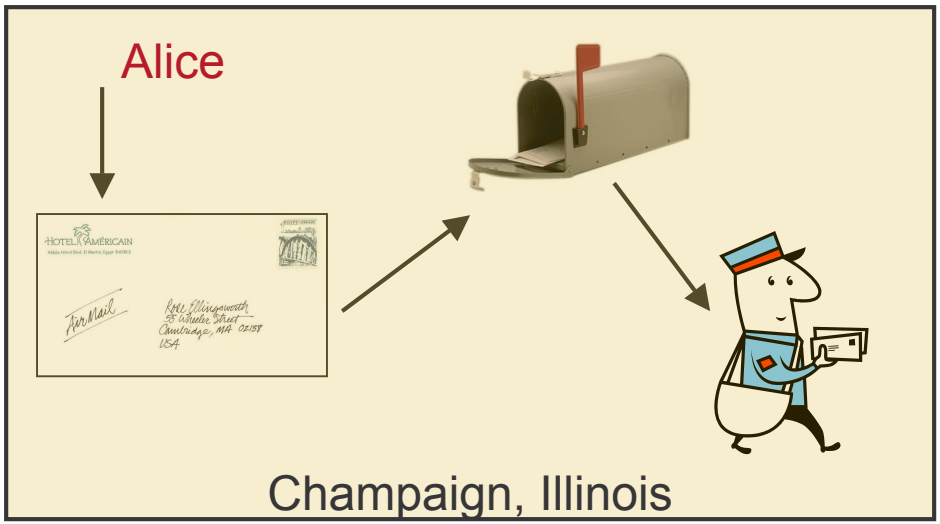
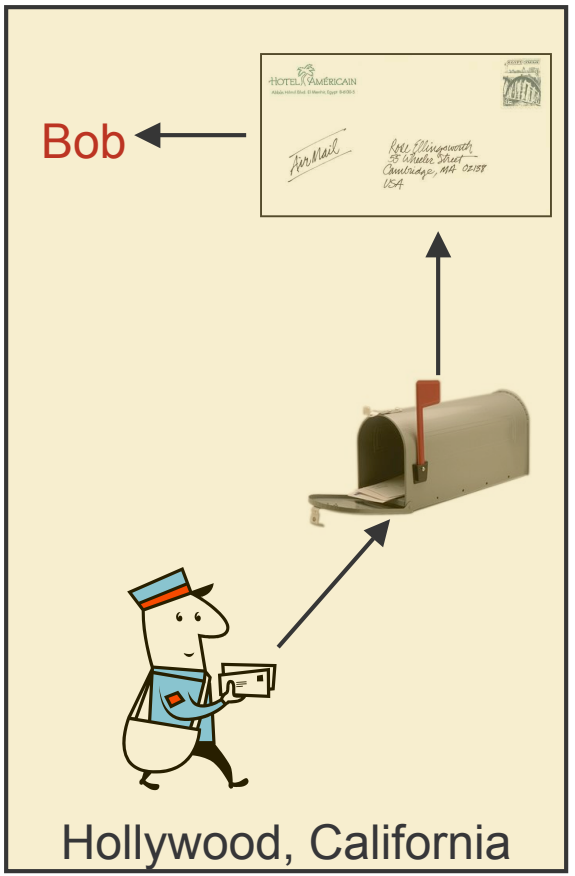
Introduction to Unix Network Programming

Reference: Stevens Unix Network Programming

How do we Communicate?

- Send a mail from Alice to Bob
  - Alice in Champaign, Bob in Hollywood
- Example:
  - US Postal Service





What does Alice do?

Alice
200 Cornfield Rd.
Champaign, IL 61820

Bob
100 Santa Monica Blvd.
Hollywood, CA 90028

- Bob's address (to a mailbox)
- Bob's name – in case people share mailbox
- Postage – have to pay!
- Alice's own name and address – in case Bob wants to return a message

What does Bob do?

