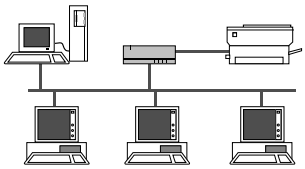


Operating Systems
600.418

Distributed Operating Systems

Department of Computer Science
The Johns Hopkins University

Distributed Operating Systems

Lecture 10

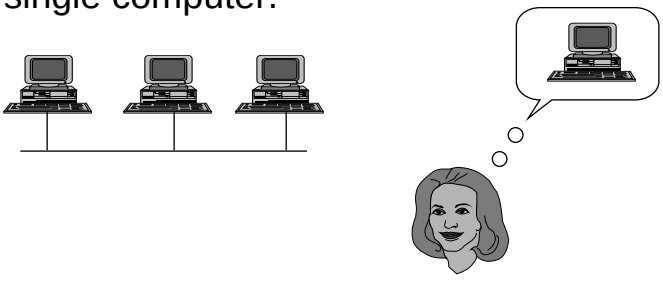
Reading:
- Silberschatz & Galvin
  chapter 16, 17

Additional Reading:
- Stallings
  chapter 13, 14

Final Exam
Thursday, October 30,
10am - 12pm and
11am - 1pm
MD 214

A Distributed Operating System

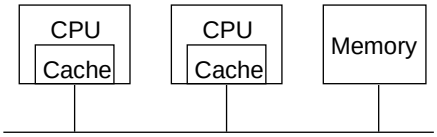
Goal: An operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.



Hardware Architectures

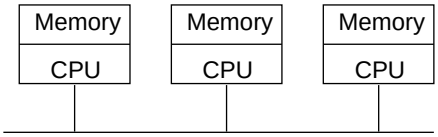
- Multiprocessors
  - **Tightly coupled.**
  - **Shared memory.**

Parallel architecture



- Multicomputers.
  - **Loosely coupled.**
  - **Private memory.**
  - **Autonomous.**

Distributed architecture



Software Architectures

- Multiprocessor OS
  - Looks like a virtual uniprocessor, contains only one copy of the operating system, communication via shared memory, single run-queue.
- Network OS
  - Does not look like a virtual uniprocessor, contains n copies of the operating system, communication via shared files, n run-queues.
- Distributed OS
  - Looks like a virtual uniprocessor (more or less), contains n copies of the operating system, communication via messages, n run-queues.

Design Issues

- Transparency
  - **Location transparency.**
    - Location of processes.
    - Location of cpu's and other devices.
    - Location of files.
  - **Replication transparency (of files).**
  - **Concurrency transparency (the user should not notice the existence of other users).**
  - **Parallelism transparency (the user should write a serial program, and the compiler and OS will handle the rest) - not doable today.**

Design Issues (cont.)

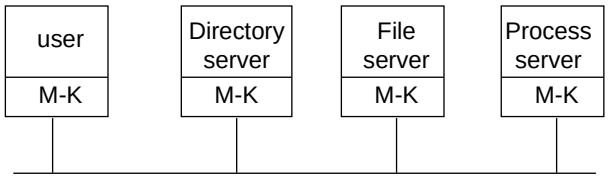
- Performance.
  - **Throughput / response time.**
  - **Load balancing (static/dynamic).**
  - **Communication is slow compared to processing speed : Fine grain / coarse grain.**
- Scalability.
- Reliability.
- Flexibility (Micro-kernel architecture).

Micro Kernel

Provides:

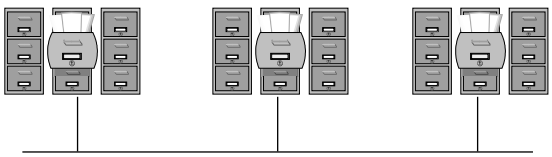
- Inter-process communication mechanisms.
- Low level I/O.
- Some memory management.
- Process management and scheduling.

All the rest is done in user level.



