

Prsize illustrates using **opendir/readdir/closedir**, **stat**, recursion, building path names, and finding hard links.

prsize1

Open working (current) directory

Get inode information of selected file

Get file size from the inode infomation.

File Edit View Search Tools Documents Help

Open Save Undo

prsize1.c

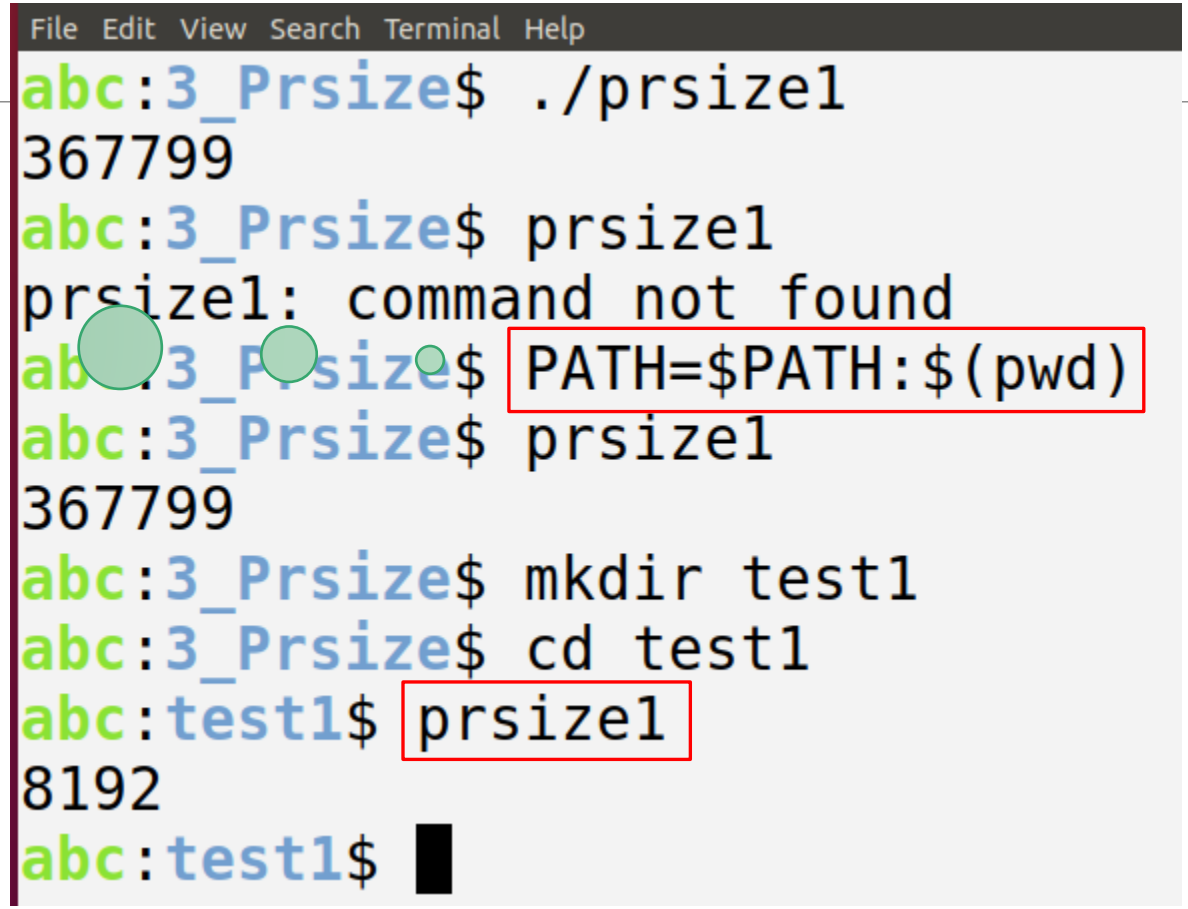
```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
main()
{
  DIR *d;
  struct dirent *de;
  struct stat buf;
  int exists;
  int total_size;

  d = opendir(".");
  if (d == NULL) {
    perror("prsize");
    exit(1);
  }
  total_size = 0;
  for (de = readdir(d); de != NULL; de = readdir(d)) {
    exists = stat(de->d_name, &buf);
    if (exists < 0) {
      fprintf(stderr, "Couldn't stat %s\n", de->d_name);
    } else {
      total_size += buf.st_size;
    }
  }
  closedir(d);
  printf("%d\n", total_size);
}
```

C Tab Width: 8 Ln 22, Col 4 INS

prsize1

We temporarily added current directory to PATH so that we can able to use ***prsize*** programs in other directories.



prsize2 defined function

The program that lists the file sizes has been defined as a function. It will help us for recursive directory traversal.

