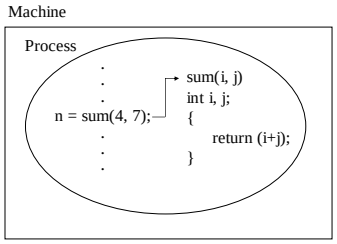
CS 194: Distributed Systems
Review

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

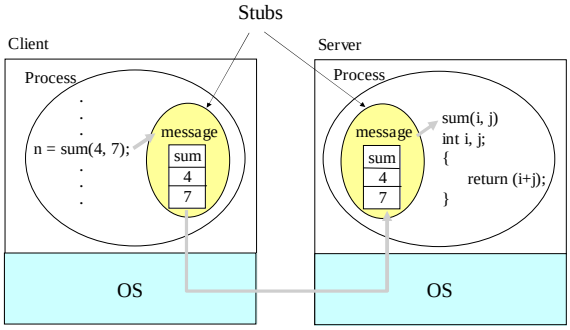
Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security

Example: Local Procedure Call

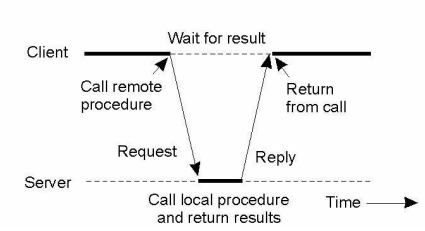


Example: Remote Procedure Call



Client and Server Stubs

- Principle of RPC between a client and server program



Parameter Passing

- Server and client may encode parameters differently
  - E.g., big endian vs. little endian
- How to send parameters “call-by-reference”?
  - Basically do “call-by-copy/restore”
  - Woks when there is an array of fixed size
  - How about arbitrary data structures?

RPC Semantics: Discussion

- The original goal: provide the same semantics as a local call
- Impossible to achieve in a distributed system
  - Dealing with remote failures fundamentally affects transparency
- Ideal interface: balance the easy of use with making visible the errors to users

7

Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security

8

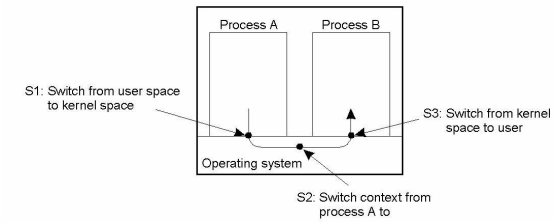
Process vs. Thread

- Process: unit of allocation
  - Resources, privileges, etc
- Thread: unit of execution
  - PC, SP, registers
- Each process has one or more threads
- Each thread belong to one process

9

Process vs. Thread

- Processes
  - Inter-process communication is expensive: need to context switch
  - Secure: one process cannot corrupt another process



10

Process vs. Thread

- Threads
  - Inter-thread communication cheap: can use process memory and may not need to context switch
  - Not secure: a thread can write the memory used by another thread

11

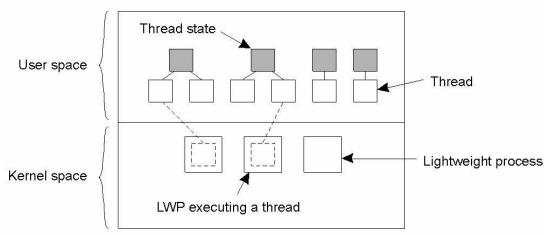
User Level vs. Kernel Level Threads

- User level: use user-level thread package; totally transparent to OS
  - Light-weight
  - If a thread blocks, all threads in the process block
- Kernel level: threads are scheduled by OS
  - A thread blocking won't affect other threads in the same process
  - Can take advantage of multi-processors
  - Still requires context switch, but cheaper than process context switching

12

Thread Implementation

- Combining kernel-level lightweight processes and user-level threads
  - LWPs are transparent to applications
  - A thread package can be shared by multiple LWPs
  - A LWP looks constantly after runnable threads



