Distributed Systems Intro

CS 537: Introduction to Operating Systems

Louis Oliphant & Tej Chajed

University of Wisconsin - Madison

Spring 2024

Administrivia

- Project 7 due April 30th @ 11:59pm
- Final Exam:
  - **Cumulative, focusing on new material**
  - Lec 1 - May 8th, 12:25-2:25 (Biochem 1125)
  - Lec 2 - May 6th, 2:45-4:45 (Sterling Hall 1310)
  - McBurney: May 6th, 2:40-6:50 (Nancy Nicholas Hall 1135)
  - If you can't take the exam for a *legitimate reason* at your designated time, please fill out the alternate exam form to take the exam with the other lecture. Legitimate Reasons include:
    - Another exam at the same time, Religious conflict, University Sanctioned conflict, Scheduled Medical conflict, Civic Duty (e.g. jury duty), Military Service, Family Caregiving Responsibility, Family Emergency, Serious Illness, 3 or more exams scheduled during a 24 hour period

Review SSDs

- Physical Organization (SLC,MLC,TLC and Blocks and Pages)
- IO and translation to Read (a page), Erase (a block), Program (a page) Operations
- Flash Translation Layer (FTL) including mapping tables
- Log System, including data organization and garbage collection

Persistence Summary

- Managing I/O devices significant part of OS
- Disk Drives, SSDs (pages, blocks)
- File Systems: OS provided API to access disk
- Simple FS: FS layout with supberblock, bitmaps, inodes, datablocks
- Fast File System: Key idea – put inode & data close together, namespace locality
- FSCK, Journaling – Handling/Preventing data inconsistencies
- Log Structured File System - Organize data based on writes

Quiz 22 SSDs

https://tinyurl.com/cs537-sp24-q22

Distributed Systems

Building Distributed Systems That Work When Components Fail

- System objectives of performance, security, **communication**
- Unreliable Communication Layers (**UDP**)
  - **checksum**
- Reliable Communication Layers (**TCP**)
  - acknowledgement, timeout/retry
  - sequence counter
- Communication Abstractions
  - Distributed Shared Memory (DSM)
  - Remote Procedure Call (**RPC**)
    - Stub Generator
    - Run-Time Library
    - Other Issues: fragmentation/reassembly, byte ordering, synchronicity

WHAT IS A DISTRIBUTED SYSTEM?

A distributed system is one where a machine I've never heard of can cause my program to fail.
— *Leslie Lamport*

Definition: More than one machine working together to solve a problem

Examples:

- client/server: web server and web client
- cluster: page rank computation

WHY GO DISTRIBUTED?

More computing power

More storage capacity

Fault tolerance

Data sharing

NEW CHALLENGES

System failure: need to worry about partial failure

Communication failure: links unreliable

- bit errors
- packet loss
- node/link failure

COMMUNICATION OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC

RAW MESSAGES: UDP

UDP : User Datagram Protocol

API:

- reads and writes over socket file descriptors
- messages sent from/to ports to target a process on machine

Provide minimal reliability features:

- messages may be lost
- messages may be reordered
- messages may be duplicated
- only protection: checksums to ensure data not corrupted

RAW MESSAGES: UDP

Advantages

- Lightweight
- Some applications make better reliability decisions themselves (e.g., video conferencing programs)