File Edit View Search Tools Documents Help

Open Save Undo

prsize2.c ×

```c
int get_size(char *fn)
{
  DIR *d;
  struct dirent *de;
  struct stat buf;
  int exists;
  int total_size;

  d = opendir(fn);
  if (d == NULL) {
    perror("prsize");
    exit(1);
  }

  total_size = 0;

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    exists = stat(de->d_name, &buf);
    if (exists < 0) {
      fprintf(stderr, "Couldn't stat %s\n", de->d_name);
    } else {
      total_size += buf.st_size;
    }
  }
  closedir(d);
  return total_size;
}

main()
{
  printf("%d\n", get_size("."));
}
```

C Tab Width: 8 Ln 43, Col 2 INS

prsize3 : recursive implementation

- Whenever we encounter a directory, we want to find out the size of everything in that directory, so we will call get_size() recursively on that directory.
- S_ISDIR() checks if a file directory or not.
- If it is a directory, all the files it contains should be measured.
- But this results in the error you see below.

```
File Edit View Search Terminal Help
test:prsize3
prsize: Too many open files
test:
```

File Edit View Search Tools Documents Help

Open Save Undo

prsize3.c ×

```c
int get_size(char *fn)
{
  DIR *d;
  struct dirent *de;
  struct stat buf;
  int exists;
  int total_size;

  d = opendir(fn);
  if (d == NULL) {
    perror("prsize");
    exit(1);
  }

  total_size = 0;

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    exists = stat(de->d_name, &buf);
    if (exists < 0) {
      fprintf(stderr, "Couldn't stat %s\n", de->d_name);
    } else {
      total_size += buf.st_size;
    }
    /* Make the recursive call if the file is a directory */
    if (S_ISDIR(buf.st_mode)) {
      total_size += get_size(de->d_name);
    }
  }
  closedir(d);
  return total_size;
}

main()
{
  printf("%d\n", get_size("."));
}
```

C Tab Width: 8 Ln 22, Col 22 INS

prsize3a: Analysis of Error

- I put a print statement into prsize3a.c to see when it's making the recursive calls:
- When the program is run (./prsize3a) repeatedly calls the "." (current) directory.
- I other words it calls itself infinitely.
- This goes into an infinite loop until you run out of open file descriptors at which point opendir() fails.

File Edit View Search Tools Documents Help

Open Save Undo

prsize3a.c ×

```
total_size = 0;

for (de = readdir(d); de != NULL; de = readdir(d)) {
  exists = stat(de->d_name, &buf);
  if (exists < 0) {
    fprintf(stderr, "Couldn't stat %s\n", de->d_name);
  } else {
    total_size += buf.st_size;
  }
  /* Make the recursive call if the file is a directory */
  if (S_ISDIR(buf.st_mode)) {
    printf("Making recursive call on directory %s\n", de->d_name);
    total_size += get_size(de->d_name);
  }
}
closedir(d);
return total_size;


main()
```

C ▾ Tab Width: 8 ▾ Ln 38, Col 42 INS

File Edit View Search Terminal Help

```
Making recursive call on directory .
Making recursive call on directory .
Making recursive call on directory .
Making recursive call on directory .
Making recursive call on directory .
Making recursive call on directory .
prsize: Too many open files
```

prsize4 : excluding “.” and “..”directories

If we exclude “.” and “..” inside recursive loop program doen’t go into infinite loop.

File Edit View Search Tools Documents Help

Open Save Undo

prsize4.c ×

```
for (de = readdir(d); de != NULL; de = readdir(d)) {
  exists = stat(de->d_name, &buf);
  if (exists < 0) {
    fprintf(stderr, "Couldn't stat %s\n", de->d_name);
  } else {
    total_size += buf.st_size;
  }
  /* Make the recursive call if the file is a directory and is not
     . or .. */
  if (S_ISDIR(buf.st_mode) && strcmp(de->d_name, ".") != 0 &&
      strcmp(de->d_name, "..") != 0) {
    total_size += get_size(de->d_name);
  }
}
closedir(d);
return total_size;
```

