4 Nov, 2023 - Sa

System Programming Notes

chapter 4:- Unix I/O.

→ Unix uses a uniform device interface, throug file descriptors, that allows the sam I/O calls to be used for terminals, disks, tapes, audio and even network communication.

4.1 Device terminology:

→ A phoripheral device is a piece of hardware accessed by computer system like disks, tapes, screen etc.

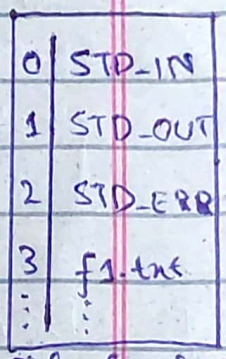
→ User program perform control and I/O to these devices through calls to OS modules called device drivers.

→ Unix has provided uniform access to most devices via five functions: 'open', 'close', 'read', 'write', 'ioctl'. All devices are represented by files, called special files that are located in '/dev' directory

(2) ~ : shows parameters;
* : shows return value.

4.3 Opening and Closing Files:

→ The 'open' system call add an entry in file descriptor table, of a file or device.



File descriptor table

→ SYNOPSIS:

```
#include <fcntl.h>
#include <sys/stat.h>
int open(const char* path, int oflag, ...);
```

~ 'path' points to pathname of file or device.
~ 'oflag' parameter specifies status flags and access modes for opened file.
~ Include a third parameter if you are creating a file. to specify access permissions.

* If succesful open return a non-negative iteger representing file-descriptor
* If unsuccessful, returns -1 and set errno.

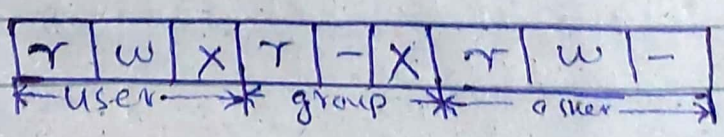
→ Construct the oflag or permissions arguments by bitwise OR (|) of the desired combinations.

→ Some oflags are:

- O_RDONLY, O_WRONLY, O_RDWR for read only, write only and read write only access.
- O_APPEND, O_CREAT, O_EXCL, O_NONCTTY, O_NONBLOCK and O_TRUNC for adding to existing content, creating a file, over-writing a file, prevents an open file from becoming controling terminal, return imidiatly or block untill the device is ready and truncate the open file for writing to 0 respectively

→ Each file has three classe associated with it: a user (owner), a group and everybody else (others).

→ possible permissions are: read (r), write (w) and execute (x)



(4)

Posix symbolic names for permissions.

Symbol	meaning	Symbol	meaning.
S_IRUSR	read by owner	S_IWUSR	write by owner
S_IXUSR	execute " "	S_IRWXU	r, w, x " "
S_IRGRP	read by group	S_IWGRP	write by group
S_IXGRP	execute " "	S_IRWXG	r, w, x " "
S_IROTH	read by other	S_IRWOTH	write by other
S_IXOTH	execute by other	S_IRWXO	r, w, x " "
S_ISUID	set user id on x	S_ISGID	set group id on x

→ The 'close' function is used to remove the entry of a file or device from file descriptor table.

→ Synopsis:

```
#include <unistd.h>
int close (int fildes);
```

~ 'fildes' represent the file whose resources are to be released.

* If successful return 0
* If unsuccessful close returns -1 and set errno.

4.2 Reading and Writing:

→ Unix provides sequential access to files and other devices through the 'read' and 'write' functions.

→ The 'read' function attempts to retrieve 'nbyte' bytes from the file or device represented by filedes into the user variable 'buf'

→ Synopsis:

```
#include <unistd.h>

ssize_t read (int filedes, void *buf,
              sizet nbyte);
```

(ssize_t: signed integer → int.)

~ filedes → file to read
~ buf
~ nbyte

* Succesful: number of byte acctually read.
* Unsuccesful: returns -1 and set errno.
* returns 0; when reached to EOF ('\0' or ctrl+D);

→ When we execute a program from the shell, the program starts with three open streams associated with file descriptors STDIN_FILENO (0), STDOUT_FILENO (1) and STDERR_FILENO (2).

keyboard ← STDIN (0)

STDOUT (1) → screen

→ The write function attempts to output 'nbyte' bytes from the user buffer 'buf' to the file represented by file descriptor 'filedes'.

→ Synopsis:

```
#include <unistd.h>
ssize_t write(int fildes,
const void* buf, size_t nbyte);
```

~ fildes → file to write
~ buff → pointer to buff
~ nbyte → number of bytes to write from buff.

* Successful: returns number of bytes actually written.
* Unsuccessful: returns -1 and set errno.

