Advanced Operating Systems
Lecture notes

Dr. Clifford Neuman

University of Southern California
Information Sciences Institute

CSci555: Advanced Operating Systems

Lecture 8,9 – October 19, 26 2012

File Systems

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

File Systems

- **Provide set of primitives that abstract users from details of storage access and management.**

Distributed File Systems

- **Promote sharing across machine boundaries.**
- **Transparent access to files.**
- **Make diskless machines viable.**
- **Increase disk space availability by avoiding duplication.**
- **Balance load among multiple servers.**

Sun Network File System 1

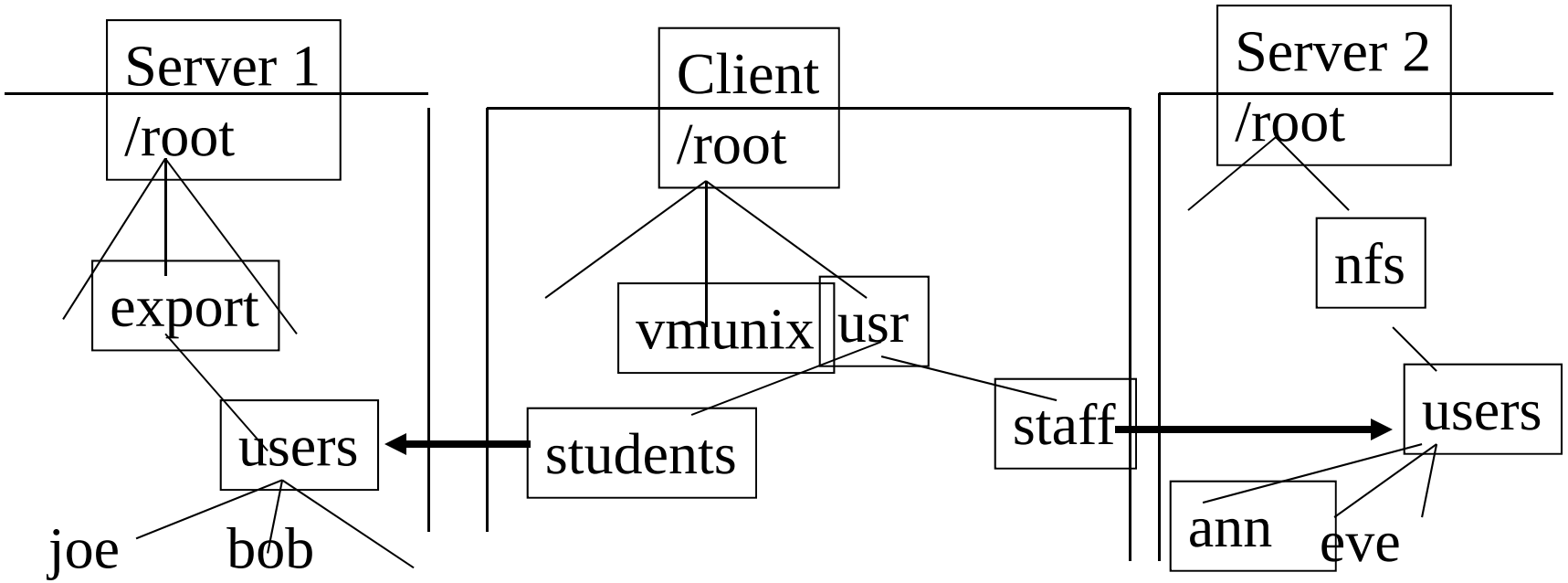
- **De facto standard:**
  - **Mid 80's.**
  - **Widely adopted in academia and industry.**
- **Provides transparent access to remote files.**
- **Uses Sun RPC and XDR.**
  - **NFS protocol defined as set of procedures and corresponding arguments.**
  - **Synchronous RPC**

Sun NFS 2

- **Stateless server:**
  - **Remote procedure calls are self-contained.**
  - **Servers don't need to keep state about previous requests.**
    - **Flush all modified data to disk before returning from RPC call.**
  - **Robustness.**
    - **No state to recover.**
    - **Clients retry.**

Location Transparency

- **Client's file name space includes remote files.**
  - **Shared remote files are *exported* by server.**
  - **They need to be *remote-mounted* by client.**



Achieving Transparency 1

- **Mount service.**
  - **Mount remote file systems in the client's local file name space.**
  - **Mount service process runs on each node to provide RPC interface for mounting and unmounting file systems at client.**
  - **Runs at system boot time or user login time.**

Achieving Transparency 2

- **Automounter.**
  - **Dynamically mounts file systems.**
  - **Runs as user-level process on clients (daemon).**
  - **Resolves references to unmounted pathnames by mounting them on demand.**
  - **Maintains a table of mount points and the corresponding server(s); sends probes to server(s).**
  - **Primitive form of replication**

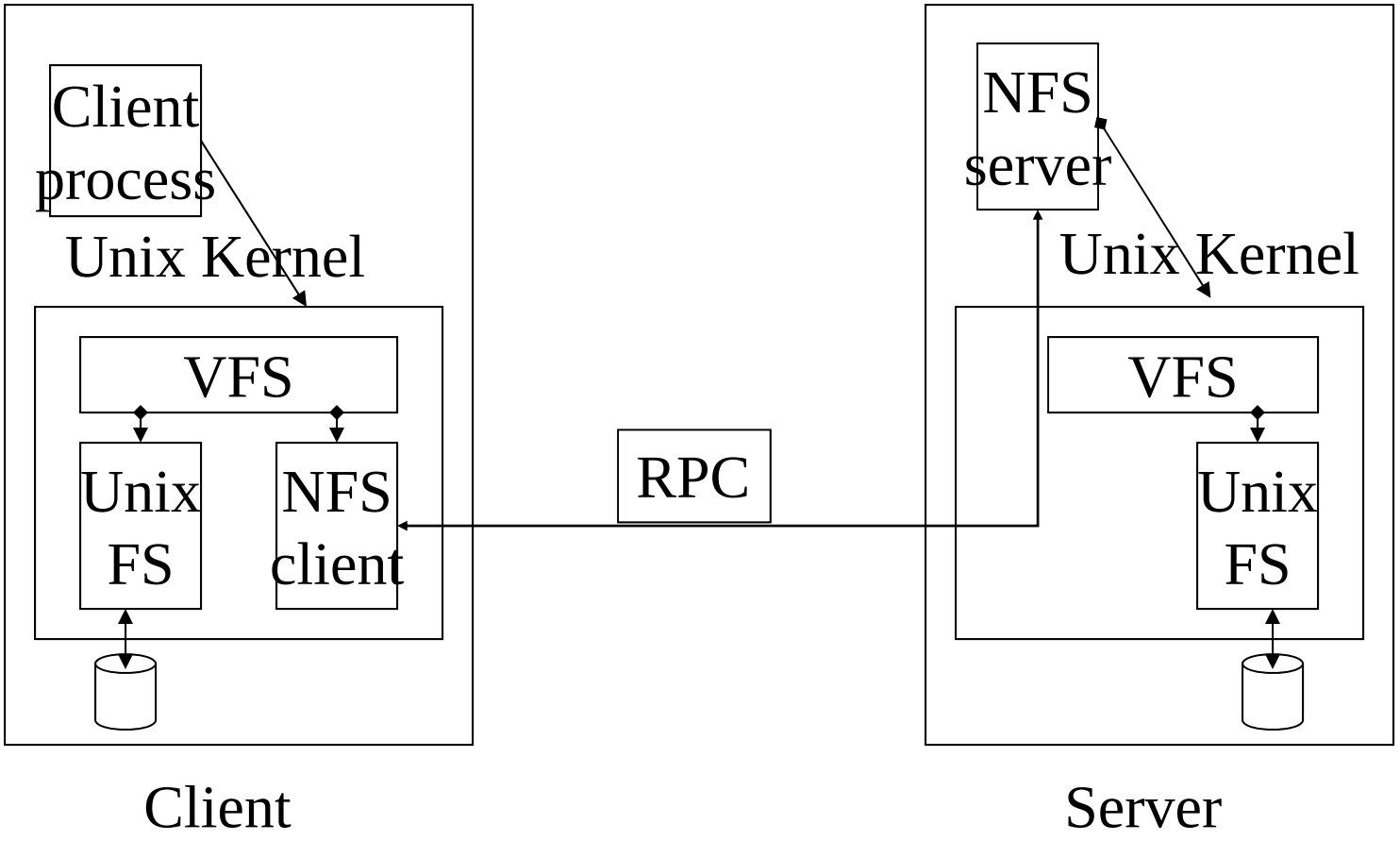
Transparency?

- **Early binding.**
  - **Mount system call attaches remote file system to local mount point.**
  - **Client deals with host name once.**
  - **But, mount needs to happen before remote files become accessible.**

Other Functions

- **NFS file and directory operations:**
  - **read, write, create, delete, getattr, etc.**
- **Access control:**
  - **File and directory access permissions.**
- **Path name translation:**
  - **Lookup for each path component.**
  - **Caching.**

Implementation

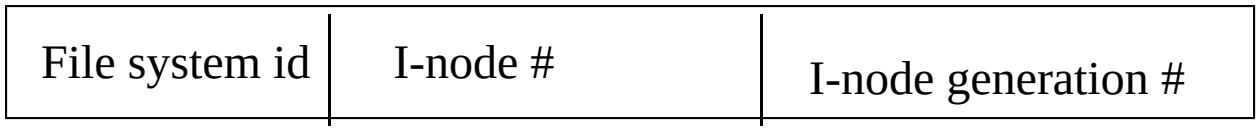


Virtual File System

- **VFS added to UNIX kernel.**
  - **Location-transparent file access.**
  - **Distinguishes between local and remote access.**
- **@ client:**
  - **Processes file system system calls to determine whether access is local (passes it to UNIX FS) or remote (passes it to NFS client).**
- **@ server:**
  - **NFS server receives request and passes it to local FS through VFS.**

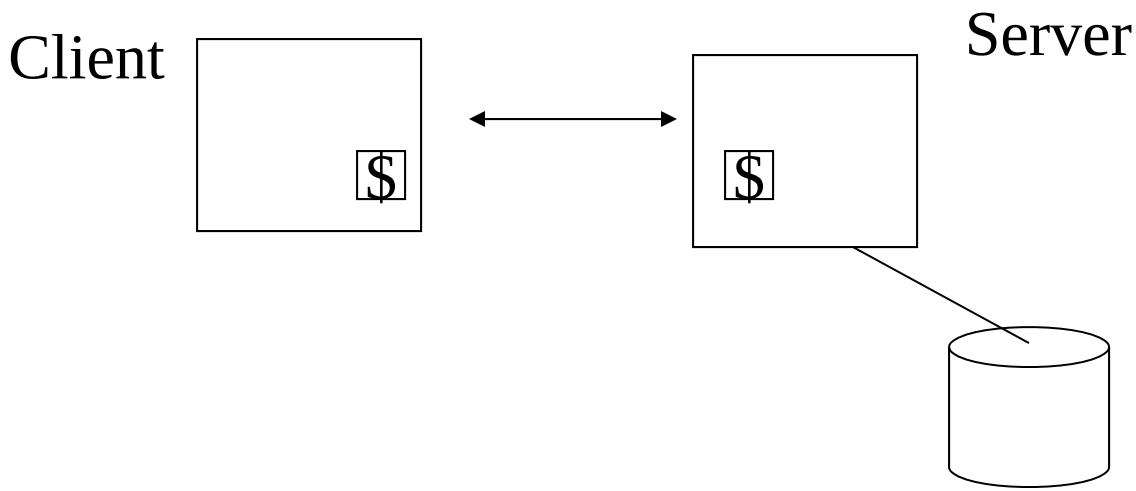
VFS

- **If local, translates file handle to internal file id's (in UNIX i-nodes).**
- **V-node:**
  - **If file local, reference to file's i-node.**
  - **If file remote, reference to file handle.**
- **File handle: uniquely distinguishes file.**



NFS Caching

- **File contents and attributes.**
- **Client versus server caching.**



Server Caching

- **Read:**
  - **Same as UNIX FS.**
  - **Caching of file pages and attributes.**
  - **Cache replacement uses LRU.**
- **Write:**
  - **Write through (as opposed to delayed writes of conventional UNIX FS). Why?**
  - **[Delayed writes: modified pages written to disk when buffer space needed, sync operation (every 30 sec), file close].**

Client Caching 1

- **Timestamp-based cache invalidation.**
- **Read:**
  - **(T-T*c* < *TTL*) V (T*m*c=T*m*s)**
  - **Cached entries have TS with last-modified time.**
  - **Blocks assumed to be valid for TTL.**
    - **TTL specified at mount time.**
    - **Typically 3 sec for files.**

Client Caching 1

- **Timestamp-based cache validation.**
- **Read:**
  - **Validity condition:**

    (T-T*c* < *TTL*) V (T*m*$_c$=T*m*$_s$)

- **Write:**
  - **Modified pages marked and flushed to server at file close or sync.**

Client Caching 2

- **Consistency?**
  - **Not always guaranteed!**
  - **e.g., client modifies file; delay for modification to reach servers + 3-sec (TTL) window for cache validation from clients sharing file.**

Cache Validation

- **Validation check performed when:**
  - **First reference to file after TTL expires.**
  - **File open or new block fetched from server.**
- **Done for all files, even if not being shared.**
  - **Why?**
- **Expensive!**
  - **Potentially, every 3 sec get file attributes.**
  - **If needed invalidate all blocks.**
  - **Fetch fresh copy when file is next accessed.**

The Sprite File System

- **Main memory caching on both client and server.**
- **Write-sharing consistency guarantees.**
- **Variable size caches.**
  - **VM and FS negotiate amount of memory needed.**
  - **According to caching needs, cache size changes.**

Sprite

- **Sprite supports concurrent writes by disabling caching of write-shared files.**
  - **If file shared, server notifies client that has file open for writing to write modified blocks back to server.**
  - **Server notifies all client that have file open for read that file is no longer cacheable; clients discard all cached blocks, so access goes through server.**

Sprite

- **Sprite servers are stateful.**
  - **Need to keep state about current accesses.**
  - **Centralized points for cache consistency.**
    - **Bottleneck?**
    - **Single point of failure?**
- **Tradeoff: consistency versus performance/robustness.**