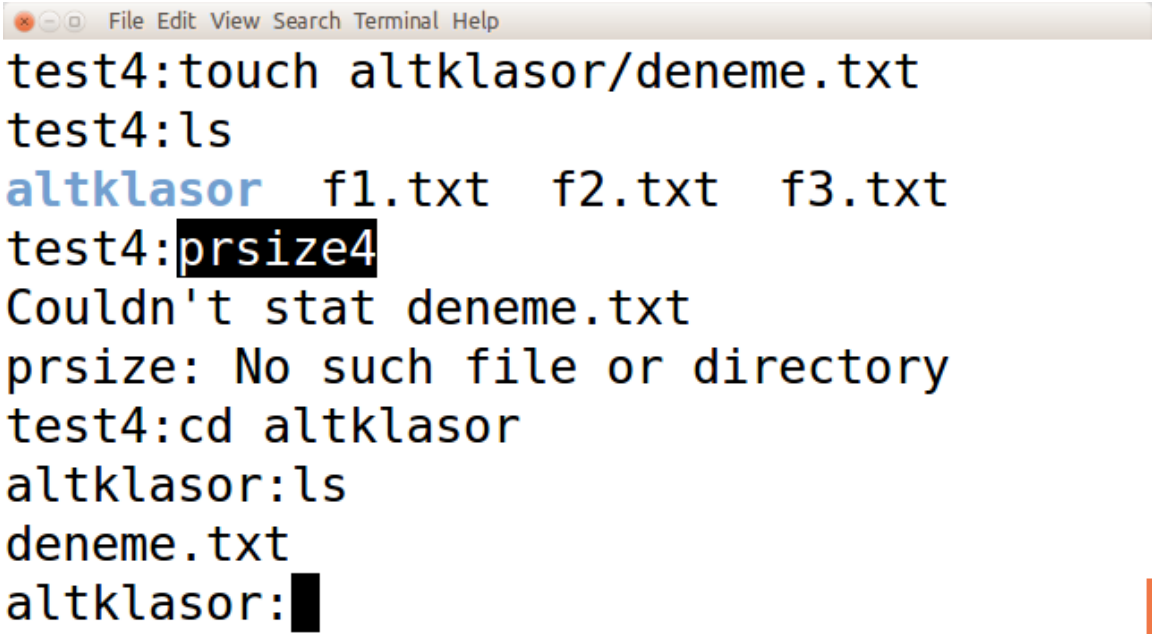
C ▾ Tab Width: 8 ▾ Ln 33, Col 33 INS

File Edit View Search Terminal Help

```
test4:ls -la
total 20
drwxr-xr-x 2 root root 4096 Mar 28 21:44 .
drwx------ 6 bilg bilg 4096 Mar 28 21:43 ..
-rw-r--r-- 1 root root    9 Mar 28 21:44 f1.txt
-rw-r--r-- 1 root root   18 Mar 28 21:44 f2.txt
-rw-r--r-- 1 root root   51 Mar 28 21:44 f3.txt
test4:prsize4
8270
test4:
```

prsize4 : path problem

We solved infinite loop problem but there remains another problem with file paths. It is necessary to write full path starting from «.» (working directory) before doing a recursive call.



prsize5: solution to path problem

256 character file name + 2 character (/ and null) + file path(strlen(fn))

Print the file name to *s variable* together with file path.

File Edit View Search Tools Documents Help

Open Save Undo

prsize5.c ×

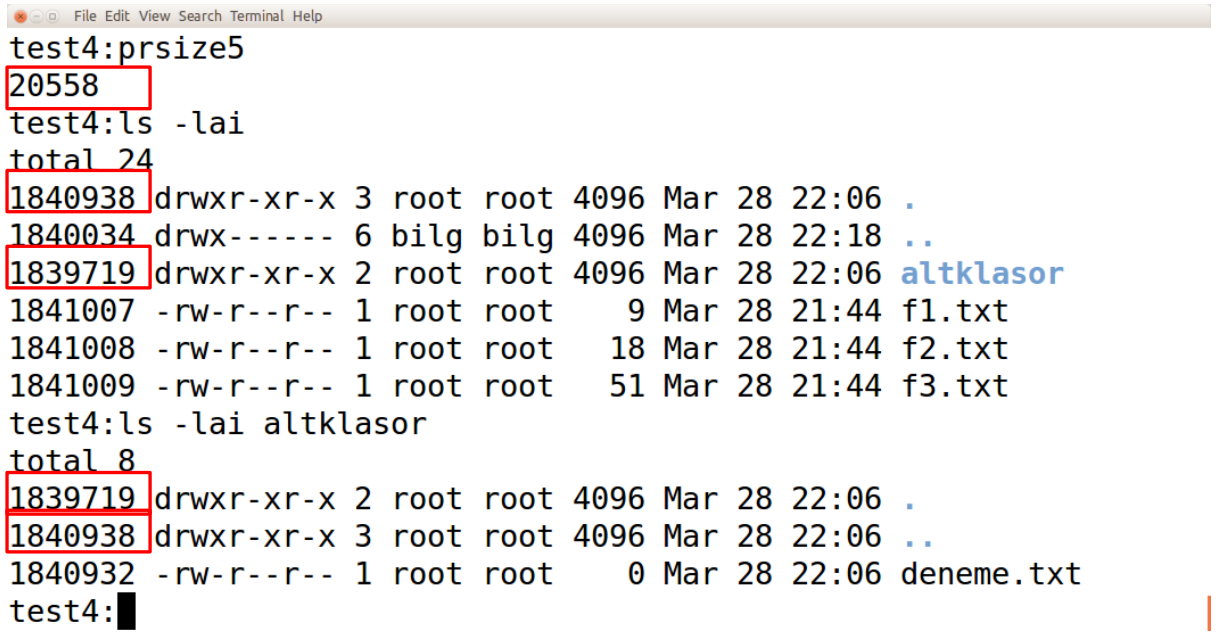
```
char *s;
d = opendir(fn);
if (d == NULL) {
  perror("prsize");
  exit(1);
}
s = (char *) malloc(sizeof(char)*(strlen(fn)+258));

for (de = readdir(d); de != NULL; de = readdir(d)) {
  /* Look for fn/de->d_name */
  sprintf(s, "%s/%s", fn, de->d_name);
  exists = stat(s, &buf);
  if (exists < 0) {
    fprintf(stderr, "Couldn't stat %s\n", s);
  } else {
    total_size += buf.st_size;
  }
  if (S_ISDIR(buf.st_mode) && strcmp(de->d_name, ".") != 0 &&
      strcmp(de->d_name, "..") != 0) {
    total_size += get_size(s);
  }
}
closedir(d);
```

