CS 5523 Operating Systems: Midterm II - review

Instructor: Dr. Tongping Liu
Department Computer Science
The University of Texas at San Antonio

Spring 2015
Midterm on 04/07

- Come here around 5:45pm
- It may last until 7:30pm
Outline

- Memory Management and Advanced Topics: 40
- Intro of DS and communications: 45
- Naming: 20
Memory Management and Advanced Topics

- Simple memory management: swap etc.
- Virtual memory and paging
  - Virtual address and physical address
  - Page table and address translation
  - Translation lookaside buffer (TLB)
  - How to integrate TLB, cache with main memory
- How to design the page table?
  - What is size of a frame?
- Working set of processes? How to decide the working set?
Memory Management and Advanced Topics

- Page replacement policy
  - LRU
  - FIFO
  - Optimal
- Kernel memory management
  - Buddy and Slabs
- False sharing vs. true sharing
Introduction to DS

- Different Distributed Systems
  - Distributed computing systems
  - Distributed information systems
  - Distributed pervasive systems

- OS in distributed systems
  - Distributed OS vs. Network OS vs. Middleware

- Design objectives of distributed systems
  - Transparency, openness and scalability

- Architecture of distributed systems
  - Software vs. system architectures
Communication: network

- Layered network models
  - OSI 7-layer model (Open System Interconnection)
- Ethernet: local area network
- Inter-network Protocols (IP)
  - Addressing and routing etc.
- TCP/UDP protocols
  - communication ports and sockets
- Multicast: more than one recipients
Communication: applications

■ Fundamentals

➢ Client/Server communication protocols
  ✓ Request vs. Request-reply vs. Request-reply-acknowledge

➢ Invocation semantics
  ✓ Exact once vs. at least once vs. at most once

➢ Communication types
  ✓ Transient vs. persistent
  ✓ Synchronous vs. asynchronous

■ Models for application communications

➢ RPC: remote procedure call
➢ Message-oriented communication
➢ Stream-Oriented communication
➢ Multicast communication
Remote Objects and RMI

- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects;
Naming

- Overview: name and name services
  - Naming space and implementation

- Flat name and simple resolutions

- Structure name
  - Name space and resolution
  - Case study: DNS

- Attributed-based naming
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations
Details of Different Topics
Memory Management and Advanced Topics

- Simple memory management: swap etc.
- Virtual memory and paging
  - Virtual address and physical address
  - Page table and address translation
  - Translation lookaside buffer (TLB)
  - How to integrate TLB, cache with main memory
- How to design the page table
- Working set of processes? How to decide the working set?
Memory Management and Advanced Topics

- Page replacement policy
  - LRU
  - FIFO
  - Optimal

- Kernel memory management
  - Buddy and Slabs

- False sharing vs. true sharing
Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.

- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.

- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.

- Swapping would be needed to free up memory for additional processes.
Motivations for Virtual Memory

- **Use physical DRAM as cache for the disk**
  - Virtual pages of processes can exceed physical memory size

- **Simplify memory management**
  - Multiple processes resident in main memory
    - Each with its own address space
  - Only “active” code and data is actually in memory

- **Provide protection**
  - One process can’t interfere with another
    - Because they operate in different address spaces
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions
Address Translation Architecture

How big the page table is?
Page Table Size

Modern Systems/Applications
- 32 bits virtual address
- System with 1GB physical memory → 30 bits physical address
- Suppose the size of one page/frame is 4KB (12 bits)

Page table size
- # of virtual pages: $32 - 12 = 20$ bits → $2^{20}$ PTEs
- Page table size = PTE size * $2^{20} = 4 \text{ MB per process} → 2^{10}$ frames

If there are 128 processes
- Page tables occupy $128 \times 4\text{MB} = 512$ MB
- 50% of memory will be used by page tables?

How can we get smaller page table?!
Memory Requirement of Page Tables

- Only the 1st level page table and the required 2nd level page tables need to be in memory.

