



PennState

CMPSC 311 - Introduction to Systems Programming

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Network Programming

Add WeChat edu_assist_pro

Professors:

Suman Saha

Slides are mostly by *Professor Patrick McDaniel*
and *Professor Abutalib Aghayev*)

What is a network?



PennState

- A network is a collection of computing devices that share a transmission media

- Traditional wired networks (ethernet)
- High-speed backbone (
- Wireless (radio)
- Microwave
- Infrared

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- The way the network is organized is called the **network topology**

The Internet



The Internet is an interconnected collection of many networks.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Internet protocol stack (TCP/IP)



- **application**: supporting network applications

- FTP, SMTP, HTTP, DNS

- **transport**: process-process data transfer

- TCP, UDP

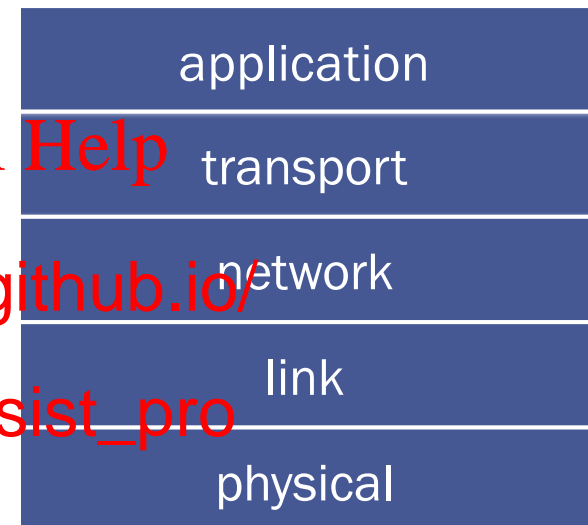
- **network**: routing of data to destination

- IP, routing protocols

- **link**: data transfer between neighboring network elements

- PPP, Ethernet

- **physical**: bits “on the wire”



Network vs. Web



PennState

- The network is a service ...

- A conduit for data to be passed between systems.
- Layers services (generally) to allow flexibility.
- Highly scalable.
- This is a public channel

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- The Web is an application

- This is an application for viewing/manipulating content.
- This can either be public (as in CNN's website), or private (as in enterprise internal HR websites).

Networks Systems



- Conceptually, think about network programming as two or more programs on the same or different computers talking to each other
 - The send messages back and forth
 - The “flow” of messages content is called the [network protocol](https://eduassistpro.github.io/) or just

Add WeChat edu_assist_pro



What's a Protocol?



- Example: A human protocol and a computer protocol:

Assignment Project Exam Help



<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



- Question: What are some other human protocols?

Socket Programming



- Almost all meaningful careers in programming involve at least some level of network programming.
- Most of them involve sockets programming
 - Berkeley sockets origin
 - it is the standard A
 - available on most OSs
 - POSIX socket API
 - a slight updating of the Berkeley sockets API
 - a few functions were deprecated or replaced
 - better support for multi-threading was added

