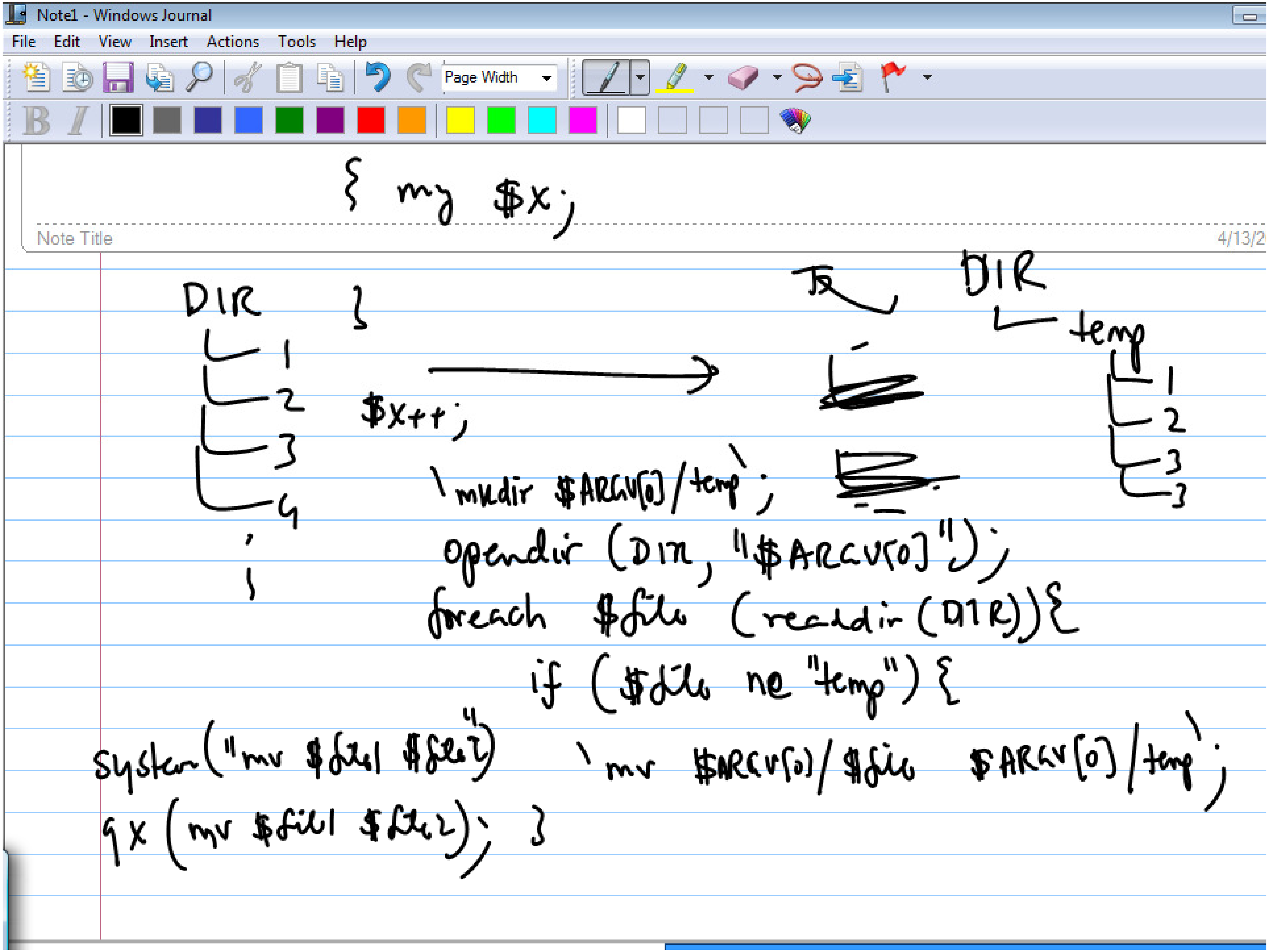
Systems Programming with C

15-123

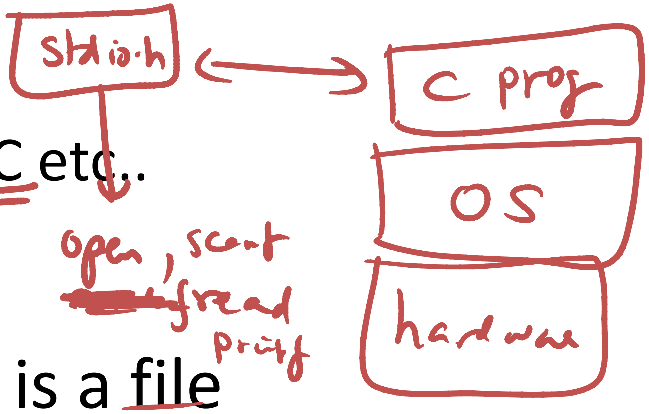
Systems Skills in C and Unix



```
{ my $x;

DIR   }

$x++;

`mkdir $ARGV[0]/temp`;
opendir (DIR, "$ARGV[0]");
foreach $file (readdir(DIR)){
    if ($file ne "temp") {
system("mv $fil1 $fil2")    `mv $ARGV[0]/$file  $ARGV[0]/temp`;
qx(mv $fil1 $fil2);   }
```

Why Systems Programming?

