INTER-PROCESS COMMUNICATION

Module Number 2. Section 9a
COP4600 – Operating Systems
Richard Newman

NEED FOR INTERPROCESS COMMUNICATION

We often think of processes as acting in isolation

In fact, much of the role of the OS is to insure that processes do not interfere with each other

But it is common for processes to communicate

One process may record audio, another may record video, yet another may take audio and video files and edit them into a movie, which is paid by yet another.

Processes may need to communicate more interactively, e.g. a web server and a database back end, a dispatcher and worker processes, nodes of a botnet

SOLVING INTERPROCESS COMMUNICATION

Two problems must be solved:

- actually getting data from one process to another
- synchronizing the processes

INTERPROCESS COMMUNICATION MECHANISMS

Three IPC mechanisms:

- Shared memory
- Message passing
  - including pipes
- Remote procedure calls (RPC)
  - a.k.a. remote method invocation (RMI)

SHARED MEMORY

- Access
  - Write/read
- Issues
  - How to support shared space
  - Caching
  - Consistency

MESSAGE-PASSING MECHANISMS

- Pipes
  - Unnamed
  - Named
- Message queues/Channels
- Mailboxes
  - Multiple senders/receivers per mailbox
- Networking protocols
  - TCP/IP
  - UDP/IP
  - Many others …

MESSAGE-PASSING ADDRESSING

- Destination Addressing (at sender)
  - Broadcast
  - Multicast
  - Unicast
    - Process/interface/port
    - Mailbox – does more than one receiver get a copy?
- Source Addressing (at receiver)
  - Specific process/interface/port/channel
  - Mailbox – see above
  - Any process – collect sender ID/return address on receipt

MESSAGE-PASSING ISSUES

- Message size
  - Fixed size
  - Variable size – what is maximum?
- Capacity/buffering
  - How many messages/how much data can be stored in transit?
  - Where are in-transit messages stored?
    - Sender? Receiver? Mailbox? OS? In the network?
- Reliability
  - Unicast
  - Multicast
- Ordering
  - Duplicates
  - Omissions
  - Order consistency definitions in multisource multicast/broadcast

MESSAGE-PASSING SYNCHRONIZATION

- Blocking
  - Block on receive
  - Block on send
  - Both?
    - Rendezvous mechanism
  - Neither?
- Has implications for storage

RPC/RMI

- Remote procedure calls/remote method invocation
  - Procedural vs. object-oriented – but same idea
- Client-server communication
  - Built on top of message-passing
- Stub procedures for transparency
  - Client stub – provides procedure call API
    - Looks like normal procedure call, handles message-passing
  - Server stub
    - Handles messages from client stub, invokes service