Alice
200 Cornfield Rd.
Champaign, IL 61820

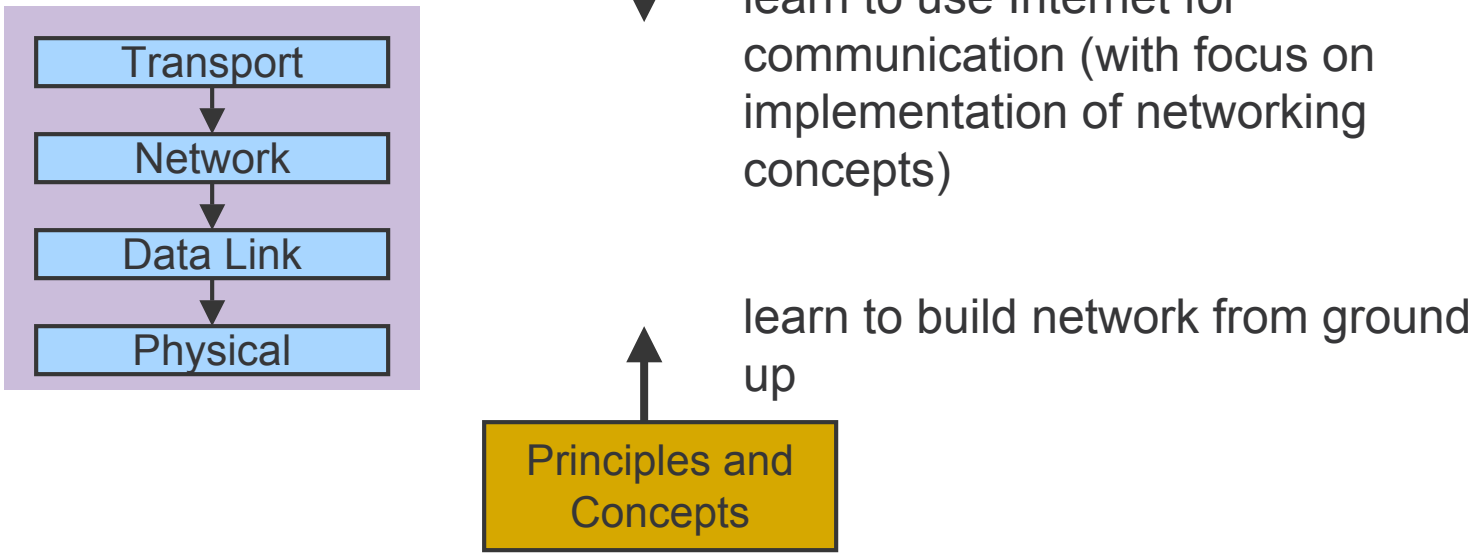
Bob
100 Santa Monica Blvd.
Hollywood, CA 90028

- Install a mailbox
- Receive the mail
- Get rid of envelope
- Read the message

What about sending a TCP/IP packet?

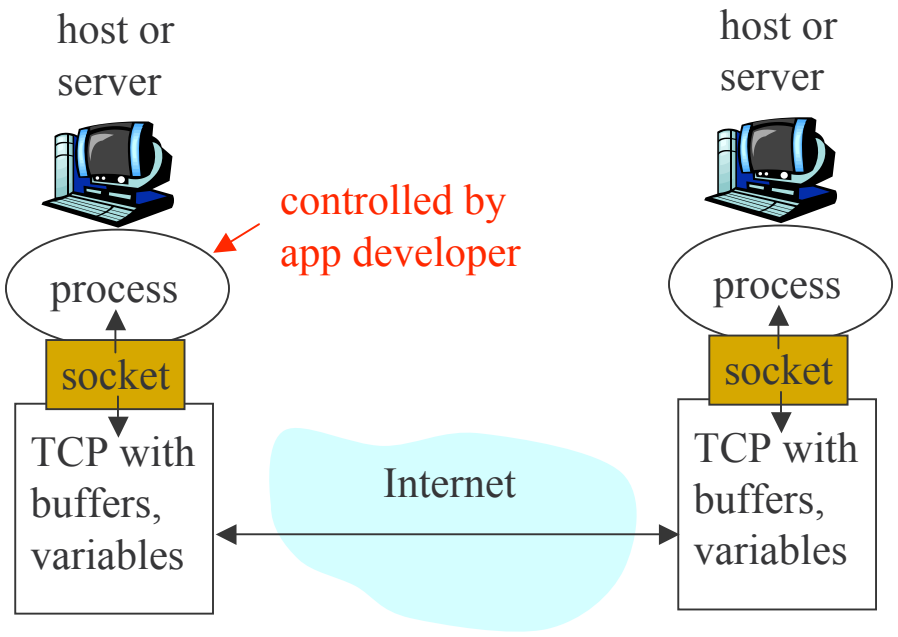
- Very similar to Alice-mailing-to-Bob
- Different terminologies – very confusing
  - We have to remember
- Different technologies
  - Suppose to be better (faster, more reliable, cheaper, …)

Direction and Principles



Sockets

- process sends/receives messages to/from its socket
- socket analogous to mailbox
  - sending process relies on transport infrastructure which brings message to socket at receiving process

