File Management

Application Programming vs System Programming

Application Programming

- High level functions
- Using standard C library
- Man page level 3
- Libraries maintains buffers to read or write data from system calls. So libraries also called Buffered I/O.
- Libraries doesn't create performance penalty.

System Programming

- low level function
- Using System calls.
- Man page level 2
- System calls doesn't have buffers.
- System calls create performance penalty.

File System

- The file system
  - manages files,
  - allocating file space,
  - administrating free space,
  - controlling access to files
  - and retrieving data for user.
- The internal representation of file is given an ***inode table***.

OS vs File System

DOS: FAT32 (**F**ile **A**llocation **T**able)

Windows: FAT, NTFS (**N**ew **T**echnology **F**ile **S**ystem)

LINUX: ext2, ext3 and ext4 (Extended file system), JFS (Journaling file system), btrfs (b-tree File System)

File System Layout

Boot Block: contain bootstrap code that is read into machine to boot, or initialize, the operating system.

Super Block: Describes the state of a file system – How large it is, how many files it can store, where to find free space on the file system.

Inode (index node)list: inode represents the type of the file.

Data Block: contain file data and administrative data.

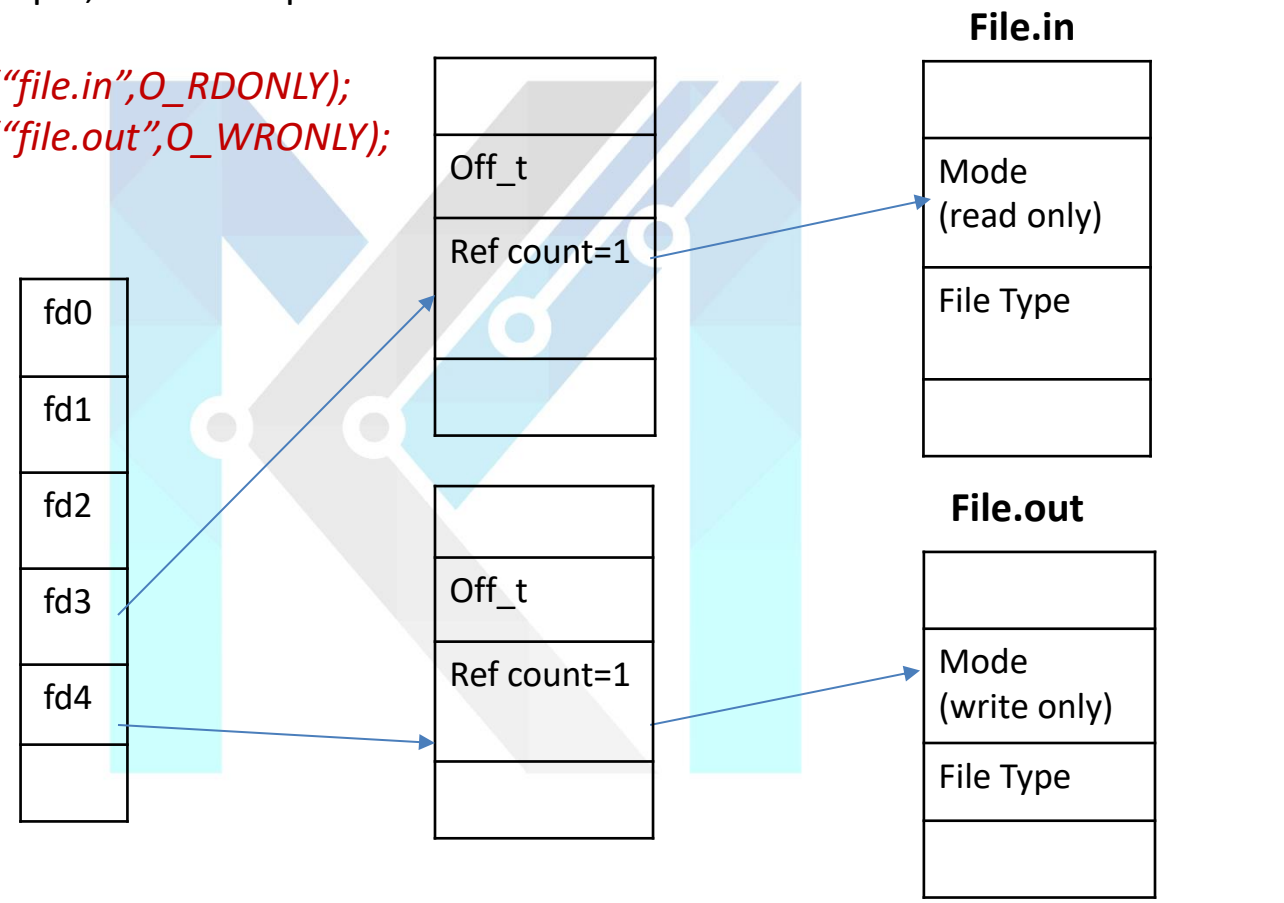
File System - Kernel Data Structures

- ***User File Descriptor table:*** Each process has its own separate *descriptor table* whose entries are indexed by the process's open file descriptors. Each open descriptor entry points to an entry in the *file table.*
- ***File table:*** Each file table entry consists of (for our purposes) the current file position, a *reference count* of the number of descriptor entries that currently point to it, and a pointer to an entry in the *inode table*.
- ***Inode table:*** Each entry contains most of the information in the stat structure, including the st_mode and st_size members.
- ***File table*** is global kernel structure where as ***user file descriptor table*** is allocated per process.

File System - Kernel Data Structures

In this example, two descriptors reference distinct files. **There is no sharing.**

fd3=open("file.in",O_RDONLY);
fd4=open("file.out",O_WRONLY);



User File descriptor Table
(one table per process)

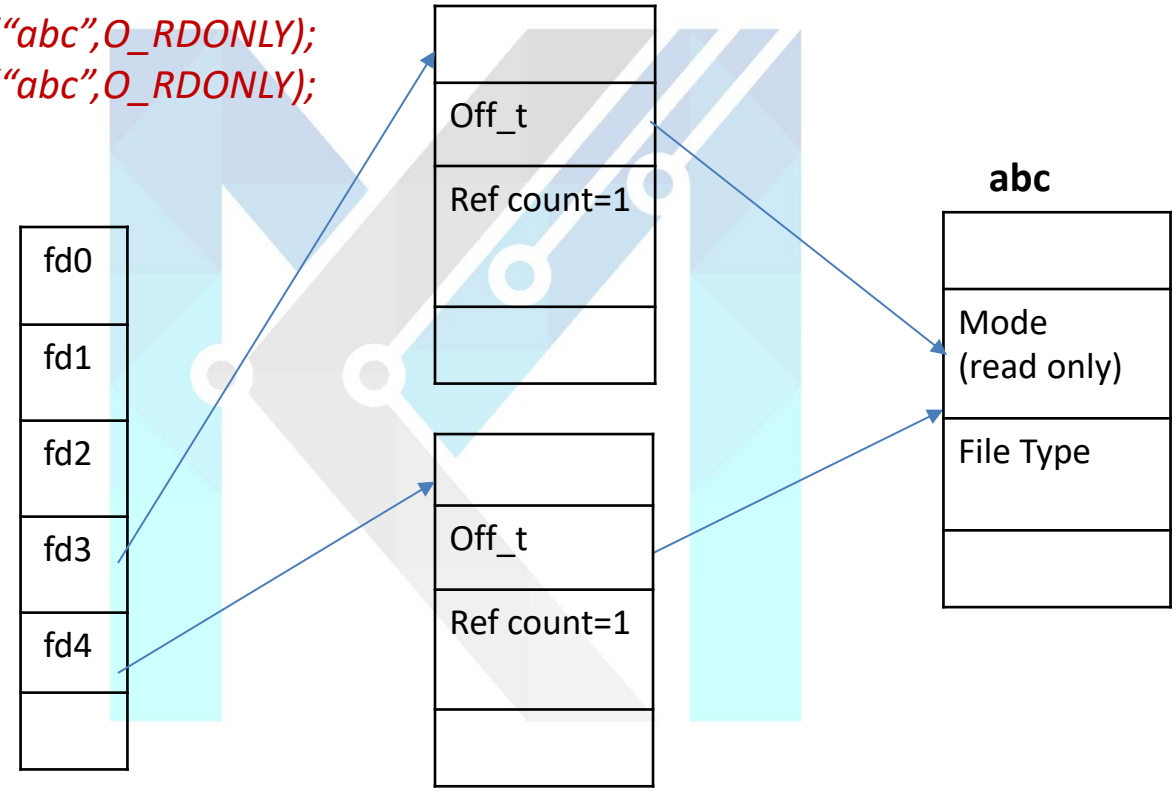
File Table
(shared by all process)

