Internet Sockets

Beej's Guide to Network Programming
(http://www.ecst.csuchico.edu/~beej/guide/net/)

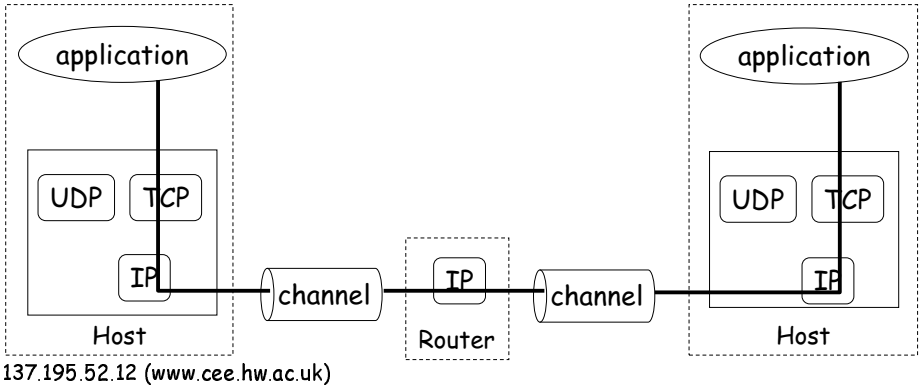
1

This topic

- Introduction to internet protocols & sockets
- Data structs and data handling
- System calls
- Client-server example
- Datagram sockets
- Summary
- praxis

Intro. to internet protocols & sockets

IPC using the internet



(IPC = inter-process communication)

ICP using the internet

- TCP/IP protocol suit
  - set of protocols used on the internet
    - ⇒IP = Internet Protocol (network or routing layer)
    - ⇒TCP = Transmission Control Protocol (end-to-end transport layer)
    - ⇒UDP = User Datagram Protocol (end-to-end transport layer)
- Often provided by OS
  - supported by many UNIXes
- Host
  - end point containing application
- Router
  - provided to route packets through the internet
- Communication channel
  - underlying comms. channel (e.g. Ethernet)

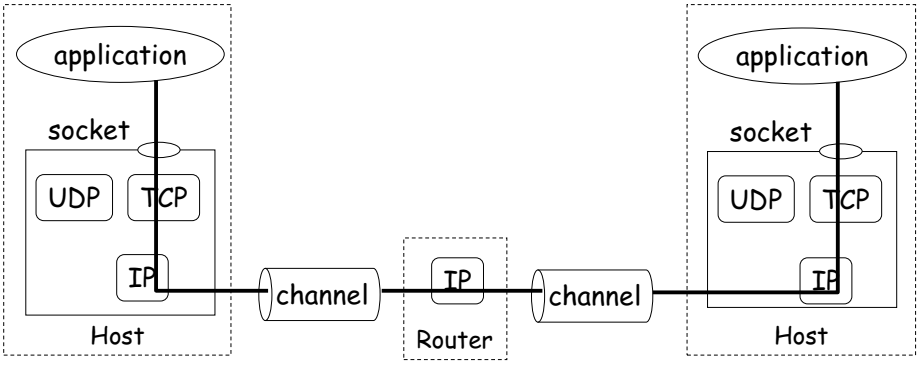
ICP using the internet(2)

- IP protocol
  - network layer 'datagram' service
    - ⇒info. Sent as separate packets
    - ⇒each packet needs the 'internet address' of the destination
    - ⇒packets can be lost, duplicated, delivered out of order …
- Internet address
  - used by IP
  - address of a host's connection onto the internet (e.g. associated with an Ethernet card)
  - 32 bit, dotted-quad notation: 137.195.52.12 (www.cee.hw.ac.uk)
- TCP & UDP
  - two different types of end-to-end (transport) protocols
  - both use 'port numbers' to identify applications within a host
    - ⇒16 bit number

ICP using the internet(3)

