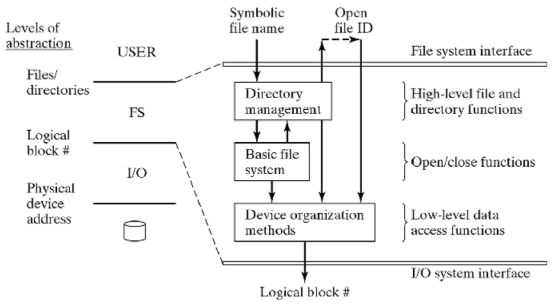
Graduate Operating Systems

Spring 2023

1

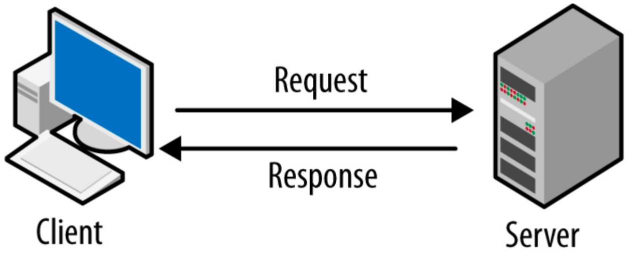
File Systems



2

Client Server Model

- Client: active
- Server: passive



3

Caching in CS Model

- Caching reduces
  - Network delay
  - Disk access delay
- Server caching - simple
  - No disk access on subsequent access
  - No cache coherence problems
  - But network delay still exists
- Client caching - more complicated
  - When to update file on server?
  - When/how to inform other processes when files is updated on server?

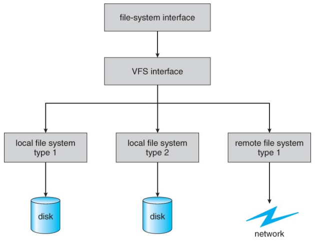
4

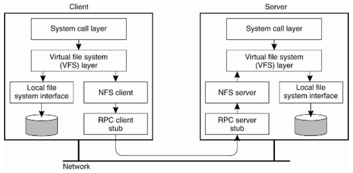
Updating Server Data

- When to update file on server?
  - Write-through
    - Overhead can be significant
  - Delayed writing
    - Requires weaker semantics
      - Only propagate update when file is closed or at end of transactions
- How to propagate changes to other caches?
  - Server initiates/informs other processes
    - Violates client/server relationship
  - Clients check periodically
    - Checking before each access defeats purpose of caching
    - Checking less frequently requires weaker semantics
      - Session semantics: only check when opening the file

5

Paper “Distributed FS”

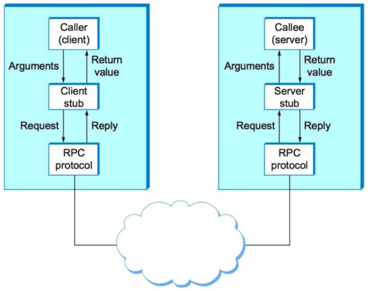



Client
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
Server
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

6

RPC

- Remote Procedure Call

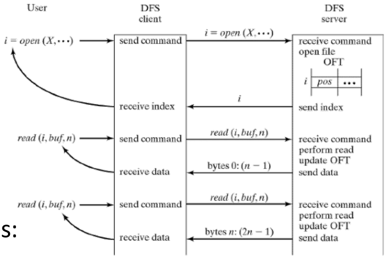


Remote Procedure Call (RPC)

7

Stateful Servers

- Stateful = Maintain state of open files
- Client passes commands & data between user process & server

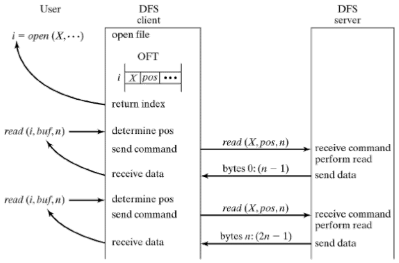


- Problem when server crashes:
  - State of open files is lost
  - Client must restore state when server recovers

