Linux Operating System

Quang Hoang - 2024

Viettel Digital Talent 2024 - Software & Data Engineering

Lesson Outline

1. **Linux Kernel deep dive.**
    - **Introduction**
    - **Process**
    - **Kernel synchronization**
    - **Virtual Filesystem**
2. Linux command lines.

Why Linux?

- Performance
  - Fast and Stable
  - No requirement for latest hardware
- Security
- Customizability
- It is FREE
  - GNU General Public License (GPL)
    - free to **download** the source code and make any **modifications**
    - if you distribute your changes, it should be GPL license as well.

What is Linux?

- Before Linux
  - MS DoS was the dominated OS for PC: single user, closed source.
  - Apple MacOS was better but expensive.
  - UNIX was much better (multitasking and multi-user) but much expensive.
  - People were looking for UNIX-like OS which is cheaper.

What is Linux?

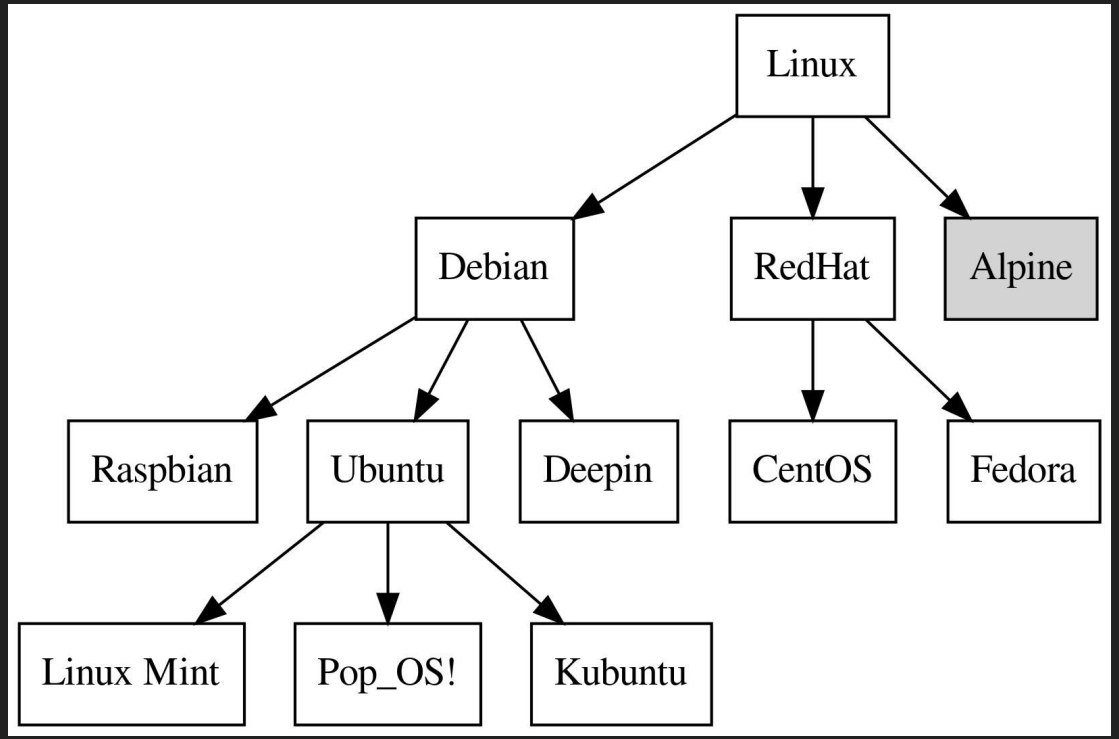
- Before Linux
  - MS DoS was the dominated OS for PC: closed source.
  - Apple MAC was better but expensive.
  - UNIX was much better but much expensive.
  - People were looking for UNIX-like OS which is cheaper.
- Beginning of Linux
  - 1991: Linus Torvalds developed Linux v0.0.1

What is Linux?

- Before Linux
  - MS DoS was the dominated OS for PC: closed source.
  - Apple MAC was better but expensive.
  - UNIX was much better but much expensive.
  - People were looking for UNIX-like OS which is cheaper.
- Beginning of Linux
  - 1991: Linus Torvalds developed Linux v0.0.1
- Linux today
  - Power smartphone, smartwatch, PC, Supercomputers
    - 71.9% of smartphone using Android as of 2022
    - 96.55% of web server run Linux as of 2015

Linux “Distro”

- short for distribution
- modification version of Linux kernel.
- full list



Operating System and Kernel

OS is a program:

- manage and protect hardware (HW) resource.
- provide HW resource to application via well-defined APIs.

