System Programming

3. File IO (2): System Call

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng

Linux System Calls

- File descriptor I/O
  - ***open(); close(); creat(), read(); write();***
  - ***seek();*** *// random access*
  - ***fcntl();*** *// for file/record locking*
- Process control
- Thread programming
- IPC
- Signal handling
- Memory management
- *Synchronization*
- *Time management*
- *Network socket API (TCP, UDP)*

System Calls & Library Calls for File I/O

- System Calls for File descriptor I/O
  - ***open(); close(); creat(), read(); write();***
  - ***seek();*** *// random access*
  - ***fcntl();*** *// for file/record locking*

- Library Calls for File I/O
  - ***fopen(); freopen(); fclose(); fread(); fwrite();***
  - ***fgetc(), fgetchar(); fputc, putchar(); …***
  - ***fseek(), fprintf(); fscanf();..***

System Calls vs. Library Calls

- System Calls
  - they are entry points into kernel code where their functions are implemented.
  - documented in section 2 of the linux manual (e.g. `write`(2) or `man` 2 `write`)

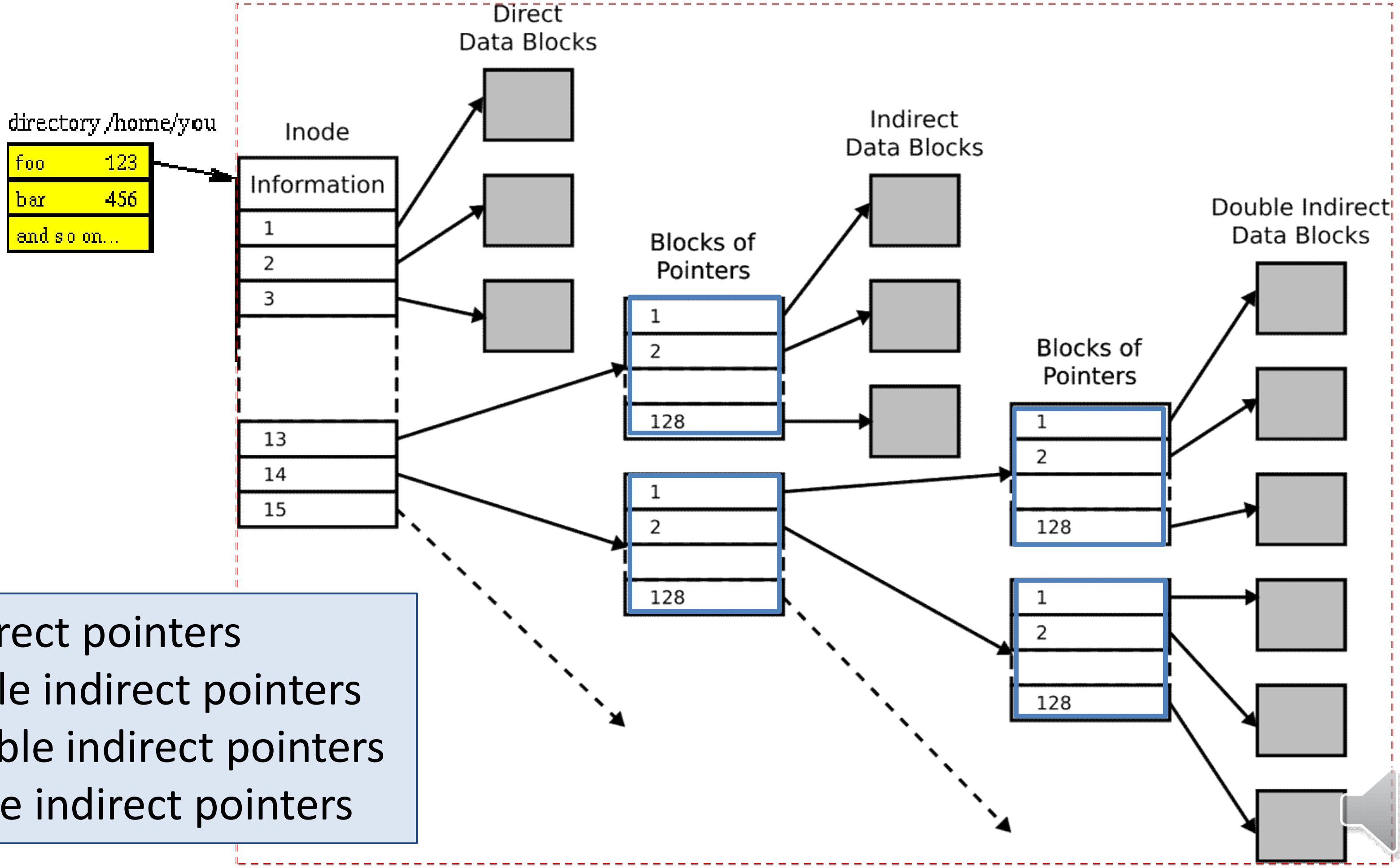
- Library Calls
  - they are transfers to user code which performs the desired functions.
  - documented in section 3 of the linux manual (e.g. `printf`(3)).
  - also called *API*(application programming interfece)

Linux File System (1)

- Each file in a file system has its own ***inode***
- An inode is a data structure having all information on a file.
- inodes of all files reside in a ***disk***
- inode contents (C struct)
  - file name
  - file type (regular, directory,...)
  - file owner id
  - access permission
    rwxr-xr-x (for owner, group, others)
  - creation/modified time
  - file size
  - file data block addr. table (see the next page!)
  - ...