inode Table
(shared by all process)

File System - Kernel Data Structures

This example shows two descriptors **sharing the same disk file** through two open file table entries

fd3=open("abc",O_RDONLY);
fd4=open("abc",O_RDONLY);



User File descriptor Table
(one table per process)

File Table
(shared by all process)

inode Table
(shared by all process)

File System - Kernel Data Structures

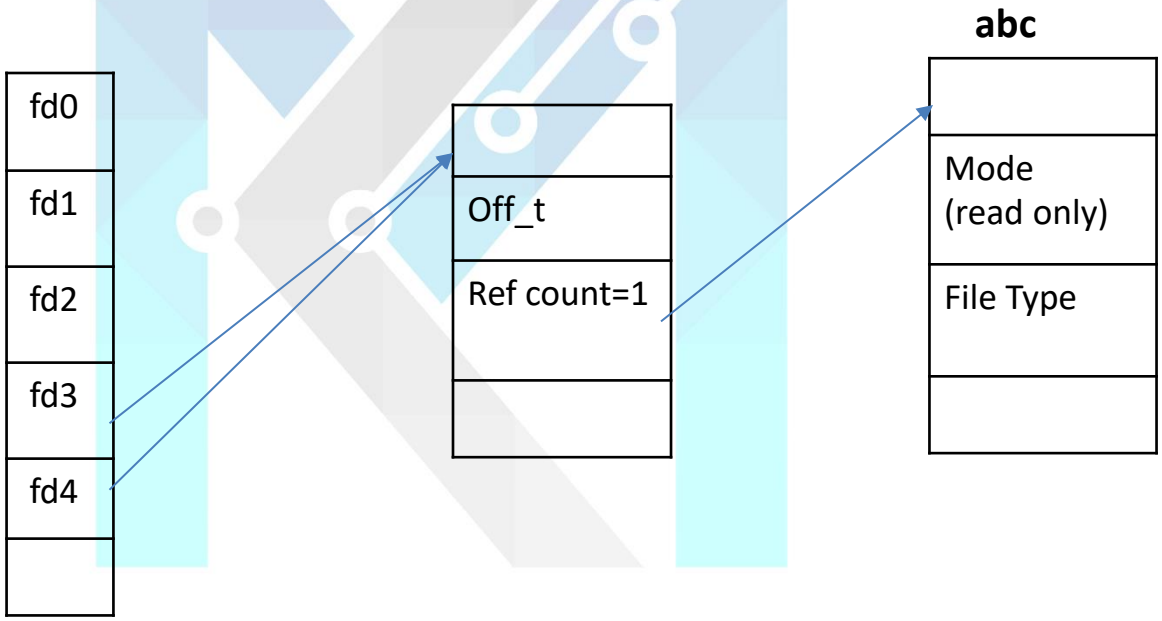
dup() Example:

dup(int oldfd)

fd=open("abc",O_RDONLY);

Close(fd);

dup_fd=dup(fd);