Example: a process with working-set size of 32 MB (recall that 1GB memory and 32 bits virtual address)

- 4KB / page $\rightarrow$ process has 8KB ($8 \times 2^{10}$) virtual pages
- One 2nd level page table maps $2^{10}$ pages;
- Number (minimum) of 2nd level page table needed: 8
- Total (minimum) memory for page table: 1st level page table + 8; in total of 9 page tables $\rightarrow$ 9 X 4KB = 36 KB

Department of Computer Science @ UTSA
Page table size

- 32bit machine, page size 4k, each entry 4 bytes, one level page table (full 4GB linear address)

Page table size = \(2^{20}\) pages = \(2^{22}\) = 4M

- 32bit machine, page size 4k, each entry 4 bytes, two level page table (two apges: 0x00000000, and 0xFFFFF000)

Page table size = \((2^{10} \text{ level-0 entries}) \times 4\text{bytes} + (2^{10} \text{ level-1 entries} \times 4 \text{bytes}) \times 2 = 12\text{Kbytes}\)
Size of Page/Frame: How Big?

- Determined by **number of bits in offset** (12Bit → 4KB)
- Smaller pages have advantages
  - Less internal fragmentation
  - Better fit for various data structures, code sections
- Larger pages are better because
  - Less overhead to keep track of them
  - More efficient to transfer larger pages to and from disk
- **One principle: page table → fit into one frame**
  - 32bits machine, 10 bits for each level

How can we make the address translation faster?
Integrating TLB and Cache

“Translation Lookaside Buffer” (TLB)
Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- If one out of 1,000 access causes a page fault, then
  EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!
- If we want just 10% performance degradation, then p should be
  220 > (1 – p) x 200 + p (8 milliseconds)
  p < 0.0000025, i.e., 1 page fault out of 400,000 accesses
Locality and Thrashing

- To prevent thrashing we should give **enough** frames to each process
- But how much is “**enough**”

**Locality model**

- Process migrates from one locality to another
- Localities may overlap

When \( \Sigma \) size of locality > total memory size

thrashing occurs...

Increase locality in your programs!
Working-Set Model

\[ \Delta \equiv \text{working-set window} \equiv \text{a fixed number of page references}, \text{ example: 10,000 instruction} \]

\[ WSS_i (\text{working set of Process } P_i) = \text{total number of pages referenced in the most recent } \Delta \]

- if \( \Delta \) too small, will not encompass entire locality
- if \( \Delta \) too large, will encompass localities
- if \( \Delta = \infty \) \( \Rightarrow \) will encompass entire program

\[ D = \Sigma WSS_i \equiv \text{total demand frames} \]

if \( D > (\text{available frames}) m \Rightarrow \text{Thrashing} \)

Thus, if \( D > m \), then suspend one of the processes
Basic Page Replacement

- Find the desired page on disk
- If there is a free frame, use it
- If there is no free frame, use a page replacement algorithm
  1. Select a **victim** frame, swap it out (use dirty bit to swap out only modified frames)
  2. Bring the desired page into the (newly) free frame;
  3. Update the page and frame tables

Restart the process
Page Replacement Algorithms

- How to select the victim frame?
  - You can select any frame, the page replacement will work; but the performance???
  - Gives the lowest page-fault rate

- Evaluate an algorithm by running it on a particular string of memory references (reference string) and compute the number of page faults on that string.
  In all our examples, we will have 3 frames and the following reference string:

```
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```
First-In-First-Out (FIFO) Algorithm

- Maintain an FIFO buffer
  - + The page used before may not be needed
  - - An array used early, might be used again and again
- Easy to implement
- Belady’s Anomaly: more frames ⇒ more page faults

reference string

```
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```

page frames

```
7 7 7 2 2 2 4 4 4 0 0 0 7 7 7
0 0 0 3 3 3 2 2 2 1 1 1 1 0 0
1 1 1 0 0 0 3 3 3 2 2 2
```
Least Recently Used (LRU) Algorithm

- Use recent past as an approximation of the future
- Select the page that is not used for a long time…
  - OPT if you look at from backward
  - NO Belady’s Anomaly: so more frames $\Rightarrow$ less page faults

Reference string:

```
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```

Page frames:

```
7 7 7 2 2 4 4 4 0 1 1 1
0 0 0 0 0 3 3 3 3 0 0 0
1 1 3 3 2 2 2 2 2 2 2 7
```
Page Allocation

order  free_area_t
zone->free_area

0
1
2
3
4
5
6
7
8
9
MAX_ORDER

Requesting Process

$2 \times 2^2$ block

$2 \times 2^3$ block

$1 \times 2^4$ block
Buddy Allocation

Example: Need to allocate 65 contiguous page frames.

- Look in list of free 128-page-frame blocks.
- If free block exists, allocate it, else look in next highest order list (here, 256-page-frame blocks).
- If first free block is in 256-page-frame list, allocate a 128-page-frame block and put remaining 128-page-frame block in lower order list.
- If first free block is in 512-page-frame list, allocate a 128-page-frame block and split remaining 384 page frames into 2 blocks of 256 and 128 page frames. These blocks are allocated to the corresponding free lists.

Question: What is the worst-case internal fragmentation?
Buddy De-Allocation

- When blocks of page frames are released the kernel tries to merge pairs of “buddy” blocks of size $b$ into blocks of size $2b$.

- Two blocks are buddies if:
  - They have equal size $b$.
  - They are located at contiguous physical addresses.
  - The address of the first page frame in the first block is aligned on a multiple of $2b \times 2^{12}$.

- The process repeats by attempting to merge buddies of size $2b$, $4b$, $8b$ etc…
Slab Allocator

- Performs the following functions
  - Allocate memory
  - Initialize objects/structures
  - Use objects/structures
  - Deconstruct objects/structures
  - Free memory

- /proc/slabinfo – gives full information about memory usage on the slab level. (see also /usr/bin/slabtop)
Introduction to DS

- Different Distributed Systems
  - Distributed computing systems
  - Distributed information systems
  - Distributed pervasive systems

- OS in distributed systems
  - Distributed OS vs. Network OS vs. Middleware

- Design objectives of distributed systems
  - Transparency, openness and scalability

- Architecture of distributed systems
  - Software vs. system architectures
Different Types of Distributed Systems

- Distributed **Computing** Systems: HP computing task
  - Cluster computing: similar components
  - Grid computing / Cloud computing: different components

- Distributed **Information** Systems
  - Web servers
  - Distributed database applications

- Distributed **Pervasive** Systems: instable
  - Smart home systems
  - Electronic health systems: monitor
  - Sensor networks: surveillance systems
Cluster Computing Systems

- High-performance computing
  - a group of high-end systems connected through a LAN
  - Homogeneous: same OS, near-identical hardware
  - Single managing node

![Diagram of Cluster Computing Systems](Image)
Grid Computing Systems

- Lots of nodes from everywhere share resources and collaborate
  - Heterogeneous
  - Dispersed across several organizations
  - Can easily span a wide-area network

- To allow for collaborations, grids generally use virtual organizations.
  - Same organization: a grouping of users (or better: their IDs) have the same access rights
  - The key questions are
    - Authorize users from different administrative domains
    - Provide authorized users with the access to specific resources
Organizations have legacy networked applications, but it is hard to make them interoperate.

Middleware can help.

Integration can take place at several levels:

- **Client-servers** wrap a number of requests into one and have it executed as a **Distributed Transaction** (all or none of requests would be executed).

- Applications can be detached from their databases and may need to directly communicate with each other: **Enterprise Application Integration (EAI)**
DPS: Distributed Pervasive Systems

- DCS and DIS: stable distributed systems (fixed nodes good connections)
- Unstable with mobile and embedded devices
- Distributed Pervasive Systems:
  - Computing anywhere and anytime
  - Contextual change: environment changes should be immediately react.
  - Ad hoc composition: Each node may be used in a very different ways by different users. Requires ease-of-configuration.
  - Sharing is the default: Nodes come and go, providing sharable services and information. Calls again for simplicity.

Expose distribution instead of hiding it!
Distributed Operating Systems

- **Full transparency**: users feel a big system and are not aware of multiple different machines
- Access to *remote* services similar to local resources
Network Operating Systems (NOS)

- NO single view of the distributed system
- Users are aware of the multiplicity of the machines
- Applications use **NOS services** to access resources
Middleware-Based Systems

Middleware: a higher level of abstraction on top of network operating systems for efficient transparency
Middleware-Based Systems (cont.)

- Each local system is a part of underlying NOS

- Target of *middleware*:
  - Resolve the integration problems of various networked applications

- Examples: Remote Procedure Calls (RPC), Remote Method Invocations (RMI), Distributed File Systems, Distributed Object Systems
## Distribution Transparency

<table>
<thead>