Assignment Project Exam Help

<https://eduassistpro.github.io/>

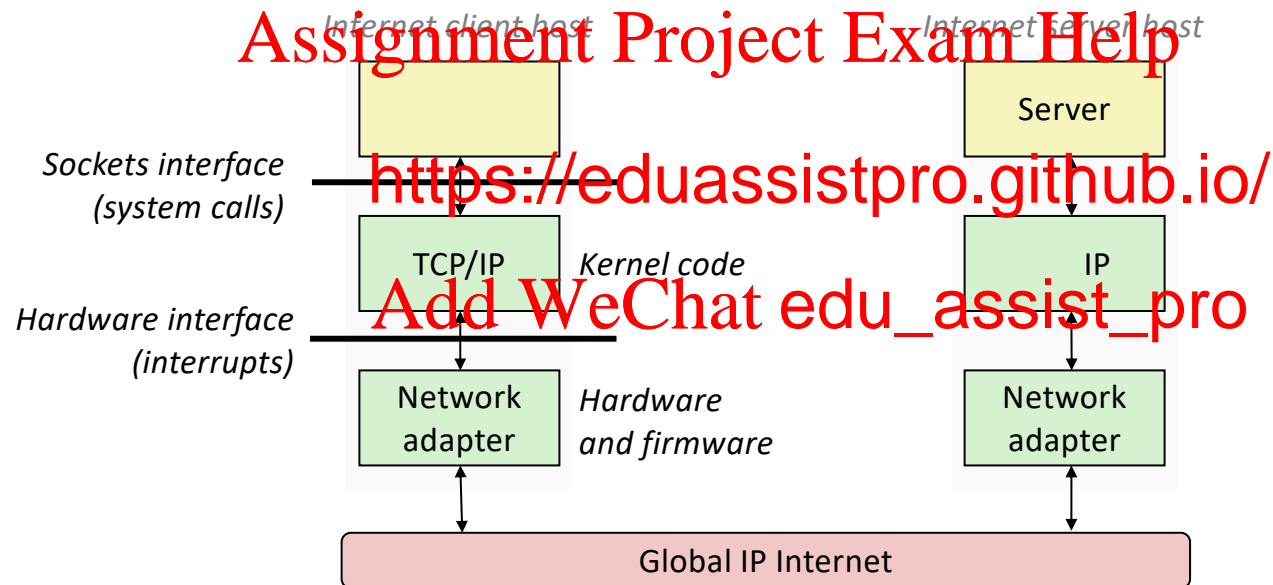
Add WeChat edu_assist_pro



Hardware and Software Organization of Internet Application



PennState



- Bryant and O'Hallaron, Computer Systems: A Programmer's Approach

IP addresses



PennState

- Every device on the Internet needs to have an address
 - Needs to be unique to make sure that it can be reached
- For IPv4, an IP address is a 4-byte tuple
 - e.g., 128.95.4.1 (80:5
- For IPv6, an IP address is
 - e.g., 2d01:0db8:f188:0000:0000:0000:0000:0000
 - 2d01:0db8:f188::1f33 in shorthand

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

A Programmer's View of the Internet



PennState

- Hosts are mapped to a set of 32-bit IP Address
 - 146.186.145.12
- The set of IP addresses are called Internet domain names
 - 146.186.145.12 is mapped to www.eecs.psu.edu
- A process on one Internet host can communicate with a process on another Internet host over a connection

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Recall file descriptors



PennState

- Remember open, read, write, and close?

- POSIX system calls interacting with files

- recall `open ()` returns a file descriptor

- an integer that repr

- inside the OS, it's a lack of any state associated with your interactions, such as the file p

- you pass the file descriptor into read, write

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



Networks and sockets



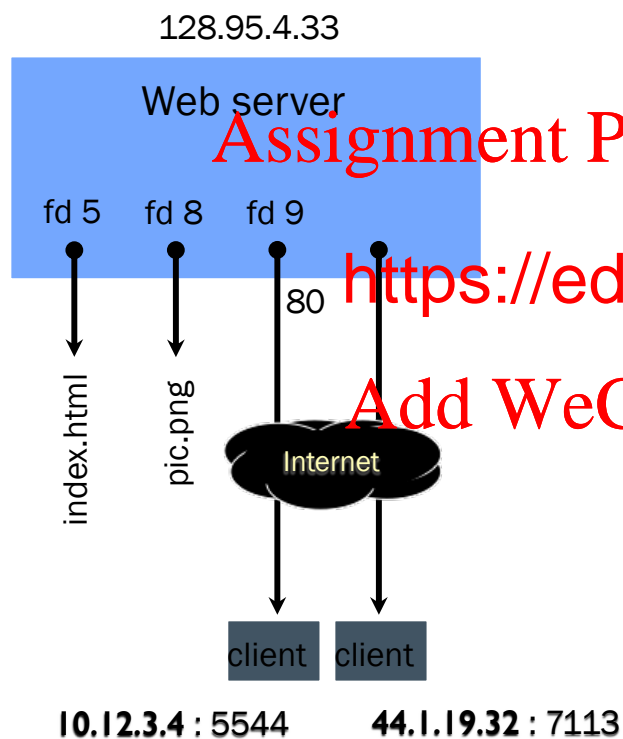
- UNIX makes all I/O look like file I/O
 - the good news is that you can use `read()` and `write()` to interact with remote computers over a network.
 - just like with files....
 - A program can have `read()` and `write()` at once
 - you need to pass `read()` and `write()` or to let the OS know which network channel write to or read from
 - The file descriptor used for network communications is a socket



