Linux File I/O System Calls - Notes

File Descriptors (FDs)

Every process starts with three open FDs:

0 = stdin, 1 = stdout, 2 = stderr

write(2)

Purpose: write bytes to a file descriptor.

Syntax:

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbytes);
```

read(2)

Purpose: read bytes from a file descriptor.

Syntax:

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t nbytes);
```

open(2)

Purpose: open file/device and get FD.

Syntax:

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

Flags: O_RDONLY, O_WRONLY, O_RDWR

O_APPEND, O_TRUNC, O_CREAT, O_EXCL

Return: FD (>=0), -1=error.

close(2)

Purpose: release FD, flush buffers.

Syntax:

```
#include <unistd.h>
int close(int fd);
```

creat(2), unlink(2)

creat(path, mode) -> open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

unlink(path) -> delete directory entry.

lseek(2)

Purpose: reposition file offset.

Syntax:

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

Linux File I/O System Calls - Notes

whence: SEEK_SET, SEEK_CUR, SEEK_END.

Examples

Redirect stdout to file:

```
close(1);
open("out.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644);
```

Append-only writes:

```
open("log.txt", O_WRONLY|O_CREAT|O_APPEND, 0644);
```

Copy file (char-by-char):

```
while (read(in, &c, 1) == 1) write(out, &c, 1);
```

Checklist

- Always check return values (-1=error).

- Handle partial reads/writes in loops.

- Consider umask when creating files.

- Check close() errors in writers.

- lseek() only on seekable FDs.

Practice Ideas

- Implement simple grep.

- Implement simple more (paginate output).

- Play with printf specifiers.

- Toggle file permissions with chmod().

- Use lseek() for partial copy.

Linux File I/O System Calls - Syntax & Explanations

Compact explanations for the most-used low-level file I/O calls. Each entry includes purpose, key parameters, returns, common errors, behavior notes, and a minimal example.

File Descriptors (FDs)

Small integers indexing per-process open file table. 0=stdin, 1=stdout, 2=stderr. Many syscalls take an FD instead of a path (e.g., read/write/close/lseek).

open(2)

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

Purpose: Open a file/device and return a new FD.

Params: path (filename), oflags (access mode + options). With O_CREAT, provide mode (permission bits before umask).

Access modes (choose one): O_RDONLY, O_WRONLY, O_RDWR.

Useful flags: O_CREAT (create if missing), O_EXCL (fail if exists), O_TRUNC (truncate to 0), O_APPEND (append-only writes), O_CLOEXEC (close on exec), O_NONBLOCK (nonblocking), O_SYNC/O_DSYNC (sync writes).

Returns: FD >= 0 on success; -1 on error (errno set).

Common errno: ENOENT (no such file), EEXIST (with O_EXCL), EACCES/EPERM (permissions), EMFILE/ENFILE (FD/process/system limit), EISDIR (writing to dir).

```
int fd = open("log.txt", O_WRONLY|O_CREAT|O_APPEND, 0644);
if (fd == -1) /* handle error */ ;
```

close(2)

```
#include <unistd.h>
int close(int fd);
```

Purpose: Release an FD and underlying kernel resources.

Returns: 0 on success, -1 on error (e.g., EBADF).

Notes: For writers, errors may appear on close (e.g., NFS/disk full). Always check close() after writing.

```
if (close(fd) == -1) {/* report close error */}
```

read(2)

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Purpose: Read up to count bytes from FD into buf.

Returns: >0 number of bytes read; 0 on EOF (regular files); -1 on error (errno set).

Notes: May read fewer than requested (short read). On pipes/sockets/ttys, may block unless O_NONBLOCK set. Interrupted by signals (EINTR).

Linux File I/O System Calls - Syntax & Explanations

```
ssize_t n = read(fd, buf, sizeof buf);
if (n == -1) {/* handle error */}
```

write(2)

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

Purpose: Write up to count bytes from buf to FD.

Returns: >=0 number of bytes written (can be short); -1 on error.

Notes: Short writes must be retried in a loop. O_APPEND makes each write atomically append. EINTR possible; EPIPE when writing to a closed pipe/socket (SIGPIPE may terminate unless ignored).

```
ssize_t left = len; const char *p = buf;
while (left > 0) {
    ssize_t n = write(fd, p, left);
    if (n <= 0) {/* handle -1 or 0 */ break; }
    p += n; left -= n;
}
```

lseek(2)

```
#include <unistd.h>
#include <sys/types.h>
off_t lseek(int fd, off_t offset, int whence); // SEEK_SET, SEEK_CUR, SEEK_END
```

Purpose: Reposition the file offset for random access.

Returns: New absolute offset >= 0; -1 on error (e.g., ESPIPE if FD not seekable like pipes/terminals).

Notes: Use lseek(fd, 0, SEEK_END) to get size; combine with read/write for partial copies. Creating sparse files: lseek beyond EOF then write.

```
off_t size = lseek(fd, 0, SEEK_END);
if (size == (off_t)-1) {/* handle error */}
```

creat(2)

```
#include <fcntl.h>
int creat(const char *path, mode_t mode); // legacy
```

Purpose: Legacy shorthand for creating/truncating a file for writing.

Equivalent to: open(path, O_WRONLYIO_CREATIO_TRUNC, mode).

Recommendation: Prefer open(...O_CREAT...).

```
int fd = creat("new.txt", 0644); // or use open("new.txt", O_WRONLY|O_CREAT|O_TRUNC,
0644)
```

unlink(2)

```
#include <unistd.h>
int unlink(const char *path);
```

Linux File I/O System Calls - Syntax & Explanations

Purpose: Remove a directory entry (name -> inode link). File data is freed when link count hits zero and no FDs reference it.

