Unix System Programming

File I/O

Chandravva Hebbi
Department of Computer Science and Engineering
chandravvahebbi@pes.edu

Unix System Programming

File I/O

Chandravva Hebbi

Department of Computer Science and Engineering

UNIX SYSTEM PROGRAMMING

Topics to be Covered

- Read and write sysem calls.
- Programming Examples on Basic system calls.

Read sysetm call

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

- read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

- If count is zero, read() returns zero
- If count is greater than SSIZE_MAX, the result is unspecified.
- On error, -1 is returned. Errno is set, check the error no.
- Error: EBADF

Write system call

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

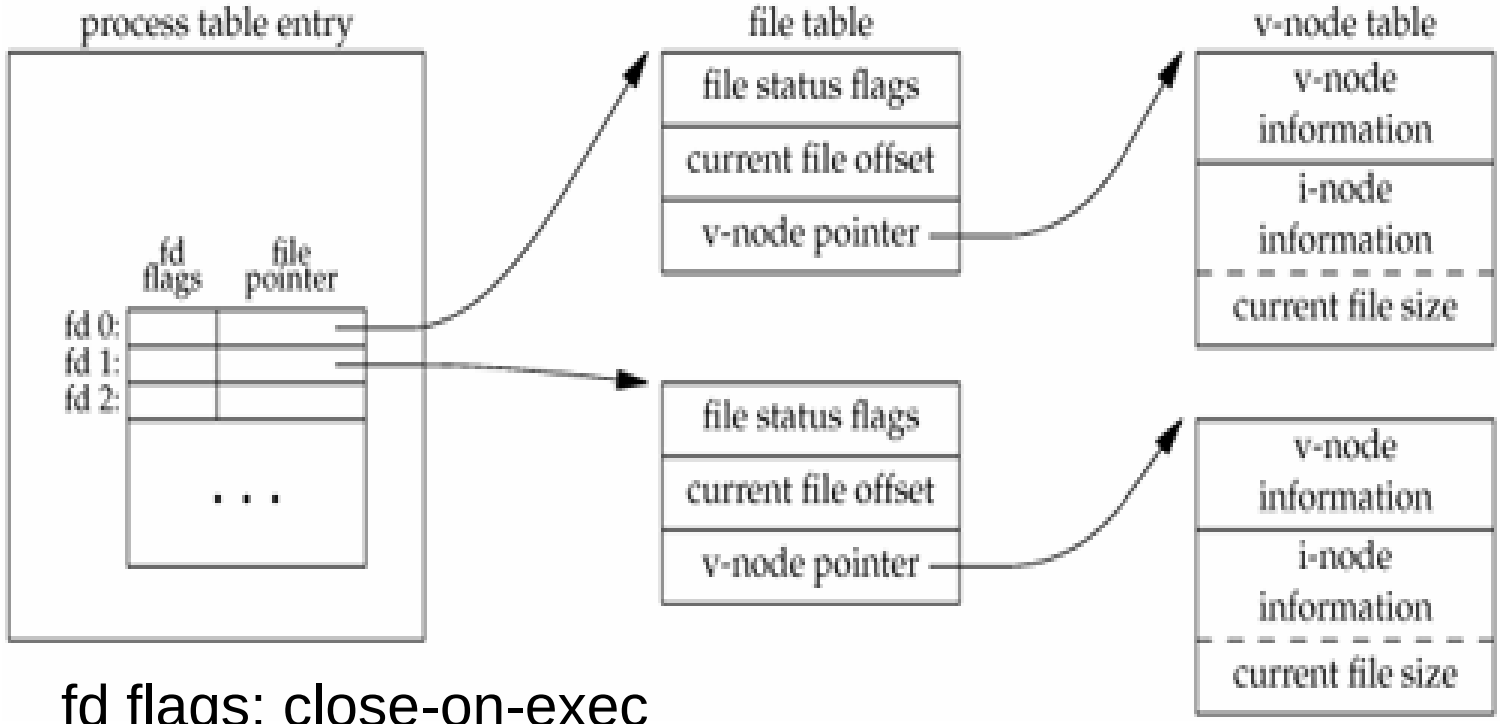
- writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf.
- On success, the number of bytes written are returned.
- Write: current position in a file or to the end if O_APPEND
- On error, -1 is returned, and errno is set appropriately.