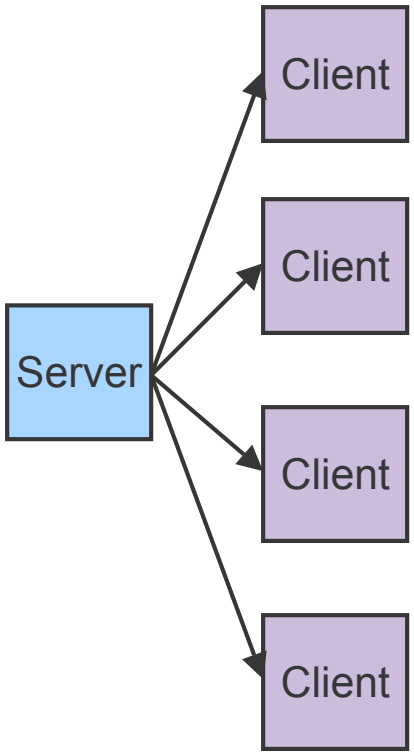


Network Programming with Sockets

- Reading:
  - Stevens 2nd or 3rd ed., Ch. 1-6 or 1st ed., Ch. 1-3, 6

- Sockets API:
  - A transport layer service interface
    - Introduced in 1981 by BSD 4.1
    - Implemented as library and/or system calls
    - Similar interfaces to TCP and UDP
    - Can also serve as interface to IP (for super-user); known as “raw sockets”

Client-Server Model



- Asymmetric Communication
  - Client sends requests
  - Server sends replies
- Server/Daemon
  - Well-known name (e.g., IP address + port)
  - Waits for contact
  - Processes requests, sends replies
- Client
  - Initiates contact
  - Waits for response

Client-Server Communication Model

- Service Model
  - Concurrent:
    - Server processes multiple clients' requests simultaneously
  - Sequential:
    - Server processes only one client's requests at a time
  - Hybrid:
    - Server maintains multiple connections, but processes responses sequentially
- Client and server categories are not disjoint
  - A server can be a client of another server
  - A server can be a client of its own client

```
int main(int argc, char **argv) {
  int sockfd = socket(PF_INET, SOCK_STREAM, 0);
  struct hostent * he = gethostbyname(argv[1]);
  struct sockaddr_in their_addr;
  their_addr.sin_addr = *((struct in_addr *)he->h_addr);
  their_addr.sin_port = htons(atoi(argv[2]));
  their_addr.sin_family = AF_INET;
  memset(their_addr.sin_zero, '\0', sizeof their_addr.sin_zero);
  connect(sockfd, (struct sockaddr *) their_addr, sizeof their_addr);
  write(sockfd, "GET /\r\n", strlen("GET /\r\n"));
  while (1) {
    char buf[100];
    int len = read(sockfd, buf, 100);
    if (len <= 0) {
       close(sockfd);
         return 0;
    } else {
       write(1, buf, len);
    }
  }
}
```


Sockets

```
int sockfd = socket(PF_INET, SOCK_STREAM, 0);
```

- Creates a new socket
- `SOCK_STREAM` = TCP
- `SOCK_DGRAM` = UDP

TCP Connections

- Transmission Control Protocol (TCP) Service
  - OSI Transport Layer
  - Service Model
    - Reliable byte stream (interpreted by application)
    - 16-bit port space allows multiple connections on a single host
    - Connection-oriented
      - Set up connection before communicating
      - Tear down connection when done

TCP Service

- Reliable Data Transfer
  - Guarantees delivery of all data
  - Exactly once if no catastrophic failures
- Sequenced Data Transfer
  - Guarantees in-order delivery of data
  - If A sends M1 followed by M2 to B, B never receives M2 before M1
- Regulated Data Flow
  - Monitors network and adjusts transmission appropriately
  - Prevents senders from wasting bandwidth
  - Reduces global congestion problems
- Data Transmission
  - Full-Duplex byte stream

UDP Services

- User Datagram Protocol Service
  - OSI Transport Layer
  - Provides a thin layer over IP
  - 16-bit port space (distinct from TCP ports) allows multiple recipients on a single host

UDP Services

- Unit of Transfer
  - Datagram (variable length packet)
- Unreliable
  - No guaranteed delivery
  - Drops packets silently
- Unordered
  - No guarantee of maintained order of delivery
- Unlimited Transmission
  - No flow control

Addresses

```
struct hostent * he = gethostbyname(argv[1]);
struct sockaddr_in their_addr;
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
```