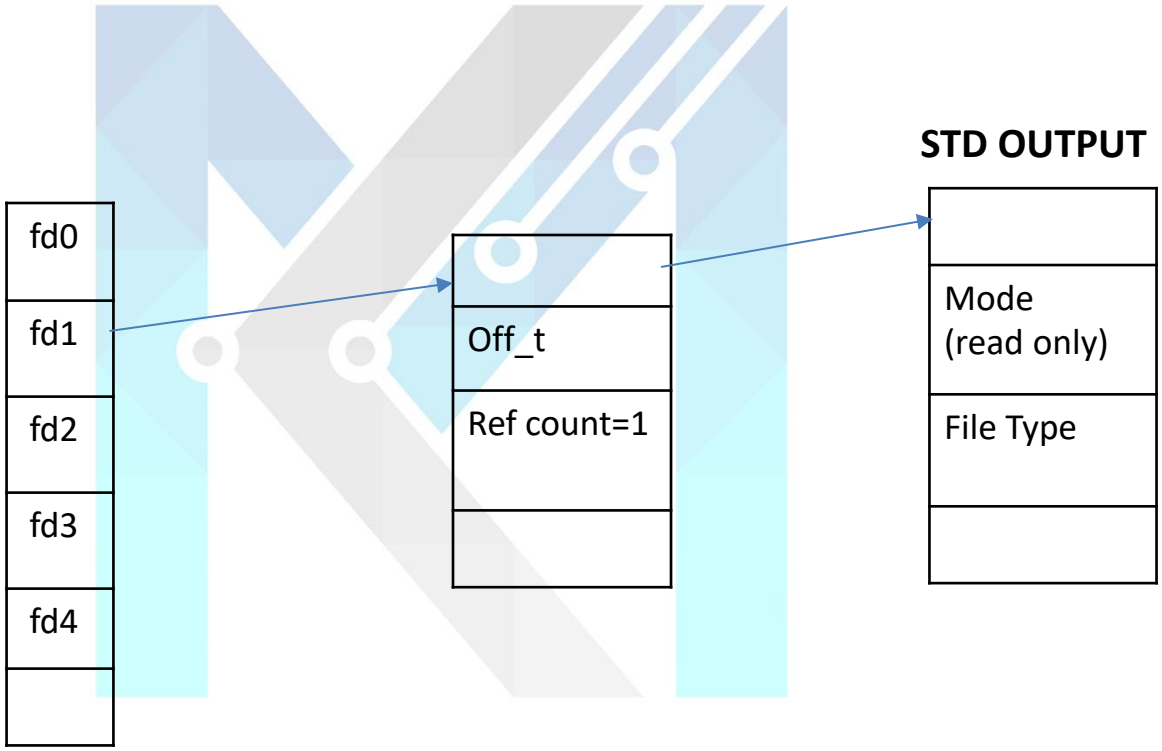


User File descriptor Table
(one table per process)

File Table
(shared by all process)

inode Table
(shared by all process)

ls command o/p: STD OUT



User File descriptor Table
(one table per process)