Andrew

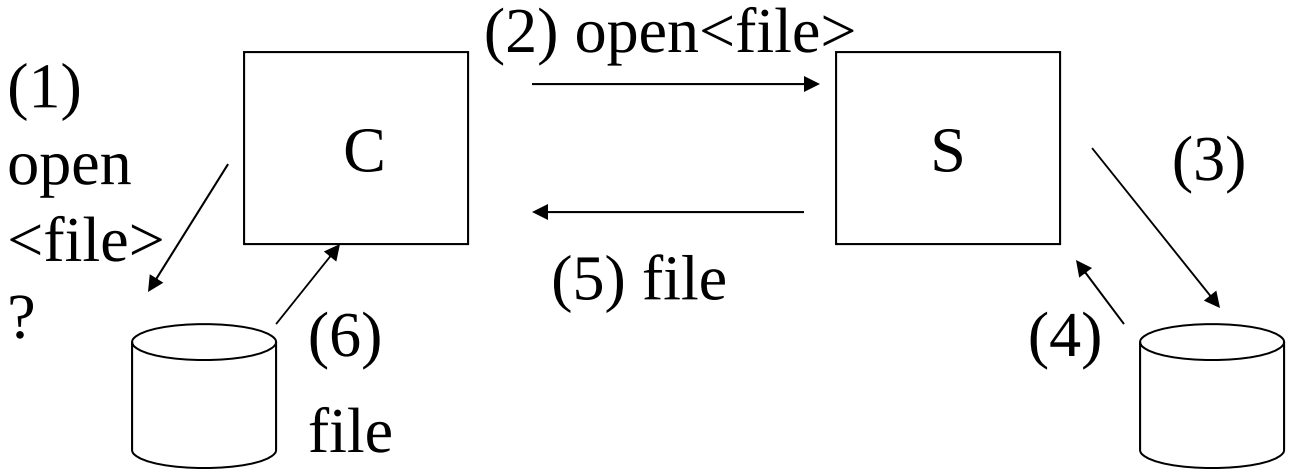
- **Distributed computing environment developed at CMU.**
- **Campus wide computing system.**
  - **Between 5 and 10K workstations.**
  - **1991: ~ 800 workstations, 40 servers.**

Andrew FS

- **Goals:**
  - **Information sharing.**
  - **Scalability.**
    - **Key strategy: caching of** whole **files at client.**
    - **Whole file serving**
      - **– Entire file transferred to client.**
    - **Whole file caching**
      - **– Local copy of file cached on client's local disk.**
      - **– Survive client's reboots and server unavailability.**

Whole File Caching

- **Local cache contains several most recently used files.**

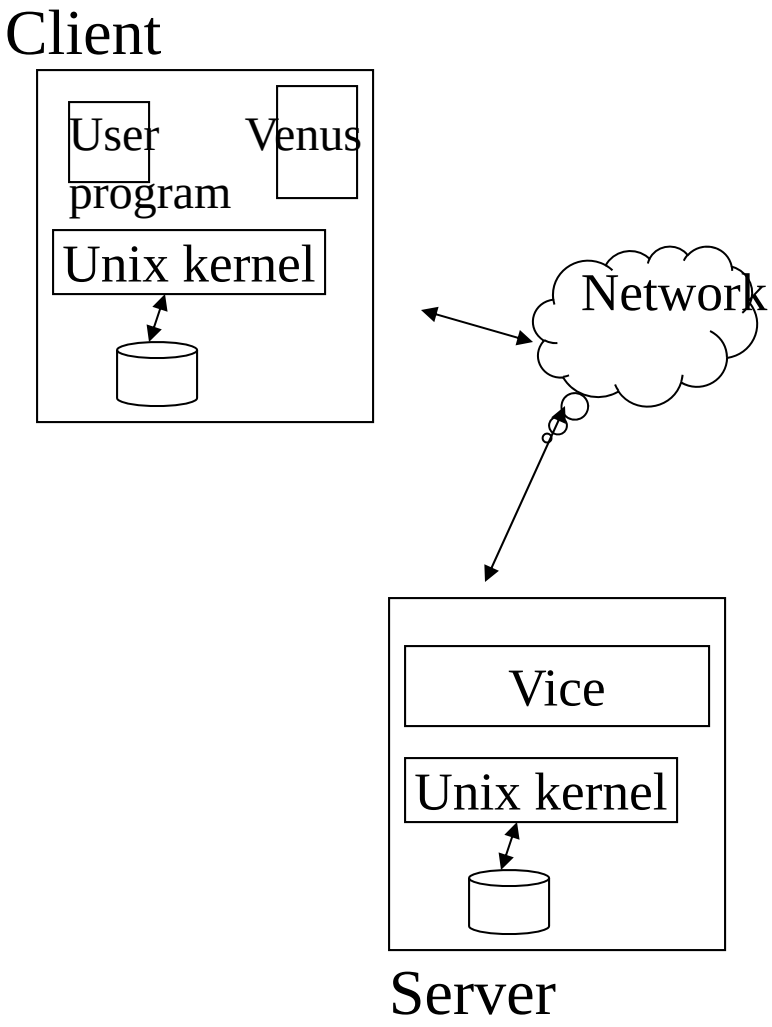


- Subsequent operations on file applied to local copy.
- On close, if file modified, sent back to server.

Implementation 1

- **Network of workstations running Unix BSD 4.3 and Mach.**
- **Implemented as 2 user-level processes:**
  - **Vice: runs at each Andrew server.**
  - **Venus: runs at each Andrew client.**

Implementation 2

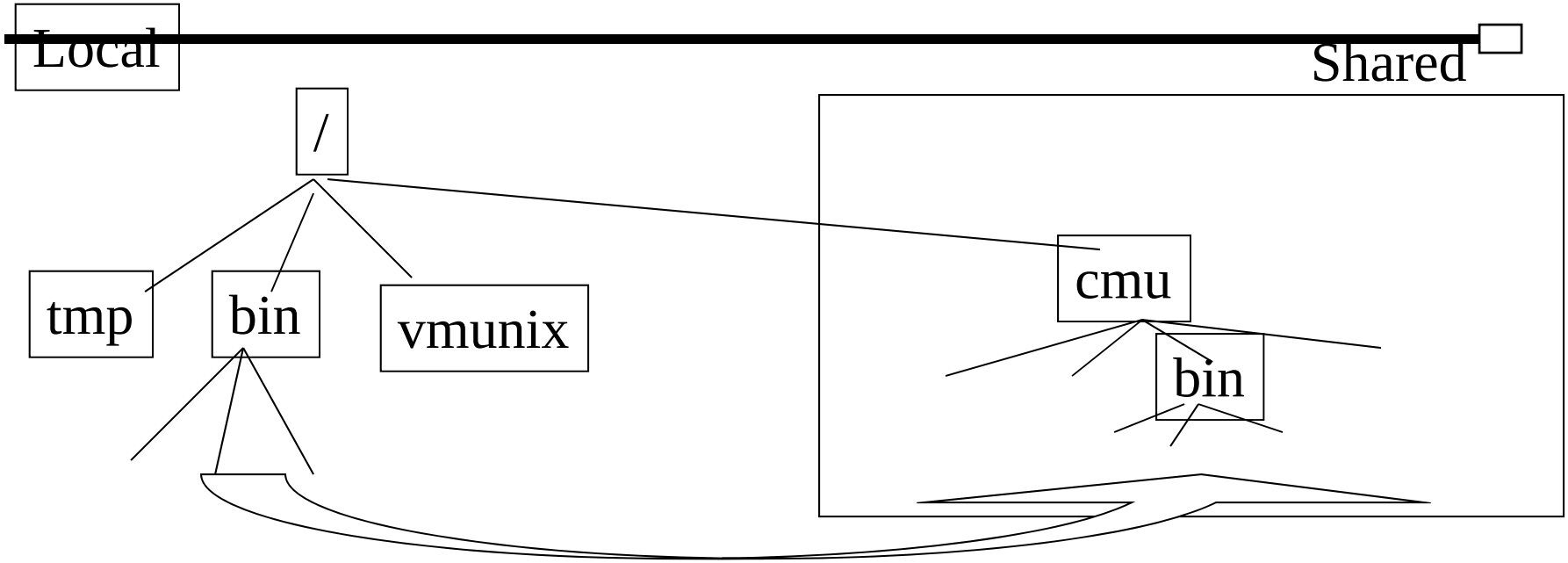


- **Modified BSD 4.3 Unix kernel.**
  - **At client, intercept file system calls (open, close, etc.) and pass them to Venus when referring to shared files.**
- **File partition on local disk used as cache.**
- **Venus manages cache.**
  - **LRU replacement policy.**
  - **Cache large enough to hold 100's of average-sized files.**

File Sharing

- **Files are *shared* or *local*.**
  - **Shared files**
    - **Utilities (/bin, /lib): infrequently updated or files accessed by single user (user's home directory).**
    - **Stored on servers and cached on clients.**
    - **Local copies remain valid for long time.**
  - **Local files**
    - **Temporary files (/tmp) and files used for start-up.**
    - **Stored on local machine's disk.**

File Name Space



- **Regular UNIX directory hierarchy.**
- **“cmu” subtree contains shared files.**
- **Local files stored on local machine.**
- **Shared files: symbolic links to shared files.**

AFS Caching

- **AFS-1 uses timestamp-based cache invalidation.**
- **AFS-2 and 3 use *callbacks*.**
  - **When serving file, Vice server promises to notify Venus client when file is modified.**
  - **Stateless servers?**
  - **Callback stored with cached file.**
    - **Valid.**
    - **Canceled: when client is notified by server that file has been modified.**

AFS Caching

- **Callbacks implemented using RPC.**
- **When accessing file, Venus checks if file exists and if callback valid; if canceled, fetches fresh copy from server.**
- **Failure recovery:**
  - **When restarting after failure, Venus checks each cached file by sending validation request to server.**
  - **Also periodic checks in case of communication failures.**

AFS Caching

- **At file close time, Venus on client modifying file sends update to Vice server.**
- **Server updates its own copy and sends callback cancellation to all clients caching file.**
- **Consistency?**
- **Concurrent updates?**

AFS Replication

- **Read-only replication.**
  - **Only read-only files allowed to be replicated at several servers.**

Coda

- Evolved from AFS.
- Goal: constant data availability.
  - Improved replication.
    - Replication of read-write volumes.
  - Disconnected operation: mobility.
    - Extension of AFS's whole file caching mechanism.
- Access to shared file repository (servers) versus relying on local resources when server not available.

Replication in Coda

- **Replication unit: file volume (set of files).**
- **Set of replicas of file volume: volume storage group (VSG).**
- **Subset of replicas available to client: AVSG.**
  - **Different clients have different AVSGs.**
  - **AVSG membership changes as server availability changes.**
  - **On write: when file is closed, copies of modified file broadcast to AVSG.**

Optimistic Replication

- **Goal is availability!**
- **Replicated files are allowed to be modified even in the presence of partitions or during disconnected operation.**

Disconnected Operation

- **AVSG = { }.**
- **Network/server failures or host on the move.**
- **Rely on local cache to serve all needed files.**
- **Loading the cache:**
  - **User intervention: list of files to be cached.**
  - **Learning usage patterns over time.**
- **Upon reconnection, cached copies validated against server's files.**

Normal and Disconnected Operation

- **During normal operation:**
  - **Coda behaves like AFS.**
  - **Cache miss transparent to user; only performance penalty.**
  - **Load balancing across replicas.**
  - **Cost: replica consistency + cache consistency.**
- **Disconnected operation:**
  - **No replicas are accessible; cache miss prevents further progress; need to load cache before disconnection.**

Replication and Caching

- **Coda integrates server replication and client caching.**
  - **On cache hit and valid data: Venus does not need to contact server.**
  - **On cache miss: Venus gets data from an AVSG server, i.e., the preferred server (PS).**
    - **PS chosen at random or based on proximity, load.**
  - **Venus also contacts other AVSG servers and collect their versions; if conflict, abort operation; if replicas stale, update them off-line.**

Next File Systems Topics

- **Leases**
  - **Continuum of cache consistency mechanisms.**
- **Log Structured File System and RAID.**
  - **FS performance from the storage management point of view.**

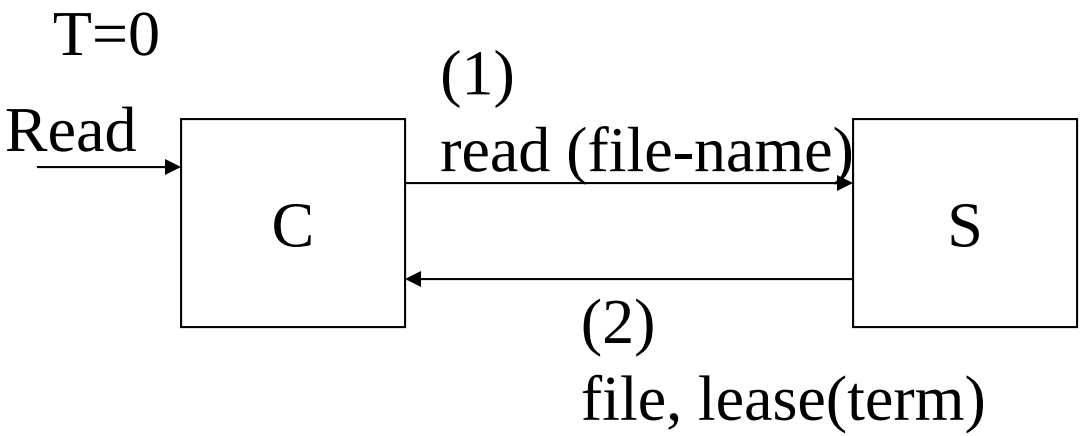
Caching

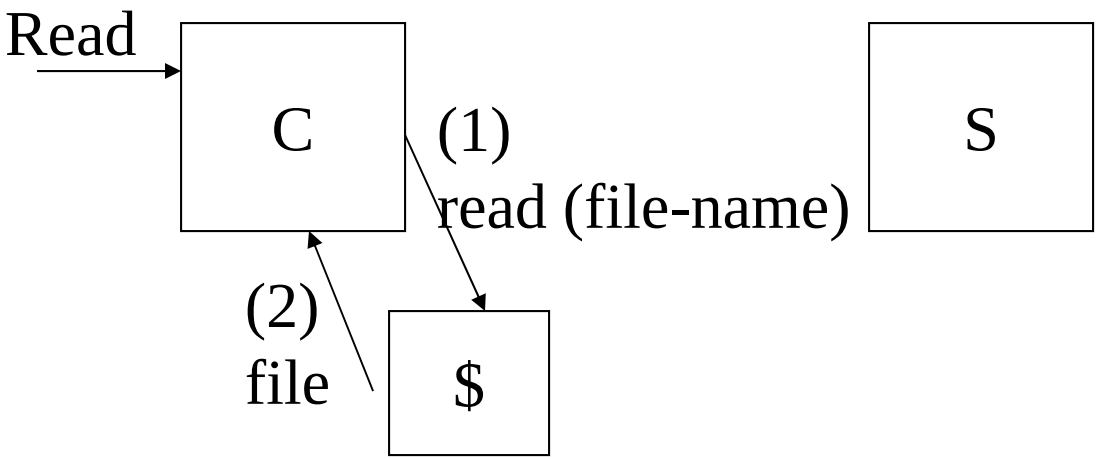
- **Improves performance in terms of response time, availability during disconnected operation, and fault tolerance.**
- **Price: consistency**
  - **Methods:**
    - **Timestamp-based invalidation**
      - **Check on use**
    - **Callbacks**