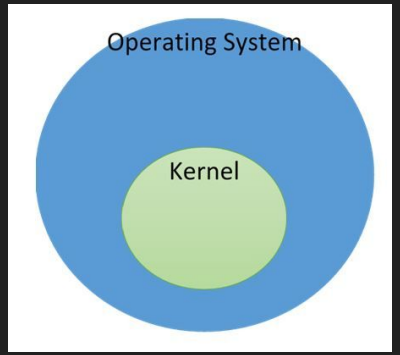
Operating System and Kernel

OS is a program:

- manage and protect hardware (HW) resource.
- provide HW resource to application via well-defined APIs.

Kernel

- core of OS.



The Process

Process

- A program in the midst of execution.
- Act like a mini virtual machine:
  - CPU: time slicing of processor.
  - Memory: address space.

Thread

- A thread is an object of activity within the process. Smallest sequence of programmed instructions that can be managed independently by a scheduler
- One process can have multiple threads.

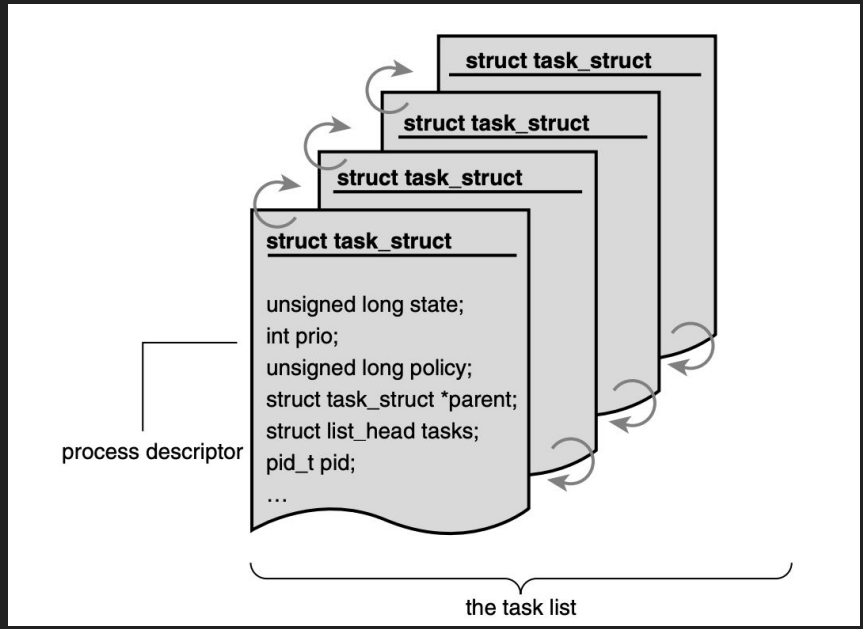
In Linux

- there is no differentiation between a **process** and a **thread.** Thread is just a special kind of process.

The Process

Process and Thread in Linux are implemented using the same data structure

A doubly-linked list of **task_struct**



Process life-cycle

- Process creation

Init process: the first process created at system boot up. pid = 0.

2 steps:

- fork(): create a child process that is a copy of current process
- exec(): load new executable into the address space and begins executing

Process life-cycle

- Process creation

```
main() {
  printf("Hello\n");
  fork();
  printf("world\n");
}
```

Process life-cycle

- Process creation

What is the output?

Hello

world

world

```
main() {
  printf("Hello\n");
  fork();
  printf("world\n");
}
```

Process life-cycle

- Process creation

How many process were created?

```
int main()
{
    int i;
    for(i=0;i<4;i++)
        fork();
    return 0;
}
```

Process life-cycle

- Process creation

fork();

fork();

fork();

fork();

```c
int main()
{
    int i;
    for(i=0;i<4;i++)
        fork();
    return 0;
}
```

