CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Sockets

Today's Topics

- TCP Connections
- More details about the inet domain

Fancier UDP Application

- Let's make our application a little bit fancier
- We want it to stop if we don't receive a datagram after 1 minute
- We also want it to stop if anything is written in stdin
- The problem: Having a read/recv call hanging from several descriptors at once
- Solution: New kernel call!

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

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

void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

Fancy UDP Server I

```
#include <sys/select.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
void printsin(s_in, s1, s2)
struct sockaddr_in *s_in; char *s1, *s2;
{
   printf ("Program: %s\n%s ", s1, s2);
   printf ("(%d,%d)\n", s_in->sin_addr.s_addr, s_in->sin_port);
}

main() {
   int socket_fd, cc, h_len, fsize, namelen, hits;
   fd_set mask;
   struct timeval timeout;
   struct sockaddr_in s_in, from;

   struct { char head; u_long body; char tail; } msg;
```

Fancy UDP Server II

```
socket_fd = socket(AF_INET, SOCK_DGRAM, 0);

bzero((char *) &s_in, sizeof(s_in));
s_in.sin_family = (short) AF_INET;
s_in.sin_addr.s_addr = htons(INADDR_ANY);
s_in.sin_port = htons((u_short)0x3333);
bind(socket_fd, (struct sockaddr *)&s_in, sizeof(s_in));

for(;;) {
  fsize = sizeof(from);
  FD_ZERO(&mask); FD_SET(0, &mask); FD_SET(socket_fd, &mask);
  timeout.tv_sec = 60; timeout.tv_usec = 0;

  if ((hits = select(socket_fd+1, &mask, (fd_set *)0,
            (fd_set *)0, &timeout)) < 0) {
    perror("recv_udp:select"); exit(1);
  }

  if( (hits == 0) || ((hits > 0) && (FD_ISSET(0, &mask)))) {
    printf("Shutting␣down\n"); exit(0);
  }

  cc = recvfrom(socket_fd, &msg, sizeof(msg), 0,
          (struct sockaddr *)&from, &fsize);
```

Fancy UDP Server III

```
        printsin(&from, "recv_udp: ", "Packet from:");
        printf("Got data ::%c%ld%c\n",msg.head,
          ntohl(msg.body),msg.tail);
        fflush(stdout);
    }
}
```

Internet Virtual Circuits (TCP/IP)

- We want to extend the pipe abstraction onto the Internet
- This means a reliable byte stream with no visible packets/messages!
- Also implies a point to point connection. In the entire Internet, the tuple
  - ((host1, port1), (host2, port2))

  is unique
- It should be noted that the cost for setting up, maintaining and tearing down the connection has a non-trivial cost
- TCP/IP imples a client-server model!

Making TCP/IP Connections!

- New kernel calls for TCP/IP
  - To mark a socket as being capable of accepting TCP/IP connections, the server calls:

```
#include <sys/socket.h>
int listen(int s, int backlog);
```

  - To mark a socket as being capable of accepting TCP/IP connections, the server calls:

```
#include <sys/socket.h>
int listen(int s, int backlog);
```