Leases

- **Time-based cache consistency protocol.**
- **Contract between client and server.**
  - **Lease grants holder control over writes to corresponding data item** during lease term**.**
  - **Server must obtain approval from holder of lease before modifying data.**
  - **When holder grants approval for write, it invalidates its local copy.**

Protocol Description 1

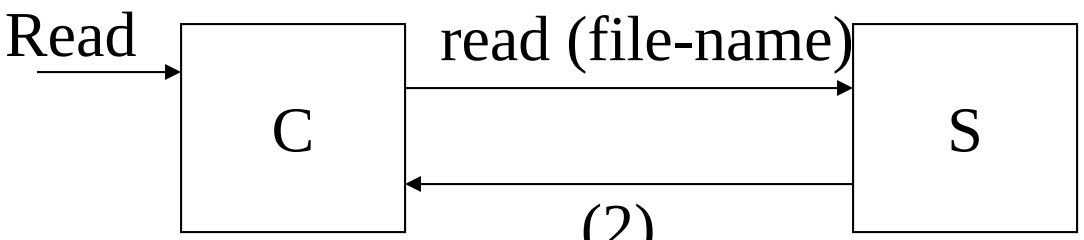




If file still in cache:
if lease is still valid, no need to go to server.

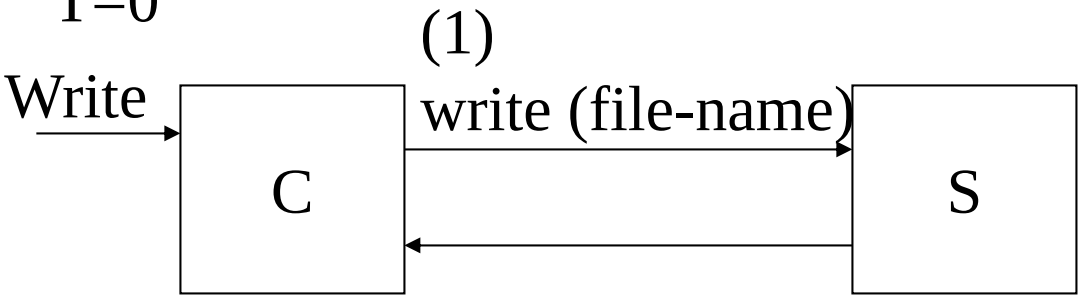
Protocol Description 2

T > term



On writes:

T=0



Server defers write request till: approval from lease holder(s) or lease expires.

Considerations

- **Unreachable lease holder(s)?**
- **Leases and callbacks.**
  - **Consistency?**
  - **Lease term**

Lease Term

- **Short leases:**
  - **Minimize delays due to failures.**
  - **Minimize impact of false sharing.**
  - **Reduce storage requirements at server (expired leases reclaimed).**
- **Long leases:**
  - **More efficient for repeated access with little write sharing.**

Lease Management 1

- **Client requests lease extension before lease expires in anticipation of file being accessed.**
  - **Performance improvement?**

Lease Management 2

- **Multiple files per lease.**
  - **Performance improvement?**
  - **Example: one lease per directory.**
  - **System files: widely shared but infrequently written.**
  - **False sharing?**
  - **Multicast lease extensions periodically.**

Lease Management 3

- **Lease term based on file access characteristics.**
  - **Heavily write-shared file: lease term = 0.**
  - **Longer lease terms for distant clients.**

Clock Synchronization Issues

- **Servers and clients should be roughly synchronized.**
  - **If server clock advances too fast or client's clock too slow: inconsistencies.**

Next...

- **Papers on file system performance from storage management perspective.**
- **Issues:**
  - **Disk access time >>> memory access time.**
  - **Discrepancy between disk access time improvements and other components (e.g., CPU).**
- **Minimize impact of disk access time by:**
  - **Reducing # of disk accesses or**
  - **Reducing access time by performing parallel access.**

Log-Structured File System

- **Built as extension to Sprite FS (Sprite LFS).**
- **New disk storage technique that tries to use disks more efficiently.**
- **Assumes main memory cache for files.**
- **Larger memory makes cache more efficient in satisfying reads.**
  - **Most of the working set is cached.**
- **Thus, most disk access cost due to writes!**

Main Idea

- **Batch multiple writes in file cache.**
  - **Transform many small writes into 1 large one.**
  - **Close to disk's full bandwidth utilization.**
- **Write to disk in one write in a contiguous region of disk called *log*.**
  - **Eliminates seeks.**
  - **Improves crash recovery.**
    - **Sequential structure of log.**
    - **Only most recent portion of log needs to be examined.**

LSFS Structure

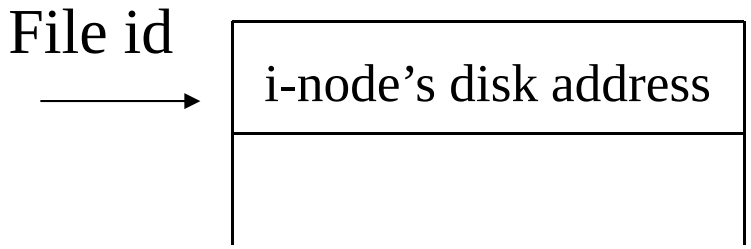
- **Two key functions:**
  - **How to retrieve information from log.**
  - **How to manage free disk space.**

File Location and Retrieval 1

- **Allows random access to information in the log.**
  - **Goal is to match or increase read performance.**
  - **Keeps indexing structures with log.**
- **Each file has i-node containing:**
  - **File attributes (type, owner, permissions).**
  - **Disk address of first 10 blocks.**
  - **Files > 10 blocks, i-node contains pointer to more data.**

File Location and Retrieval 2

- **In UNIX FS:**
  - **Fixed mapping between disk address and file i-node: disk address as function of file id.**
- **In LFS:**
  - **I-nodes written to log.**
  - **I-node map keeps current location of each i-node.**



  - **I-node maps usually fit in main memory cache.**

Free Space Management

- **Goal: maintain large, contiguous free chunks of disk space for writing data.**
- **Problem: fragmentation.**
- **Approaches:**
  - **Thread around used blocks.**
    - **Skip over active blocks and thread log through free extents.**
  - **Copying.**
    - **Active data copied in compacted form at head of log.**
    - **Generates contiguous free space.**
    - **But, expensive!**

Free Space Management in LFS

- **Divide disk into large, fixed-size segments.**
  - **Segment size is large enough so that transfer time (for read/write) >>> seek time.**
- **Hybrid approach.**
  - **Combination of threading and copying.**
  - **Copying: segment cleaning.**
  - **Threading between segments.**

Segment Cleaning

- **Process of copying “live” data out of segment before rewriting segment.**
- **Number of segments read into memory; identify live data; write live data back to smaller number of clean, contiguous segments.**
- **Segments read are marked as “clean”.**
- **Some bookkeeping needed: update files’ i-nodes to point to new block locations, etc.**

Crash Recovery

- **When crash occurs, last few disk operations may have left disk in inconsistent state.**
  - **E.g., new file written but directory entry not updated.**
- **At reboot time, OS must correct possible inconsistencies.**
- **Traditional UNIX FS: need to scan whole disk.**

Crash Recovery in Sprite LFS 1

- **Locations of last disk operations are at the end of the log.**
  - **Easy to perform crash recovery.**
- **2 recovery strategies:**
  - **Checkpoints and roll-forward.**
- **Checkpoints:**
  - **Positions in the log where everything is consistent.**

Crash Recovery in Sprite LFS 2

- **After crash, scan disk backward from end of log to checkpoint, then scan forward to recover as much information as possible: *roll forward*.**

More on LFS

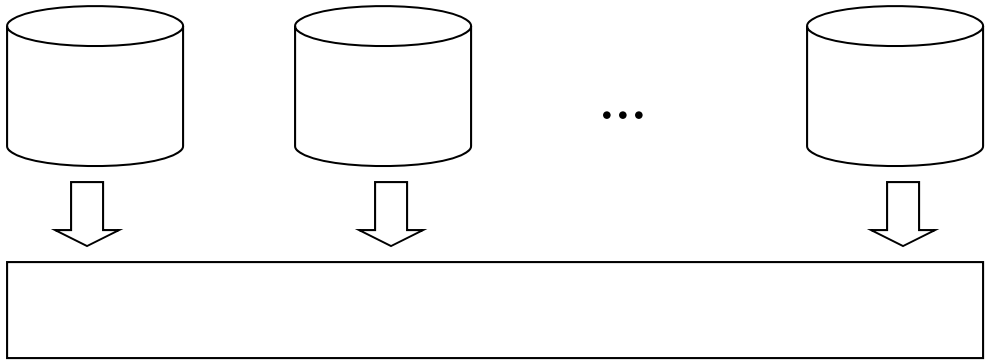
- **Paper talks about their experience implementing and using LFS.**
- **Performance evaluation using benchmarks.**
- **Cleaning overhead.**

Redundant Arrays of Inexpensive Disks (RAID)

- **Improve disk access time by using arrays of disks.**
- **Motivation:**
  - **Disks are getting inexpensive.**
  - **Lower cost disks:**
    - **Less capacity.**
    - **But cheaper, smaller, and lower power.**
- **Paper proposal: build I/O systems as arrays of inexpensive disks.**
  - **E.g., 75 inexpensive disks have 12 * I/O bandwidth of expensive disks with same capacity.**

RAID Organization 1

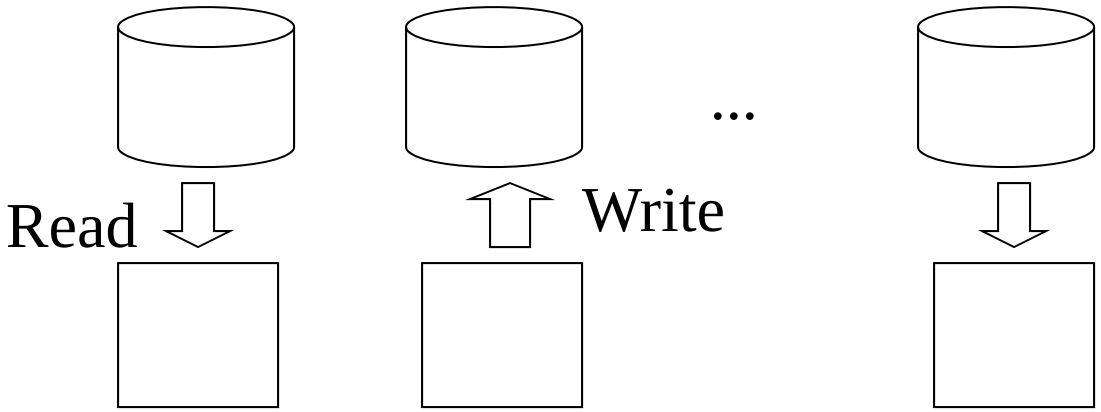
- **Interleaving disks.**
  - **Supercomputing applications.**
  - **Transfer of large blocks of data at high rates.**



Grouped read: single read spread over multiple disks

RAID Organization 2

- **Independent disks.**
  - **Transaction processing applications.**
  - **Database partitioned across disks.**
  - **Concurrent access to independent items.**



Problem: Reliability

- **Disk unreliability causes frequent backups.**
- **What happens with 100*number of disks?**
  - **MTTF becomes prohibitive**
  - **Fault tolerance otherwise disk arrays are too unreliable to be useful.**
- **RAID: use of extra disks containing redundant information.**
  - **Similar to redundant transmission of data.**

RAID Levels

- **Different levels provide different reliability, cost, and performance.**
- **MTTF as function of total number of disks, number of data disks in a group (G), number of check disks per group (C), and number of groups.**
- **C determined by RAID level.**

First RAID Level

- **Mirrors.**
  - **Most expensive approach.**
  - **All disks duplicated (G=1 and C=1).**
  - **Every write to data disk results in write to check disk.**
  - **Double cost and half capacity.**

Second RAID Level

- **Hamming code.**
- **Interleave data across disks in a group.**
- **Add enough check disks to detect/correct error.**
- **Single parity disk detects single error.**
- **Makes sense for large data transfers.**
- **Small transfers mean all disks must be accessed (to check if data is correct).**

Third RAID Level

- **Lower cost by reducing C to 1.**
  - **Single parity disk.**
- **Rationale:**
  - **Most check disks in RAID 2 used to detect which disks failed.**
  - **Disk controllers do that.**
  - **Data on failed disk can be reconstructed by computing the parity on remaining disks and comparing it with parity for full group.**

Fourth RAID Level

- **Try to improve performance of small transfers using parallelism.**
- **Transfer units stored in single sector.**
  - **Reads are independent, i.e., errors can be detected without having to use other disks (rely on controller).**
  - **Also, maximum disk rate.**
  - **Writes still need multiple disk access.**

Fifth RAID Level

- **Tries to achieve parallelism for writes as well.**
- **Distributes data as well as check information across all disks.**

The Google File System

- **Focused on special cases:**
  - **Permanent failure normal**
  - **Files are huge – aggregated**
  - **Few random writes – mostly append**
  - **Designed together with the application**
    - **And implemented as library**