Process life-cycle

- Process creation

p0

fork(); p1

fork(); p2, p3

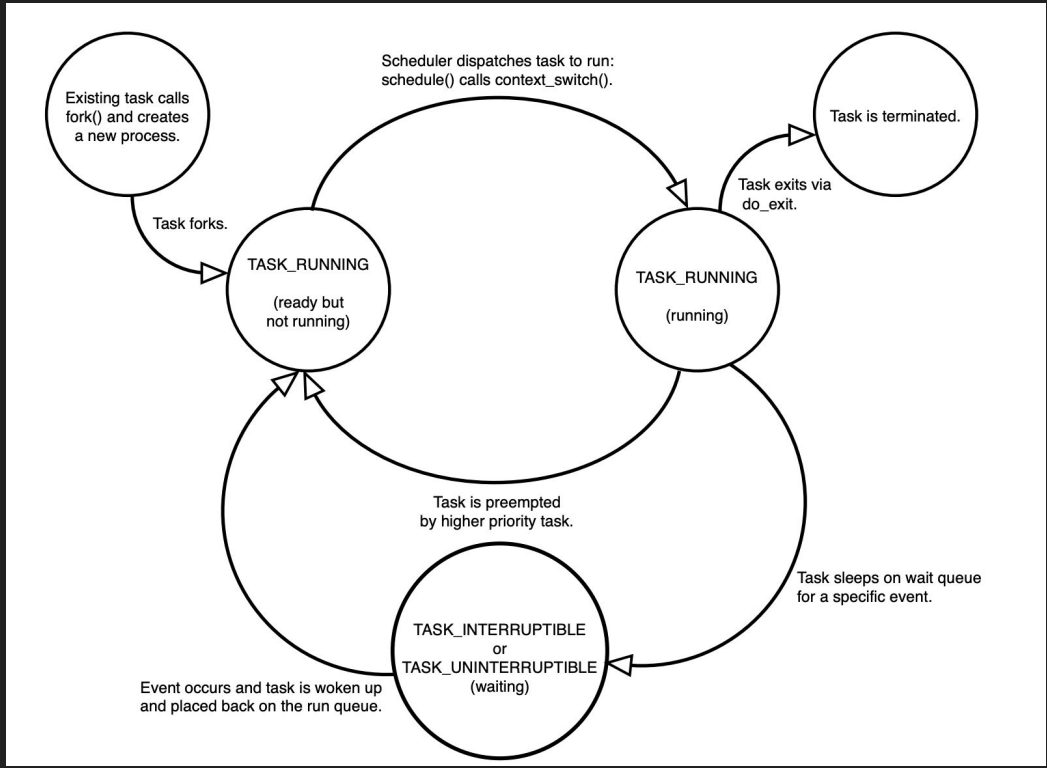
fork(); p4, p5, p6, p7

fork(); p8, p9, p10, p11, p12, p13, p14, p15

```c
int main()
{
    int i;
    for(i=0;i<4;i++)
        fork();
    return 0;
}
```

Process life-cycle

- runnable
- running
- sleeping (waiting)
- stopped



Process life-cycle

Linux is preemptive

- No matter what, scheduler will force a process to stop if there is a higher priority task pending.
- running => runnable => running

Process life-cycle

Process termination

- when a process terminates, kernel releases resource owned by process and notifies its parent.
- a process terminates when:

Process life-cycle

Process termination

- when a process terminates, kernel releases resource owned by process and notifies its parent.
- a process terminates when:
  - the process call exit() system call. (self-termination)
  - unhandled exception.

Process life-cycle

Process termination

- What if parent process is terminated before its children?

Process life-cycle

Process termination

- What if parent process is terminated before its children?
  - child processes are called **orphan** processes. (A common misconception is that child processes are by default killed when their parent exits)
  - reparent to init process.

Process scheduling

- Process scheduler decides which process runs, when, and for how long.
- 2 types of process:
  - **I/O bound: process spend much time submitting and waiting on I/O requests (e.g. access hard disk, wait for network, keyboard input …)**
    - GUI applications

Process scheduling

- Process scheduler decides which process runs, when, and for how long.
- 2 types of process:
  - I/O bound: process spend much time submitting and waiting on I/O requests (e.g. access hard disk, wait for network, keyboard input …)
    - GUI applications
  - **CPU bound: process spend much time executing code.**
    - executing an infinite loop
    - math calculation application: MATLAB, ssh-keygen

Linux favors I/O-bound processes over CPU-bound processes.

Process scheduling

Goal of process scheduler:

1. **low latency**: fast process response time.
2. **high throughput**: maximum system utilization.

Timeslice: a number presents how long a process can run until it is preempted.

- long timeslice => high latency
- short timeslice => waste time on context switching.

