


# INF333

# Operating Systems

## Lecture II

Burak Arslan  
ext-inf333@burakarslan.com 

Galatasaray Üniversitesi

Lecture II  
2025-02-19

## Course website

[burakarslan.com/inf333](http://burakarslan.com/inf333) 

## Based On

[cs111.stanford.edu](https://cs111.stanford.edu) 

[cs212.stanford.edu](https://cs212.stanford.edu) 

[OSC-10 Slides](#) 

# **File Systems**

A Gentle Introduction

# File Systems

## Memory (RAM)

- ▶ Fast, less space, more expensive
- ▶ Byte-addressable: can quickly access any byte of data by address, but not individual bits by address
- ▶ Not persistent: cannot store data between power-offs

## Storage

- ▶ Slower, more space, cheaper
- ▶ Sector-addressable: cannot read/write just one byte of data – can only read/write “sectors” of data at a time
- ▶ Persistent: stores data between power-offs

# File Systems

File systems are designed to work on hardware like **hard disk drives** or **solid state drives**

- ▶ They only understand sectors.
- ▶ This is the only api we are ever going to get:

```
void readSector(size_t sectorNumber, void *data);  
void writeSector(size_t sectorNumber, const void *data);
```

# File Systems

But!

- ▶ We want files and folders and permissions and links and ...
- ▶ We need a software layer that translates between file system primitives and sector operations

# File Systems

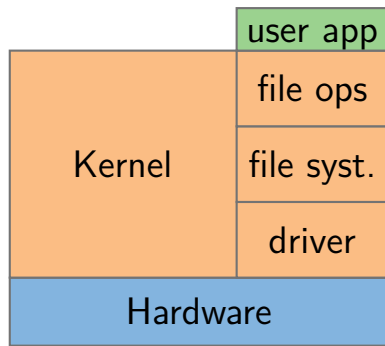
But!

- ▶ We want files and folders and permissions and links and ...
- ▶ We need a software layer that translates between file system primitives and sector operations
- ▶ This layer is called a **file system**.



# File Systems

- ▶ File systems translate file operations to sector operations
- ▶ They sit on top of protocols like SATA, USB, NVME etc.



# File Systems

File systems implement operations like:

- ▶ Creating a new file on disk
- ▶ Looking up the location of a file on disk
- ▶ Reading/editing all or part of an existing file from disk
  - e.g., sequential/random access creating folders on disk
- ▶ Getting the contents of folders on disk
- ▶ etc.

# File Systems

File systems are still a very active field.

Certainly not “a solved problem”  
though pretty mature implementations exist

# File Systems

Problems that file systems have to deal with:

## **Space Management**

- Fast access to files (maximize locality)

- Sharing space between users

- Efficient use of disk space

## **Naming** How do users find files?

## **Reliability** Information must survive OS crashes and hardware failures.

## **Protection** Isolation between users, controlled sharing.

# File Systems

File systems also deal with two classes of data:

- ▶ Payload (contents of files)
- ▶ Metadata (file names, permissions, directory contents, etc.)

Since both are held in persistent storage, some blocks must be reserved for metadata.

# File Systems

Many designs exist, some wildly different than others.

From here onwards, we will focus on native Linux filesystems

# File Systems

## Terminology

A filesystem generally defines its own unit of data, a "block", that it reads/writes at a time.

**Sector** Hardware storage unit

**Block** Filesystem storage unit (1 or more sectors) – software abstraction related to storage

**Page** Kernel's I/O unit, another software abstraction related to I/O in general — be it RAM, block storage, pipes, etc.

# File Systems

## Terminology

Sector sizes are medium-dependent and sometimes customizable:

```
# nvme id-ns /dev/nvme0n1 | grep lbads
lbaf  0 : ms:0    lbads:9  rp:0x2 (in use)
lbaf  1 : ms:0    lbads:12 rp:0x1
```