- To access computers resources at a lower level using system calls
  - Examples
    - Managing files, processes, IPC etc..



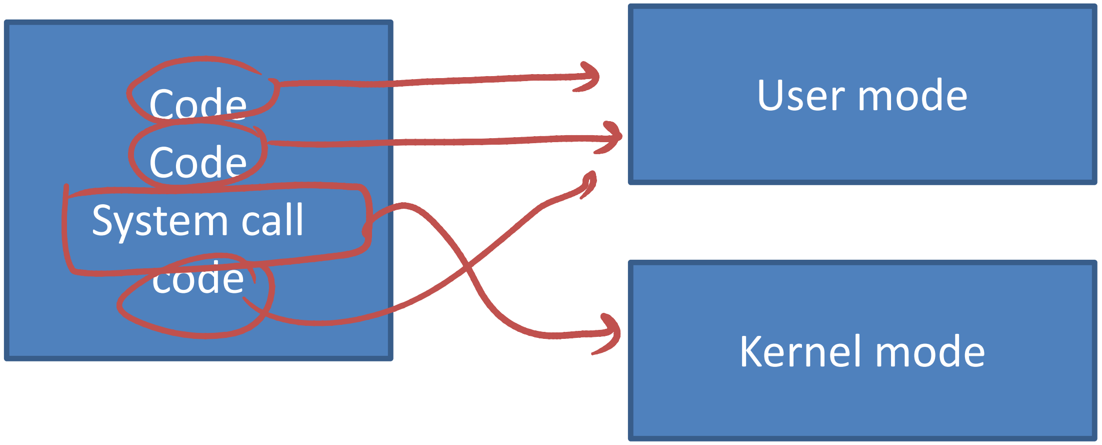
- Managing Files
  - In Unix, any I/O component is a file
    - stdin, stdout, device files, sockets
  - All files created, open, read the same way

scanf fread

read — low level

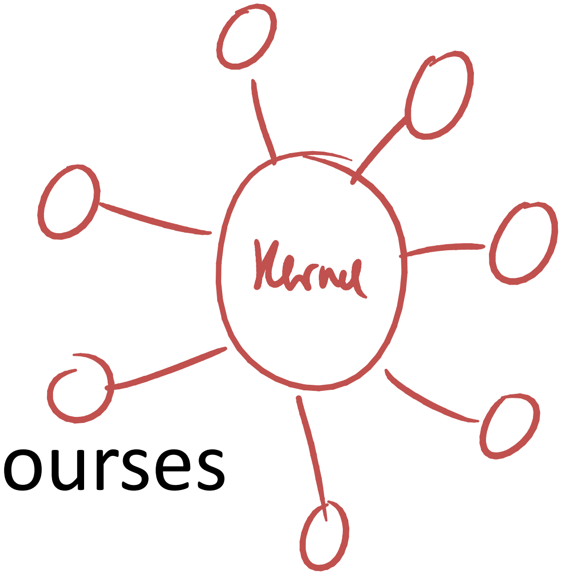
What is a system call?