Pictorially



PennState



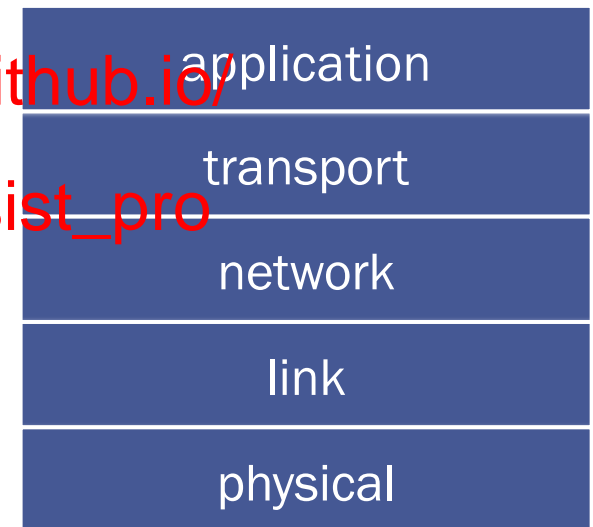
file descriptor	type	connected to?
0	pipe	stdin (console)
	pipe	stdout (console)
	pipe	stderr (console)
	TCP socket	local: 128.95.4.33:80 remote: 44.1.19.32:7113
5	file	index.html
8	file	pic.png
9	TCP socket	local: 128.95.4.33:80 remote: 10.12.3.4:5544

OS's descriptor table

Types of sockets



- Stream sockets
 - for connection-oriented, point-to-point, reliable bytestreams
 - uses **TCP**, SCTP, or other stream transports
- Datagram sockets
 - for connection-less, on
 - uses **UDP** or other packet transports
- Raw sockets
 - for layer-3 communication (raw IP packet manipulation)



Stream (TCP) sockets



PennState

- Typically used for client / server communications
 - but also for other architectures, like peer-to-peer

- Client

- an application that est

- Server

- an application that receives connections fro



1. establish connection



2. communicate



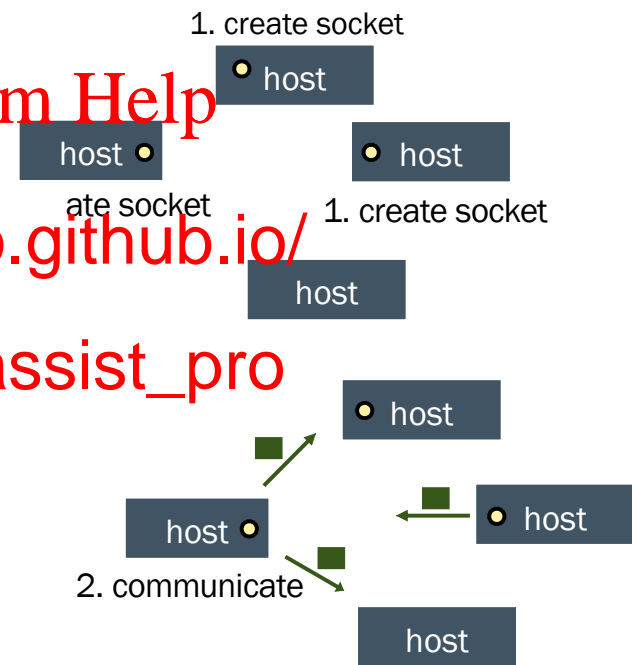
3. close connection

Datagram (UDP) sockets



PennState

- Used less frequently than stream sockets
 - they provide no flow control, ordering, or reliability
 - really, provides best effort communication
- Often used as a building block
 - streaming media applications
 - sometimes, DNS lookups



Note: this is also called “*connectionless*” communication

TCP connections



PennState

- Clients and servers communicate by sending streams of bytes over **connections**. Each connection is:
 - point-to-point: connects a pair of processes
 - full-duplex: data can flow in both directions
 - reliable: data sent/received in order
- A **socket** is an endpoint
 - socket address is: IP address and port number
- A **port** is a 16-bit integer that identifies a
 - **ephemeral port**: assigned automatically by the client kernel when client makes a connection
 - **well-known port**: associated with some service provided by a server:
(e.g. 80 is HTTP/Web)

Network Ports



PennState

- Every computer has a numbered set of locations that represent the available “services” can be reached at
 - Ports are numbered 0 to 65535
 - 0 to 1023 are called **privileged ports**, where you need special (root) privileges to run
 - Each transport (UDP/TCP)
- Interesting port numbers
 - 20/21 - file transfer protocol (file passing)
 - 22 - secure shell (remote access)
 - 25 - Simple mail transfer protocol (email)
 - 53 - domain name service (internet naming)
 - 80 - HTTP (web)