Inode structure example

- block size: 512 byte, block pointer: 4 bytes



1-12: direct pointers
13: single indirect pointers
14: double indirect pointers
15: triple indirect pointers

Linux File System (2)

- File types
  - regular file
  - directory file
  - FIFO file (pipe)
  - special files (IO devices)
  - symbolic link files

A “john” directory file

i	. (john)
j	.. (parent)
k	File name A
l	File name B
m	File name C

- Directory file
  - A directory is just a file whose content is the list of (inode #, file name) in the same directory.
  - inode is a data structure which contains all the information about the file and file data blocks
  - inode # is a unique file id number in the file system
  - “`ls -al john`” is a shell command that just displays the “john” directory file

Linux File Types

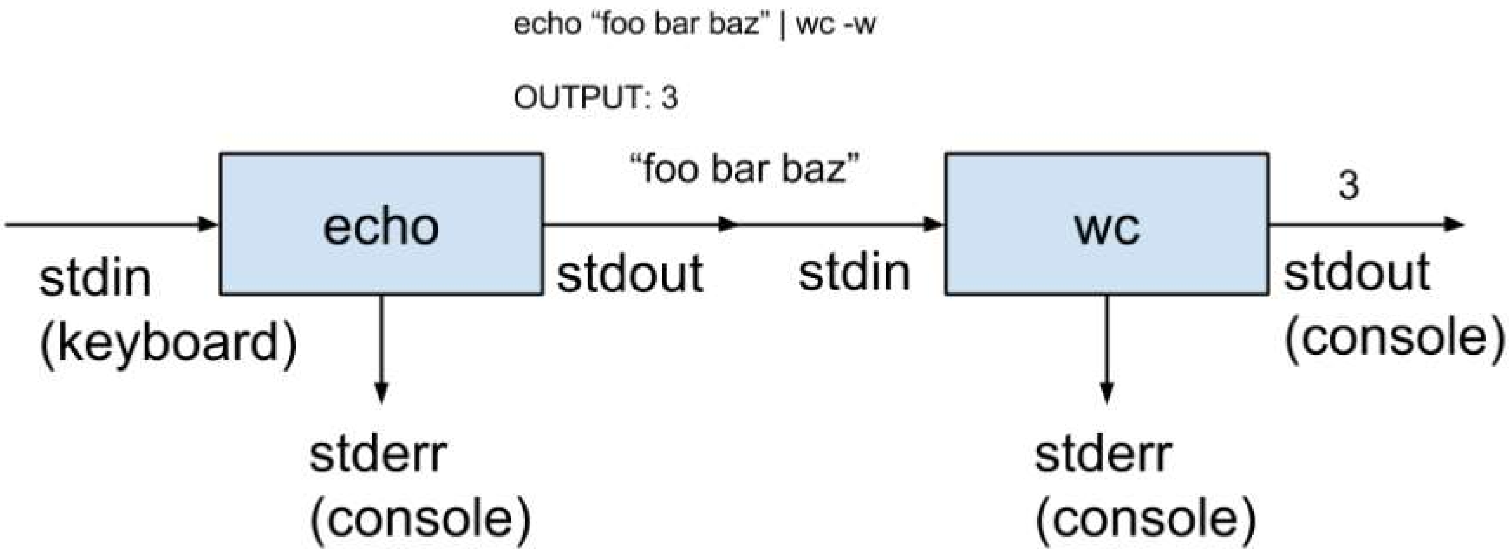
- Ordinary File (Regular File)
  - Text, binary files
- Directory File
  - A file that includes the set of (*file-name, inode #*) of the directory.
- Character Special File
  - Character-oriented device (e.g. Keyboard)
- Block Special File
  - Block-oriented device (e.g. HDD file systems, eth0)
- FIFO file
  - Named *pipe / Unnamed pipe*

  cf. pipe in a process is usually unnamed.
- Symbolic link file
  - a file which points to another file

  cf. *hardlink* is NOT a file.

File Descriptor (1)

- A *file descriptor* (or *file handle*) is a small, non-negative integer which identifies a file to the kernel.
  - Traditionally, `stdin`, `stdout` and `stderr` are 0, 1 and 2 respectively.



```
echo "foo bar baz" | wc -w

OUTPUT: 3
```

  - Relying on "magic numbers" is BAD.
    - Use `STDIN_FILENO`, `STDOUT_FILENO` and `STDERR_FILENO` defined in or stdin, stdout, and stderr defined in .