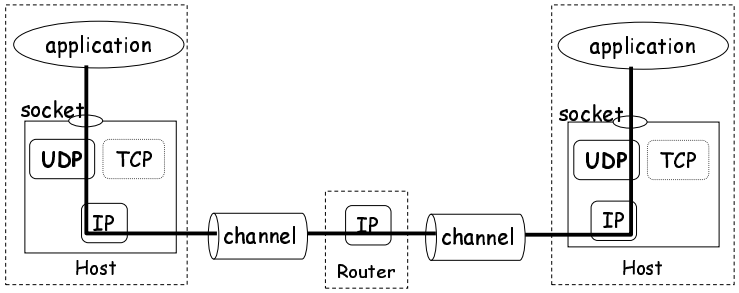
- Domain names
  - e.g. www.cee.hw.ac.uk
  - *resolved* into address by a *name-resolution service*
  - primary source: *Domain Name System (DNS)*
  - DNS protocol allows hosts to exchange and update local domain name databases
- Port numbers
  - internet convention associating certain numbers with certain applications
  - e.g. ftp (file transport protocol) service is normally on port 21
  - see *http://www.iana.org/assignments/port-numbers*

Sockets



de facto API for TCP/IP
(API = application programming interface)

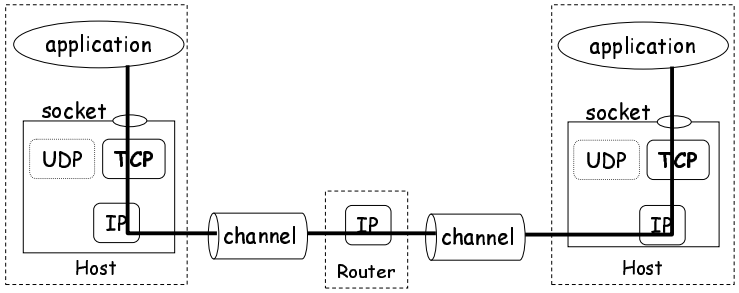
Datagram Sockets (SOCK_DGRAM)



'datagram' (postal) service

- Use UDP (User Datagram Protocol)
- Packets may not be delivered in order in which they are sent
- They may be lost or duplicated
- No need to set up a 'circuit' first
- Do not provide error recovery

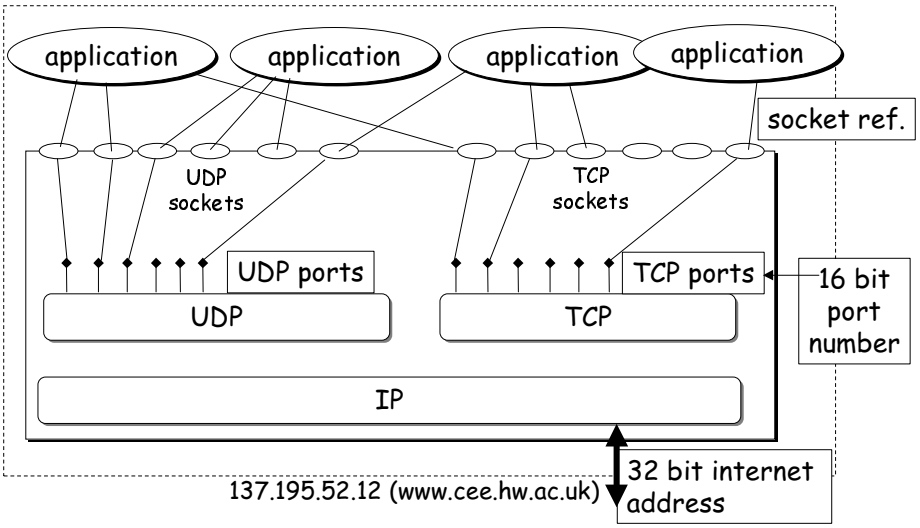
Stream Sockets (SOCK_STREAM)