- A direct request to the operating system to do something on behalf of the program
- Typically programs are executed in user mode
- System call allows a switch from user mode to kernel mode



Unix Kernel

- The core of the unix operating system
- Managing
  - Processes
  - Files
  - Networking etc..
- More details from OS courses



in Kernel Mode

- All programs run in
  - user mode
    - can be replaced by another process at any time
  - kernel mode
    - cannot be arbitrarily replaced by another process.
- A process in kernel mode
  - can be suspended by an **interrupt** or **exception**.
- A C system call
  - A software instruction that generates an OS interrupt or **operating system trap**
  - Assembly instruction Xo80

Using System Calls

- To manage
  - **the file system**
    - **Open, creat, close, read**
  - **control processes**
    - ~~folk~~ fork, exec
  - **provide communication** between multiple processes.
    - pipes

File Systems

Create System Call

RWX
111 = 7
100 = 4
010 = 2
001 = 1

Creat("guna", 0444);

#include <fcntl.h>

int creat(char* filename, mode_t mode)

└ guna

- The mode
  - is an octal number
    - **Example: 0444** indicates that r access for USER, GROUP and ALL for the file.
  - If the file exists, the creat is ignored and prior content and rights are maintained.

Opening Files

open("gunn", O-RDONLY | O-WRONLY ,0444)

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int open(char* filename, int flags, mode_t mode);

- Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC, O_APPEND
- Mode: Specifies permission bits of the file
  - S_IRUSR, S_IWUSR, S_IXUSR – owner permission
  - S_IRGRP, S_IWGRP, S_IXGRP – group permission
  - S_IROTH, S_IWOTH, S_IXOTH – other permission

More on open

- Each open call generates a file descriptor (by kernel)
- Kernel keeps track of all open files
  - Up to 16 in general
- Each unix shell starts with 3 standard files
  - stdin (descriptor 0)
  - stdout (descriptor 1)
  - stderr (descriptor 2)

```
int fd = open ("gu ve", —, —);
```

- All other file descriptors are assigned sequentially

Reading/Writing Files

write(1, buf, 19);

↳ loop printf("%c", buf[i]);

- Low level read and write
- #include <unistd.h>
- **ssize_t read(int fd, void *buf, size_t n);**
  - Returns num bytes read or -1
- **ssize_t write(int fd, const void *buf, size_t n);**
  - Returns num bytes written or -1

char buf[20];

read(0, buf, 19); → loop scanf("%c", &buf[i]);

↑ stdin ↑ ↑

↓

fscanf(stdin, "%c", &buf[i]);

lseek function

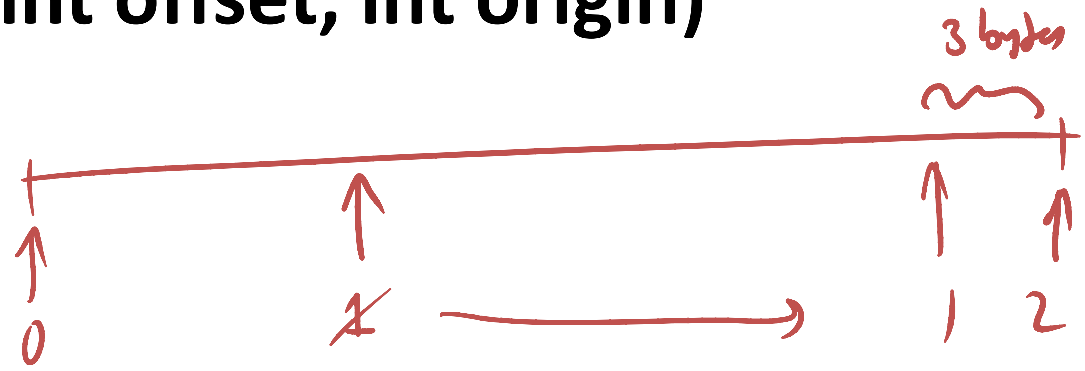
lseek(3, 54, 0);

- #include <sys/types.h>
- #include <unistd.h>
- lseek moves the cursor to a desired position

long lseek(int fd, int offset, int origin)

origin	position
0	beginning of the file
1	Current position
2	End of the file

End of the file



- **Examples**

lseek(3, -3, 2);

Closing a file

- include <unistd.h>
- int close(int fd);
  - Return 0 (success)
  - Return -1 (error)

Example

```
int main(void){
    char c;
    while (read(0,&c,1) != 0)
        write(1, &c, 1);
    exit(0);
}
```

- What does it do?