Saving file '/home/bilg/Documents/h... C Tab Width: 8 Ln 20, Col 11 INS

prsize5: repeating files problem

- Prsize5 computed the size of directory as 20558.
- However, the files with some inode numbers are taken into account two times.
- What we need is for **prsize** to be able to recognize hard links, and only count them once.
- How do you recognize whether two files are links to the same disk file?
- You use the inode number. This is held in **buf.st_ino**.



prsize6: solution to repeating files problem

- In order to prevent re-use of the same folders, the inode number of each file included in the calculation can be kept in a BST and used for control purposes.
- The red-black tree in the libfdr library is used to keep the inode list.
- It prevents previously used inode for getting the size of a file to be added to the tree again

File Edit View Search Tools Documents Help

Open Save Undo

prsize6.c ×

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include "jrb.h"

int get_size(char *fn, JRB inodes)
{
  DIR *d;
  struct dirent *de;
  struct stat buf;
  int exists;
  int total_size;
  char *s;

  d = opendir(fn);
  if (d == NULL) {
    perror("prsize");
    exit(1);
  }

  total_size = 0;
  s = (char *) malloc(sizeof(char)*(strlen(fn)+258));

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    /* Look for fn/de->d_name inodes*/
    sprintf(s, "%s/%s", fn, de->d_name);
    exists = stat(s, &buf);
```

C Tab Width: 8 Ln 12, Col 1 INS

File Edit View Search Tools Documents Help

Open Save Undo

prsize6.c ×

```
  s = (char *) malloc(sizeof(char)*(strlen(fn)+258));

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    /* Look for fn/de->d_name inodes*/
    sprintf(s, "%s/%s", fn, de->d_name);
    exists = stat(s, &buf);
    if (exists < 0) {
      fprintf(stderr, "Couldn't stat %s\n", s);
    } else {
      if (jrb_find_int(inodes, buf.st_ino) == NULL) {
        total_size += buf.st_size;
        jrb_insert_int(inodes, buf.st_ino, JNULL);
      }
    }
    if (S_ISDIR(buf.st_mode) && strcmp(de->d_name, ".") != 0 &
        strcmp(de->d_name, "..") != 0) {
      total_size += get_size(s, inodes);
    }
  }
  closedir(d);
  free(s);
  return total_size;
}

main()
{
  JRB inodes;
  inodes = make_jrb();
  printf("%d\n", get_size(".", inodes));
}
```

C Tab Width: 8 Ln 12, Col 1 INS

- Search for the inode in tree. Insert if not in the tree.
- Inode is added as key to prevent repetition.
- We don’t use value (value=JNULL)

prsize7.c: soft (symbolic) link problem

We used the lstat () system call for soft links.

File Edit View Search Tools Documents Help

Open Save Undo

prsize7.c ×

```
s = (char *) malloc(sizeof(char)*(strlen(fn)+258));

for (de = readdir(d); de != NULL; de = readdir(d)) {
  /* Look for fn/de->d_name */
  sprintf(s, "%s/%s", fn, de->d_name);
  exists = lstat(s, &buf);
  if (exists < 0) {
    fprintf(stderr, "Couldn't stat %s\n", s);
  } else {
    if (jrb_find_int(inodes, buf.st_ino) == NULL) {
```

The result using stat()

The result using lstat()

File Edit View Search Terminal Help

```
test4:ln -s f1.txt softf1
test4:ls
altklasor  f1.txt  f2.txt  f3.txt  softf1
test4:prsize6
12366
test4:prsize7
12372
test4:ls -li
total 16
1839719 drwxr-xr-x 2 root root 4096 Mar 28 22:06 altklasor
1841007 -rw-r--r-- 1 root root    9 Mar 28 21:44 f1.txt
1841008 -rw-r--r-- 1 root root   18 Mar 28 21:44 f2.txt
1841009 -rw-r--r-- 1 root root   51 Mar 28 21:44 f3.txt
1841073 lrwxrwxrwx 1 root root    6 Mar 29 00:36 softf1 -> f1.txt
test4:
```

prsize7a: printing visited paths

```
File  Edit  View  Search  Terminal  Help
abc:1_Prsize$ ./directory 10
abc:1_Prsize$ ./prsize7a
Testing .
Testing ./d10
Testing ./d10/d9
Testing ./d10/d9/d8
Testing ./d10/d9/d8/d7
Testing ./d10/d9/d8/d7/d6
Testing ./d10/d9/d8/d7/d6/d5
Testing ./d10/d9/d8/d7/d6/d5/d4
Testing ./d10/d9/d8/d7/d6/d5/d4/d3
Testing ./d10/d9/d8/d7/d6/d5/d4/d3/d2
Testing ./d10/d9/d8/d7/d6/d5/d4/d3/d2/d1
Testing ./test1
413002
```

- *The program starts from the folder in which it is run (“.”) and applies the same process to all subfolders.*
- *While going to a subfolder, the previous folder remains open.*

Open ▾ | directory ~/Documents/h11/1_Prsize | Save

