CS 640 Introduction to Computer Networks

Lecture 2

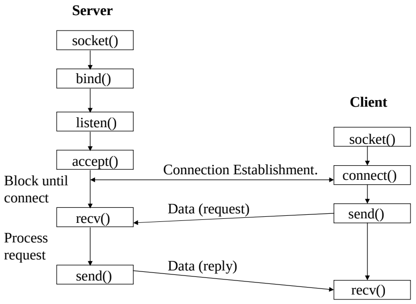
Today’s lecture

- Application programming interface (sockets)
- For the project
  - A mini-introduction to IP (Internet protocol)
  - Details on project

Berkeley Sockets

- Networking protocols are implemented as part of the OS
  - The networking API exported by most OS’s is the *socket interface*
  - Originally provided by BSD 4.1c ~1982.
- The principal abstraction is a socket
  - Point where an application attaches to the network
  - Operations: creating connections, attaching to network, sending/receiving data, closing.

Connection-oriented example (TCP)



Server

socket()

bind()

listen()

accept()

Block until connect

recv()

Process request

send()

Client

socket()

connect()

send()

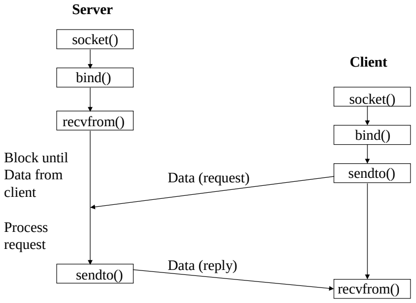
recv()

Connection Establishment.

Data (request)

Data (reply)

Connectionless example (UDP)



Server

socket()

bind()

recvfrom()

Block until Data from client

Process request

sendto()

Client

socket()

bind()

sendto()

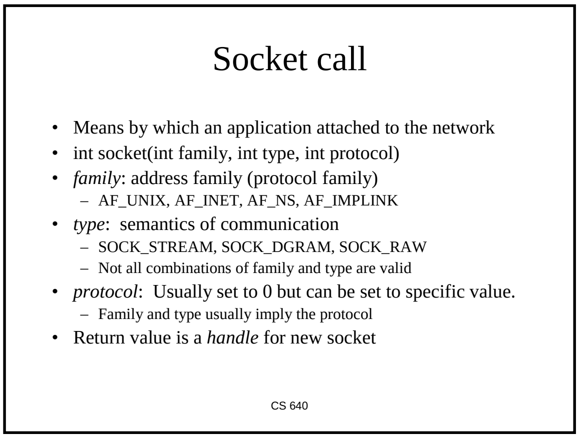
recvfrom()

Data (request)

Data (reply)

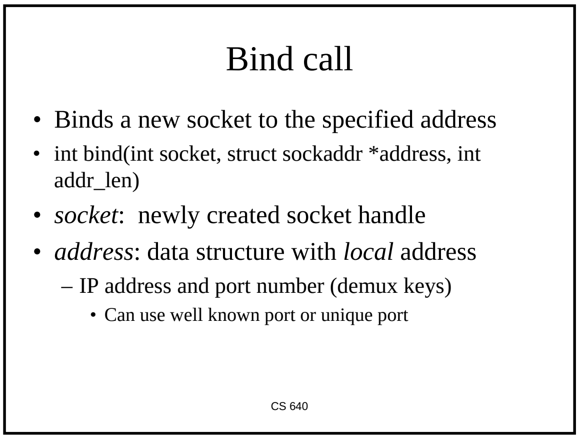
Ports (multiplexing)

- How does the OS know whether one wants to connect to the web server or the email server?
- How does the OS know which process to deliver the data to?
- 16 bit port numbers are used
  - Both source and destination have a port number
  - Servers have well known port numbers <1024
- How can the OS tell TCP packets from UDP?
  - Protocol number is part of IP header



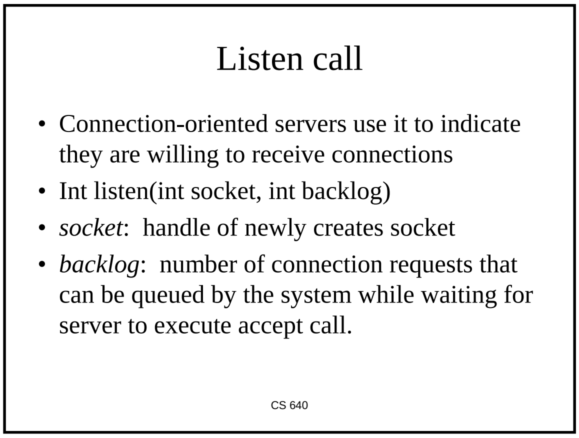
Socket call

- Means by which an application attached to the network
- int socket(int family, int type, int protocol)
- *family*: address family (protocol family)
  - AF_UNIX, AF_INET, AF_NS, AF_IMPLINK
- *type*: semantics of communication
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  - Not all combinations of family and type are valid
- *protocol*: Usually set to 0 but can be set to specific value.
  - Family and type usually imply the protocol
- Return value is a *handle* for new socket



Bind call

- Binds a new socket to the specified address
- int bind(int socket, struct sockaddr *address, int addr_len)
- *socket*: newly created socket handle
- *address*: data structure with *local* address
  - IP address and port number (demux keys)
    - Can use well known port or unique port



Listen call

- Connection-oriented servers use it to indicate they are willing to receive connections
- Int listen(int socket, int backlog)
- *socket*: handle of newly creates socket
- *backlog*: number of connection requests that can be queued by the system while waiting for server to execute accept call.