- Look up destination host name

Addresses and Data

- Internet domain names
  - Human readable
  - Variable length
  - Ex: **`sal.cs.uiuc.edu`**
- IP addresses
  - Easily handled by routers/computers
  - Fixed length
  - Somewhat geographical
  - Ex: **`128.174.252.217`**

Address Access/Conversion Functions

- All binary values are network byte ordered

```
struct hostent* gethostbyname (const char*
   hostname);
```

  - Translate English host name to IP address (uses DNS)

```
struct hostent* gethostbyaddr (const char*
   addr, size_t len, int family);
```

  - Translate IP address to English host name (not secure)

```
char* inet_ntoa (struct in_addr inaddr);
```

  - Translate IP address to ASCII dotted-decimal notation (e.g., "128.32.36.37")

Structure: hostent

- The **`hostent`** data structure (from **`/usr/include/netdb.`**`h`)
  - canonical domain name and aliases
  - list of addresses associated with machine
  - also address type and length information

```
struct hostent {
    char* h_name;     /* official name of host */
    char** h_aliases;         /* NULL-terminated alias list */
    int h_addrtype    /* address type (AF_INET) */
    int h_length;     /* length of addresses (4B) */
    char** h_addr_list; /* NULL-terminated address list */
#define h_addr h_addr_list[0];/* backward-compatibility */
};
```


Choose port

```
their_addr.sin_port = htons(atoi(argv[2]));
```

- Select a destination port
- Convert byte order

Byte Ordering

- Big Endian vs. Little Endian
  - Little Endian (Intel, DEC):
    - Least significant byte of word is stored in the lowest memory address
  - Big Endian (Sun, SGI, HP):
    - Most significant byte of word is stored in the lowest memory address
  - Network Byte Order = Big Endian
    - Allows both sides to communicate
    - Must be used for some data (i.e. IP Addresses)
    - Good form for all binary data

Byte Ordering Functions

- 16- and 32-bit conversion functions (for platform independence)
- Examples:

```
int m, n;
short int s,t;

m = ntohl (n)   net-to-host long (32-bit) translation
s = ntohs (t)   net-to-host short (16-bit) translation
n = htonl (m)   host-to-net long (32-bit) translation
t = htons (s)   host-to-net short (16-bit) translation
```


Socket address

```
their_addr.sin_family = AF_INET;
memset(their_addr.sin_zero, '\0', sizeof
        their_addr.sin_zero);
```

- Fill in a few fields in the socket address structure
- "Magic"

Socket Address Structure

- IP address:

```
struct in_addr {
    in_addr_t s_addr;        /* 32-bit IP address */
};
```

- TCP or UDP address:

```
struct sockaddr_in {
    short sin_family;        /* e.g., AF_INET */
    ushort sin_port;         /* TCP/UDP port */
    struct in_addr;          /* IP address */
};
```

- all but sin_family in network byte order

Connecting the socket

```
connect(sockfd, (struct sockaddr *) their_addr, sizeof
                        their_addr);
```

- Actually creates a TCP connection, contacting a server
- Afterwards, use file descriptor like a regular file - read, write, close

Server

```
int main(int argc, char **argv) {
  int sockfd = socket(PF_INET, SOCK_STREAM, 0);
  struct sockaddr_in my_addr;
 my_addr.sin_addr = INADDR_ANY;
 my_addr.sin_port = htons(atoi(argv[1]));
 my_addr.sin_family = AF_INET;
  memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero);
  bind(sockfd, (struct sockaddr *) my_addr, sizeof my_addr);
  listen(sockfd, 10);
  while (1) {
    struct sockaddr_in their_addr;
    int sin_size;
    int newfd = accept(sockfd, (struct sockaddr *) & their_addr,
    &sin_size);
    write(newfd, "Hello world!\n", 14);
    close(newfd);
} }
```


Address specification

```
my_addr.sin_addr = INADDR_ANY;
```

- Accept connections from any address
- Can also select a specific interface here