Completely Fair Scheduler

Idea: device CPU time equally among processes:

N processes, each should get **(100/N)%** of CPU time.

Process	Time
A	8ms
B	4ms
C	16ms
D	4ms

Completely Fair Scheduler (CFS)

How it works:

- CFS will run each process for some amount of time, selecting next the process that has run the least.
- The runtime of each process is proportional to its priority.
- When CFS is deciding what process to run next, it picks the process with the smallest runtime.
  - use a Red-Black Tree to store list of runnable processes.
  - Pick smallest node in O(1)
  - Insert in O(logn)

Kernel Synchronization

Race condition

What is the output?

```
define foo_g = 0
function Foo:
    define foo = foo_g
    foo = foo + 1
    foo_g = foo

function main:
    create a thread to run Foo
    create a thread to run Foo
    wait for all the thread finish
    if foo_g equal to 2
        print "expected"
    else
        print "unexpected"
```

Race condition

The situation when 2 or more threads update the share resource at the same time lead to the uncertain state of the shared resource.

The code that access and update the shared data is called Critical Section.

Cause of Race condition

1. Kernel preemption: one process can preempt another
2. Symmetric multiprocessing: 2 or more processors can execute kernel code at the same time
3. Interrupts: an interrupt can occur any time, interrupting the currently running code.

…

Spin Lock

- The most common lock in Linux Kernel.

Thread 1

try to lock data using spin lock

successed: acquired lock

access and update data

unlock

…

Thread 2

try to lock data using spin lock

failed: waiting…

waiting…

waiting…

successed: acquired lock

access and update data

Spin Lock

Thread will spin (busy loop) while waiting for the lock to be released.

- we should not hold the spin lock too long to avoid wasting CPU.
- you cannot sleep while holding a spin lock

Spin Lock

- you cannot sleep while holding a spin lock
- why?

thread A acquires spin lock L, then sleeps.

Thread B tries to acquire L, but because L was acquired by A, B has to wait.

If the computer only has 1 core (or multicores but A and B run on the same core), we have a deadlock:

- B waits for A to release L
- A is sleeping and cannot wake up because B is using CPU for spinning.

Semaphore

- Unlike Spin lock, semaphores in Linux are sleeping locks.

Thread 1

try to lock data using semaphore

successed: acquired lock

access and update data

unlock

…

Thread 2

try to lock data using semaphore

failed: sleep

sleeping…

sleeping…

successed: acquired lock

access and update data

Semaphore

- Suitable for locks that are held for a long time.
- Not optimal for short period lock because of the overhead of sleeping, wait queue.

N resources, M threads.

Counting semaphore

Semaphore allows an arbitrary number of simultaneous lock holder.

2 functions:

- down(): acquire the semaphore and decrease count by 1
- up(): release the semaphore and increase count by 1.

When count == 0, thread goes to sleep and waits for being woken up.

S = new Semaphore(count)

t1.down()

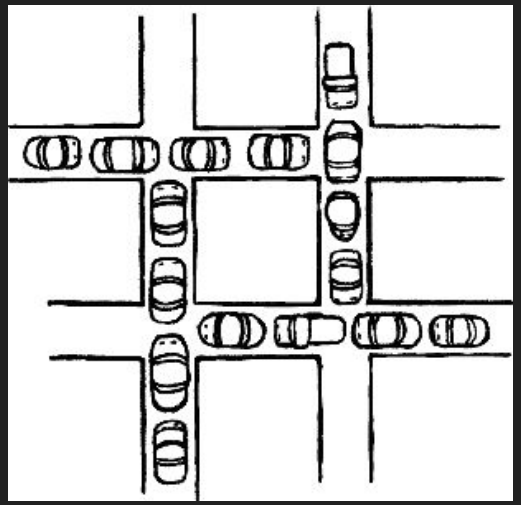
t1.up()