```
#!/bin/sh
x=$1
while [ $x -gt 0 ]
do
        mkdir d$x
        cd d$x
        echo "test" > f$x
        x=`expr $x - 1`
done
```

sh ▾ Tab Width: 8 ▾ Ln 1, Col 8 ▾ INS

prsize8: closing a directory before visiting another

- To reduce the number of files opened from a process, the directory can be closed before going to subfolders.
- In this implementation, subdirectories in the visited directory are added to the list.
- Then directory is closed and the subdirectories are visited.
- The same is true for every visited directory.

File Edit View Search Tools Documents Help

Open Save Undo

prsize8.c ×

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include "jrb.h"
#include "dllist.h"

int get_size(char *fn, JRB inodes)
{
  DIR *d;
  struct dirent *de;
  struct stat buf;
  int exists;
  int total_size;
  char *s;
  Dllist directories, tmp;

  d = opendir(fn);
  if (d == NULL) {
    perror("prsize");
    exit(1);
  }

  total_size = 0;
  s = (char *) malloc(sizeof(char)*(strlen(fn)+258));

  directories = new_dllist();
  for (de = readdir(d); de != NULL; de = readdir(d)) {
    /* Look for fn/de->d_name */
```

C Tab Width: 8 Ln 54, Col 34 INS

prsize8: closing a directory before visiting another

Add to the list if there are subdirectories

Close directory before going to another one

Clean after getting size

File Edit View Search Tools Documents Help

Open Save Undo

prsize8.c ×

```
  directories = new_dllist();
  for (de = readdir(d); de != NULL; de = readdir(d)) {
    /* Look for fn/de->d_name */
    sprintf(s, "%s/%s", fn, de->d_name);
    exists = lstat(s, &buf);
    if (exists < 0) {
      fprintf(stderr, "Couldn't stat %s\n", s);
    } else {
      if (jrb_find_int(inodes, buf.st_ino) == NULL) {
        total_size += buf.st_size;
        jrb_insert_int(inodes, buf.st_ino, JNULL);
      }
    }
    if (S_ISDIR(buf.st_mode) && strcmp(de->d_name, ".") != 0 &&
        strcmp(de->d_name, "..") != 0) {
      dll_append(directories, new_jval_s(strdup(s)));
    }
  }
  closedir(d);
  dll_traverse(tmp, directories) {
     total_size += get_size(tmp->val.s, inodes);
     /* This keeps the program from overgrowing its memory */
     free(tmp->val.s);
  }

  /* As does this */
  free_dllist(directories);
  free(s);
  return total_size;
}

main()
{
  JRB inodes;
  inodes = make_jrb();
  printf("%d\n", get_size(".", inodes));
```

C Tab Width: 8 Ln 67, Col 14 INS

Assignment

Write a program that calculates the total size of the files it takes as parameters using the stat call.

- **UMASK, CHMOD, MKDİR, LINK, UTIME**
- **UNLINK, RENAME, SYMLINK, READLINK**

umask

umask is a system call that handles the "file mode creation mask" .

"file creation mask" (which I will call the "umask" out of habit) is a nine-bit number. If a bit in the umask is set, then whenever you make a system call that creates a file, that bit in the protection mode will be turned off.

Formally, when you specify a mode when you open a file, the real protection mode will be:

(mode & ~umask)

umask

the umask "turns off" protection bits.

The point of the umask is to allow programs to create files with the following protection modes:

- Regular text and data files may be opened with the mode 0666.
- Directories and executable files may be opened with the mode 0777.

umask

```
File  Edit  View  Search  Terminal  Help
abc:~$ #default umask value
abc:~$ umask
0022
abc:~$ echo "test file" > file1
abc:~$ ls -l file1
-rw-r--r-- 1 abc abc 10 Apr 19 00:24 file1
abc:~$
```

- ***Regular text and data files may be opened with the mode 0666.***

rwxrwxrwx

Umask = 0022 =**000010010**

~Umask = 0755 =**111101101**

Mode = 0666 =110110110

mode&~umask =110100100 =0644=**rw-r--r--**

umask

```
abc:~$ umask 0
abc:~$ umask
0000
abc:~$ echo "test file" > file2
abc:~$ ls -l file2
-rw-rw-rw- 1 abc abc 10 Apr 19 00:43 file2
abc:~$
```

- ***Regular text and data files may be opened with the mode 0666.***

rwxrwxrwx

Umask = 0000 =000000000

~Umask = 0777 =111111111

Mode = 0666 =110110110

mode&~umask =110110110 =0666=rw-rw-rw-

umask

```
abc:~$ umask 0777
abc:~$ umask
0777
abc:~$ echo "test file" > file3
abc:~$ ls -l file3
---------- 1 abc abc 10 Apr 19 00:58 file3
```

When my umask is 0777, then all nine bits, so the file has a protection mode of 0000. You'll note, that I was still allowed to write "Hi" into it, because the **open()** call gave me a legal file descriptor for writing. It's just that no other process can now open the file.

