15-410

"...Mooooo!..."

IPC & RPC
Apr. 7, 2004

Dave Eckhardt

Bruce Maggs

Synchronization

Project 3 tactical considerations

- Getting the shell running (step 17/24) is important
  - We won't build a hand-load kernel for each test!
  - Test harness will rely on shell to launch programs
- Getting a body of code solid is important
  - Better for `exec()` to work 1000 times than `thr_fork` once
- Run tests as soon as you can
- Carefully consider the overtime week
  - In general, getting a really solid kernel is the best thing
    - » For your grade
    - » For your education!

Outline

IPC – InterProcess Communication

RPC – Remote Procedure Call

Textbook

- **Sections 4.5, 4.6**

Scope of “IPC”

Communicating processes on one machine

Multiple machines?

- Virtualize single-machine IPC
- Switch to a “network” model
  - Failures happen
  - Administrative domain switch
  - ...
  - (RPC)

IPC parts

Naming

Synchronization/buffering

Message body issues

- Copy vs. reference
- Size

Naming

Message sent to *process* or to *mailbox*?

Process model

- send(P, msg)
- receive(Q, &msg) or receive(&id, &msg)

No need to set up "communication link"

- But you need to know process id's
- You get only one "link" per process pair

Naming

Mailbox model

- send(box1, msg)
- receive(box1, &msg) or receive(&box, &msg)

Where do mailbox id's come from?

- “name server” approach

```
box = createmailbox();
register(box1, “Terry's process”);
boxT = lookup(“Terry's process”);
```

File system approach – *great* (if you have one)

Multiple Senders

Problem

- Receiver needs to know who sent request

First-cut solution

- Sender includes identifier in message body
- Problem?

Typical solution

- "Message" not just a byte array
- OS imposes structure
  - sender id (maybe process id and mailbox id)
  - maybe: type, priority, ...

Multiple Receivers

Problem

- Service may be "multi-threaded"
- Multiple receives waiting for one mailbox

Typical solution

- OS "arbitrarily" chooses receiver per message
  - (Can you guess how?)

Synchronization

Issue

- Does communication imply synchronization?

Blocking send()?

- Ok for request/response pattern
- Provides assurance of message delivery
- *Bad* for producer/consumer pattern

Non-blocking send()?

- Raises buffering issue (below)

Synchronization

Blocking receive()?

- Ok/good for "server thread"
  - Remember, de-scheduling is a kernel *service*
- Ok/good for request/response pattern
- Awkward for some servers
  - Abort connection when client is "too idle"

Pure-non-blocking receive?

- Ok for polling
- Polling is costly

Synchronization

Receive-with-timeout

- Wait for message
- Abort if timeout expires
- Can be good for real-time systems
- What timeout value is appropriate?

Synchronization

Meta-receive

- Specify a group of mailboxes
- Wake up on first message

Receive-scan

- Specify list of mailboxes, timeout
- OS indicates which mailbox(es) are “ready” for what
- Unix: select(), poll()

Buffering

Issue

- How much space does OS provide "for free"?
- "Kernel memory" limited!

Options

- No buffering
  - implies blocking send
- Fixed size, undefined size
  - Send blocks *unpredictably*

A Buffering Problem

P1

```
send(P2, p1-my-status)
receive(P2, &p1-peer-status)
```

P2

```
send(P1, p2-my-status)
receive(P1, &p2-peer-status)
```

What's the problem?

- **Can you draw a picture of it?**

Message Size Issue

Ok to copy *small* messages sender ⇒ receiver

Bad to copy *1-megabyte* messages

- **(Why?)**

"Chop up large messages" evades the issue

“Out-of-line” Data

Message can *refer to* memory regions

- (page-aligned, multiple-page)
- Either “copy” or *transfer ownership* to receiver
- Can share the physical memory
  - Mooooo!

“Rendezvous”

Concept

- Blocking send
- Blocking receive

Great for OS

- No buffering required!

Theoretically interesting

Popular in a variety of languages

- (most of them called “Ada”)

Example: Mach IPC

Why study Mach?

- "Pure" "clean" capability/message-passing system
- Low abstraction count
- This *is* CMU...

Why not?

- Failed to reach market
- Performance problems with multi-server approach?

Verdict: hmm... (GNU Hurd? Godot??)

Mach IPC – ports

Port: Mach "mailbox" object

- One receiver
  - (one "backup" receiver)
- Potentially many senders

Ports identify system objects

- Each task identified/controlled by a port
- Each *thread* identified/controlled by a port
- Kernel exceptions delivered to "exception port"
  - "External Pager Interface" - page faults in user space!

Mach IPC – Port Rights

Receive rights

- "Receive end" of a port
- Held by one task
- Capability typically unpublished
  - receive rights imply ownership

Send rights

- "Send end" - ability to transmit message to mailbox
- Frequently published via "name server" task
- Confer no rights (beyond "denial of service")

Mach IPC – Message

Memory region

- In-line for "small" messages (copied)
- Out-of-line for "large" messages
  - Sender may de-allocate on send
  - Otherwise, copy-on-write

"Port rights"

- Sender specifies task-local port #
- OS translates to internal port-id while queued
- Receiver observes task-local port #

Mach IPC – Operations

send

- block, block(n milliseconds), don't-block
- “send just one”
  - when destination full, queue 1 message in *sender thread*
  - sender notified when transfer completes

receive

- receive from port
- receive from *port set*
- block, block(n milliseconds), don't-block

Mach IPC – "RPC"

Common pattern: "Remote" Procedure Call

Client synchronization/message flow

- Blocking send, blocking receive

Client must allow server to respond

- Transfer "send rights" in message
  - "Send-once rights" speed hack

Server message flow (N threads)

- Blocking receive, non-blocking send

Mach IPC – Naming

Port send rights are OS-managed capabilities

- unguessable, unforgeable

How to contact a server?

