File SubSystem System Calls Handbook

This handbook is designed for our **Linux System Programming (LSP) batch** to understand one of the most important parts of the Unix/Linux kernel: the **file subsystem**.
Every file you open, every byte you read or write, every directory you create or delete, ultimately goes through a small set of **system calls** provided by the kernel.
As a system programmer, you must know these calls by name, their prototypes, parameters, return values, and—most importantly—**when and why to use each one**.

1. creat()

Header File

#include <fcntl.h>

Prototype

int creat(const char *pathname, mode_t mode);

Parameters

- pathname – Path of file to create (and open for writing).
- mode – Permission bits (e.g. 0644).

Description

Historical system call. Equivalent to:

open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
Used to create a new file or truncate an existing one.

Return value

- >= 0 – New file descriptor
- -1 – Error, errno set

2. open()

Header File

#include <fcntl.h>

Prototype

int open(const char *pathname, int flags, ... /* mode_t mode */);

Parameters

- pathname – File path to open or create.
- flags – Access mode + options (O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_APPEND, O_TRUNC, etc.).
- mode – Permission bits (used only if O_CREAT is set).

Description

Opens or creates a file and returns a file descriptor for further I/O.

Return value

- >= 0 – File descriptor
- -1 – Error

3. openat()

Header File

#include <fcntl.h>

Prototype

int openat(int dirfd, const char *pathname, int flags, ... /* mode_t mode */);

Parameters

- dirfd – Directory FD, or AT_FDCWD (current directory).
- pathname – Path relative to dirfd.
- flags, mode – Same as open().

Description
Opens a file relative to a directory file descriptor. Used for secure, race-free path handling.

Return value
Same as open().

4. close()

Header File

```
#include <unistd.h>
```

Prototype

```
int close(int fd);
```

Parameters

- fd – File descriptor to close.

Description
Releases a file descriptor and frees its kernel resources.

Return value

- 0 – Success
- -1 – Error

5. read()

Header File

```
#include <unistd.h>
```

Prototype

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

Parameters

- fd – File descriptor.
- buf – Buffer where data is read into.
- count – Maximum bytes to read.

Description

Reads raw bytes from a file, pipe, socket, or device.

Return value

- > 0 – Number of bytes read
- 0 – End of file
- -1 – Error

6. write()

Header File

#include <unistd.h>

Prototype

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

Parameters

- fd – File descriptor.
- buf – Data to write.
- count – Number of bytes to attempt to write.

Description

Writes raw bytes to a file, device, pipe, or socket.

Return value

- >= 0 – Bytes actually written (may be less than count)
- -1 – Error

7. pread()

Header File

#include <unistd.h>

Prototype

ssize_t pread(int fd, void *buf, size_t count, off_t offset);

Parameters

- fd – File descriptor.
- buf – Buffer for data.
- count – Bytes to read.

- offset – Absolute file offset.

Description

Reads from a specific offset without changing the file's current offset. Good for multithreaded file access.

Return value
Same pattern as read().

8. pwrite()

Header File

#include <unistd.h>

Prototype

ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset);

Parameters

- Same meanings as pread().

Description

Writes at a specific offset without changing the current file offset.

Return value
Same pattern as write().

9. lseek()

Header File

#include <unistd.h>

Prototype

off_t lseek(int fd, off_t offset, int whence);

Parameters

- fd – File descriptor.

- offset – Offset value.
- whence – SEEK_SET, SEEK_CUR, or SEEK_END.

Description
Moves the file offset ("cursor") for random access.

Return value

- >= 0 – New file offset
- -1 – Error

10. pipe()

Header File

#include <unistd.h>

Prototype

int pipe(int pipefd[2]);

Parameters

- pipefd – Array of 2 integers:
  - pipefd[0] – Read end
  - pipefd[1] – Write end

Description

Creates an anonymous pipe: a unidirectional data channel.

Return value

- 0 – Success
- -1 – Error

11. pipe2()

Header File

#include <fcntl.h>

Prototype

int pipe2(int pipefd[2], int flags);

Parameters

- pipefd – Same as pipe().
- flags – Optional flags (O_NONBLOCK, O_CLOEXEC).

Description

Creates a pipe and sets additional flags atomically.

Return value

Same as pipe().

12. stat()

Header File

#include <sys/stat.h>

Prototype

int stat(const char *path, struct stat *buf);

Parameters

- path – Path of file.
- buf – Pointer to struct stat to receive metadata.