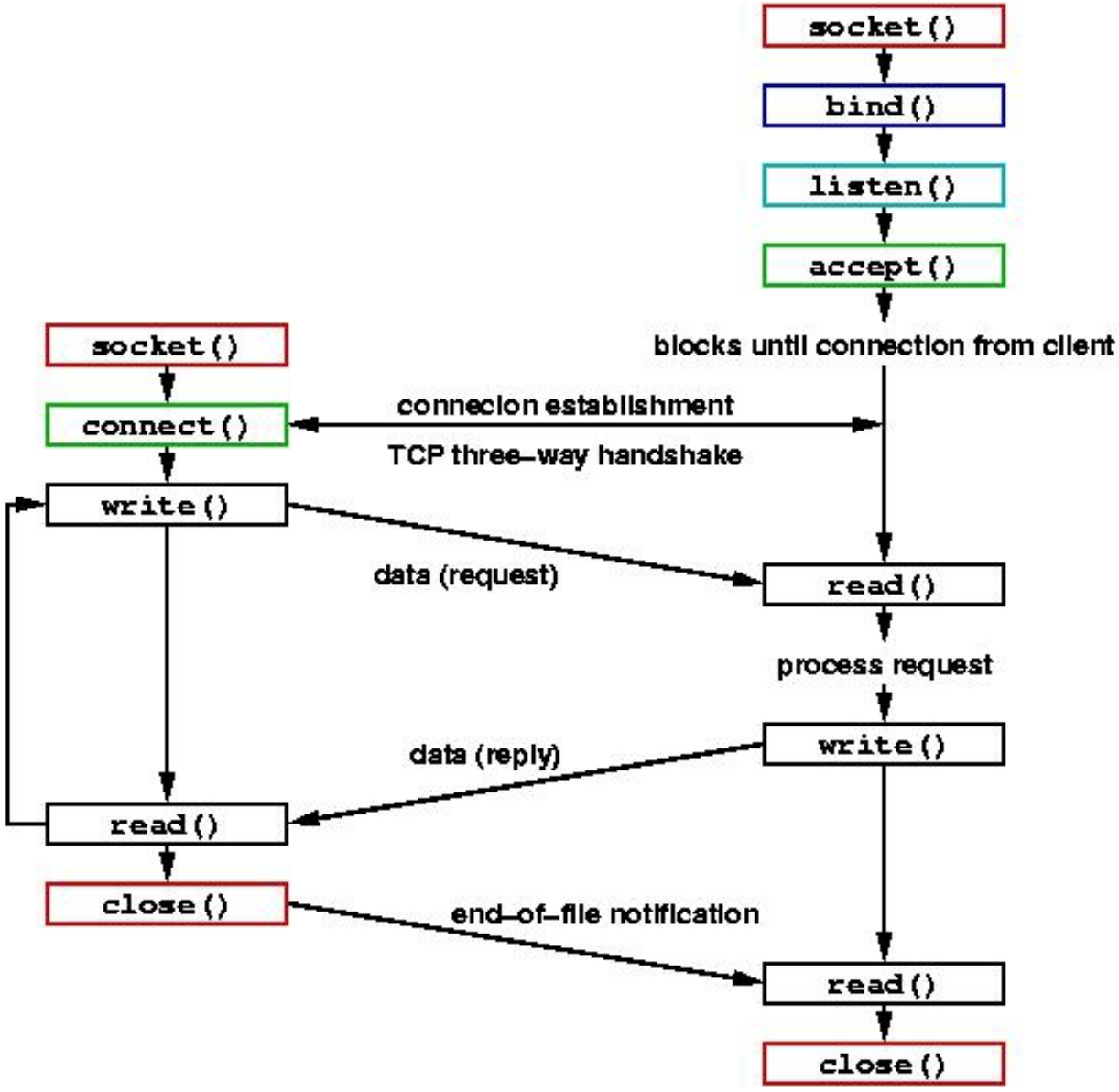
  - To accept a TCP/IP connection on a listening socket, server can then do:

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, int *addrlen);
```

  - To initiate a connection to a server, client does:

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr,
        int addrlen );
```

  - Then we can use read and write to pass data over the circuit!


socket()
bind()
listen()
accept()
blocks until connection from client
socket()
connect()
connecion establishment
TCP three-way handshake
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
end-of-file notification
read()
close()

TCP/IP Server I

```
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
#include <netdb.h>
#include <stdlib.h>
#include <stdio.h>
void printsin(s_in, s1, s2)
struct sockaddr_in *s_in; char *s1, *s2;
{
   printf ("Program: %s\n%s ", s1, s2);
   printf ("(%d,%d)\n", s_in->sin_addr.s_addr, s_in->sin_port);
}
main() {
   int listener, conn, length; char ch;
   struct sockaddr_in s1, s2;

   listener = socket( AF_INET, SOCK_STREAM, 0 );

   bzero((char *) &s1, sizeof(s1));
   s1.sin_family = (short) AF_INET;
   s1.sin_addr.s_addr = htonl(INADDR_ANY);
```

TCP/IP Server II

```
    s1.sin_port = htons(0); /* bind() will gimme unique port. */
    bind(listener, (struct sockaddr *)&s1, sizeof(s1));
    length = sizeof(s1);
    getsockname(listener, (struct sockaddr *)&s1, &length);
    printf("RSTREAM:: assigned port number %d\n",
         ntohs(s1.sin_port));

    listen(listener,1);
    length = sizeof(s2);
    conn=accept(listener, (struct sockaddr *)&s2, &length);
    printsin(&s2,"RSTREAM::", "accepted connection from");
    printf("\n\nRSTREAM:: data from stream:\n");
    while ( read(conn, &ch, 1) == 1) putchar(ch);
    putchar('\n');
}
```

