## Where We Are

- so far: networking
- rest of semester: distributed computing
  - how to use networks to do interesting things
  - connection to operating systems, programming languages
- today
  - remote procedure call
  - naming



## Client-Server Model: Examples

- file server: stores files and accepts open/close/read/write requests
- print server
- DNA sequence server
- chess server
- Web server
- directory server



## Client-Server Programming

- interaction between client and service
  - client sends command and arguments
  - server computes
  - server sends back reply data
- abstractly, just like calling a procedure
- many chances to make programming errors
  - data format
  - command codes
  - reply codes

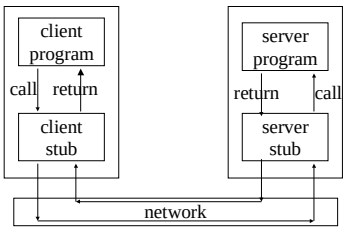


## Remote Procedure Call (RPC)

- idea
  - make it look just like procedure call
  - automate the writing of data-handling and formatting code
- advantages
  - fewer errors
  - faster to code and evolve interfaces
  - can connect to language mechanisms

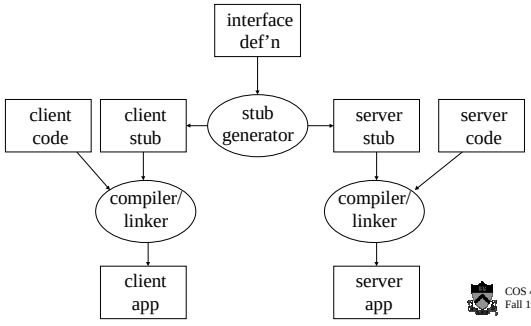


## RPC Structure






## Building an RPC App



## Interface Definition Example

```
PROCEDURE add(IN int x, IN int y, OUT int sum);

STRUCT foo {
        int   x;
        boolean b;
}

PROCEDURE p(INOUT foo f);
```

## Client Stub Example

```
void remote_add(Server s, int* x, int* y, int* sum) {
        s.sendInt(AddProcedureCode);
        s.sendInt(*x);
        s.sendInt(*y);
        s.flush();
        status = s.receiveInt();
        /* check for error status */
        *sum = s.receiveInt();
}

void remote_p(Server s, struct foo * f) {
        ...
```

## Server Stub Example

```
void serverLoop(Client c) {
        while(1) {
                int procedureCode = c.receiveInt();
                switch(procedureCode) {
                case AddProcedureCode:
                        int x = c.receiveInt();
                        int y = c.receiveInt();
                        int sum;
                        add(*x, *y, *sum);
                        c.sendInt(StatusOK);
                        c.sendInt(sum);
                        break;
                ...
```

## RPC and Threads

- local procedure call: thread jumps from caller to callee
- RPC: thread can't jump
- alternative: block calling thread, transfer control to thread on callee
  - one thread on callee: can deadlock
  - one thread for each client: too many threads
  - make threads on demand: can be slow

## RPC Binding

- How does the client find the server?
  - can hard-wire location
  - can use "name server"
  - hard-wire name server location
- client represents "handle" to a server as a Binding object
  - often, Bindings can be passed between clients
- a process can be both a client and a server

## Coping with Errors in RPC

- RPC isn't quite like local procedure call
  - communication errors
  - call-by-copy, not -value or -reference
- comm errors may leave client uncertain about whether the call really happened
  - various semantics possible: at-least-once, at-most-once, exactly-once
  - difference is visible, unless call is idempotent

## RPC and Objects

- RPC and object-oriented programming fit together very nicely.
  - encapsulation
  - use method calls, not direct data access
- several systems put an object face on RPC
  - hides many of the warts of RPC
- next lecture: look at one system in depth

## Naming

- What's in a name?
- naming, binding and distribution
- name consistency
- scaling
- capabilities
- examples

## Things that are Named

- people
- machines
- files
- programs
- services
- variables
- datatypes
- mailing lists

## What Names Do

- uniquely identify something
  - can be placeholder for not-yet-created thing
- provide information about location, function
- specify membership in a group
- specify a role being played by the named object
- convey knowledge of a secret

## Names and Locations

- location-dependent names
  - convey location information
  - implied commitment not to move the object
- local names
  - valid only in one place
- pure names
  - can be located anywhere
  - not globally unique

## Pure Names and Location

- pure names don't convey location; how do you find the object?
- short answer: name servers
- details
  - use hack to find a name server
  - name contains location of name server
  - hierarchical names

## Hierarchical Names

- example: /usa/edu/princeton/cs/www/courses/cs461
- advantages
  - globally unique
  - implement using hierarchy of name servers
- disadvantages
  - must agree on who is the root
  - heavy load on the root

## Relative Names

- everybody runs a name server
- can bind somebody else's server to a name in yours (done by hand)
  - you can bind /felten to point to my server
- long paths jump from server to server
  - /felten/mom/secretary is Felten's Mom's secretary
- combines features of local and global naming

## Naming and Security

- the power to name is the power to control
- names are "trusted" to refer to certain things
  - examples: "printer", "bank"
- name server(s) must be trusted
- capabilities combine naming and security

## Capabilities

- A capability is a secret name for an object; knowing the capability implies permission to operate on the object
  - can use different capabilities for different operations
- common implementation
  - digitally-signed statement from some authority
    - » "the bearer may do operations X,Y,Z on object O"

## Fun with Naming

- symbolic link: one name is an alias for another
- filter link: "view" of a directory that hides or modifies some names
- union directory: single directory that appears to contain contents of multiple other directories
- computed directories: not a directory at all, but the output of some program