<tr>
<th>Access</th>
<th>Hides differences in data representation and invocation mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Hides where a resource resides</td>
</tr>
<tr>
<td>Migration</td>
<td>Hides that a resource may move</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hides that a resource may be moved while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hides that a resource is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hides that other users may access the same resource</td>
</tr>
<tr>
<td>Failure</td>
<td>Hides failure and possible recovery of resources</td>
</tr>
</tbody>
</table>
Scalability in Distributed Systems

- Three aspects of scalability
  - **size**: number of users and/or processes
  - **geographical**: Maximum distance between nodes
  - **administrative**: Number of administrative domains

- Most systems account only, to a certain extent, for size scalability: powerful servers (supercomputer)

- Challenge nowadays: geographical and administrative scalability
Techniques for Scalability

- Hiding communication latency: (Geographical)
  - Use **asynchronous** communication:
    - +: separate handler for incoming response and do something while waiting.
    - -: what if there is nothing else to do

- Distribution: splitting it to small parts
  - Domain naming systems (DNS)
  - Decentralized data, information systems (WWW)
  - Decentralized algorithm (Distance Vector)

- Replicate:
  - Increase availability
  - Load balance
System Architectures for DS

Consider how and where to place software components and realize their interactions.

About physical realization, placement of software components & interactions:
- Centralized: client-server
- Decentralized: P2P (Structured vs. unstructured)
- Hybrid: Combination of centralized and P2P
Decentralized Architectures: P2P Systems

Key question:
- How to organize processes in an overlay network, where links are usually TCP channels

Three approaches to organize nodes into overlay networks through which data is routed:
- **Structured P2P**: nodes are organized following a specific distributed data structure and deterministic algorithms
- **Unstructured P2P**: randomly selected neighbors
- **Hybrid P2P**: some nodes are appointed special functions in a well-organized fashion
Network Communication

- Layered network models
  - OSI 7-layer model (Open System Interconnection)
- Ethernet: local area network
- Inter-network Protocols (IP)
  - Addressing and routing etc.
- TCP/UDP protocols
  - Communication ports and sockets
- Multicast: more than one recipients
# Summary for OSI Protocols

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.</td>
<td>HTTP, FTP, SMTP, CORBA IIOP</td>
</tr>
<tr>
<td>Presentation</td>
<td>Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.</td>
<td>Secure Sockets (SSL), CORBA Data Rep.</td>
</tr>
<tr>
<td>Session</td>
<td>At this level reliability and adaptation are performed, such as detection of failures and automatic recovery.</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>Transport</td>
<td>This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports attached to processes,</td>
<td>IP, ATM virtual circuits</td>
</tr>
<tr>
<td>Network</td>
<td>Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through routers. In a single LAN no routing is required.</td>
<td>Ethernet MAC, ATM cell transfer, PPP</td>
</tr>
<tr>
<td>Data link</td>
<td>Responsible for transmission of packets between nodes that are directly connected by a physical link. In a WAN transmission is between pairs of routers or between routers and hosts. In a LAN it is between any pair of hosts.</td>
<td>Ethernet base-band signalling, ISDN</td>
</tr>
<tr>
<td>Physical</td>
<td>The circuits and hardware that drive the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre optic circuits) or other electromagnetic signals (on radio and microwave circuits).</td>
<td></td>
</tr>
</tbody>
</table>
Ethernet: Local Area Network (LAN)

- **Shared medium**: Carrier Sensing Multi-Access.
  - CSMA/CD: collision detection
- Every Ethernet interface has a **unique** 48 bit address (a.k.a. *hardware address*).
  - Example: `C0:B3:44:17:21:17`
- Addresses are assigned to vendors by a central authority (IEEE to manufacturers)
Approaches for Packet Delivery

**Datagram (vs. mailed letters) - UDP**
- each packet contains full network address of source-to-destination;
- no setup of paths, one-at-a-time, hop-by-hop transmission of packets,
- unreliable, e.g., Internet IP datagram in network layer

**Virtual circuits (vs. phone call) - TCP**
- set up end-to-end path, packets contains virtual circuit #,
- more reliable,
- links can be shared
What is IP?

- **Internet Protocol**
  - packet delivery service (**host-to-host**).
  - translation on data-link protocols (Ethernet).

- IP provides **connectionless, unreliable** delivery of **IP datagram**.
  - Connectionless: each datagram is independent.
  - Unreliable (**best effort**): no guarantee for datagrams to be delivered correctly or at all.
TCP: Transmission Control Protocol

- TCP is *connection-oriented*.