'virtual circuit' service

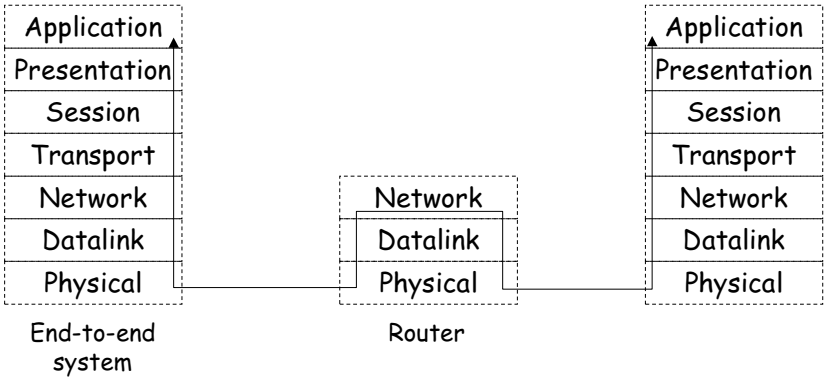
- Use TCP (transmission Control Protocol)
- Packets delivered in order in which they are sent
- No packet duplication
- 'circuit' must be set up and disconnected
- Often provide error recovery

Sockets



Layered comm.s protocols

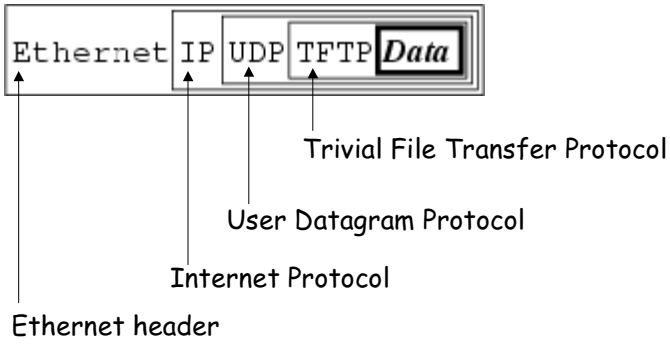
OSI reference model



Unix protocols

- Application Layer (*telnet, ftp, etc.*)
- Host-to-Host Transport Layer (*TCP, UDP*)
- Internet Layer (*IP and routing*)
- Network Access Layer (*Ethernet, ATM, or whatever*)

Data Encapsulation



Data structs and Data Handling

(Beej section 3)

Socket address data structures

struct **sockaddr**: holds address info. for many socket types

unsigned short	*sa_family*	→ addr. family
char	*sa_data[14]*	→ includes 32 bit IP dest. addr. & 16 bit port number

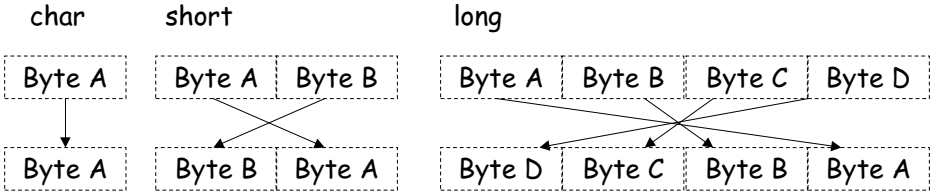
cast

struct **sockaddr_in**: internet version

short int	*sin_family*	→ addr. family
unsigned short int	*sin_port*	→ 16 bit port No.
struct in_addr	*sin_addr*	→ 32 bit IP addr.
unsigned char	*sin_zero[8]*	→ padding

network byte order

Byte swapping



- Network byte order (BigEndian byte order - 68000)
- Host byte order

htons() -- "Host to Network Short"

htonl() -- "Host to Network Long"

ntohs() -- "Network to Host Short"

ntohl() -- "Network to Host Long"

Loading in an IP address

sockaddr_in my_addr

sin_family
sin_port
sin_addr
sin_zero[8]