4.4 The select function:

→ The handling of I/O from multiple sources is an important problem that arise in many different forms.

→ One method of monitoring multiple file discriptors is to use a seperate process for each one.

→ Second method is using 'select' call provides a method of monitoring file descriptors from a single process

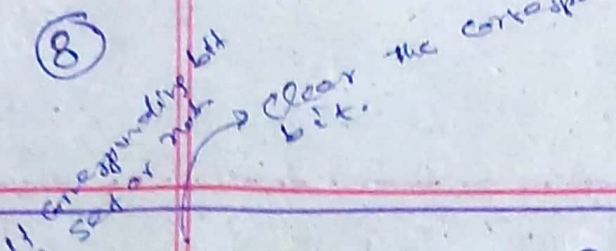
→ It can monitor for three possible conditions:
- a read without blocking
- a write without blocking
- file descriptor has error pending.

→ Synopsis: → old version uses time.h>

#include <sys/select.h> POSIX

int select(int nfds, fd_set *restrict readfs, fd_set *restrict writefs, fd_set *restrict errorfds, struct timeval *restrict timeout);

(8)



Void FD_CLR(int fd, fd_set *fdset);
int ~~void~~ FD_ISSET(int fd, fd_set *fdset);
Void FD_SET(int fd, fd_set *fdset);
Void FD_ZERO(~~int~~ fd_set *fdset);

~ nfds range of file descripton to be monitored. It must be atleast one greater than the largest file descriptor to be checked.
~ readfds set of descriptor to be monitored for reading.
~ writefds set of descriptor to b monitored for writing.
~ errorfds monitors for error conditions.
~ timeout forces a return after a certain period of time. has elapsed
→ Any of these parameter may be null.

* Successful: clears all the descriptors in each of readfds, writefds and errorfds. except those that are ready. returns the number of filedescriptor that is ready
* Unsuccessful: returns -1 and

set errno. Only unsuccessful when timout is passed else it remain block for infinite time.

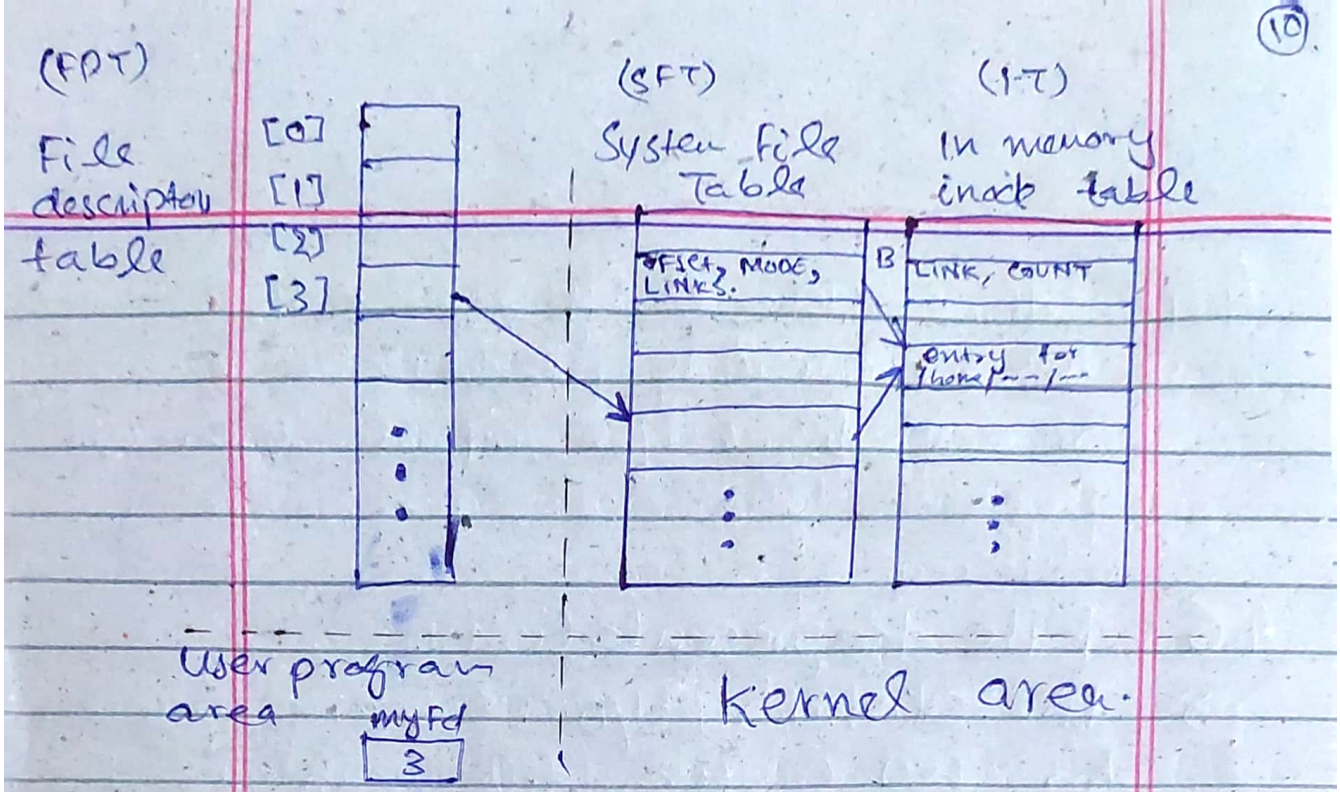
4.6 File Representation:

→ The standard I/O library functions for ISO C uses file pointers. The Unix I/O functions uses file descriptors.

→ Symbolic names for file pointers are 'stdin, 'stdout, 'stderr' defined in 'stdio.h'. Symbolic names for file descriptor are 'STDIN_FILENO', 'STDOUT_FILENO', and 'STDERR_FILENO' defined in 'unistd.h'.

→ A Library function is an ordinary function that is placed in a collection of functions called a library, usually because it is useful, widely used or part of a specification such as C.

→ A system call is a request to OS for service, It invalues a trap to OS and often a content switch.



→ The 'open' function creates an entry in the file descriptor table that points to an entry in System file table.

→ The system file table is shared by all process in system, has entry for each active open, Each SFT entry contains the file offset, an indication of access mode and a count of FDT entries pointing to it.

→ The Inmemory inode table contains an entry for each active file in system. Several SFT entries may point to same physical file in IT in memory.

→ The IT have also a count of SFT that are pointing to it. If count is zero the system deletes inode entry from memory.

→ When fork creates a child, the child inherits a copy of most of parent environment and content. i.e. including signal state, scheduling parameters & FDT. Thus share the same offset of files opened before fork.

4.7 Filters & Redirections:

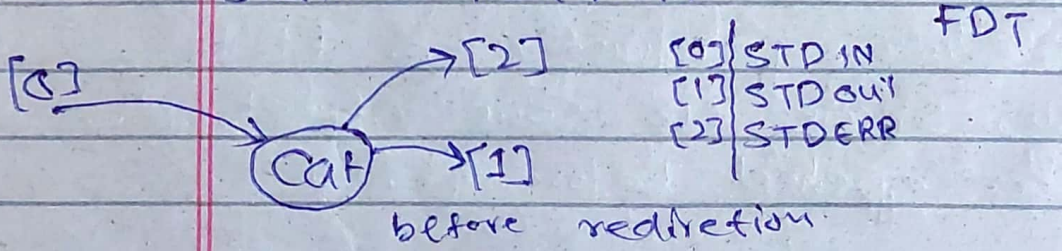
→ A filter reads from stdin, perform a transportation and output results to stdout.
eg; head, tail, more, sort, grep, and awk.

→ Redirection is a way to redirect input & output of command to or from a file or another command. It is a form of interprocess communication.

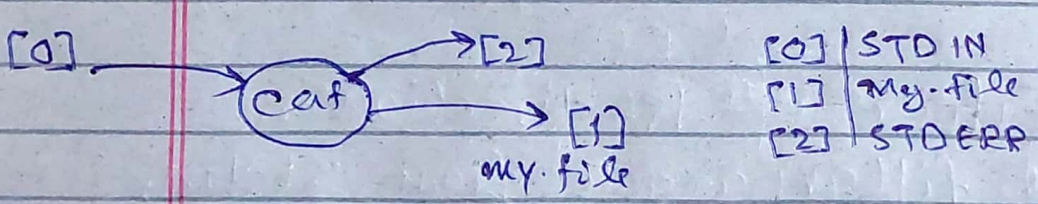
(12)

→ A program that can modify the FDT entry so that it points to a different entry in SFT. this action is known as redirection.

→ Most shells uses greater than (>) and CL^ as redirection of stdout and less than (<) for stdin redirection.



before redirection



after redirection

→ 'dup2' function is used to close 'filedes2' of FDT and then copy 'filedes1' into entry 'filedes2'

- Synopsis

```
#include <unistd.h>
int dup2(int filedes1, int filedes2);
```

~ filedes1 copies entry of filedes1 to fildes2.

~ filedes2 close this fole. and replace it with filedes1.

* Successful: returns fildescriptor value that was duplicated
* Unsuccessful; returns -1 and set errno.