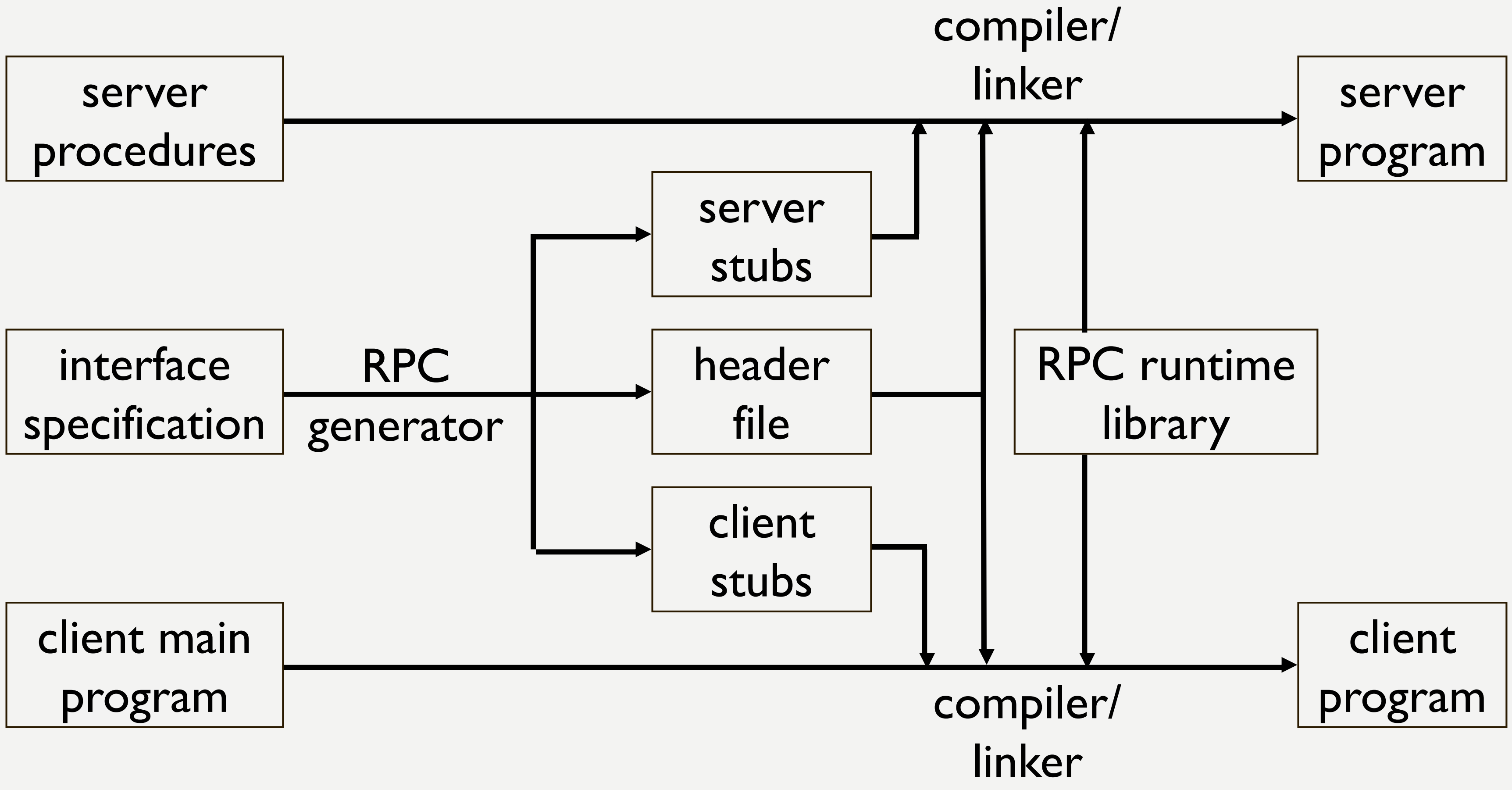
RPC STUB PROCEDURES

Client stub	Server stub
• Called like regular procedure	• Waits for message
• Marshals parameters	
• Sends message to server stub	• Receives message from client stub
• Waits for reply	• Unmarshals parameters
	• Invokes service
	• Marshals return value(s)
• Receives reply from server stub	• Sends reply message to client stub
• Unmarshals return values	• Waits for next message
• Returns to caller	

RPC ISSUES

- Compilation
  - Getting versions, message-passing right
  - RPCgen
- Binding
  - How does client process associate with server?
    - Hardwired, configured
    - Brokers
  - How does client stub find server stub?
    - Portmapper

RPC COMPILATION



MORE RPC ISSUES

- Data representation
  - Big-endian vs. little-endian
  - Network data format
  - Java object serialization
- Parameter Passing
  - Pass-by-value
  - Pass-by-reference
    - Address space is *not shared*!
  - Copy-in-copy-out
    - What if object is very large?
  - Pass-by-name
    - Client must also have server!

YET MORE RPC ISSUES

- Failure handling
  - Server failure
    - At most once semantics – no retry
    - At least once semantics – idempotent actions
    - Exactly once semantics – reliable atomic transactions
  - Client failure
    - Orphan computations/requests
    - Resurrection

UP NEXT:

- Process synchronization
  - Concurrent processing
- Classical synchronization problems