TCP/IP Client I

```
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
#include <netdb.h>
#include <stdlib.h>
char msg[] = { "the quick brown fox jumps over the lazy dog" };
main(int argc, char **argv) {
  char *remhost; u_short remport;
  int sock, left, num, put;
  struct sockaddr_in remote;
  struct hostent *h;

  remhost = argv[1]; remport = atoi(argv[2]);

  sock = socket(AF_INET, SOCK_STREAM, 0);

  bzero((char *) &remote, sizeof(remote));
  remote.sin_family = (short) AF_INET;
  h = gethostbyname(remhost);
  bcopy((char *)h->h_addr, (char *)&remote.sin_addr,
      h->h_length);
```

TCP/IP Client II

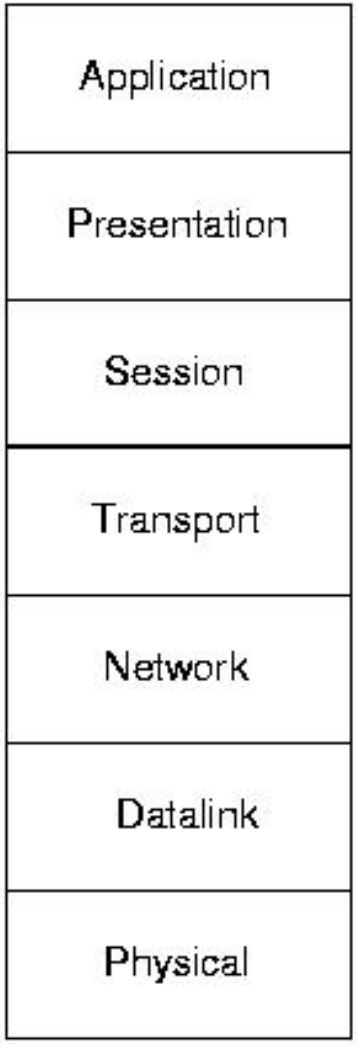
```
    remote.sin_port = htons(remport);
    connect(sock, (struct sockaddr *)&remote, sizeof(remote));

    left = sizeof(msg); put = 0;
    while(left > 0) {
      if((num = write(sock, msg+put, left)) < 0) {
        perror("inet_wstream: write");
        exit(1);
      }
      else {
        left -= num;
        put += num;
      }
    }
}
```

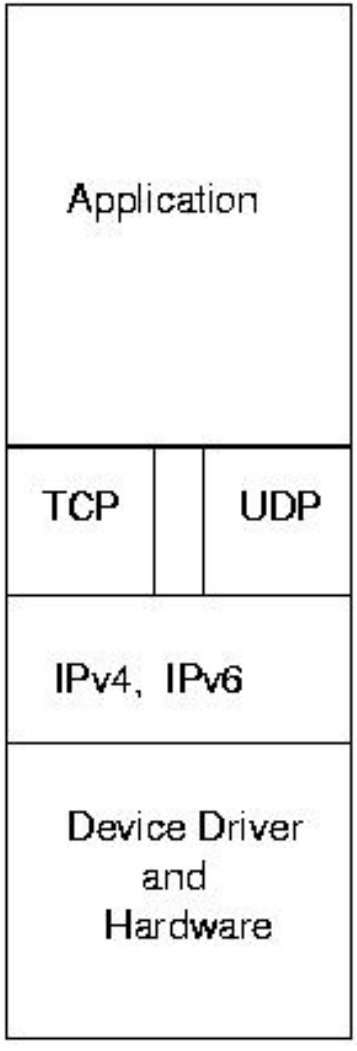
A Typical TCP/IP Client/Server Server

```
. . .
listener = socket(...);
. . .
bind(listener, ...);
listen(listener, ...);
...
while (1) {
  new = accept(listener, ...);
  if (fork() == 0) {
    close(listener);
    /* read lots of stuff from new */
  }
  close(new);
  while(wait3(status, WNOHANG, NULL)); \
  /* Can also handle SIGCHLD */
}
```