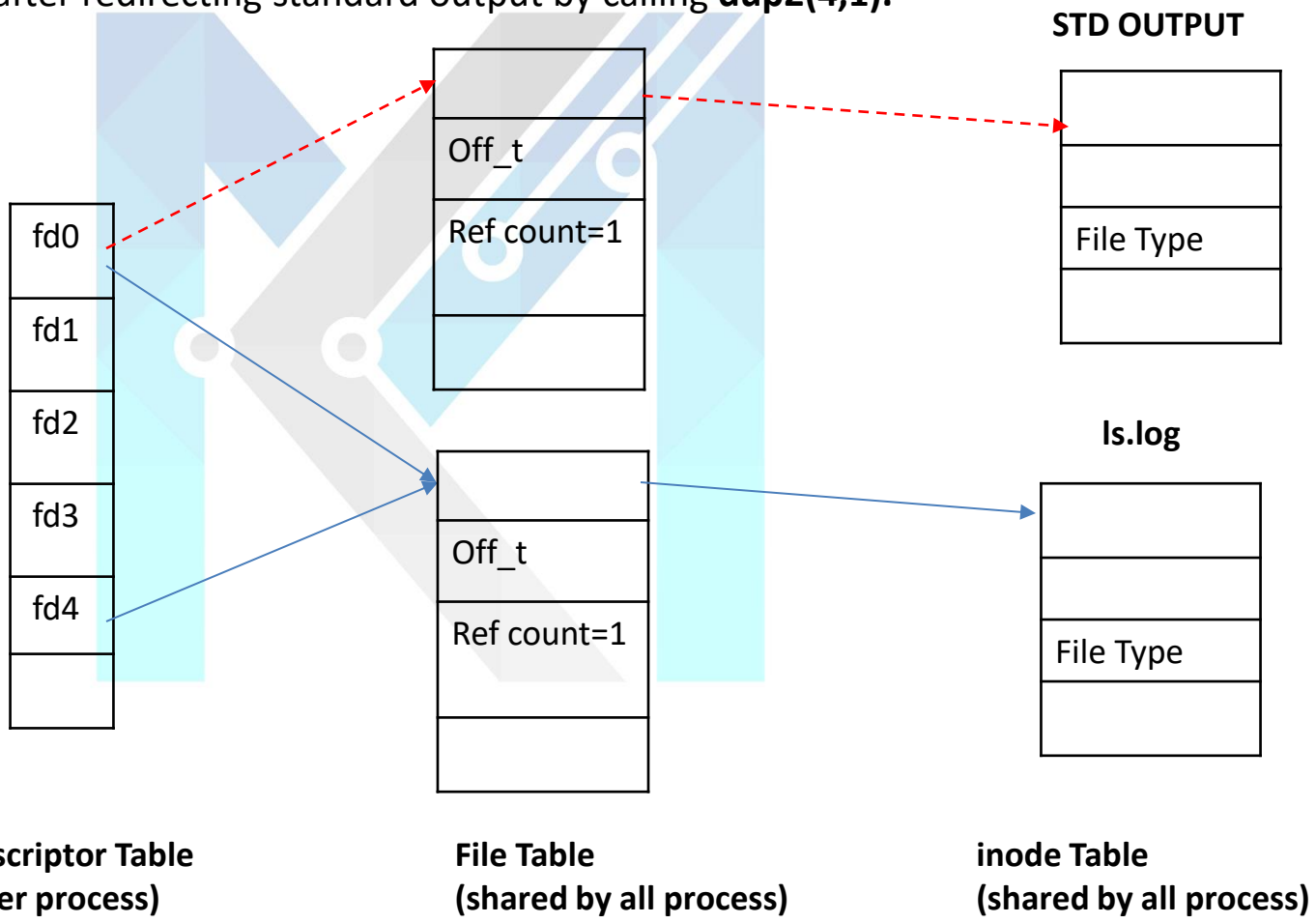
File Table
(shared by all process)

inode Table
(shared by all process)

I/O Redirection – ls > ls.log ?

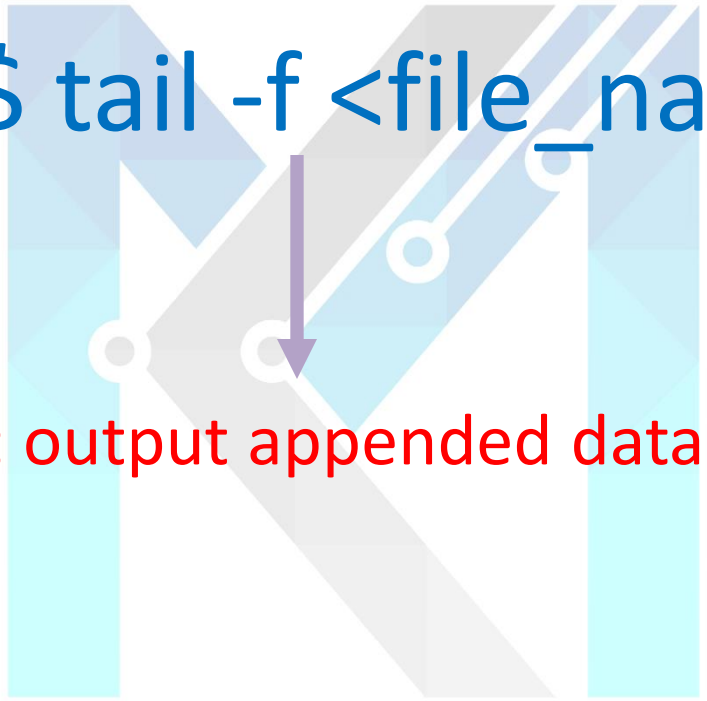
dup2(int oldfd, int newfd) Example:

Kernel data structures after redirecting standard output by calling **dup2(4,1).**



Monitoring File System Events

$ tail -f <file_name>

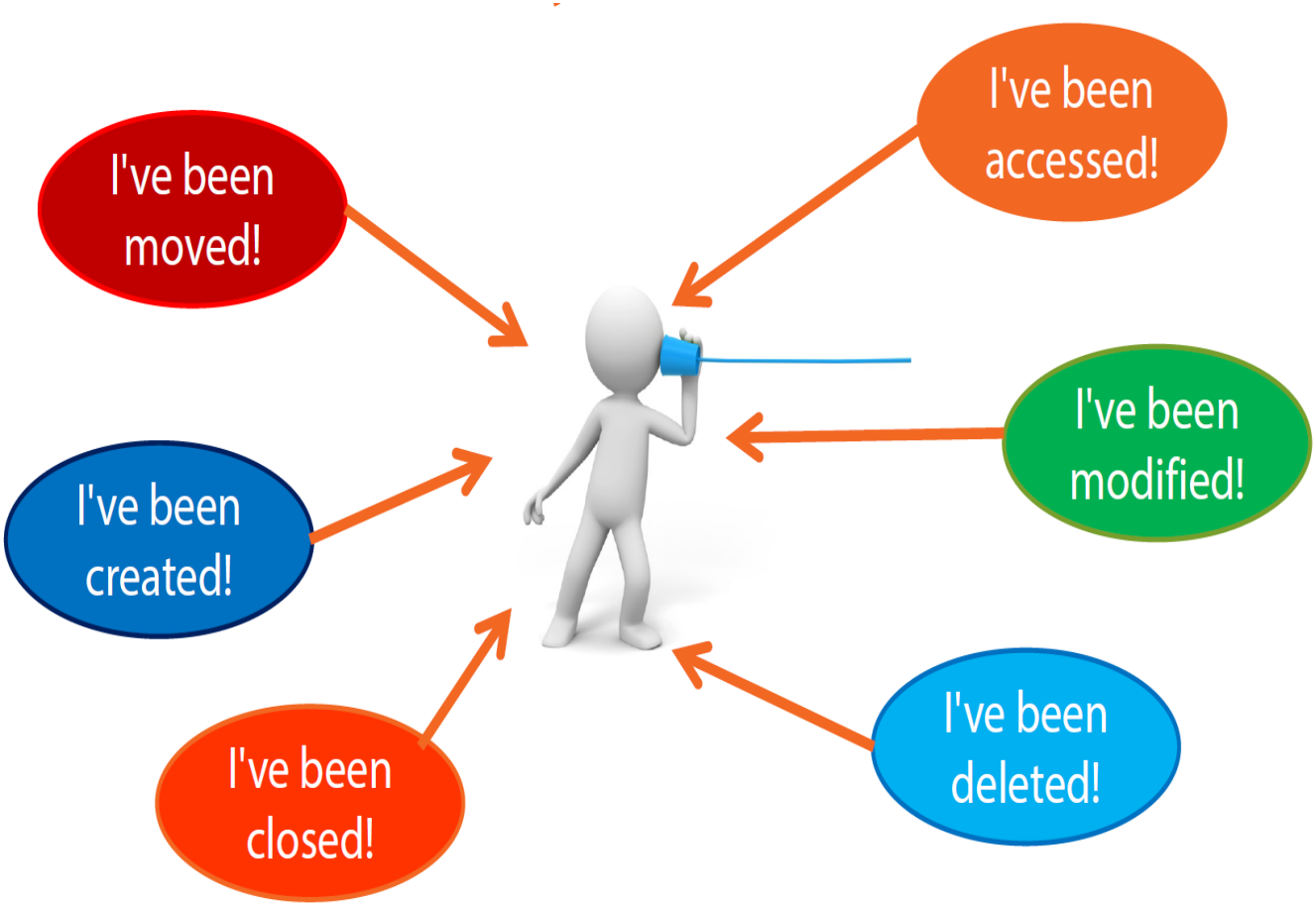