I/O Efficiency

Most file systems support read-ahead to improve performance

Sequential reads – system tries to read in more data than an application requests

Kernel Data Structures

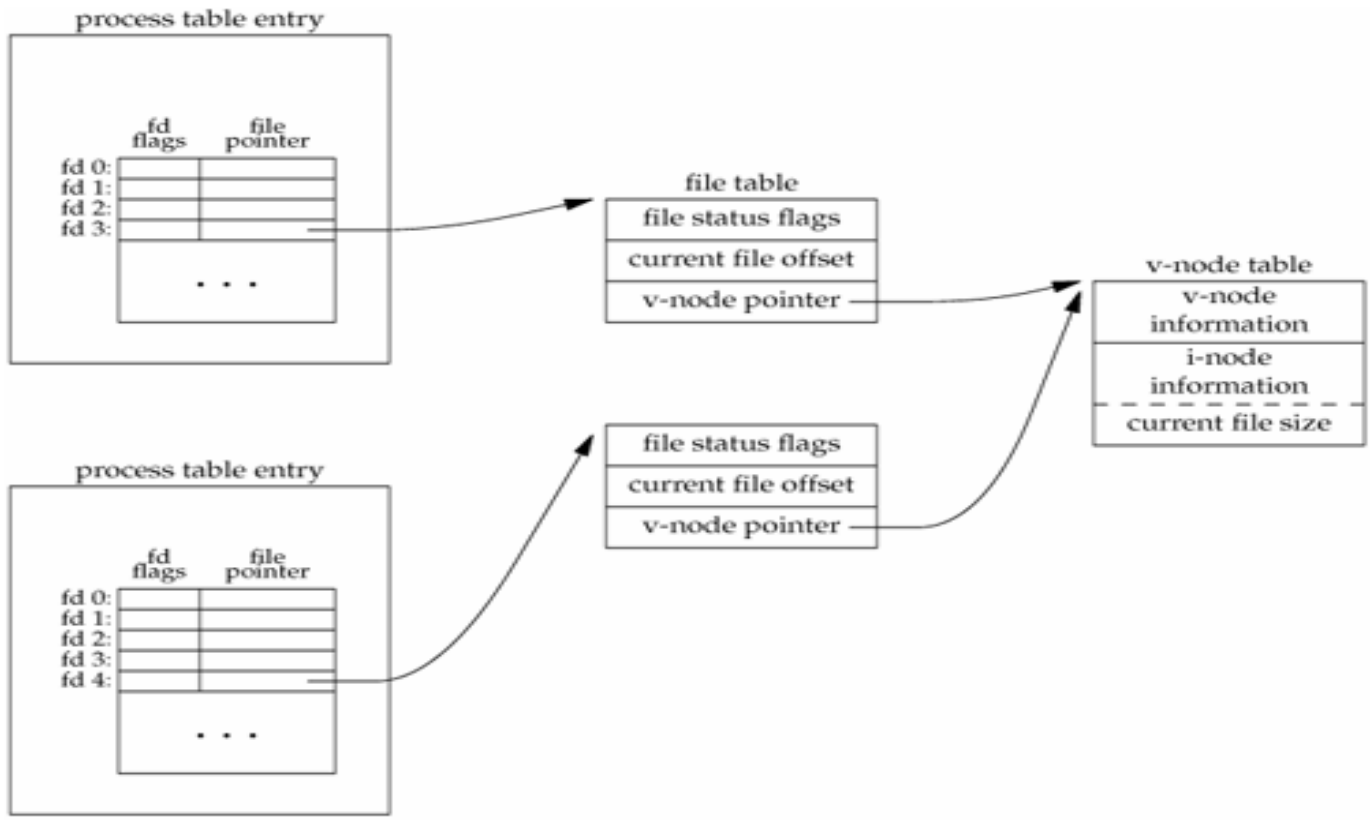


fd flags: close-on-exec
Value: 0,1

v-node structure that contains information about the type of file and pointers to functions that operate on the file.