- Ask the name server task
  - *Trusted* – source of all capabilities

How to contact the name server?

- Task creator specifies name server for new task
  - Can create custom environment for task tree

IPC Summary

Naming

- Name server?
- File system?

Queueing/blocking

Copy/share/transfer

A Unix surprise

- sendmsg()/recvmsg() pass file descriptors!

RPC Overview

RPC = Remote *Procedure Call*

Concept: extend IPC across machines

- **Maybe across "administrative domains"**

Marshalling

Server location

Call semantics

Request flow

RPC Model

Approach

```
d = computeNthDigit(CONST_PI, 3000);
```

- **Abstract away from "who computes it"**
- **Should "work the same" when remote Cray does the job**

Issues

- **Must specify server *somehow***
- **What "digit value" is "server down"?**
  - **Exceptions useful in "modern" languages**

Marshalling

Values must cross the network

Machine formats differ

- **Integer byte order**
  - **www.scieng.com/ByteOrder.PDF**
- **Floating point format**
  - **IEEE 754 or not**
- **Memory packing/alignment issues**

Marshalling

Define a "network format"

- **ASN.1 - "self-describing" via in-line tags**
- **XDR – not**

"Serialize" language-level object to byte stream

- **Rules typically recursive**
  - **Serialize a struct by serializing its fields in order**
- **Implementation probably should *not* be recursive**
  - **(Why not?)**

Marshalling

Issues

- Some types don't translate well
  - Ada has ranged integers, e.g., 44..59
  - Not everybody really likes 64-bit ints
  - Floating point formats are religious issues
- Performance!
  - Memory speed $\cong$ network speed
- The dreaded “pointer problem”

Marshalling

```
struct node {
  int value;
  struct node *neighbors[4];
} nodes[1024];

nnodes = sizeof(nodes)/sizeof(nodes[0]);

n = occupancy(nodes, nnodes);
bn = best_neighbor(node);
i = value(node);
```

Implications?

Marshalling

```
n = occupancy(nodes, nnodes);
```

- **Marshall array – ok**

```
bn = best_neighbor(node);
```

- **Marshall graph structure – not so ok**

```
i = value(node);
```

- ***Avoiding* marshalling graph – not obvious**
  - **“Node fault”??**

Server Location

Which machine?

- Multiple AFS cells on the planet
- Each has multiple file servers

Approaches

- Special hostnames: *www*.cmu.edu
- Machine lists
  - AFS CellSrvDB /usr/vice/etc/CellServDB
- DNS SRV records (RFC 2782)

Server Location

Which port?

- Must distinguish services on one machine
  - Single machine can be AFS volume, vldb, pt server
- Fixed port assignment
  - AFS: fileserver UDP 7000, volume location 7003
  - /etc/services or www.iana.org/assignments/port-numbers
  - RFC 2468 www.rfc-editor.org/rfc/rfc2468.txt
- Dynamic port assignment
  - Contact “courier” / “matchmaker” service via RPC
  - ...on a fixed port assignment!

Call Semantics

Typically, caller blocks

- Matches procedure call semantics

Blocking can be expensive

- By a factor of *a million(!!)* over real procedure call

"Asynchronous RPC"

- Transmit request, do other work, check for reply
- Not really "PC" any more
- More like programming language "futures"

Fun Call Semantics

Batch RPC

- Send *list* of procedure calls
- Later calls can use results of earlier calls

Issues

- Abort batch if one call fails?
  - Yet another programming language?
- Typically wrecks "procedure call" abstraction
  - Your code must make N calls before 1st answer

Fun Call Semantics

Batch RPC Examples

- NFS v4 (eventually), RFC 3010
- Bloch, A Practical Approach to Replication of Abstract Data Objects

Sad Call semantics

Network failure

- Retransmit request
  - How long?

Server reboot

- Does client deal with RPC session restart?
- Did the call "happen" or not?
  - Retransmitting "remove foo.c" all day long may not be safe!

Client Flow

Client code calls *stub* routine

- "Regular code" which encapsulates the magic

Stub routine

- Locates communication channel
  - If not established: costly location/set-up/authentication
- Marshals information
  - Procedure #, parameters
- Sends message, awaits reply
- Unmarshals reply, returns to user code

Server Flow

Thread pool runs *skeleton* code

Skeleton code

- Waits for request from a client
- Locates client state
  - Authentication/encryption context
- Unmarshals parameters
- Calls “real code”
- Marshals reply
- Sends reply

RPC Deployment

Define interface

- Get it right, you'll live with it for a while!
- AFS & NFS RPC layers ~15 years old

“Stub generator”

- Special-purpose compiler
- Turns “interface spec” into stubs & skeleton

Link stub code with client & server

Run a server!

Java RMI

*R*emote *M*ethod *I*nvocation

Serialization: programmer/language cooperation

- *Dangerously* subtle!
  - Bloch, Effective Java

RMI > RPC

- Remote methods $\cong$ remote procedures
- *Parameters* can be (differently) remote
  - Client on A can call method of class implemented on B passing object located on C
    - » (slowly)

RPC Summary

RPC is lots of fun

So much fun that lots of things don't do it

- SMTP
- HTTP

RPC = IPC

- + server location, marshalling, network failure, delays
- - special copy tricks, speed

Remote Objects? Effective Java, Bitter Java