Distributed File Systems

- File and directory naming
- Semantics of file sharing
- Implementation considerations
  - Caching
  - Update protocols
  - Replication



File and Directory Naming

- Machine + path naming /machine/path.
  - **One name space, but not transparent.**
- Mounting remote file systems onto the local file hierarchy.
  - **The view of the file system** may be **different at each computer.**
- A single name space that looks the same on all machines.
  - **Full naming transparency.**
  - **If a server is down, its files will not be accessible.**

File Sharing Semantics

- One-copy semantics (unix semantics).
  - **Updates are written to the single copy and are available immediately.**
- Serializability.
  - **Transaction semantics (locking files - share for read and exclusive for write).**
- Session semantics.
  - **Copy the file on open, work on local copy, and copy back on close.**

Remote Caching

- Reduces network traffic by retaining recently accessed disk blocks in a cache - repeated accesses to the same information is handled locally:
  - **If needed data not already cached, a copy is brought from the server to the user.**
  - **Accesses are performed on the cached copy.**
  - **Files identified with one master copy residing on the server machine, but copies of (parts of) the file are scattered in different caches.**
  - *Cache-consistency* problem **– keeping the cached copies consistent with the master file.**

Cache Update Policy

- Write-through:
  - **write data through to the master disk as soon as they are placed on any cache.**
  - **Reliable, but poor performance.**
- Delayed-write
  - **modifications written to the cache and then written through to the server later.**
  - **Fast: some data may be overwritten before they are written back, and so need never be written at all.**
  - **Poor reliability: unwritten data will be lost whenever a user machine crashes.**
  - **Variations: write on close, update on regular intervals.**

Consistency

- Is locally cached copy of the data consistent with the master copy?
- Client-initiated approach
  - **The client initiates a validity check.**
  - **The server checks whether the local data are consistent with the master copy.**
- Server-initiated approach
  - **The server records, for each client, the (parts of) files it caches.**
  - **When the server detects a potential inconsistency, it must react.**

Stateful File Service

- Mechanism.
  - **Client opens a file.**
  - **Server fetches information about the file from its disk, stores it in its memory, and gives the client a connection identifier unique to the client and the open file.**
  - **Identifier is used for subsequent accesses until the session ends.**
  - **Server must reclaim the main-memory space used by clients who are no longer active.**
- Increased performance.
  - **Fewer disk accesses.**
  - **Stateful server knows if a file was opened for sequential access and can thus read ahead the next blocks.**

Stateless File Server

- Avoids state information by making each request self-contained.
- Each request identifies the file and position in the file.
- No need to establish and terminate a connection by open and close operations.

Stateful & Stateless Comparison

- Failure Recovery.
  - **A stateful server loses all its volatile state in a crash.**
    - Restore state using a recovery protocol based on a dialog with clients, or abort operations that were underway when the crash occurred.
    - Server needs to be aware of client failures in order to reclaim space allocated to record the state of crashed client processes (orphan detection and elimination).
  - **With stateless server, the effects of server failure and recovery are almost unnoticeable. A newly reincarnated server can respond to a self-contained request without any difficulty.**

Stateful & Stateless Comparison (cont.)

- Penalties for using the robust stateless service:
  - **Longer request messages.**
  - **Slower request processing.**
  - **Additional constraints imposed on DFS design - operations need to be idempotent (can be executed several times without problems) because of retransmissions.**
- Some environments require stateful service.
  - **A server employing server-initiated cache validation cannot provide stateless service, since it maintains a record of which files are cached by which clients.**

File Replication

- Replicas of the same file reside on failure-independent machines
  - Improves availability and can shorten service time.
- Naming scheme:
  - Maps a replicated file name to a particular replica.
  - Existence of replicas should be invisible to higher levels.
  - Replicas must be distinguished from one another by different lower-level names.
- Updates:
  - Replicas of a file denote the same logical entity - an update to any replica must be reflected on all other replicas.
- Demand replication:
  - Reading a nonlocal replica causes it to be cached locally, generating a new nonprimary replica.

