System Programming

Week 3: File I/O

Seongjin Lee

Updated: 2016-09-21

02-fileio

insight@hanyang.ac.kr

http://esos.hanyang.ac.kr

Esos Lab. Hanyang University

Table of contents

File I/Os

File Descriptors

A *file descriptor* (or *file handle*) is a small, non-negative integer which identifies a file to the kernel.

File Descriptors

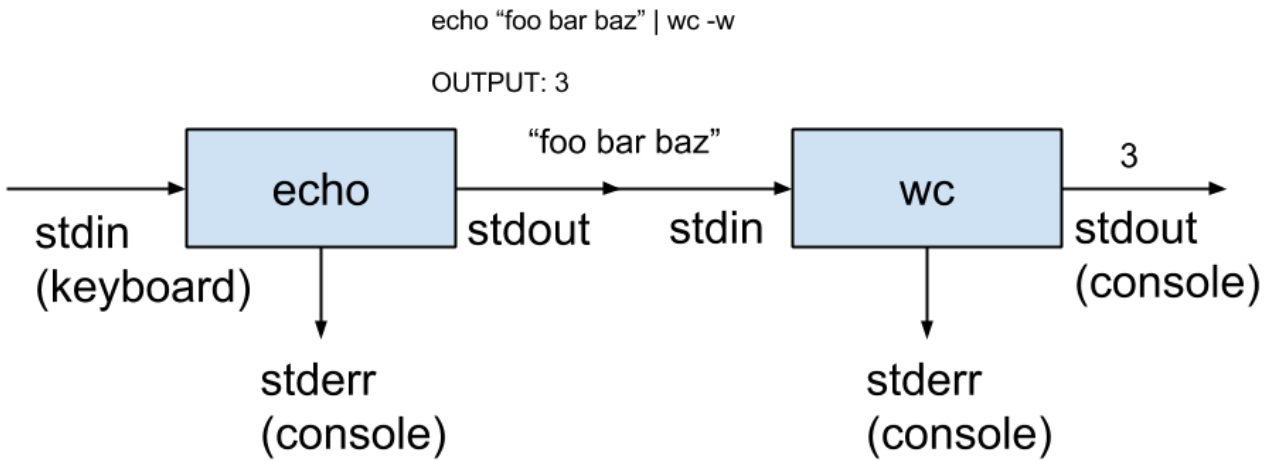
A *file descriptor* (or *file handle*) is a small, non-negative integer which identifies a file to the kernel.

Traditionally, `stdin`, `stdout` and `stderr` are 0, 1 and 2 respectively.

File Descriptors

A *file descriptor* (or *file handle*) is a small, non-negative integer which identifies a file to the kernel.

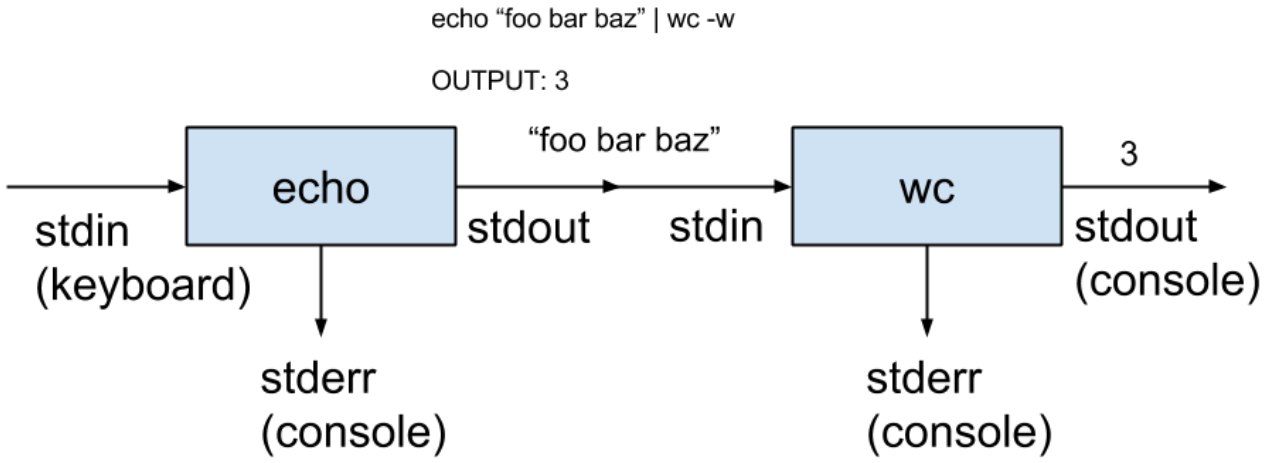
Traditionally, `stdin`, `stdout` and `stderr` are 0, 1 and 2 respectively.



