Systems Programming

The Stat System Call

Shell revisited : Redirecting stderr

- 3.3 Saving stderr to a file
- gcc prog.c 2> compile-errors.txt

- 3.5 Saving stderr to stdout
- gcc prog.c 2> &1

- 3.5 Saving stderr and stdout to a file
- rm -f $(find / -name core) &> /dev/null

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html

/proc (Proc File System)

- “an interface to kernel data structures in a form that looks like files and directories”
- /proc/<pid> - directory of info about running process
  - /proc/1 – information on init
- /proc/cpuinfo
- /proc/sys/fs/file-max

tlpi-dist/list_files.c

```
DIR *dirp;
struct dirent *dp;
Boolean isCurrent; /* True if 'dirpath' is "." */
isCurrent = strcmp(dirpath, ".") == 0;
dirp = opendir(dirpath);
if (dirp  == NULL) {
    errMsg("opendir failed on '%s'", dirpath);
    return;
}
```

tlpi-dist/list_files.c

```
  /* For each entry in this directory, print directory +
filename */
  for (;;) {
      errno = 0; // To distinguish error from end-of-directory
      dp = readdir(dirp);
      if (dp == NULL)
          break;

      if (strcmp(dp->d_name, ".") == 0 || strcmp(dp->d_name,
"..") == 0)
          continue;             /* Skip . and .. */

      if (!isCurrent)
          printf("%s/", dirpath);
      printf("%s\n", dp->d_name);
  }
```

tlpi-dist/list_files.c

```
if (errno != 0)
    errExit("readdir");
if (closedir(dirp) == -1)
    errMsg("closedir");
```

TLPI Source Code for practice

Please Download the book's source code:

https://github.com/posborne/linux-programming-interface-exercises

or

https://projects.ui.ac.id/attachments/download/7181/linux-programming-interface-exercises-master.zip

Contents in "tlpi-dist" directory:

```
root> ls
acl                getopt             pmsg       svmsg
…
daemons  Makefile.inc.MacOSX      pshm       tty README
dirs_links  Makefile.inc.Solaris  pty         users_groups
…
files              pgsjc              sockets
filesys            pipes              svipc
```

NB: Please Compile the source codes using "make"

TLPI/dirs_links

- **cd dir_links**
- **ls**

```
bad_symlink.c
list_files_readdir_r.c
t_dirbasename.c
file_type_stats.c
Makefile
t_unlink.c
list_files.c
nftw_dir_tree.c
view_symlink.c
```

list_files_readdir_r.c : listFiles()

```
DIR *dirp;
Boolean isCurrent; /* True if 'dirpath'
is "." */
struct dirent *result, *entryp;
int nameMax;

isCurrent = strcmp(dirpath, ".") == 0;
nameMax = pathconf(dirpath,
_PC_NAME_MAX);
if (nameMax == -1)  /* Indeterminate or
error */
    nameMax = 255; /* So take a guess */
entryp = malloc(offsetof(struct dirent,
d_name) + nameMax + 1);
if (entryp == NULL)
    errExit("malloc");
dirp = opendir(dirpath);
if (dirp == NULL) {
    errMsg("opendir failed on '%s'",
dirpath);
    return;
}
```

```
for (;;) {
    errno = readdir_r(dirp, entryp,
&result);
    if (errno != 0)
        errExit("readdir_r");

    if (result == NULL) /* End of stream */
        break;

    /* Skip . and .. */

    if (strcmp(entryp->d_name, ".") == 0 ||
            strcmp(entryp->d_name, "..")
== 0)
        continue;

    /* Print directory + filename */

    if (!isCurrent) printf("%s/", dirpath);
    printf("%s\n", entryp->d_name);
}

if (closedir(dirp) == -1)
    errMsg("closedir");
```

Example 18-2. Scanning a directory

readdir_r - reentrant version of readdir

The list_files.c use readdir

The list_files_readdir_r.c use readdir_r

Want to find out readdir vs readdir_r ?

Please call the man

man readdir

So where does ls get all that information on files?

The inode table contains information on files: permissions, times, etc.