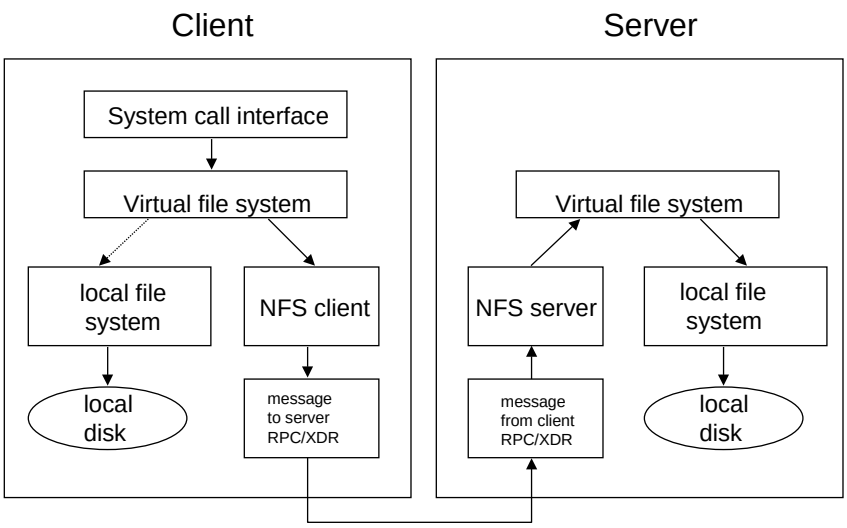
Sun-NFS

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs).
- Architecture:
  - **Server exports one or more directory trees for access by remote clients.**
  - **Clients access exported directory trees by mounting them to the client local tree.**
  - **Diskless clients can mount exported directory to their root directory.**
  - **Auto-mount (on the first access).**

Sun-NFS (cont.)

- Protocols
  - **Mounting protocol**
    - Affects the client state
    - Has no effect on the server.
  - **Directory and file accessing protocol**
    - Stateless
    - No open / close messages.
    - Each read / write message contains the full path and file description position.
- Semantics
  - **Not entirely Unix due to lack of locking mechanisms inside NFS (stateless).**

Sun-NFS Structure

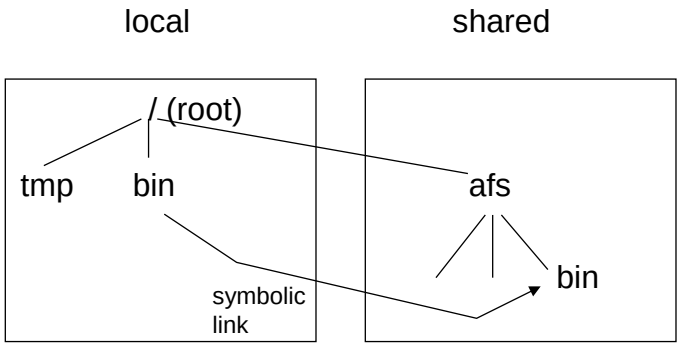


Andrew File System

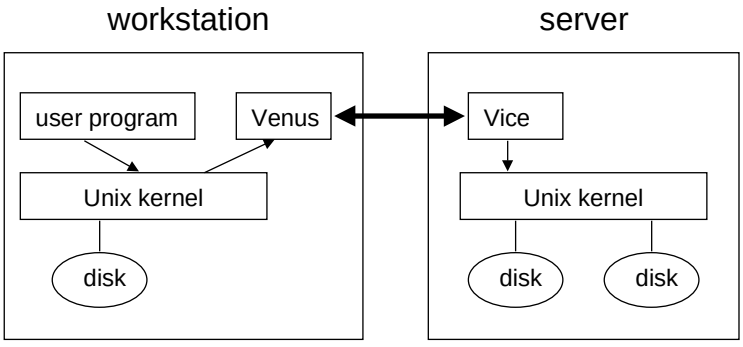
- Supports information sharing on a large scale (thousands of workstations).
- Uses a session semantics.
- The entire file is copied to the local machine (Venus) from the server (Vice) when open. If the file is changed - it is copied to the server when closed.

- The method works because in practice most files are changed by one person only (non-database).
- Measurements show that only 0.4% of changed files were updated by more than one user during one week.