Assignment Project Exam Help

<https://eduassistpro.github.io/>

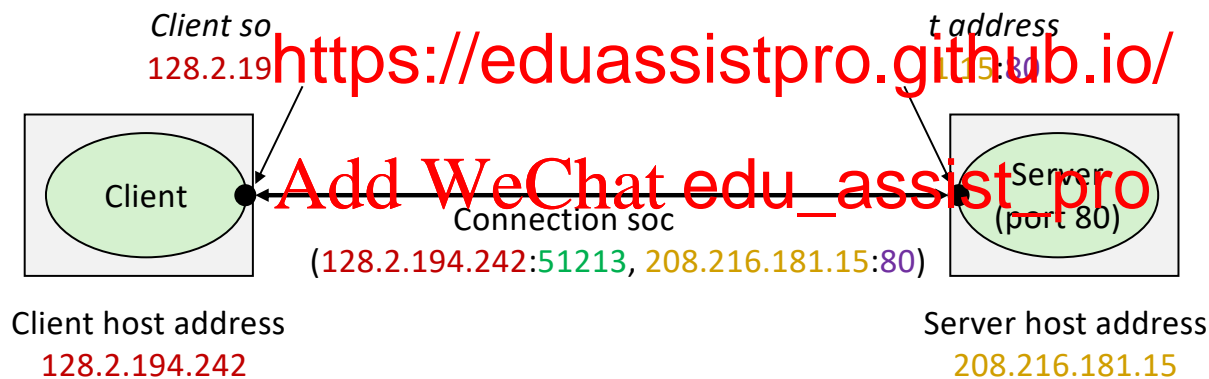
Add WeChat edu_assist_pro

Anatomy of a connection



- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)

- (client IP:client port, server IP:server port)



51213 is an ephemeral port
allocated by the kernel

80 is a well-known port
associated with Web servers

Programming a client



- We'll start by looking at the API from the point of view of a client connecting to a server over TCP

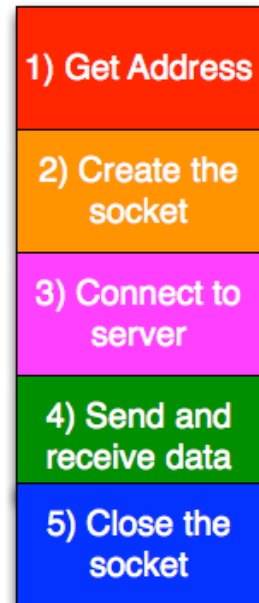
- there are five steps:

- figure out the addr
- create a socket
- connect the socket to the remote server
- `read()` and `write()` data using the s
- close the socket

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



inet_aton()

1) Get Address



PennState

- The `inet_aton()` converts a IPv4 address into the UNIX structure used for processing:

```
int inet_aton(const char *addr, struct in_addr *inp);
```

- Where,

- `addr` is a string containing the IPv4 address (e.g., "6.84.7.99")
- `inp` is a pointer to the `struct in_addr` structure, which is a binary representation of an address, used in later network communication

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

An example:

- IPv4 to Binary: 166.84.7.99 → 10100110 01010100 00000111 01100011
(2790524771)
- Little Endian → Big Endian: 01100011 00000111 01010100 10100110
- Binary → Decimal: 1661424806

`inet_aton()` returns 0 if failure!

Putting it to use ...

1) Get Address



PennState

Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

```
#include <stdio.h>
#include <arpa/inet.h>

int main(int arg
    struct sockadd
    struct sockadd

    // IPv4 string to sockaddr_in. (both
    inet_aton("192.0.2.1", &sa.sin_addr
    inet_pton(AF_INET, "192.0.2.1", &(sa

    // IPv6 string to sockaddr_in6.
    inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));

    return( 0 );
}
```

Getting back to strings?