User-level, Kernel-level and Combined

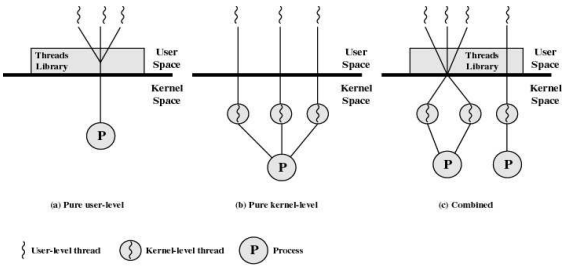
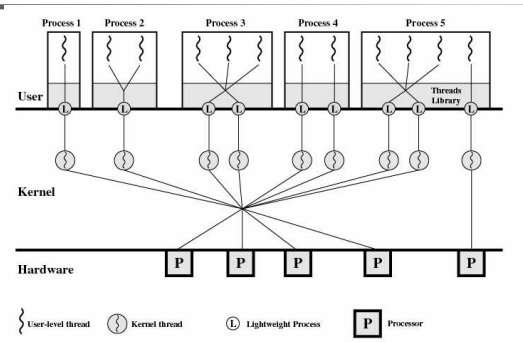


Figure 4.6 User-Level and Kernel-Level Threads

(Operating Systems, Stallings)

Example of Combined Threads



(Operating Systems, Stallings)

Figure 4.15 Solaris Multithreaded Architecture Example

Trade-offs

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls

Overview

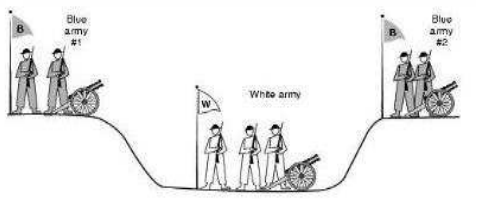
- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security

Agreement in Faulty Systems

- Many things can go wrong…
- Communication
  - Message can be lost
  - Time taken to deliver a message is unbounded
  - Adversary can intercept messages
- Processes
  - Can fail or collude with other processes to produce wrong results
- Agreement very hard, sometime impossible, to achieve!

Two-Army Problem

- "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."



- What is the solution?

19

Byzantine Agreement [Lamport et al. (1982)]

- Goal:
  - Each process learn the true values sent by correct processes
- Assumptions:
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded

20

Byzantine Agreement Result

- In a system with m faulty processes agreement can be achieved only if there are $2m+1$ functioning correctly
- Note: This result only guarantees that each process receives the true values sent by correct processors, but it does not identify the correct processors!

21

Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
  - Reliable multicast
  - Atomic multicast
- Distributed commit
- Security

22

Reliable Group Communication

- **Reliable multicast**: all non-faulty processes which do not join/leave during communication receive the message
- **Atomic multicast**: all messages are delivered in the same order to all processes

23

Scalable Reliable Multicast (SRM) [Floyd et al '95]

- Receivers use timers to send NACKS and retransmissions
  - Randomized: prevent implosion
  - Uses latency estimates
    - Short timer → cause duplicates when there is reordering
    - Long timer → causes excess delay
- Any node retransmits
  - Sender can use its bandwidth more efficiently
  - Overall group throughput is higher
- Duplicate NACK/retransmission suppression

24

Repair Request Timer Randomization

- Chosen from the uniform distribution on

$$2^i[C_1 d_{S,A}, (C_1 + C_2) d_{S,A}]$$

  - A – node that lost the packet
  - S – source
  - C_1, C_2 – constants
  - $d_{S,A}$ – latency between source (S) and A
  - i – iteration of repair request tries seen
- Algorithm
  - Detect loss → set timer
  - Receive request for same data → cancel timer, set new timer
  - Timer expires → send repair request