File Descriptor (2)

- Maximum number of files
  - a process can open 1024 files
  - we can check the system resource configuration

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 194273
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
.......
```

Basic File I/Os

- 5 fundamental Unix/Linux file I/Os
  - `open(2)`
  - `close(2)`
  - `lseek(2)`
  - `read(2)`
  - `write(2)`

File open (1)

```
#include <fcntl.h>

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

- parameters
  - *path*: name of the file to open or create
  - *oflag*: file open options
  - *mode*: access permission (at file creation)
- return
  - *file descriptor* if OK
  - 1 on error

File open (2)

- *oflag* options
  - must be one of these

option1	meaning	<fcntl.h> defined
O_RDONLY	open for reading only	0
O_WRONLY	open for writing only	1
O_RDWR	open for reading & writing	2

  - and can be OR’ed with any of these (by “|”)

option2	meaning
O_CREAT	create a file if the file does not exist.
O_EXCL	used with O_CREAT, return an error if the file already exists.
O_TRUNC	if the file exists, make it empty.
O_APPEND	write from the end of the file.
O_SYNC	do disk synchronization when does file I/O.

File access modes

mode	meaning
S_ISUID	set-user-id at execution
S_ISGID	set-group-id at execution
S_ISVTX	set sticky bit
S_IRWXU	owner RWX
S_IRUSR	owner R
S_IWUSR	owner W
S_IXUSR	owner X
S_IRWXG	group RWX
S_IRGRP	group R
S_IWGRP	group W
S_IXGRP	group X
S_IRWXO	others RWX
S_IROTH	others R
S_IWOTH	others W
S_IXOTH	others X

File close

```
#include <unistd.h>

int close(int fd);
```

- parameters
  - *fd*: file descriptor
- return
  - 0 if OK
  - -1 on error

File open example

open-ex.c

```c
#include <fcntl.h>

int main(int argc, char * argv[])
{
        FILE *fpo;  // file pointer
        int fdo;    // file descriptor

        if(argc != 2) {
                perror(argv[0]);
                return 1;
        }
        if((fdo = open(argv[1], O_RDWR | O_CREAT | O_TRUNC,
                                S_IRUSR | S_IWUSR)) == -1) {
                perror(argv[1]);
                return 1;
        }
        if((fpo = fdopen(fdo, "r+")) == NULL) {
                perror("fdopen");
                return 2;
        }
        fprintf(fpo, "Hello, world! \n");
        fclose(fpo);
}
```

```
$./a.out test.txt
$ cat test.txt
Hello, world!
$
```

File creation

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

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

- parameters
  - *path* : file path name
  - *mode* : access permission
- return
  - *file descriptor* if OK
  - -1 on error

File seeking

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

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

- parameters
  - *fd* : file descriptor
  - *offset* : offset from the beginning to seek (move)
  - *whence* : SEEK_SET, SEEK_CUR, SEEK_END
- return
  - new offset value if OK
  - -1 on error

File reading

```
#include <unistd.h>

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

- parameters
  - *fd* : file descriptor
  - *buf* : buffer address
  - *nbyte* : number of bytes to read
- return
  - number of bytes read successfully if OK
  - -1 on error

File writing

```
#include <unistd.h>

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

- parameters
  - *fd* : file descriptor
  - *buf* : buffer address
  - *nbyte* : number of bytes to write
- return
  - number of bytes written successfully if OK
  - -1 on error

File create/lseek example

create-ex.c

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

int main(void)
{         int fd;
          char buf1[] = "Test1 data";
          char buf2[] = "Test2 data";

          if ((fd == creat ("test.txt", S_IRUSR | S_IWUSR | S_IRGRP |
                    S_IROTH)) < 0) {
                    printf("creat error");
                    return 1;
          }
          write(fd, buf1, 10);
          if(lseek(fd, 6L, SEEK_SET) == -1) {
                    printf("lseek error");
                    return 2
          }
          write(fd, buf2, 10);

          return 0;
}
```

```
$ ls
a.out   test.c
$ ./a.out
$ ls
a.out    test.c test.txt
$ cat  test.txt
Test1 Test2 data
$
```

File copy example (1)

fcopy2-ex.c

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

#define BUFFER_SIZE   1024

int main(int argc, char *argv[])
{
        int fdi, fdo;
        char buf[BUFFER_SIZE];
        ssize_t n;

        if(argc != 3) {
                perror(argv[0]);
                return 1;
        }
```

File copy example (2)

fcopy2-ex.c

```
        if((fdi = open(argv[1], O_RDONLY)) == -1) {
                perror(argv[1]);
                return 2;
        }

        if((fdo = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC |
                            O_EXCL, S_IRUSR | S_IWUSR)) == -1) {
                perror(argv[2]);
                return 3;
        }
        while((n = read(fdi, buf, BUFFER_SIZE)) > 0)
                write(fdo, buf, n);

        close(fdi);
        close(fdo);
        return 0;
}
```