CS 194:
Distributed Systems
Communication Protocols, RPC

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

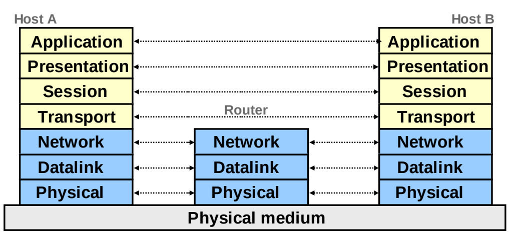
1

ISO OSI Reference Model for Layers

Application
Presentation
Session
Transport
Network
Datalink
Physical

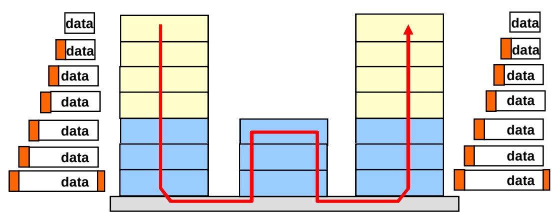
Mapping Layers onto Routers and Hosts

– Lower three layers are implemented everywhere
– Next four layers are implemented only at hosts



Encapsulation

- A layer can use only the service provided by the layer immediate below it
- Each layer may change and add a header to data packet
  – higher layer's header is treated as payload

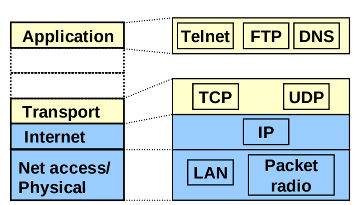


OSI Model Concepts

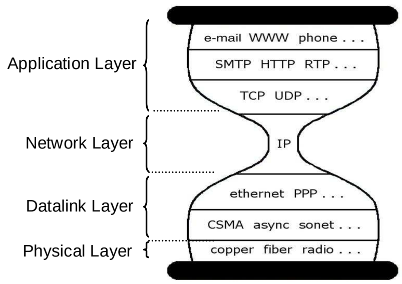
- Service – says what a layer does
- Interface – says how to access the service
- Protocol – says how is the service implemented
  – A set of rules and formats that govern the communication between two peers

Layering: Internet

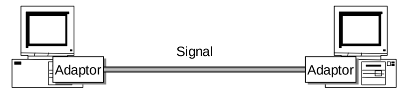
- Universal Internet layer:
- Internet has only IP at the Internet layer
- Many options for modules above IP
- Many options for modules below IP



Internet's Hourglass



Physical Layer



- **Service**: move information between two systems connected by a physical link
- **Interface**: specifies how to send a bit
- **Protocol**: coding scheme used to represent a bit, voltage levels, duration of a bit
- Examples: coaxial cable, optical fiber links; transmitters, receivers

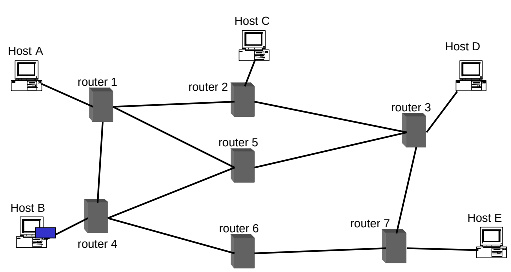
Datalink Layer

- **Service**:
  - Framing (attach frame separators)
  - Send data frames between peers
  - Medium access: arbitrate the access to common physical media
  - Error detection and correction
- **Interface**: send a data unit (packet) to a machine connected to the same physical media
- **Protocol**: layer addresses, implement Medium Access Control (MAC) (e.g., CSMA/CD)…

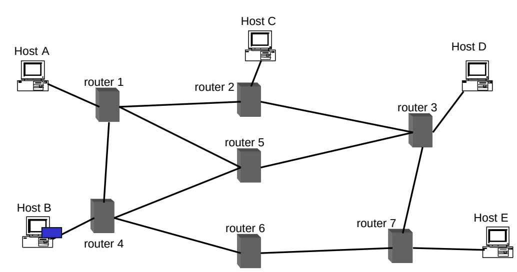
Network Layer

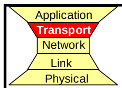
- **Service**:
  - Deliver a packet to specified network destination
  - Perform segmentation/reassembly
  - Others
    - Packet scheduling
    - Buffer management
- **Interface**: send a packet to a specified destination
- **Protocol**: define global unique addresses; construct routing tables