Accept call

- After *listen*, the accept call performs a *passive open* (server prepared to accept connects).
- int accept(int socket, struct sockaddr *address, int addr_len)
- It blocks until a remote client carries out a connection request
- When it does return, it returns with a *new* socket that corresponds with new connection and the address contains the clients address

Connect call

- Client executes an *active open* of a connection
- Int connect(int socket, struct sockaddr *address, int addr_len)
- Call does not return until the three-way TCP handshake is complete
- Address field has remote system’s address
- Client OS usually selects random, unused port

send(to), recv(from)

- After connection has been made, application uses send/recv to data
- int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- int recv(int scoket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer

IP addresses

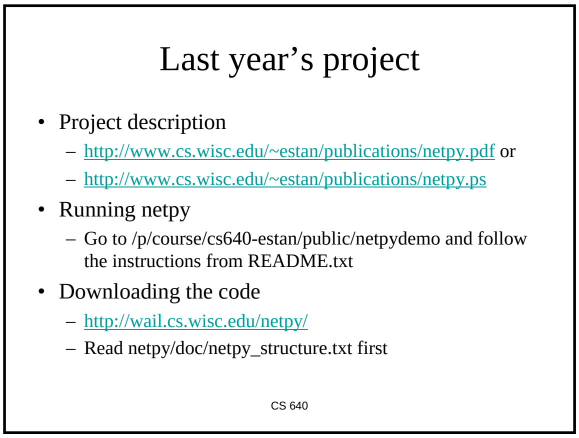
- IP address: 4byte-string that identifies a node
  - Usually unique (some exceptions)
  - Dotted decimal notation: 128.92.54.32
  - Structure: network part + host part (e.g. 3 bytes + 1 byte)
- IP prefix has IP addresses with same network part
  - Represented as network part / number of bits in net. part
    - Examples: 120.0.0.0/8 , 128.96.0.0/14
  - Hierarchical networks typically use prefix hierarchies
    - Example: university network (128.105.0.0/16) includes departmental network (128.105.167.0/24)

Domain Name System (DNS)

- A distributed database mapping human readable host names to IP addresses
  - Other mappings too: from IP addresses to host names, from domain names to mail servers, etc.
- DNS names have hierarchical structure:
  - www.cs.wisc.edu is host name
  - cs.wisc.edu is domain name for department
  - wisc.edu is domain name for university
  - edu is domain of U.S. educational institutions

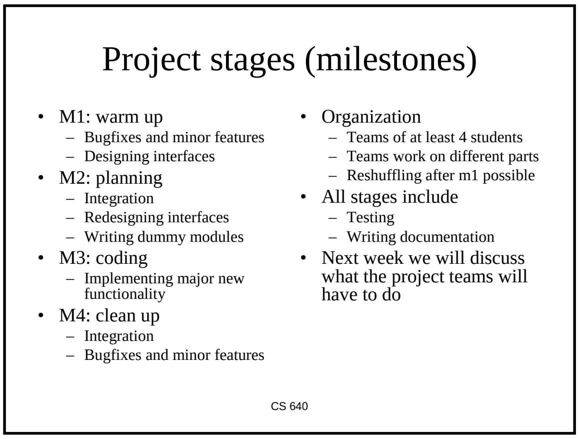
Software developers spend their time on

- Naïve view
  - 80% write code
  - 20% other things
- Reality is more like
  - 20% understand problem
  - 20% write code
  - 20% test and debug
  - 20% rewrite code
  - 10% document stuff
  - 10% other things



Last year's project

- Project description
  - http://www.cs.wisc.edu/~estan/publications/netpy.pdf or
  - http://www.cs.wisc.edu/~estan/publications/netpy.ps
- Running netpy
  - Go to /p/course/cs640-estan/public/netpydemo and follow the instructions from README.txt
- Downloading the code
  - http://wail.cs.wisc.edu/netpy/
  - Read netpy/doc/netpy_structure.txt first



Project stages (milestones)

- M1: warm up
  - Bugfixes and minor features
  - Designing interfaces
- M2: planning
  - Integration
  - Redesigning interfaces
  - Writing dummy modules
- M3: coding
  - Implementing major new functionality
- M4: clean up
  - Integration
  - Bugfixes and minor features
- Organization
  - Teams of at least 4 students
  - Teams work on different parts
  - Reshuffling after m1 possible
- All stages include
  - Testing
  - Writing documentation
- Next week we will discuss what the project teams will have to do