- ***Regular text and data files may be opened with the mode 0666.***

rwxrwxrwx

Umask = 0777 =111111111

~Umask = 0000 =000000000

Mode = 0666 =110110110

mode&~umask =000000000 =0000=---------

umask

```
abc:~$ umask 022
abc:~$ umask
0022
abc:~$ gcc prog1.c -o prog1
abc:~$ ls -l prog1
-rwxr-xr-x 1 abc abc 8304 Apr 19 01:02 prog1
```

- ***Directories and executable files may be opened with the mode 0777.***

rwxrwxrwx

Umask = 0022 =000010010

~Umask = 0755 =111101101

Mode = 0777 =111111111

mode&~umask =111101101 =0755=rwxr-xr-x

umask

```
abc:~$ umask 0
abc:~$ gcc prog1.c -o prog2
abc:~$ ls -l prog2
-rwxrwxrwx 1 abc abc 8304 Apr 19 01:07 prog2
```

- ***Directories and executable files may be opened with the mode 0777.***

rwxrwxrwx

Umask = 0000 =**000000000**

~Umask = 0777 =**111111111**

Mode = 0777 =111111111

mode&~umask =111111111 =0777=**rwxrwxrwx**

umask

```
abc:~$ umask 022
abc:~$ mkdir d1
abc:~$ ls -ld d1
drwxr-xr-x 2 abc abc 4096 Apr 19 01:13 d1
abc:~$ umask 0
abc:~$ mkdir d2
abc:~$ ls -ld d2
drwxrwxrwx 2 abc abc 4096 Apr 19 01:14 d2
abc:~$ umask 0777
abc:~$ mkdir d3
abc:~$ ls -ld d3
d--------- 2 abc abc 4096 Apr 19 01:14 d3
```

chmod

Works just like chmod when executed from the shell

chmod("f1", 0600) will set the protection of file **f1** to be "rw-" for you, and "---" for everyone else

```
CHMOD(2)                    Linux Programmer's Manual                   CHMOD(2)

NAME
       chmod, fchmod, fchmodat - change permissions of a file

SYNOPSIS
       #include <sys/stat.h>

       int chmod(const char *pathname, mode_t mode);
       int fchmod(int fd, mode_t mode);

       #include <fcntl.h>           /* Definition of AT_* constants */
       #include <sys/stat.h>

       int fchmodat(int dirfd, const char *pathname, mode_t mode, int flags);
```

chmod

```c
int main()
{
  int fd;

  printf("Opening the file:\n");
  fd = open("f1.txt", O_WRONLY | O_CREAT | O_TRUNC);
  sleep(1);

  printf("Doing chmod\n");
  chmod("f1.txt", 0000);
  sleep(1);

  printf("Doing write\n");
  write(fd, "Hi\n", 3);

  return 0;
}
```

chmod

```
abc:2_Umask-And-Others$ ./o1
Opening the file:
Doing chmod
Doing write
abc:2_Umask-And-Others$ ls -l f1.txt
---------- 1 abc abc 3 Apr 19 16:47 f1.txt
abc:2_Umask-And-Others$ cat f1.txt
cat: f1.txt: Permission denied
```

link and unlink

link(char *f1, char *f2) works just like:

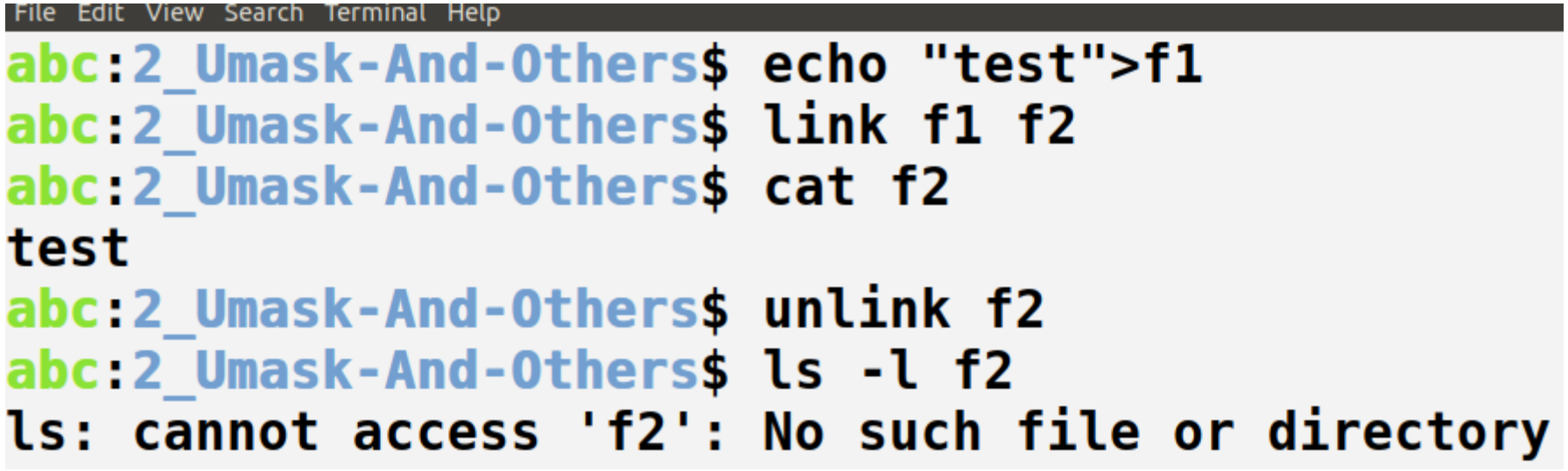
ln f1 f2

---

f2 has to be a file -- it cannot be a directory.