# File Systems

## Terminology

Similarly, block sizes are fs-dependent and can be tuned <sup>1</sup>.

```
# xfs_info /dev/nvme0n1p2
meta-data=/dev/nvme0n1p2 isize=512 agcount=4, agsize=7864320 blks
          sectsz=512   attr=2, projid32bit=1 crc=1 finobt=1, sparse=1
          rmapbt=1 reflink=1 bigtime=1 inobtcount=1 nrext64=0
data      =                       bsize=4096   blocks=31457280, imaxpct=25
          =                       sunit=0       swidth=0 blks
naming    =version 2             bsize=4096   ascii-ci=0, ftype=1
log       =internal log         bsize=4096   blocks=16384, version=2
          =                       sectsz=512    sunit=0 blks, lazy-count=1
realtime  =none                 extsz=4096    blocks=0, rtextents=0
```

---

<sup>1</sup>It's very tricky to do so though! eg. XFS: Mount overrides sunit and swidth options

# File Systems

## Terminology

A page is the I/O unit of the kernel – it's the same everywhere.

```
$ getconf PAGESIZE  
4096
```

Changing it is a whole project:

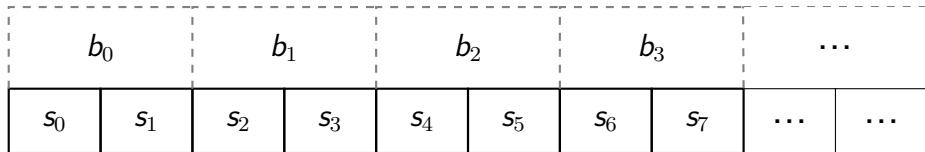
- ▶ HugePages support in Linux [↗](#)
- ▶ This got stabilized circa 2010: Huge pages: Introduction [↗](#)

# File Systems

## Terminology

A correctly-tuned block size is an essential factor in the performance of storage systems.

- ▶ Fewer transfer operations if larger
- ▶ But smaller files may read in more data than necessary



# File Systems

## Terminology

An **inode** ("index node") is a data structure that describes a file-system object such as a file or a directory.

```
# stat /etc/passwd
  File: /etc/passwd
  Size: 2325      Blocks: 33      IO Block: 2560    regular file
Device: 0,25     Inode: 10004479   Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2024-02-06 08:39:38.557196113 +0300
Modify: 2024-02-03 00:56:48.431922259 +0300
Change: 2024-02-03 00:56:48.431922259 +0300
  Birth: 2024-02-03 00:56:48.431922259 +0300
```

# File Systems

## Terminology

A **directory** is a list of inodes with their assigned names. The list includes an entry for itself (.), its parent (..), and each of its children.

```
# ls -if /etc
67108993 .          67109039 inittab.d
      128 ..        134462512 nanorc
134348929 pam.d      68298267 gentoo-release ...
201326721 gpm        203011796 protocols
 67512281 subgid      67109044 .pwd.lock

...
```

# File Systems

## Terminology

### Discussion

Can a file be retrieved by its inode? 

# File Systems

## Terminology

### Discussion

What happens when more than one directory contains an entry for the same inode?

# File Systems

## Terminology

It's called a **hard link**:

```
$ echo foo > a
$ stat a
  File: a          Size: 4
Device: 0,32  Inode: 2646944  Links: 1
$ ln a b
$ stat b
  File: b          Size: 4
Device: 0,32  Inode: 2646944  Links: 2
$ cat a
foo
$ echo bar > a
$ cat b
bar
$
```



# File Systems

## Terminology

### Discussion

Creating hard links to directories is not allowed. Why do you think this is the case?

# File Systems

An inode is "scheduled for deletion" when its refcount reaches zero

- ▶ When a file is opened, it will remain on the fs until closed
- ▶ Deleting a file immediately after opening it is a nice way to implement temporary files on Linux.

# Remaining parts of the OS

A not-so-gentle continuation

You can handle it now 😊

# Unix System Calls

- ▶ Applications still need to work with the hardware
- ▶ OS supplies a well-defined **system call** interface

