Working with Files

Files

- Any passive entity in Linux is a file
- A file is a linear stream of bytes
- Unification of File I/O and Device I/O
- File Types

❖ Regular	-
❖ Directory	d
❖ Link	l
❖ Character Device	c
❖ Block Device	b
❖ Socket	s
❖ Named Pipe	p

Files

- File Permissions
  - Read (r)
  - Write (w)
  - Execute (x)
- User Categories

  -user

  -group

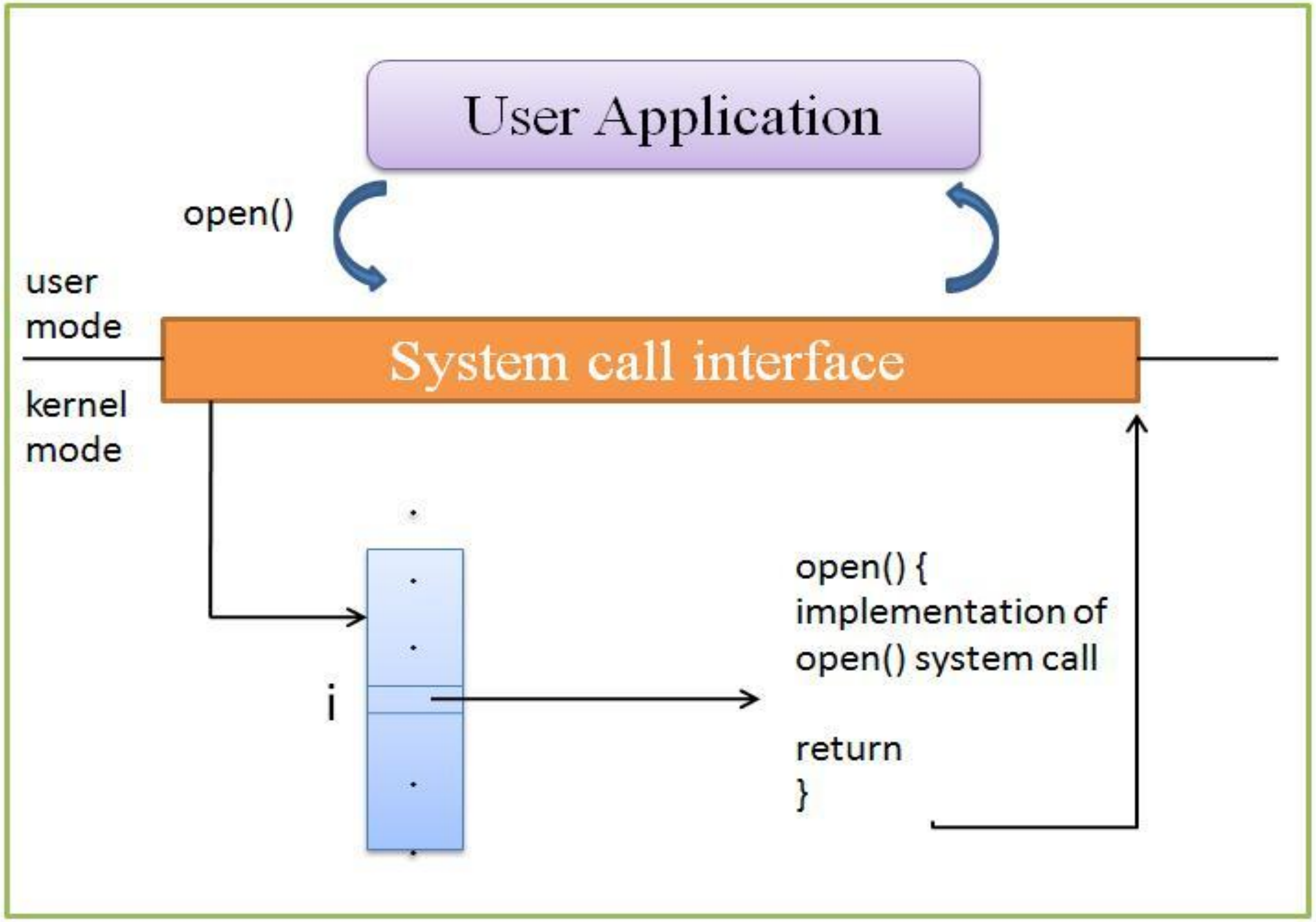
  -other
- File Attributes

```
-rw-rw-r--  1  raju  sec 31744 Feb 21 17:56  intro.txt
```

File System Structure

• /	Begins file system structure, called the root
• /home	Users home directories
• /bin	Standard commands and utility programs
• /usr	Files and commands used by system
• /usr/lib	Host libraries for programming languages
• /dev	Holds file interfaces for devices
• /etc	Holds system configuration files
• /var	Storage for all variable files and temporary files
• /root	The administrative user's home directory
• /sbin	Programs for use by the system and the system administrator.

Access to Kernel Service



System Call Error

- When a system call discovers and error, it returns -1 and stores the reason in an external variable named "errno“

- The "/usr/include/errno.h" file maps these error numbers to manifest constants

File I/O System Calls

- The file related system calls available in LINUX system let you create, open, and close files, read and write files, randomly access files, alias and remove files, get information about files, check the accessibility of files, change protections, owner, and group of files, and control devices.

