CSci555: Advanced Operating Systems
Lecture 1 – August 31, 2012

Dr. Clifford Neuman
University of Southern California
Information Sciences Institute
Administration

- **Instructors**
  - Dr. Clifford Neuman
    - csci555f12@clifford.neuman.name
    - Office hours – SAL 212
      - Friday 12:55 PM – 1:55 PM

- **TA**
  - T.B.D
    - T.B.D. at usc dot edu
    - Office Hours – TBD

- **Class details**
  - http://gost.isi.edu/555
Administration

- Class Home Page
  http://gost.isi.edu/555/
  - Announcements
  - Syllabus
  - Lecture Slides
  - Reading list

- Class e-mail: csci555@usc.edu
Administrative Information

- Reading list
  - ~65 papers and
  - ~20 book chapters
  - Concentrated toward the first half

- Text
  - Principles of Computer System Design
    - By Saltzer and Kaashoek
  - Also using second volume online
Administrative Information

- **Assignments**
  - 4 Reports,
    - Due 11 p.m. Wednesday nights
  - Research Paper
    - Due: last class
  - Exams
    - Mid-Term: Friday, October 19
    - Final Exam: Friday, December 14
Administrative Information

- DEN site - Blackboard
  - Lecture webcast
  - Class forum on DEN
  - Grades

- Lecture notes to be posted before lecture
- Academic Integrity
  - READ IT – It applies to you
Academic Integrity

I take Academic Integrity Seriously
- Every year I have too many cases of cheating
- Last year I assigned multiple F’s for the class
- Occasionally students leave USC

What is and is not OK
- I encourage you to work with others to learn the material
- Do not to turn in the work of others
- Do not give others your work to use as their own
- Do not plagiarize from others (published or not)
- Do not try to deceive the instructors

See section on web site and assignments
- More guidelines on academic integrity
- Links to university resources
- Don’t just assume you know what is acceptable.
Grading

- 20%: Reading Reports
- 20%: Midterm
- 20%: Final
- 30%: Research Paper
- 10%: Class Participation & Quizes
  - Class forum participation
  - In class participation
How to survive?

❖ Read the survival guide
❖ How to read papers
  ❖ Read the papers in advance
    ▪ Be critical
  ❖ At least skim through
❖ Build your own notes
❖ Study group
What you should learn in this course

- You will gain a basic understanding of distributed system concepts.
- You will develop intuition for which approaches work, and which don’t.
- You will develop the ability to sense where bottlenecks lie in system design.
- You will remember where to look for more information when you are faced with a distributed system problem.
- Above all, you will learn how to be critical of what you are told by system designers.
Some things an operating system does (review)

- Memory Management
- Scheduling / Resource management
- Communication
- Protection and Security
- File Management - I/O
- Naming
- Synchronization
- User Interface
Progression of Operating Systems

Primary goal of a distributed system:
- Sharing

Progression over past years
- Dedicated machines
- Batch Processing
- Time Sharing
- Workstations and PC’s
- Distributed Systems
- Devices
Structure of Distributed Systems

- Kernel
  - Basic functionality and protection
- Application Level
  - Does the real work
- Servers
  - Service and support functions needed by applications
  - Many functions that used to be in Kernel are now in servers.
Structure of Distributed Systems

User Space

Kernel

UP

SVR

User Space

Kernel

SVR
Network vs. OS Layering
(No direct mapping, colors to stimulate discussion)
Characteristics of a Distributed System

- Basic characteristics:
  - Multiple Computers
  - Interconnections
  - Shared State
Why Distributed Systems are Hard

- Scale:
  - Numeric
  - Geographic
  - Administrative

- Loss of control over parts of the system

- Unreliability of Messages

- Parts of the system down or inaccessible
  - Lamport: You know you have a distributed system when the crash of a computer you have never heard of stops you from getting any work done.
End-to-End Argument

- QUESTION: Where to place distributed systems functions?
- Layered system design:
  - Different levels of abstraction for simplicity.
  - Lower layer provides service to upper layer.
  - Very well defined interfaces.
E2E Argument (continued)

- E2E paper argues that functions should be moved closer to the application that uses them.
E2E Argument (continued)

❖ Rationale:
  ❑ Some functions can only be completely and correctly implemented with application’s knowledge.
    ▪ Example:
      – Reliable message delivery, security
      – Encrypted e-mail
      – Streaming media vs. Banking
  ❑ Applications that do not need certain functions should not have to pay for them.
E2E Counter-Argument

- Performance
  - Example: File transfer
    - Reliability checks at lower layers detect problems earlier.
    - Abort transfer and re-try without having to wait till whole file is transmitted.

- Abstraction
  - Less repetition across apps

Bottom line: “spread” functionality across layers.
Lecture Transition

- This concludes the slides for lecture 1.
- The following slides are for lecture 2.
CSci555: Advanced Operating Systems
Lecture 2 – September 7, 2012
Communication Models

Dr. Clifford Neuman

University of Southern California
Information Sciences Institute
Outline: Communications Models

- Communication Models:
  - General concepts.
  - Message passing.
  - Distributed shared memory (DSM).
  - Remote procedure call (RPC) [Birrel et al.]
    - Light-weight RPC [Bershad et al.]
  - DSM case studies
    - IVY [Li et al.]
    - Linda [Carriero et al.]
Communication Models

- Support for processes to communicate among themselves.
- Traditional (centralized) OS’s:
  - Provide local (within single machine) communication support.
- Distributed OS’s: must provide support for communication across machine boundaries.
  - Over LAN or WAN.
Communication Paradigms

- 2 paradigms
  - Message Passing (MP)
  - Distributed Shared Memory (DSM)

- Message Passing
  - Processes communicate by sending messages.

- Distributed Shared Memory
  - Communication through a “virtual shared memory”.
Message Passing

- Basic communication primitives:
  - Send message.
  - Receive message.

- Modes of communication:
  - Synchronous versus asynchronous.

- Semantics:
  - Reliable versus unreliable.
Synchronous Communication

- **Blocking send**
  - Blocks until message is transmitted
  - Blocks until message acknowledged

- **Blocking receive**
  - Waits for message to be received

- **Process synchronization.**
Asynchronous Communication

- Non-blocking send: sending process continues as soon message is queued.
- Blocking or non-blocking receive:
  - Blocking:
    - Timeout.
    - Threads.
  - Non-blocking: proceeds while waiting for message.
    - Message is queued upon arrival.
    - Process needs to poll or be interrupted.
Reliability of Communication

- Unreliable communication:
  - “best effort” - send and hope for the best
  - No ACKs or retransmissions.
  - Application must provide its own reliability.
  - Example: User Datagram Protocol (UDP)
    - Applications using UDP either don’t need reliability or build their own (e.g., UNIX NFS and DNS (both UDP and TCP), some audio or video applications)
Reliability of Communication

- Reliable communication:
  - Different degrees of reliability.
  - Processes have some guarantee that messages will be delivered.
  - Example: Transmission Control Protocol (TCP)
  - Reliability mechanisms:
    - Positive acknowledgments (ACKs).
    - Negative Acknowledgments (NACKs).
  - Possible to build reliability atop unreliable service (E2E argument).
Distributed Shared Memory

- Motivated by development of shared-memory multiprocessors which do share memory.
- Abstraction used for sharing data among processes running on machines that do not share memory.
- Processes think they read from and write to a “virtual shared memory”.

Copyright © 1995-2012 Clifford Neuman - UNIVERSITY OF SOUTHERN CALIFORNIA - INFORMATION SCIENCES INSTITUTE
DSM 2

- Primitives: read and write.
- OS ensures that all processes see all updates.
  - Happens transparently to processes.
DSM and MP

- DSM is an abstraction!
  - Gives programmers the flavor of a centralized memory system, which is a well-known programming environment.
  - No need to worry about communication and synchronization.
- But, it is implemented atop MP.
  - No physically shared memory.
  - OS takes care of required communication.
Caching in DSM

- For performance, DSM caches data locally.
  - More efficient access (locality).
  - But, must keep caches consistent.
  - Caching of pages for of page-based DSM.