To allow programmers to get at this information Unix provides the stat system call

man 2 stat

STAT(2) Linux Programmer's Manual STAT(2)

NAME

stat, fstat, lstat - get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *path, struct stat *buf);  /* file path*/
int fstat(int fd, struct stat *buf);  /* file descriptors */
int lstat(const char *path, struct stat *buf);  /* symbolic link path */
```

Stat structure (man 2 stat)

```
struct stat {
        dev_t     st_dev;     /* ID of device containing file */
        ino_t     st_ino;     /* inode number */
        mode_t    st_mode;    /* file type and permissions */
        nlink_t   st_nlink;   /* number of hard links */
        uid_t     st_uid;     /* user ID of owner */
        gid_t     st_gid;     /* group ID of owner */
        dev_t     st_rdev;    /* device ID (if special file) */
        off_t     st_size;    /* total size, in bytes */
        blksize_t st_blksize; /* blocksize for file system I/O */
        blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
        time_t    st_atime;   /* time of last access */
        time_t    st_mtime;   /* time of last modification */
        time_t    st_ctime;   /* time of last status change */
    };
```

st_mode macros (man 2 stat)

S_ISREG(m) is it a regular file?

S_ISDIR(m) directory?

S_ISCHR(m) character device?

- A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).
- Examples for Character Devices: serial ports, parallel ports, sounds cards.

S_ISBLK(m) block device?

- A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data.
- Examples for Block Devices: hard disks,

S_ISFIFO(m) FIFO (named pipe)?

S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)

S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

st_mode Flags (man 2 stat)

- ls /usr/include/sys

…

- less /usr/include/sys/stat.h

…

```
#define S_IRUSR __S_IREAD     /* Read by owner.  */
#define S_IWUSR __S_IWRITE    /* Write by owner.  */
#define S_IXUSR __S_IEXEC     /* Execute by owner.  */
```

…

```
#define S_IRGRP (S_IRUSR >> 3)  /* Read by group.  */
```

Finding Foo

```
root> man  -s  5   ar
No manual entry for ar
See 'man 7 undocumented'
```

- Finding the command foo try the following options :
  - foo --help, foo -h, foo -?
  - info foo
  - whatis foo, apropos foo
  - dpkg --listfiles foo, dpkg --search foo
  - locate '*foo*'
  - find / -name '*foo*‘
  - check /usr/share/doc/foo, /usr/lib/foo.

Chapter 4. File I/O: The Universal I/O Model

- Unix I/O = files
- File descriptors - index into open file table
- _iob[_NFILES] struct - per process open file table
- standard descriptors
  - 0 = STDIN_FILENO
  - 1 = STDOUT_FILENO
  - 2 = STDERR_FILENO

- Remapping - duplicate [Read Ch 5.4]

Four Key System Calls Performing file I/O

- fd = open (pathname, flags, mode)
- numread = read (fd, buffer, count)
- numwritten = write (fd, buffer, count)
- status = close (fd)

Unix I/O system calls

- ssize_t read(int fildes, void *buf, size_t nbyte);
- ssize_t write(int fildes, const void *buf, size_t nbyte);
- int open(const char *path, int oflag, /* mode_t mode */...);
- int close(int fildes);

- int creat(const char *path, mode_t mode);
  - equivalent to open(path, O_WRONLY | O_CREAT | O_TRUNC, mode)
- off_t lseek(int fildes, off_t offset, int whence);

TLPI/fileio/copy.c

```
…
int
main(int argc, char *argv[])
{
    int inputFd, outputFd, openFlags;
    mode_t filePerms;
    ssize_t numRead;
    char buf[BUF_SIZE];

    if (argc != 3 || strcmp(argv[1],
"--help") == 0)
        usageErr("%s old-file new-file\n",
argv[0]);

    /* Open input and output files */
    inputFd = open(argv[1], O_RDONLY);
    if (inputFd == -1)
        errExit("opening file %s",
argv[1]);
    openFlags = O_CREAT | O_WRONLY | O_TRUNC;
    filePerms = S_IRUSR | S_IWUSR | S_IRGRP |
S_IWGRP | S_IROTH | S_IWOTH;  /* rw-rw-rw- */
    outputFd = open(argv[2], openFlags,
filePerms);
    if (outputFd == -1)
        errExit("opening file %s", argv[2]);

    /* Transfer data until we encountered EOF
or an error */
    while ((numRead = read(inputFd, buf,
BUF_SIZE)) > 0)
        if (write(outputFd, buf, numRead) !=
numRead)
            fatal("couldn't write whole
buffer");

    if (numRead == -1)
        errExit("read");
    if (close(inputFd) == -1)
        errExit("close input");
    if (close(outputFd) == -1)
        errExit("close output");
      exit(EXIT_SUCCESS);
}
```

Universality of I/O