bind

```
bind(sockfd, (struct sockaddr *) my_addr, sizeof my_addr);
```

- Connects a socket with a well-known *incoming* port on a local system

Listen

```
listen(sockfd, 10);
```

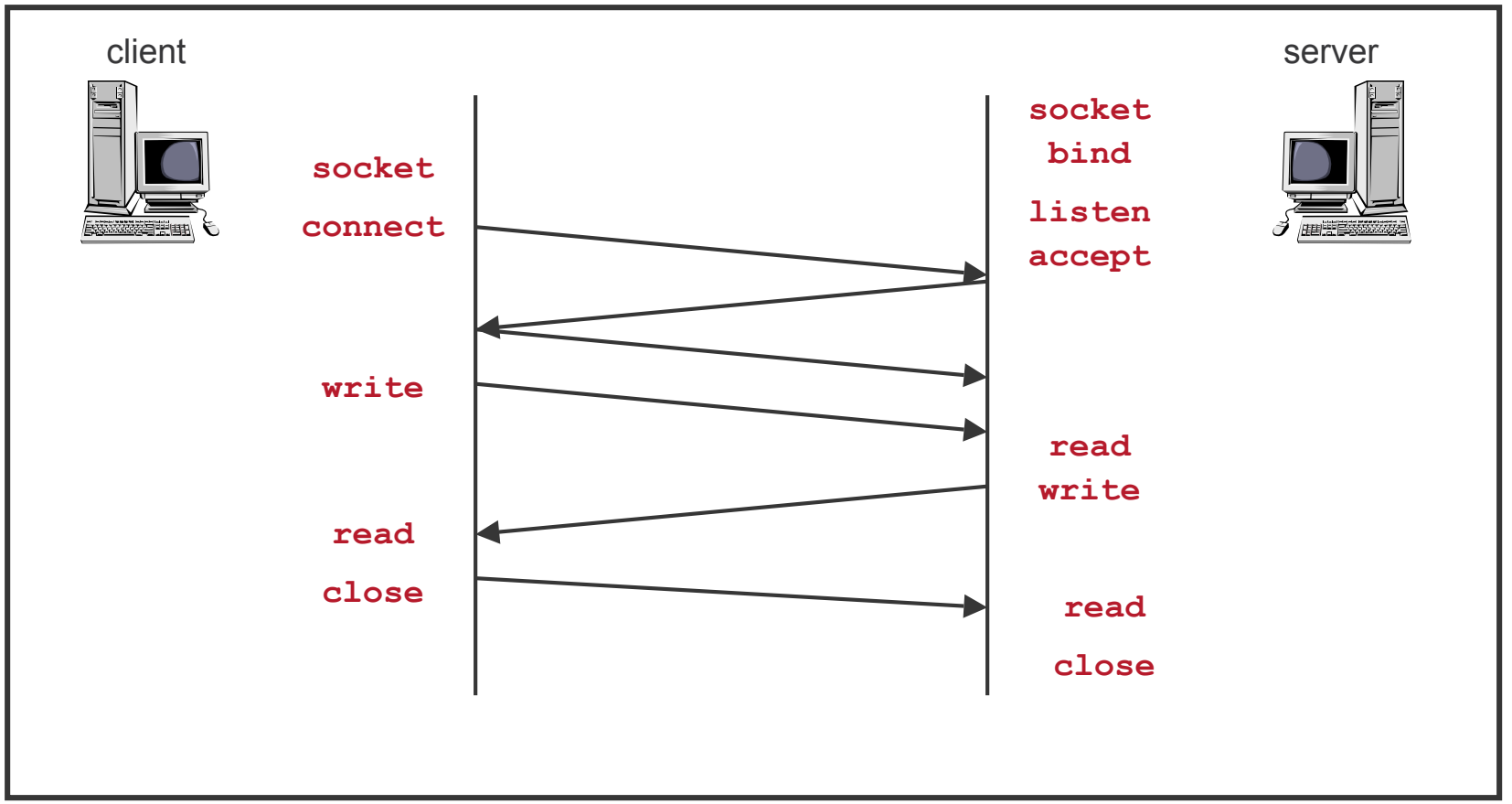
- Start accepting connections
- 10 is the amount of backlog connections
  - If more than 10 clients waiting, computer will refuse new connections