  - 3-way handshake used for connection setup
  - Acknowledge each pack (pigback)

---

### Connection Setup

**3-way handshake**

- **(Active)**
  - Client
  - Server

- **Syn**

- **Syn + Ack**

- **Ack**

---

### Acknowledgement

**data packets**

- **(Active)**
  - Client

- **Data:N**

- **(Data :N+1) Ack**

---
Communication: applications

- **Fundamentals**
  - Client/Server communication protocols
    - Request vs. Request-reply vs. Request-reply-acknowledge
  - Invocation semantics
    - Exact once vs. at least once vs. at most once
  - Communication types
    - Transient vs. persistent
    - Synchronous vs. asynchronous

- **Models for application communications**
  - **RPC**: remote procedure call
  - Message-oriented communication
  - Stream-Oriented communication
  - Multicast communication
Should Servers Re-Do Operations?

- **Idempotent** operations: which can be performed repeatedly with the same effect.
  - Suppose $x$ is input message $\Rightarrow f(f(x)) = f(x)$
  - No state needs to maintain on the server

- Are the following operations idempotent?
  - HTTP GET ... 
  - UNIX file operations: read, write etc.
  - **yes**
  - **NO**
## Server Invocation Semantics in RR

<table>
<thead>
<tr>
<th>Retransmit request message</th>
<th>Duplicate filtering</th>
<th>Re-execute procedure or retransmit reply</th>
<th>Invocation semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Maybe</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Re-execute procedure</td>
<td>At-least-once</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Retransmit reply</td>
<td>At-most-once</td>
</tr>
</tbody>
</table>

**Fault tolerance measures**

- Retransmit request message
- Duplicate filtering
- Re-execute procedure or retransmit reply
Server Invocation Semantics (cont.)

- **Maybe**: if no reply, the client does not know if method was executed or not
- **At least once**: will guarantee that invocation be carried out at least once, but possibly more
- **At most once**: Will guarantee that RPC has been carried out at most once, but possibly none at all
  - Detect duplicated requests with sequence numbers
  - **No guarantees**: When a server crashes, the client gets no help and no promises about what happened

- **Local invocation**: exactly once - ideal case
Types of Communications

- **Asynchronous communication**
  - Sender *continues* immediately after it has submitted the request (unblocked, need a local buffer at the sender)

- **Synchronous communication**
  - Sender *blocks* until the sender receives an OK to continue; *where the OK may come?*
## Type of Communications (cont.)

<table>
<thead>
<tr>
<th>Persistent</th>
<th>Asynchronous</th>
<th>Synchronous at Submission, delivery, after service</th>
</tr>
</thead>
</table>
| **Message-oriented middleware (MOM)** | • Processes send each other messages (queued)  
• Sender does not need to wait for immediate reply  
• Middleware often ensures fault tolerance | |

<table>
<thead>
<tr>
<th>Transient</th>
<th>Client/Server, RPC, TCP</th>
<th></th>
</tr>
</thead>
</table>
| • Client and server have to be active at time of communication  
• Client issues request and blocks until it receives reply  
• Server essentially waits only for incoming requests, and subsequently processes them |

**Drawbacks of synchronous communication**
• Client cannot do any other work while waiting for reply  
• Failures have to be handled immediately: the client is waiting  
• The model may simply not be appropriate (mail, news)
RPC Steps

A remote procedure call occurs in the following steps:

1. The client procedure calls the client stub.
2. The client stub builds a message and calls the local operating system.
3. The client’s OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server’s OS sends the message to the client’s OS.
9. The client’s OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Sounds simple but there are several issues…
RPC Mechanism

Client computer

- Local return
- Local call
- Unmarshal results
- Marshal arguments
- Receive reply
- Send request

Server computer

- Execute procedure
- Unmarshal arguments
- Marshal results
- Select procedure
- Receive request
- Send reply

Communication module

server stub proc.

client stub proc.
pack value parameters into a message and send it to the server. Would it be that easy?
Problem 1: different data representations

A process on an Intel machine sends a message of an integer and four-character string (“5, JILL”) to another process on a Sun SPARC machine.

(a) original message on Intel (x86, Little Endian)
(b) receipt message on SPARC (Big Endian) “5000, JILL”
(c) simple reverse: message after converted “5, LLIJ”

The number is reversed by the string is not 63
Approaches for Exchanging Information

How people from different countries talk to each other?

- With listener’s language
  - English, Chinese, Indian, ..., //hundreds

- Common language
  - English

How can clients make servers on different machines understand them?

- Clients send information in servers’ data representation

- A common external data representation
  - Language defined representation