Timer Randomization

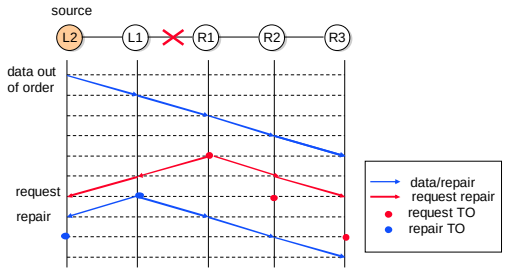
- Repair timer similar
  - Every node that receives repair request sets repair timer
  - Latency estimate is between node and node requesting repair
  - Use following formula:

$$2^i[D_1 d_{R,A}, (D_1 + D_2) d_{R,A}]$$

    - D_1, D_2 – constants
    - $d_{R,A}$ – latency between node requesting repair (R) and A
- Timer properties – minimize probability of duplicate packets
  - Reduce likelihood of implosion (duplicates still possible)
  - Reduce delay to repair

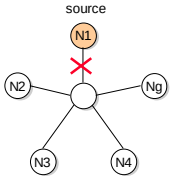
Chain Topology

- $C_1 = D_1 = 1$, $C_2 = D_2 = 0$ → timers = 1, 1
- All link distances are 1



Star Topology

- $C_1 = D_1 = 0$, → timers = [0, $2C_2$], [0, $2D_2$]
- Tradeoff between (1) number of requests and (2) time to receive the repair
- $C_2 <= 1$
  - E(# of requests) = g –1
- $C_2 > 1$
  - E(# of requests) = 1 + (g-2)/C_2
  - E(time until first timer expires) = $2C_2$/g
- $C_2 = \sqrt{g}$
  - E(# of requests) = $\sqrt{g}$
  - E(time until first timer expires) = $1/\sqrt{g}$



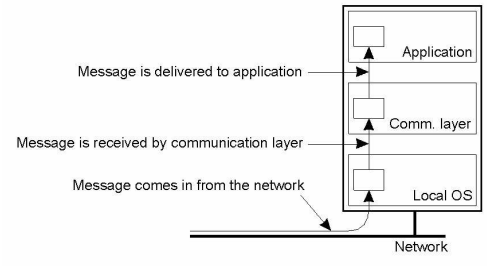
Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
  - Reliable multicast
  - Atomic multicast
- Distributed commit
- Security

Atomic Multicast

- All messages are delivered in the same order to “all” processes
- **Group view**: the set of processes known by the sender when it multicast the message
- **Virtual synchronous multicast**: a message m multicast to a group view G is delivered to all non-faulty processes in G
  - If sender fails “before” m reaches a non-faulty process, none of the processes deliver m

Virtual Synchrony System Model



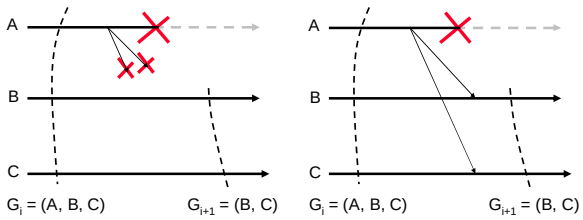
- The logical organization of a distributed system to distinguish between message receipt and message delivery

31

Virtual Synchronous Multicast

a) Message is **not** delivered

b) Message is delivered

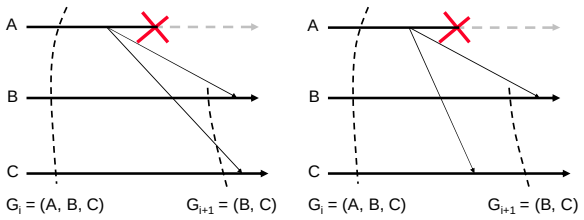


32

Virtual Synchronous Multicast

a) Message is **not** delivered

b) Message is **not** delivered



33

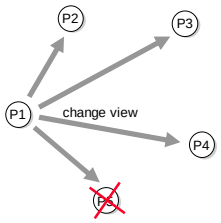
Virtual Synchrony Implementation: [Birman et al., 1991]

- Only **stable** messages are delivered
- **Stable message**: a message received by all processes in the message's group view
- Assumptions (can be ensured by using TCP):
  - Point-to-point communication is reliable
  - Point-to-point communication ensures FIFO-ordering