- Issues:
  - Page size.
  - Consistency mechanism.
Approaches to DSM

- **Hardware-based:**
  - Multi-processor architectures with processor-memory modules connected by high-speed LAN (E.g., Stanford’s DASH).
  - Specialized hardware to handle reads and writes and perform required consistency mechanisms.
Approaches to DSM

- Page-based:
  - Example: IVY.
  - DSM implemented as region of processor’s virtual memory; occupies same address space range for every participating process.
  - OS keeps DSM data consistency as part of page fault handling.
Library-based:
- Or language-based.
- Example: Linda.
- Language or language extensions.
- Compiler inserts appropriate library calls whenever processes access DSM items.
- Library calls access local data and communicate when necessary.
DSM Case Studies: IVY

- Environment: "loosely coupled" multiprocessor.
  - Memory is physically distributed.
  - Memory mapping managers (OS kernel):
    - Map local memories to shared virtual space.
    - Local memory as cache of shared virtual space.
    - Memory reference may cause page fault; page retrieved and consistency handled.
Issues:

- Read-only versus writable data.
- Locality of reference.
- Granularity (1 Kbyte page size).
  - Bigger pages versus smaller pages.
Memory coherence strategies:

- Page synchronization
  - Invalidation
  - Write broadcast
- Page ownership
  - Fixed: page always owned by same processor
  - Dynamic
IVY Page Synchronization

- **Invalidation:**
  - On write fault, invalidate all copies; give faulting process write access; gets copy of page if not already there.
  - Problem: must update page on reads.

- **Write broadcast:**
  - On write fault, fault handler writes to all copies.
  - Expensive!
IVY Memory Coherence

- Paper discusses approaches to memory coherence in page-based DSM.
  - Centralized: single manager residing on a single processor managing all pages.
  - Distributed: multiple managers on multiple processors managing subset of pages.
Linda

- **DSM**: tuple space.

- **Basic operations**:
  - out (data): data added to tuple space.
  - in (data): removes matching data from TS; destructive.
  - read (data): same as “in”, but tuple remains in TS (non-destructive).
Linda Primitives: Examples

- `out ("P", 5, false) : tuple ("P", 5, false)` added to TS.
  - "P" : name
  - Other components are data values.
  - Implementation reported on the paper: every node stores complete copy of TS.
  - `out (data)` causes data to be broadcast to every node.
Linda Primitives: Examples

- in ("P", int I, bool b): tuple ("P", 5, false) removed from TS.
  - If matching tuple found locally, local kernel tries to delete tuple on all nodes.
Remote Procedure Call

- Builds on MP.
- Main idea: extend traditional (local) procedure call to perform transfer of control and data across network.
- Easy to use: analogous to local calls.
- But, procedure is executed by a different process, probably on a different machine.
- Fits very well with client-server model.
RPC Mechanism

1. Invoke RPC.
2. Calling process suspends.
3. Parameters passed across network to target machine.
4. Procedure executed remotely.
5. When done, results passed back to caller.
6. Caller resumes execution.

Is this synchronous or asynchronous?
RPC Advantages

- Easy to use.
- Well-known mechanism.
- Abstract data type
  - Client-server model.
  - Server as collection of exported procedures on some shared resource.
  - Example: file server.
- Reliable.
MP Reliability of Communication

- Unreliable communication:
  - “best effort” - send and hope for the best
  - No ACKs or retransmissions.
  - Application must provide its own reliability.
  - Example: User Datagram Protocol (UDP)
    - Applications using UDP either don’t need reliability or build their own (e.g., UNIX NFS and DNS (both UDP and TCP), some audio or video applications)
RPC Semantics 1

- Delivery guarantees.

- “Maybe call”:
  - Clients cannot tell for sure whether remote procedure was executed or not due to message loss, server crash, etc.
  - Usually not acceptable.
RPC Semantics 2

“At-least-once” call:

- Remote procedure executed at least once, but maybe more than once.
- Retransmissions but no duplicate filtering.
- Idempotent operations OK; e.g., reading data that is read-only.
RPC Semantics 3

❖ “At-most-once” call
  ❑ Most appropriate for non-idempotent operations.
  ❑ Remote procedure executed 0 or 1 time, ie, exactly once or not at all.
  ❑ Use of retransmissions and duplicate filtering.
  ❑ Example: Birrel et al. implementation.
    ▪ Use of probes to check if server crashed.
RPC Implementation (Birrel et al.)

**Call**

**User** stub

**RPC runtime**

**Call packet**

**Result**

**Server**

**Callee**

**RPC runtime**

**Server stub**

**Result**

**Server**

**call**

**work**

**return**

Copyright © 1995-2012 Clifford Neuman - UNIVERSITY OF SOUTHERN CALIFORNIA - INFORMATION SCIENCES INSTITUTE
RPC Implementation 2

- RPC runtime mechanism responsible for retransmissions, acknowledgments.
- Stubs responsible for data packaging and un-packaging;
  - AKA marshalling and un-marshalling: putting data in form suitable for transmission. Example: Sun’s XDR.
How to determine where server is? Which procedure to call?
  - "Resource discovery" problem
    - Name service: advertises servers and services.
    - Example: Birrel et al. uses Grapevine.

Early versus late binding.
  - Early: server address and procedure name hard-coded in client.
  - Late: go to name service.
Synchronous & Asynchronous RPC

Synchronous

Asynchronous

Copyright © 1995-2012 Clifford Neuman - UNIVERSITY OF SOUTHERN CALIFORNIA - INFORMATION SCIENCES INSTITUTE
RPC Performance

- Sources of overhead
  - data copying
  - scheduling and context switch.

- Light-Weight RPC
  - Shows that most invocations took place on a single machine.
  - Optimizes data copying and thread scheduling for local case.
LW-RPC 1

- Argument copying
  - RPC: 4 times
  - copying between kernel and user space.

- LW-RPC: common data area (A-stack) shared by client and server and used to pass parameters and results; access by client or server, one at a time.
**LW-RPC 2**

- A-stack avoids copying between kernel and user spaces.
- Client and server share the same thread: less context switch (like regular calls).

![Diagram showing the process of client and server interaction with kernel. The process includes: 1. copy args, 2. traps, 3. upcall, 4. executes & returns.]

*Copyright © 1995-2012 Clifford Neuman - UNIVERSITY OF SOUTHERN CALIFORNIA - INFORMATION SCIENCES INSTITUTE*
Lecture Transition

- This concludes the slides for lecture 2.
- The following slides are for lecture 3.
Concurrency Control and Synchronization

- How to control and synchronize possibly conflicting operations on shared data by concurrent processes?
- First, some terminology.
  - Processes.
  - Light-weight processes.
  - Threads.
  - Tasks.
Processes

Text book:

- Processing activity associated with an execution environment, ie, address space and resources (such as communication and synchronization resources).
Threads

- OS abstraction of an activity/task.
  - Execution environment expensive to create and manage.
  - Multiple threads share single execution environment.
  - Single process may spawn multiple threads.
  - Maximize degree of concurrency among related activities.
  - Example: multi-threaded servers allow concurrent processing of client requests.
Other Terminology

- Process versus task/thread.
  - Process: heavy-weight unit of execution.
  - Task/thread: light-weight unit of execution.
Threads Case Study 1

- Hauser et al.
- Examine use of user-level threads in 2 OS’s:
  - Xerox Parc’s Cedar (research).
  - GVX (commercial version of Cedar).
- Study dynamic thread behavior.
  - Classes of threads (eternal, worker, transient)
  - Number of threads.
  - Thread lifetime.
Thread Paradigms

- Different categories of usage:
  - Defer work: thread does work not vital to the main activity.
    - Examples: printing a document, sending mail.
  - Pumps: used in pipelining; use output of a thread as input and produce output to be consumed by another task.
  - Sleepers: tasks that repeatedly wait for an event to execute; e.g., check for network connectivity every x seconds.
Synchronization

- So far, how one defines/activates concurrent activities.
- But how to control access to shared data and still get work done?
- Synchronization via:
  - Shared data [DSM model].
  - Communication [MP model].
Synchronization by Shared Data

- **Primitives**
  - Semaphores.
  - Conditional critical regions.
  - Monitors.
Synchronization by MP

- Explicit communication.
- Primitives send and receive
  - *Blocking send, blocking receive*: sender and receiver are blocked until message is delivered (rendezvous)
  - *Nonblocking send, blocking receive*: sender continues processing receiver is blocked until the requested message arrives
  - *Nonblocking send, nonblocking receive*: messages are sent to a shared data structure consisting of queues (mailboxes)

Deadlocks?