unlink(char *f1) works like:

rm f1



remove

remove(char *f1) works like unlink(), but it also works for (empty) directories. Unlink() fails on directories.

```c
#include <unistd.h>

int main()
{
  int fd;
  char s[11];
  int i;

  printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
  strcpy(s, "Fun Fun\n");
  fd = open("f1.txt", O_RDONLY);
  sleep(1);

  printf("Removing f1.txt\n");
  remove("f1.txt");
  sleep(1);

  printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
  system("ls -l f1.txt");
  i = read(fd, s, 10);
  s[i] = '\0';
  printf("Read returned %d: %d %s\n", i, fd, s);
  return 0;
}
```

This program opens **f1.txt** for reading, sleeps a second, and then removes **f1.txt**. It sleeps again, performs a long listing and then tries to read 10 bytes from the open file. The question is -- what happens when we remove **f1.txt**? Will the read call succeed, or fail because the file is gone?

remove

The **ls** command shows that **f1.txt** is indeed gone after the **remove()** call.

However, the operating system does not delete the file until the last file descriptor to it is closed. For that reason, the **read()** call succeed

```
#include <unistd.h>

int main()
{
  int fd;
  char s[11];
  int i;

  printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
  strcpy(s, "Fun Fun\n");
  fd = open("f1.txt", O_RDONLY);
  sleep(1);

  printf("Removing f1.txt\n");
  remove("f1.txt");
  sleep(1);

  printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
  system("ls -l f1.txt");
  i = read(fd, s, 10);
  s[i] = '\0';
  printf("Read returned %d: %d %s\n", i, fd, s);
  return 0;
}
```

remove

```
abc:2_Umask-And-Others$ echo "system programming">f1.txt
abc:2_Umask-And-Others$ ./o2
Opening f1.txt and putting "Fun Fun" into s.
Removing f1.txt
Listing f1.txt, and reading 10 bytes from the open file descriptor.
ls: cannot access 'f1.txt': No such file or directory
Read returned 10: 3 system pro
```

```
  int i;

  printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
  strcpy(s, "Fun Fun\n");
  fd = open("f1.txt", O_RDONLY);
  sleep(1);

  printf("Removing f1.txt\n");
  remove("f1.txt");
  sleep(1);

  printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
  system("ls -l f1.txt");
  i = read(fd, s, 10);
  s[i] = '\0';
  printf("Read returned %d: %d %s\n", i, fd, s);
  return 0;
}
```

remove

```
abc:2_Umask-And-Others$ ./o2
Opening f1.txt and putting "Fun Fun" into s.
Removing f1.txt
Listing f1.txt, and reading 10 bytes from the open file descriptor.
ls: cannot access 'f1.txt': No such file or directory
Read returned -1: -1 Fun Fun
```

What happened?

First, the **open()** call failed and returned -1. Thus, the **read()** call also failed and returned -1.

Since the **read** call failed, the bytes of **s** were never overwritten - thus when we printed them out, we got "Fun Fun."

Make sure you understand this code and its output.

It is deterministic -- we are not getting segmentation violations or random behavior with these calls -- we are simply getting well-defined errors in our system calls.

rename

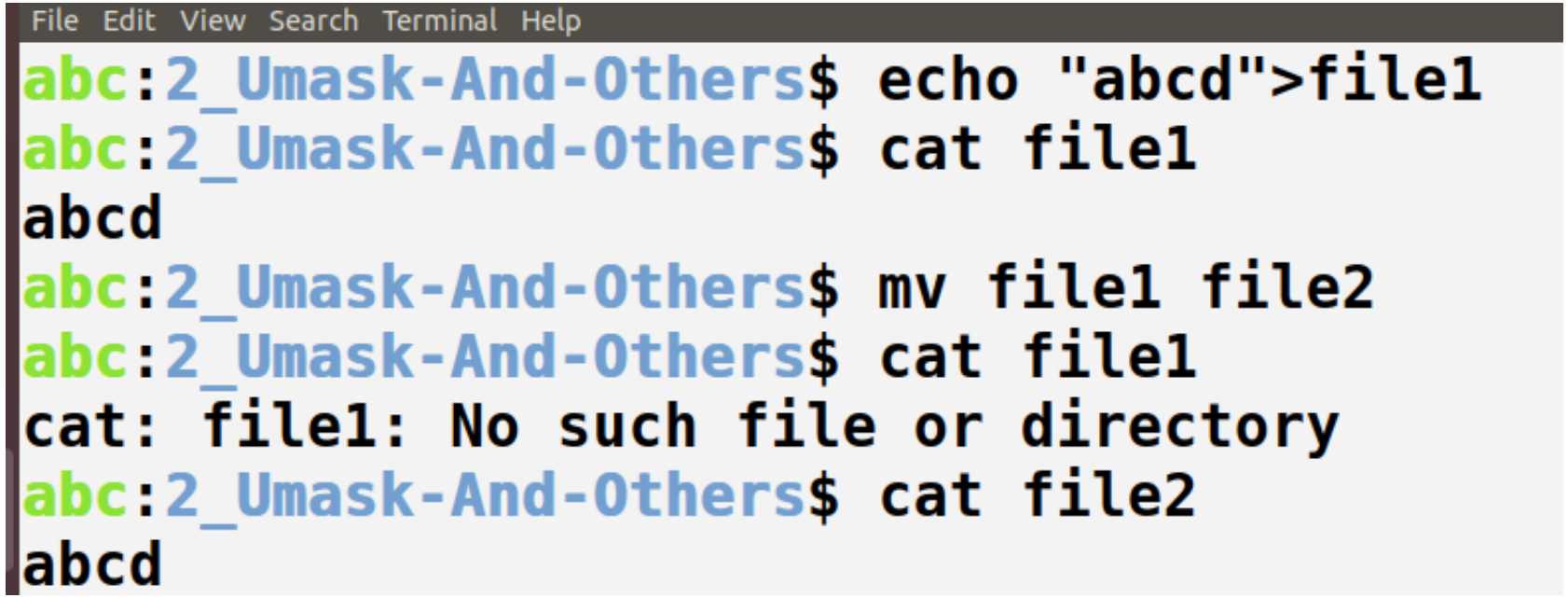
rename() renames a file moving it between directories if required.