```
$ ./copy test test.old
```
-- Copy a regular file

```
$ ./copy a.txt  /dev/tty
$ ./copy /dev/tty b.txt
$ ./copy /dev/pts/16  /dev/tty
```

/dev/null

OPEN

NAME

open, creat - open and possibly create a file or device

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);

int creat(const char *pathname, mode_t mode);
```

Table 4-2. File access modes

Access mode	Description
O_RDONLY	Open the file for reading only
O_WRONLY	Open the file for writing only
O_RDWR	Open the file for both reading and writing

Table 4.3 Flags for open

Flag	Purpose	SUS?
O_RDONLY	Open for reading only	v3
O_WRONLY	Open for writing only	v3
O_RDWR	Open for reading and writing	v3
O_CLOEXEC	Set the close-on-exec flag	v4
O_CREAT	Create file if it doesn't already exist	v3
O_DIRECT	File I/ O bypasses buffer cache	
O_DIRECTORY	Fail if pathname is not a directory	v4
O_EXCL With O_CREAT	create file exclusively	v3

Btw, what is SUS ?

Is it a Sus Cake (Kue soes merdeka) ? :)

Chapter 3.6 - Portability Issue

Table 4.3 Flags for open

Flag	Purpose	SUS?
O_RDONLY	Open for reading only	v3
O_WRONLY	Open for writing only	v3
O_RDWR	Open for reading and writing	v3
O_CLOEXEC	Set the close-on-exec flag (since Linux 2.6.23)	v4
O_CREAT	Create file if it doesn't already exist	v3
O_DIRECT	File I/O bypasses buffer cache	
O_DIRECTORY	Fail if *pathname* is not a directory	v4
O_EXCL	With O_CREAT: create file exclusively	v3
O_LARGEFILE	Used on 32-bit systems to open large files	
O_NOATIME	Don't update file last access time on *read()* (since Linux 2.6.8)	
O_NOCTTY	Don't let *pathname* become the controlling terminal	v3
O_NOFOLLOW	Don't dereference symbolic links	v4
O_TRUNC	Truncate existing file to zero length	v3
O_APPEND	Writes are always appended to end of file	v3
O_ASYNC	Generate a signal when I/O is possible	
O_DSYNC	Provide synchronized I/O data integrity (since Linux 2.6.33)	v3
O_NONBLOCK	Open in nonblocking mode	v3
O_SYNC	Make file writes synchronous	v3

Kerrisk, Michael (2011-02-11). The Linux Programming Interface: A Linux and UNIX System Programming Handbook.

Listing 4.2 - open file for Reading

Please find the code at:

tlpi-dist/fileio/copy.c

File Descriptor to read source file

File Descriptor to write destination file

```
/* Open input and output files */
    inputFd = open(argv[1], O_RDONLY);
    if (inputFd == -1)
        errExit("opening file %s", argv[1]);

    openFlags = O_CREAT | O_WRONLY | O_TRUNC;
    filePerms = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
                S_IROTH | S_IWOTH;      /* rw-rw-rw- */
    outputFd = open(argv[2], openFlags, filePerms);
    if (outputFd == -1)
        errExit("opening file %s", argv[2]);
```

Errors from open()

- **Retval = -1**
- **Errno**
  - EACCES
  - EISDIR
  - EMFILE
  - ENFILE
  - ENOENT – file does not exist and O_CREAT not specified
  - EROFS – file system read only
  - ETXTBSY – executable that is currently running

Creat() system call

- **Creat equivalent to open with certain flags**
- `fd = open( pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`

- **Less powerful compared to open(), thus obsolete**

The Read system call

NAME

read - read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

DESCRIPTION

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

If count is zero, read() returns zero and has no other results.

If count is greater than SSIZE_MAX, the result is unspecified.

Write, Close

#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION - write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.

SYNOPSIS

…….

int close(int fd);

DESCRIPTION - close() closes a file descriptor, so that it no longer refers to any file and may be reused.