The Google File System

- **Some requirements**
  - **Well defined semantics for concurrent append.**
  - **High bandwidth (more important than latency)**
  - **Highly scalable**
    - **Master handles meta-data (only)**

The Google File System

- **Chunks**
  - **Replicated**
    - **Provides location updates to master**
- **Consistency**
  - **Atomic namespace**
  - **Leases maintain mutation order**
  - **Atomic appends**
  - **Concurrent writes can be inconsistent**

CSci555: Advanced Operating Systems

Lecture 10 – November 2nd 2012

Case Studies: Locus, Athena, Andrew, HCS, others

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

The LOCUS System

- **Developed at UCLA in early 80's**
  - **Essentially a distributed Unix**
- **Major contribution was transparency**
  - **Transparency took many forms**
- **Environment:**
  - **VAX 750's and/or IBM PCs connected by an Ethernet**
- **UNIX compatible.**

LOCUS

- **Network/location transparency:**
  - **Network of machines appear as single machine to user.**
  - **Hide machine boundaries.**
  - **Local and remote resources look the same to user.**

Transparency in Locus

- **Network Transparency**
  - **Ability to hide boundaries**
- **Syntactic Transparency**
  - **Local and remote calls take same form**
- **Semantic Transparency**
  - **Independence from Operand Location**
- **Name Transparency**
  - **A name always refers to the same object**
  - **No need for closure, only one namespace**

Transparency in Locus (cont)

- **Location Transparency**
  - **Location can't be inferred from name**
  - **Makes it easier to move objects**
- **Syntactic Transparency**
  - **Local and remote calls take same form**
- **Performance Transparency**
  - **Programs with timing assumptions work**
- **Failure Transparency**
  - **Remote errors indistinguishable from local**
- **Execution Transparency**
  - **Results don't change with location**

LOCUS Distributed File System

- **Tree-structured file name space.**
  - **File name tree covers all file system objects in all machines.**
  - **Location transparency.**
  - **File groups (UNIX file systems) “glued” via mount.**
- **File replication.**
  - **Varying degrees of replication.**
  - **Locus responsible for consistency: propagate updates, serve from most up-to-date copy, and handle partitions.**

Replication in LOCUS

- **File group replicated at multiple servers.**
- **Replicas of a file group may contain different subsets of files belonging to that file group.**
- **All copies of file assigned same descriptor (i-node #).**
  - **File unique name: <file group#, i-node #).**

Replica Consistency

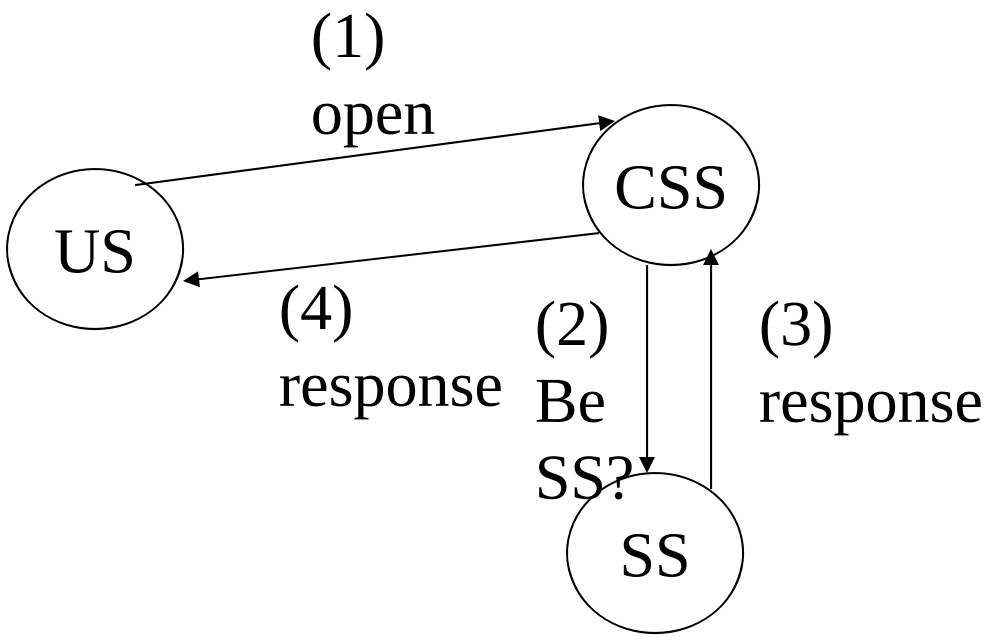
- **Version vectors.**
  - **Version vector associated with each copy of a file.**
  - **Maintain update history information.**
  - **Used to ensure latest copies will be used and to help updating outdated copies.**
  - **Optimistic consistency.**
    - **Potential inconsistencies.**

File System Operations 1

- **Using site (US): client.**
- **Storage site (SS): server.**
- **Current synchronization site (CSS): synchronization site; chooses the SS for a file request.**
  - **Knowledge of which files replicated where.**

File System Operations 2

❖**Open:**



File Modification

- **At US:**
  - **After each change, page sent to SS.**
  - **At file close, all modified pages flushed to SS.**
- **At SS: atomic commit.**
  - **Changes to a file handled atomically.**
  - **No changes are permanent until committed.**
  - ***Commit* and *abort* system calls.**
  - **At file close time, changes are committed.**
  - **Logging and shadow pages.**

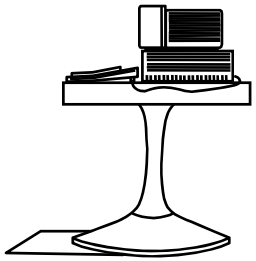
CSS

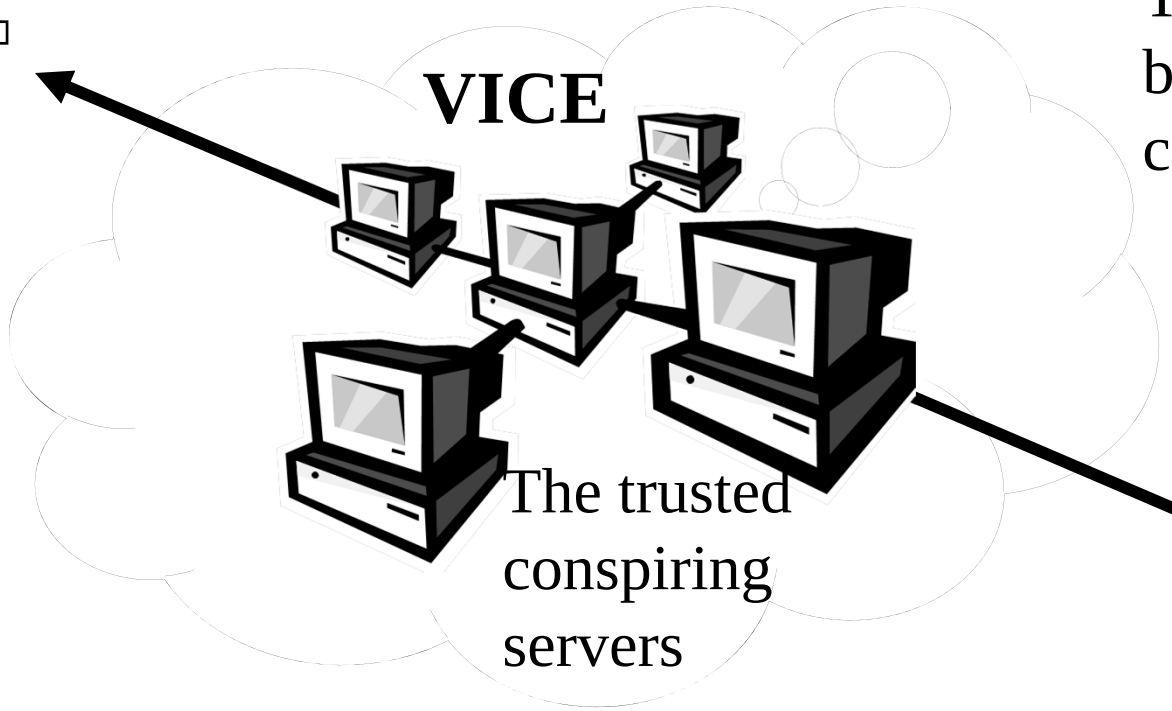
- **Can implement variety of synchronization policies.**
  - **Enforce them upon file access.**
  - **E.g., if sharing policy allows only read-only sharing, CSS disallows concurrent accesses.**

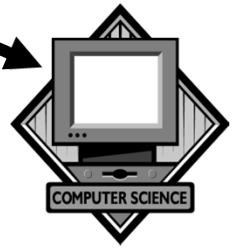
Andrew System

- Developed at CMU starting in 1982
  - With support from IBM
  - To get computers used as a tool in basic curriculum
- The 3M workstation
  - 1 MIP
  - 1 MegaPixel
  - 1 MegaByte
  - Approx $10K and 10 Mbps network, local disks

Vice and Virtue







VIRTUE

The untrusted, but independent clients

Andrew System (key contributions)

- **Network Communication**
  - **Vice (trusted)**
  - **Virtue (untrusted)**
  - **High level communication using RPC w/ authentication**
  - **Security has since switched to Kerberos**
- **The File System**
  - **AFS (led to DFS, Coda)**
- **Applications and user interface**
  - **Mail and FTP subsumed by file system (w/ gateways)**
- **Window manager**
  - **similar to X, but tiled**
  - **toolkits were priority**
  - **Since moved to X (and contributed to X)**

Project Athena

- Developed at MIT about same time
  - With support from DEC and IBM (and others)
    - MIT retained all rights
  - To get computers used as a tool in basic curriculum
- Heterogeneity
  - Equipment from multiple vendors
- Coherence
  - None
  - → Protocol
  - → Execution abstraction (e.g. programming environment)
  - Instruction set/binary

Mainframe/WS vs Unified Model (athena)

- **Unified model**
  - **Services provided by system as a whole**
- **Mainframe / Workstation Model**
  - **Independent hosts connected by e-mail/FTP**
- **Athena**
  - **Unified model**
  - **Centralized management**
  - **Pooled resources**
  - **Servers are not trusted (as much as in Andrew)**
  - **Clients and network not trusted (like Andrew)**

Project Athena - File system evolution

- **Remote Virtual Disk (RVD)**
  - **Remotely read and write blocks of disk device**
  - **Manage file system locally**
  - **Sharing not possible for mutable data**
  - **Very efficient for read only data**
- **Remote File System (RFS)**
  - **Remote execution of file system calls**
  - **Target host is part of argument (no syntactic transparency).**
- **SUN's Network File System (NFS) - covered**
- **The Andrew File System (AFS) - covered**

Project Athena - Other Services

- **Security**
  - **Kerberos**
- **Notification/location**
  - **Zephyr**
- **Mail**
  - **POP**
- **Printing/configuration**
  - **Hesiod-Printcap / Palladium**
- **Naming**
  - **Hesiod**
- **Management**
  - **Moira/RDIST**

- **Developed**
  - **University of Washington, late 1980s**
- **Why Heterogeneity**
  - **Organizational diversity**
  - **Need for capabilities from different systems**
- **Problems caused by heterogeneity**
  - **Need to support duplicate infrastructure**
  - **Isolation**
  - **Lack of transparency**

HCS Aproach

- **Common service to support heterogeneity**
  - **Common API for HCS systems**
  - **Accommodate multiple protocols**
- **Transparency**
  - **For new systems accessing existing systems**
  - **Not for existing systems**

HCS Subsystems

- **HRPC**
  - **Common API, modular organization**
  - **Bind time connection of modules**
- **HNS (heterogeneous name service)**
  - **Accesses data in existing name service**
  - **Maps global name to local lower level names**
- **THERE**
  - **Remote execution (by wrapping data)**
- **HFS (filing)**
  - **Storage repository**
  - **Description of data similar to RPC marshalling**

CORBA (Common Object Request Broker Architecture)

- **Distributed Object Abstraction**
  - **Similar level of abstraction as RPC**
- **Correspondence**
  - **IDL vs. procedure prototype**
  - **ORB supports binding**
  - **IR allows one to discover prototypes**
  - **Distributed Document Component Facility vs. file system**

Microsoft Cluster Service

- A case study in binding
  - The virtual service is a key abstraction
- Nodes claim ownership of resources
  - Including IP addresses
- On failure
  - Server is restarted, new node claims ownership of the IP resource associated with failed instance.
  - But clients must still retry request and recover.

CSci555: Advanced Operating Systems

Lecture 11 – November 9 2012

Kernels

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

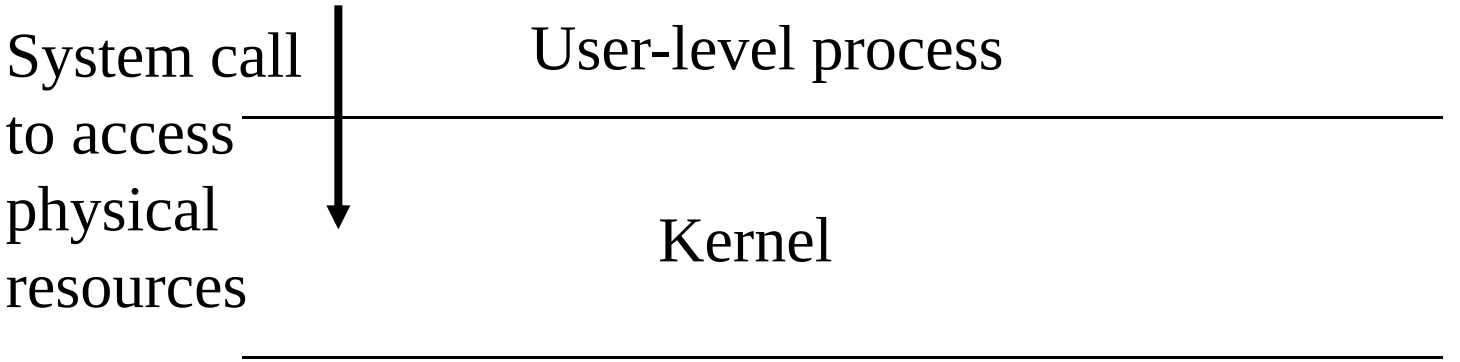
Kernels

- **Executes in supervisory mode.**
  - **Privilege to access machine's physical resources.**
- **User-level process: executes in "user" mode.**
  - **Restricted access to resources.**
  - **Address space boundary restrictions.**

Kernel Functions

- **Memory management.**
  - **Address space allocation.**
  - **Memory protection.**
- **Process management.**
  - **Process creation, deletion.**
  - **Scheduling.**
- **Resource management.**
  - **Device drivers/handlers.**

System Calls



System call: implemented by hardware interrupt (trap) which puts processor in supervisory mode and kernel address space; executes kernel-supplied handler routine (device driver) executing with interrupts disabled.