- **Mailboxes** one process sends a message to the mailbox and the other process picks up the message from the mailbox
  
  Example:
  Send (mailbox, msg)
  Receive (mailbox, msg)
Transactions

- Database term.
  - Execution of program that accesses a database.

- In distributed systems,
  - Concurrency control in the client/server model.
  - From client’s point of view, sequence of operations executed by server in servicing client’s request in a single step.
Transaction Properties

- **ACID:**
  - **Atomicity:** a transaction is an atomic unit of processing and it is either performed entirely or not at all
  - **Consistency:** a transaction's correct execution must take the database from one correct state to another
  - **Isolation:** the updates of a transaction must not be made visible to other transactions until it is committed
  - **Durability:** if transaction commits, the results must never be lost because of subsequent failure
Transaction Atomicity

- “All or nothing”.
- Sequence of operations to service client’s request are performed in one step, ie, either all of them are executed or none are.
- Start of a transaction is a continuation point to which it can roll back.

Issues:
- Multiple concurrent clients: “isolation”.
  1. Each transaction accesses resources as if there were no other concurrent transactions.
  2. Modifications of the transaction are not visible to other resources before it finishes.
  3. Modifications of other transactions are not visible during the transaction at all.

- Server failures: “failure atomicity”.
Transaction Features

- **Recoverability**: server should be able to “roll back” to state before transaction execution.

- **Serializability**: transactions executing concurrently must be interleaved in such a way that the resulting state is equal to some serial execution of the transactions.

- **Durability**: effects of transactions are permanent.
  - A completed transaction is always persistent (though values may be changed by later transactions).
  - Modified resources must be held on persistent storage before transaction can complete. May not just be disk but can include battery-backed RAM.
Concurrency Control

Maintain transaction serializability:

- establish order of concurrent transaction execution
- Interleave execution of operations to ensure serializability

- Basic Server operations: read or write.

- 3 mechanisms:
  - Locks.
  - Optimistic concurrency control.
  - Timestamp ordering.
Locks

- Lock granularity: affects level of concurrency.
  - 1 lock per shared data item.
  - Shared Read
    - Exists when concurrent transactions granted READ access
    - Issued when transaction wants to read and exclusive lock not held on item
  - Exclusive Write
    - Exists when access reserved for locking transaction
    - Used when potential for conflict exists
    - Issued when transaction wants to update unlocked data

- Many Read locks simultaneously possible for a given item, but only one Write lock

- Transaction that requests a lock that cannot be granted must wait
Lock Implementation

- Server lock manager
  - Maintains table of locks for server data items.
  - Lock and unlock operations.
  - Clients wait on a lock for given data until data is released; then client is signalled.
  - Each client’s request runs as separate server thread.
Deadlock

- Use of locks can lead to deadlock.
- **Deadlock**: each transaction waits for another transaction to release a lock forming a wait cycle.
- **Deadlock condition**: cycle in the wait-for graph.
- **Deadlock prevention and detection.**
  - require all locks to be acquired at once
  - Problems?
  
  - Ordering of data items: once a transaction locks an item, it cannot lock anything occurring earlier in the ordering

- **Deadlock resolution**: lock timeout.
Optimistic Concurrency Control

- Assume that most of the time, probability of conflict is low.
- Transactions allowed to proceed in parallel until close transaction request from client.
- Upon close transaction, checks for conflict; if so, some transactions aborted.
Optimistic Concurrency 2

- **Read phase**
  - Transactions have *tentative* version of data items it accesses.
    - Transaction reads data and stores in local variables
    - Any writes are made to local variables without updating the actual data
  - *Tentative* versions allow transactions to abort without making their effect permanent.

- **Validation phase**
  - Executed upon *close transaction*.
  - Checks serially equivalence.
  - If validation fails, conflict resolution decides which transaction(s) to abort.
Optimistic Concurrency 3

- **Write phase**
  - If transaction is validated, all of its tentative versions are made permanent.
  - Read-only transactions commit immediately.
  - Write transactions commit only after their tentative versions are recorded in permanent storage.
Timestamp Ordering

- Uses timestamps to order transactions accessing same data items according to their starting times.

- Assigning timestamps:
  - Clock based: assign global unique time stamp to each transaction
  - Monotonically increasing counter.

- Some time stamping necessary to avoid “livelock”: where a transaction cannot acquire any locks because of unfair waiting algorithm
Local versus Distributed Transactions

- Local transactions:
  - All transaction operations executed by single server.

- Distributed transactions:
  - Involve multiple servers.

- Both local and distributed transactions can be simple or nested.
  - Nesting: increase level of concurrency.
Distributed Transactions 1

Simple Distributed Transaction

Nested Distributed Transaction
Distributed Transactions 2

- Transaction coordinator
  - First server contacted by client.
  - Responsible for aborting/committing.
  - Adding workers.

- Workers
  - Other servers involved report their results to the coordinator and follow its decisions.
Atomicity in Distributed Transactions

- Harder: several servers involved.
- Atomic commit protocols
  - 1-phase commit
    - Example: coordinator sends “commit” or “abort” to workers; keeps re-broadcasting until it gets ACK from all of them that request was performed.
    - Inefficient.
    - How to ensure that all of the servers vote + that they all reach the same decision. It is simple if no errors occur, but the protocol must work correctly even when server fails, messages are lost, etc.
2-Phase Commit 1

- First phase: voting
  - Each server votes to commit or abort transaction.

- Second phase: carrying out joint decision.
  - If any server votes to abort, joint decision is to abort.
2-Phase Commit 2

- **Phase I:**
  - Each participant votes for the transaction to be committed or aborted.
  - Participants must ensure to carry out its part of commit protocol. (prepared state).
  - Each participant saves in permanent storage all of the objects that it has altered in transaction to be in 'prepared state'.

- **Phase II:**
  - Every participant in the transaction carries out the joint decision.
  - If any one participant votes to abort, then the decision is to abort.
  - If all participants vote to commit, then the decision is to commit.

---

**Coordinator**

1. Prepared to commit?
2. Prepared to commit/abort
3. Committed.
5. End.

**Workers**

- Can commit?
- Yes/No
- Do commit/abort
- Have committed/aborted
Concurrency Control in Distributed Transactions

Locks

- Each server manages locks for its own data.
- Locks cannot be released until transaction committed or aborted on all servers involved.
- Lock managers in different servers set their locks independently, there are chances of different transaction orderings.
- The different ordering lead to cyclic dependencies between transactions and a distributed deadlock situation.
- When a deadlock is detected, a transaction is aborted to resolve the deadlock.
Concurrency Control in Distributed Transactions 2

- Timestamp Ordering
  - Globally unique timestamps.
    - Coordinator issues globally unique TS and passes it around.
    - TS: <server id, local TS>
  - Servers are jointly responsible for ensuring that they performed in a serially equivalent manner.
  - Clock synchronization issues
Concurrency Control in Distributed Transactions 3

- **Optimistic concurrency control**
  - Each transaction should be validated before it is allowed to commit.
  - The validation at all servers takes place during the first phase of the 2-Phase Commit Protocol.
Camelot [Spector et al.]

