Recap: Socket abstraction

Remote Procedure Call

- Hide complexity of message-based communication from developers
- Procedure calls more natural for inter-process communication
- Goals of RPC:
  - Client sending request → function call
  - Client receiving response → returning from function
  - Server receiving request → function invocation
  - Server sending response → returning to caller

RPC abstraction via stub functions on client and server

RPC stubs

- Client stub:
  - Construct message with function name and parameters
  - Send request message to server
  - Receive response from server
  - Return response to client

- Server stub:
  - Receive request message
  - Invoke correct function with specified params
  - Construct response message with return value
  - Send response to client stub

Producer-consumer in client-server paradigm

```
client_produce() {
    send produce request to server
    wait for response
}
server() {
    receive request
    if (produce request) {
        server_produce()
    } else {
        server_consume()
    }
    send response
}
```

Producer-consumer using RPC

```
int produce(int n) {
    int status;
    send (sock, &n, sizeof(n));
    recv (sock, &status, sizeof(status));
    return(status);
}
```

```
void produce_stub(int n) {
    int status;
    recv (sock, &status, sizeof(status));
    status = produce(n);
    send (sock, &status, sizeof(status));
    return(status);
}
```
Generation of stubs

- Stubs can be generated automatically
- What does OS need to know to do this?
- Interface description:
  - Types of arguments and return value
- Try rpcgen on Linux

RPC Transparency

- RPC makes remote communication look like local procedure calls
  - Basis of CORBA, Thrift, SOAP, Java RMI, ...
- What factors break illusion?
  - Failures – remote nodes/networks can fail in more ways than with local procedure calls
  - Performance – remote communication is inherently slower than local communication
  - Service discovery – client stub needs to bind to server stub on appropriate machine

RPC Arguments

- Cannot pass pointers as arguments
  - Why?
- How to pass a pointer as argument?
  - Client stub transfers data at the pointer
  - Server stub stores received data and passes pointer
- Challenge:
  - Data representation should be same on either end
  - Example: Little endian vs. big endian
  - 0xDE AD BE EF will become 0xEF BE AD DE

Making a distributed system look like a local system

- RPC: make request/response look like function call/return
- DSM: make multiple memories look like a single memory
- DFS: make disks on multiple computers look like a single file system
- Parallelizing compilers: make multiple CPUs look like one CPU
- Process migration (and RPC): allow users to easily use remote processors

Building distributed applications

- Why build distributed applications?
- Performance
  - Aggregate performance of many computers can be faster than that of (even a fast) single computer
- Reliability
  - Try to provide continuous service, even if some computers fail
  - Try to preserve data, even if some storage fails

Concurrency and distribution

- Distributed programs are multi-threaded, since each computer has at least one thread
- Need two mechanisms to write multi-threaded programs
  - Atomic primitive to synchronize threads
  - A way to share data between threads
- How do we do these in a distributed system?
  - Sending/receiving of messages
- Can there be race conditions without shared data?
Structuring a concurrent system

- One multi-threaded process on one computer

- Several multi-threaded processes on several computers

December 3, 2015

Administration

- Please submit course evaluations!
- Incentive:
  - No. of bonus submissions on Project 4 = max(3, floor(% of students who have submitted eval / 10))
- Topics that you would like me to review?
  - Email me before class on Tuesday
  - I’ll review most popular requests

December 3, 2015

Structuring a concurrent system

- Several multi-threaded process on each of several computers

- Why separate threads on one computer into separate address spaces, then use send/receive to communicate and synchronize?
  - Protects modules from each other
  - Microkernels
    - OS structure that separates OS functionality into several server processes, each in its own address space

December 3, 2015