1) Get Address



PennState

- The `inet_ntoa()` converts a UNIX structure for an IPv4 address into an ASCII string:

`char *inet_ntoa(struct in_addr in);`

```
struct sockaddr_in caddr;
struct sockaddr_in6 sa6;
char astring[INET6_ADDRSTRLEN]; // IPv6

// Start by converting
inet_aton( "192.168.8.9", &caddr.sin_addr );
inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr));
inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));

// Return to ASCII strings
inet_ntop(AF_INET6, &(sa6.sin6_addr), astring, INET6_ADDRSTRLEN);
printf( "IPv4 : %s\n", inet_ntoa(caddr.sin_addr) );
printf( "IPv6 : %s\n", astring );
```


Domain Name Service (DNS)

1) Get Address



PennState

- People tend to use DNS names, not IP addresses
 - the sockets API lets you convert between the two
 - it's a complicated process, though.
 - a given DNS name
 - many different DNS addresses
 - an IP address will map onto at most one address, and maybe none
 - a DNS lookup may require interacting with multiple servers

Note: The “dig” Linux program is used to check DNS entries.

Domain Name Service (DNS)

1) Get Address



PennState

- People

- the soc
- it's a co
- a
- m
- a

```
$ dig lion.cse.psu.edu
```

```
; <<>> DiG 9.9.2-P1 <<>> lion.cse.psu.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53447
;; flags: qr rd ra
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; MBZ: 0005 , udp: 4000
;; QUESTION SECTION:
;lion.cse.psu.edu. IN A
;; ANSWER SECTION:
lion.cse.psu.edu.      5      IN      A      130.203.22.184
;; Query time: 38 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Tue Nov 12 14:02:11 2013
;; MSG SIZE rcvd: 61
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat: edu_assist_pro

Note: The “dig” Linux program is used to check DNS entries.

The FQDN

1) Get Address



PennState

- Every system that is supported by DNS has a unique **fully qualified domain name**

Assignment Project Exam Help

<https://eduassistpro.github.io/>

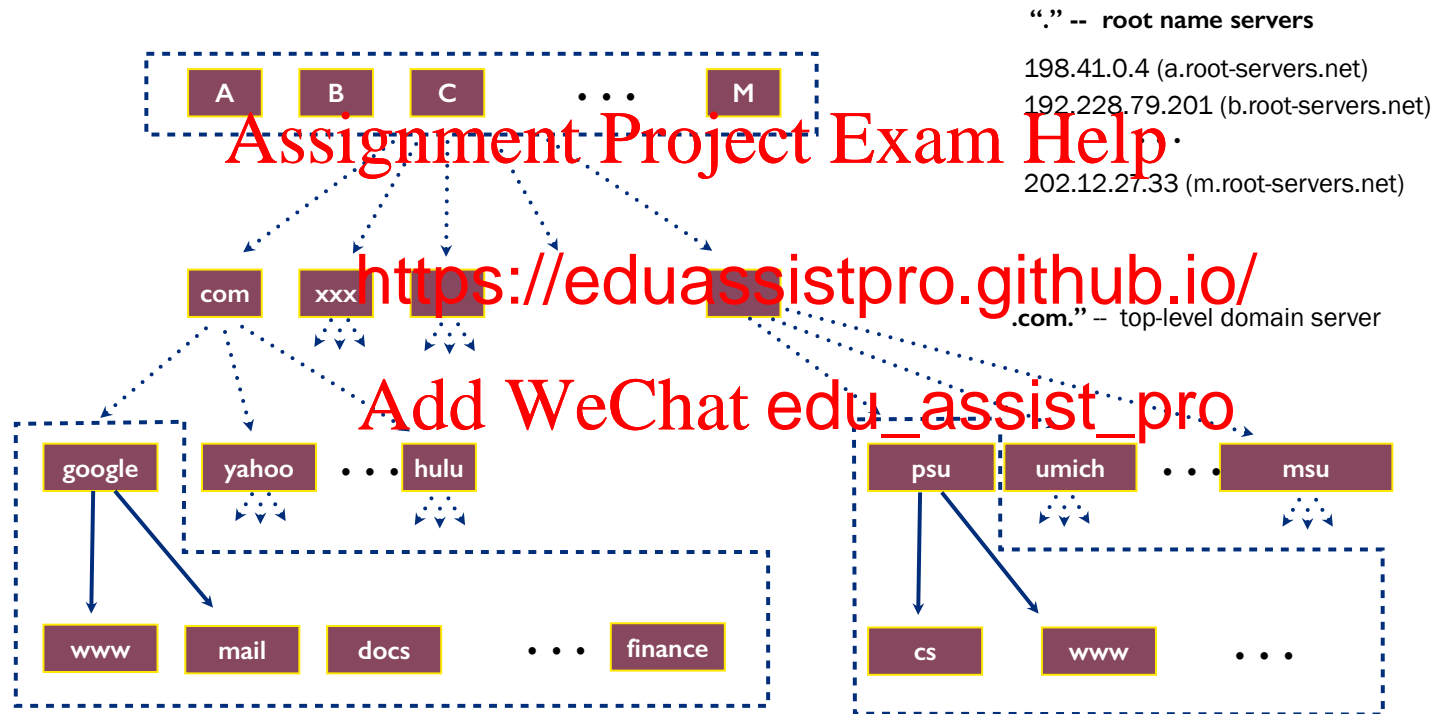
Add WeChat edu_assist_pro

DNS hierarchy

1) Get Address



PennState



Resolving DNS names

1) Get Address



PennState

- The POSIX way is to use `getaddrinfo()`
- a pretty complicated system call; the basic idea...
 - set up a “hints” structure with constraints you want respected
 - e.g., IPv6, IPv4, or e
 - indicate which host an
 - host: a string representation; DNS name
 - returns a list of results packet in an “`addrinfo`”
 - free the `addrinfo` structure using `freeaddrinfo()`

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

DNS resolution (the easy way)



PennState

1) Get Address

- The `gethostbyname()` uses DNS to look up a name and return the host information

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

- Where,

- ▶ `name` is a string containing a domain name. `name` is either a hostname, or an IPv4 address in standard dot notation. `name` includes:
- ▶ `hostent->h_name` (fully qualified domain name)
- ▶ `hostent->h_addr_list` (list of pointers to IP addresses)

DNS resolution (the easy way)

1) Get Address



PennState

- The `gethostbyname()` uses DNS to look up a name and return the host information

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

- Where,

- ▶ `name` is

- ▶ `hostent`

or an IPv

- ▶ `host`

- ▶ `host`

