Announcements

- Reading Chapter 17 (skip 17.6.1 and 17.6.4)
  - problems: 17.1, 17.3, 17.4
Remote Procedure Calls

- Provide a way to access remote services
- Look like “normal” procedure calls
- Issues:
  - binding functions to services
    - can use static binding (like kernel trap #’s)
    - can use a nameserver
  - data format
    - different machine may have different formats
    - translation is called *marshalling*
      - pick a common way to encode info (e.g. XDR)
      - always send in this common format
  - failures
    - what if a host dies while and RPC is active?
RPC Example

deficit("Jones", 123, 45.00)

deficit(char *name, int acct, float amt) {
    XDR_string(buffer, name);
    XDR_int(buffer, acct);
    XDR_float(buffer, amt);
    send(Server, DEBIT, buffer);
    receive(Server, ret, NULL);
    return(ret);
}

Stub

Server

Receive(caller, request, buffer)

    case DEBIT:
        XDR_string(buffer, name);
        XDR_int(buffer, acct);
        XDR_float(buffer, amt);
        ret = deficit(name, acct, amt);
        Send(caller, ret)

Messages

Int “45”

String “Jones”

Int “123”

Float “45.00”

Int “45”
RPC Generators

- Given a list of functions to make into RPC
- Generate the code for:
  - RPC stubs (for clients to call)
    - marshalling code for each parameter
    - utility routines to marshal structures/records
    - code to send messages and wait for responses
  - Server code
    - case statement for each RPC type
    - un-marshall parameters
    - call local routine
  - detecting errors
  - checking version numbers between client/server
Failures

- **Fail Stop**
  - system either produces the correct answer or no answer
  - hard to know “what” failed
    - local network card
    - network link
    - remote network card
    - remote system
    - remote software

- **Byzantine Failure**
  - systems can “lie” and produce wrong answers
    - a message shows up but some of the data is wrong
  - can use check sums to detect this failure mode
    - does not deal with malicious failure
  - considered a “hard” problem
Distributed Filesystems

● Provide the same semantics as a local filesystem
  – data is stored at various locations in the system
    • often stored in central fileservers
    • can be stored in serverless fileservers

● Naming
  – location transparency
    • filenames don’t imply information about location
  – location independence
    • can move the file without changing names
  – naming files
    • host:local-name
      – not transparent
    • global-name
      – transparent, requires something to coordinate names
DFS Performance Issues

- **“normal” filesystem issues**
  - disk parameters: seeks time, rotational latency
  - filesystem time: directory structure, fat/inodes

- **distributed system issues**
  - network:
    - latency (time for small requests)
    - bandwidth (time to move entire disk blocks)
  - coordination
    - time to access servers
      - namespace server
      - fileserver
Caching

● To improve performance, cache DFS information
  – goal: improve response times for overall DFS

● Local Cache
  – memory cache
    • data is stored in memory of local system
  – disk cache
    • data is stored on the disk of the local system

● Server Cache
  – memory
    • can put lots of memory here so most “popular” files are in memory
Caching (cont)

- Need to maintain consistency
  - Client initiated caching
    - client contacts the server “Is this still OK?”
  - Server initiated caching
    - server calls back to the client “dispose of those stale bits”

- What happens on write?
  - write-though caching
    - slow for writes
  - delayed writes
    - faster for writes
    - what happens when a failure occurs?