File Descriptors

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.



Relying on “magic numbers” is BAD. Use `STDIN_FILENO`, `STDOUT_FILENO` and `STDERR_FILENO` defined in <unistd.h> or `stdin`, `stdout`, and `stderr` defined in <stdio.h>.

How many files can you open? (./codes/fdcount.c)

```
1  long open_max(void)
2  {
3      if (openmax == 0) {
4          errno = 0;
5      /* first time through */
6          if ((openmax = sysconf(_SC_OPEN_MAX)) < 0) {
7              if (errno == 0)
8                  openmax = OPEN_MAX_GUESS; /* it is indeterminate */
9              else
10                 fprintf(stderr, "sysconf error for _SC_OPEN_MAX");
11         }
12     }
13     return(openmax);
14 }
15
16 int main(){
17    printf("The number of file decriptors a process can open: %d\n", (int)RLIMIT_
          NOFILE);
18    printf("The number of file decriptors an user can open: %ld\n", openmax );
19    return 0;
20 }
```

How to compile

```
$ cd codes; cc -Wall -g -o fdcount fdcount.c
```

File I/Os : Standard I/O

Basic File I/Os

There are five fundamental Unix file I/O related functions:

- `open(2)`
- `close(2)`
- `lseek(2)`
- `read(2)`
- `write(2)`

open(2)

```
#include <fcntl.h>
int open(const char *path, int oflag, ... /* mode_t mode */ );
int openat(int fd, const char *path, int oflag, ... /* mode_t mode */ );
```

Both return: file descriptor if OK, 1 on error

The *path* parameter is the name of the file to open or create.

Options are specified by the *oflag*

Options are

oflag must be one (and only one) of:

- `O_RDONLY` – Open for reading only
- `O_WRONLY` – Open for writing only
- `O_RDWR` – Open for reading and writing

and may be OR'd with any of these:

- `O_APPEND` – Append to end of file for each write
- `O_CREAT` – Create the file if it doesn't exist. Requires *mode* argument
- `O_EXCL` – Generate error if `O_CREAT` and file already exists. (atomic)
- `O_TRUNC` – If file exists and successfully open in `O_WRONLY` or `O_RDWR`, make length = 0
- `O_NOCTTY` – If pathname refers to a terminal device, do not allocate the device as a controlling terminal
- `O_NONBLOCK` – If pathname refers to a FIFO, block special, or char special, set nonblocking mode (open and I/O)
- `O_SYNC` – Each write waits for physical I/O to complete

openat(2)

`openat(2)` function is equivalent to the open() function except in the case where the path specifies a relative path in an atomic fashion

- `O_EXEC` – Open for execute only
- `O_SEARCH` – Open for search only (applies to directories)
- `O_DIRECTORY` – If path resolves to a non-directory file, fail and set errno to `ENOTDIR`.
- `O_DSYNC` – Wait for physical I/O for data, except file attributes
- `O_RSYNC` – Block read operations on any pending writes.
- `O_PATH` – Obtain a file descriptor purely for fd-level operations. (Linux >2.6.36 only)

File Sharing

File Sharing

Atomicity of the file fundamental file I/O functions

File sharing

manipulation of file descriptors