COMP/ELEC 429/556 Introduction to Computer Networks

Creating a Network Application

Some slides used with permissions from Edward W. Knightly, T. S. Eugene Ng, Ion Stoica, Hui Zhang

How to Programmatically Use the DNS System?

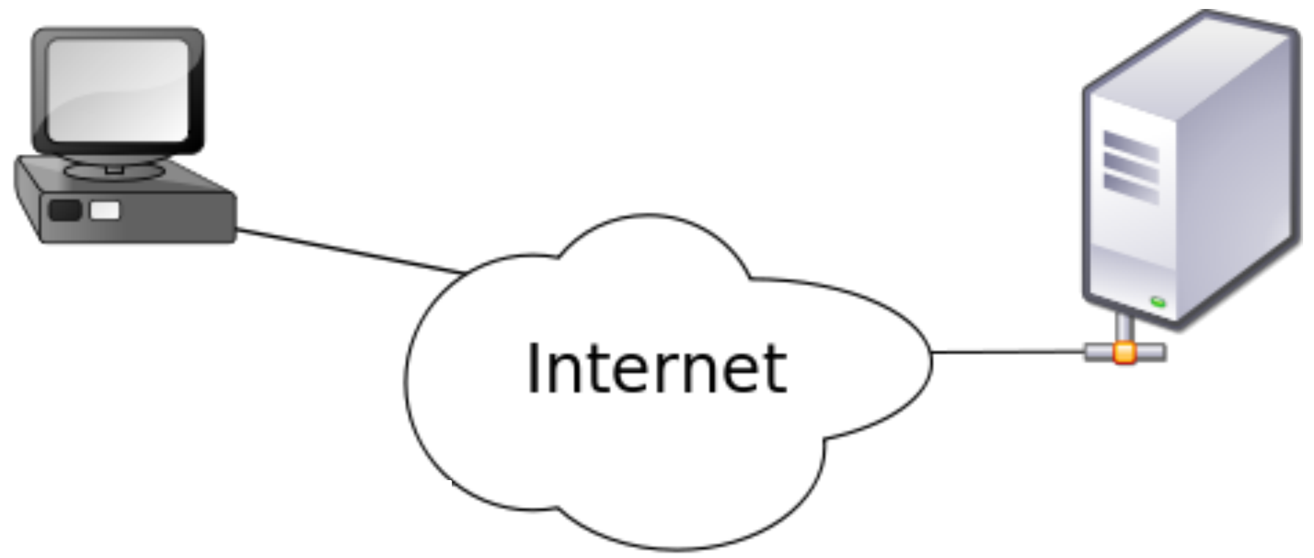
Operating system comes to the rescue

- gethostbyname()
- gethostbyaddr()

- These functions are not re-entrant
  - Cannot have 2 program threads calling them concurrently
  - Call them in your main thread or use the more complicated thread-safe alternatives gethostbyname_r() and gethostbyaddr_r()

- On CLEAR, you can type “man gethostbyname” to get a detailed description of the function
  - “man” stands for manual

How to Programmatically Send/Receive Data over the Internet?

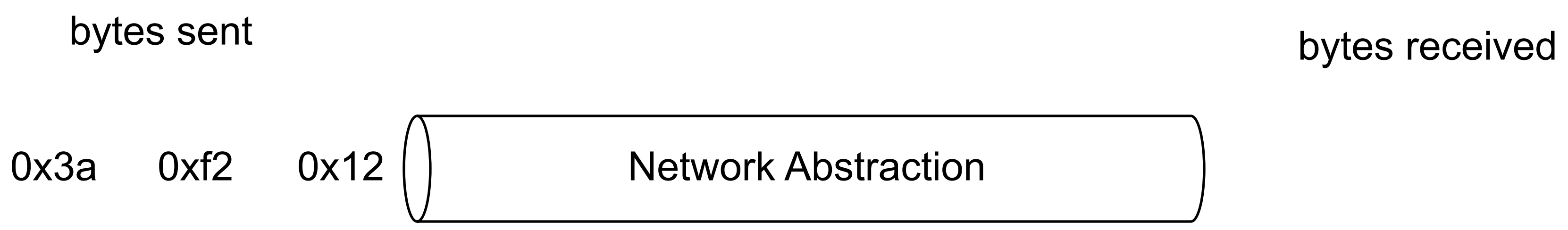


Operating System comes to the rescue

- Reliable byte stream network communications service
  - The most common model
  - Underlies almost every network application you use

