Operating System Lab

Lab 2

Basic File Concepts

System Calls

- An application program can not access H/W directly .
- An application program can't do more privileged works (ex: Create a process).
- It says OS to do these works, using system calls.
- System calls are routine build into kernel.
- These calls are often written in assembly language.
- For assembly language programmers “Every system call has a number associated with it”.
- For C programmers there are C-like function interface.
- Many commands and system calls has same names (ex: chmod).

File Related System Calls

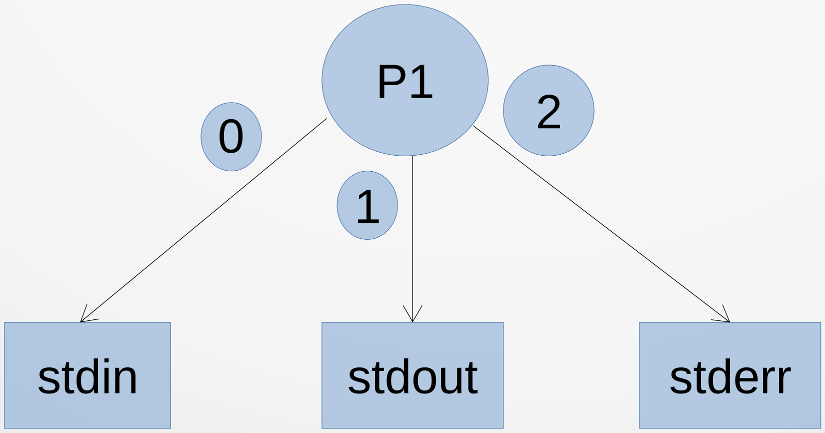
- open(): Open a file. Used by fopen().
- read(): read data from file. Used by fread(), fget(), fgetc(), scanf().
- write(): write data to the file. Used by fputc(),..., printf();
- close(): close file.
- lseek(): Moves file offset pointer to specified point.
- dup(): Duplicates file descriptor.

Types of file in Linux

- Ordinary file
  - Text file
  - Binary file
- Directory file: Contains file name and a number(INODE).
- Device file: Represent all devices as file.

Processes and Files

- Each process has three default opened files (stdin, stdout, stderr).
- Processes recognise all files with file descriptor.
- File descriptor is a number assigned for each opened file.
- Stdin(0), stdout(1), stderr(2)

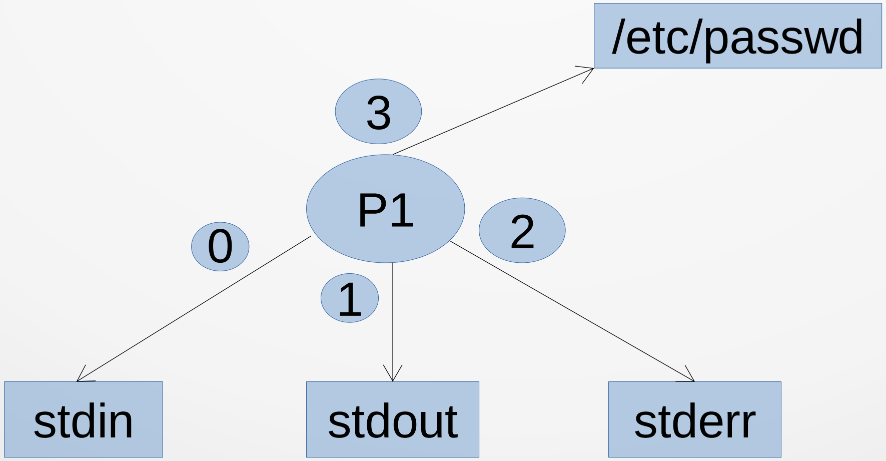


creat() System Call

- int creat(char *filename, mode_t mode)
- Return first unused file descriptor on success and return -1 when error.
- Modes: S_IRGRP, S_IROTH, S_IRUSR, S_IRWXG etc.