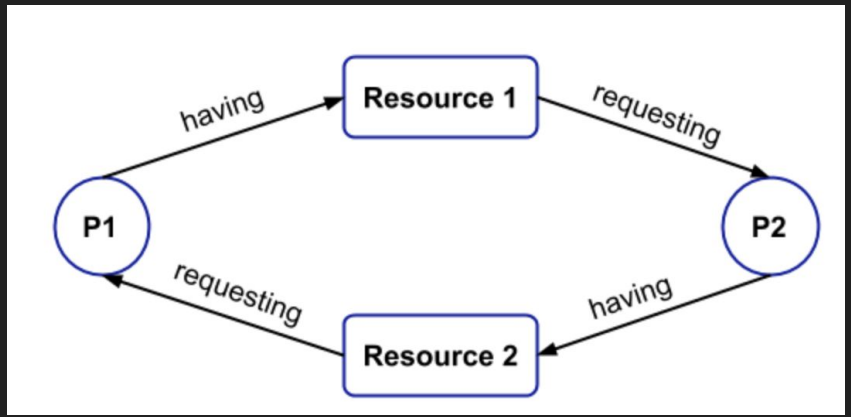
Mutex

- mutual exclusion lock
- a sleep lock
- work like a binary semaphore.
- the lock can be acquired and released by the same thread at a time.

Deadlock

- A deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.





Deadlock

- Relock (AA)

```
P0

spin_lock(&A);
...
spin_lock(&A);
```

Deadlock

- ABBA

```
P0                          P1

spin_lock(&A);              spin_lock(&B);
...                         ...
spin_lock(&B);              spin_lock(&A);
```

Deadlock

- ABBCCA deadlocks or more

```
P0                  P1                  P2

spin_lock(&A);      spin_lock(&B);      spin_lock(&C);
...                 ...                 ...
spin_lock(&B);      spin_lock(&C);      spin_lock(&A);
```

Deadlock

Rules to prevent deadlock

- Implement lock ordering. Nested locks must always be obtained in the same order.
- Do not double acquire the same lock.
- Design for simplicity. Complexity in your locking scheme invites deadlocks.

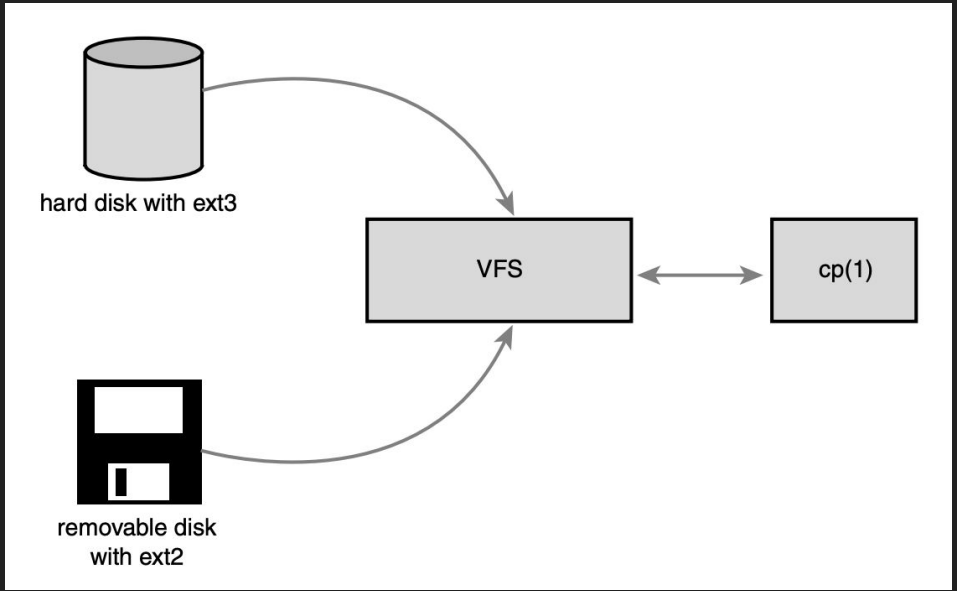
Virtual File System (VFS)

VFS is part of Linux Kernel that implements the file and file-system-related interfaces.