Reliable Byte Stream Model

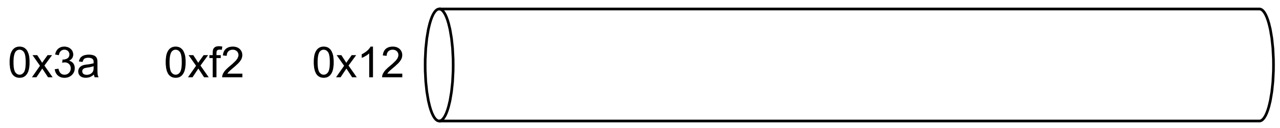
- The elemental transmission unit is a byte (8 bits)
- Bytes are delivered reliably, first in first out
  - We will learn how this is accomplished later



Reliable Byte Stream

- Bytes are delivered without any notion of application message units

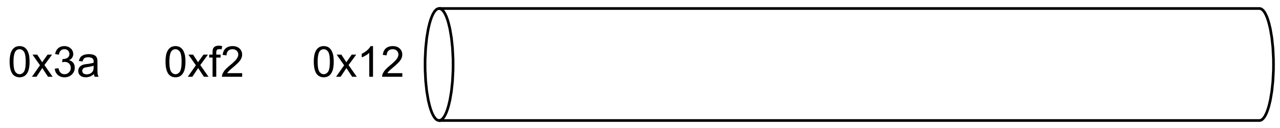
Example 1



1st recv() got 0x12

2nd recv() got 0xf2, 0x3a

Example 2



1st recv() got nothing due to timeout

2nd recv() got 0x12, 0xf2

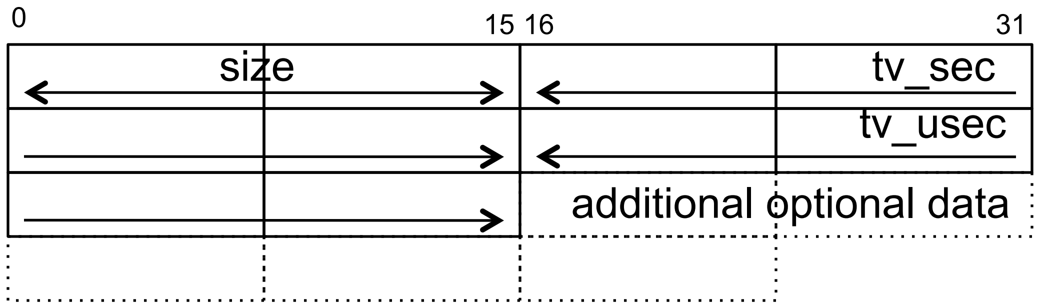
3rd recv() got 0x3a

Application decides message format and how to interpret the byte stream

E.g.

- Each message is a null (0x00) terminated string
- In this case, a program reads in bytes until encountering 0x00 to form a valid message

Project 1 Ping/Pong Message



2 size bytes

0xf2 0x12

What is the size?

0x12f2 = 4,850

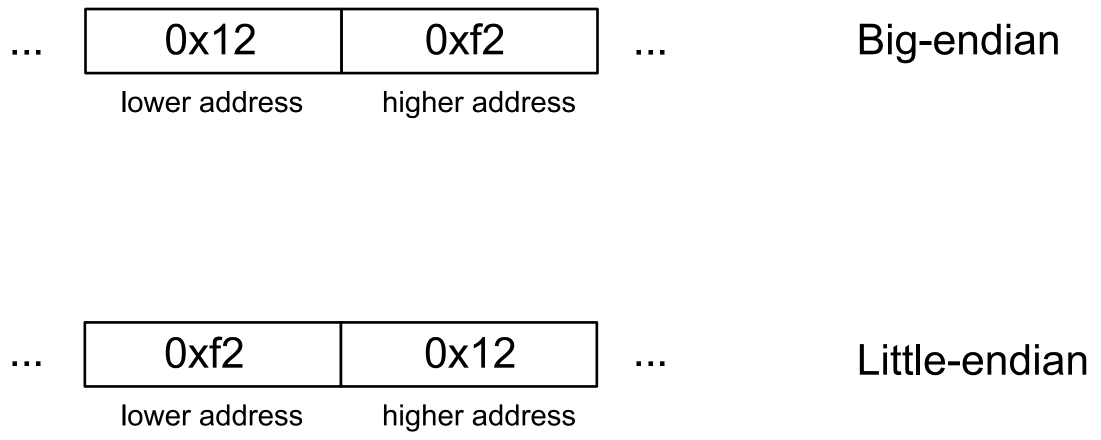
0xf212 = 61,970

Network byte order (a.k.a. Big endian) – first byte sent/received is the most significant

