CS 4/55231
Internet Engineering

Kent State University
Dept. of Computer Science

LECT-2

Application Architecture

Class Mechanics

- Send email and get listed in class email list. Use "IN2004S" in the email subject field.
- Project group formation at the end of this class

INTERNET ENGINEERING

Topics to Cover

- **Network/Socket Programming.**
- **Server Client Design.**
- **Internet Architecture**
  - **LAN**
  - **WAN**
  - **SCALABILITY**
  - **INTERNETWORK**
- **INTERDOMAIN ROUTING**
  - **IGP, BGP**
  - **BGP Experiment in Internet Engineering Lab (GROUPS OF 2)**
- **INTERNET SYSTEMS**
  - **MAIL, FTP, DNS**
- **OPEN EMBEDDED INTERNET PROTOCOLS**
  - **HTTP, FIREWALL, PROXY, CACHE**
- **ADVANCED TOPICS**
  - **DIGITAL SIGNATURE, CERTIFICATES AND CERTIFICATE SERVERS.**

INTERNET ENGINEERING

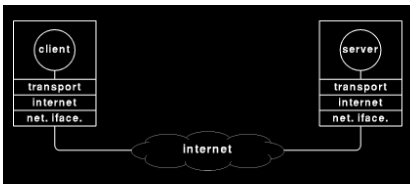
Today's Topic

How to Build a Network Application

We will build a
baby Server/Client today!

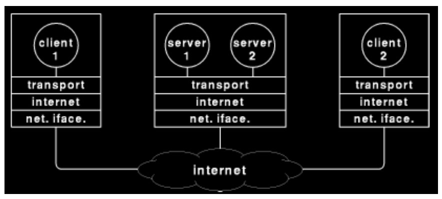
INTERNET ENGINEERING

Client Server Interaction

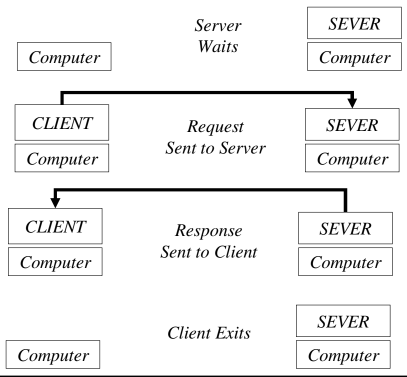


INTERNET ENGINEERING

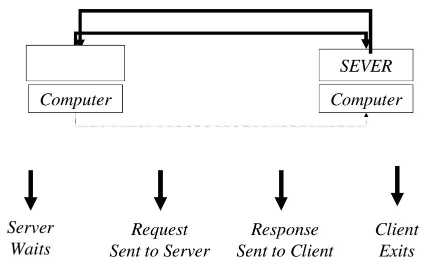
Multiple Services on One Computer



INTERNET ENGINEERING

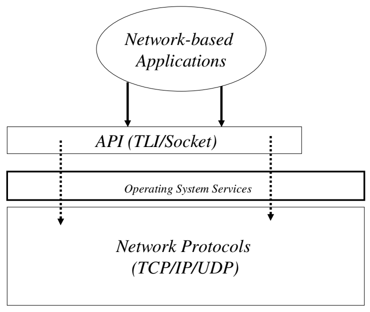


INTERNET ENGINEERING



INTERNET ENGINEERING

Application Platform


Network-based Applications
API (TLI/Socket)
Operating System Services
Network Protocols (TCP/IP/UDP)

INTERNET ENGINEERING

API to Network

Network APIs

- Transport Layer Interface (TLI) was developed in mid 1980s by AT&T with release 3 of system V UNIX. Later it became a part of Sun systems. Now this is available everywhere.
- Socket API is the original API developed by Berkeley UNIX group in late 70's and early 80s. Available on BSD UNIX Systems.
- WinSock is the API version provided for Microsoft Windows.

INTERNET ENGINEERING

What is Socket API?

- Socket Application Program Interface can be used to access a network protocol stack from any programming language.
- Socket API has been inspired by Unix *open-read-write-close* file access paradigm (original is Multics).
- However, accessing network is substantially more complex than file access.

INTERNET ENGINEERING

Sockets & Files

- An integer called file descriptor is returned, when a file is opened.
- In the same way a socket descriptor is returned when a socket is opened.
- A file descriptors binds to a file when it is open.
- But, a socket can be created without binding to a specific destination. Applications can choose when to bind
  - Datagram binds each time when it sends, therefore same socket can be used to send to many.
  - TCP binds once, and it remains, thus avoids repeated binding.