This allows common system calls

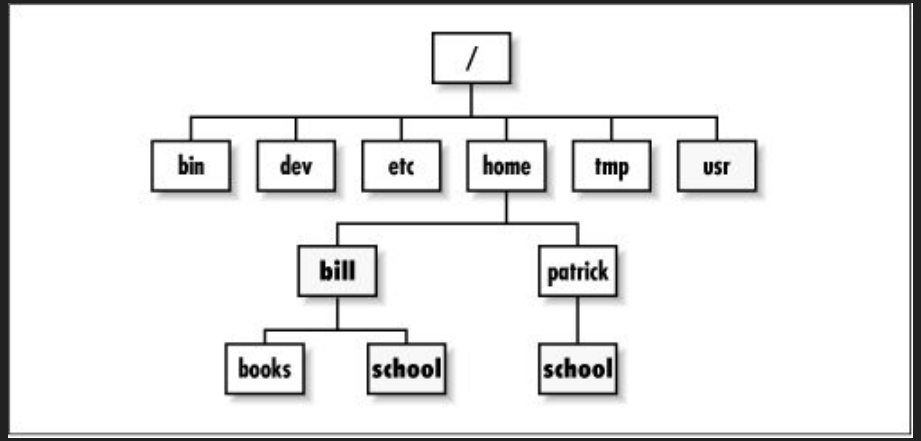
to work on different type

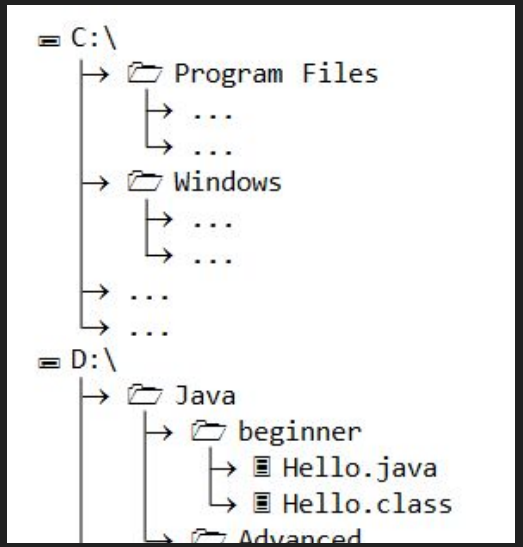
of hardware.



VFS

- Single tree
  - unlike Windows
- Mounted filesystem appear as a node in the tree.
- Everything is a file.

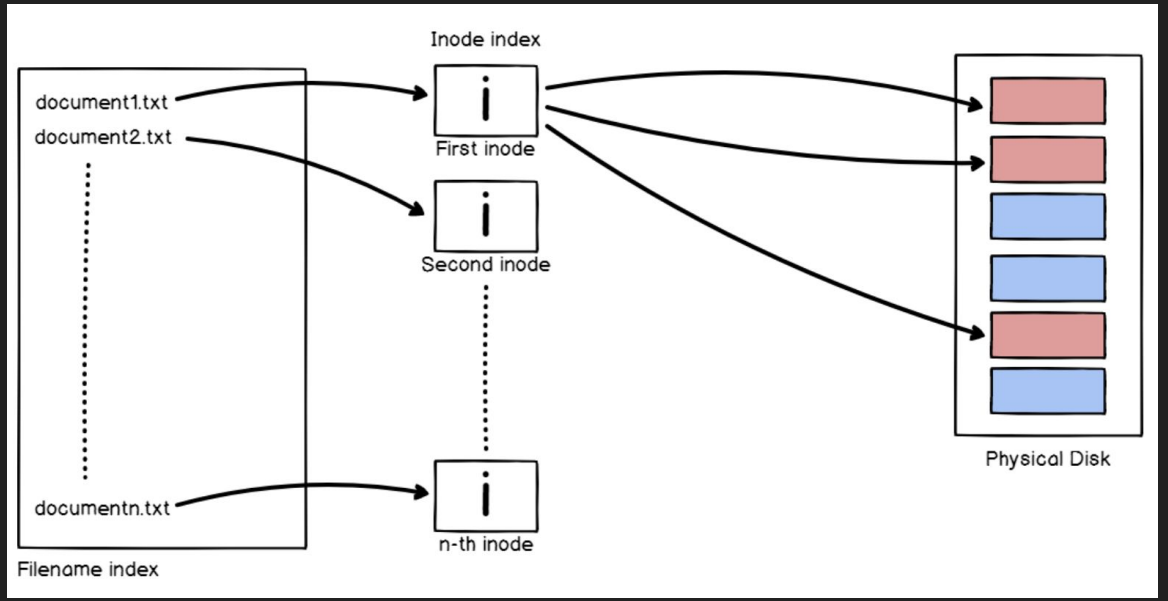




VFS objects

- superblock: represent a specific mounted filesystem
- dentry: represent a directory entry, which is a single component of a path
- inode: represent a specific file, contain metadata (size, owner, location, creation time …)
  - create
  - link, unlink
  - symlink
- file: represents an open file as associated with a process
  - read
  - write
  - lock
  - open…