File I/O System Calls

- open()
- close()
- read()
- write()
- lseek()
- dup()
- dup2
- link()
- unlink()
- stat()
- fstat()
- access()
- chmod()
- chown()
- umask()
- ioctl()

File Descriptor

- Each LINUX process has 256 file descriptors, numbered 0 through 255
- The first three are already opened when the process begins
  - 0: The standard input
  - 1: The standard output
  - 2: The standard error output
- When the parent process forks a process, the child process inherits the file descriptors of the parent

open System Call

- The prototype for the open() system call is:

```
#include <fcntl.h>

int open(file_name, option_flags [, mode]);
char *file_name;
int option_flags, mode;
```

open System Call

The allowable option_flags as defined in "/usr/include/fcntl.h" are:

- #define O_RDONLY 0 /* Open the file for reading only */
- #define O_WRONLY 1 /* Open the file for writing only */
- #define O_RDWR 2 /* Open the file for both reading and writing*/
- #define O_NDELAY 04 /* Non-blocking I/O */
- #define O_APPEND 010 /* append (writes guaranteed at the end) */
- #define O_CREAT 00400 /*open with file create (uses third open arg) */
- #define O_TRUNC 01000 /* open with truncation */
- #define O_EXCL 02000 /* exclusive open */

Multiple values are combined using the | operator (i.e.bitwise OR).

close System Call

- To close a file descriptor, use the close() system call.

- The prototype for the close() system call is:

```
int close(file_descriptor);
int file_descriptor;
```

read System Call

- The read() system call does all input

```
int read(file_descriptor, buffer_pointer, size);
int file_descriptor;
void *buffer_pointer;
unsigned size;
```

write System Call

- write() system call does all output

```
int write(file_descriptor, buffer_pointer, size);
int file_descriptor;
void *buffer_pointer;
unsigned size;
```

lseek System Call

- The LINUX system file system treats an ordinary file as a sequence of bytes.
- Generally, a file is read or written sequentially -- that is, from beginning to the end of the file.
- Sometimes sequential reading and writing is not appropriate.
- Random access I/O is achieved by changing the value of this file pointer using the lseek() system call.

lseek System Call

- long lseek(file_descriptor, offset, whence)

  int file_descriptor;

  long offset;

  int whence;
- whence new position

  ----

  0 offset bytes into the file

  1 current position in the file plus offset

  2 current end-of-file position plus offset

dup System Call

- The dup() system call duplicates an open file descriptor and returns the new file descriptor.
- The new file descriptor has the following properties in common with the original file descriptor:
  - refers to the same open file or pipe.
  - has the same file pointer -- that is, both file
  - descriptors share one file pointer.
  - has the same access mode, whether read, write, or read and write.

dup System Call

- dup() is guaranteed to return a file descriptor with the lowest integer value available.
- It is because of this feature of returning the lowest unused file descriptor available that processes accomplish I/O redirection.
- int dup(file_descriptor);

  int file_descriptor;

dup2 System Call

- dup2 — is similar to dup ( ), but duplicates an open file descriptor to a specific slot.

- int dup2(int fildes, int fildes2);

  fildes is a file descriptor obtained from open(), dup(), fcntl(), or pipe() system call.

  fildes2 is a non-negative integer less than the maximum value allowed for file descriptors.

- dup2() causes *fildes2* to refer to the same file as *fildes*. If *fildes2* refers to an already open file, the open file is closed first.

fstat System Call

- The fstat system call returns status information about the file associated with an open file descriptor
- The information is written into a structure, buf the address of which is passed as an argument

  `int fstat(int fildes,struct stat *buf);`
- stat( ) and lstat ( ) are similar type of system calls

Directory Access

- opendir
- readdir
- telldir
- seekdir
- closedir

Directory Access

- opendir

```
DIR *opendir(const char *name);
```

Directory Access

- readdir

```
struct dirent *readdir(DIR *dirp);
```

Directory Access

- telldir

```
long int telldir(DIR *dirp);
```

Directory Access

- seekdir

```
void seekdir(DIR *dirp, long int loc);
```

Directory Access

- closedir

```
int closedir(DIR *dirp);
```

File and Directory Maintenance

- chmod
- chown
- unlink
- link
- symlink
- mkdir
- rmdir
- chdir

chmod system call

```
int chmod(const char *path, mode_t mode);
```

chown system call

```
int chown(const char *path, uid_t owner,
    gid_t group);
```

unlink System Call

- The opposite of the link() system call is the unlink() system call

- The prototype for unlink() is:

  int unlink(file_name)

  char *file_name;

link System Call

- The LINUX system file structure allows more than one named reference to a given file, a feature called "aliasing“
- Making an alias to a file means that the file has more than one name, but all names of the file refer to the same data
- int link(original_name, alias_name)

  char *original_name, *alias_name;

symlink System Call

int symlink(const char *path1,const char *path2);

mkdir System Call

```
int mkdir(const char *path, mode_t mode);
```

rmdir System Call

```
int rmdir(const char *path);
```

chdir System Call

```
int chdir(const char *path);
```

fcntl System Call

```
int fcntl(int fildes, int cmd);
int fcntl(int fildes,int cmd,long arg);
```

mmap Function

void mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);

Thank You