Datagram (Packet) Switching



Datagram (Packet) Switching



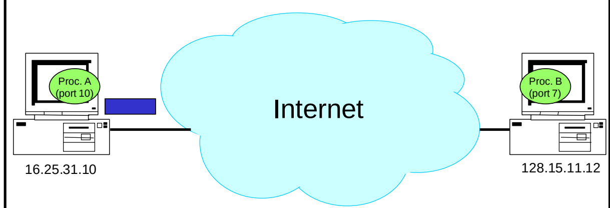


Transport Layer

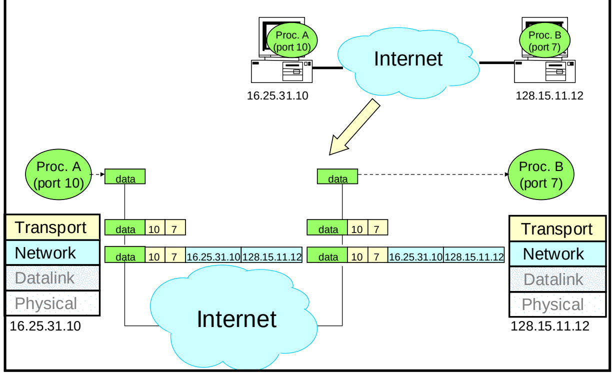
- **Services:**
  - Multiplex multiple transport connections to one network connection
  - Provide an error-free and flow-controlled end-to-end connection
  - Split one transport connection in multiple network connections
- **Interface:** send a packet to specify destination
- **Protocols:** implement reliability and flow control
- Examples: TCP and UDP

End-to-End View

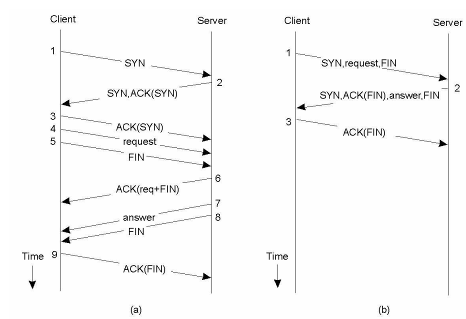
- Process A sends a packet to process B



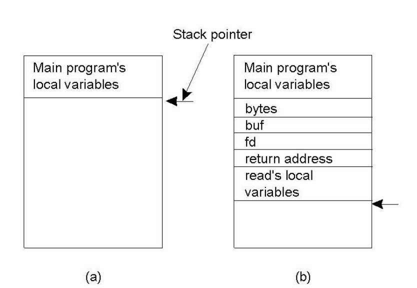
End-to-End Layering View



Client-Server TCP



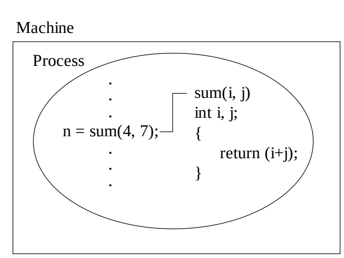
Conventional Procedure Call



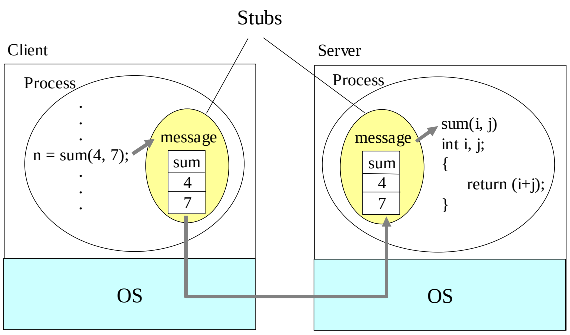
a) Parameter passing in a local procedure call: the stack before the call to read

b) The stack while the called procedure is active

Example: Local Procedure Call

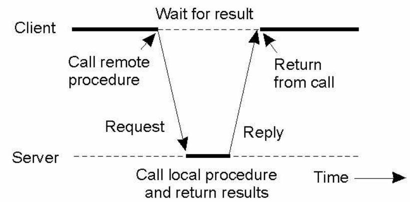


Example: Remote Procedure Call



Client and Server Stubs

- Principle of RPC between a client and server program.