Disadvantages

- More difficult to write applications correctly

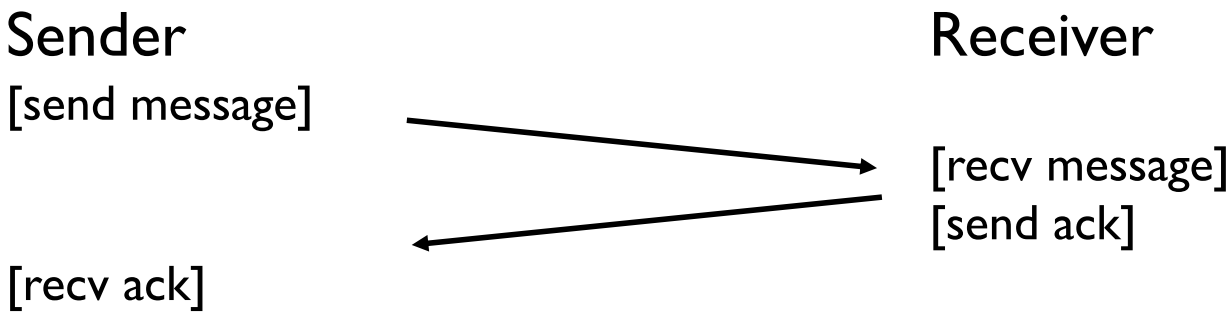
RELIABLE MESSAGES: LAYERING STRATEGY

TCP: Transmission Control Protocol

Using software to build

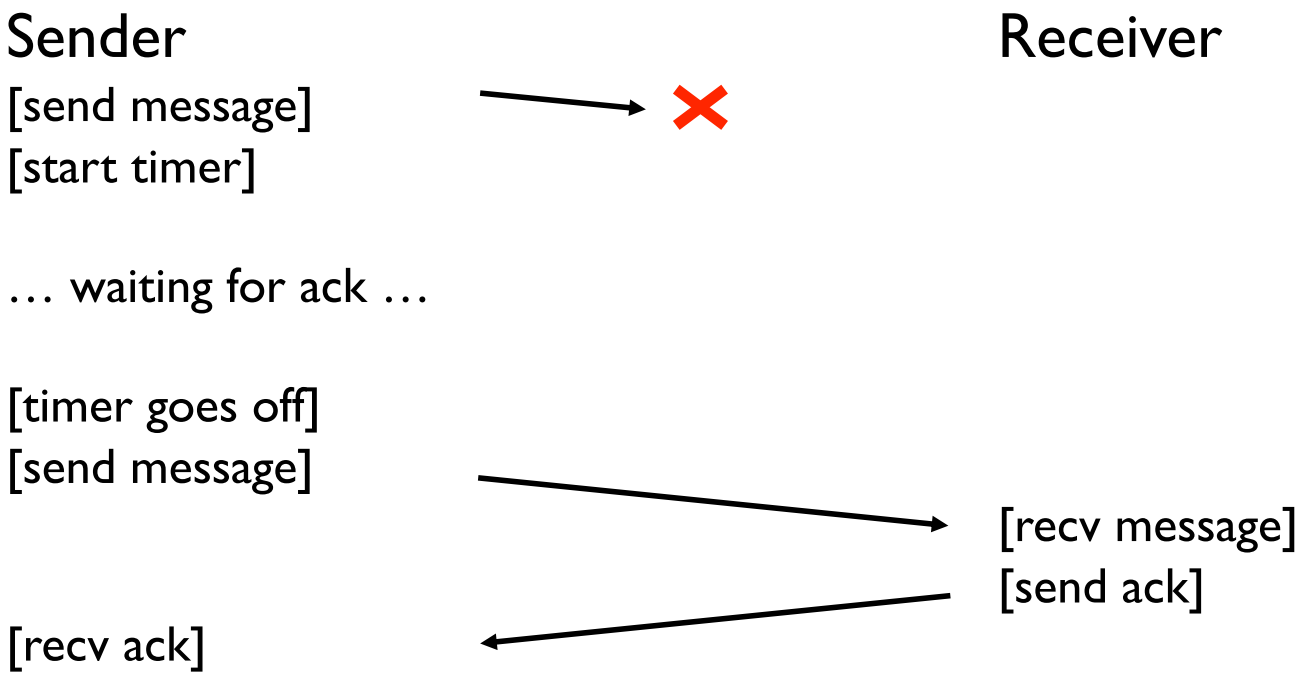
reliable logical connections over unreliable physical connections

TECHNIQUE #1: ACK



Ack: Sender knows message was received
What to do about message loss?

TECHNIQUE #2: TIMEOUT



TIMEOUT

How long to wait?