Changing the file OFFSET - lseek

NAME

lseek - reposition read/write file offset

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

off_t   lseek(int fd,   off_t    offset,   int   whence);
```

DESCRIPTION - The lseek() function repositions the offset of the open file associated with the file descriptor fd to the argument offset according to the directive whence as follows:

SEEK_SET - The offset is set to offset bytes.

SEEK_CUR - The offset is set to its current location plus offset bytes.

SEEK_END - The offset is set to the size of the file plus offset bytes.

Figure 4.1 illustrating lseek

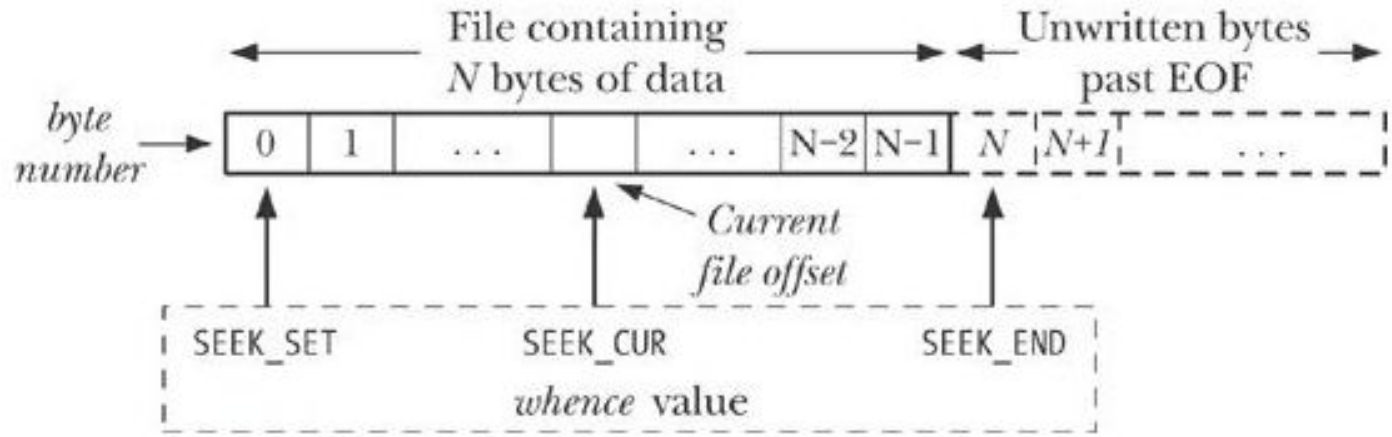

Figure 4-1. Interpreting the whence argument of lseek()

```
lseek(fd, 0 SEEK_SET); /* Start of file */
lseek(fd, 0, SEEK_END); /* Next byte after the EOF */
lseek(fd, -1, SEEK_END); /* Last byte of file */
lseek(fd, -10, SEEK_CUR); /* 10 bytes prior to current */
lseek(fd, 100, SEEK_CUR); /* 100 bytes past of current */
```

ioctl

IOCTL(2) Linux Programmer's Manual IOCTL(2)

NAME

ioctl - control device

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int d, int request, ...);

DESCRIPTION - The ioctl() function manipulates the underlying device parameters of special files.

Example IOCTL usage

- Example 1: IOCTL on CDROM drive

  First, create the file descriptor
  fd = open('/dev/cdrom', O_RDONLY | O_NONBLOCK);

  Eject:
  ioctl(fd, CDROMEJECT, 0);

  Close Tray:
  ioctl(fd, CDROMCLOSETRAY, 0);

QA

Example IOCTL usage

- Example 2: Keyboard LED

```
#include <stdio.h>
#include <linux/kd.h>

int main(int argc, char *argv[])
{
  ioctl(1, KDSETLED, NUM_LED);
  return 0;
}
```

Where is the open file descriptor ?

Why accessing the Keyboard can use FD = 1 ?

More flags

- O_EXCL With O_CREAT: create file exclusively v3
- O_LARGEFILE Used on 32-bit systems to open large files
- O_NOATIME Don’t update file last access time on read() (since Linux 2.6.8)
- O_NOCTTY Don’t let pathname become the controlling terminal v3
- O_NOFOLLOW Don’t dereference symbolic links v4
- O_TRUNC Truncate existing file to zero length v3
- O_APPEND Writes are always appended to end of file v3
- O_ASYNC Generate a signal when I/ O is possible
- O_DSYNC Provide synchronized I/ O data integrity (since Linux 2.6.33) v3
- O_NONBLOCK Open in nonblocking mode v3
- O_SYNC Make file writes synchronous v3

Open file structures

- per process table
- open file table
- vtable