INTERNET ENGINEERING

Socket Creation & Closing

- **descriptor=socket(pfamily, type, protocol)**
  - **pfamily** =PF_INET| PF_APPLETALK | PF_UNIX| PF_PUP
  - type=SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, etc.
  - protocol=subtype of protocol family if any. Can be obtained by getprotobyname(), getprotobynumber() etc.
- Unix uses fork() and execv() to create and spawn new program. Child always inherits all parent sockets. UNIX maintains a count of owners.
- A process can close a socket by **close(socket)**

INTERNET ENGINEERING

Specifying a Local Address

- Initially a socket is created without any association to a local or remote address.
- For TCP/IP it means no protocol port number.
- Some application may not care (clients generally). Some do (all servers). The call:
- **bind(socket, localaddress, addrlen)**
  - socket is the socket descriptor returned by socket()
  - localaddress is a complex structure with several fields, and may vary for protocols.
  - For TCP/IP it contains both the port number and the IP address of the host is in it.
  - addrlen is the length of the address.

INTERNET ENGINEERING

Address Structures

- Barkley code defines a generic sockaddr structure to represent address of a connection end-point.

```
struct sockaddr {
    u_char sa_len;                /*total length of the address*/
    u_char sa_family;             /*family of address*/
    char sa_data[14];             /*the address itself*/
};
```

- Example: TCP/IP defines its own exact sockaddr_in:

```
struct sockaddr_in {
    u_char sin_len;               /*same as sa_len*/
    u_char sin_family;            /*same as sa_family*/
    u_short_sin_port;             /*protocol port number */
    struct in_addr sin_addr       /*4 bytes IP address of the host*/
    char sin_zero[8];             /*not used 6+8=14*/
};
```

Who supplies which part?

- Generally server calls bind to specify a server port number at which it will accept connection.
- A server on a multi-homed host can write down INADDR_ANY instead of the IP address to say it will accept the connection in any of the computers IP addresses.

INTERNET ENGINEERING

Listening for Connection

- listen creates a buffer for the pending connection requests from the remote clients.
- Listen(socket, queuelength)
  - socket is the descriptor that has been created and is bound to a local address.
  - queuelength specifies how many request can wait while server is busy with one.
  - OS maintains a separate request queue for each socket. It the queue is full, OS refuses new requests.

INTERNET ENGINEERING

Waiting for Accepting a Connection

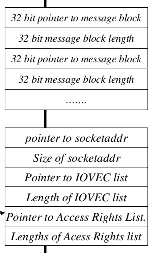
- After a server executes *socket-bind-listen*, it can go to sleep by calling accept. The Operating System will wake up the server when there is a request in the request queue.
- newsock=accept(socket, &addr, &addrlen)
  - socket is on which it was waiting,
  - addr is a pointer to a structure of type *sockaddr*, in which the address of the client will be returned by the OS. Addrlen is length of this address.
  - newsock is a new socket created by system which has its destination pre-connected to the client.
- The server can keep on communicating with the requesting client with the new socket, and close it when done. Meanwhile, the original socket remains intact to accept request from other clients.

Connecting to a Destination Address

- Initially a socket is created without any destination address. An client application program must call connect to establish connection.
- **connect(socket, destaddr, destaddrlen)**

Sending Data

- **write(socket, buffer, bytelength)**
- **writev(socket, iovector, vectorlen)**
  - iovector is an array of addresses and their lengths. The system gathers all the data.
- **send(socket, message, length, flags)**
  - flags allows to invoke special TCP features such as "URGENT" message, "do not use local routing table" etc.
- **sendto(socket, message, length, flags, daddr, daddrlen)**
- **sendmsg(socket,messagestruct,flags)**


32 bit pointer to message block
32 bit message block length
32 bit pointer to message block
32 bit message block length
....
pointer to socketaddr
Size of socketaddr
Pointer to IOVEC list
Length of IOVEC list
Pointer to Access Rights List.
Lengths of Acess Rights list

Receiving Data

- **read(descriptor, buffer, length)**
- **readv(descriptor, iovector, vectorlen)**
- **recv(socket, buffer, length, flags)**
- **recvfrom(socket,buffer,length,flags,fromaddr,addrlen)**
- **recvmsg(socket,messagestruct,flags)**

Obtaining Local & Peer Address

- **getpeername(socket, &destaddr,&addlen)**
- **getsockname(socket, &localaddr,&addlen)**

Obtaining & Setting Socket Options

- **Getsockopt(socket,level,optionid,&optionval,&length)**
- **Setsockopt(socket,level,optionid,optionval,length)**
- Example options are timeout parameters, allocated buffer space etc.

Handling Multiple Request

- **nready=select(ndesc, &indesc, &outdesc,&execdesc,timeout)**
  - a call to select will allow server to wait till one of the descriptor is ready.
  - ndesc=number of descriptors. System checks descriptors from 0 to ndesc-1. There are 3 bit masks to wait on a selected subset.
  - Indesc, outdesc and exedesc are bit masks that identifies input, output, and exception event in which sockets to check.
  - Timeout says how long to wait. 0 means wait indefinitely.
  - It returns the number of ready descriptors and also updates the masks to reflect which of the selected sockets are ready.
  - To use an application must create multiple sockets and then call select(). Once, a socket becomes ready OS wakes the process.