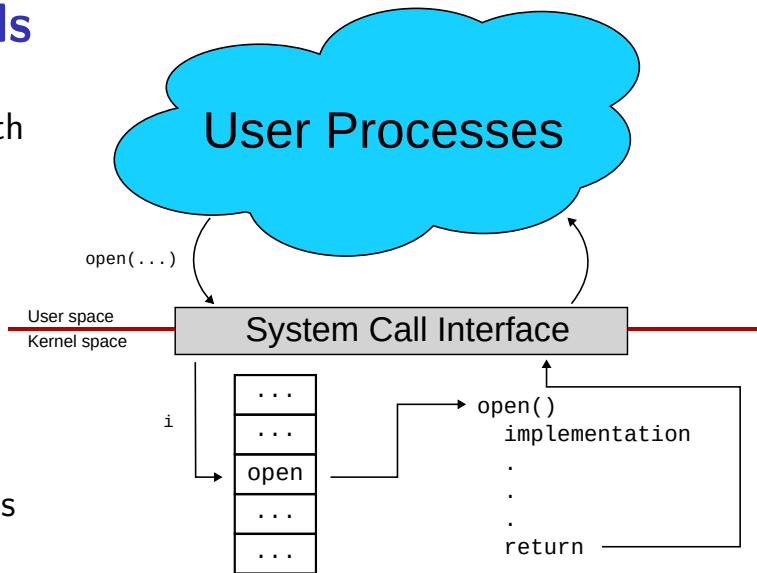
## Example:

- ▶ Apps normally don't write to storage directly, but to designated regions in the storage device sanctioned by the OS
- ▶ Uses system calls to (try to) obtain access to said addresses.

# Unix System Calls

To open a file in a FS with the `open()` system call:

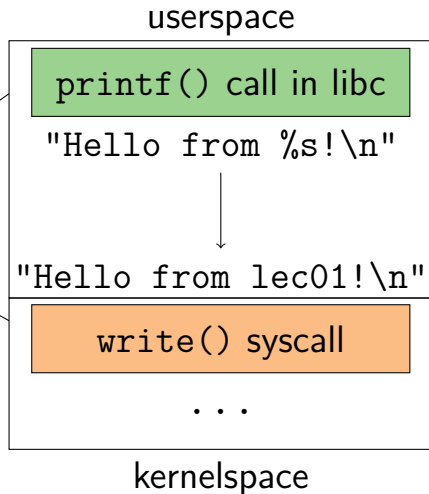
- ▶ App sets up the system call id and arguments and lets the kernel know
- ▶ The kernel executes the requested operation and returns the result



# Unix System Calls

## Closer look: printf()

```
$ cat lec01.c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("Hello from %s!\n", argv[0]);
    return 0;
}
$ gcc -o lec01 lec01.c && ./lec01
Hello from lec01!
```



# Unix System Calls

Closer look: `open()`

In Unix, applications “open” files (or devices) by name;

- ▶ `int open(char *path, int flags, /*int mode*/...);`
- ▶ Returns a **file descriptor** (fd) – used for all I/O to files and file-like objects

# Unix System Calls

Closer look: `open()`

```
int open(char *path, int flags, /*int mode*/...);
```

- ▶ flags: `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - ▶ `O_CREAT`: create the file if non-existent
  - ▶ `O_EXCL`: (w. `O_CREAT`) err out if file already exists
  - ▶ `O_TRUNC`: Truncate the file
  - ▶ `O_APPEND`: Start writing from end of file
- ▶ mode: final argument with `O_CREAT`, see `chmod`



# Unix System Calls

## Error handling

- ▶ What if open fails? Returns -1 (invalid fd)
- ▶ Most system calls return -1 on failure
  - ▶ Specific kind of error in global thread-local int `errno`
  - ▶ In retrospect, bad design decision for threads/modularity
- ▶ `#include <sys/errno.h>` for possible values
- ▶ `perror` function prints human-readable message
- ▶ Use the `strace` command <sup>2</sup> to log all the system calls that a process makes

---

<sup>2</sup>Linux-only but equivalents exist on all popular platforms

# File Descriptors

A file descriptor is a simple integer that is:

- ▶ Inherited by processes when one process spawns another,
- ▶ By convention, descriptors 0, 1, and 2 have special meaning:
  - ▶ 0 – “standard input” (`stdin` in C)
  - ▶ 1 – “standard output” (`stdout`, `printf` default in C)
  - ▶ 2 – “standard error” (`stderr`, `perror` default in C)
  - ▶ Normally all three attached to terminal

# File Descriptors

type.c

```
void
typefile (char *filename)
{
    int fd, nread;
    char buf[1024];

    fd = open (filename, O_RDONLY);
    if (fd == -1) {
        perror (filename);
        return;
    }

    while ((nread = read (fd, buf, sizeof (buf))) > 0)
        write (1, buf, nread);

    close (fd);
}
```