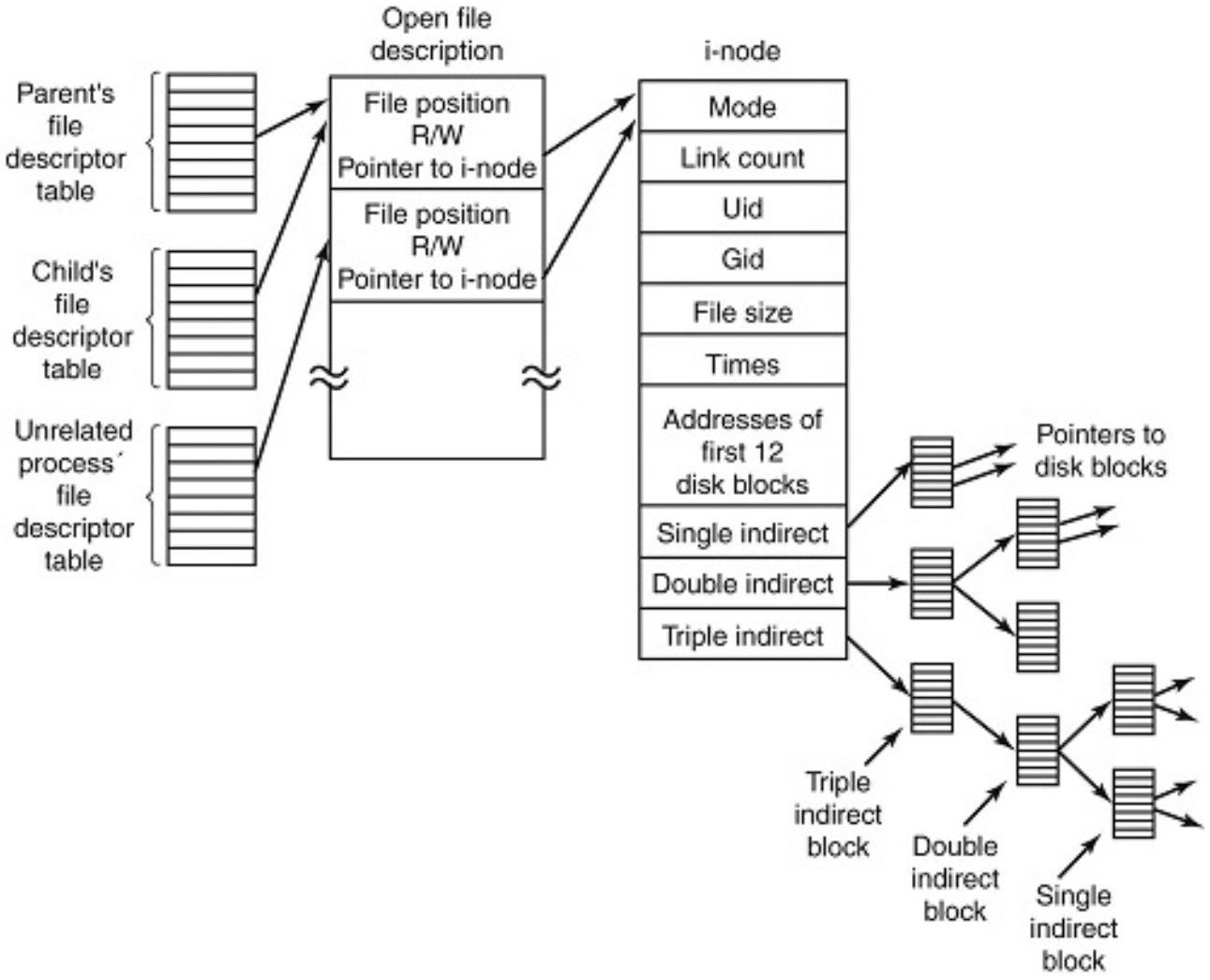
UNIX SYSTEM PROGRAMMING

File Sharing



UNIX SYSTEM PROGRAMMING

i-node Structure



Source: https://practice.geeksforgeeks.org/problems/explain-the-structure-of-inode-table