size is 0x12f2 = 4,850

Why the need to specify network byte ordering convention for numbers?

- CPUs may choose little-endian or big-endian format when storing numbers in registers and RAM
  - Most Intel CPUs are “little-endian” (1st byte least significant)
  - Network byte ordering convention is “big-endian” (1st byte most significant)
- The number 4,850 (0x12f2), which is stored in “big-endian” ?



What about bit ordering?

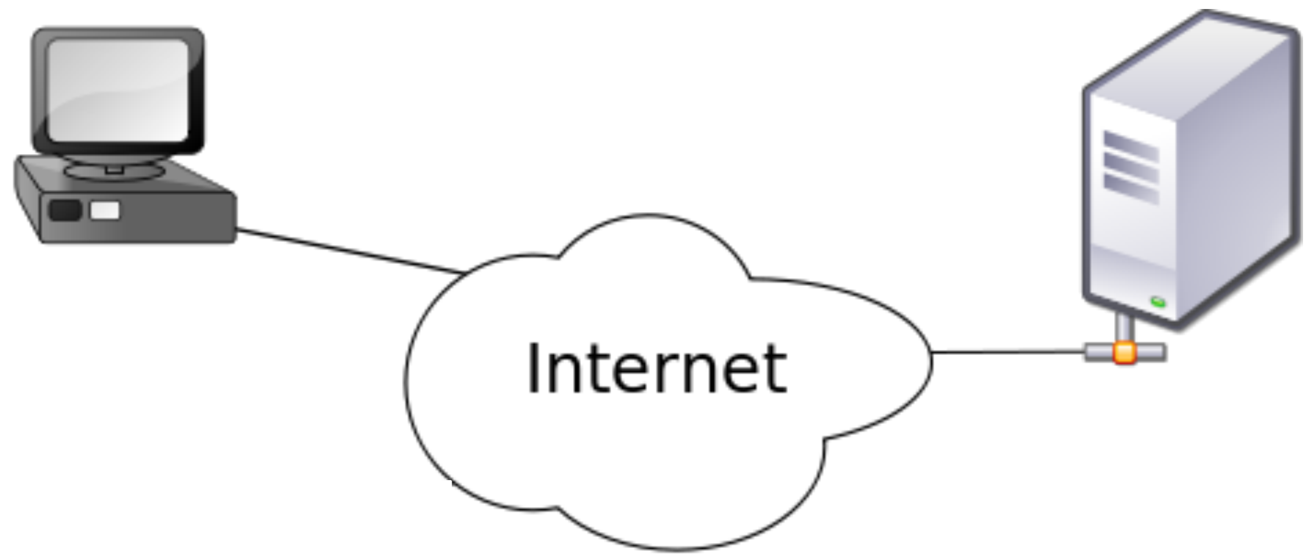
- Which bit in a byte received over the network is most significant?
- OMG, not again!

- Bit ordering within a byte is handled entirely in hardware

Byte ordering conversion functions

- host order to network order
  - htons() for short integer (i.e. 16 bit)
  - htonl() for long integer (i.e. 32 bit)
  - htobe64() for 64 bit integer
- network order to host order
  - ntohs() for short integer (i.e. 16 bit)
  - ntohl() for long integer (i.e. 32 bit)
  - be64toh() for 64 bit integer
- Must be careful in deciding whether conversion is required
  - Is a variable numeric?
  - Is a numeric variable already stored in network byte order?
- These functions are “no-op” on CPUs that use network byte ordering (big-endian) for internal number representation
  - But a program must compile and work regardless of CPU, so these functions have to be used for compatibility