  - e.g., CORBA CDR

- External self-descriptive data representation
  - e.g., XML, Web Services
Problem 2: Un/Marshaling

- More than just wrapping parameters into a message
- Client and server machines may have **different data representations** (think of byte ordering)
- Client and server have to agree on a standard representation (e.g., external data representation (XDR))
  - How are basic data values represented (integers, floats, characters)
  - How are complex data values represented (arrays, unions)
- Client and server need to properly interpret messages, and transform them into machine-dependent representations.
Other Problems in Parameter Passing

- **Passing reference parameters**
  - Pointers, and in general, reference parameters are passed with considerable difficulty

- **Solutions**
  - Forbid reference parameters
  - Copy the entire data structure (e.g. an entire array may be sent if the size is known). In the case of the server input parameter, it does not need to be copied back.
  - How to handle complex data structures (e.g. general graphs)?
Parameter Passing Semantics in RPC

RPC parameter passing:
- RPC assumes copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values
- RPC assumes all data that is to be operated on is passed by parameters. Excludes passing references to data.

Conclusion: full access transparency cannot be realized.

Observation: If we introduce a remote reference, access transparency can be enhanced:
- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs
Message-Oriented Middleware

- **Asynchronous persistent** communication through support of middleware-level queues.
  - Queues correspond to buffers at communication servers
  - Communicated processes may **NOT** be active at the same time

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue</td>
</tr>
</tbody>
</table>
Stream-Oriented Communication

- Applications
  - Video-conference
  - Online multimedia player/viewer (YouTube, CNN Live etc.)

- Support for continuous media
- Streams in distributed systems
- Stream management
Transmission of Continuous Media

- Different timing guarantees: 3 types of transmission

- **Asynchronous:** no restrictions with respect to *when* data is to be delivered

- **Synchronous:** define a maximum end-to-end delay for individual data packets

- **Isochronous:** define a maximum and minimum end-to-end delay (*jitter* is bounded)
Stream

Definition: A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission

Common stream characteristics
- Streams are unidirectional
- A single source, and one or more sinks
- Often, either the sink and/or source is a wrapper around hardware (e.g., CD device, TV monitor, dedicated storage)

Two types of streams:
- Simple: single flow of data, e.g., audio or video
- Complex: multiple data flows, e.g., stereo audio or combination audio/video
Enforce QoS: Example

- **Interleaving**: multiple samples in a single packet: the effects of packet loss $\rightarrow$ spread the samples

![Diagram](attachment:image.png)
Enforce QoS: Example

- Network-level tools: prioritize packets (differentiated services)
- Use **buffers** to reduce jitter
Remote Objects and RMI

- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects;
Remote Object Model

- **Remote objects** -- can receive remote invocations; having state information.
- **Remote object reference** – identify remote objects in distributed environments
- **Remote interface** – specifies methods to be invoked remotely
- Process contains objects (local/remote)
  - Local objects: accept only local invocations
  - Remote object: accept both local/remote invocations
  - Remote invocation – different processes (same or different hosts)
- Exceptions – application level
Remote Object Reference (ROR)

- **Uniquely** identify remote objects in distributed systems
- Needed to invoke remote method of a remote object
- Remote object references may be passed as input arguments or returned as output arguments.

Compared with ordinary object reference, what additional information is needed for remote object reference?

<table>
<thead>
<tr>
<th>32 bits</th>
<th>32 bits</th>
<th>32 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet address</td>
<td>port number</td>
<td>time</td>
<td>object number</td>
</tr>
</tbody>
</table>
RPC Review

Middleware generates stubs on both sides

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
RMI Overview

How do clients know where the remote objects are?

Binding...

- RMI register: the string name of the object, the remote object itself
- The registry returns to the caller a reference (called stub) to the remote object.
- Invoke methods on the object (through the stub).
RPC vs. RMI

- **Similarity:**
  - Marshaling and parameter passing

- **Difference:**
  - **RPC:** C based, structure based semantics. RMI: java and object-oriented
  - **RPC:** call remote functions, passed everything. RMI: remote/system-wide object reference and invoke methods. We can also pass and return objects that can be distributed among many JVM instances, much more powerful.
  - **RMI** can support dynamic invocations.

