Introduction

Dmitri Loguinov
Texas A&M University

August 27, 2020
 Updates

- Recv loop reminder
  - `timeout.tv_usec` must be initialized to zero
  - NULL-terminate buf before searching with `strchr` or `strstr`

```c
while (true) {
  FD_SET (...);
  if ((ret = select (0, &fd, ..., &timeout)) > 0)
    {
      // new data available; now read the next segment
      int bytes = recv (sock, buf + curPos, allocatedSize - curPos, ...);
      if (errors)
        // print WSAGetLastError() & return false;
      if (connection closed)
        {
          buf[curPos] = NULL;
          return true; // normal completion
        }
      curPos += bytes; // adjust where the next recv goes
      if (allocatedSize - curPos < THRESHOLD)
        // realloc() buf to double its size
    }
  else if (timeout)
    // report timeout & return false;
  else
    // print WSAGetLastError() & return false;
}
```
Chapter 1: Roadmap

1.1 What is the Internet?
1.2 Network edge
1.3 Network core
1.4 Network access and physical media
1.5 Internet structure and ISPs
1.6 Delay & loss in packet-switched networks
1.7 Protocol layers, service models
1.8 History
The Internet: “Nuts and Bolts” View

1) **Hosts** (end systems)
   - Computing devices (servers, desktops, phones, laptops)
   - Run network apps

2) **Routers**
   - Forward packets (chunks of data) to destinations

3) **Communication links**
   - Connect hosts & routers
   - Fiber, copper, radio, satellite
   - Transmission rate = **bandwidth**
The Internet: “Nuts and Bolts” View

4) Protocols
- Control sending/receiving of messages (e.g., TCP, IP, HTTP, FTP, SMTP)

Internet: “network of networks”
- Loosely hierarchical

Who rules the Internet?
- No single authority, mostly decentralized

Internet standards
- IETF: Internet Engineering Task Force
- RFC: Request for comments
What’s a Protocol?

**Human protocols:**
- “What’s the time?”
- “I have a question”
- Introductions

... specific msgs sent
... specific actions taken when msgs received or other events take place

**Network protocols:**
- Machines rather than humans
- All communication activity in the Internet governed by protocols

Protocols define format, order of messages sent and received among network entities, and actions taken on message transmission/receipt
What’s a Protocol?

A human protocol and a computer network protocol:

Hi

Got the time?

2:00

Thanks

You’re welcome

TCP connection request

TCP connection accept

GET index.html

<file>

connection close

OK
Closer Look at Network Structure

- **Network edge:**
  - Applications and hosts
- **Network core:**
  - Routers
  - Links
- **How large is the edge?**
  - Billions of hosts, trillions of web pages, zettabytes of information
- **Large ISPs form the Internet backbone**
  - Terabits per second router speed
Chapter 1: Roadmap

1.1 What is the Internet?
1.2 Network edge
1.3 Network core
1.4 Network access and physical media
1.5 Internet structure and ISPs
1.6 Delay & loss in packet-switched networks
1.7 Protocol layers, service models
1.8 History
Network Edge

- The edge:
  - Responsible for almost all data supply/demand
  - Protocols impact performance

- Client/server model
  - Client host requests, receives service from always-on server
  - Example: web browser/server; email client/server

- Peer-to-peer (P2P) model:
  - Minimal use of dedicated servers; user hosts talk to each other
  - Example: Skype, BitTorrent
Network Edge: Reliable Service

- **Goal**: data transfer between sockets
- **TCP** – Transmission Control Protocol
  - Internet’s reliable service
- **Connection-oriented**
  - *Handshaking*: send connection messages (prepare) for data transfer ahead of time
  - Set up *state* in two communicating hosts

TCP service [RFC 793]

- **Reliable, in-order** byte-stream data transfer
  - Packet loss handled through acknowledgements and retransmissions

- **Flow control**:
  - Sender won’t overwhelm receiver

- **Congestion control**:
  - Senders reduce transmission rate when network becomes congested
Network Edge: Unreliable Service

- **Goal:** data transfer between sockets
  - Same as before!