open() System Call

- int open(const char *path, int *oflag*, int *sflag*).
- Returns least available file descriptor of newly opened file.
- Int fd = open(“/etc/passwd”, O_RDONLY); // fd=3.
- The first open call sets *file offset pointer* to beginning of the file.
-
-
-



oflags

- These constants are defined in “*fcntl.h*”.
- O_RDONLY: Opens file for reading.
- O_WRONLY: Opens file for writing.
- O_RDWR: Opens file for reading and writing.
- O_APPEND: Opens file in append mode.
- O_TRUNC: Truncates file to zero length.
- O_CREAT: Create file if it doesn't exist.
- O_SYNC: Synchronizes read-write operations.

s_iflags (also for creat())

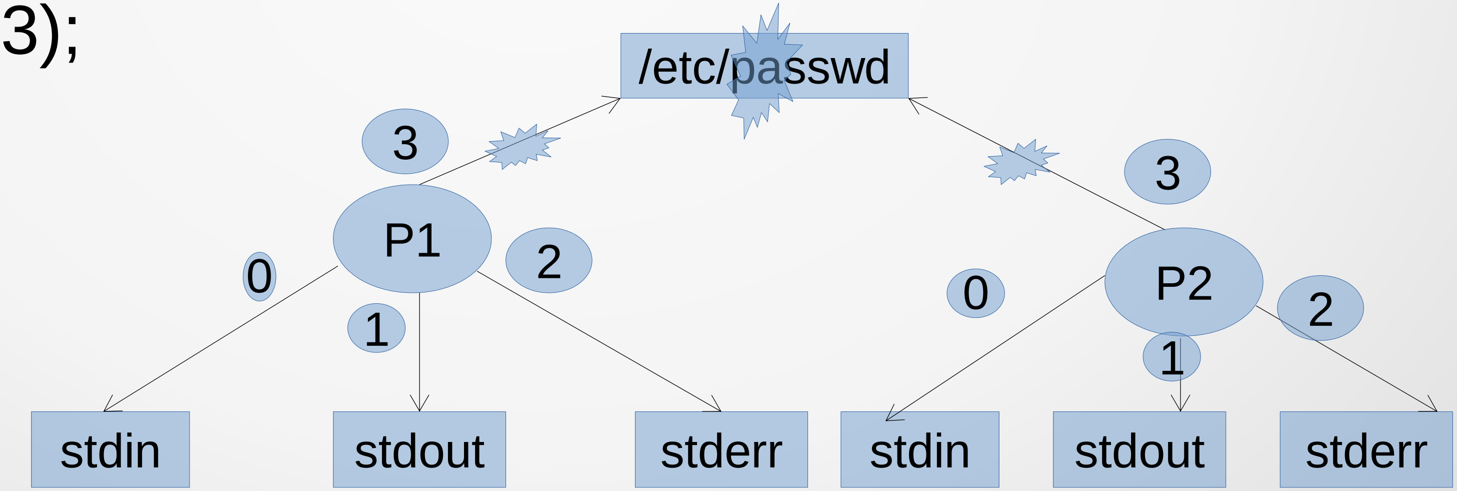
- S_IRUSR: Read permission for User.
- S_IWUSR: Write permission for User.
- S_IXUSR: Execute permission for User.
- S_IRGRP: Read permission for Group.
- S_IWGRP: Write permission for Group.
- S_IXGRP: Execute permission for Group.
- S_IROTH: Read permission for Other.
- S_IWOTH: Write permission for Other.
- S_IXOTH: Execute permission for Other.
- S_IRWXU: All permission to User.
- S_IRWXG: All permission to Group.
- S_IRWXO: All permission to Others.