34

Virtual Synchrony Implementation: Example

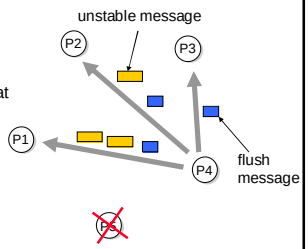
- G_i = {P1, P2, P3, P4, P5}
- P5 fails
- P1 detects that P5 has failed
- P1 send a "view change" message to every process in G_{i+1} = {P1, P2, P3, P4}



35

Virtual Synchrony Implementation: Example

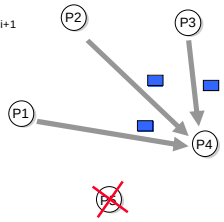
- Every process
  - Send each unstable message m from G_i to members in G_{i+1}
  - Marks *m* as being stable
  - Send a flush message to mark that all unstable messages have been sent



36

Virtual Synchrony Implementation: Example

- Every process
  - After receiving a flush message from any process in G_{i+1} installs G_{i+1}



37

Message Ordering and Atomicity

Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

38

Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security

39

Distributed Commit

- **Goal**: Either **all** members of a group decide to perform an operation, or **none** of them perform the operation

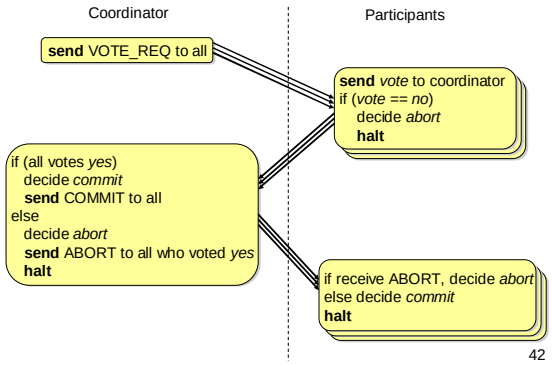
40

Assumptions

- Failures:
  - Crash failures that can be recovered
  - Communication failures detectable by timeouts
- Notes:
  - Commit requires a set of processes to agree…
  - …similar to the Byzantine general problem…
  - … but the solution much simpler because stronger assumptions

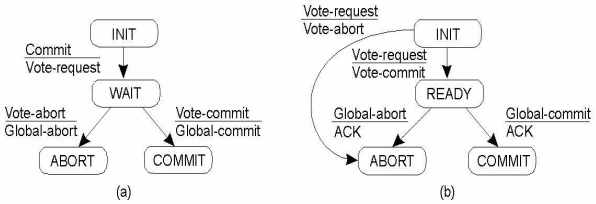
41

Two Phase Commit (2PC)



42

2PC State Machine



a) The finite state machine for the coordinator in 2PC
b) The finite state machine for a participant

Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security
  - Cryptographic Algorithms (Confidentiality and Integrity)
  - Authentication

Security Requirements

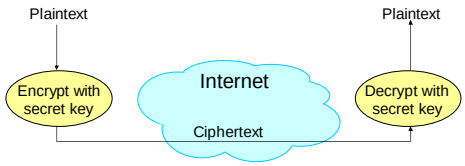
- **Authentication**: ensures that sender and receiver are who they are claiming to be
- **Data integrity**: ensure that data is not changed from source to destination
- **Confidentiality**: ensures that data is red only by authorized users
- **Non-repudiation**: ensures that the sender has strong evidence that the receiver has received the message, and the receiver has strong evidence of the sender identity (not discussed here)
  - The sender cannot deny that it has sent the message and the receiver cannot deny that it has received the message

Cryptographic Algorithms

- Security foundation: cryptographic algorithms
  - Secret key cryptography, Data Encryption Standard (DES)
  - Public key cryptography, RSA algorithm
  - Message digest, MD5

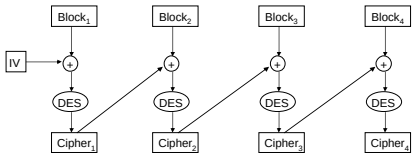
Symmetric Key

- Both the sender and the receiver use the same secret keys



Encrypting Larger Messages

- Initialization Vector (IV) is a random number generated by sender and sent together with the ciphertext