Obtaining & Setting Hostnames

- **gethostname(name,length)**
- **sethostname(name,length) ***

Obtaining & Setting Domain Names

- **setdomainname(name,length)**
- **getdomainname(name, length) ***

INTERNET ENGINEERING

Network Byte Order

- **Localshort=ntohs(netshort)**
- **Locallong=ntohl(netlong)**
- **Netshort=htons(localshort)**
- **Netlong=htonl(locallong)**

INTERNET ENGINEERING

IP Address Manipulation

String to 32 bit network byteordered address

- **255.255.255.0 vs. xffff ffff**
- **Address=inet_addr(string)**
- **Address=inet_network(string)**
  - address is a 32 bit IP address in network byte order.
  - string is an ASCII stirng with IP in dotted decimal notation
  - inet_network returns o for host part.
- **Str=inet_ntoa(internetaddr)**
- **Internetaddr=inet_makeaddr(net,local)**
- **Net=inet_netof(internetaddr)**
- **Local=inet_lnaof(internetaddr)**

INTERNET ENGINEERING

Accessing Domain Name System

- Each computer now also have a symbolic domain name.
  - Such as www.kent.edu or shimana.facnet.mcs.kent.edu
- A set of designated computers (knows as DNS servers) scattered across the internet maintains the mapping of DNS to the actual IP.
- Translation from domain name to IP address or the opposite, requires communication with these servers.

INTERNET ENGINEERING

Accessing Domain Name System

- **res_init()** — *Initialize DNS comm.*
- **res_mkquery(op,dname,class,type,**
  data, datalen,newrr, buffer,buflen) — *Form Query.*
- **res-send(buffer,buflen,answer,anslen)** — *Send Query.*

- **dn_expand(msg,eom,compressed,full,fullen)**
- **dn_comp(full,compressed,cmprlen,prevptr,lastptr)**

Conversion between ASCII name and compressed domain name format.

- **ptr=gethostbyname(namestr)**
- **ptr=gethostbyaddr(add,len,type)**

Takes a domain name and returns a structure with information about the domain.

INTERNET ENGINEERING

Obtaining Information about Network Services

- WHOIS a special server service, which allows a client in one machine to obtain information about user who has account in server machine. It runs on Port 43.
  - **Ptr=getservebyrname(name, proto)**
- Name is the address of a desired service, and proto is usually TCP or UDP. It returns a structure which contains name of the service, a list of aliases, protocol identifier for this service, and an integer protocol port number.
  - **Ptr=getservbyport(port,proto)**

INTERNET ENGINEERING

Obtaining information about Network & Protocol

INTERNET ENGINEERING

- **Ptr=getnetbyname(namestr)**
- **Ptr=getnetbyaddr(netaddr, addrtype)**

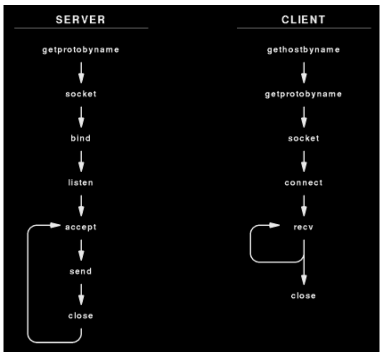
Namestr is the name of the network in ASCII, ptr is a data structure which contain 32 bit IP address and other information about the net.

- Each protocol has official name, number and registered aliases. These routines can be used to obtain complete information from name or port number of it.
- **Ptr=getprotobyname(name)**
- **Ptr=getprotobynumber(number)**

An Example Service

INTERNET ENGINEERING

- A client connect to a server, and waits for output.
- Server returns the count of the times it has been contacted by any client.
- Upon receiving the data the client prints it to screen.
- Command line arguments:
  - client <hostname> <portname>
  - server <portnumber>
  - hostname and portnumbers are optional.
  - Default host is localhost
  - Default port is 5193.
- Output on client machine:
  - This server has been contacted 10 times.



INTERNET ENGINEERING

INTERNET ENGINEERING

server.c

Click Here

http://www.animasters.com/menu/vrml/estuary/estuaryintro.html

INTERNET ENGINEERING

client.c

Click Here

http://www.animasters.com/menu/vrml/estuary/estuaryintro.html

More References

INTERNET ENGINEERING

- *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*, Prentice Hall, 1999.
- *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*, Prentice Hall, 1998.
- More example programs & source codes
  - http://www.kohala.com/start/unpv12e.html