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
```

structrue addr

ASCII IP address

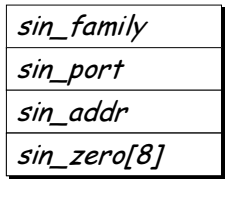
internet ASCII to network

older version: inet_addr (IP address)
(used in examples)

Loading in an IP address (2)

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
```

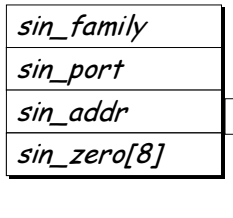
sockaddr_in my_addr




```
struct sockaddr_in my_addr;

my_addr.sin_family = AF_INET;                    // host byte order
my_addr.sin_port = htons(MYPORT);                // short, network byte order
inet_aton("10.12.110.57", &(my_addr.sin_addr));
memset(&(my_addr.sin_zero), '\0', 8);            // zero the rest of the struct
```


Printing out an IP address

sockaddr_in my_addr



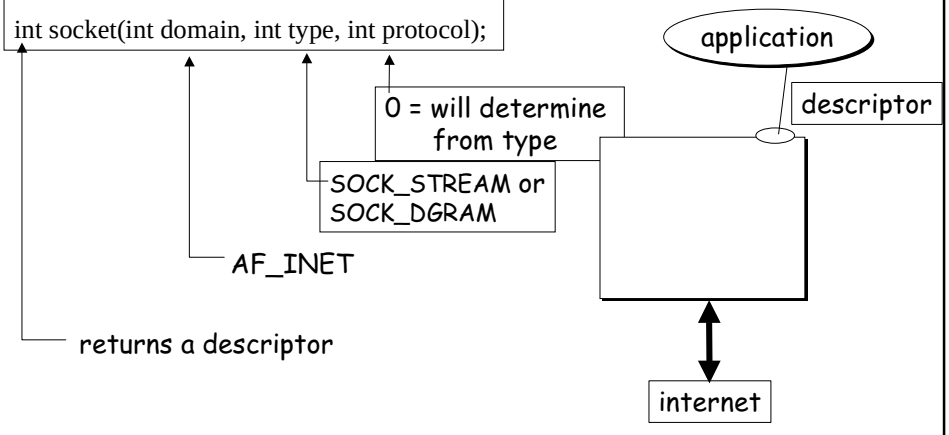

```
printf("$s", inet_ntoa(my_addr.sin_addr);
```

ntoa = network to ASCII

System calls
(Beej section 4)