8

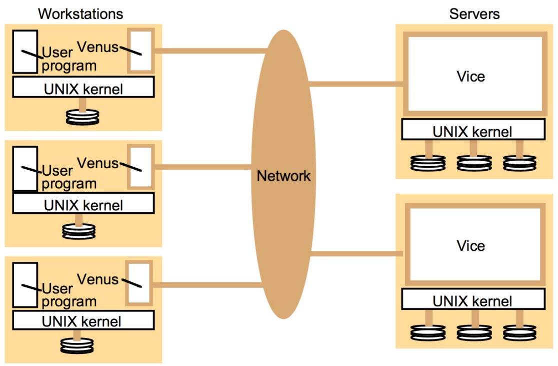
Stateless Servers

- Stateless Server (e.g., NFS pre v4) = Client maintains state of open files
- When server crashes:
  - Client waits until server recovers
  - Client reissues read/write commands



9

Paper "Distributed FS"



10

Paper “Distributed FS”

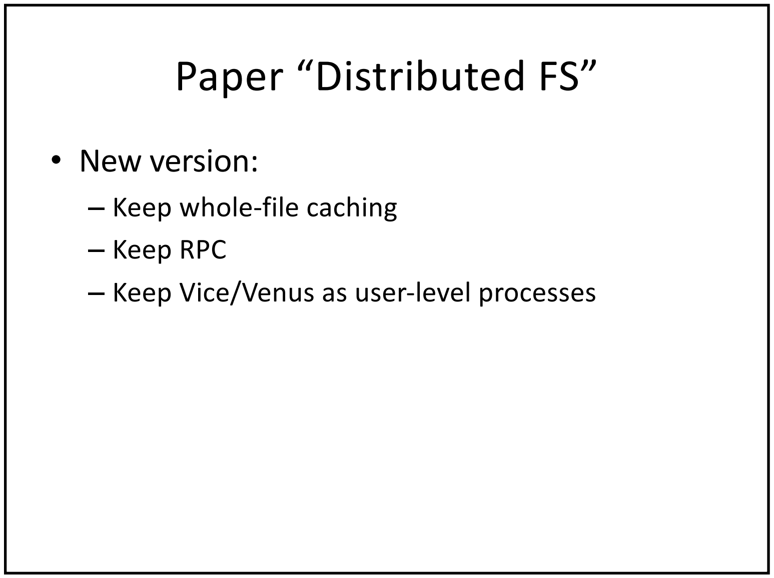
- Current system:
  - Dedicated process per client (page 53)
  - Address space sharing & IPC via files (page 53)
  - Full pathnames
  - Stub directories
  - Asynchronous slow-propagation
  - Verify file timestamp upon opening
  - Whole-file caching

11

Paper “Distributed FS”

- Observations:
  - CPU-bound vs. I/O-bound
  - ’stat’ primitive; includes cache validity check
  - Difficult to operate & maintain
  - Critical resource limits; context switches; high virtual memory paging
  - Benchmarks: Table I
  - Vice calls: Table II
  - Prototype benchmarks: Table III & Table IV

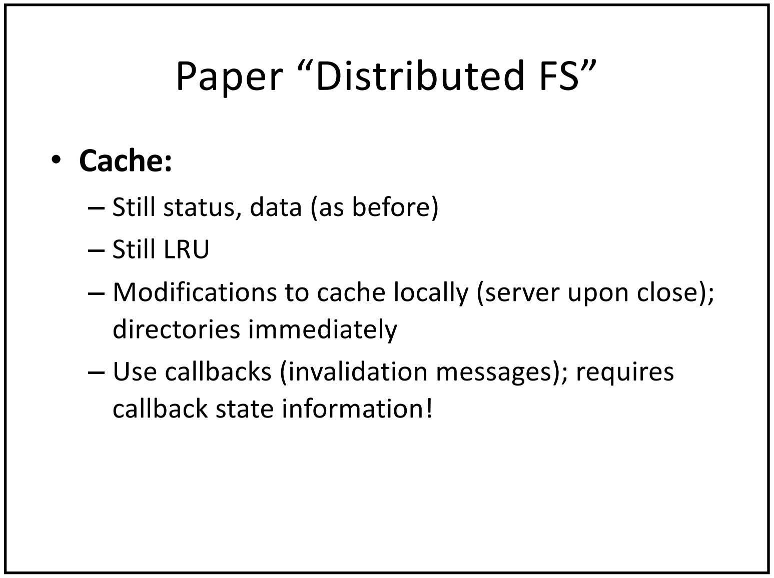
12



Paper “Distributed FS”