- **UDP** – User Datagram Protocol [RFC 768]:
  - Connectionless
  - Unreliable data transfer
  - No flow control
  - No congestion control

- **Less overhead and delay**
  - TCP connection setup & termination is 7 packets
  - TCP retransmission delay is potentially unbounded

**Apps using TCP:**
- HTTP (Web), FTP (file transfer), SSH (remote login), SMTP (email)

**Apps using UDP:**
- DNS, SNMP
  - Short (single-packet) transfers
  - No need for congestion management

- Streaming media, online games, IP telephony
  - More sensitive to delay than packet loss
Chapter 1: Roadmap

1.1 What is the Internet?
1.2 Network edge
1.3 Network core
1.4 Network access and physical media
1.5 Internet structure and ISPs
1.6 Delay & loss in packet-switched networks
1.7 Protocol layers, service models
1.8 History
The Network Core

- Supports end-host communication
- **Fundamental question:** how is data transferred through the network?
  - Circuit switching: dedicated circuit per call (telephone network, origin 1800s)
  - Packet-switching: data sent in discrete “chunks” (1960s)
- **Notation**
  - Call = connection = flow
Network Core: Packet Switching

- End-end data stream divided into *packets*
  - Packets of users A and B *share* network resources
  - Each packet uses full link bandwidth

- Resource contention:
  - Aggregate resource demand can exceed amount available
  - *Congestion*: packets queue, wait for link use

- Store-and-forward:
  - Packets move one hop (router) at a time
  - Node receives complete packet before forwarding
Packet Switching: Statistical Multiplexing

- Sequence of A’s and B’s packets does not have a fixed pattern \(\Rightarrow\) *statistical multiplexing*
Packet Switching vs. Circuit Switching

Packet switching allows more users than circuit switching

- 1 Mbps link
- Each user:
  - 100 Kbps when “active”
  - Active 10% of time
- Circuit-switching:
  - Supports 10 users
- Packet switching:
  - With 35 users, probability that more than 10 are active is 0.0424%; with 50 users – 0.94%
  - Max 100 users (if perfectly unsynchronized)
Packet Switching: Store-and-Forward

- Takes $L/R$ seconds to transmit (push out) packet of $L$ bits on to link of $R$ bps
- Entire packet must arrive at router before it can be transmitted on next link: *store and forward*
- Path delay $= 3L/R$

**Example:**
- $L = 7.5$ Mbits
- $R = 1.5$ Mbps
- End-to-end delay $= 15$ sec
**Multi-Threading**

- Threads execute concurrently as part of a process
- **Benefits:**
  - Allows for parallelism in a multiprocessor/multicore system
  - If a blocking call is made in one thread, other threads can continue executing
- **Issues:**
  - Memory is shared between threads, concurrent access requires proper synchronization
  - Order of execution of threads is non-deterministic
- **Homework note:** pass shared parameters to threads using a dedicated class instead of using global variables (see 463-sample.zip on course site)
Multi-Threading 2

• Reasons for using multiple threads in hw #1
  – Web servers respond slowly (1-10 seconds/request)
  – While a thread is suspended waiting for `connect()` and `recv()`, other threads should be allowed to work

• Multiple threads achieve significant speed-up
  – You could run thousands of threads, but limit your testing to 10 until you know it works correctly

• Common synchronization mechanisms
  – **Mutex** (mutual exclusion): allows only one thread access to critical section; others must wait
  – **Semaphore**: allows up to N concurrent threads
  – **Event**: binary (i.e., ON or OFF) signal
Multi-Threading 3

• Mutex usage
  - Any data structure (e.g., queue) or resource (e.g., screen or disk) modified by parallel threads needs to be protected
  - If not, inconsistencies (data corruption) may result

```c
CRITICAL_SECTION cs;
InitializeCriticalSection (&cs);

EnterCriticalSection (&cs); // lock
// critical section here ...
LeaveCriticalSection (&cs); // unlock
```

• Events
  - CreateEvent, WaitForSingleObject, CloseHandle

• See MSDN for additional details
Multi-Threading 4

- A semaphore has a numerical value $s$ attached to it.
- **Wait on semaphore (operation P)**
  - If $s = 0$, the semaphore suspends the calling thread.
  - If $s > 0$, the thread is allowed access and $s$ is set to $s-1$.
- **Release semaphore (operation V)**
  - If threads are waiting, unblock one of them and run it.
  - Otherwise, increment $s = s + 1$.

```c
HANDLE sema = CreateSemaphore (...);
DWORD ret = WaitForSingleObject(sema, INFINITE);  // wait
if (ret != WAIT_OBJECT_0)  // wait
   // report error
   // critical section...

if (ReleaseSemaphore (sema, ...) == FALSE)  // release
   // report error
```