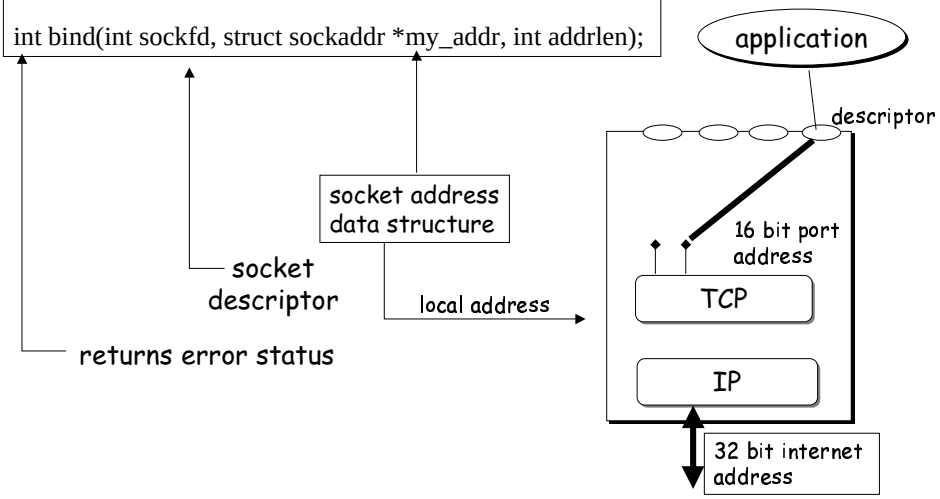
socket() system call

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```



Binding a socket to a port

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```



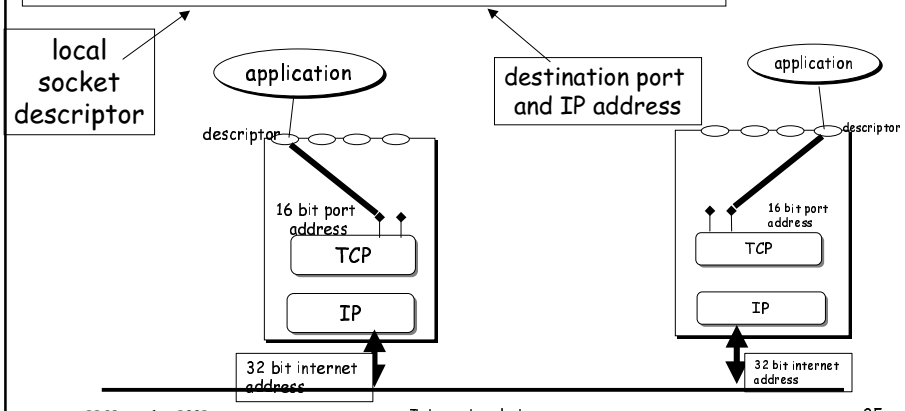
Example

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MYPORT 3490

main()
{
   int sockfd;
   struct sockaddr_in my_addr;

   sockfd = socket(AF_INET, SOCK_STREAM, 0);  // create socket
   my_addr.sin_family = AF_INET;              // host byte order
   my_addr.sin_port = htons(MYPORT);          // short, network byte order
   my_addr.sin_addr.s_addr = inet_addr ("10.12.110.57");
   memset(&(my_addr.sin_zero), '\0', 8);      // zero the rest of the struct

   // don't forget your error checking for bind():
   bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
   . . .
```

Error checking!!

The client: connecting to a remote host

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```



Connecting to a remote host(2)

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define DEST_IP "10.12.110.57"
#define DEST_PORT 23

main() {
int sockfd;
struct sockaddr_in dest_addr; // will hold the destination addr
sockfd = socket(AF_INET, SOCK_STREAM, 0); // do some error checking!
dest_addr.sin_family = AF_INET; // host byte order
dest_addr.sin_port = htons(DEST_PORT); // short, network byte order
dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
memset(&(dest_addr.sin_zero), '\0', 8); // zero the rest of the struct

// don't forget to error check the connect()!
connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
. . .
```

note: did not call bind as we don't care about the local port number

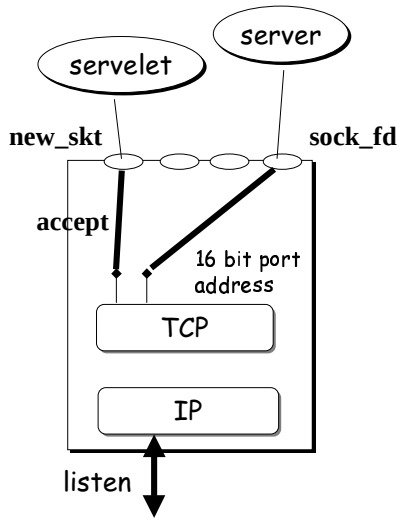
The server: listening & accepting connections

int **listen**(int sockfd, int backlog);

No. of connections allowed on the incoming queue

socket descriptor

```
sock_fd = socket( );
//set up local sockaddr
bind( );
listen( );

        new_skt = accept( );
        .
        .
```



Listening & accepting connections

```
main() {
  int sockfd, new_fd;         // listen on sock_fd, new connection on new_fd
  struct sockaddr_in my_addr; // my address information
  struct sockaddr_in their_addr; // connector's address information
  int sin_size;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  my_addr.sin_family = AF_INET;             // host byte order
  my_addr.sin_port = htons(MYPORT);         // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY; // auto-fill with my IP
  memset(&(my_addr.sin_zero), '\0', 8);     // zero the rest of the struct

  // don't forget your error checking for these calls:
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
  listen(sockfd, BACKLOG);
  sin_size = sizeof(struct sockaddr_in);
  new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
```

sending data (virt. cct.)