- New version:
  - – Keep whole-file caching
  - – Keep RPC
  - – Keep Vice/Venus as user-level processes

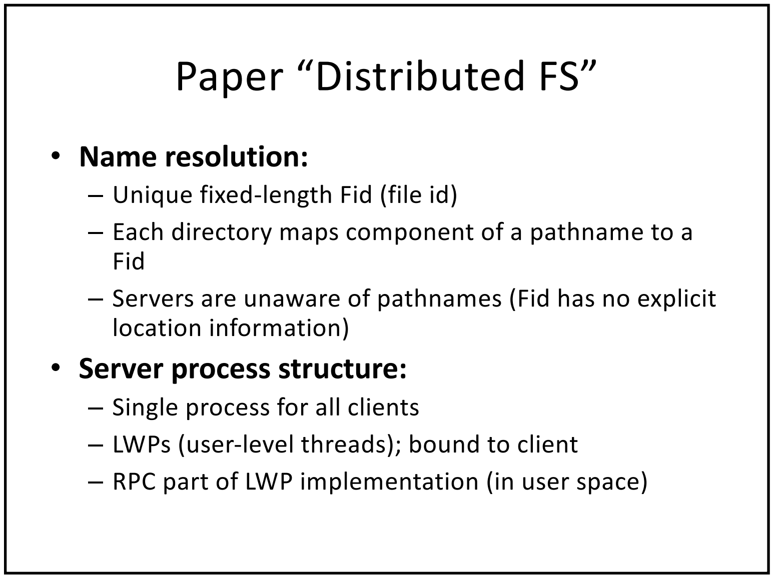
13



Paper “Distributed FS”

- **Cache:**
  - – Still status, data (as before)
  - – Still LRU
  - – Modifications to cache locally (server upon close); directories immediately
  - – Use callbacks (invalidation messages); requires callback state information!

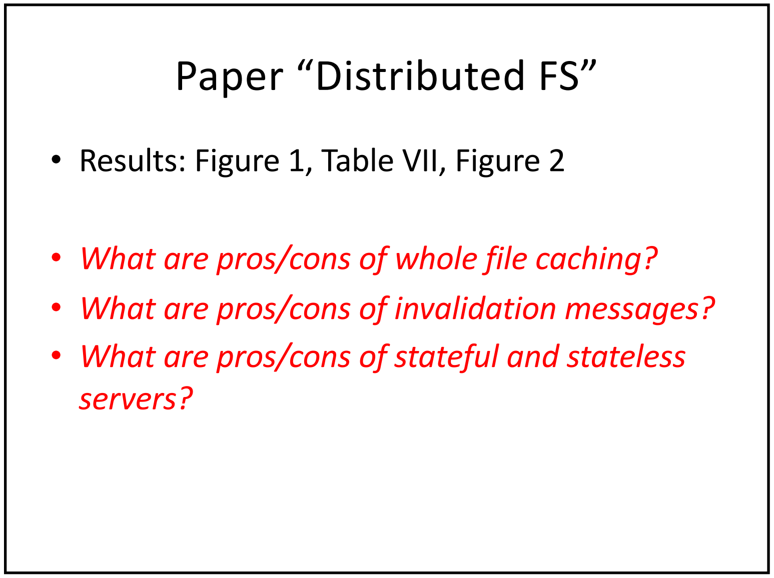
14



Paper “Distributed FS”

- **Name resolution:**
  - Unique fixed-length Fid (file id)
  - Each directory maps component of a pathname to a Fid
  - Servers are unaware of pathnames (Fid has no explicit location information)
- **Server process structure:**
  - Single process for all clients
  - LWPs (user-level threads); bound to client
  - RPC part of LWP implementation (in user space)

15



Paper “Distributed FS”

- Results: Figure 1, Table VII, Figure 2

- *What are pros/cons of whole file caching?*
- *What are pros/cons of invalidation messages?*
- *What are pros/cons of stateful and stateless servers?*

16