- Supports execution of distributed transactions.
- Specialized functions:
  - Disk management
    - Allocation of large contiguous chunks.
  - Recovery management
    - Transaction abort and failure recovery.
  - Transaction management
    - Abort, commit, and nest transactions.
Distributed Deadlock 1

- When locks are used, deadlock can occur.
- Circular wait in wait-for graph means deadlock.
- Centralized deadlock detection, prevention, and resolutions schemes.
  - Examples:
    - Detection of cycle in wait-for graph.
    - Lock timeouts: hard to set TO value, aborting unnecessarily.
Distributed Deadlock 2

- Much harder to detect, prevent, and resolve. Why?
  - No global view.
  - No central agent.
  - Communication-related problems
    - Unreliability.
    - Delay.
    - Cost.
Distributed Deadlock Detection

- Cycle in the global wait-for graph.
- Global graph can be constructed from local graphs: hard!
  - Servers need to communicate to find cycles.
Distributed Deadlock Detection Algorithms 1

- [Chandy et al.]
- Message sequencing is preserved.
- Resource versus communication models.
  - Resource model
    - Processes, resources, and controllers.
    - Process requests resource from controller.
  - Communication model
    - Processes communicate directly via messages (request, grant, etc) requesting resources.
Resource versus Communication Models

- In resource model, controllers are deadlock detection agents; in communication model, processes.

- In resource model, process cannot continue until all requested resources granted; in communication model, process cannot proceed until it can communicate with at least one process it’s waiting for.

- Different models, different detection alg’s.
Distributed Deadlock Detection Schemes

- Graph-theory based.
- Resource model: deadlock when cycle among dependent processes.
- Communication model: deadlock when knot (all vertices that can be reached from $i$ can also reach $i$) of waiting processes.
Deadlock Detection in Resource Model

- Use probe messages to follow edges of wait-for graph (aka edge chasing).
- Probe carries transaction wait-for relations representing path in global wait-for graph.
Deadlock Detection Example

1. Server 1 detects transaction T is waiting for U, which is waiting for data from server 2.
2. Server 1 sends probe T->U to server 2.
3. Server 2 gets probe and checks if U is also waiting; if so (say for V), it adds V to probe T->U->V. If V is waiting for data from server 3, server 2 forwards probe.
4. Paths are built one edge at a time. Before forwarding probe, server checks for cycle (e.g., T->U->V->T).
5. If cycle detected, a transaction is aborted.
Replication requires synchronization

- Keep more than one copy of data item.
- Technique for improving performance in distributed systems.
- In the context of concurrent access to data, replicate data for increase availability.
  - Improved response time.
  - Improved availability.
  - Improved fault tolerance.
But nothing comes for free.
What’s the tradeoff?
- Consistency maintenance.
Consistency maintenance approaches:
  - Lazy consistency (gossip approach).
    - An operation call is executed at just one replica; updating of other replicas happens by lazy exchange of “gossip” messages.
  - Quorum consensus is based on voting techniques.
  - Process group.
Quorum Consensus

- **Goal:** prevent partitions from producing inconsistent results.

- **Quorum:** subgroup of replicas whose size gives it the right to carry out operations.

- **Quorum consensus replication:**
  - Update will propagate successfully to a subgroup of replicas.
  - Other replicas will have outdated copies but will be updated off-line.
Weighted Voting [Gifford] 1

- Every copy assigned a number of votes (weight assigned to a particular replica).
- Read: Must obtain $R$ votes to read from any up-to-date copy.
- Write: Must obtain write quorum of $W$ before performing update.
Weighted Voting 2

- $W > 1/2$ total votes, $R+W >$ total votes.
- Ensures non-null intersection between every read quorum and write quorum.
- Read quorum guaranteed to have current copy.
- Freshness is determined by version numbers.
Weighted Voting 3

On read:
- Try to find enough copies, i.e., total votes no less than R. Not all copies need to be current.
- Since it overlaps with write quorum, at least one copy is current.

On write:
- Try to find set of up-to-date replicas whose votes no less than W.
- If no sufficient quorum, current copies replace old ones, then update.
Time in Distributed Systems

- Notion of time is critical.
- “Happened before” notion.
  - Example: concurrency control using TSs.
  - “Happened before” notion is not straightforward in distributed systems.
    - No guarantees of synchronized clocks.
    - Communication latency.
Event Ordering

Lamport defines partial ordering (→):
1. If X and Y events occurred in the same process, and X comes before Y, then X→Y.
2. Whenever X sends a message to Y, then X→Y.
3. If X→Y and Y→Z, then X→Z.
4. X and Y are concurrent if X↔Y and Y→Z
Causal Ordering

- “Happened before” also called causal ordering.
- In summary, possible to draw happened-before relationship between events if they happen in same process or there’s chain of messages between them.
Logical Clocks

- Monotonically increasing counter.
- No relation with real clock.
- Each process keeps its own logical clock $C_p$ used to timestamp events.
Causal Ordering and
Logical Clocks

1. $C_p$ incremented before each event.
   
   $C_p = C_p + 1.$

2. When $p$ sends message $m$, it piggybacks $t = C_p$.

3. When $q$ receives $(m, t)$, it computes:

   $C_q = \max(C_q, t)$ before timestamping message receipt event.

Total Ordering

- Extending partial to total order.
- Global timestamps:
  - \((T_a, p_a)\), where \(T_a\) is local TS and \(p_a\) is the process id.
  - \((T_a, p_a) < (T_b, p_b) \) iff \(T_a < T_b\) or \(T_a = T_b\) and \(p_a < p_b\)
  - Total order consistent with partial order.
Virtual Time [Jefferson]

- Time warp mechanism.
- May or may not have connection with real time.
- Uses optimistic approach, i.e., events and messages are processed in the order received: “look-ahead”.
Local Virtual Clock

- Process virtual clock set to TS of next message in input queue.
- If next message’s TS is in the past, rollback!
  - Can happen due to different computation rates, communication latency, and unsynchronized clocks.
Rolling Back

- Process goes back to TS(last message).
- Cancels all intermediate effects of events whose TS > TS(last message).
- Then, executes forward.
- Rolling back is expensive!
  - Messages may have been sent to other processes causing them to send messages, etc.
Anti-Messages 1

- For every message, there is an anti-message with same content but different sign.
- When sending message, message goes to receiver input queue and a copy with “-” sign is enqueued in the sender’s output queue.
- Message is retained for use in case of roll back.
Anti-Message 2

- Message + its anti-message = 0 when in the same queue.
- Processes must keep log to “undo” operations.
Implementation

- Local control.
- Global control
  - How to make sure system as a whole progresses.
  - “Committing” errors and I/O.
  - Avoid running out of memory.
Global Virtual Clock

- Snapshot of system at given real time.
- Minimum of all local virtual times.
- Lower bound on how far processes rollback.
- Purge state before GVT.
- GVT computed concurrently with rest of time warp mechanism.
  - Tradeoff?
Goal: provide programming environment for development of distributed systems.

Assumptions:
- DS as a set of processes with disjoint address spaces, communicating over LAN via MP.
- Processes and nodes can crash.
- Partitions may occur.
ISIS 2

- Distinguishing feature: group communication mechanisms
  - Process group: processes cooperating in implementing task.
  - Process can belong to multiple groups.
  - Dynamic group membership.
Virtual Synchrony

- Real synchronous systems
  - Events (e.g., message delivery) occur in the same order everywhere.
  - Expensive and not very efficient.
- Virtual synchronous systems
  - Illusion of synchrony.
  - Weaker ordering guarantees when applications allow it.
Atomic Multicast 1

- All destinations receive a message or none.

- Primitives:
  - ABCAST: delivers messages atomically and in the same order everywhere.
  - CBCAST: causally ordered multicast.
    - "Happened before" order.
    - Messages from given process in order.
  - GBCAST
    - used by system to manage group addressing.
Other Features

- **Process groups**
  - Group membership management.

- **Broadcast and group RPC**
  - RPC-like interface to CBCAST, ABCAST, and GBCAST protocols.
  - **Delivery guarantees**
    - Caller indicates how many responses required.
      - No responses: asynchronous.
      - 1 or more: synchronous.
Implementation

- Set of library calls on top of UNIX.
- Commercially available.
- In the paper, example of distributed DB implementation using ISIS.
- HORUS: extension to WANs.
CSci555: Advanced Operating Systems
Lecture 4 – September 21 2012
Naming and Binding

Dr. Clifford Neuman
University of Southern California
Information Sciences Institute
Naming Concepts

- **Name**
  - What you call something

- **Address**
  - Where it is located

- **Route**
  - How one gets to it

  What is http://www.isi.edu/~bcn?

