Linux For Embedded Systems

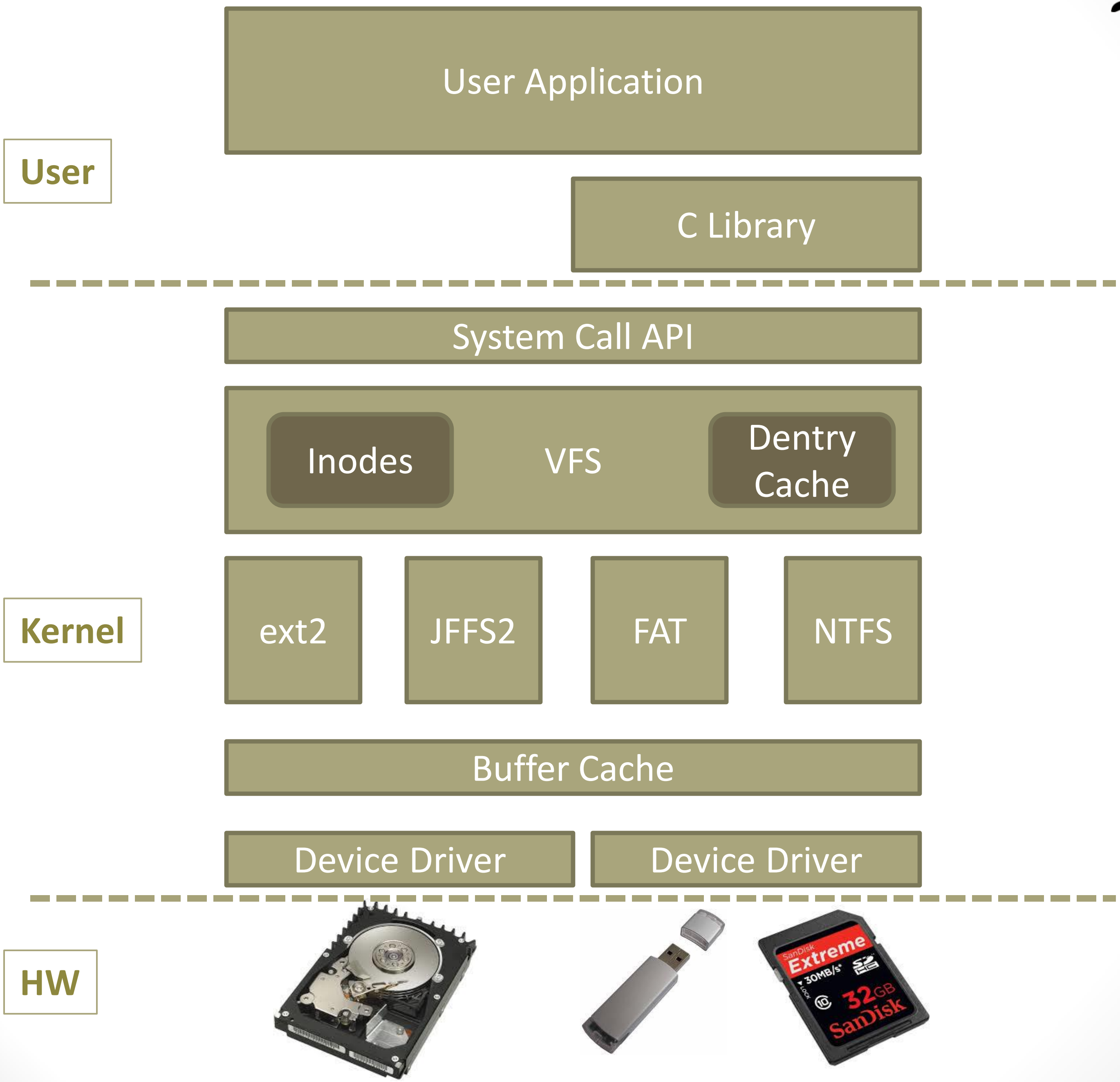
For Arabs

Course 102:

Understanding Linux

Ahmed ElArabawy

Lecture 27:
FileSystems in Linux (Part 2)