OSI Reference Model



OSI REFERENCE MODEL

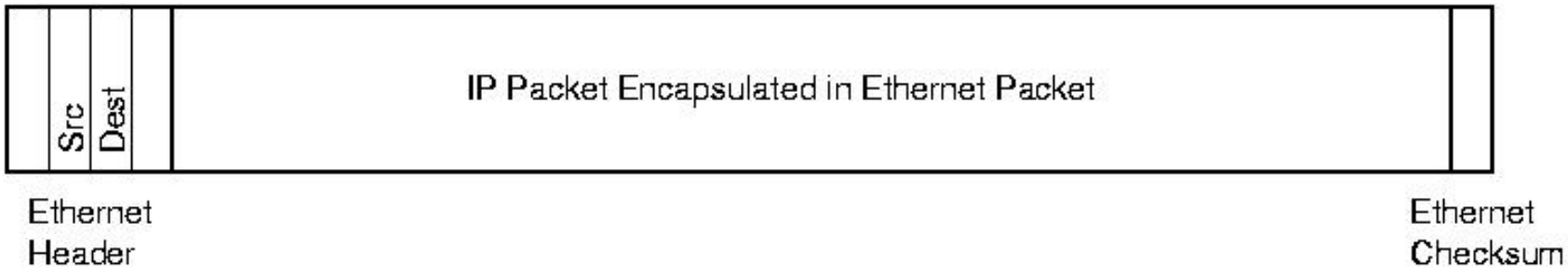


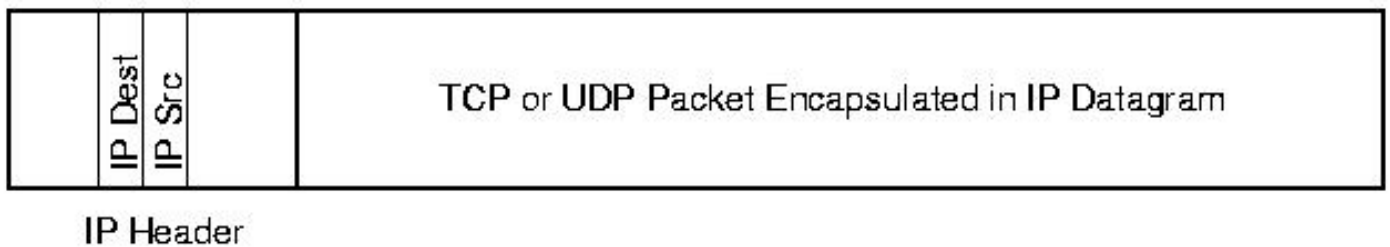
INTERNET PROTOCOL SUITE

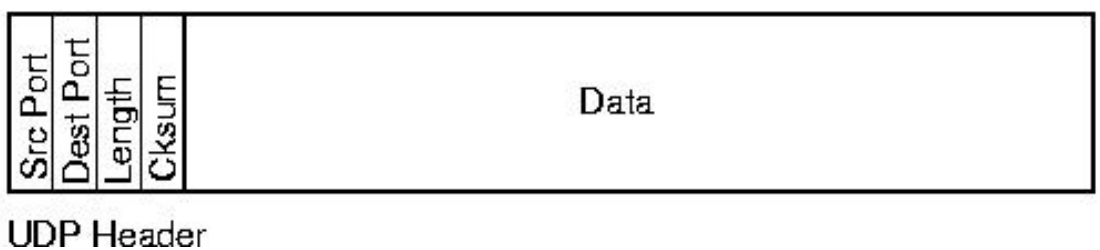
user process

kernel

Packet Encapsulation







Ethernet

- IEEE Standard 802: CSMA/CD, token bus, token ring
- IEEE Standard 802.3: CSMA/CD
  - Ethernet is implementation of 802.3
- Typically 6 byte address: 00:62:97:B8:C9:4A
- ARP: protocol for mapping IPv4 address the hardware address
- RARP: protocol for mapping hardware address to IPv4 address
- Ethernet address outermost
- Routing protocols determine correct IP address on a hop-by-hop basis
- ARP maps IP address to ethernet address to transmit message across next hop
- man arp and man traceroute