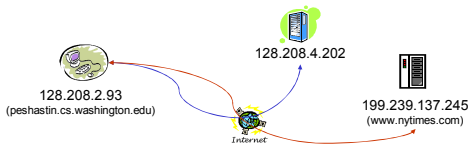
CSE 451: Operating Systems
Winter 2006

Module 21
Web Servers / Services

John Zahorjan
zahorjan@cs.washington.edu
Allen Center 534

Web Browsers / Servers: Intro



Fetching *http://www.nytimes.com/index.html*

DNS

- A distributed, hierarchically structured service
- 13 logical "root servers"; some are backed by multiple physical servers
- Names are resolved from right to left:
  www.nytimes.com
- Root servers know about the rightmost names (com, org, edu, net, etc.)
- They redirect the client to look at the server for "com", which redirects to the "nytimes" server, which sends back the address for "www"
- Caching is used extensively to limit traffic to root (and all) servers
  - Most root server traffic is for mis-typed names

Web Servers

- In the simplest case, the URL path is used to locate a file
  - The web server is configured with so that www.nytimes.edu corresponds to a particular directory in the machine's file system
  - The rest of the URL path is then traversed in the file system
  - The file arrived at this way is returned
- Things are actually more complex, of course.
  - The web server actually implements a generalization of symbolic links – rules that cause URL paths to be rewritten as they are transformed into file system paths
  - The web server also implements its own security system, independently of that implemented by the file system
    - Why?

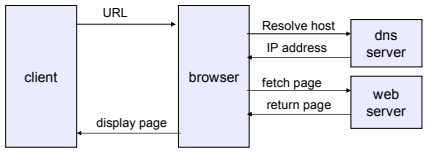
cgi's

- Quite soon after the web became popular, people found they needed "dynamic content" – pages whose content was created programmatically at the time of the request
  - E.g., amazon.com
- No problem – the Web server has rules that indicate which URLs actually name programs
  - When one of these URLs is received, the program is run, and its output is returned to the client (browser)
- Some (many!) web requests are therefore actually "calls" that execute remote code

Revisiting the Original Scenario: I

User → Browser	http://www.nytimes.com/index.html
Browser → 128.208.4.202 : 53	*"query www.nytimes.com"*
DNS Server → Browser	*199.239.137.245*
Browser → 199.239.137.245 : 80	GET /index.html HTTP/1.1 Host: www.nytimes.com
199.239.137.245 → Browser	HTTP/1.1 200 OK Server: Sun-ONE-Web-Server/6.1 Date: Fri, 02 Jun 2006 06:59:52 GMT Content-type: text/html Cache-control: no-cache Pragma: no-cache <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> <html> <head> <title>The New York Times - Breaking News, World News & Multimedia</title> <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
Browser	Formats and displays page

Revisiting the Original Scenario: II



We've seen this kind of control flow before… It's procedure call!

Unfortunately, we're going over a network, so…

The Old Fashioned (Current?) Method

1. Server process starts
2. Server process creates socket and listens for connection
3. Client process starts
4. Client process creates socket and binds it to server socket
5. Server (thread) accepts connection
6. Client formats a message and writes it to socket
7. Server reads from socket, parses messages, executes, formats a reply, and writes it to socket
8. Client parses reply

What about this seems wrong?

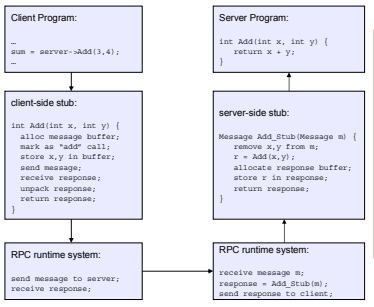
Remote Procedure Call (RPC)

- Use procedure calls as the model for distributed (remote) communication
  - have servers export a set of procedures that can be called by client programs
    - similar to library API, class definitions, etc.
  - clients do a local procedure call, as though they were directly linked with the server
    - under the covers, the procedure call is converted into a message exchange with the server
    - ***largely invisible to the programmer!***
- Interfaces are type checked; data is automatically packed into messages at sender and unpacked at receiver

RPC issues

- There are a bunch of hard issues:
  - how do we make the "remote" part of RPC invisible to the programmer?
    - and is that a good idea?
  - what are the semantics of parameter passing?
    - what if we try to pass by reference?
  - how do we bind (locate/connect-to) servers?
  - how do we handle heterogeneity?
    - OS, language, architecture, …
  - how do we make it go fast?