Inode

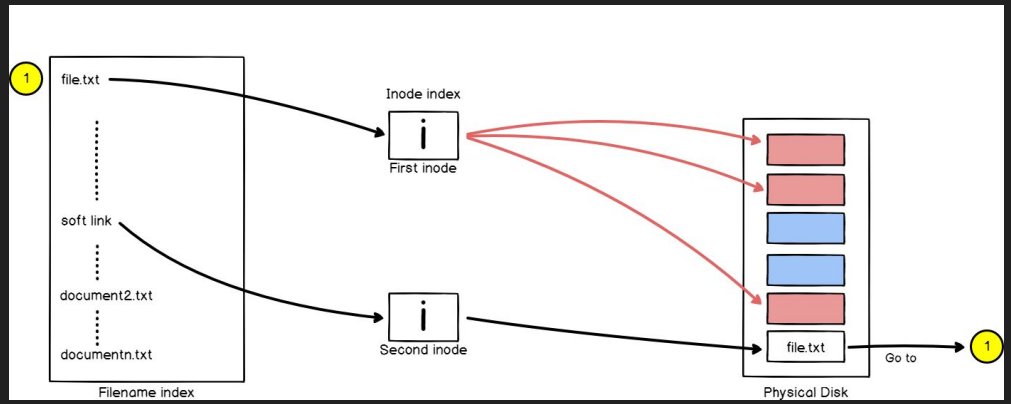


- to show inode number: ls -i <file-name>
- to view data stored in an inode: stat <file-name>

Soft Link, Hard Link

- In Linux, multiple files can point to the same Inode using Link.
- Link:
  - a pointer pointing to a file.
  - 2 types:
    - soft link (symbolic link)
    - hard link

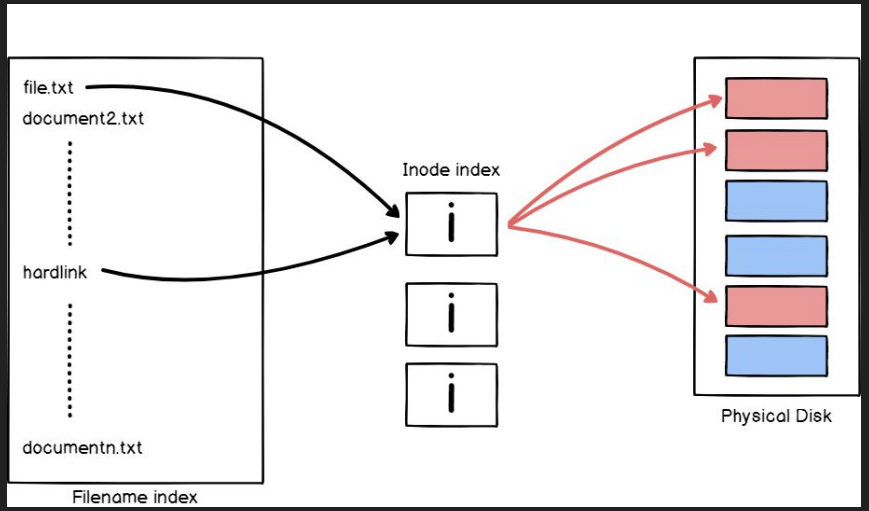
Soft Link (Symbolic Link)



Soft links are files pointing to other file on the file system (like a shortcut in Windows)

- the original file and its soft link share the same content: modify one affects other.
- the original file inode is pointing directly to the file content, soft link inode is pointing to a block containing the path to the original file
- delete or rename original file, the soft link is corrupted.

Hard Link



Hard link are instances of the file under a different name on the filesystem.

- share the inode of the original file.
- delete/rename original file doesn’t affect the hard link.
- hard link vs copy file?

Prerequisite for next lesson

1. Install docker
2. Download this folder
3. Run Ubuntu by docker, access bash terminal and install vim

```
docker run -it ubuntu bash
apt-get update
apt install vim
```

4. Find your ubuntu docker container id

```
docker ps
```

5. Open a new terminal tab, copy VT folder to docker container

```
docker cp VT <id-from-step-4>:/VT
```

6. Verify by running `ls -l VT` in docker terminal.

n processes, sum of memory n processes consume > RAM.

Virtual Memory of OS.

Hard Disk

swap memory
cache invalidation: LRU (Least recently used)