Description

Gets metadata of a file (type, permissions, owner, size, timestamps, link count, etc.).

Return value

- 0 – Success
- -1 – Error

13. lstat()

Header File

#include <sys/stat.h>

Prototype

int lstat(const char *path, struct stat *buf);

Parameters
Same as stat().

Description
Like stat(), but if path is a symbolic link, it returns info about the **link itself**, not the target.

Return value
Same as stat().

14. fstat()

Header File

#include <sys/stat.h>

Prototype

int fstat(int fd, struct stat *buf);

Parameters

- fd – File descriptor.
- buf – struct stat to fill.

Description

Gets metadata of an open file via its descriptor.

Return value
0 on success, −1 on error.

15. fstatat()

Header File

#include <sys/stat.h>

Prototype

```
int fstatat(int dirfd, const char *path,
            struct stat *buf, int flags);
```

Parameters

- dirfd – Directory FD or AT_FDCWD.
- path – Relative path.
- buf – Metadata structure.
- flags – AT_SYMLINK_NOFOLLOW, etc.

Description

Stat relative to a directory FD; supports extra flags.

Return value

0 on success, −1 on error.

16. statx() (Linux-specific, advanced)

Header File

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

Prototype

```
int statx(int dirfd, const char *pathname,
          int flags, unsigned int mask,
          struct statx *buf);
```

Parameters

- dirfd, pathname – Like other *at calls.
- flags – Behavior flags.
- mask – Which fields to retrieve.
- buf – Pointer to struct statx.

Description
Newer, more detailed metadata syscall than stat() / fstat().

Return value

0 on success, −1 on error.

17. access()

Header File

```
#include <unistd.h>
```

Prototype

```
int access(const char *path, int mode);
```

Parameters

- path – Path to test.
- mode – R_OK, W_OK, X_OK, F_OK.

Description

Checks if the calling process's **real** UID/GID has requested access.

Return value

- 0 – Access allowed
- -1 – Access denied or error

18. faccessat()

Header File

```
#include <fcntl.h>
#include <unistd.h>
```

Prototype

```
int faccessat(int dirfd, const char *path, int mode, int flags);
```

Parameters

- dirfd – Base directory FD.
- path – Relative path.
- mode – Same as access().
- flags – E.g. AT_EACCESS.

Description

Access check relative to a directory FD, with extra options.

Return value
Same as access().

19. umask()

Header File

#include <sys/stat.h>

Prototype

mode_t umask(mode_t mask);

Parameters

- mask – New process umask.

Description
Sets the calling process's file creation mask. Affects default permissions of newly created files/dirs.

Return value

- Returns previous mask.

20. chmod()

Header File

#include <sys/stat.h>

Prototype

int chmod(const char *path, mode_t mode);

Parameters

- path – File path.
- mode – New permission bits (e.g. 0755).

Description
Changes file's permission bits.

Return value
0 on success, −1 on error.

21. fchmod()

Header File

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

Prototype

```
int fchmod(int fd, mode_t mode);
```

Parameters

- fd – File descriptor.
- mode – New permissions.

Description
Changes permissions using an open file descriptor.

Return value
0 on success, −1 on error.

22. fchmodat()

Header File

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

Prototype

```
int fchmodat(int dirfd, const char *path,
             mode_t mode, int flags);
```

Parameters

- dirfd – Base directory FD.
- path – Relative path.
- mode – New permissions.
- flags – Optional (e.g. AT_SYMLINK_NOFOLLOW).

Description
chmod relative to directory FD.

Return value
0 on success, −1 on error.

23. chown()

Header File

#include <unistd.h>

Prototype

int chown(const char *path, uid_t owner, gid_t group);

Parameters

- path – File path.
- owner – New UID or -1.
- group – New GID or -1.

Description

Changes owner and/or group of a file.

Return value
0 on success, −1 on error.

24. fchown()

Header File

#include <unistd.h>

Prototype

int fchown(int fd, uid_t owner, gid_t group);

Description
Same as chown() but uses file descriptor.

Return value
0 on success, −1 on error.

25. lchown()

Header File

#include <unistd.h>

Prototype

int lchown(const char *path, uid_t owner, gid_t group);

Description
Changes ownership of a symbolic link itself (does not follow link).

Return value
0 on success, −1 on error.