int **send**(int sockfd, const void *msg, int len, int flags);

```
int len, bytes_sent;
.
.
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.
.
.
```

char *msg = "Beej was here!";

socket descriptor

No. bytes sent (may be less than len!)

receiving data (virt. cct.)

int **recv**(int sockfd, void *buf, int len, unsigned int flags);

buffer

peek/waitall

socket descriptor

-1 = error
0 = remote system has closed the connection
+ve = number of bytes received

Sending & receiving (datagrams)

int **sendto**(int sockfd, const void *msg, int len, unsigned int flags,
const struct sockaddr ***to**, int **tolen**);

to addr. length

destination address

int **recvfrom**(int sockfd, void *buf, int len, unsigned int flags,
struct sockaddr ***from**, int ***fromlen**);

source address filled in
by recvfrom

closing down

close(sockfd);

shutdown (sockfd, how);

0 = disable receives
1 = disable sends
2 = disable both

Peer and host names

int **getpeername**(int sockfd, struct sockaddr *addr, int *addrlen);

local socket descriptor

sockaddr of remote system returned

int **gethostname**(char *hostname, size_t size);

address of string to contain the host name

```
#include <unistd.h>
main() {
        char hostname[50];

        gethostname(hostname, 50);
        printf("%s", hostname);
}
```


DNS: Domain Name Service

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

h_name	-- Official name of the host (e.g www.cee.hw.ac.uk)
h_aliases	-- A NULL-terminated array of alternate names for the host.
h_addrtype	-- The type of address being returned; usually *AF_INET*.
h_length	-- The length of the address in bytes.
h_addr_list	-- A zero-terminated array of network addresses for the host. Host addresses are in Network Byte Order.
h_addr	-- The first IP address in *h_addr_list*. (e.g. 137.195.52.12)

DNS: Domain Name Service

getip.c

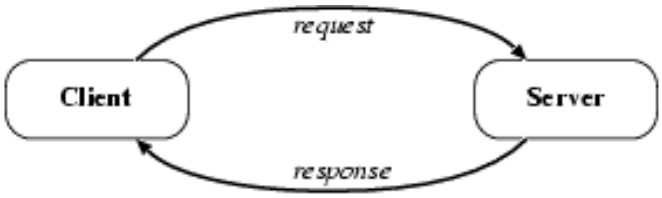
```
int main(int argc, char *argv[ ]) {
        struct hostent *h;

        if (argc != 2) { // error check the command line
                fprintf(stderr,"usage: getip address\n");
                exit(1);
        }
        if ((h=gethostbyname(argv[1])) == NULL) { // get the host info
                herror("gethostbyname");
                exit(1);
        }
        printf("Host name : %s\n",  h->h_name);
        printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));

        return 0;
}
```


DNS: Domain Name Service

odin:

```
$ gethostname
odin
$ getip odin
Host name  : odin
IP Address : 137.195.52.4
$
$ getip www.cee.hw.ac.uk
Host name  : www.cee.hw.ac.uk
IP Address : 137.195.52.12
$
```

Client-Server example
(Beej section 5)



Client-Server example(1)

server.c

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,sizeof(int));

//Load up my_addr structure with local addresses

bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
listen(sockfd, BACKLOG);

while(1)
{
    new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
    printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));
    if (!fork())
    {
        close(sockfd); // child doesn't need the listener
        send(new_fd, "Hello, world!\n", 14, 0) ;
        close(new_fd);
        exit(0);
    }
    close(new_fd); // parent doesn't need this
}
```

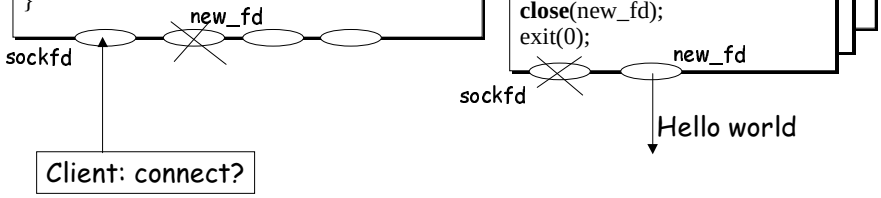
parent

child

Client-Server example(2)

server.c