```
char *hn = "lion
struct hostent *
struct in_addr *
if ((hstinfo = g
    return -1;
}
```

```
addr_list = (struct in_addr **)hstinfo->h_addr_list;
printf( "DNS lookup [%s] address [%s]\n", hstinfo->h_name,
        inet_ntoa(*addr_list[0]) );
```

```
$ ./network
```

```
DNS lookup [lion.cse.psu.edu] address [130.203.22.184]
```

```
$
```

hostname,

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Programming a client



PennState

- We'll start by looking at the API from the point of view of a client connecting to a server over TCP
 - there are five steps:
 - figure out the addr
 - create a socket
 - connect the socket to the remote server
 - read() and write() data using the socket
 - close the socket



Creating a socket

2) Create the socket



PennState

- The `socket()` function creates a file handle for use in communication:

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

- Where,

- ▶ `domain` is the communication domain
 - ▶ `AF_INET` (IPv4), `AF_INET6`
- ▶ `type` is the communication semantics (socket type)
 - ▶ `SOCK_STREAM` is stream (using TCP by default)
 - ▶ `SOCK_DGRAM` is datagram (using UDP by default)
- ▶ `protocol` selects a protocol from available (not used often)

Note: creating a socket doesn't connect to anything

Creating a socket

2) Create the socket



PennState

- The `socket()` function creates a file handle for use in communication:

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

- Where,

- `domain`

- `AF_`

- `type`

- `SOC`

- `SOCK_DGRAM` is datagram (using UDP by default)

- `protocol` selects a protocol from available (not used often)

```
// Create the soc
int sockfd;
sockfd = socket(AF_
if (sockfd == -1) {
    printf( "Error on socket creation [ %s\n", strerror(errno) );
    return( -1 );
}
```

Note: creating a socket doesn't connect to anything

Specifying an address

3) Connect to server



PennState

- The next step is to create an address to connect to by specifying the address and port in the proper form.

- protocol family (`addr.sin_family`)
- port (`addr.sin_port`)
- IP address (`addr.sin_addr`)

Assignment Project Exam Help

<https://eduassistpro.github.io/>

```
// Variables
char *ip = "127.0.0.1";
unsigned short port = 16453;
struct sockaddr_in caddr;