Kernel and Distributed Systems

- **Inter-process communication: RPC, MP, DSM.**
- **File systems.**
- **Some parts may run as user-level and some as kernel processes.**

Be or not to be in the kernel?

- **Monolithic kernels versus microkernels.**

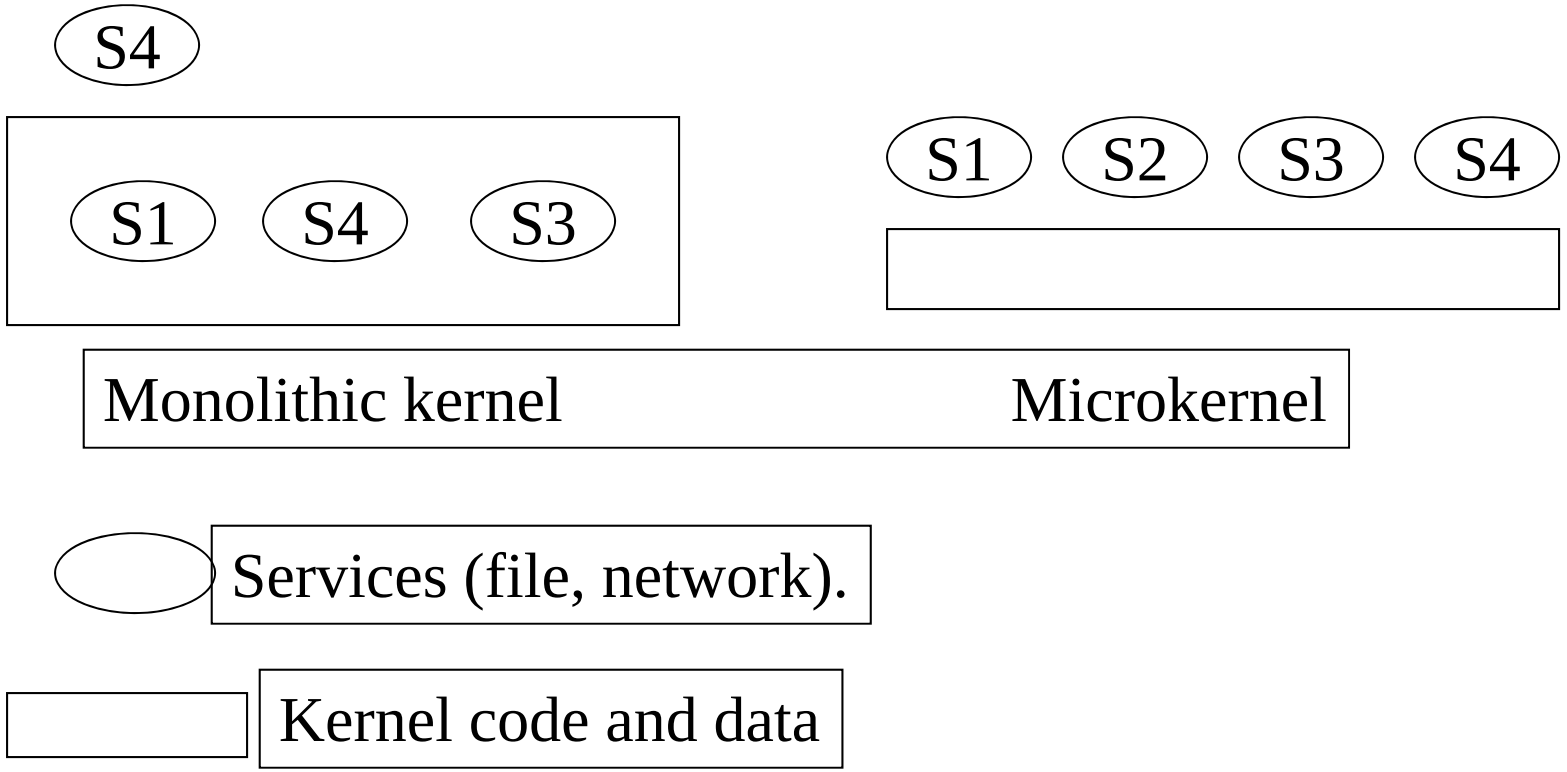
Monolithic kernels

- **Examples: Unix, Sprite.**
- **“Kernel does it all” approach.**
- **Based on argument that inside kernel, processes execute more efficiently and securely.**
- **Problems: massive, non-modular, hard to maintain and extend.**

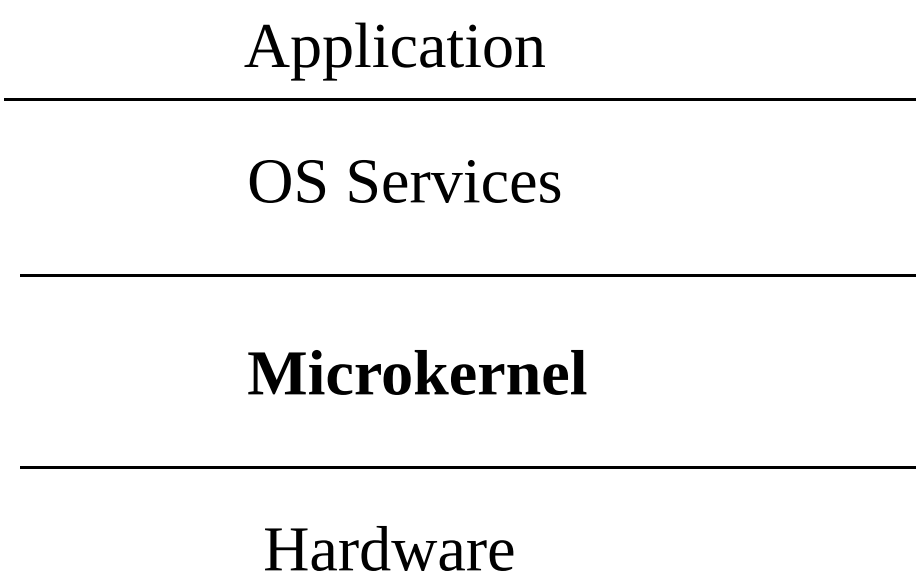
Microkernels

- **Take as much out of the kernel as possible.**
- **Minimalist approach.**
- **Modular and small.**
  - **10KBytes -> several hundred Kbytes.**
  - **Easier to port, maintain and extend.**
  - **No fixed definition of what should be in the kernel.**
  - **Typically process management, memory management, IPC.**

Micro- versus Monolithic Kernels



Microkernel



- Services dynamically loaded at appropriate servers.
- Some microkernels run service processes only @ user space; others allow them to be loaded into either kernel or user space.

The V Distributed System

- **Stanford (early 80's) by Cheriton et al.**
- **Distributed OS designed to manage cluster of workstations connected by LAN.**
- **System structure:**
  - **Relatively small kernel common to all machines.**
  - **Service modules: e.g., file service.**
  - **Run-time libraries: language support (Pascal I/O, C stdio)**
  - **Commands and applications.**

V's Design Goals

- **High performance communication.**
  - **Considered the most critical service.**
    - **Efficient file transfer.**
  - **"Uniform" protocol approach for open system interconnection.**
    - **Interconnect heterogeneous nodes.**
  - **"Protocols, not software, define the system".**

The V Kernel

- **Small kernel with basic protocols and services.**
- **Precursor to microkernel approach.**
- **Kernel as a “software backplane”.**
  - **Provides “slots” into which higher-level OS services can be “plugged”.**

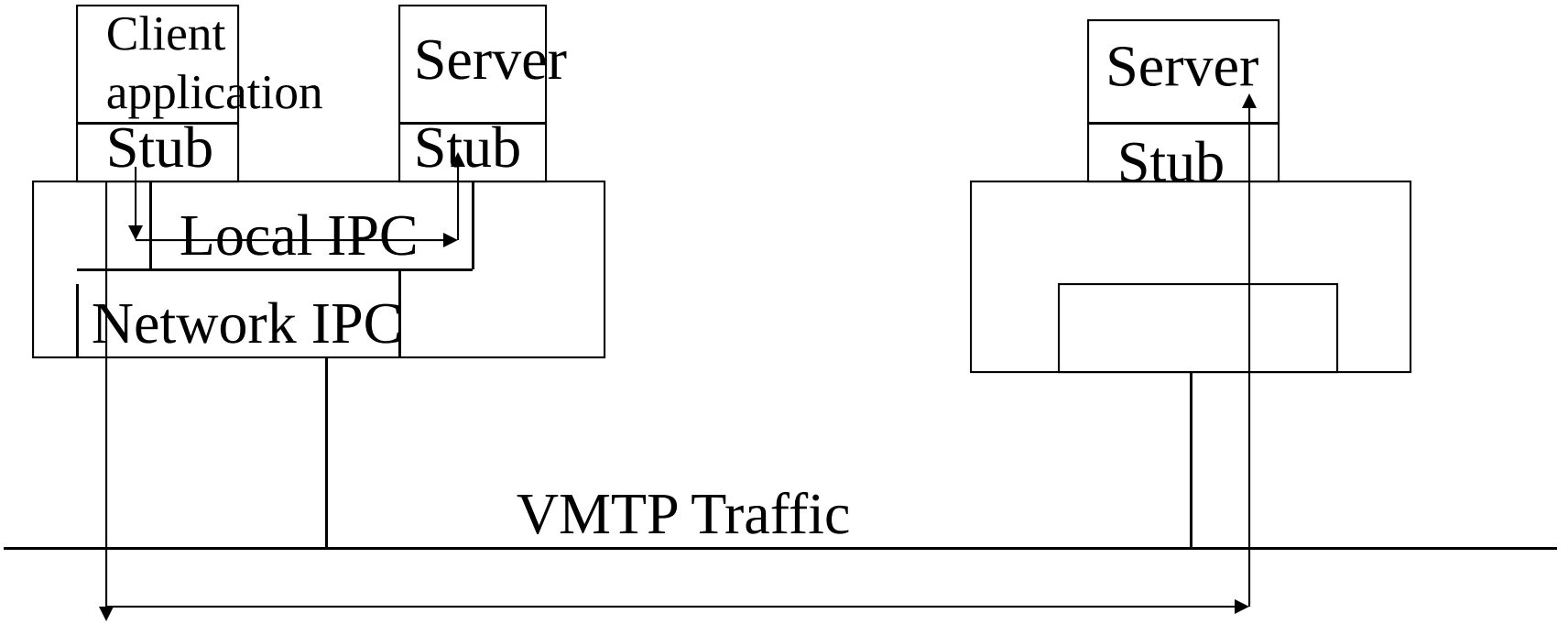
Distributed Kernel

- **Separate copies of kernel executes on each node.**
- **They cooperate to provide "single system" abstraction.**
- **Services: address spaces, LWP, and IPC.**

V's IPC Support

- **Fast and efficient transport-level service.**
  - **Support for RPC and file transfer.**
- **V's IPC is RPC-like.**
  - **Send primitive: send + receive.**
    - **Client sends request and blocks waiting for reply.**
    - **Server: processes request serially or concurrently.**
    - **Server response is both ACK and flow control.**
      - **It authorizes new request.**
      - **Simplifies transport protocol.**

V's IPC



Support for short, fixed size messages of 32 bytes with optional data segment of up to 16 Kbytes; simplifies buffering, transmission, and processing.

VMTP (1)

- **Transport protocol implemented in V.**
- **Optimized for request-response interactions.**
  - **No connection setup/teardown.**
  - **Response ACKs request.**
  - **Server maintains state about clients.**
    - **Duplicate suppression, caching of client information (e.g., authentication information).**

VMTP (2)

- **Support for group communication.**
  - **Multicast.**
  - **Process groups (e.g., group of file servers).**
    - **Identified by group id.**
    - **Operations: send to group, receive multiple responses to a request.**

VMTP Optimizations

- **Template of VMTP header + some fields initialized in process descriptor.**
  - **Less overhead when sending message.**
- **Short, fixed-size messages carried in the VMTP header: efficiency.**

V Kernel: Other Functions

- **Time, process, memory, and device management.**
- **Each implemented by separate kernel module (or server) replicated in each node.**
  - **Communicate via IPC.**
  - **Examples: kernel process server creates processes, kernel disk server reads disk blocks.**

Time

- **Kernel keeps current time of day (GMT).**
- **Processes can get(time), set(time), delay(time), wake up.**
- **Time synchronization among nodes: outside V kernel using IPC.**

Process Management

- **Create, destroy, schedule, migrate processes.**
- **Process management optimization.**
  - **Process initiation separated from address space allocation.**
    - **Process initiation = allocating/initializing new process descriptor.**
  - **Simplifies process termination (fewer kernel-level resources to reclaim).**
  - **Simplifies process scheduling: simple priority based scheduler; 2nd. level outside kernel.**

Memory Management 1

- **Protect kernel and other processes from corruption and unauthorized access.**
- **Address space: ranges of addresses (regions).**
  - **Bound to an open file (UIO like file descriptor).**
  - **Page fault references a portion of a region that is not in memory.**
  - **Kernel performs binding, caching, and consistency services.**

Memory Management 2

- **Virtual memory management: demand paging.**
  - **Pages are brought in from disk as needed.**
  - **Update kernel page tables.**
- **Consistency:**
  - **Same block may be stored in multiple caches simultaneously.**
  - **Make sure they are kept consistent.**

Device Management

- **Supports access to devices: disk, network interface, mouse, keyboard, serial line.**
- **Uniform I/O interface (UIO).**
  - **Devices are UIO objects (like file descriptors).**
  - **Example: mouse appears as an open file containing x & y coordinates & button positions.**
  - **Kernel mouse driver performs polling and interrupt handling.**
  - **But events associated with mouse changes (moving cursor) performed outside kernel.**

More on V...