- But it is not that clear anymore, it depends on perspective. A name from one perspective may be an address from another.
  - Perspective means layer of abstraction
What are the things we name

- **Users**
  - To direct, and to identify
- **Hosts (computers)**
  - High level and low level
- **Services**
  - Service and instance
- **Files and other “objects”**
  - Content and repository
- **Groups**
  - Of any of the above
How we name things

- **Host-Based Naming**
  - Host-name is required part of object name

- **Global Naming**
  - Must look-up name in global database to find address
  - Name transparency

- **User/Object Centered Naming**
  - Namespace is centered around user or object

- **Attribute-Based Naming**
  - Object identified by unique characteristics
  - Related to resource discovery / search / indexes
Namespace

- A name space maps:

\[ \sum^* \rightarrow X \in \mathcal{O} \]

At a particular point in time.

- The rest of the definition, and even some of the above, is open to discussion/debate.

- What is a “flat namespace”
  - Implementation issue
Case Studies

- **Host Table**
  - Flat namespace (?)
  - Global namespace (?)

- **Grapevine**
  - Two-level, iterative lookup
  - Clearinghouse 3 level

- **Domain name system**
  - Arbitrary depth
  - Iterative or recursive(chained) lookup
  - Multi-level caching

```cpp
GetHostByName(usc.arpa){
    scan(host file);
    return(matching entry);
}
```

```
GetHostByName(host.sc)
```

```cpp
GetHostByName(host.sc)
```
Domain Name System

Iterative query

Lookup(venera.isi.edu)
Chained query

Lookup(venera.isi.edu)
Scalability of naming

❖ Scalability
  ❑ Ability to continue to operate efficiently as a system grows large, either numerically, geographically, or administratively.

❖ Affected by
  ❑ Frequency of update
  ❑ Granularity
  ❑ Evolution/reconfiguration

❖ DNS characteristics
  ❑ Multi-level implementation
  ❑ Replication of root and other servers
  ❑ Multi-level caching
Closure

- Closure binds an object to the namespace within which names embedded in the object are to be resolved.
  - “Object” may as small as the name itself
    - GNS binds the names to namespaces
    - Prospero binds enclosing object to multiple namespaces
    - Tilde and quicksilver bind users to namespaces
    - NFS mount table constructs system centered namespace
  - Movement of objects can cause problems
    - When closure is associated with wrong entity
Other implementations of naming

- **Broadcast**
  - Limited scalability, but faster local response

- **Prefix tables**
  - Essentially a form of caching

- **Capabilities**
  - Combines security and naming
  - Traditional name service built over capability based addresses
Advanced Name Systems

- DEC’s Global Naming
  - Support for reorganization the key idea
  - Little coordination needed in advance

- Half Closure
  - Names are all tagged with namespace identifiers
    - DID - Directory Identifier
    - Hidden part of name - makes it global
    - Upon reorganization, new DID assigned
    - Old names relative to old root
  - But the DID’s must be unique - how do we assign?
Prospero Directory Service

- Multiple namespace centered around a “root” node that is specific to each namespace.
  - Closure binds objects to this “root” node.

- Layers of naming
  - User level names are “object” centered
  - Objects still have an address which is global
  - Namespaces also have global addresses

- Customization in Prospero
  - Filters create user level derived namespaces on the fly
  - Union links support merging of views
Resource Discovery

- Similar to naming
  - Browsing related to directory services
  - Indexing and search similar to attribute based naming

- Attribute based naming
  - Profile
  - Multi-structured naming

- Search engines

- Computing resource discovery
The Web

- **Object handles**
  - Uniform Resource Identifier (URI’s)
  - Uniform Resource Locators (URL’s)
  - Uniform Resource Names (URN’s)

- **XML**
  - Definitions provide a form of closure
    - Conceptual level rather than the “namespace” level.
LDAP

- Manage information about users, services
  - Lighter weight than X.500 DAP
    - Heavier than DNS
  - Applications have conventions on where to look
    - Often data is duplicated because of multiple conventions
  - Performance enhancements not as well defined
    - Caching harder because of less constrained patterns of access
New Computing Environments

- Ubiquitous and Pervasive Computing
  - Including Sensor Nodes
  - Managing Devices
- Mobile computing
  - Portable Devices
Characteristics

- Low power availability
- Constrained resources
- Transient relationships
- Ad-hoc deployment
- Peer to Peer relationships
- Weakly managed
- Context aware
Ubiquitous computing

- According to Mark Weiser at Xerox:
  - Transparent computing is the ultimate goal
  - Computers should disappear into the background
  - Computation becomes part of the environment
Ubiquitous Computing

- Computing everywhere
  - Desktop, Laptop, Palmtop
  - Cars, Cell phones
  - Shoes, Clothing, Walls (paper / paint)
- Connectivity everywhere
  - Broadband
  - Wireless
- Mobile everywhere
  - Users move around
  - Disposable devices
Ubiquitous Computing

❖ Structure
   - Resource and service discovery critical
   - User location an issue
   - Interface discovery
   - Disconnected operation
   - Ad-hoc organization

❖ Security
   - Small devices with limited power
   - Intermittent connectivity

❖ Agents

❖ Sensor Networks
Mobile Computing

- Often managed devices
  - Cell phones
  - PDA’s
  - Subscription Services
Ad-hoc Networking

- Peer-to-peer of network routing
- Transient devices
- Issues:
  - Discovery
  - Security
  - Power
- Examples:
  - Many Sensor Networks
Assume: 10kbit/sec. Radio, 10 m range.

**Large cost of communications relative to computation continues**

*(source – talk by Deborah Estrin)*
Active Badges

- Provides location information and one or more buttons
  - Beacons

- Simple interface
  - Controlled access
  - Carries context
    - Two way, device knows where your are, and your location knows you are present
    - Moves your environment
Today’s Mobile Devices

- Smart Phones
  - Communication
    - Wifi, 3G, 4G
    - Mostly infrastructure based
  - Location Services & Power
    - GPS, Cell Tower, Beacons, Inertial Navigation
In-Network Processing

- Communication is expensive, computing less-so, so pre-process to reduce data sent.
- Send information, not-data.
- Requires more knowledge at the edges so that query can be distributed.
- Intermediate nodes correlate and aggregate results.
Directed Diffusion

- Publish/Subscribe model
- Data named, not nodes
- But what are the implications
  - discussion
A Taxonomy

- Approaches/Products/Devices differ by placement/nature of:
  - Management
  - Storage
  - Computing
  - Communication
  - Context maintenance
  - Authority
Classes

- Mobile Terminals
- Passive devices
- Personal Devices
- Remote Sensors/Actuators
- Communicating Devices
- Sensor Networks
Examples

- Cell Phones
- i-phone
- PDA
- Home automation
- Proximity cards
- Laptop computer
- In Vehicle networks
- Active Badges
Security Goals

- Confidentiality
  - inappropriate information is not disclosed

- Integrity
  - Authenticity of document
  - That it hasn’t changed

- Availability
  - the ability of authorized entities to use the information or resource
System Security: Terminology

- **vulnerability** is a weakness in the system that might be exploited to cause loss or harm.
- **threat** is a potential violation of security and includes a capability to exploit a vulnerability.
- **attack** is the actual attempt to violate security. It is the manifestation of the threat:
  - Interception
  - Modification
  - Disruption

- **security policy** defines what is and is not allowed
- **security mechanism** is a method or tool for enforcing security policy:
  - Prevention
  - Detection
  - Reaction
Basic Security Services

Protection
Authentication
Access Control, Authorization
Accounting
Payment
Audit
Assurance
Privacy
Policy
Security Models

- **Discretionary Access Control**
  - Users have complete control over his/her resources

- **Mandatory Access Control**
  - Administrators decide what you have access to as well as what you can give access to (as opposed to discretionary access control).
  - Users must deal with not having control over how they use their own resources.
Security Policy