Accept

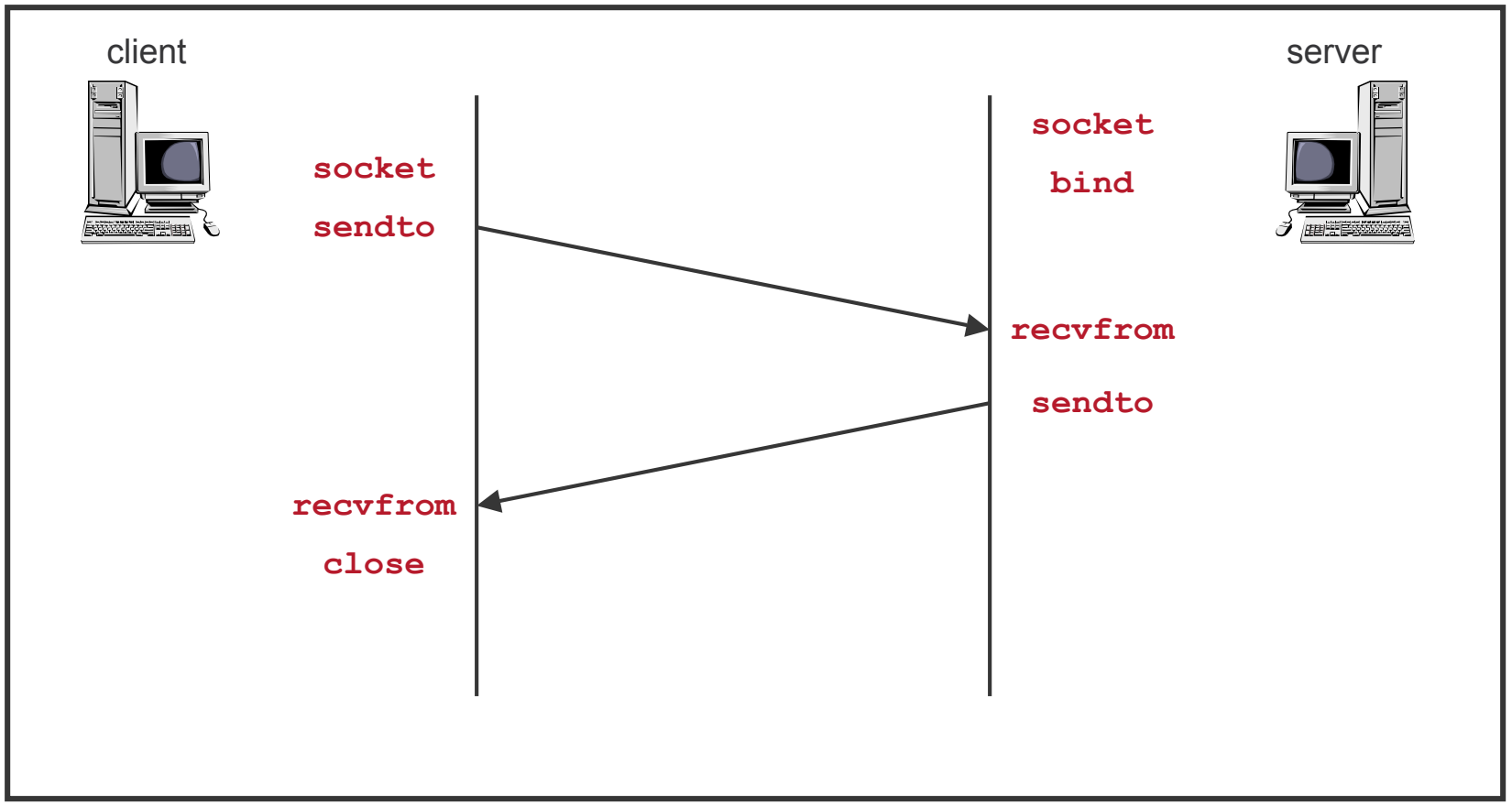
```
int newfd = accept(sockfd, (struct sockaddr *) & their_addr,
                   &sin_size);
```

- Receive a new connection
  - their_addr contains the address of the remote host
- newfd is the socket descriptor for communications
  - Used like a regular socket, can read write close

TCP Connection Example



UDP Connection Example



Functions: sendto

```
int sendto (int sockfd, char* buf, size_t nbytes,
    int flags, struct sockaddr* destaddr, int
    addrlen);
```

- Send a datagram to another UDP socket.
  - Returns number of bytes written or -1. Also sets **`errno`** on failure.
  - **`sockfd`**: socket file descriptor (returned from **socke**t)
  - **`buf`**: data buffer
  - **`nbytes`**: number of bytes to try to read
  - **`flags`**: see man page for details; typically use 0
  - **`destaddr`**: IP address and port number of destination socket
  - **`addrlen`**: length of address structure
    - **`= sizeof (struct sockaddr_in)`**

Functions: recvfrom

```
int recvfrom (int sockfd, char* buf, size_t
   nbytes, int flags, struct sockaddr* srcaddr,
   int* addrlen);
```

- Read a datagram from a UDP socket.
  - Returns number of bytes read (0 is valid) or -1. Also sets **`errno`** on failure.
  - **`sockfd`**: socket file descriptor (returned from **`socket`**)
  - **`buf`**: data buffer
  - **`nbytes`**: number of bytes to try to read
  - **`flags`**: see man page for details; typically use 0
  - **`srcaddr`**: IP address and port number of sending socket (returned from call)
  - **`addrlen`**: length of address structure = pointer to **`int`** set to **`sizeof (struct sockaddr_in)`**

