Linux Boot Process

- BIOS (after Power-On Self Test)
  - Usually tries to read the first sector of the first floppy, then tries to read the boot sector from the first hard disk
- Master Boot Record (MBR)
  - Determine the active disk partition
  - Load the boot sector (the first sector) of the partition, using BIOS
  - Jump to boot sector at offset 0
- Linux is loaded

Kernel Initialization

- arch/i386/boot/setup.S: entry point start:
  - initialize the hardware
  - switch to kernel mode
  - jmp 0x1000, KERNEL_CS
- arch/i386/kernel/head.S: label startup_32:
  - MMU, interrupt descriptor table initialized
  - stack, environment for kernel set up;
  - start_kernel() in init/main.c is called;

Distributed File Systems

- Properties of DFS:
  - network transparency
  - location transparency and location independence
  - user mobility and file mobility
  - fault tolerance
  - scalability

Typical Distributed File Systems

- Primitive ones: ftp, WWW
- NFS
- AFS (Andrew File System)
- CIFS
- Support for disconnected/mobile operations

Design Considerations

- name space: uniform or customizable
- stateful or stateless operation
- semantics of sharing
- remote access methods

Network File System (NFS)

- History:
  - introduced by Sun Microsystems in 1985
  - Sun published the protocol and licensed reference implementation
  - since then, NFS has been supported by every UNIX variants and PC-NFS

NFS Design Goals

- Machine and OS independence
- Crash recovery
- Transparent access
- UNIX semantics maintained on client
- Reasonable performance

Notable Features of NFS

- Protocol does not depend on any hardware or software
  - any OS can implement an NFS client or server
- Very simple recovery scheme
- No recompilation or special library required at application level
- Maintain UNIX semantics (*)
- Performance comparable to local disk FS

Basic Design

- NFS protocol: stateless, use RPC on UDP/IP
- External Data Representation (XDR)
  - integer: big endian, two's complement
  - array: size + elements
  - structure: list of components, padded to four-byte

File Handles

- Basic structure: file handle
- Procedures
  - lookup(dirfh, name) -> fh
  - create(dirfh, name, attr) -> fh
  - remove(dirfh, name) -> status
  - read/write(fh, offset, count) -> (attr, data)
  - rename/link(dirfh, name, tofh, toname)-> status

Server-side Changes

- Generation number added to inode
- file system id
- file handle = (inode number, inode generation number, file system id)

Mount Operation

- server exports file system: /etc/exports
  - permissions for client machines
- hard mount and soft mount
- Examples: server exports /usr/ and /usr/local
  - mount -t nfs nfssrv:/usr /usr
  - mount -t nfs nfssrv:/usr/u1 /u1
  - mount -t nfs nfssrv:/usr/local /usr/local

Other NFS Components

- mount protocol: OS dependent
- network lock manager, network status monitor
- daemon processes: nfsd, biod, lockd, statd

Vnode/Vfs

- Goal:
  - separate file system independent code and file system dependent code
  - allows OS to support multiple file system types simultaneously
- Vnode/Vfs architecture:
  - interface between the kernel and the modules that implement specific file systems;

Vnode/Vfs

- Vnode: represents a file, initialized during open or create
  - data fields: v_count, v_type, v_vfsmountedhere, v_data, v_op
  - v_data: points to file system dependent data
  - v_op: vop_open, vop_read, vop_lookup, ...
  - macros
- Vfs: vfs_mount, vfs_unmount, vfs_statfs, ...

NFS Implementation

- Control Flow
- File handles
  - client sees them as opaque, 32-byte objects
  - server provides a unique one-to-one mapping between files and handles

NFS Implementation

- Mount Operation:
  - client allocates vfs and calls nfs_mount
  - server's mountd parses dir name and checks permission, returns file handle of dir
  - what if client or server crashes?
- Pathname lookup
  - involves vop_lookup, vfs_vget, vop_lookup, vop_getattr, vop_fid
  - component-by-component search: why?

Statelessness in NFS Design

- Advantages
- Problems

Preserving UNIX Semantics

- open file permissions
  - difference between checking at open time versus checking at every operation
- deletion of open files
- sharing semantics

Performance Problems

- Problems:
  - synchronous writes
  - fetching of attributes
  - retransmission of requests
- Client-side caching
  - attributes and file blocks
- Deferral of writes
  - client-side delayed writes
  - write-gathering