User
User Application
C Library
System Call API
Inodes
VFS
Dentry Cache
Kernel
ext2
JFFS2
FAT
NTFS
Buffer Cache
Device Driver
Device Driver
HW
Extreme
32GB
SanDisk

The Big Picture

- User applications access files using a standard API
- This does not change no matter what storage media, partition, or used file system type
- The used API results in a system call towards the kernel (generated by the C-Library)
- The system call is received by the VFS (Virtual File System) subsystem of the Linux Kernel
- This subsystem provides a unified access to the file via the concepts of nodes and dentries (A previous lecture covered VFS in details)
- VFS then communicate with the used FileSystem that contain this file
- The filesystem in turn Communicate with the device driver for the storage media using the block abstraction
- The device driver is responsible for interfacing with the storage hardware device

Popular FileSystem Types

Linux Extended FileSystems ext2

- The ***ext2*** used to be the most popular filesystem for Linux distributions running on a hard disk
- It uses block sizes of 1024, 2048, or 4096 Bytes
  - Large values waste disk space if we have a lot of small files
  - Small values increase overhead for metadata to point to the different blocks in the case of large files
- A problem with the ext2 was the risk of filesystem corruption due to unexpected reboots or power failures
  - A planned reboot will unmount all filesystems before the system is shutdown
  - This guarantees that the shutdown does not happen in the middle of a write operation
  - However, in case of a sudden power failure or a system crash, the filesystem will not be unmounted, and we may run into a corruption of the data in the fileSystem (some of the data is written without updating the filesystem metadata)
- The solution for this problem is "***Journaling***"

Journaling

- The filesystem that support journaling will have a special file "***The Journal File***"
- Every time there is a modification to the filesystem (a write operation), this change is tracked first in the journal file then committed to the actual file in the filesystem
- This way, we will have a journaling points that we can revert to in case of a corruption (mismatch between filesystem contents and its meta-data)
- Upon a boot after a sudden shutdown/reboot, the journal file is tracked, and compared to the contents of the filesystem
- Changes in the journal may be applied or removed to maintain data consistency
- Most newer filesystems support Journaling to maintain the filesystem contents data integrity

Linux Extended FileSystems ext3

- The **ext3** filesystem is an extension to the **ext2** filesystem to support Journalling
- It is both forward and backword compatible with **ext2** (we can convert **ext2** filesystem to **ext3** and vice versa)
- In **ext2** when the system shuts down abruptly, the next boot takes long time, because a consistency check is run on the filesystem
- In **ext3**, no consistency check is needed, the journaling file is checked to verify changes, which is a much faster process

Linux Extended FileSystems ext4

- The **ext4** filesystem is an extension to the **ext3** filesystem
- Currently, it is the default filesystem for Linux
- It also supports journaling
- It removes some of the limitations of **ext3**,
  - The **ext4** filesystem can support filesystem size of more than the 16 terabyte which is the limit for **ext3**
  - The **ext4** filesystem can support file sizes of up to 1 Terabyte

Second Generation Journaling FileSystem
JFFS2 FileSystems

- The jffs2 filesystem is used with flash memory storage devices
- Use of flash memory storage is very common in embedded systems
- Flash memory has the following features:
  - Multiple files per Block:
    - The block size is large (tens to hundreds of kilobytes). A typical value is 128KB
    - This means, we need to be able to store multiple files in the same block
    - Accordingly, one block on the flash may contain several small files
  - Slow wrtie operations
    - Writing a value to an empty place in the flash is performed one byte (or word) at a time (as normal devices)
    - However, you can not erase (or modify) a single byte (word)
    - Erasing requires the whole block to be erased
    - This means, if we need to modify a small file, this will require the whole block to be erased, then the block (or another empty one) will be re-written
    - This makes write operations in the flash much slower than other devices
    - This results in a higher chance of corruption due to power failures during a write operation
  - Flash Lifetime
    - Another limitation, flash memory has limited lifetime (specified in number of write operations)
    - Flash memory life time is measured by the number of write operations
    - We need to even out the write operations to avoid damaging parts of the flash too early