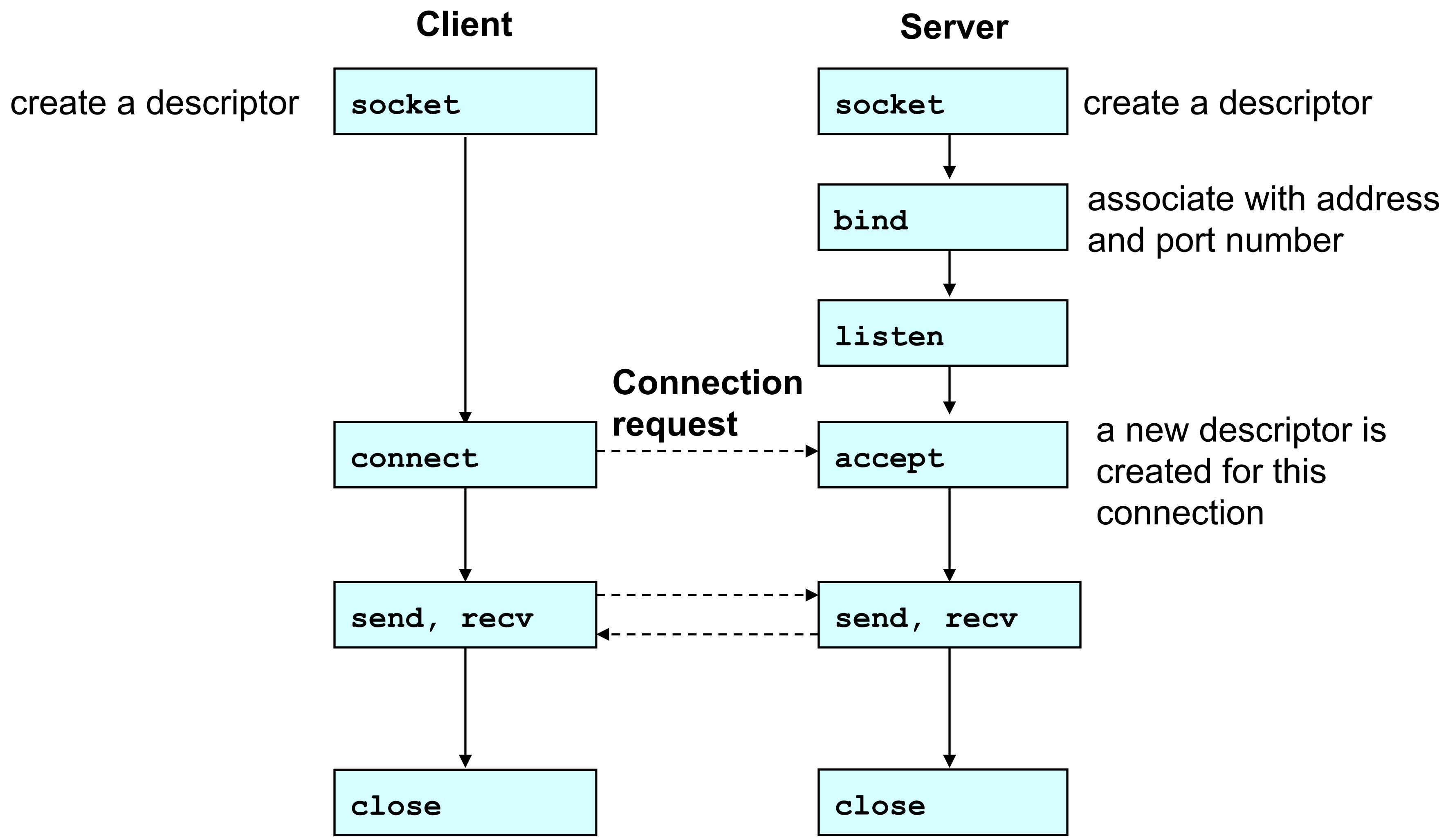
How to Programmatically Send/Receive Data over the Internet?