RPC example invocation



Topics:
- interface description
- stubs
- stub generation
- parameter marshalling
- binding
- runtime system
- error handling
- performance
- thread pools

RPC model

- A server defines the service interface using an interface definition language (IDL)
  - the IDL specifies the names, parameters, and types for all client-callable server procedures
    - example: ASN.1 in the OSI reference model
    - example: Sun's XDR (external data representation)
- A "stub compiler" reads the IDL declarations and produces two stub procedures for each server procedure
  - the server programmer implements the service's procedures and links them with the server-side stubs
  - the client programmer implements the client program and links it with the client-side stubs
  - the stubs manage all of the details of remote communication between client and server using the RPC runtime system

RPC stubs

- A client-side stub is a procedure that looks to the client as if it were a callable server procedure
  - it has the same API as the server's implementation of the procedure
  - a client-side stub is just called a "stub" in Java RMI
- A server-side stub looks like a caller to the server
  - it looks like a hunk of code that invokes the server procedure
  - a server-side stub is called a "skeleton" or "skel" in Java RMI
- The client program thinks it's invoking the server
  - but it's calling into the client-side stub
- The server program thinks it's called by the client
  - but it's really called by the server-side stub
- The stubs send messages to each other, via the runtime, to make the RPC happen transparently

RPC marshalling

- Marshalling is the packing of procedure parameters into a message packet
  - the RPC stubs call type-specific procedure to marshal or unmarshal the parameters of an RPC
    - the client stub marshals the parameters into a message
    - the server stub unmarshals the parameters and uses them to invoke the service's procedure
  - on return:
    - the server stub marshals the return value
    - the client stub unmarshals the return value, and returns them to the client program

RPC binding

- Binding is the process of connecting the client to the server
  - the server, when it starts up, exports its interface
    - identifies itself to a network name server
    - tells RPC runtime that it is alive and ready to accept calls
  - the client, before issuing any calls, imports the server
    - RPC runtime uses the name server to find the location of the server and establish a connection
- The import and export operations are explicit in the server and client programs
  - a slight breakdown in transparency
    - more to come…

RPC transparency

- One goal of RPC is to be as transparent as possible
  - make remote procedure calls look like local procedure calls
  - we've seen that binding breaks this transparency
- What else breaks transparency?
  - failures: remote nodes/networks can fail in more ways than with local procedure calls
    - network partition, server crash
    - need extra support to handle failures
    - server can fail independently from client
      - "partial failure": a big issue in distributed systems
      - if an RPC fails, was it invoked on the server?
  - performance: remote communication is inherently slower than local communication
    - if you're not aware you're doing a remote procedure call, your program might slow down an awful lot…

RPC and thread pools

- What happens if two client threads (or client programs) simultaneously invoke the same server procedure using RPC?
  - ideally, two separate threads will run on the server
  - so, the RPC run-time system on the server needs to spawn or dispatch threads into server-side stubs when messages arrive
    - is there a limit on the number of threads?
    - if so, does this change semantics?
    - if not, what if 1,000,000 clients simultaneously RPC into the same server?

Web Services

- RPC accessible remote code execution
  - Some specific RPC system is used to access them (e.g., Java RMI/Jini, SOAP, CORBA)
- Example: Google Web Services
  - Web search
    - search() [like the online search, but with more options]
    - cache() [returns page contents last time it was visited]
    - Spell() [did you mean…?]
  - Google earth
  - …

Web Services (cont.)

- Example: Amazon
  - E-commerce interfaces (item lookup, cart management, etc.)
  - Alexa (web crawling, structure, traffic, etc. measurements)
  - S3 (network storage: $.15/GB/month + $.20/GB transferred)
  - Historical pricing (pricing and sales information since 2002)
  - Mechanical Turk (remote AI, $.005/HIT (Human Intelligence Task))
    - read (photo);
      photoContainsHuman = callMechanicalTurk(photo);
      if (photoContainsHuman == TRUE){
        acceptPhoto;
      }
      else {
        rejectPhoto;
      }

    How do they do this?