TCP and UDP Ports

- Allocated and assigned by the Internet Assigned Numbers Authority
  - see **`RFC 1700`** or

`ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers`

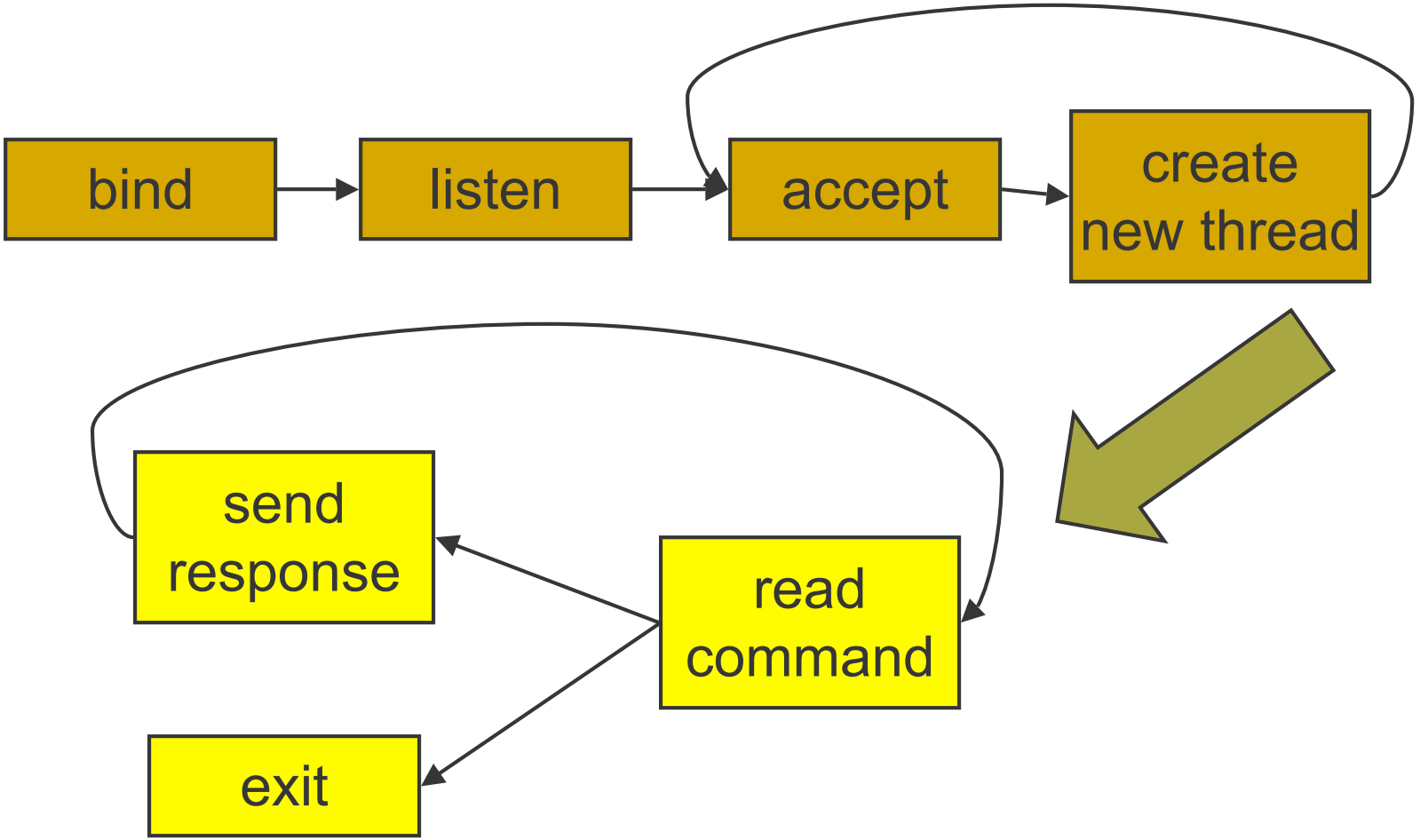
1-512	▪ standard services (see **`/etc/services`**) ▪ super-user only
513-1023	▪ registered and controlled, also used for identity verification ▪ super-user only
1024-49151	▪ registered services/ephemeral ports
49152-65535	▪ private/ephemeral ports

Review

- Find the bug in this code

```
host = gethostbyname("www.cs.uiuc.edu");
if (host == NULL)
      return -1;
addr.sin_family = AF_INET;
addr.sin_port = 80;
addr.sin_addr =
     *((struct in_addr *) host->h_addr);
```