```
fobject.append(int)               Invoke(fobject, id(append), int)
```
Middleware for Remote Objects

- Layer between application and communication/remote reference modules
- Automatically create proxy, skeleton and dispatcher from remote interface definition
- Client side: one proxy for each remote object
  - Implement the methods in remote objects
- Server side: one dispatcher and one skeleton
  - Dispatcher accepts message and select appropriate method in the skeleton: methodID
  - Skeleton: Marshall / unmarshall messages and invokes corresponding method in the remote object
Proxy and Skeleton

**Proxy** - makes RMI transparent to client. Class implements remote interface. Marshals requests and unmarshals results. Forwards request.

**Skeleton** - implements methods in remote interface. Unmarshals requests and marshals results. Invokes method in remote object.
Steps in RMI

1. Naming Service
2. Object B
3. Remote reference module
4. Communication module
5. ROR
6. Request
7. Remote reference module
8. Communication module
9. Proxy for B
10. Request
11. Class for B
12. Skeleton & dispatcher
13. Remote object B
14. Remote reference module
15. Communication module
16. Reply
17. Remote reference module
18. Communication module
Case Study: Java RMI

- Define a remote interface
  HelloInterface.java

- Server side:
  - Implement the interface
    Hello.java
  - Develop the server
    HelloServer.java

- Client side:
  - Develop a client
    HelloClient.java

- Run the RMI registry, the server, and the client