26. fchownat()

Header File

#include <unistd.h>

Prototype

```
int fchownat(int dirfd, const char *path,
             uid_t owner, gid_t group, int flags);
```

Description
Owner/group change relative to directory FD, with extra flags.

Return value
0 on success, −1 on error.

27. utime()

Header File

#include <utime.h>

Prototype

int utime(const char *filename, const struct utimebuf *times);

Parameters

- filename – Path.
- times – Pointer to access and modification times (or NULL to set to current time).

Description
Updates file access and modification times (old interface).

Return value
0 on success, −1 on error.

28. utimes()

Header File

```
#include <sys/time.h>
```

Prototype

```
int utimes(const char *filename, const struct timeval times[2]);
```

Description
More precise microsecond-resolution version of utime().

Return value
0 on success, −1 on error.

29. futimens()

Header File

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

Prototype

```
int futimens(int fd, const struct timespec times[2]);
```

Description
Updates timestamps using a file descriptor with nanosecond precision.

Return value
0 on success, −1 on error.

30. utimensat()

Header File

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

Prototype

```
int utimensat(int dirfd, const char *path,
              const struct timespec times[2], int flags);
```

Description
Updates timestamps relative to directory FD, with flags.

Return value
0 on success, −1 on error.

31. link()

Header File

#include <unistd.h>

Prototype

```
int link(const char *oldpath, const char *newpath);
```

Parameters

- oldpath – Existing file.
- newpath – New directory entry name.

Description
Creates a **hard link** to the same inode as oldpath.

Return value
0 on success, −1 on error.

32. linkat()

Header File

#include <unistd.h>

Prototype

```
int linkat(int olddirfd, const char *oldpath,
           int newdirfd, const char *newpath,
           int flags);
```

Description
Hard link creation using directory FDs; supports flags like AT_SYMLINK_FOLLOW.

Return value
0 on success, −1 on error.

33. symlink()

Header File

```
#include <unistd.h>
```

Prototype

```
int symlink(const char *target, const char *linkpath);
```

Parameters

- target – String stored in link.
- linkpath – Path of new symlink.

Description
Creates a symbolic link.

Return value
0 on success, −1 on error.

34. symlinkat()

Header File

```
#include <unistd.h>
```

Prototype

```
int symlinkat(const char *target, int newdirfd,
              const char *linkpath);
```

Description
Symlink creation relative to directory FD.

Return value
0 on success, −1 on error.

35. readlink()

Header File

```
#include <unistd.h>
```

Prototype

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz);
```

Parameters

- path – Symbolic link path.
- buf – Buffer to store target.
- bufsiz – Size of buffer.

Description
Reads the contents (target path) of a symbolic link without following it.

Return value

- > 0 – Number of bytes placed in buf (no '\0' added)
- -1 – Error

36. readlinkat()

Header File

```
#include <unistd.h>
```

Prototype

```
ssize_t readlinkat(int dirfd, const char *path,
                   char *buf, size_t bufsiz);
```

Description
Same as readlink() but relative to directory FD.

Return value
Same pattern as readlink().

37. unlink()

Header File

```
#include <unistd.h>
```

Prototype

```
int unlink(const char *path);
```

Parameters

- path – Name to remove.

Description
Removes a directory entry. The actual file is deleted when link count and open FDs go to zero.

Return value
0 on success, −1 on error.

38. unlinkat()

Header File

#include <unistd.h>

Prototype

int unlinkat(int dirfd, const char *path, int flags);

Parameters

- dirfd – Base directory FD.
- path – Relative path.
- flags – 0 or AT_REMOVEDIR.

Description
Removes a file or (with AT_REMOVEDIR) a directory, relative to dirfd.

Return value
0 on success, −1 on error.

39. rename()

Header File

#include <stdio.h>

Prototype

int rename(const char *oldpath, const char *newpath);

Parameters

- oldpath – Old name.
- newpath – New name.

Description
Atomically renames or moves a file/directory.

Return value
0 on success, −1 on error.

40. renameat()

Header File

```
#include <fcntl.h>
```

Prototype

```
int renameat(int olddirfd, const char *oldpath,
             int newdirfd, const char *newpath);
```

Description
Rename using directory FDs.

Return value
0 on success, −1 on error.

41. renameat2()

Header File