Concurrent Server Flowchart



Fork Concurrent Server

```
while (1) {
   fd = accept(...);
   if (fork() == 0) {
      handle_connection(fd);
      exit();
   }
}

void handle_connection(int fd) {
   while (1) {
     read(fd, request);
     write(fd, response);
   }
}
```


Fork call

- `pid_t fork(void);`
- Splits process into two, parent and child
- Both processes are exact copies of each other
- Returns PID of child process to the parent and 0 to the child

Threaded server

```
while (1) {
   pthread_t thread;
   pthread_attr_t attr;
   fd = accept(...);
   pthread_create(&thread, &attr, handle_connection, (void *) fd);
}

void handle_connection(int fd) {
   while (1) {
     read(fd, request);
     write(fd, response);
   }
}
```


Posix Threads (pthreads)

- When coding
  - Include **`<pthread.h>`** first in all source files
- When compiling
  - Use compiler flag **`-D_REENTRANT`**
- When linking
  - Link library **`-lpthread`**

pthreads Attributes

- Attributes
  - Data structure `pthread_attr_t`
  - Set of choices for a thread
  - Passed in thread creation routine
- Choices
  - Inherit scheduling specifics from parent thread?
  - Compete at process or thread level?
  - Scheduling policy
  - Thread priority
  - Detached state
    - Join operation required at termination?
    - Only useful if thread returns a value

pthreads Attributes

- Initialize an attributes structure to the default values
  - `int pthread_attr_init (pthread_attr_t* attr);`
- Set the detached state value in an attributes structure
  - `int pthread_attr_setdetachedstate (pthread_attr_t* attr, int value);`
  - Value
    - **`PTHREAD_CREATE_DETACHED`**
    - **`PTHREAD_CREATE_JOINABLE`**

pthread Error Handling

- pthreads functions do not follow the usual Unix conventions
  - Similarity:
    - Returns 0 on success
  - Differences:
    - Return error code on failure
    - Does not set **`errno`**
  - What about **`errno`**?
    - Each thread has its own
    - Define **`_REENTRANT`** (**`-D_REENTRANT`** switch to compiler) when using pthreads

pthread Creation

```
int pthread_create (pthread_t* tid, pthread_attr_t*
    attr, void*(child_main), void* arg);
```

- Spawn a new posix thread
- Parameters:
  - **`tid`**:
    - Unique thread identifier returned from call
  - **`attr`**:
    - Attributes structure used to define new thread
    - Use **`NULL`** for default values
  - **`child_main`**:
    - Main routine for child thread
    - Takes a pointer **`(void*)`**, returns a pointer **`(void*)`**
  - **`arg`**:
    - Argument passed to child thread

pthread Cleanup

- Problem
  - A thread can be terminated by an external source (i.e. by other threads)
  - Deallocation and other cleanup must still be performed
- Solution
  - Push and pop cleanup routines
  - Defer external cancellation until after performing cleanup to avoid race conditions

pthread Cleanup

```
{
    pthread_cleanup_push (my_cleanup, heap_data);
    …
    {
        …
    }
    pthread_setcancel_type(PTHREAD_CANCEL_DEFERRED,
    &old);
    pthread_cleanup_pop(1);
}
```


pthreads vs. fork

- fork creates entire new process, with a copy of all variables
- thread creates a new thread of execution, but uses a single copy of data
- Why use one vs. the other?
  - fork: don't have to worry about concurrency
  - thread: can have easy data sharing and communication between threads
  - thread: (much) lighter weight

Multithreading-Safe Functions

- Functions in original libraries
  - Use static storage
  - Not compatible with multiple threads
- Reentrant versions
  - Dubbed **<name>_r**
  - Pass state back and forth to routine
  - Example:
    - Tokenizing a string using state local to the current thread

```
char* strtok_r (char* s, const char* delim,
   char** last_p);
```


Example: getpwuid

- `static passwd *getpwuid(uid_t uid);`
- Returns a pointer to static password buffer
- Will conflict if two threads call getpwuid simultaneously

getpwuid_r

- ```
  int getpwuid_r(uid_t uid,
    struct passwd *pwbuf,
    char *buf, size_t buflen,
    struct passwd **pwbufp);
  ```
- pwbuf used to store result
- buf is used to store pointers from pwbuf
- buflen - size of the buffer
- pwbufp - pointer to pwbuf or NULL if failure
- return error number


  ```

Announcements

- Office hours scheduled
  - Nikita: 3-4pm Mondays and 10-11am Fridays
  - Steven: 3:30-5:30pm Tuesdays
  - Location for Steven's office hours TBA