JFFS2 FileSystems

- The jffs2 filesystem handles the flash as follows,
  - Due to the slow write operation, journaling becomes very essential, and hence journaling is supported in JFFS2
  - It makes sure that blocks are used evenly to distribute the write operations on the flash. This is called **Wear Leveling**
- Special care is needed with the use of tools that keep updating files such as logging tools (ex. ***syslogd***, and ***klogd***)
  - If we have other forms of storage, it would be better to direct the output away from the flash memory
  - Use of caching may be useful to reduce the number of modifications
  - This affects both system performance and flash life-time

Cram FileSystem cramfs

- This filesystem objective is to compress a filesystem into a small ROM
- It is a read-only filesystem
- It supports compression of the data in the filesystem
- Useful for small embedded systems to store read only data in a small ROM or flash
- Ideal for boot ROMs

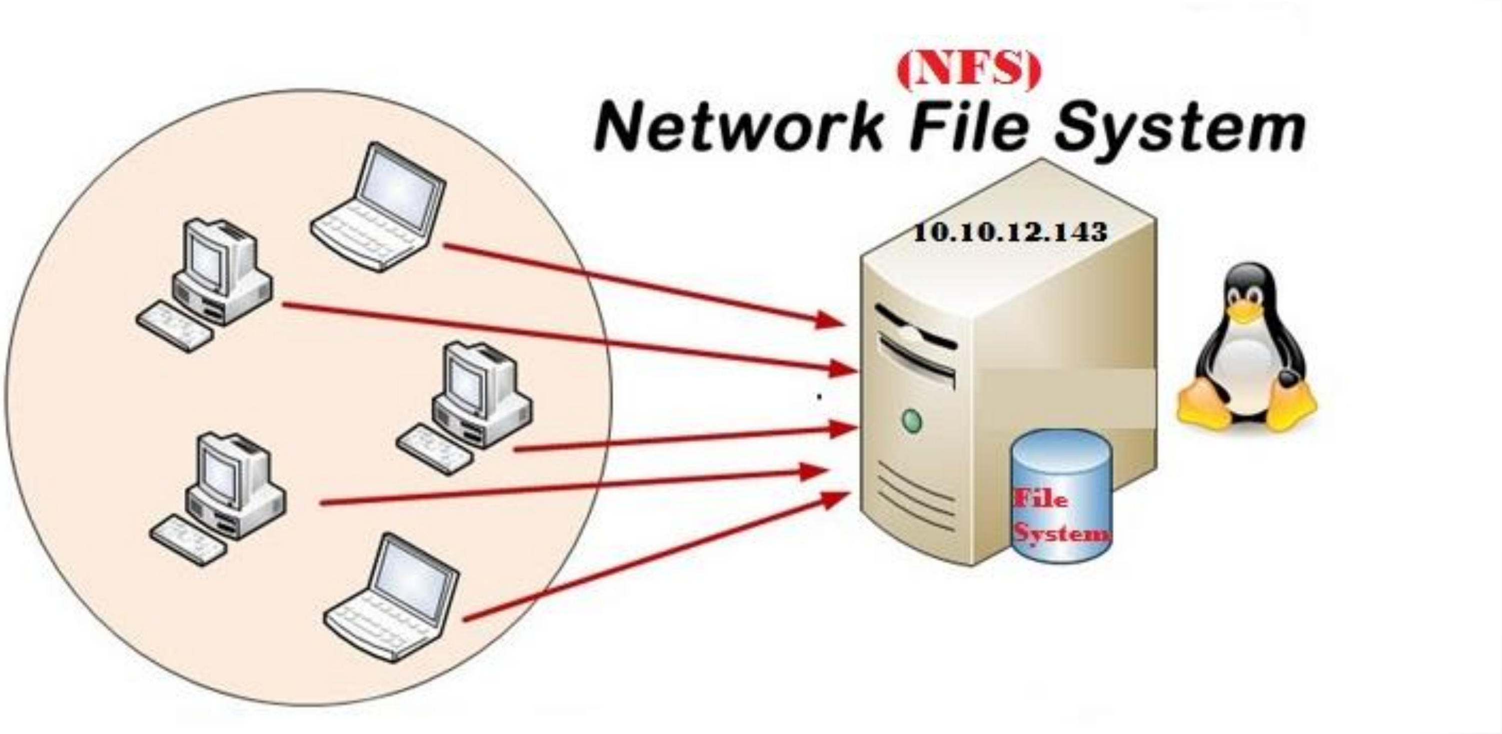
Memory Hosted FileSystems ramfs

- A filesystem that lives in the system memory (RAM)
- This provides high access speed, but it is volatile (erased at reboot time or at shutdown)
- It is different from RamDisks that it can grow and shrink based on the need

Memory Hosted FileSystems tmpfs

- Like **ramfs**, everything is stored in system volatile memory (RAM)
- Accordingly, contents of this filesystem are lost on power failure or reboot
- Different from the ramfs in that it can not grow dynamically
- It can also use the swap while the ramfs can not
- Normally mount to ***/tmp***

Network File System



Network FileSystem NFS

- This filesystem will exist on a remote machine and will be accessed through the network
- Useful for sharing folders in the network
  - A central NFS server will contain the filesystem data
  - All machines that need to have access to the data will need to contain a NFS Client
  - Machines with NFS Clients will need to mount the NFS filesystem (Map to the network drive)

NFS in Embedded Systems

- Another very useful application for NFS is development of Embedded Systems
  - Embedded target flash memory size may not be able to hold all the tools and utilities used during development
  - Hence, all the tools and utilities can be located on a remote machine (NFS Server) and the target would mount an NFS filesystem to have access to it
