System Programming

by Mohamad H. Danesh

I/O Routines

- Working with files
  - Pointer-base IO
    - Ex. fprintf(FILE * ….)
  - File descriptor-based IO
    - Low-Level, in fact standard C library I/O routines are implemented on top of the Linux low-level I/O system calls

File Descriptors

- A file descriptor is simply an integer that is used as an index into a table of open files associated with each process.
- The values 0, 1, and 2 are special and refer to the stdin, stdout, and stderr streams;

Read Data

- Note:
  - Include the header files <fcntl.h>, <sys/types.h>, <sys/stat.h>, and <unistd.h> if you use any of the low-level I/O functions
- The open() Call

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags).
int open(const char *pathname, int flags, mode_t mode);
```

TABLE 9.1 FLAGS FOR THE `open()` CALL

Flag	*Description*
`O_RDONLY`	Open file for read-only access.
`O_WRONLY`	Open file for write-only access.
`O_RDWR`	Open file for read and write access.
`O_CREAT`	Create the file if it does not exist.
`O_EXCL`	Fail if the file already exists.
`O_NOCTTY`	Don't become controlling `tty` if opening `tty` and the process had no controlling `tty`.
`O_TRUNC`	Truncate the file to length 0 if it exists.
`O_APPEND`	Append file pointer will be positioned at end of file.
`O_NONBLOCK`	If an operation cannot complete without delay, return before completing the operation. (See Chapter 22, "Non-blocking Socket I/O.")
`O_NODELAY`	Same as `O_NONBLOCK`.
`O_SYNC`	Operations will not return until the data has been physically written to the disk or other device.

- The close() Call

  #include <unistd.h>

  int close(int fd);

  Any locks held by the process on the file are released
- The read() Call

  #include <unistd.h>

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

- The write() Call

  #include <unistd.h>

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

- The flock() Call

  #include <sys/file.h>

  int flock(int fd, int operation)

  apply or remove an advisory lock on an open file
- operation, will be
  - LOCK_SH for a shared lock,
  - LOCK_EX for an exclusive lock,
  - LOCK_UN to unlock;

Exercise

- Write a program that writes “hello world!” into a file using low level I/O. Ensure proper error handling is performed on the open() and write() function calls.