```
#include <fcntl.h>
```

Prototype

```
int renameat2(int olddirfd, const char *oldpath,
              int newdirfd, const char *newpath,
              unsigned int flags);
```

Description
Extended rename with flags like RENAME_NOREPLACE, RENAME_EXCHANGE.

Return value
0 on success, −1 on error.

42. mkdir()

Header File

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

Prototype

```
int mkdir(const char *path, mode_t mode);
```

Parameters

- path – Directory name.
- mode – Initial permissions bits.

Description
Creates a new directory.

Return value
0 on success, −1 on error.

43. mkdirat()

Header File

#include <sys/stat.h>

Prototype

int mkdirat(int dirfd, const char *path, mode_t mode);

Description
Directory creation relative to directory FD.

Return value
0 on success, −1 on error.

44. rmdir()

Header File

#include <unistd.h>

Prototype

int rmdir(const char *path);

Description
Removes an **empty** directory.

Return value
0 on success, −1 on error.

45. chdir()

Header File

#include <unistd.h>

Prototype

int chdir(const char *path);

Parameters

- path – New working directory path.

Description
Changes current working directory of the process.

Return value
0 on success, −1 on error.

46. fchdir()

Header File

#include <unistd.h>

Prototype

int fchdir(int fd);

Description
Changes working directory to the directory referred by file descriptor.

Return value
0 on success, −1 on error.

47. getcwd()

Header File

#include <unistd.h>

Prototype

char *getcwd(char *buf, size_t size);

Parameters

- buf – Buffer for path.
- size – Size of buffer.

Description
Gets the absolute pathname of the current working directory.

Return value

- Pointer to buf on success
- NULL on error

48. mknod()

Header File

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

Prototype

```
int mknod(const char *path, mode_t mode, dev_t dev);
```

Parameters

- path – Name of special file.
- mode – File type + permissions (S_IFIFO, S_IFCHR, S_IFBLK, etc.).
- dev – Device ID for special files.

Description
Creates special files (device nodes, FIFOs). Typically requires root privileges.

Return value
0 on success, −1 on error.

49. mknodat()

Header File

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

Prototype

```
int mknodat(int dirfd, const char *path,
            mode_t mode, dev_t dev);
```

Description
mknod relative to a directory FD.

Return value
0 on success, −1 on error.

50. mkfifo()

Header File

#include <sys/stat.h>

Prototype

int mkfifo(const char *path, mode_t mode);

Description
Creates a named pipe (FIFO).

Return value
0 on success, −1 on error.

51. mkfifoat()

Header File

#include <sys/stat.h>

Prototype

int mkfifoat(int dirfd, const char *path, mode_t mode);

Description
Creates FIFO relative to directory FD.

Return value
0 on success, −1 on error.

52. truncate()

Header File

#include <unistd.h>

Prototype

int truncate(const char *path, off_t length);

Description
Changes size of a named file.

Return value
0 on success, −1 on error.

53. ftruncate()

Header File

#include <unistd.h>

Prototype

int ftruncate(int fd, off_t length);

Description
Changes size of file referred by descriptor.

Return value
0 on success, −1 on error.

54. fsync()

Header File

#include <unistd.h>

Prototype

int fsync(int fd);

Description
Flushes all modified data and metadata of that FD to disk.

Return value
0 on success, −1 on error.

55. fdatasync()

Header File

#include <unistd.h>

Prototype

int fdatasync(int fd);

Description
Flushes file data and minimal metadata required to retrieve it.

Return value
0 on success, −1 on error.

56. sync()

Header File

#include <unistd.h>

Prototype

void sync(void);

Description
Schedules all pending filesystem I/O for writing to disk.

Return value
None (void).

57. syncfs()

Header File

#include <unistd.h>

Prototype

int syncfs(int fd);

Description
Flushes all pending I/O on the filesystem containing the file referred by fd.

Return value
0 on success, −1 on error.

58. sendfile()

Header File

```
#include <sys/sendfile.h>
```

Prototype

```
ssize_t sendfile(int out_fd, int in_fd,
                 off_t *offset, size_t count);
```

Parameters

- out_fd – Destination descriptor (usually socket).
- in_fd – Source descriptor (usually file).
- offset – Starting offset in input (or NULL).
- count – Bytes to transfer.

Description
Efficiently transfers data between file descriptors in kernel space.

Return value

- >= 0 – Bytes transferred
- -1 – Error

59. copy_file_range()

Header File