How to use open()

```
fd2=open(“a.txt”, O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);
```

close() System Call

- int close(int fd);
- Deallocates the file descriptor. Cut its connection from file.
- But file is still open. May be used by some other process.
- Returns 0 if successful and -1 otherwise.
- P1:- close(3);



read() System Call

- int read(int fd, void *buf, int nbytes).
- fd is file descriptor. buf is Buffer Pointer. nbytes is size of Buffer.
- Read from file fd and store it in buffer buf.
- Returns number of bytes it reads, and set *file offset pointer* to next group of character.
- Returns -1 when it cannot read(file ends).

```
int fd,n;
char buf;
n=read(fd, &buf, 1)
```

```
int fd, n;
char buf[100];
n=read(fd, buf, 100);
```

write() System Call

- int write(int fd, void *buf, int nbytes).
- fd is file descriptor. buf is Buffer Pointer. nbytes is size of Buffer.
- Content of buf will be written on file fd.
- Returns number of bytes it writes.
- Returns -1 when it cannot write(file size exceeds system limit).

```
int fd;
char buf=‘a’;
write(fd, &buf, 1));
```

```
int fd;
char buf1[12] = "hello world";
write(fd, buf, 12));
```

lseek() System Call

- int lseek(int fd, int offset, int whence).
- fd is file descriptor. offset is distance to move.
- whence: From where to start.
  - SEEK_SET: Offset pointer set to beginning to file.
  - SEEK_END: Offset pointer set to end to file.
  - SEEK_CUR: Start from current position.
  - *+ve whence* means move towards end of file.
  - *-ve whence* means move towards start of file.
- Returns position of the pointer from beginning of the file.

lseek() Examples

lseek(fd, 10, SEEK_CUR);
Start from current position and go to 10 bytes towards end of file.

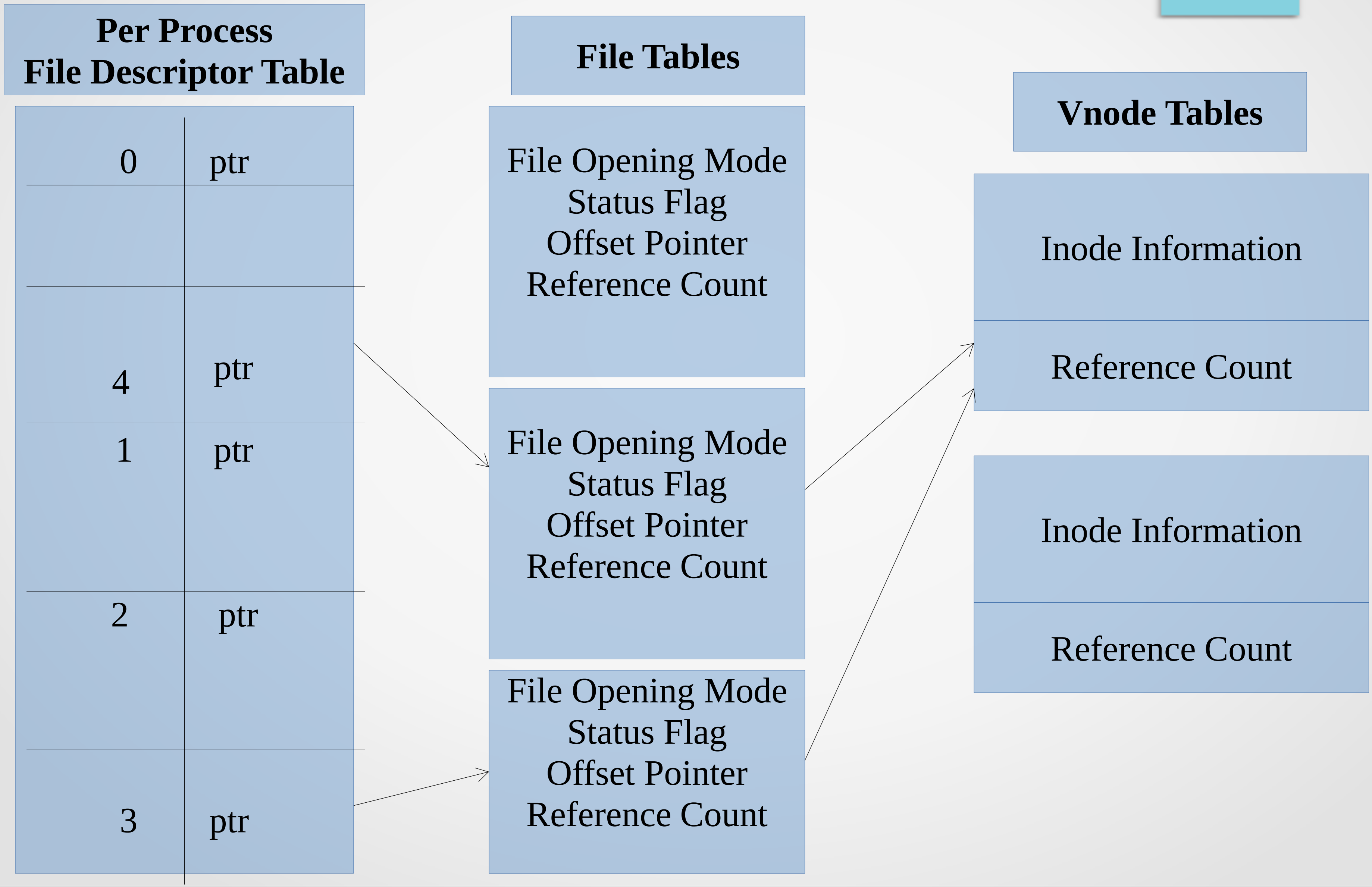
lseek(fd, -10, SEEK_END);
Start from end and go to 10 bytes towards beginning of file.