Access Matrix

- implemented as:
  - Capabilities or
  - Access Control list

<table>
<thead>
<tr>
<th>Subject</th>
<th>OBJ1</th>
<th>OBJ2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcn</td>
<td>RW</td>
<td>R</td>
</tr>
<tr>
<td>gost-group</td>
<td>RW</td>
<td>-</td>
</tr>
<tr>
<td>obraczka</td>
<td>R</td>
<td>RW</td>
</tr>
<tr>
<td>tyao</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Csci555</td>
<td>R</td>
<td>-</td>
</tr>
</tbody>
</table>
Access Control Lists

❖ Advantages
  □ Easy to see who has access
  □ Easy to change/revoke access

❖ Disadvantages
  □ Time consuming to check access

❖ Extensions to ease management
  □ Groups
  □ EACLs
Extended Access Control Lists

- Conditional authorization
  - Implemented as restrictions on ACL entries and embedded as restrictions in authentication and authorization credentials

<table>
<thead>
<tr>
<th>Principal</th>
<th>Rights</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcn</td>
<td>RW</td>
<td>HW-Authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retain Old Items</td>
</tr>
<tr>
<td>post-group</td>
<td>RW</td>
<td>TIME: 9AM-5PM</td>
</tr>
<tr>
<td>authorization server</td>
<td>R</td>
<td>Delegated-Access</td>
</tr>
<tr>
<td>*</td>
<td>R</td>
<td>Load Limit 8 Use: Non-Commercial</td>
</tr>
<tr>
<td>*</td>
<td>R</td>
<td>Payment: $Price</td>
</tr>
</tbody>
</table>
Example Conditions

- **Authentication method** specifies mechanisms suitable for authentication.
- **Payment** specifies currency and amount.
- **Time** time periods expressed as time of day or days of week when access is granted.
- **Location** access is granted to principals connecting from specific hosts.
- **Notification** enables automatic generation of notification messages.
- **Audit** enables automatic generation of application level audit data.
- **System Threat Level** specifies system threat level, e.g., high, medium or low.
Capabilities

❖ Advantages
  ❑ Easy and efficient to check access
  ❑ Easily propagated

❖ Disadvantages
  ❑ Hard to protect capabilities
  ❑ Easily propagated
  ❑ Hard to revoke

❖ Hybrid approach
  ❑ EACL’s/proxies
Protecting capabilities

- **Stored in TCB**
  - Only protected calls manipulate
- **Limitations ?**
  - Works in centralized systems
- **Distributed Systems**
  - Tokens with random or special coding
  - Possibly protect through encryption
  - How does Amoeba do it? (claimed)
Network Threats

- Unauthorized release of data
- Unauthorized modification of data
- Impersonation (spurious association initiation)
- Denial of use
- Traffic analysis

- Attacks may be
  - Active or passive
Likely points of attack (location)
Likely points of attack (module)

- Against the protocols
  - Sniffing for passwords and credit card numbers
  - Interception of data returned to user
  - Hijacking of connections
- Against the server
  - The commerce protocol is not the only way in
  - Once an attacker is in, all bets are off
- Against the client’s system
  - You have little control over the client’s system
Network Attacks

Eavesdropping
   Listening for passwords or credit card numbers
Message stream modification
   Changing links and data returned by server
Hijacking
   Killing client and taking over connection
Network Attack Countermeasures

Don’t send anything important
Not everything needs to be protected

Encryption
For everything else
Mechanism limited by client side software
Encryption for confidentiality and integrity

- Encryption used to scramble data

**Diagram:**

plaintext → encryption → ciphertext → decryption → plaintext

(Encryption) + (Key) → (Ciphertext) + (Key) → (Decryption)
Authentication

- Proving knowledge of encryption key
  - Nonce = Non repeating value

\[ \{\text{Nonce or timestamp}\}K_{c.s} \]
Today’s security deployment

Most of the deployment of security services today handles the easy stuff, implementing security at a single point in the network, or at a single layer in the protocol stack:

- Firewalls, VPN’s
- IPSec
- SSL

Unfortunately, security isn’t that easy. It must be better integrated with the application.

- At the level at which it must ultimately be specified, security policies pertain to application level objects, and identify application level entities (users).
Conclusion: Integration is hard to do

- The majority of applications were not being modified to use security services.
  - In fact, the only widespread interoperable integration of security services with applications was SSL integration with the web, and SSL is used primarily as a confidentiality mechanism and only rarely for user authentication.
Conclusion: Integration is hard to do

❖ The reason
  ❖ Integration with applications involved many changes:
    ▪ Multiple calls to GSS-API or other authentication interfaces
    ▪ Calls to decide what the user is authorized to do
      – Home grown policy databases or protocol extensions requiring even more calls to complete.
    ▪ Custom integration with other security services
      – Confidentiality, integrity, payment, audit
Focus on Authorization

- Focusing on authorization and the management of policies used in the authorization decision.
  - Not really new - this is a reference monitor.
  - Applications shouldn’t care about authentication or identity.
    - Separate policy from mechanism
  - Authorization may be easier to integrate with applications.
  - Hide the calls to the key management and authentication functions.
Kerberos

Third-party authentication service
- Distributes session keys for authentication, confidentiality, and integrity

1. Req
2. T+{Reply}Kc
3. TgsReq
4. Ts+{Reply}Kt
5. Ts + {ts}Kcs
Global Authentication Service

- Pair-wise trust in hierarchy
  - Name is derived from path followed
  - Shortcuts allowed, but changes name
  - Exposure of path is important for security

- Compared to Kerberos
  - Transited field in Kerberos - doesn’t change name

- Compared with X.509
  - X.509 has single path from root
  - X.509 is for public key systems

- Compared with PGP
  - PGP evaluates path at end, but may have name conflicts
Capability Based Systems - Amoeba

“Authentication not an end in itself”

- Theft of capabilities an issue
  - Claims about no direct access to network
  - Replay an issue
- Modification of capabilities a problem
  - One way functions provide a good solution
- Where to store capabilities for convenience
  - In the user-level naming system/directory
  - 3 columns
- Where is authentication in Amoeba
  - To obtain initial capability
Capability Directories in Amoeba
Distributed Authorization

- It must be possible to maintain authorization information separate from the end servers
  - Less duplication of authorization database
  - Less need for specific prior arrangement
  - Simplified management
- Based on restricted proxies which support
  - Authorization servers
  - Group Servers
  - Capabilities
  - Delegation
Security Architectures

- **DSSA**
  - Delegation is the important issue
    - Workstation can act as user
    - Software can act as workstation - if given key
    - Software can act as developer - if checksum validated
  - Complete chain needed to assume authority
  - Roles provide limits on authority - new sub-principal

- **Proxies** - Also based on delegation
  - Limits on authority explicitly embedded in proxy
  - Works well with access control lists
Next Generation Secure Computing Base -> Longhorn -> Vista

- Secure booting provides known hardware and OS software base.
- Security Kernel in OS provides assurance about the application.
- Security Kernel in application manages credentials granted to application.
- Security servers enforce rules on what software they will interact with.
Hardware Requirements for Trusted Computing

How do we know what software is running?

- Exercise: Suppose a virus infected our application.
- Suppose we were running a hacked kernel.
- Suppose we were running in a virtual machine.
- Suppose someone replaced our hardware with something similar, but with special operations.
A basis step for trust

- The Hardware must be secure.
  - And it must be able to prove that it has not been modified or replaced.
  - This requires special keys accessible only to the hardware.
  - It requires tamper resistance that destroys the key if someone tries to open the chip.
  - (we also need validation of the hardware implementation)
But this is an OS class

- The OS must be secure
  - And it must be able to prove that it has not been modified or replaced.
  - This requires special keys accessible to OS.
    - These keys are provided when booted.
    - They are bound to a checksum.
    - They may be managed in hardware or by the OS itself.
  - What does this protect against?
What does the OS do

- The OS must perform special functions
  - Must provide certified keys for the applications (as the HW did for the OS).
  - The OS must protect itself, and its own keys – so that malware or other users can not act as the OS.
  - The OS must protect process from one another. Some functions may require stronger separation than typically provided today.
  - The Trusted Applications themselves must similarly apply application specific protections to the data they manipulate.
OS Concepts

- Trusted computing base
- Trusted path
- Separation of processes
The Trusted Computing Bases (TCB)

- That part of the system which is critical for security.
  - Vulnerability of the TCB affects the core security of the system.
  - Trusted Computing Extends the TCB across physical system boundaries.
    - Allows remote components to be part of the TCB for a particular function.
Trusted Path

- Provides attestation of the system to the user.
  - Requires confidence in the hardware by the user.
  - Requires training of the user on how to invoke trusted path.
Separation of Processes

- Allows process that are trusted to run without interference from other processes.
  - Requires isolation that is provided by lower level trusted modules.
  - Include hardware support, much of which is already standard in chips, but some which is not.
Vista Security Technologies

- Summary of some of the support for trusted computing in Vista (on the following slides)
Trusted Platform Module (TPM)?