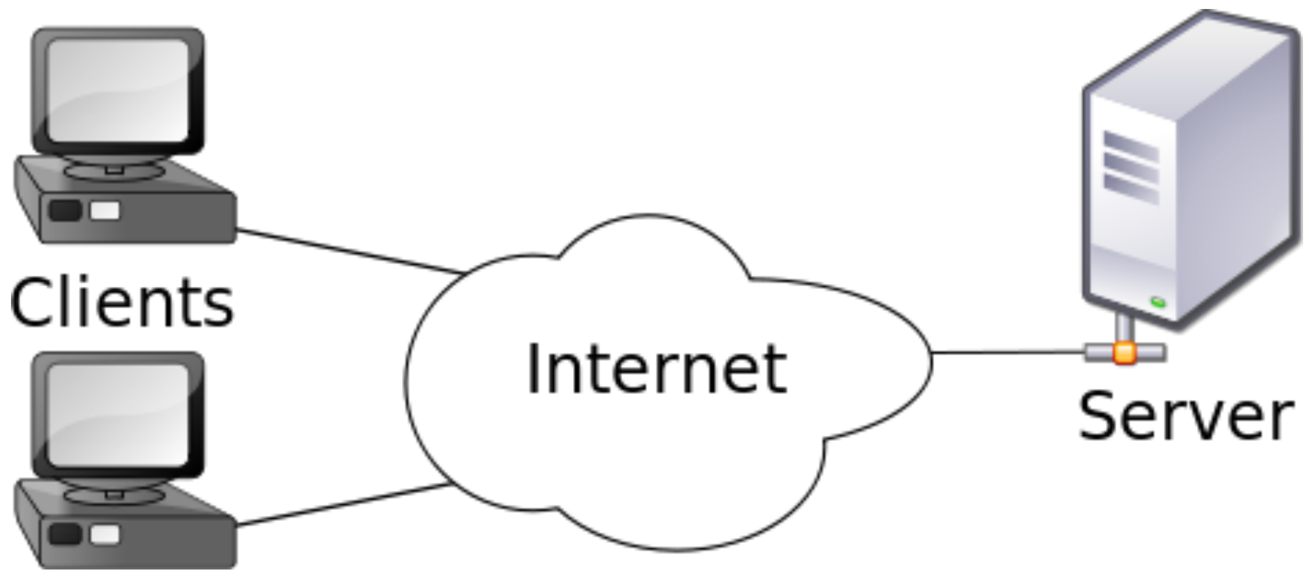
Operating System comes to the rescue

- Socket API
  - Refresh your memory of the mechanics of using socket for network communications

Overview of the Socket API



How to support multiple simultaneous connections?



Operating System comes to the rescue

- Event-driven concurrency
  - Refresh your memory of one way to create the illusion of concurrently handling multiple network conversations even on a uni-processor
- Another way is multi-threading
  - Can achieve real concurrency on multi-processors
  - Refer to COMP321 material

- A combination of event-driven concurrency and multi-thread concurrency is needed to achieve the highest server scalability

Event-Based Concurrent Servers

- Maintain a pool of descriptors
- Repeat the following forever:
  - Block until:
    - New connection request arrives on the listening descriptor
    - New data arrives on an existing connected descriptor
    - A connected descriptor is ready to be written to
    - A timeout occurs (can be configured to have no timeout)
  - If new connection request, add the new connection to the pool of connections
  - If new data arrives, read any available data from the connection
  - If descriptor is ready to be written, write whatever data is pending
  - If timeout, do whatever is appropriate for timeout
- Can wait for input from local I/O (standard input) and remote I/O (socket) simultaneously!
- Many implementations: select(), poll(), epoll(), etc.

Event-Based Concurrent I/O

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

- `readfds, writefds`
  - Opaque bit vector (max `FD_SETSIZE` bits) that indicates membership in a descriptor set
  - If bit *k* is 1, then descriptor *k* is a member of the descriptor set

- `nfds`
  - Maximum descriptor value + 1 in the set
  - Tests descriptors 0, 1, 2, ..., `nfds` - 1 for set membership

- `select()` returns the number of ready descriptors and sets each bit of `readfds, writefds` to indicate the ready status of its corresponding descriptor

Macros for Manipulating Set Descriptors

- `void FD_ZERO(fd_set *fdset);`
  - Turn off all bits in `fdset`
- `void FD_SET(int fd, fd_set *fdset);`
  - Turn on bit `fd` in `fdset`
- `void FD_CLR(int fd, fd_set *fdset);`
  - Turn off bit `fd` in `fdset`
- `int FD_ISSET(int fd, *fdset);`
  - Is bit `fd` in `fdset` turned on?