Andrew File System (cont)



AFS Structure



AFS File Validation

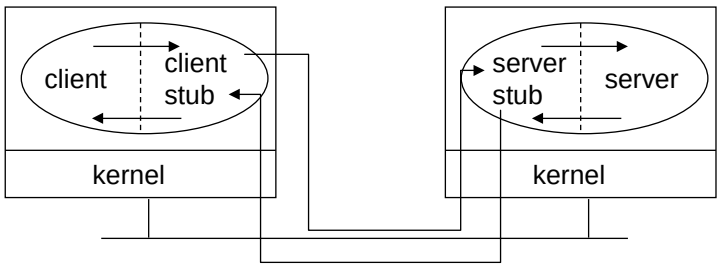
- Older AFS versions:
  - **On open: the Venus access the vice to see if its copy of the file is still valid. This causes a substantial delay even if the copy is valid.**
  - **The Vice is stateless.**
- Newer AFS versions:
  - **The Vice maintains lists of valid copies. If a file is modified the Vice invalidates other copies.**
  - **On open: if the Venus has a valid copy it can open it immediately.**
  - **If Venus crashes it has to invalidate its version or check their validity.**

The Coda File System

- Descendant of AFS that is substantially more resilient to server and network failures.
- Support for “mobile” users.
- Directories are replicated in several servers (Vice)
- When the Venus is disconnected, it uses local versions of files. When Venus reconnects, it reintegrates using optimistic update scheme.

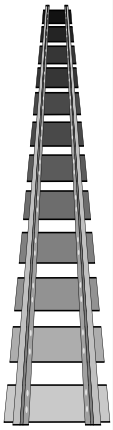
Remote Procedure Call

A convenient way to construct a client-server connection without explicitly writing send/ receive type programs (helps maintain transparency).



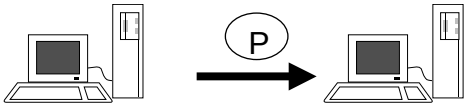
Remote Procedure Call (cont.)

- Client procedure **calls** the client stub in a normal way
- Client stub **builds** a message and **traps** to the kernel
- Kernel **sends** the message to remote kernel
- Remote kernel **gives** the message to server stub
- Server stub **unpacks** parameters and **calls** the server
- Server **computes** results and **returns** it to server stub
- Server stub **packs** results in a message and **traps** to kernel
- Remote kernel **sends** message to client kernel
- Client kernel **gives** message to client stub
- Client stub **unpacks** results and **returns** to client



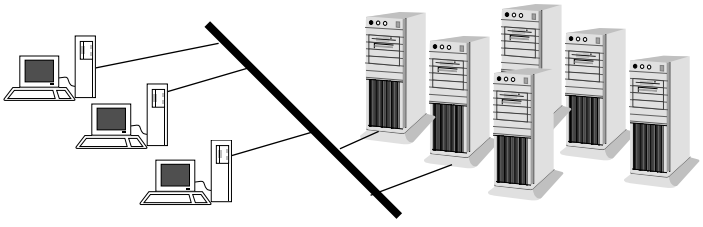
Process Migration

- Load Balancing
  - Static load balancing - CPU is determined at process creation.
  - Dynamic load balancing - processes dynamically migrate to other computers to balance the CPU (or memory) load.
- Migration architecture
  - One image system
  - point of entrance dependent system (the deputy concept)

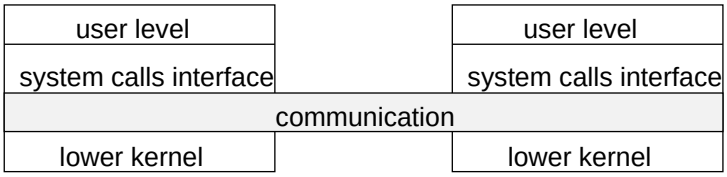


A Mosix Cluster

- Mosix (from Hebrew U): Kernel level enhancement to Linux that provides dynamic load balancing in a network of workstations.
- Dozens of PC computers connected by local area network (Fast-Ethernet or Myrinet).
- Any process can migrate anywhere anytime.



An Architecture for Migration

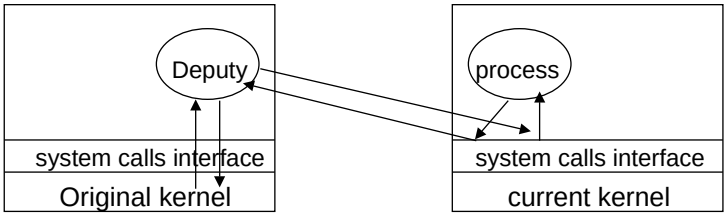


Architecture that fits one system image.
Needs location transparent file system.