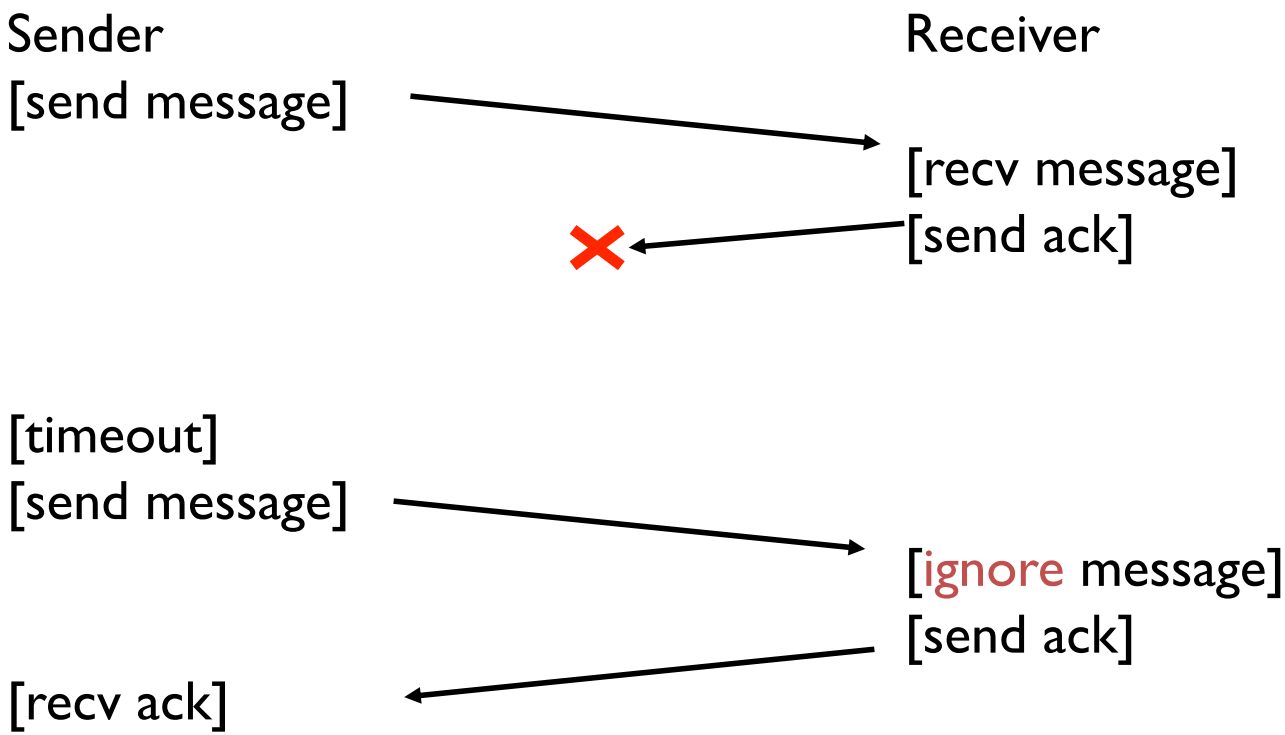
Too long?

- System feels unresponsive

Too short?

- Messages needlessly re-sent
- Messages may have been dropped due to overloaded server. Resending makes overload worse!

LOST ACK PROBLEM



SEQUENCE NUMBERS

Sequence numbers

- senders gives each message an increasing unique seq number
- receiver knows it has seen all messages before N

Suppose message K is received.

- if K <= N, Msg K is already delivered, ignore it
- if K = N + 1, first time seeing this message
- if K > N + 1 ?

TCP

TCP: Transmission Control Protocol

Most popular protocol based on seq nums

Buffers messages so arrive in order

Timeouts are adaptive

COMMUNICATIONS OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC

RPC

Remote **P**rocedure **C**all

What could be easier than calling a function?

Approach: create wrappers so calling a function on another machine feels just like calling a local function!