Example

```
int foo(char s[], int size){
    char* tmp = s;
    while (--size>0 && read(0,tmp,1)!=0 &&
    *tmp++ != '\n');
    *tmp = '\0';
    return (tmp-s);
}
```

- What does it do?

What about size_t and ssize_t

- size_t – unsigned int
- ssize_t - signed int
- How does this affect the range of values in each type?
  - with 32-bit int?

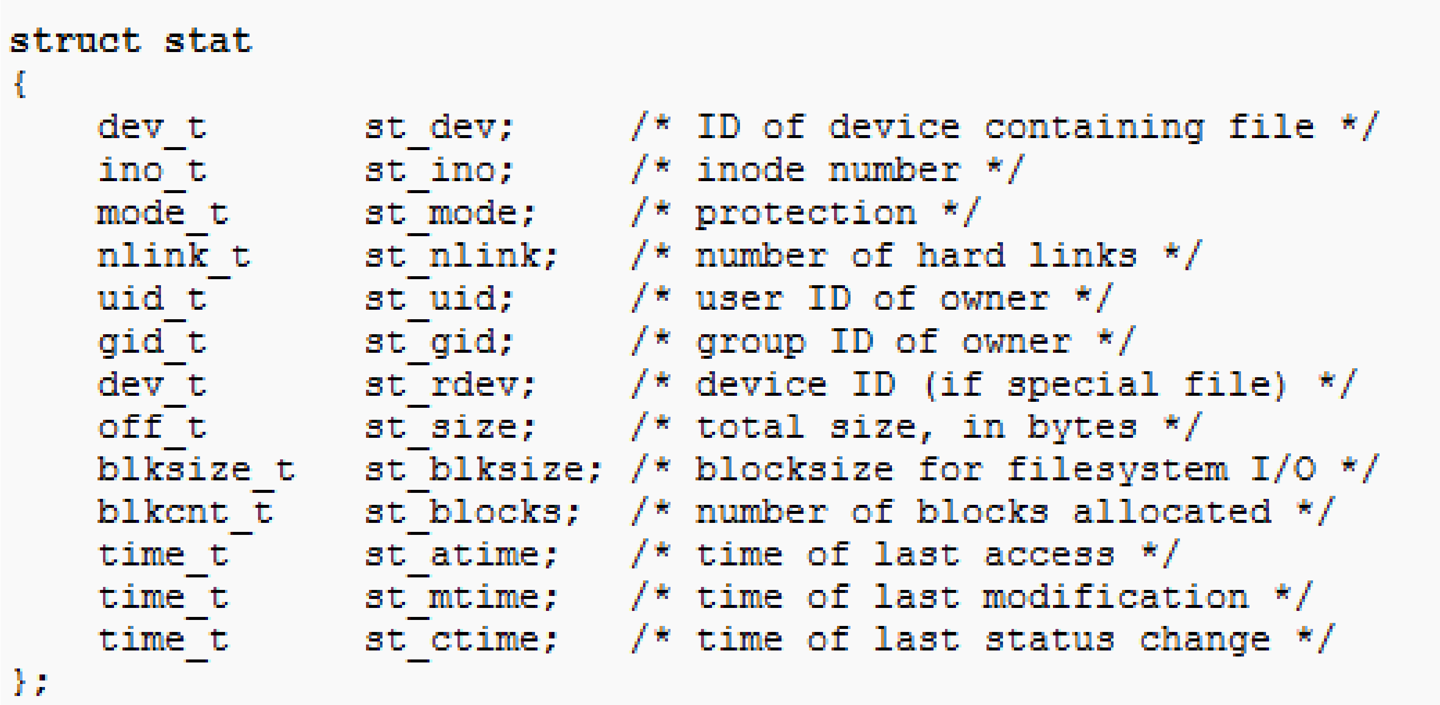
What can go wrong with read and write?