Smartcard-like module on the motherboard that:

- Performs cryptographic functions
  - RSA, SHA-1, RNG
  - Meets encryption export requirements
- Can create, store and manage keys
  - Provides a unique Endorsement Key (EK)
  - Provides a unique Storage Root Key (SRK)
- Performs digital signature operations
- Holds Platform Measurements (hashes)
- Anchors chain of trust for keys and credentials
- Protects itself against attacks

TPM 1.2 spec: www.trustedcomputinggroup.org

Slide From Steve Lamb at Microsoft
Why Use A TPM?

- Trusted Platforms use Roots-of-Trust
  - A TPM is an implementation of a Root-of-Trust
- A hardware Root-of-Trust has distinct advantages
  - Software can be hacked by Software
    - Difficult to root trust in software that has to validate itself
  - Hardware can be made to be robust against attacks
    - Certified to be tamper resistant
  - Hardware and software combined can protect root secrets better than software alone
- A TPM can ensure that keys and secrets are only available for use when the environment is appropriate
  - Security can be tied to specific hardware and software configurations

Slide From Steve Lamb at Microsoft
Disk Layout & Key Storage

Windows Partition Contains
- Encrypted OS
- Encrypted Page File
- Encrypted Temp Files
- Encrypted Data
- Encrypted Hibernation File

Where's the Encryption Key?
1. SRK (Storage Root Key) contained in TPM
2. SRK encrypts VEK (Volume Encryption Key) protected by TPM/PIN/Dongle
3. VEK stored (encrypted by SRK) on hard drive in Boot Partition

Boot Partition Contains: MBR, Loader, Boot Utilities (Unencrypted, small)

Slide From Steve Lamb at Microsoft
BitLocker™ Architecture
Static Root of Trust Measurement of early boot components

Slide From Steve Lamb at Microsoft

[Diagram showing the BitLocker™ Architecture with stages including PreOS, Static OS, and Volume Blob of Target OS, with components like TPM Init, BIOS, MBR, BootSector, BootBlock, BootManager, and OS Loader]
Hardware Requirements for Trusted Computing

How do we know what software is running?

- Exercise: Suppose a virus infected our application.
- Suppose we were running a hacked kernel.
- Suppose we were running in a virtual machine.
- Suppose someone replaced our hardware with something similar, but with special operations.
This helps to contain breaches

- But it doesn’t prevent breaches.
  - A buggy OS can still be compromised.
  - Bugs in applications still leave vulnerabilities.
  - But one can at least be more sure about what version of software one is communicating with.
Virtualization and Trusted Computing

- The separation provided by virtualization may be just what is needed to keep data managed by trusted applications out of the hands of other processes.
- But a trusted Guest OS would have to make sure the data is protected on disk as well.
CSci555: Advanced Operating Systems
Lecture 7 – October 12 2012
Virtualization

Dr. Clifford Neuman
University of Southern California
Information Sciences Institute
Virtualization and Trusted Computing

- The separation provided by virtualization may be just what is needed to keep data managed by trusted applications out of the hands of other processes.
- But a trusted Guest OS would have to make sure the data is protected on disk as well.
Protecting Data Within an OS

- Trusted computing requires protection of processes and resources from access or modification by untrusted processes.
  - Don’t allow running of untrusted processes
    - Limits the usefulness of the OS
    - But OK for embedded computing
  - Provide strong separation of processes
    - Together with data used by those processes
  - Protection of data as stored
    - Encryption by OS / Disk
    - Encryption by trusted application
    - Protection of hardware, and only trusted boot
Protection by the OS

- The OS provides
  - Protection of its own data, keys, and those of other applications.
    - The OS protect process from one another. Some functions may require stronger separation than typically provided today, especially from “administrator”.
  - The trusted applications themselves must similarly apply application specific protections to the data they manipulate.
Strong Separation

- **OS Support**
  - Ability to encrypt parts of file system
  - Access to files strongly mediated
  - Some protections enforced against even “Administrator”

- **Mandatory Access Controls**
  - Another form of OS support
  - Policies are usually simpler

- **Virtualization**
Virtualization

- Operating Systems are all about virtualization

- One of the most important function of a modern operating system is managing virtual address spaces.

- But most operating systems do this for applications, not for other OSs.
Virtualization of the OS

- Some have said that all problems in computer science can be handled by adding a later of indirection.
  - Others have described solutions as reducing the problem to a previously unsolved problem.
- Virtualization of OS’s does both.
  - It provides a useful abstraction for running guest OS’s.
  - But the guest OS’s have the same problems as if they were running natively.
What is the benefit of virtualization

- Management
  - You can running many more “machines” and create new ones in an automated manner.
  - This is useful for server farms.

- Separation
  - “Separate” machines provide a fairly strong, though coarse grained level of protection.
  - Because the isolation can be configured to be almost total, there are fewer special cases or management interfaces to get wrong.
Is Virtualization Different?

- Same problems
  - Most of the problems handled by hypervisors are the same problems handled by traditional OS’s

- But the Abstractions are different
  - Hypervisors present a hardware abstraction.
    - E.g. disk blocks
  - OS’s present and application abstraction.
    - E.g. files
Virtualization

- Running multiple operating systems simultaneously.
  - OS protects its own objects from within
  - Hypervisor provides partitioning of resources between guest OS’s.
Managing Virtual Resource

- Page faults typically trap to the Hypervisor (host OS).
  - Issues arise from the need to replace page tables when switching between guest OS’s.
  - Xen places itself in the Guest OS’s first region of memory so that the page table does not need to be rewritten for traps to the Hypervisor.
- Disks managed as block devices allocated to guest OS’s, so that the Xen code to protect disk extents can be as simple as possible.
Virtualization

- Operating Systems are all about virtualization
  - One of the most important functions of a modern operating system is managing virtual address spaces.
  - But most operating systems do this for applications, not for other OSs.
Virtualization of the OS

- Some have said that all problems in computer science can be handled by adding a layer of indirection.
  - Others have described solutions as reducing the problem to a previously unsolved problem.
- Virtualization of OS’s does both.
  - It provides a useful abstraction for running guest OS’s.
  - But the guest OS’s have the same problems as if they were running natively.
What is the benefit of virtualization

- **Management**
  - You can run many more “machines” and create new ones in an automated manner.
  - This is useful for server farms.

- **Separation**
  - “Separate” machines provide a fairly strong, though coarse grained level of protection.
  - Because the isolation *can be* configured to be almost total, there are fewer special cases or management interfaces to get wrong.
What makes virtualization hard

- Operating systems are usually written to assume that they run in privileged mode.
- The Hypervisor (the OS of OS’s) manages the guest OS’s as if they are applications.
- Some architecture provide more than two “Rings” which allows the guest OS to reside between the two states.
  - But there are still often assumptions in coding that need to be corrected in the guest OS.
Managing Virtual Resource

Page faults typically trap to the Hypervisor (host OS).

- Issues arise from the need to replace page tables when switching between guest OS’s.
- Xen places itself in the Guest OS’s first region of memory so that the page table does not need to be rewritten for traps to the Hypervisor.