(Mosix previous versions)

Architecture for Migration (cont.)

The process migrated from the original computer to the current computer and now is initiating a system call.



- Architecture that fits entrance dependent system.
- Easier to implement based on current Unix.

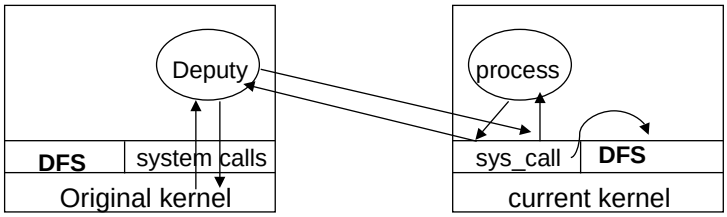
(Mosix current versions)

Mosix: File Access

Each file access must go back to deputy…

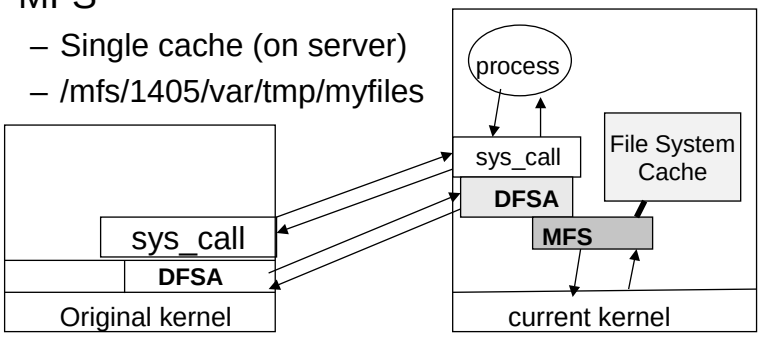
== Very Slow for I/O apps.

Optimization: Allow processes to access a distributed file system through the local kernel.

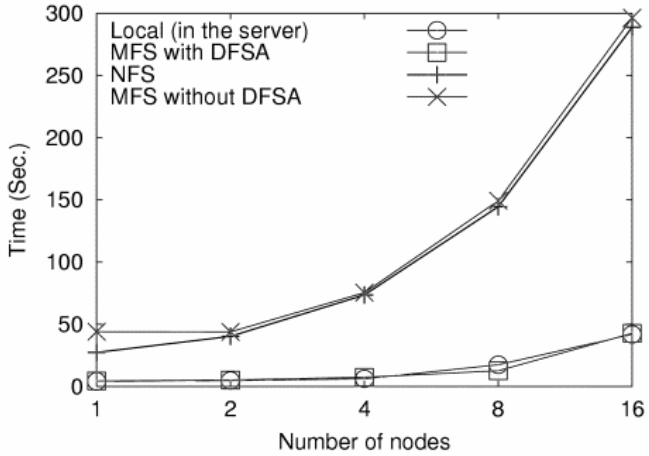


Mosix: File Access

- DFSA
  - Requirements (cache coherent, monotonic timestamps, files not deleted until all nodes finished)
  - Bring the process to the files.
- MFS
  - Single cache (on server)
  - /mfs/1405/var/tmp/myfiles



Mosix: DFSA/MFS Performance

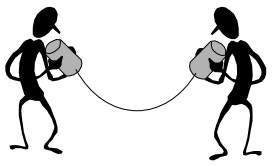


Network RAM

- Disk paging is very expensive, so why not use other machines' memory in order to implement paging.
- Access time to disk - 10 millisecond.
- Access time to a machine - 1 millisecond.
- Ability to run a program that is larger than any of the machines' memory, without going to disk.

Management Capabilities in Distributed Operating Systems

- For management purposes, “one system image” view, (even if it is not reflecting the architecture) is almost essential.
  - **Process control**
    - Show all the processes in all of the machines, track specific processes where ever they are, etc.
  - **Device management.**
    - Manage the disks in the system regardless of where they are hooked.
    - Logical terminals.
  - **Unified access control.**

Other Considerations for Migration

- Not only CPU load!!!
- Memory.
- I/O - where is the physical device?
  - **Bring the process to the data.**
- Communication - which processes communicate with which other processes ?