RPC

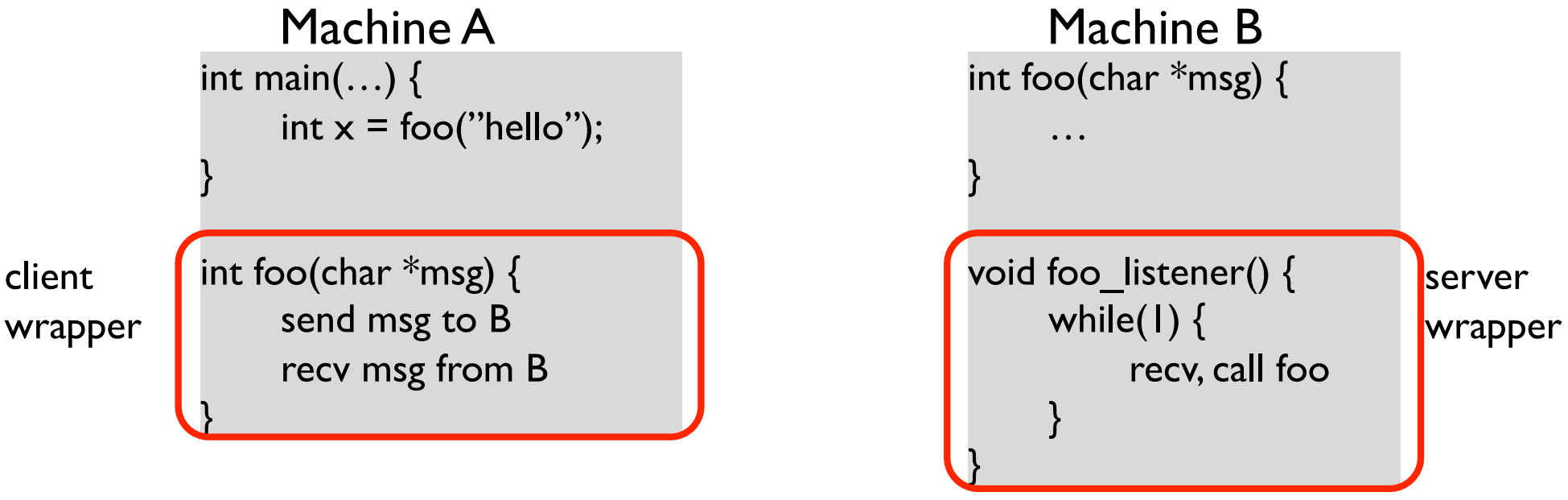
Machine A

```
int main(…) {
        int x = foo(”hello”);
}

int foo(char *msg) {
        send msg to B
        recv msg from B
}
```

Machine B

```
int foo(char *msg) {
        …
}

void foo_listener() {
        while(1) {
                recv, call foo
        }
}
```

RPC


Machine A
int main(…) {
int x = foo("hello");
}
client wrapper
int foo(char *msg) {
send msg to B
recv msg from B
}
Machine B
int foo(char *msg) {
…
}
void foo_listener() {
while(1) {
recv, call foo
}
}
server wrapper

RPC TOOLS

RPC packages help with two components

(1) Runtime library

- Thread pool
- Socket listeners call functions on server

(2) Stub generation

- Create wrappers automatically
- Many tools available (rpcgen, thrift, protobufs)

WRAPPER GENERATION

Wrappers must do conversions:

- client arguments to message
- message to server arguments
- convert server return value to message
- convert message to client return value

Need uniform endianness (wrappers do this)