-follow: output appended data as the file grows.

Monitoring File System Events

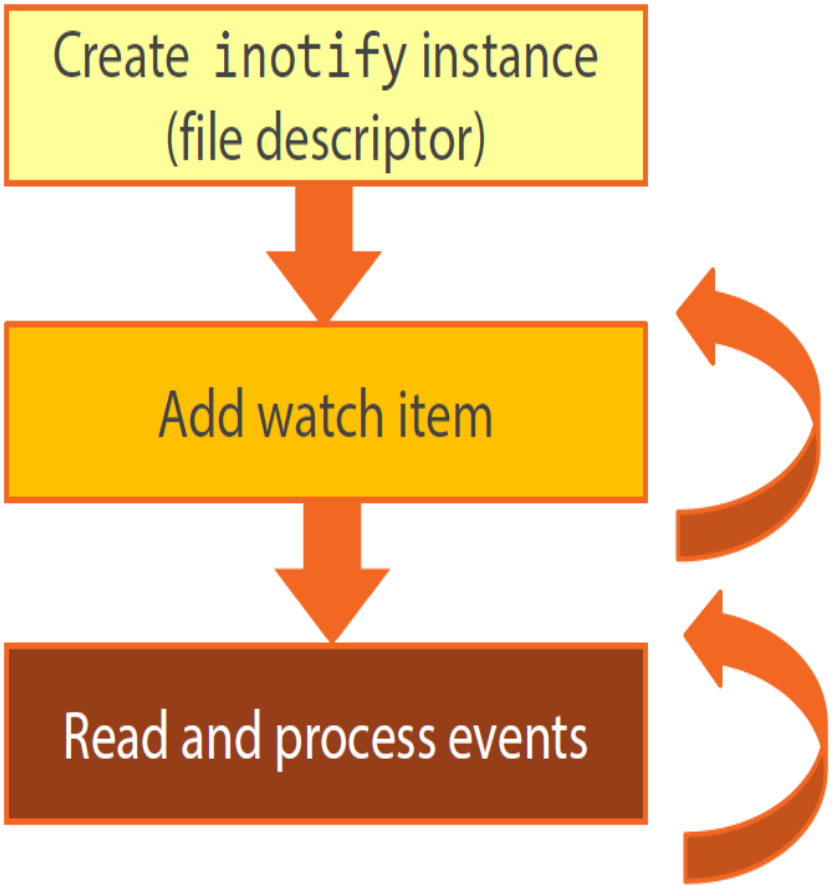
- The inotifyAPI
- Creating an inotifyinstance
- Adding to the watch list
- Reading events