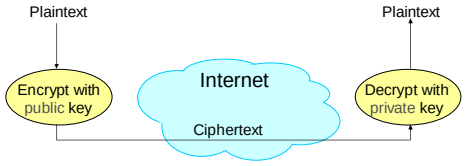
DES Properties

- Provide confidentiality
  - No mathematical proof, but practical evidence suggests that decrypting a message without knowing the key requires exhaustive search
  - To increase security use triple-DES, i.e., encrypt the message three times

49

Public-Key Cryptography: RSA (Rivest, Shamir, and Adleman)

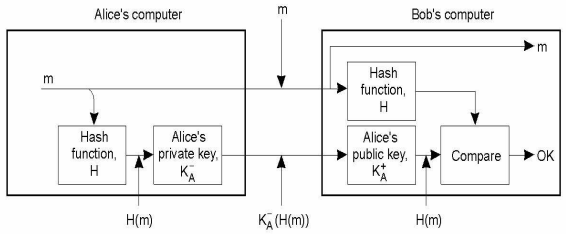
- Sender uses a public key
  - Advertised to everyone
- Receiver uses a private key



50

Digital Signature

- In practice someone cannot alter the message without modifying the digest
  - Digest operation very hard to invert
- Encrypt digest with sender's private key
- K_A^-, K_A^+: private and public keys of A



Digital Signature Properties

- Integrity: an attacker cannot change the message without knowing A's private key
- Confidentiality: if needed, encrypt message with B's public key

52

Overview

- Remote Procedure Call (RPC)
- Threads
- Agreement
- Group communication
- Distributed commit
- Security
  - Cryptographic Algorithms (Confidentiality and Integrity)
  - Authentication

53

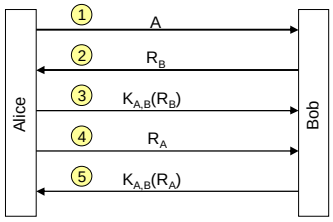
Authentication

- Goal: Make sure that the sender an receiver are the ones they claim to be
- Solutions based on secret key cryptography (e.g., DES)
  - Three-way handshaking
  - Trusted third party (key distribution center)
- One solution based on public key cryptography (e.g., RSA)
  - Public key authentication

54

Authentication

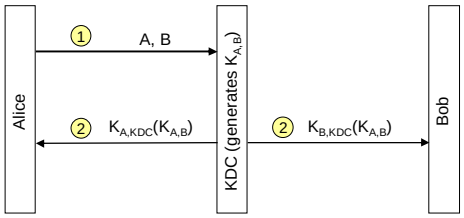
- Authentication based on a shared secret key
  - A, B: sender and receiver identities
  - $K_{A,B}$: shared secret key
  - R_A,R_B: random keys exchanged by A and B to verify identities



55

Authentication using KDC (Basic Protocol)

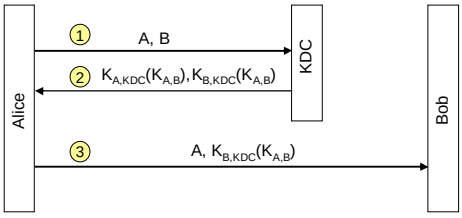
- KDC – Key Distribution Center
- Maintain only N keys in the system: one for each node



56

Authentication using KDC (Ticket Based)

- No need for KDC to contact Bob

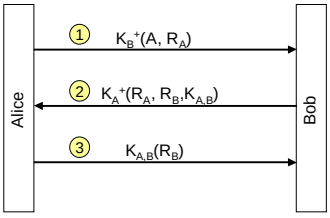


- Vulnerable to replay attacks if Chuck gets hold on $K_{B,KDC}^{old}$

57

Authentication Using Public-Key Cryptography

- K_A^+, K_B^+: public keys



58

Midterm Information

- Closed books; 8,5”x11” crib sheet (both sides)
- No calculators, PDAs, cell phones with cameras, etc
- Please use PENCIL and ERASER
- Expect also questions from project (e.g., XML-RMI)

59