6.033 Lecture 4: Client/server

Modularity
how to impose order on complex programs?
break into modules
[big Therac-25 blob => components, interactions]
goal: decouple, narrow set of interactions

Modularity (2)
what form does modularity take?
interactions are procedure calls
C -> P, P returns to C
procedure clarifies interface, hides implementation
P(a) { ... }
C() { ... y = P(x); ... }

Enforced?
is the interface enforced?
is the implementation hidden?

What actually happens when C calls P?

6.004
y they communicate via a shared stack, in memory
[stack: regs, args, RA, P's vars, maybe args...]
C: push regs, push args, push PC+1, jmp P, pop args, pop regs, ... R0
P: push vars, ..., pop vars, mov xx -> R0, pop PC

Call Contract
P returns
P returns to where C said
P restores stack pointer
P doesn't modify stack (or C's other memory)
P doesn't wedge machine (e.g. use up all heap mem)

Soft modularity
at a low level, none of call contract is enforced!
want: spec + contract
spec: we cannot hope to enforce (e.g. does sqrt() return the right answer?)
contract: is purely mechanical, we can try to enforce
goal: programmer need only worry about spec, not contract
== Enforced Modularity
there are many ways to enforce modularity
we'll look at one today

Client/server
note much of prob from sharing same machine and memory
so: put C and P on separate machines
interact only w/ msgs
[diagram: two boxes with a wire]
Examples: web client / server, AFS, X windows (about to read)

Code + time diagram
Client:
put args in msg (e.g. URL)
send msg
wait for reply
Server:
  wait for req msg
  get args
  compute...
  put result in reply msg (e.g. content of index.html) send reply
  goto start

C/S Benefits
1. protects control flow (e.g. return address)
2. protects private data
3. no fate sharing (crashes and wedges don't propagate)
4. forces a narrow spec (for better or worse, no global vars)
c/s enforces most of call contract
  bugs can still propagate via messages
  but that's a fairly restricted channel
  easier to examine than all of memory
spec still programmer's problem: ensure server returns the right answer

C/S helps with Multiple Parties
  it turns out c/s has benefits more than just enforced modularity
  [diagram: emacs, AFS, print]
  emacs on workstation, AFS, print server
  workstation uses multiple servers
  AFS has multiple clients
  print server might be both server and client of AFS
  e.g. client sends file name to print server

AFS server is a "Trusted Intermediary"
1. get at your data from multiple physical locations
2. share with other clients
3. control the sharing (private vs public data, ACLs)
   private, friends, public
   bugs in one client don't affect other clients
4. servers physically secure, reliable, easy to find
these c/s benefits are as important as enforcing modularity
  e.g. e-mail server, eBay

RPC
  c/s involves a lot of nasty error-prone msg formatting &c
  why not hide details behind real procedure interface?

Example:
  imagine this is your original single-program ebay implementation:
  ui(){
    print bid("123", 10.00);
  }
  bid(item, amt) {
    ...
    return winning;
  }
you want to split at bid() w/o re-writing this code

Client stub
  idea: procedure that *looks* like bid() but sends msg
  client stub:
bid(item, amt){
    put item and amt in msg;
    send msg;
    wait for reply;
    winning <- reply msg;
    return winning;
}

now original ui() is using a server!

Server stub
want to use original bid() code on server
server stub:
dispatch(){
    while(1){
        read msg
        item, amt <- msg;
        w = bid(item, amt)
        msg <- w;
        send msg;
    }
}

Marshal / unmarshal
need standard way to put data types into messages
    a message is a sequence of bytes
standard size for integers, flat layout for strings and arrays
typical request message format:
    proc# id 3 '1' '2' '3' 10.00???

Automatic stub generation
    tool to look at argument and return types: run it on bid()
generate marshal and unmarshal code
generate stub procedures
    saves programming
    ensures agreement on e.g. arg types

RPC very successful at simplifying c/s, but

RPC != PC
    despite syntactic similarity
    amazon web, warehouse back-end
    [time-line]
    user (browser) wants to place an order
    RPC 1: check inventory(isbn) -> count
        what if network loses/corrupts request?
        loses reply?
        warehouse crashes?
        stub can hide these failures by timeout + resend
        wrap loop around stub
        leads to duplicates -- OK, no side effects
    RPC 2: ship(isbn, addr) -> yes/no
        request lost?
        reply lost?
        warehouse crashes?
        can stub hide these failures by retrying?
        are duplicates OK?
How do RPC systems handle failure?

A couple of approaches, often called "failure semantics"
1. At least once
   - Client keeps re-sending until server responds
   - More than once
   - Error return AND success
   - Only OK for RPCs w/o side effects e.g. check inventory
2. At most once
   - Client keeps re-sending (may time out + error)
   - Server remembers requests and suppress duplicates
   - Error return AND success
   - Is it OK for ship()?
     - Not if we also want to charge credit card iff book shipped
     - If client gives up, or crashes, did the book ship?
3. Exactly once
   - This is often what we really want: just do it
   - Hard to do in a useful way, will see practical versions later

Most RPC systems do #1 or #2

RMI code slide

You feed first part to the rmic stub generator
   - It produces implementation of BidInterface for main() to call
   - And server dispatch stubs
   - The server stubs call your implementation of bid() (not shown)
   - Public interface ensures client/server agree on arguments (msg format)
   - Procedure call to bid() looks very ordinary!
   - But around it there are non-standard aspects
   - Naming.lookup() -- which server to talk to?
   - Server has registered itself with Naming system
   - Lookup() returns a proxy BidInterface object
   - Has bid() method, also remembers server identity
   - Try/catch
     - This is new, every remote call must have this, for timeouts/no reply

Summary
   - Enforced modularity via C/S
   - RPC automates details
   - RPC semantics are different
   - Next: enforced mod on same machine

RMI code slide (l4.ppt):

public interface BidInterface extends Remote {
   public String bid(String) throws RemoteException;
}

public static void main (String[] argv) {
   try {
      BidInterface srvr = (BidInterface)
         Naming.lookup("//xxx.ebay.com/Bid");
      winning = srvr.bid("123");
      System.out.println(winning);
   } catch (Exception e) {
      System.out.println ("BidClient exception: " + e);
   }
}