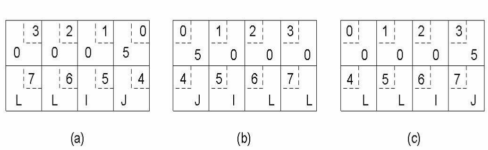
Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

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?

Different Encodings



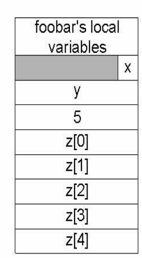
a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

```
foobar( char x; float y; int z[5] )
{
  ....
}
```

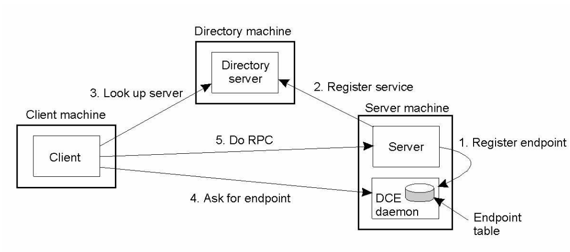
(a)



(b)

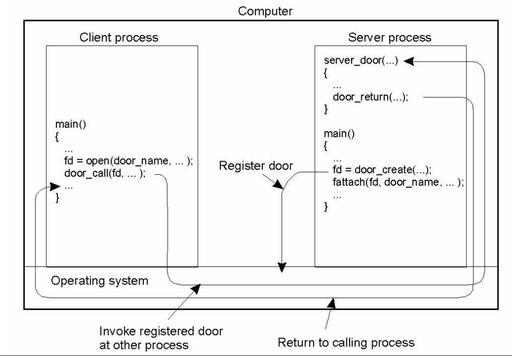
Binding a Client to a Server

- Client-to-server binding in DCE.



Doors

- The principle of using doors as IPC mechanism.



RPC Semantics in the Presence of Failures

- The client is unable to locate the server
- The request message from the client to server is lost
- The reply message from the client is lost
- The server crashes after sending a request
- The client crashes after sending a request

Client is Unable to Locate Server

- Causes: server down, different version of server binary, …
- Fixes
  - Return -1 to indicate failure (in Unix use *errno* to indicate failure type)
    - What if -1 is a legal return value?
  - Use exceptions
    - Transparency is lost

Lost Request Message

- Easiest to deal with
- Just retransmit the message!
- If multiple message are lost then
  - “client is unable to locate server” error

Lost Reply Message

- Far more difficult to deal with: client doesn’t know what happened at server
  - Did server execute the procedure or not?
- Possible fixes
  - Retransmit the request
    - Only works if operation is **idempontent**: it’s fine to execute it twice
  - What if operation not idempotent?
    - Assign unique sequence numbers to every request

Server Crashes

- Two cases
  - Crash after execution
  - Crash before execution
- Three possible semantics
  - At least once semantics
    - Client keeps trying until it gets a reply
  - At most once semantics
    - Client gives up on failure
  - Exactly once semantics
    - Can this be correctly implemented?

Client Crashes

- Let's the server computation **orphan**
- Orphans can
  - Waste CPU cycles
  - Lock files
  - Client reboots and it gets the old reply immediately

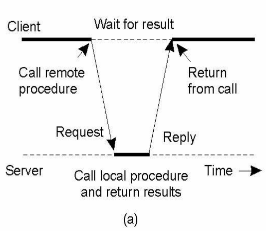
Client Crashes: Possible Solutions

- Extermination:
  - Client keeps a log, reads it when reboots, and kills the orphan
  - Disadvantage: high overhead to maintain the log
- Reincarnation:
  - Divide times in epochs
  - Client broadcasts epoch when reboots
  - Upon hearing a new epoch servers kills the orphans
  - Disadvantage: doesn't solve problem when network partitioned
- Expiration:
  - Each RPC is given a lease T to finish computation
  - If it does not, it needs to ask for another lease
  - If client reboots after T sec all orphans are gone
  - Problem: what is a good value of T?

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

Asynchronous RPC (1)



a) The interconnection between client and server in a traditional RPC
b) The interaction using asynchronous RPC

Asynchronous RPC (2)

- A client and server interacting through two asynchronous RPCs