size = lseek(fd, 0, SEEK_END);
Returns size of the file in bytes.

lseek(fd, 10, SEEK_END);
Go beyond EOF, create a file with hole.
This is called sparse file

lseek(fd, -10, SEEK_SET);
ERROR...!!!!

Behind The Scene



Per Process File Descriptor Table

0	ptr
4	ptr
1	ptr
2	ptr
3	ptr

File Tables

File Opening Mode
Status Flag
Offset Pointer
Reference Count

File Opening Mode
Status Flag
Offset Pointer
Reference Count

File Opening Mode
Status Flag
Offset Pointer
Reference Count

Vnode Tables

Inode Information
Reference Count

Inode Information
Reference Count

dup System Call

- Duplicating the file-descriptor.
- Int dup(int fildes)
- Duplicates file descriptor to lowest possible descriptor.

```
fd=open(...)// fd=3
close(1);
dup(fd);// file connected to stdout(1) and fd=3 both
```

dup2 System Call

- Int dup2(int fildes, int fildes2)
- Replicate fildes to fildes2.
- Close fildes2 if it is already open.

```
fd=open(...)// fd=3
dup2(fd, 1);// file connected to stdout(1) and fd=3 both
```

*pipe(|)*In command

- who | wc -l
- Connect stdout of who to stdin of wc with a temporary file called pipe.

Task 1

Create a file manually, and write a program to read that file and print on screen character by character.

(Use system call only, Do not use “printf”)

Task 2

Repeat Task1 for group of 100 characters.

Task 3

Create a file manually, and write a program to read that file and print on another file character by character.

(Use system call only, Do not use “printf”)

Task 4

Repeat Task3 for group of 100 characters.

Task 5

Print a file in reverse order using lseek().

Task 6

Write a program in C, where two files are connected with stdin and stdout. Content of file with stdin should go to file connected with stdout.

Note: Do not use regular copy program. Do not use dup() or dup(2)

Task 8

Repeat Task 7 using dup()

Task 9

Repeat Task 7 using dup2()

Task 10

Write a C program to create 3 child processes which read from 3 different files and write to the same pipe in the parent process. Each child should wait a random amount of time (3 -10 seconds) between writing each 50 characters. The father should read from the pipe and write everything he gets (from all 3 files) into one new file.