  - Also, during development, we don’t need to upload the image of the binary everytime we make a new build, instead, we keep the binary on the development host machine
  - The target can even have its root file system mounted as NFS, so the target will only carry the <u>bootloader and kernel</u>. All the rest will be on the remote machine (development host)

Virtual FileSystems procfs & sysfs

- Those filesystems are not stored in any storage device, but they are managed by the kernel
- Reading from a file in this filesystem results in a query to the kernel
- Writing to a file results in sending some info to the kernel
- Those filesystems will be studied in detail in separate lectures

LINUX COMMANDS

Show FileSystem Disk Space Usage (df Comand)

$ df

```
aelarabawy@aelarabawy-demo-backup64: ~
aelarabawy@aelarabawy-demo-backup64:~$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda2       98430596 15735912  77694700  17% /
udev             4070772        4   4070768   1% /dev
tmpfs            1632048      924   1631124   1% /run
none                5120        0      5120   0% /run/lock
none             4080116      464   4079652   1% /run/shm
cgroup           4080116        0   4080116   0% /sys/fs/cgroup
/dev/sda3      757071712 99267280 619347344  14% /home
/dev/sda1       96318212  5078604  86346888   6% /media/ec95cd0b-aebf-4161-8fb4-
b24d1384a905
aelarabawy@aelarabawy-demo-backup64:~$
```

Show FileSystem Disk Space Usage (df Comand)

$ df -i   (Show FileSystem inode Usage)

```
aelarabawy@aelarabawy-demo-backup64: ~
aelarabawy@aelarabawy-demo-backup64:~$ df -i
Filesystem        Inodes  IUsed    IFree IUse% Mounted on
/dev/sda2        6250496 363722  5886774    6% /
udev             1017693    518  1017175    1% /dev
tmpfs            1020029    464  1019565    1% /run
none             1020029      5  1020024    1% /run/lock
none             1020029     51  1019978    1% /run/shm
cgroup           1020029      9  1020020    1% /sys/fs/cgroup
/dev/sda3       48078848 470652 47608196    1% /home
/dev/sda1        6119424 256011  5863413    5% /media/ec95cd0b-aebf-4161-8fb4-b24
d1384a905
aelarabawy@aelarabawy-demo-backup64:~$
```

Show Process Disk Usage (du Command)

$ du <device>

- This command shows the disk usage per process for the disk specified by the device filename

  $ du /dev/sda1

Linux4

Embedded Systems

http://Linux4EmbeddedSystems.com