Conversion is called marshaling/unmarshaling, or serializing/deserializing

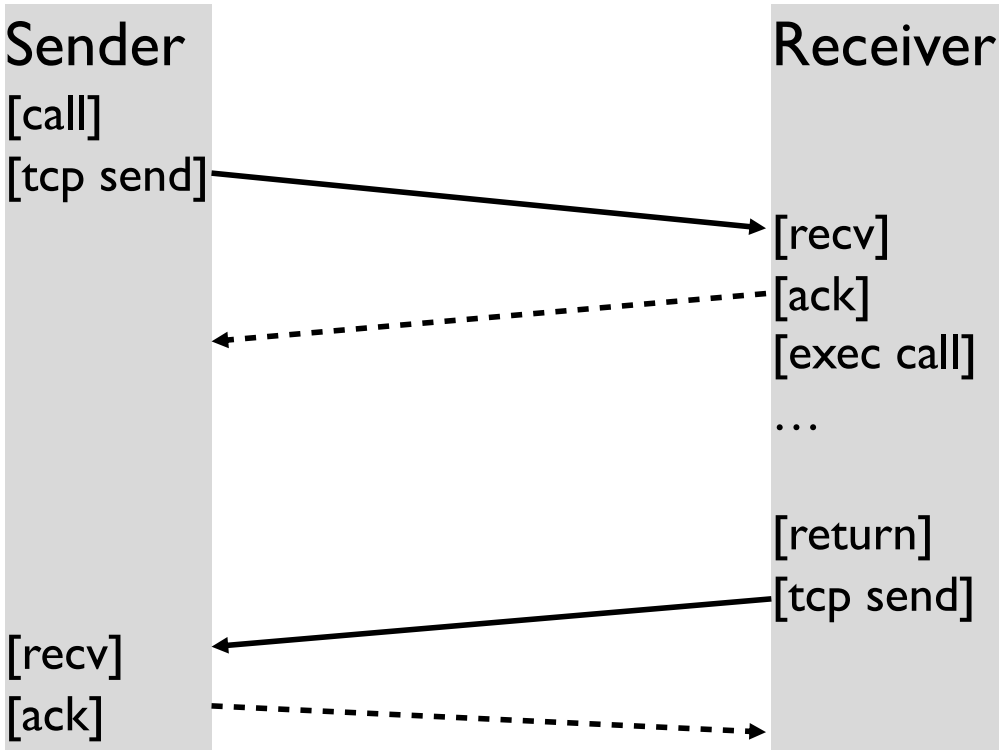
WRAPPER GENERATION: POINTERS

Why are pointers problematic?

Address passed from client not valid on server

Solutions? Smart RPC package: follow pointers and copy data

RPC OVER TCP?


Sender
[call]
[tcp send]
[recv]
[ack]
Receiver
[recv]
[ack]
[exec call]
…
[return]
[tcp send]

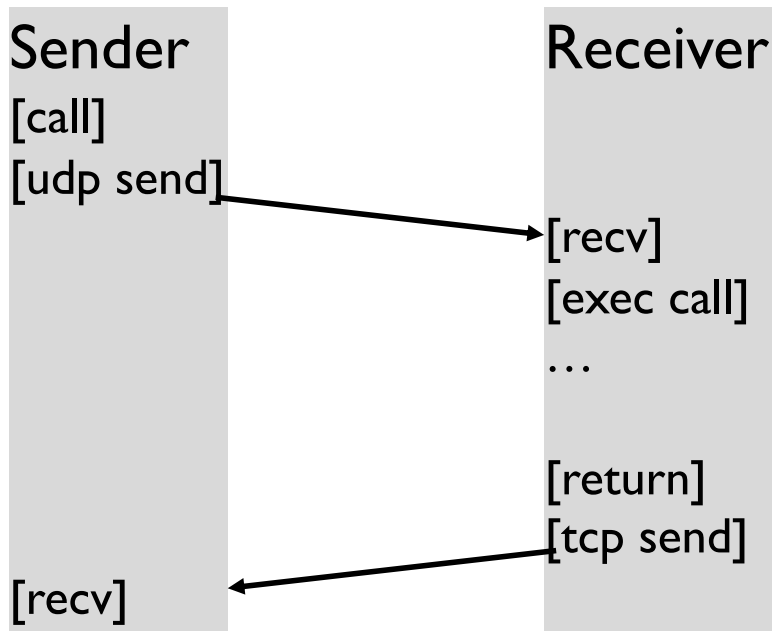
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



Other Issues

- **Long-running calls**, client periodically asks server for results
- **Data Organization** – e.g. Big-Endian vs. Little Endian
  - Sun's XDR (eXternal Data Representation) formatting standard
  - Google's gRDP uses HTTP/2
- Some systems provide both **synchronous** (i.e. wait for result) and **asynchronous** (i.e. return immediately with some type of callback)

Summary

- UDP for unreliable communication
- TCP for reliable communication
- RPC often builds on top of UDP layer, handles communication failures itself
  - has a stub generator and run-time library
  - handles issues like fragmentation and byte ordering
  - Typically synchronous calls (wait for completion)
- RPC packages include:
  - Sun's RPC system
  - Google's gRPC
  - Apache Thrift
  - JSON-RPC