Monitoring File System Events



Individual files or whole directories can be watched

Three Steps

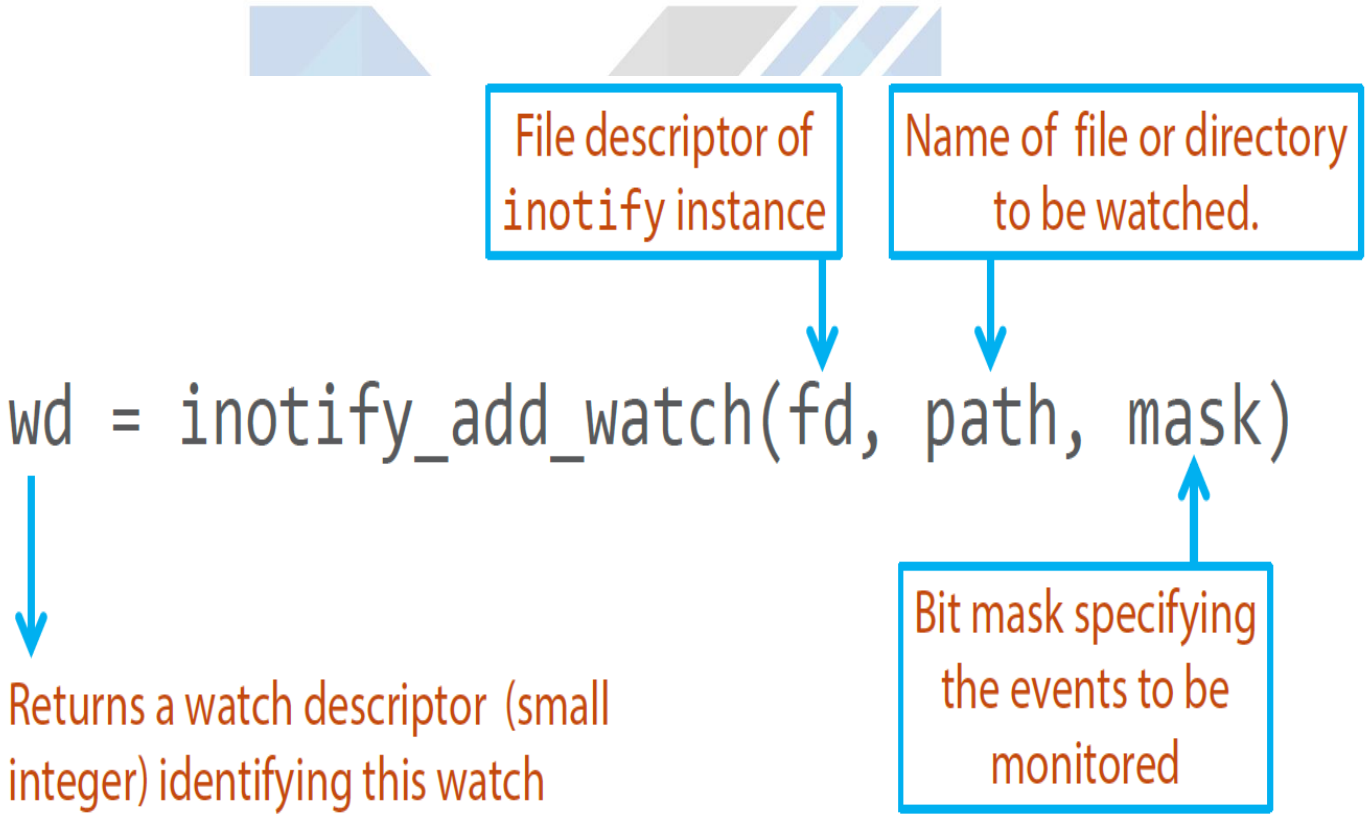


Creating an inotify Instance

```
fd = inotify_init()
```

Returns a file descriptor on whichwe can later read() the events.

Adding a watch Item



Watches and Events in Detail

Each event is specified by a single-bit constant
-- bitwise OR them together

Bit value	Meaning
IN_ACCESS	File was accessed
IN_ATTRIB	File attributes changed (ownership, permissions etc.)
IN_CREAT	File created inside watched directory
IN_DELETE	File deleted inside watched directory
IN_DELETE_SELF	Watched file deleted
IN_MODIFY	File was modified
IN_MOVE_SELF	File was moved

Reading Events