Any other hardlinks to the file are unaffected.

---

rename(char *f1, char *f2) works just like:

mv f1 f2



symlink and readlink

Symlink() creates a symbolic link

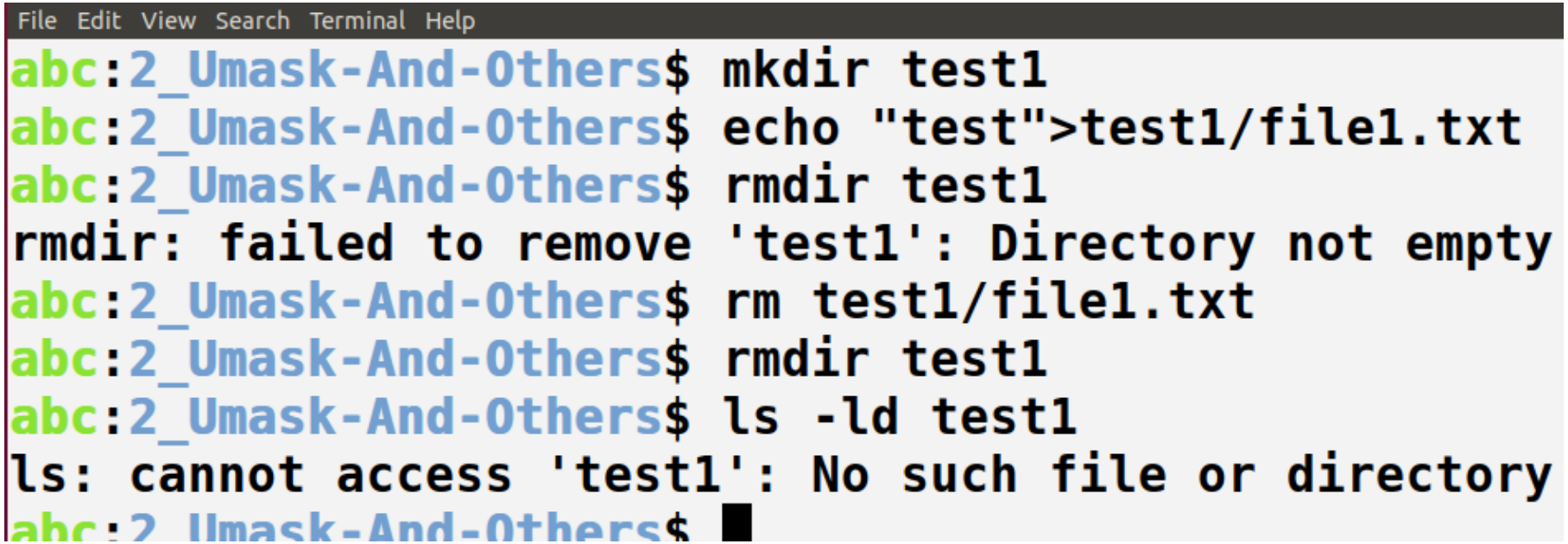
Readlink() reads the symbolic link pathname

```
abc:2_Umask-And-Others$ echo "abcd 123">file1
abc:2_Umask-And-Others$ ln -s file1 file2
abc:2_Umask-And-Others$ cat file2
abcd 123
abc:2_Umask-And-Others$ readlink file2
file1
abc:2_Umask-And-Others$
```

mkdir and rmdir

Mkdir attempts to create a directory

Rmdir deletes a directory, which must be empty



utime

Change last access and modification times

This system call lets you change the time fields of a file's inode.

It looks like it should be illegal (for example, one could write a program to make it look like one has finished his homework on time...)

```
struct tm {
    int tm_sec;         /* seconds */
    int tm_min;         /* minutes */
    int tm_hour;        /* hours */
    int tm_mday;        /* day of the month */
    int tm_mon;         /* month */
    int tm_year;        /* year */
    int tm_wday;        /* day of the week */
    int tm_yday;        /* day in the year */
    int tm_isdst;       /* daylight saving time */
};
```

utime

int **utime**(const char ***filename**, const **struct utimbuf** ***times**);

```
struct utimbuf {
        time_t actime;      /* access time */
        time_t modtime;     /* modification time */
    };
```

- int **utimes**(const char ***filename**, const **struct timeval times[2]**);

```
struct timeval {
        long tv_sec;       /* seconds */
        long tv_usec;      /* microseconds */
    };
```