Disks managed as block devices allocated to guest OS’s, so that the Xen code protects disk extents and is as simple as possible.
Partitioning of Resources

- Fixed partitioning of resources makes the job of managing the Guest OS’s easier, but it is not always the most efficient way to partition.
  - Resources unused by one OS (CPU, Memory, Disk) are not available to others.
- But fixed provisioning prevents use of resources in one guest OS from effecting performance or even denying service to applications running in other guest OSs.
The Security of Virtualization

+++ Isolation and protection between OS’s can be simple (and at a very coarse level of granularity).

+++ This coarse level of isolation may be an easier security abstraction to conceptualize than the finer grained policies typically encountered in OSs.

--- Some malware (Blue pill) can move the real OS into a virtual machine from within which the host OS (the Malware) can not be detected.
Virtualization and Trusted Computing

- The separation provided by virtualization may be just what is needed to keep data managed by trusted applications out of the hands of other processes.
- But a trusted Guest OS would have to make sure the data is protected on disk as well.
Examples of Virtualization

❖ VMWare
  ❖ Guest OS’s run under host OS
  ❖ Full Virtualization, unmodified Guest OS

❖ Xen
  ❖ Small Hypervisor as host OS
  ❖ Para-virtualization, modified guest OS

❖ Terra
  ❖ A Virtual Machine-Based TC platform

❖ Denali
  ❖ Optimized for application sized OS’s.
An x86 virtual machine monitor

- Allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion,

- Provides an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be ported with minimal effort.

- Design supports 100 virtual machine instances simultaneously on a modern server.

(Slides primarily from XEN website http://www.cl.cam.ac.uk/research/srg/netos/xen/architecture.html)
Para-Virtualization in Xen

- Xen extensions to x86 arch
  - Like x86, but Xen invoked for privileged ops
  - Avoids binary rewriting
  - Minimize number of privilege transitions into Xen
  - Modifications relatively simple and self-contained

- Modify kernel to understand virtualised env.
  - Wall-clock time vs. virtual processor time
    - Desire both types of alarm timer
  - Expose real resource availability
    - Enables OS to optimise its own behaviour
Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a XenoLinux environment.
Xen 3.0 Architecture

Arun Viswanathan
(Slides primarily from XEN website
http://www.cl.cam.ac.uk/research/srg/netos/xen/architecture.html)
<table>
<thead>
<tr>
<th><strong>Memory Management</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.</td>
</tr>
<tr>
<td>Paging</td>
<td>Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CPU</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection</td>
<td>Guest OS must run at a lower privilege level than Xen.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.</td>
</tr>
<tr>
<td>System Calls</td>
<td>Guest OS may install a ‘fast’ handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Hardware interrupts are replaced with a lightweight event system.</td>
</tr>
<tr>
<td>Time</td>
<td>Each guest OS has a timer interface and is aware of both ‘real’ and ‘virtual’ time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Device I/O</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Network, Disk, etc.</td>
<td>Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.</td>
</tr>
</tbody>
</table>
Xen reserves top of VA space
Segmentation protects Xen from kernel
System call speed unchanged
Xen 3 now supports PAE for >4GB mem
x86 CPU virtualization

- Xen runs in ring 0 (most privileged)
- Ring 1/2 for guest OS, 3 for user-space
  - GPF if guest attempts to use privileged instr
- Xen lives in top 64MB of linear addr space
  - Segmentation used to protect Xen as switching page tables too slow on standard x86
- Hypercalls jump to Xen in ring 0
- Guest OS may install ‘fast trap’ handler
  - Direct user-space to guest OS system calls
- MMU virtualisation: shadow vs. direct-mode

(Slides primarily from XEN website http://www.cl.cam.ac.uk/research/srg/netos/xen/architecture.html)
Para-Virtualizing the MMU

- Guest OSes allocate and manage own PTs
  - Hypercall to change PT base
- Xen must validate PT updates before use
  - Allows incremental updates, avoids revalidation
- Validation rules applied to each PTE:
  1. Guest may only map pages it owns*
  2. Pagetable pages may only be mapped RO
- Xen traps PTE updates and emulates, or ‘unhooks’ PTE page for bulk updates

Arun Viswanathan
(Slides primarily from XEN website
http://www.cl.cam.ac.uk/research/srg/netos/xen/architecture.html)
Denali

- Whitaker, Shaw, Gribble at University of Washington
  - Observation is that conventional Operating Systems do not provide sufficient isolation between processes.
- So, Denali focuses on use of virtualization to provide strong isolation:
  - Content and information
  - Performance
- Resource sharing itself is not the focus.
Denali
Denali Philosophy

- Run each service in a separate VM
  - Much easier to provide isolation than to use traditional OS functions which are designed more for sharing.
  - Approximation of separate hardware
  - Only low level abstractions
    - Fewer bugs or overlooked issues
Isolation Kernel

- Goes beyond, but does less than Virtual Machine Monitor
  - Don’t emulate physical hardware
  - Leave namespace isolation, hardware API running on hardware
- Isolation Kernel provides
  - Isolated resource management
How they do it

- Eliminate unnecessary parts of “hardware architecture” in the isolation kernel.
  - Segmentation, Rings, BIOS
- Change others
  - Interrupts, Memory Management
- Simplify some
  - Ethernet only supports send and receive
Comparison to Linux

Source Code Complexity

From 2002 OSDI Talk, Andrew Whitaker
Observation on Denali

- Small overhead for virtualization
  - Most costs are in network stack and physical devices
  - Ability to support huge number of virtual (guest) OS’s.
    - This means it is OK to run individual applications in separate OS.
- At time of OSDI paper, Guest OS was only a library, with no simulated protection boundary.
  - Supports a POSIX subset.
VMWare

- Goals - provide ability to run multiple operating systems, and to run untrusted code safely.
  - Isolation primarily from guest OS to the outside.
  - This can provide isolation between guest OS’s.
  - Often configured to run inside a larger host OS, but also support a VMM layer as an option.
VMWare Memory Virtualization

- Intercepts MMU manipulating functions such as functions that change page table or TLB
- Manages shadow page tables with VM to Machine Mappings
- Kept in sync using physical to page mappings of VMM.
Terra: A Virtual Machine-Based Platform for Trusted Computing

- Similar to 2004 NGSCB architecture, supports multiple, isolated compartments
  - Terra supports an arbitrary number of user-defined VMs, more flexible than NGSCB
- Provides both “open-” and “closed-box” environments
- Implemented on VMware but didn’t actually use TPM

Slide by Michael LeMay – University of Illinois
Terra Architecture
Terra Approach

- **TVMM**: Trusted Virtual Machine Monitor
- **Open-box VMs**:
  - Just like current GP systems, no protection
- **Closed-box VMs**:
  - VM protected from modification, inspection
  - Can attest to remote peer that VM is protected
  - Behaves like true closed-box, but with cost and availability benefits of open-box

*Slide by Michael LeMay – University of Illinois*
TVMM Attestation

- Each layer of software has a keypair
- Lower layers certify higher layers
- Enables attestation of entire stack

Certificate

<table>
<thead>
<tr>
<th>Hardware (TPM)</th>
<th>Firmware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootloader</td>
<td>TVMM (Terra)</td>
</tr>
<tr>
<td>Operating System</td>
<td>VM</td>
</tr>
<tr>
<td>Application</td>
<td></td>
</tr>
</tbody>
</table>

Hash of Attestable Data

Higher Public Key

Other Application Data

Signed by Lower Level

Slide by Michael LeMay – University of Illinois
Terra - Additional Benefits

- Software stack can be tailored on per-application basis
  - Game can run on thin, high-performance OS
  - Email client can run on highly-secure, locked-down OS
  - Regular applications can use standard, full-featured and permissively-configured OS
- Applications are isolated and protected from each other
  - Reduces effectiveness of email viruses and spyware against system as a whole
- Low-assurance applications can automatically be transformed into medium-assurance applications, since they are protected from external influences
Terra Example

- Online gaming: Quake
- Players often modify Quake to provide additional capabilities to their characters, or otherwise cheat
- Quake can be transformed into a closed-box VM and distributed to players
- Remote attestation shows that it is unmodified
- Very little performance degradation
- Covert channels remain, such as frame rate statistics