Atomic Operations

```
if (lseek(fd, 0L, 2) < 0)              /* position to EOF */
        err_sys("lseek error");
if (write(fd, buf, 100) != 100)         /* and write */
        err_sys("write error");
```

What happens if,

Case 1: single process access the file with descriptor,fd

Case 2: Multiple processes appending to the same file.

UNIX SYSTEM PROGRAMMING

Atomic Operaions

The Single UNIX Specification includes XSI extensions that allow applications to seek and perform I/O atomically.
These extensions are pread and pwrite.

ssize_t pread(int filedes, void *buf, size_t nbytes, off_t offset); **returns:**
number of bytes read, 0 if end of file, -1 on error

ssize_t pwrite(int filedes, const void *buf, size_t nbytes, off_t offset);
Returns: number of bytes written if OK, -1 on error

pread and pwrite

Calling pread is equivalent to calling **lseek** followed by a call to **read**, with the following exceptions.

There is no way to interrupt the two operations using pread.The file pointer is not updated.

Calling **pwrite** is equivalent to calling **lseek** followed by a call to **write**, with similar exceptions.

chown() - Unix, Linux System Call


```
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
```

UNIX SYSTEM PROGRAMMING

fcntl Function

The fcntl function can change the properties of a file that is already open.

int fcntl(int filedes, int cmd, ... /* int arg */);

Returns: depends on cmd if OK (see following), -1 on errore sysem calls.

The fcntl function is used for five different purposes.

- Duplicate an existing descriptor (cmd = F_DUPFD)
- Get/set file descriptor flags (cmd = F_GETFD or F_SETFD)
- Get/set file status flags (cmd = F_GETFL or F_SETFL)
- Get/set asynchronous I/O ownership (cmd = F_GETOWN or F_SETOWN)
- Get/set record locks (cmd = F_GETLK, F_SETLK, or F_SETLKW)

fcntl()

F_DUPFD: Duplicate the file descriptor filedes.
F_GETFD: Return the file descriptor flags for filedes as the value of the function. Currently, only one file descriptor flag is defined: the FD_CLOEXEC flag.
F_SETFD: Set the file descriptor flags for filedes. The new flag value is set from the third argument
F_GETFL:Return the file status flags for filedes as the value of the function.

File status flags for fcntl

- O_RDONLY: open for reading only
- O_WRONLY: open for writing only
- O_RDWR: open for reading and writing
- O_APPEND: append on each write
- O_NONBLOCK: nonblocking mode
- O_SYNC: wait for writes to complete (data and attributes)
- O_DSYNC: wait for writes to complete (data only)
- O_RSYNC: synchronize reads and writes
- O_FSYNC: wait for writes to complete (FreeBSD and Mac OS X only)
- O_ASYNC: asynchronous I/O (FreeBSD and Mac OS X only)

fcntl()

F_SETFL: Set the file status flags to the value of the third argument (taken as an integer).
The only flags that can be changed are O_APPEND, O_NONBLOCK, O_SYNC, O_DSYNC, O_RSYNC, O_FSYNC, and O_ASYNC.

F_GETOWN: Get the process ID or process group ID currently receiving the SIGIO and SIGURG signals.

F_SETOWN: Set the process ID or process group ID to receive the SIGIO and SIGURG signals.

THANK YOU

Chandravva Hebbi
Department of Computer Science and Engineering

chandravvahebbi@pes.edu