- processing fewer bytes than requested
  - reaching EOF
  - Reading text lines from stdin
  - Reading and writing network sockets
    - Network delays
    - Buffering constraints

Reading file metadata

- How can we find information about a file
- **#include <unistd.h>**
- **#include <sys/stat.h>**
- **int stat(const char* filename, struct stat *buf);**
- **int fstat(int fd, struct stat *buf);**

What is struct stat?



```
struct stat
{
    dev_t       st_dev;     /* ID of device containing file */
    ino_t       st_ino;     /* inode number */
    mode_t      st_mode;    /* protection */
    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 filesystem I/O */
    blkcnt_t    st_blocks;  /* number of 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 */
};
```

Accessing File Status

stat(char* file, struct stat *buf);

fstat(int fd, struct stat *buf);

struct stat buf; // defines a struct stat to hold file information

stat("filename", &buf) ; // now the file information is placed in the buf

st_atime --- Last access time

st_mtime --- last modify time

st_ctime --- Last status change time

st_size --- total size of file

st_uid – user ID of owner

st_mode – file status (directory or not)

Example

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
struct stat statbuf;

char dirpath[256];
getcwd(dirpath,256);
DIR *dir = opendir(dirpath);
struct dirent *dp;

for (dp=readdir(dir); dp != NULL ; dp=readdir(dir)){
        stat(dp->d_name, &statbuf);
        printf("the file name is %s \n", dp->d_name);
        printf("dir = %d\n", S_ISDIR(statbuf.st_mode));
        printf("file size is %ld in bytes \n", statbuf.st_size);
        printf("last modified time is %ld in seconds \n", statbuf.st_mtime);
        printf("last access time is %ld in seconds \n", statbuf.st_atime);
        printf("The device containing the file is %d\n", statbuf.st_dev);
        printf("File serial number is %d\n\n", statbuf.st_ino);
}
```

How to determine a file type

- S_ISREG
  - A regular file?
- S_ISDIR
  - Is a directory?
  - printf("dir = %d\n", S_ISDIR(statbuf.st_mode));
- S_ISSOCK
  - A network socket

Working Directory

```
#include <unistd.h>
char* getcwd(char * dirname, int );
```

Accessing Directories

struct dirent *readdir(DIR* dp)

returns a pointer to the next entry in the directory. A NULL pointer is returned when the end of the directory is reached. The struct direct has the following format.

```
struct dirent {
  u-long d_ino;                              /* i-node number for the dir
    entry */
  u_short d_reclen;                          /* length of this record */
  u_short d_namelen ;                        /* length of the string in
    d_name */
  char  d_name[MAXNAMLEN+1] ;   /* directory name */
};
```

Creating and removing Directories

- **int mkdir(char* name, int mode);**
- **int rmdir(char* name);**
  - returns 0 or -1 for success or failure.
- **mkdir(“newfiles”, 0400);**
- **rmdir(“newfiles”);**

Example

```
#include <string.h>
#include <sys/types.h>
#include <sys/dir.h>

int search (char* file, char* dir){
    DIR  *dirptr=opendir(dir);
    struct dirent *entry = readdir(dirptr);
    while (entry != NULL) {
       if ( strlen(entry->d_name) == strlen(file) && (strcmp(entry->d_name, file) == 0)
                   return 0; /* return success */
        entry = readdir(dirptr);
    }
    return 1;  /* return failure */
}
```

File Management summary

- **creat( ), open( ), close( )**
  - managing I/O channels
- **read( ), write( )**
  - handling input and output operations
- **lseek( )**
  - for random access of files
- **link(FILE1, FILE2), unlink(FILE)**
  - aliasing and removing files
- **stat( )**
  - getting file status
- **access( ), chmod( ), chown( )**
  - for access control
  - int access(const char *pathname, int mode);
- **chdir( )**
  - for changing working directory
- **mkdir( )**
  - for creating a directory

Dealing with system call interfaces

- System calls interface often change
  - place system calls in subroutines so subroutines
- Error in System Calls
  - returns -1
  - store the error number in a variable called “**errno**” given in a header file called **/usr/include/errno.h**.
- Using perror
  - When a system call returns an error, the function **perror** can be used to print a diagnostic message. If we call **perror( )**, then it displays the argument string, a colon, and then the error message, as directed by "errno", followed by a newline.

```
if (unlink("text.txt")==-1){
    perror("");
}
```

Process Control

Process Control

- **exec( ), fork( ), wait( ), exit( )**
  - for process control
- **getuid( )**
  - for process ownership
- **getpid( )**
  - for process ID
- **signal( ) , kill( ), alarm( )**
  - for process control

Other system functions

- **mmap(), shmget(), mprotect(), mlock()**
  - manipulate low level memory attributes
- **time(), gettimer(), settimer(),settimeofday(), alarm()**
  - time management functions
- **pipe( )**
  - for creating inter-process communication

Coding Examples