Returns: 0 on success; -1 on error (e.g., ENOENT, EISDIR for directories - use rmdir for empty dirs). Notes: A process can unlink an open file; content persists until last FD closes (useful for temp files).

```
int rc = unlink("old.log");
if (rc == -1) {/* handle error */}
```

dup(2), dup2(2), dup3(2)

```
#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);
#include <fcntl.h>
int dup3(int oldfd, int newfd, int flags); // e.g., O_CLOEXEC
```

Purpose: Duplicate an FD (share offset/flags on same open file description).

dup(): returns lowest-numbered unused FD.

dup2()/dup3(): force the new FD number (e.g., redirect stdout to a file FD=1).

```
int fd = open("out.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644);
if (fd != -1) {
    dup2(fd, 1); // stdout now goes to out.txt
    close(fd);
}
```

fsync(2), fdatasync(2)

```
#include <unistd.h>
int fsync(int fd);
int fdatasync(int fd);
```

Purpose: Flush dirty buffers to stable storage.

fsync() writes metadata and data; fdatasync() may omit some metadata.

Use after critical updates (logs, databases). Returns 0 on success, -1 on error.

fstat(2), stat(2)

```
#include <sys/stat.h>
int fstat(int fd, struct stat *st);
int stat(const char *path, struct stat *st);
```

Purpose: Query file attributes (size, mode, type, timestamps, link count, device/inode).

Useful fields: st_mode (use S_ISREG/S_ISDIR macros), st_size, st_nlink.

End-to-End Example: Safe Copy (looping I/O)

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

Linux File I/O System Calls - Syntax & Explanations

```
#include <errno.h>

int copy(const char *src, const char *dst){
    int in = open(src, O_RDONLY);
    if (in == -1) return -1;
    int out = open(dst, O_WRONLY|O_CREAT|O_TRUNC, 0644);
    if (out == -1){ close(in); return -1; }
    char buf[1<<15];
    for(;;){
        ssize_t n = read(in, buf, sizeof buf);
        if (n == 0) break;            // EOF
        if (n < 0){ if (errno==EINTR) continue; close(in); close(out); return -1; }
        char *p = buf; ssize_t left = n;
        while (left > 0){
            ssize_t m = write(out, p, left);
            if (m < 0){ if (errno==EINTR) continue; close(in); close(out); return -1; }
            left -= m; p += m;
        }
    }
    int rc1 = close(in); int rc2 = close(out);
    return (rc1==-1 || rc2==-1) ? -1 : 0;
}
```

```c
// lseek
#include <unistd.h>     // for read(), lseek(), close()
#include <sys/stat.h>   // for file permission constants
#include <fcntl.h>      // for open() flags
#include <stdlib.h>     // for exit()
#include <stdio.h>      // for printf(), perror()

int main() {
    int fd;
    char buffer[20];
    ssize_t bytes_read;

    // Open file for reading
    fd = open("test.txt", O_RDONLY);
    if (fd < 0) {
        perror("open");
        exit(1);
    }

    // Read first 10 bytes
    bytes_read = read(fd, buffer, 10);
    if (bytes_read < 0) {
        perror("read");
        close(fd);
        exit(1);
    }
    buffer[bytes_read] = '\0';
    printf("First 10 bytes: %s\n", buffer);

    // Move file pointer forward by 5 bytes from current position
    if (lseek(fd, 5, SEEK_CUR) == (off_t) -1) {
        perror("lseek");
        close(fd);
        exit(1);
    }

    // Read next 10 bytes from the new position
    bytes_read = read(fd, buffer, 10);
    if (bytes_read < 0) {
        perror("read");
        close(fd);
        exit(1);
    }
    buffer[bytes_read] = '\0';
    printf("Next 10 bytes (after skipping 5): %s\n", buffer);

    close(fd);
```

```
    return 0;
}
```

```c
// read system call
#include <unistd.h>
#include <stdlib.h>

int main(void)
{
    char buffer[128];
    int nread;

    nread = read(0, buffer, 128);  // read from stdin
    if (nread == -1) {
        write(2, "A read error has occurred\n", 26);
        exit(1);
    }

    if (write(1, buffer, nread) != nread) {
        write(2, "A write error has occurred\n", 27);
        exit(1);
    }

    exit(0);
}
```

```c
//write system call
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    const char *msg = "Here is some data\n";
    size_t len = strlen(msg);

    if (write(1, msg, len-1) != (ssize_t)len)
        write(2, "A write error has occurred on file descriptor 1\n", 49);

    exit(0);
}
```

```c
// io system calls
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>

int main() {
    int fd;
    char buffer[100];
    ssize_t n;

    // 1. creat() - Create a file for writing
    fd = creat("demo.txt", S_IRUSR | S_IWUSR); // Owner read & write
    if (fd == -1) {
        write(2, "Error creating file\n", 21);
        exit(1);
    }

    // 2. write() - Write bytes to file
    const char *msg = "Hello, this is a demo file.\n";
    write(fd, msg, strlen(msg));

    // 3. close() - Close the file after writing
    close(fd);

    // 4. open() - Open the file for reading
    fd = open("demo.txt", O_RDONLY);
    if (fd == -1) {
        write(2, "Error opening file\n", 20);
        exit(1);
    }

    // 5. read() - Read bytes from file
    n = read(fd, buffer, sizeof(buffer) - 1);
    buffer[n] = '\0'; // Null-terminate for printing
    write(1, "File contents:\n", 15);
    write(1, buffer, n);

    // 6. close() again
    close(fd);

    // 7. unlink() - Delete the file
    unlink("demo.txt");
    write(1, "\nFile deleted.\n", 15);

    return 0;
```

```
}
```