```
while(1)
{
    new_fd = accept(sockfd);
    printf("server: got connection from %s\n",
    inet_ntoa(their_addr.sin_addr));
    fork child;
    close(new_fd); // parent doesn't need this
}
```



Server children

```
close(sockfd); // close listener
send(new_fd, "Hello, world") ;
close(new_fd);
exit(0);
```


Client-Server example(3)

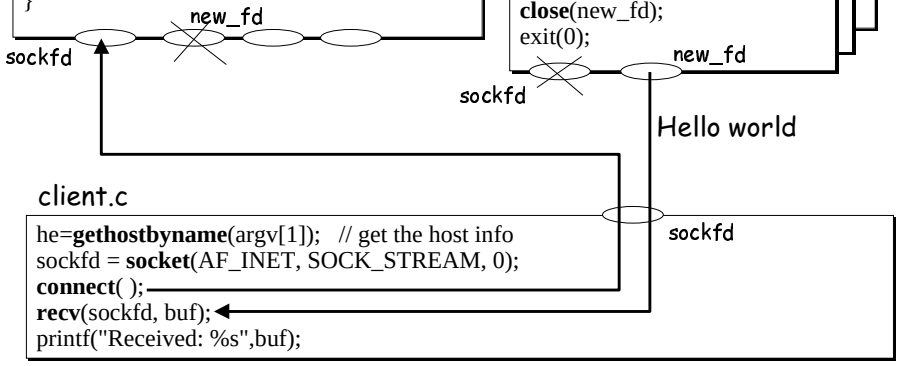
client.c

```
he=gethostbyname(argv[1]);   // get the host info
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Load up their_addr data structure

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

numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0);
buf[numbytes] = '\0';
printf("Received: %s",buf);
close(sockfd);
```

Client-Server example(4)

server.c

```
while(1)
{
    new_fd = accept(sockfd);
    printf("server: got connection from %s\n",
    inet_ntoa(their_addr.sin_addr));
    fork child;
    close(new_fd); // parent doesn't need this
}
```



client.c

```
he=gethostbyname(argv[1]);   // get the host info
sockfd = socket(AF_INET, SOCK_STREAM, 0);
connect( );
recv(sockfd, buf);
printf("Received: %s",buf);
```

sockfd

Client-Server example(5)

client.c

```
he=gethostbyname(argv[1]);   // get the host info
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Load up their_addr data structure

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

numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0);
buf[numbytes] = '\0';
printf("Received: %s",buf);
close(sockfd);
```

Client-Server example(6)

example run

osiris:

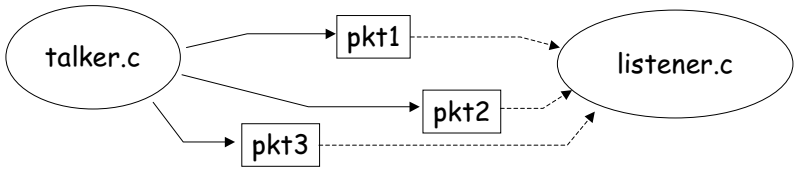
```
$ uname -a
OSF1 osiris V5.1 732 alpha


$ server
server: got connection from 137.195.52.12

server: got connection from 137.195.52.12




server: got connection from 137.195.27.130
$ >
```

osiris/linux30:

```
$ uname -a
OSF1 osiris V5.1 732 alpha

$ client osiris
Received: Hello, world!