- **Paper talks about other V functions implemented using kernel services.**
  - **File server.**
  - **Printer, window, pipe.**
- **Paper also talks about classes of applications that V targets with examples.**

The X-Kernel

- UofArizona, 1990.
- Like V, communication services are critical.
- Machines communicating through internet.
  - Heterogeneity!
  - The more protocols on user's machine, the more resources are accessible.
- The x-kernel philosophy: provide infrastructure to facilitate protocol implementation.

Virtual Protocols

- **The x-kernel provide library of protocols.**
  - **Combined differently to access different resources.**
  - **Example:**
    - **If communication between processes on the same machine, no need for any networking code.**
    - **If on the same LAN, IP layer skipped.**

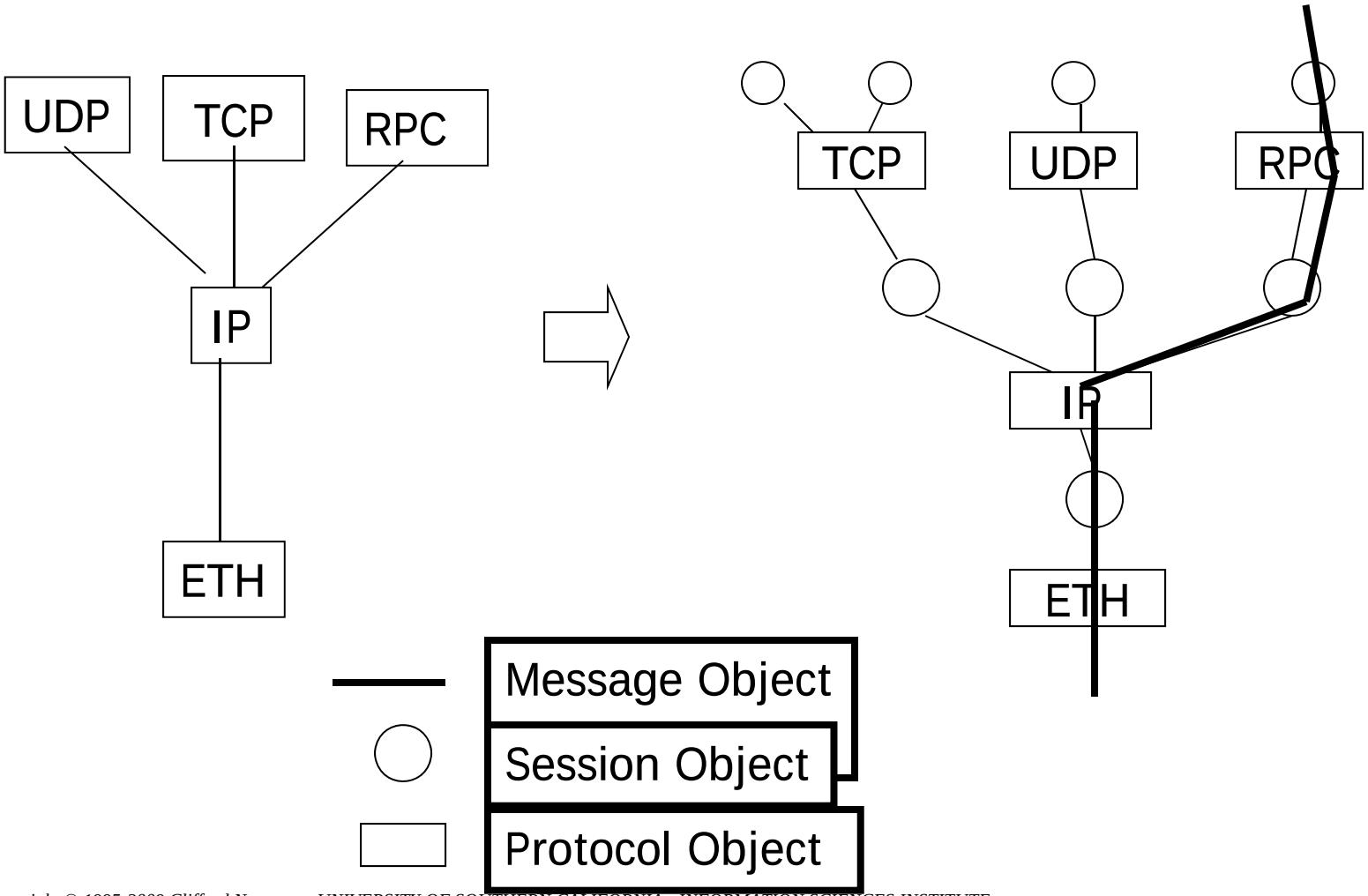
The X-Kernel : Process and Memory

- ability to pass control and data efficiently between the kernel and user programs
  - user data is accessible because kernel process executes in same address space
- kernel process -> user process
  - sets up user stack
  - pushes arguments
  - use user-stack
  - access only user data
- kernel -> user (245 usec), user -> kernel 20 usec on SUN 3/75

Communication Manager

- **Object-oriented infrastructure for implementing and composing protocols.**
- **Common protocol interface.**
- **2 abstract communication objects:**
  - **Protocols and sessions.**
  - **Example: TCP protocol object.**
    - **TCP open operation: creates a TCP session.**
    - **TCP protocol object: switches each incoming message to one of the TCP session objects.**
    - **Operations: demux, push, pop.**

X-kernel Configuration



Message Manager

- **Defines single abstract data type: message.**
  - **Manipulation of headers, data, and trailers that compose network transmission units.**
  - **Well-defined set of operations:**
    - **Add headers and trailers, strip headers and trailers, fragment/reassemble.**
  - **Efficient implementation using directed acyclic graphs of buffers to represent messages + stack data structure to avoid data copying.**

Mach

- **CMU (mid 80's).**
- **Mach is a microkernel, not a complete OS.**
- **Design goals:**
  - **As little as possible in the kernel.**
  - **Portability: most kernl code is machine independent.**
  - **Extensibility: new features can be implemented/tested alongside existing versions.**
  - **Security: minimal kernel specified and implemented in more secure way.**

Mach Features

- **OSs as Mach applications.**
- **Mach functionality:**
  - **Task and thread management.**
  - **IPC.**
  - **Memory management.**
  - **Device management.**

Mach IPC

- **Threads communicate using ports.**
- **Resources are identified with ports.**
- **To access resource, message is sent to corresponding port.**
  - **Ports not directly accessible to programmer.**
  - **Need handles to “port rights”, or capabilities (right to send/receive message to/from ports).**
- **Servers: manage several resources, or ports.**

Mach: ports

- *process port* is used to communicate with the kernel.
- *bootstrap port* is used for initialization when a process starts up.
- *exception port* is used to report exceptions caused by the process.
- *registered ports* used to provide a way for the process to communicate with standard system servers.

Protection

- **Protecting resources against illegal access:**
  - **Protecting port against illegal sends.**
- **Protection through *capabilities*.**
  - **Kernel controls port capability acquisition.**
  - **Different from Amoeba.**

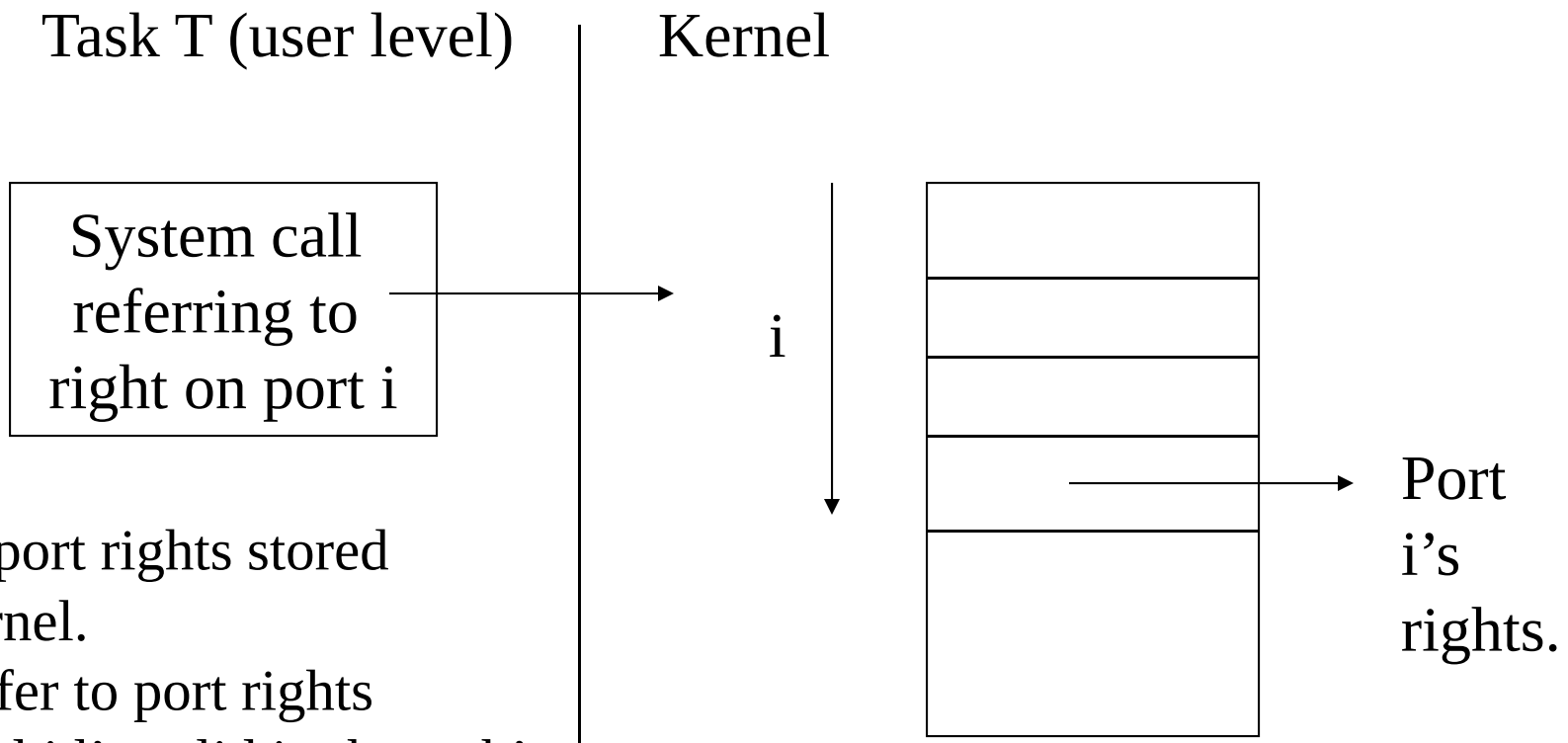
Capabilities 1

- **Capability to a port has field specifying port access rights for the task that holds the capability.**
  - **Send rights: threads belonging to task possessing capability can send message to port.**
  - **Send-once rights: allows at most 1 message to be sent; after that, right is revoked by kernel.**
  - **Receive rights: allows task to receive message from port's queue.**
    - **At most 1 task, may have receive rights at any time.**
    - **More than 1 task may have sned/send-once rights.**

Capabilities 2

- **At task creation:**
  - **Task given bootstrap port right: send right to obtain services of other tasks.**
  - **Task threads acquire further port rights either by creating ports or receiving port rights.**

Port Name Space



. Mach's port rights stored inside kernel.
. Tasks refer to port rights using local id's valid in the task's local port name space.

. Problem: kernel gets involved whenever ports are referenced.

Communication Model

- **Message passing.**
- **Messages: fixed-size headers + variable-length list of data items.**

Header	T	Port rights	T	In-line data	T	Pointer to out-of line data

Header: destination port, reply port, type of operation.
T: type of information.
Port rights: send rights: receiver acquires send rights to port.
Receive rights: automatically revoked in sending task.

Ports

- **Mach port has message queue.**
  - **Task with receive rights can set port's queue size dynamically: flow control.**
  - **If port's queue is full, sending thread is blocked; send-once sender never blocks.**
- **System calls:**
  - **Send message to kernel port.**
  - **Assigned at task creation time.**

Task and Thread Management

- **Task: execution environment (address space).**
- **Threads within task perform action.**
- **Task resources: address space, threads, port rights.**
- **PAPER:**
  - **How Mach microkernel can be used to implement other OSs.**
  - **Performace numbers comparing 4.3 BSD on top of Mach and Unix kernels.**

CSci555: Advanced Operating Systems

Lecture 12 – November 16 2012

Scheduling, Fault Tolerance
Real Time, Database Support

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

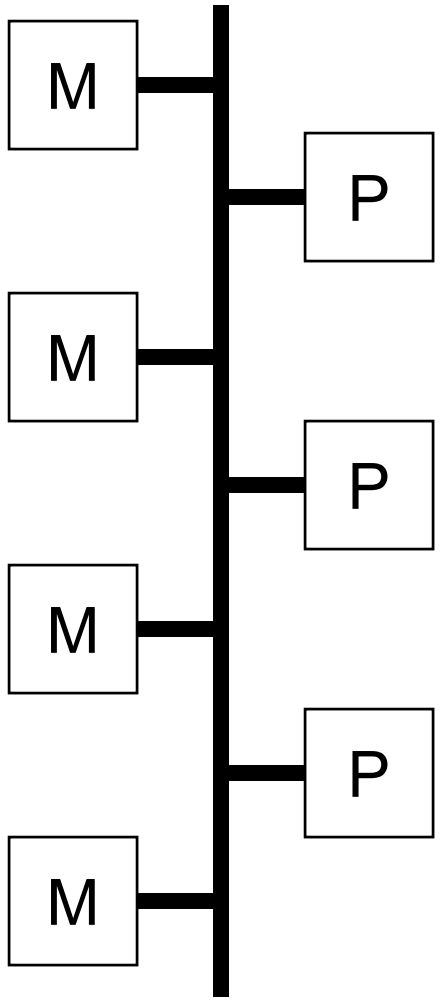
- **Scheduling**
  - **Allocation of resources at a particular point in time to jobs needing those resources, usually according to a defined policy.**
- **Focus**
  - **We will focus primarily on the scheduling of processing resources, though similar concepts apply the the scheduling of other resources including network bandwidth, memory, and special devices.**