```java
//HelloInterface.java
import java.rmi.*;

public interface HelloInterface extends Remote {

    public String add(String s) throws RemoteException;

    public String say() throws RemoteException;
}
```
Naming

- **Overview: name and name services**
  - Naming space and implementation

- **Flat name** and simple resolutions

- **Structure name**
  - Name space and resolution
  - Case study: DNS

- **Attributed-based naming**
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations
What a Name is in DS?

- A name is a string of bits or characters that is used to refer to an entity (an entity could be anything such as host, printer, file, process, mailbox, user etc.)

- To operate on an entity, we need to access it, for which we need an access point.

- Access point is a special kind of entity and its name is called an address (address of the entity, e.g., IP, port #, phone #)
  - An entity may have more than one access point/address
  - An entity may change its access points/addresses
  - So using an address as a reference is inflexible and human unfriendly
  - A better approach is to use a name that is location independent, much easier, and flexible to use: web service for different web servers
Identifier

- A special name to **uniquely** identify an entity (SSN, MAC)
- A true identifier has the following three properties:
  - **P1**: Each identifier refers to at most one entity
  - **P2**: Each entity is referred to by at most one identifier
  - **P3**: An identifier always refers to the same entity (no reuse)

Addresses and identifiers are important and used for different purposes, but they are often represented in machine readable format (MAC, memory address)
Naming Systems and Their Goals

Naming Systems

- Flat names
- Structured names
- Attributed-based names

Goals

- Scalable to arbitrary size
- Have a long lifetime
- Be highly available
- Have fault isolation
- Tolerate mistrust
Flat Naming

- **Flat name**: random bits of string, no structure
  - E.g., SSN, MAC address

- **Resolution problem**: Given a flat (unstructured) name, how can we find/locate its associated access point and its address?

- **Solutions**:
  - Simple solutions (broadcasting)
  - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service
Home-Based Approaches

How to deal with scalability problem when locating mobile entities?

- Let a **home** keep track of where the entity is!

- How will the clients continue to communicate?
  - Home agent gives the new location to the client so it can directly communicate
    - efficient but not transparent
  - Home agent forwards the messages to new location
    - Transparent but may not be efficient
Problems with home-based approaches

- The home address has to be supported as long as the entity lives.
- The home address is fixed, which means an unnecessary burden when the entity permanently moves to another location
  
  How can we solve the “permanent move” problem?
  
  Register the home at a naming service and let the client to lookup at first

- Poor geographical scalability (the entity may be next to the client)
Distributed Hash Tables (DHT)

Recall Chord from Chapter 2, which organizes many nodes into a logical ring
- Each node is assigned a random $m$-bit identifier.
- Every entity is assigned a unique $m$-bit key.
- Entity with key $k$ falls under jurisdiction of node with smallest $id >= k$ (called its successor)

Linearly resolve a key $k$ to the address of $\text{succ}(k)$
- Each node $p$ keeps two neighbors: $\text{succ}(p+1)$ and $\text{pred}(p)$
- If $k > p$ then forward to $\text{succ}(p+1)$
- if $k <= \text{pred}(p)$ then forward $k$ to $\text{pred}(p)$
- If $\text{pred}(p) < k <= p$ then return $p$’s address ($p$ holds the entity)
DHT: Finger Table for Efficiency

- Each node $p$ maintains a **finger table**
  - at most $m$ entries (short cuts) with exponentially increasing size
  $$FTp[i] = \text{succ}(p + 2^{i-1})$$
  - $FTp[i]$ points to the first node succeeding $p$ by at least $2^{i-1}$
  - To look up a key $k$, node $p$ forwards the request to node with index $j$ satisfying
    $$q = FTp[j] \leq k < FTp[j + 1]$$
    (e.g., node 0 sends req $\rightarrow 4 \rightarrow 6$)
  - If $p < k < FTp[1]$, the request is also forwarded to $FTp[1]$
- Need at most $O(\log N)$ steps, where $N$ is the number of nodes in the systems
DHT: Example

Resolve $k = 12$ from node 28

Resolve $k = 26$ from node 1
Structure Name

- **A name graph**
  - **Leaf node** represents a (named) entity.
  - **Directory node** is an entity that refers to other nodes: contains a (directory) table of *(edge label, node identifier)* pairs
Name Space Implementation (cont.)

- **Global level**: high-level directory nodes. Stable

- **Administrational level**: directory nodes that are managed within a single organization. Relatively stable.

- **Managerial level**: within a single administration. Nodes may change regularly.

### Availability and performance

<table>
<thead>
<tr>
<th>Item</th>
<th>Global</th>
<th>Administrative</th>
<th>Managerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical scale of network</td>
<td>Worldwide</td>
<td>Organization</td>
<td>Department</td>
</tr>
<tr>
<td>Total number of nodes</td>
<td>Few</td>
<td>Many</td>
<td>Vast numbers</td>
</tr>
<tr>
<td>Responsiveness to lookups</td>
<td>Seconds</td>
<td>Milliseconds</td>
<td>Immediate</td>
</tr>
<tr>
<td>Update propagation</td>
<td>Lazy</td>
<td>Immediate</td>
<td>Immediate</td>
</tr>
<tr>
<td>Number of replicas</td>
<td>Many</td>
<td>None or few</td>
<td>None</td>
</tr>
<tr>
<td>Is client-side caching applied?</td>
<td>Yes</td>
<td>Yes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>

Caching and replicas are introduced for performance, but with consistence problems
Recursive vs. Iterative Name Resolution

Recursive name resolution

Iterative name resolution

Long-distance communication

Client

Name server nl node

Name server vu node

Name server cs node
DNS issues

- Name tables change infrequently, but when they do, caching can result in the delivery of stale data.
  - Clients are responsible for detecting this and recovering
- Its design makes changes to the structure of the name space difficult. For example:
  - merging previously separate domain trees under a new root
  - moving sub-trees to a different part of the structure

Overall: it still runs well after 30 years, no need to be replaced!
Directory Services

- Entities have a set of attributes (e.g., email: send, recv, subject, ...)
- In most cases, attributes are determined manually
- Setting values consistently is a crucial problem ...
- Often organized in a hierarchy
  - Examples of directory services: X.500, Microsoft’s Active Directory Services,
- Then, **look up** entities by means of their **attributes**
- **Problem:** Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values
  - In the simplest form, **inspect all entities.**