$ client osiris
Received: Hello, world!

---------------------------------------

$ rlogin linux30
linux30> uname -a
Linux linux30 2.4.9-12 #1 ……..

linux30> cc client.c –o client
linux30> client osiris
Received: Hello, world!
linux30>
linux30>
```


Datagram sockets

(Beej section 5.3)



Datagram sockets(1)

listener.c

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0)

//Set up local addresses:
my_addr.sin_family = AF_INET; // host byte order
my_addr.sin_port = htons(MYPORT); // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr);
numbytes=recvfrom(sockfd, buf, MAXBUFLEN-1, 0, &their_addr, &addr_len)

printf("got packet from %s",inet_ntoa(their_addr.sin_addr));
printf("packet is %d bytes long",numbytes);
buf[numbytes] = '\0';
printf("packet contains %s", buf);

close(sockfd);
```

Notes: 1.no need to 'listen' or 'accept' on DGRAM sockets
2.recvfrom( ) returns address of sender

Datagram sockets(2)

talker.c

```
he=gethostbyname(argv[1]);  // get the host info


sockfd = socket(AF_INET, SOCK_DGRAM, 0);

// Set up their address:
their_addr.sin_family = AF_INET; // host byte order
their_addr.sin_port = htons(MYPORT); // short, network byte order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(their_addr.sin_zero), '\0', 8); // zero the rest of the struct

numbytes=sendto(sockfd, argv[2], strlen(argv[2]), 0, &their_addr)

printf("sent %d bytes to %s", numbytes, their_addr.sin_addr);
close(sockfd);
```

Note this is pseudo code!!

Datagram sockets(3)

listener.c

linux30> **listener**

got packet from 137.195.27.101
packet is 8 bytes long
packet contains "Hi there“
linux30>

talker.c

linux01> **talker**
usage: talker hostname message
linux01> talker linux30 "Hi there"
sent 8 bytes to 137.195.27.130
linux01>

Unix filespace:
~mjc/www/teaching/122no2-OperatingSystems/Beejs-network-programming

Summary

Summary

- Introduction
  - internet protocols & sockets
  - Datagram and virtual cct services
  - Layered comm.s + encapsulation
- Data structs and data handling
  - Socket address data structs to hold IP + port addresses
  - Byte swapping (host + network)
- System calls
  - socket ( ) ,bind ( )
  - connect ( )
  - listen ( )
- Client-server example
  - listen, accept, fork, and close unwanted sockets, then
  - recv (socketfd), send (socketfd)
- Datagram sockets
  - recvfrom(socketfd, their_address)
  - sendto (socketfd, their_address)
  - no need for listen or accept

Praxis

Lab-ex6.1
Internet communications

1. Compile and run getip.c (Beej section 4.11) from
   ~mjc/www/teaching/122no2-OperatingSystems/Beejs-network-programming
   Ask it for various addresses e.g. www.hw.ac.uk
2. Compile server.c and client.c for two different architectures. Run the client several times, then kill the server and run the client again.
3. Do the same for the datagram programs: talker.c and listener.c

Tut-6.1
Internet server

1. Design an internet server based on server.c that accepts a string from a remote client, interprets it as a set of shell commands separated by pipe symbols. Runs these commands locally (e.g. ls | wc –l) and returns the result to the client as a text message.

Revision questions for topic 6

a) What is the difference between a stream socket and a datagram socket?
b) When would use each of these types of service?
c) What is dotted quad notation?
d) What does DNS stand for and what does it do?
e) How many ports are associated with a particular IP address?
f) Which bytes are swapped with which when a 4 byte word is transformed from host to network byte order?
g) Why would you call the system service 'bind' on a socket and what does it do?
h) What does the 'accept' system service return and how would you use it?
i) What sort of error checking should you perform when calling these system services and how would you report any such errors?

Reading

- Beej's Guide to Network Programming: sections 1 to 5 (http://www.ecst.csuchico.edu/~beej/guide/net/)