Parallel Computing - General Issues

- **Speedup - the final measure of success**
  - **Parallelism vs Concurrency**
    - **Actual vs possible by application**
  - **Granularity**
    - **Size of the concurrent tasks**
    - **Reconfigurability**
  - **Number of processors**
  - **Communication cost**
  - **Preemption v. non-preemption**
  - **Co-scheduling**
    - **Some things better scheduled together**

Shared Memory Multi-Processing

- **Distributed shared memory, and shared memory multi-processors**
- **Processors usually tightly coupled to memory, often on a shared bus. Programs communicated through shared memory locations.**
- **For SMPs cache consistency is the important issue. In DSM it is memory coherence.**
  - **One level higher in the storage hierarchy**
- **Examples**
    - **Sequent, Encore Multimax, DEC Firefly, Stanford DASH**



Where is the best place for scheduling

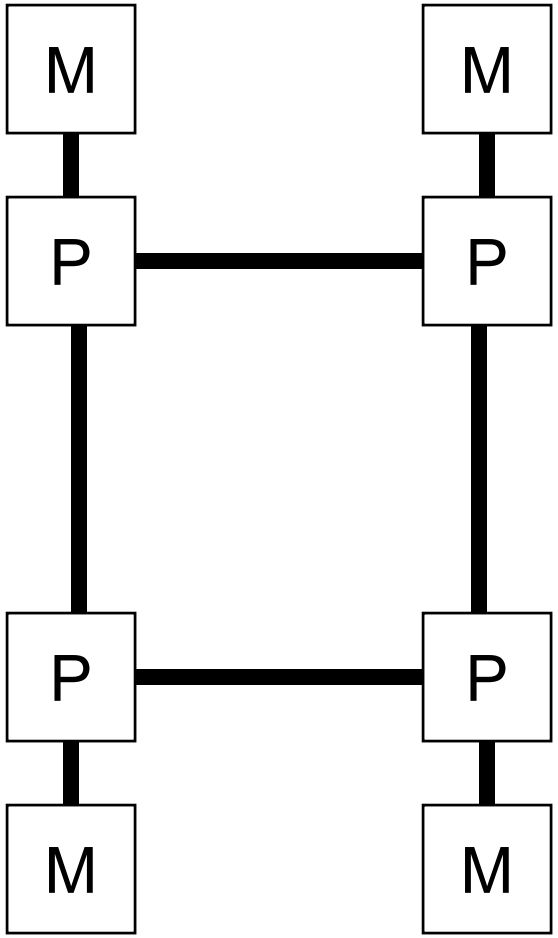
- **Application is in best position to know its own specific scheduling requirements**
  - **Which threads run best simultaneously**
  - **Which are on Critical path**
  - **But Kernel must make sure all play fairly**
- **MACH Scheduling**
  - **Lets process provide hints to discourage running**
  - **Possible to hand off processor to another thread**
    - **Makes easier for Kernel to select next thread**
    - **Allow interleaving of concurrent threads**
  - **Leaves low level scheduling in Kernel**
  - **Based on higher level info from application space**

Scheduler activations

- **User level scheduling of threads**
  - **Application maintains scheduling queue**
- **Kernel allocates threads to tasks**
  - **Makes upcall to scheduling code in application when thread is blocked for I/O or preempted**
  - **Only user level involved if blocked for critical section**
- **User level will block on kernel calls**
  - **Kernel returns control to application scheduler**

Distributed-Memory Multi-Processing

- **Processors coupled to only part of the memory**
  - **Direct access only to their own memory**
- **Processors interconnected in mesh or network**
  - **Multiple hops may be necessary**
- **May support multiple threads per task**
- **Typical characteristics**
  - **Higher communication costs**
  - **Large number of processors**
  - **Coarser granularity of tasks**
- **Message passing for communication**



Condor

- **Identifies idle workstations and schedules background jobs on them**
- **Guarantees job will eventually complete**

Condor

- **Analysis of workstation usage patterns**
  - **Only 30%**
- **Remote capacity allocation algorithms**
  - **Up-Down algorithm**
    - **Allow fair access to remote capacity**
- **Remote execution facilities**
  - **Remote Unix (RU)**

Condor

- **Leverage: performance measure**
  - **Ratio of the capacity consumed by a job remotely to the capacity consumed on the home station to support remote execution**
- **Checkpointing: save the state of a job so that its execution can be resumed**

Condor - Issues

- **Transparent placement of background jobs**
- **Automatically restart if a background job fails**
- **Users expect to receive fair access**
- **Small overhead**

Condor - scheduling

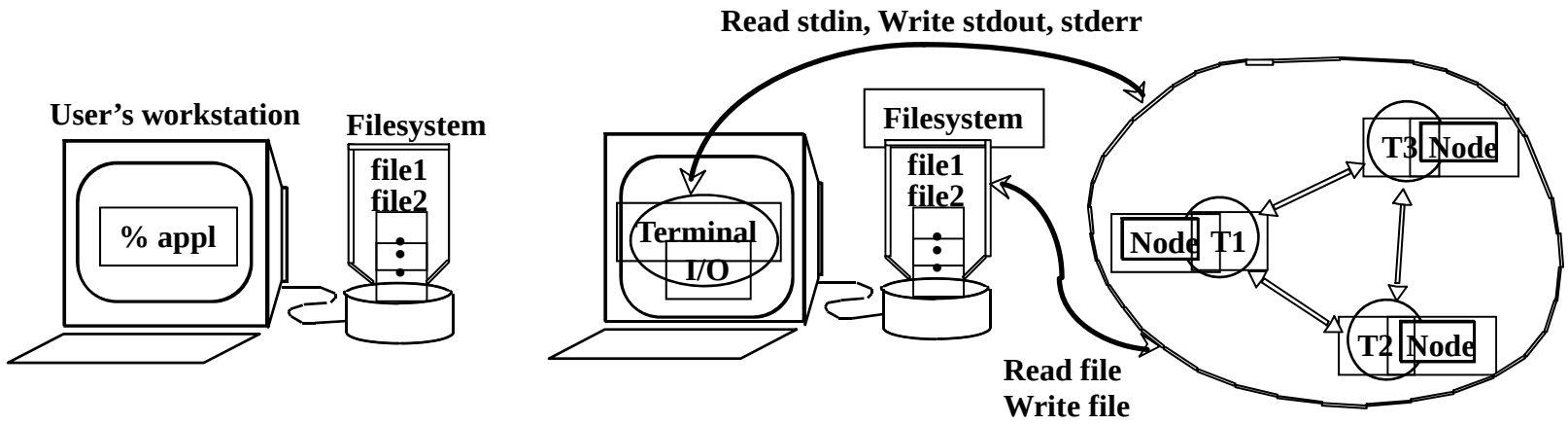
- **Hybrid of centralized static and distributed approach**
- **Each workstation keeps own state information and schedule**
- **Central coordinator assigns capacity to workstations**
  - **Workstations use capacity to schedule**

Prospero Resource Manager

Prospero Resource Manager - 3 entities

- **One or more system managers**
  - ***Each manages subset of resources***
  - ***Allocates resources to jobs as needed***
- **A job manager associated with each job**
  - ***Identifies resource requirements of the job***
  - ***Acquires resources from one or more system managers***
  - ***Allocates resources to the job's tasks***
- **A Node manager on each node**
  - ***Mediates access to the nodes resources***

The Prospero Resource Manager



A) User invokes an application program on his workstation.

b) The program begins executing on a set of nodes. Tasks perform terminal and file I/O on the user's workstation.

Advantages of the PRM

- **Scalability**
  - ***System manager does not require detailed job information***
  - ***Multiple system managers***
- **Job manager selected for application**
  - ***Knows more about job's needs than the system manager***
  - ***Alternate job managers useful for debugging, performance tuning***
- **Abstraction**
  - ***Job manager provides a single resource allocator for the job's tasks***
  - ***Single system model***

Real time Systems

- **Issues are scheduling and interrupts**
  - **Must complete task by a particular deadline**
  - **Examples:**
    - **Accepting input from real time sensors**
    - **Process control applications**
    - **Responding to environmental events**
- **How does one support real time systems**
  - **If short deadline, often use a dedicated system**
  - **Give real time tasks absolute priority**
  - **Do not support virtual memory**
    - **Use early binding**

Real time Scheduling

- **To initiate, must specify**
  - **Deadline**
  - **Estimate/upper-bound on resources**
- **System accepts or rejects**
  - **If accepted, agrees that it can meet the deadline**
  - **Places job in calendar, blocking out the resources it will need and planning when the resources will be allocated**
- **Some systems support priorities**
  - **But this can violate the RT assumption for already accepted jobs**

CSci555: Advanced Operating Systems

Lecture 12 – November 16, 2011

Fault Tolerant Computing

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

NOTE: This is a very short lecture, with much of the discussion integrated with the material on scheduling from the previous lecture.

- **Failure probabilities**
  - **Hierarchical, based on lower level probabilities**
  - **Failure Trees**
  - **Add probabilities where any failure affects you**
    - **Really (1 - ((1 - lambda)(1 -lambda) (1 - lambda)))**
  - **Multiply probabilities if all must break**
    - **Since numbers are small, this reduces failure rate**
  - **Both failure and repair rate are important**

Making systems fault tolerant

- **Involves masking failure at higher layers**
  - **Redundancy**
  - **Error correcting codes**
  - **Error detection**
- **Techniques**
  - **In hardware**
  - **Groups of servers or processors execute in parallel and provide hot backups**
- **Space Shuttle Computer Systems exampls**
- **RAID example**

Types of failures

- **Fail stop**
  - **Signals exception, or detectably does not work**
- **Returns wrong results**
  - **Must decide which component failed**
- **Byzantine**
  - **Reports difficult results to different participants**
  - **Intentional attacks may take this form**

Recovery

- **Repair of modules must be considered**
  - **Repair time estimates**
- **Reconfiguration**
  - **Allows one to run with diminished capacity**
  - **Improves fault tolerance (from catastrophic failure)**

OS Support for Databases

- **Example of OS used for particular applications**
- **End-to-end argument for applications**
  - **Much of the common services in OS's are optimized for general applications.**
  - **For DBMS applications, the DBMS might be in a better position to provide the services**
    - **Caching, Consistency, failure protection**

CSci555: Advanced Operating Systems

Lecture 13 – November 30, 2012

Grid and Cloud Computing

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

Announcements

- **No lecture November 25, Thanksgiving recess.**
- **Next lecture 2 December – final lecture**
  - **Need volunteer to do distribute class evaluations**
  - **Send email or suggest topics on discussion forum**
  - **Will include review for final exam**
  - **Research paper due 2 December**
- **Final exam on Friday December 9 – 2:00PM - 4:00PM**
  - **Location TBD**

Grids

- **Computational grids apply many distributed system techniques to meta computing (parallel applications running on large numbers of nodes across significant distances).**
  - **Libraries provide a common base for managing such systems.**
  - **Some consider grids different, but in my view the differences are not major, just the applications are.**
- **Data grids extend the grid "term" into other classes of computing.**
  - **Issues for data grids are massive storage, indexing, and retrieval.**
  - **It is a file system, indexing, and ontological problem.**

The Cloud

- **The cloud is many things to many people**
  - **Software as a service and hosted applications**
  - **Processing as a utility**
  - **Storage as a utility**
  - **Remotely hosted servers**
  - **Anything beyond the network card**

The Cloud

- **Clouds are hosted in different ways**
  - **Private Clouds**
  - **Public Clouds**
  - **Hosted Private Clouds**
  - **Hybrid Clouds**
  - **Clouds for federated enterprises**

The Cloud

- **Clouds are hosted in different ways**
  - **Private Clouds**
  - **Public Clouds**
  - **Hosted Private Clouds**
  - **Hybrid Clouds**
  - **Clouds for federated enterprises**

The Paper

- **Cloud Computing and Grid Computing 360 Degree compared.**
- **Written by one of the principal "architectures" of grid computing and provides one perspective.**
  - **Basically the paper is trying to frame cloud computing in terms of grid computing so that cloud computing does not steal the credit for many of the technological advances that was claimed by grid-computing.**
  - **In reality, many of the advances are from distributed systems research that predated the grid, and the grid did much of the same to distributes system research as cloud computing is doing to the grid.**
- **In both cases the innovation is/will be engineering and standardization in the context of particular classes of applications.**

Issues in the Grid and Cloud

- **Common interfaces and middleware**
  - **Directory services**
  - **Security services**
  - **File services**
  - **Scheduling services / allocation**
- **Support for federated environments**
  - **Security in such envrionements**

Directory Services

- **Need for a catalog of cloud or grid resources.**
- **Directory services also map locations for services once allocated to a computation.**

Security Services

- **Virtualization**
  - **Separation of “platform”**
- **VPN’s**
  - **Brings remote resources “inside”**
- **Federated Identity**
  - **Or separate identity for cloud**
- **Policy services**
  - **Much work is needed**

File Services

- **Performance often dictates storage near the computation.**
  - **But the data must be migrated**
  - **Alternatively, data accessed through callbacks to originating system.**
  - **Or in a separate storage cloud.**

Secheduling/Migration and Allocation

- **Characterize Node Capabilities in the Cloud**
  - **Security Characteristics**
    - **Accreditation of the software for managing nodes and data**
  - **Legal and Geographic Characteristics**
    - **Includes data on managing organizations and contractors**
  - **Need language to characterize**
  - **Need endorsers to certify**
- **Define Migration Policies**
  - **Who is authorized to handle data**
  - **Any geographic constraints**
  - **Necessary accreditation for servers and software**
    - **Each node that accepts data must be capable for enforcing policy before data can be redistributed.**
  - **Languages needed to describe**

Federation

- **Resources provided by parties with different interests.**
  - **No single chain of authority**
  - **Resources acquired from multiple parties and must be interconnected.**
- **Policy issues dominate**
  - **Who can use resources**
  - **Which resources is one willing to use.**
  - **Translating ID's and policies at boundaries**

CSci555: Advanced Operating Systems

Lecture 13 – November 30, 2012

Selected Topics and Scalable Systems

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

Hints for building scalable systems

- **From Lampson:**
  - **Keep it simple**
  - **Do one thing at a time**
  - **If in doubt, leave it out**
  - **But no simpler than possible**
  - **Generality can lead to poor performance**
  - **Make it fast and simple**
  - **Don’t hide power**
  - **Leave it to the client**
  - **Keep basic interfaces stable**