```
#include <unistd.h>
```

Prototype

```
ssize_t copy_file_range(int fd_in, off64_t *off_in,
                        int fd_out, off64_t *off_out,
                        size_t len, unsigned int flags);
```

Description
Copies a range of data between two files (possibly optimized by filesystem).

Return value
Bytes copied on success, −1 on error.

60. fallocate()

Header File

#include <fcntl.h>

Prototype

int fallocate(int fd, int mode, off_t offset, off_t len);

Description
Preallocates or manipulates space for a file (e.g. allocate, punch holes).

Return value
0 on success, −1 on error.

61. posix_fadvise()

Header File

#include <fcntl.h>

Prototype

int posix_fadvise(int fd, off_t offset, off_t len, int advice);

Description
Hints to kernel about expected I/O pattern (POSIX_FADV_SEQUENTIAL, etc.).

Return value
0 on success, error number on failure (not -1).

62. posix_fallocate()

Header File

#include <fcntl.h>
Prototype

int posix_fallocate(int fd, off_t offset, off_t len);

Description
Guarantees space is reserved on disk for a file region.

Return value
0 on success, error number on failure.

63. mmap()

Header File

```
#include <sys/mman.h>
```

Prototype

```
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
```

Parameters

- addr – Suggested starting address (or NULL).
- length – Mapping length.
- prot – Protection flags (PROT_READ, PROT_WRITE, …).
- flags – MAP_SHARED, MAP_PRIVATE, etc.
- fd – File descriptor to map.
- offset – File offset (page aligned).

Description
Maps a file (or device) into the process address space.

Return value

- Pointer to mapping on success
- MAP_FAILED ((void *)-1) on error

64. munmap()

Header File

```
#include <sys/mman.h>
```

Prototype

```
int munmap(void *addr, size_t length);
```

Description
Unmaps a previously mapped region.

Return value
0 on success, −1 on error.

65. msync()

Header File

#include <sys/mman.h>

Prototype

int msync(void *addr, size_t length, int flags);

Description
Writes modified pages of a mapped region back to the underlying file.

Return value
0 on success, −1 on error.

66. dup()

Header File

#include <unistd.h>

Prototype

int dup(int oldfd);

Description
Creates a duplicate of an existing file descriptor with the lowest available number.

Return value
New descriptor on success, −1 on error.

67. dup2()

Header File

#include <unistd.h>
Prototype

int dup2(int oldfd, int newfd);

Description
Duplicates oldfd into newfd (closes newfd if open).

Return value
newfd on success, −1 on error.

68. dup3()

Header File

```
#include <fcntl.h>
```

Prototype

```
int dup3(int oldfd, int newfd, int flags);
```

Description
Like dup2() but allows flags (e.g. O_CLOEXEC).

Return value
newfd on success, −1 on error.

69. fcntl()

Header File

```
#include <fcntl.h>
```

Prototype

```
int fcntl(int fd, int cmd, ... /* arg */);
```

Parameters

- fd – File descriptor.
- cmd – Control command (F_GETFL, F_SETFL, F_GETFD, F_SETFD, F_SETLK, etc.).
- arg – Command-dependent argument.

Description
Manipulates descriptor flags, file status flags, and record locks.

Return value
Depends on cmd (often 0 or some value). -1 on error.

70. flock()

Header File

#include <sys/file.h>

Prototype

int flock(int fd, int operation);

Parameters

- fd – File descriptor.
- operation – LOCK_SH, LOCK_EX, LOCK_UN (+ LOCK_NB).

Description
Advisory whole-file locking.

Return value
0 on success, −1 on error.

71. ioctl()

Header File

#include <sys/ioctl.h>

Prototype

int ioctl(int fd, unsigned long request, ...);

Parameters

- fd – Device or file descriptor.
- request – Command code.
- extra arg – Usually pointer to structure.

Description
Performs device-specific or file-descriptor-specific control operations.

Return value
0 or command-specific value on success, −1 on error.

72. poll()

Header File

#include <poll.h>

Prototype

int poll(struct pollfd *fds, nfds_t nfds, int timeout);

Description
Waits for one or more file descriptors to become ready for I/O.

Return value
Number of ready fds, 0 on timeout, −1 on error.

73. select()

Header File

#include <sys/select.h>

Prototype

```
int select(int nfds,
           fd_set *readfds,
           fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);
```

Description
Monitors multiple file descriptors for readiness.

Return value
Number of ready fds, 0 on timeout, −1 on error.