Hints for building scalable systems

- **From Lampson:**
  - **Plan to throw one away**
  - **Keep secrets**
  - **Divide and conquer**
  - **Use a good idea again**
  - **Handle normal and worst case separately**
  - **Optimize for the common case**
  - **Split resources in a fixed way**
  - **Cache results of expensive operations**
  - **Use hints**

Hints for building scalable systems

- **From Lampson:**
  - **When in doubt use brute force**
  - **Compute in the background**
  - **Use batch processing**
  - **Safety first**
  - **Shed load**
  - **End-to-end argument**
  - **Log updates**

CSci555: Advanced Operating Systems

Lecture 14 – December 7th, 2012

Scale in Distributed Systems

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

Announcements

- **Research paper due today**
  - **Late submissions with small penalty**
- **Class evaluations at Break**
  - **DEN students please return online**
- **Final Exam**
  - **Friday December 14, 2PM-4PM**
  - **Location to be determined**

- **A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.**

Three dimensions of scale

- **Numerical**
  - **Number of objects, users**
- **Geographic**
  - **Where the users and resources are**
- **Administrative**
  - **How many organizations own or use different parts of the system**

Effects of Scale

- **Reliability**
  - **Autonomy, Redundancy**
- **System Load**
  - **Order of growth**
- **Administration**
  - **Rate of change**
  - **Heterogeneity**

Techniques - Replication

- **Placement of replicas**
  - **Reliability**
  - **Performance**
  - **Partition**
  - **What if all in one place**
- **Consistency**
  - **Read-only**
  - **Update to all**
  - **Primary Site**
  - **Loose Consistency**

- **Placement of servers**
  - **Reliability**
  - **Performance**
  - **Partition**
- **Finding the right server**
  - **Hierarchy/iteration**
  - **Broadcast**

- **Placement of Caches**
  - **Multiple places**
- **Cache consistency**
  - **Timeouts**
  - **Hints**
  - **Callback**
  - **Snooping**
  - **Leases**

CSci555: Advanced Operating Systems

Lecture 14 – December 7th, 2012
Selected Topics and Discussions

Dr. Clifford Neuman
University of Southern California
Information Sciences Institute

Is the OS still relevant

- **What is the role of an OS in the internet**
  - **Are today's computers appliances for accessing the web?**

Is the OS still relevant

- **OS Manages local resources**
  - **Provides protection between applications**
  - **Though the role seems diminished, it is actually increasing in importance**

Today's File Systems

- **Network Attached Storage**
- **Cloud Storage**
- **Content Distribution Systems**
- **Peer to Peer File Systems**

Content Delivery

- **Pre-staging of content**
- **Techniques needed to redirect to local copy.**
- **Ideally need ways to avoid central bottleneck.**
- **Use of URN's can help, but needs underlying changes to browsers.**
  - **For dedicated apps, easier to deploy**

Naming Today

- **URL's vs URN's**
- **System based identifiers**
  - **Facebook**
  - **Twitter**
  - **Tiny URL's**
  - **These make the problem worse in the interest of locking users into their system.**
- **Internationalized Domain Names**

Multi-Core Systems

- **Shared Memory Multiprocessor**
  - **But few apps know how to take advantage of it**
  - **But modern OS – many processes**
- **Still leaves contention for other resources**

Internet Search Techniques

- **Issues**
  - **How much of the net to index**
    - **How much detail**
    - **How to select**
  - **Relevance of results**
    - **Ranking results – avoiding spam**
    - **Context for searching**
      - **Transitive indexing**
- **Scaling the search engines**

Internet Search Techniques - Google

- **Data Distribution**
  - **Racks and racks of servers running Linux – key data is replicated**
    - **Some for indices**
    - **Some for storing cached data**
  - **Query distributed based on load**
  - **Many machines used to for single query**
- **Page rank**
  - **When match found, ranking by number and quality of links to the page.**

The Structure of Distributed Systems

- **Client server**
- **Object Oriented**
- **Peer to Peer (additional discussion)**
- **Cloud Based**
- **Federated**
- **Agent Based**
- **Virtualized**
- **Embedded**

Peer to Peer

- **Peer to peer systems are client server systems where the client is also a server.**
- **The important issues in peer to peer systems are really:**
  - **Trust – one has less trust in servers**
  - **Unreliability – Nodes can drop out at will.**
  - **Management – need to avoid central control (a factor caused by unreliability)**
- **Ad hoc network related to peer to peer**

Future of Distributed Systems

- **More embedded systems (becoming less "embedded").**
  - **Process control / SCADA**
  - **Real time requirements**
  - **Protection from the outside**
  - **Ae they really embedded?**
- **Stronger management of data flows across applications.**
- **Better resource management across organizational domains.**
- **Multiple views of available resources.**
- **Hardware abstraction**

Hardware Abstraction

- **Many operating systems are designed today to run on heterogeneous hardware**
- **Hardware abstraction layer often part of the internal design of the OS.**
  - **Small set of functions**
  - **Called by main OS code**
- **Usually limited to some similarity in hardware, or the abstraction code becomes more complex and affects performance.**

Emulation and Simulation

- **Need techniques to test approaches before system is built.**
  - **Simulations**
  - **Need real data sets to model assumptions.**
- **Need techniques to test scalability before system is deployed.**
  - **Deployment harder than implementation**
  - **Emulations and simulations beneficial**
- **Issues in emulation and simulation**

Windows

- **XP, Win2K and successors based loosely on Mach Kernel.**
- **Techniques drawn from many other research systems.**
- **Backwards compatibility has been an issue affecting some aspects of it architecture.**
- **Despite common criticism, the current versions make a pretty good system for general computing needs.**

Miscellaneous

- **Security issues with the Domain Name System**
  - **A result of multi-level caching**
  - **And security not considered up front**
- **Neutrality in Distributed Systems**
  - **Protocols**
  - **Net Neutrality**
  - **Application frameworks / middleware**
- **Unix and Linux**
  - **Kernel Structure**
  - **Filesystems**

CSci555: Advanced Operating Systems

Lecture 14 – December 2nd, 2011

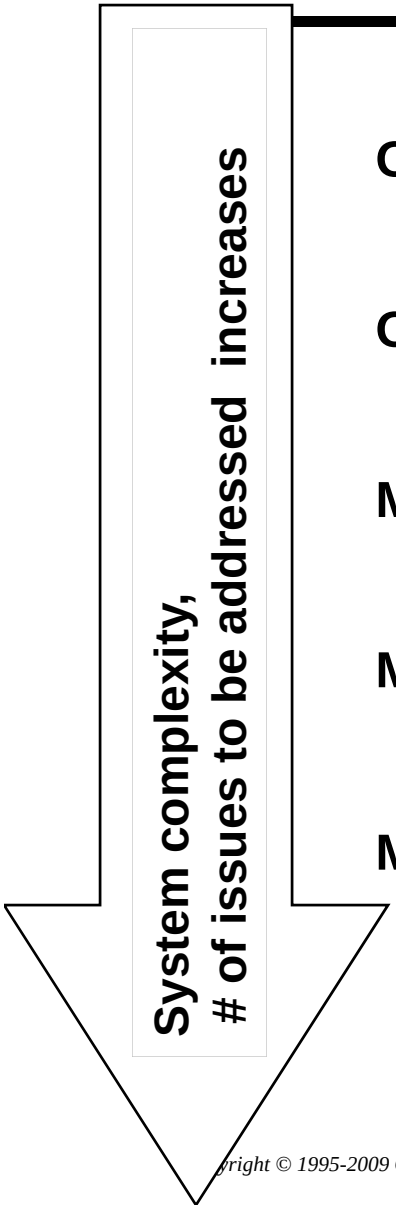
REVIEW

Dr. Clifford Neuman

University of Southern California

Information Sciences Institute

Review for final



One user, one site, one process

One user, one site, multiple processes

Multiple users, one site, multiple processes

Multiple (users, sites and processes)

Multiple (users, sites, organizations and processes)

Review for Final

- **General**
  - **Operating Systems Functions**
  - **Kernel structure - microkernels**
  - **What belongs where**
- **Communication models**
  - **Message Passing**
  - **RPC**
  - **Distributed Shared Memory**
  - **Other Models**

Review for Final

- **Synchronization - Transactions**
  - **Time Warp**
  - **Reliable multicast/broadcast**
- **Naming**
  - **Purpose of naming mechanisms**
  - **Approaches to naming**
  - **Resource Discovery**
  - **Scale**

Review for Final

- **Security – Requirements**
  - **Confidentiality**
  - **Integrity**
  - **Availability**
- **Security mechanisms (prevention/detection)**
  - **Protection**
  - **Authentication**
  - **Authorization (ACL, Capabilities)**
  - **Intrusion detection**
  - **Audit**
- **Cooperation among the security mechanisms**
- **Scale**

Review for Final

- **Distributed File Systems - Caching**
  - **Replication**
  - **Synchronization**
    - **voting,master/slave**
  - **Distribution**
  - **Access Mechanism**
  - **Access Patterns**
  - **Availability**
- **Other file systems**
  - **Log Structured**
  - **RAID**

Review for Final

- **Case Studies**
  - **Locus**
  - **Athena**
  - **Andrew**
  - **V**
  - **HCS**
  - **Amoeba**
  - **Mach**
  - **CORBA**
- **Resource Allocation**
- **Real time computing**
- **Fault tolerant computing**

SCALE

1a) System load (10 points) – Suggest some techniques that can be used to reduce the load on individual servers within a distributed system? Provide examples of how these techniques are used from each of the following systems: The Domain Name System, content delivery throughthe world wide web, remote authentication in the Kerberos system. Note that some of the systems use more than one technique.

1b) Identifying issues (20 points) for each of the techniques described in part (a) there are issues that must be addressed to make sure that the system functions properly (I am interested in the properly aspect here, not the most efficiently aspect). For each technique identify the primary issues that needs to be addressed and explain how it is addressed in each of the listed systems that uses the technique.

2006 Exam – 2 Kernel

2) For each of the operating system functions listed below list the benefits and drawbacks to placing the function in the Kernel, leaving the function to be implemented by the application, or providing the function in users space through a server (the server case includes cases where the application selects and communicates with a server, and also the case where the application calls the kernel, but the processing is redirected by the kernel to a server). For each function, suggest the best location(s) to provide this function. If needed you can make an assumption about the scenario for which the system will be used. Justify your choice for placement of this function. There may be multiple correct answers for this last part – so long as your justification is correct.

File System

Virtual Memory

Communications

Scheduling

Security

2006 Exam – 3 Design Problem – Fault Toleance

3) You are designing a database system that requires significant storage and processing power. Unfortunately, you are stuck using the hardware that was ordered by the person whose job you just filled. This morning, the day after you first arrived at work, a truck arrived with 10 processors (including memory, network cards, etc), 50 disk drives, and two uninterruptible power supplies. The failure rates of the processors (including all except the disk drives and power supplies) is λp. The failure rates on the disk drives is λd, and the failure rate for the power supplies is λe.

a) You learned from your supervisor that the reason they let the last person go is that he designed the system so that the failure of any of the components would cause the system to stop functioning. In terms of λp,d,ande, what is the failure probability for the system as a whole. (5 points)

b) The highest expected load on your system could be handled by about half the processors. The largest expected dataset size that is expected is about 1/3 the capacity of the disks that arrived. Suggest a change to the structure of the system, using the components that have already arrived, that will yield better fault tolerance. In terms of λp,d,and e, what is the failure probability for the new system? (note, there are easy things and harder things you can do here, I suggest describing the easing things, generating the probability based on that approach, and then just mentioning some of the additional steps that could be taken to further improve the fault tolerance (15 points)

c) List some of the problems that you would need to solve or some of the assumptions you would need to make, in order to construct the system described in part b from the components that arrived this morning (things like number of network interfaces per processor, how the disks are connected to processors or the network). Discuss also any assumptions you need to make regarding detect ability of failures, and describe your approach to failover (how will the failures be masked, what steps are taken when a failure occurs). (15 points)

2007 Exam – 1a Leases

For each of the following approaches to consistency, if they were to be implemented as a lease, list the corresponding lease term, and the rules for breaking the lease (i.e. if the normal rules for breaking a lease are not provided by the system, what are the effective rules of the mechanism. (16 points)

a. AFS-2/3 Callback

b. AFS-1 Check-on-use

c. Time to live in the domain name system

d. Locks in a transaction system

A. Discuss the similarity between a transaction system and the log structure file system.

B. How does the log structure file system improve the performance of writes to the file system?

C. Why does it take so much less time to recover from a system crash in a log structured file system than it does in the traditional Unix file system? How is recovery accomplished in the log structure approach?

2007 Exam – 2 Kernels

For a general purpose operating system such Linux, discuss the placement of services, listing those functions that should be provided by the kernel, by the end application itself, and by application level servers. Specifically, what OS functions should be provided in each location? Justify your answer and state your assumptions.

a) In the Kernel itself

b) In the application itself

c) In servers outside the kernel

For a system supporting embedded applications, such as process control, what changes would you make in the placement of OS functions (i.e. what would be different than what you described in a-c). Justify your answer.

2007 Exam – 3 Design Problem

You have been hired to build a system to manage ticket sales for large concerts. This system must be highly scalable supporting near simultaneous request from the “flash crowds” accessing the system the instant a new concert goes on sale. The system must accept requests fairly, so that ticket consolidators are unable to “game the system” to their advantage through automated programs on well placed client machines located close to the servers in terms of network topology. To handle the load will require multiple servers all with access to the ticketing database, yet synchronization is a must as we can’t sell the same seat to more than one person. The system must support several functions, among which are providing venue and concert information to potential attendees, displaying available seats, reserving seats, and completing the sale (collecting payment, recording the sale, and enabling the printing of a barcode ticket).

a) Describe the architecture of your system in terms of the allocation of functions across processors. Will all processors be identical in terms of their functionality, or different servers provide different functions, and if so which ones and why?

b) Explain the transactional characteristics of your system. In particular, when does a transaction begin, and when does it commit or abort, and which processors (according to the functions described by you in part a) will be participants in the transaction.

c) What objects will have associated locks and when will these object be locked and unlocked.

d) How will you use replication in your system and how will you manage consistency of such replicated data

e) How will you use distribution in your system