// Setup the address information
caddr.sin_family = AF_INET;
caddr.sin_port = htons(port);
if ( inet_aton(ip, &caddr.sin_addr) == 0 ) {
    return( -1 );
}
```

Add WeChat edu_assist_pro

Network byte order

3) Connect to server



PennState

- When sending data over a network you need to convert your integers to be in network byte order, and back to host byte order upon receive:

Assignment Project Exam Help

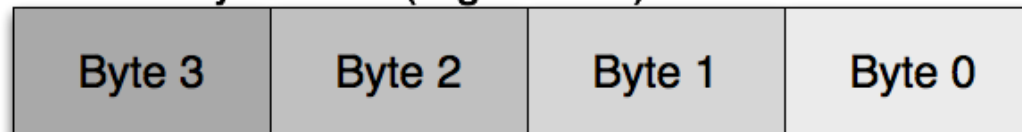
```
uint64_t htonl64(uint64_t hostlong64);  
uint32_t htonl(uint32_t netlong);  
uint16_t ntohs(uint16_t netshort);
```

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- Where each of these functions receives a 64/ 32 /16 byte and converts it to the other.

Network byte order (Big Endian)



connect()

3) Connect to server



PennState

- The connect() system call connects the socket file descriptor to the specified address

```
int connect(int sockfd, const struct sockaddr *addr,  
            socklen_t addrlen);
```

- Where,

- sockfd is the socket (file descriptor)
- addr - is the address structure
- addrlen - is the size of the address structure
- Returns 0 if successfully connected, -1 if not

```
if ( connect(sockfd, (const struct sockaddr *)&caddr,  
             sizeof(caddr)) == -1 ) {  
    return( -1 );  
}
```

Reading and Writing

4) Send and
receive data



PennState

- Primitive reading and writing only process only blocks of opaque data:

```
ssize_t write(int fd, const void *buf, size_t count);  
ssize_t read(int fd, void *buf, size_t count);
```

- Where `fd` is the file descriptor, `buf` is the buffer to write from or read into, and `count` is the number of bytes to write or read.
- The value returned is the number of bytes read or written.
 - Be sure to always check the result
- On reads, you are responsible for supplying a buffer that is large enough to put the output into.
 - look out for memory corruption when buffer is too small ...

close()

5) Close the
socket



PennState

- `close()` closes the connection and deletes the associated entry in the operating system's internal structures

Assignment Project Exam Help

<https://eduassistpro.github.io/>

socket_fd =
Add WeChat edu_assist_pro

Note: set handles to `-1` to avoid use after close.

Elementary TCP client/server



PennState

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Programming a server



- Now we'll look at the API from the point of view of a server who receives connections from clients

- there are seven steps:
 - figure out the port
 - create a socket
 - bind the service
 - begin listening for connections
 - receive connection
 - read() and write() data
 - close the socket

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Setting up a server address

1) Setup
Address



PennState

- All you need to do is specify the service port you will use for the connection:

```
struct sockaddr_in saddr;  
saddr.sin_family  
saddr.sin_port =  
saddr.sin_addr.s
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

- However, you don't specify an IP address are receiving connections at the local host.
 - Instead you use the special "any" address

`htonl(INADDR_ANY)`

Next: creating the socket is as with the client

2) Create the
server socket

Binding the service

3) Bind to the service



PennState

- The `bind()` system call associates a socket with the server connection (e.g., taking control of HTTP)

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

- Where,

- `sockfd` is the socket (
- `addr` - is the address structure
- `addrlen` - is the size of the address structure
- Returns 0 if successfully connected, -1 if not

```
if ( bind(sock, (const struct sockaddr *)&saddr,
          sizeof(saddr)) == -1 ) {
    return( -1 );
}
```

Listening for connections

4) Listen for connections



PennState

- The `listen()` system call tells the OS to receive connections for the process

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

- Where,

- `sockfd` is the socket (`int`)
- `backlog` - is the number of connections that can be queued for the process

- A program may process connections as they come in and the OS will hold the client in a waiting state until you are ready to accept them
- Beware of waiting too long (timeout)

```
if ( listen(sock, 5) == -1 ) {  
    return( -1 );  
}
```

Accepting connections

4) Accept connection



PennState

- The `accept ()` system call receives the connection from the client:

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Where,

Assignment Project Exam Help

- `sockfd` is the socket (

- `addr` - is the address s

<https://eduassistpro.github.io/>

- `addrlen` - is the size of

- Returns the new socket handle or -1 if failure

Add WeChat edu_assist_pro

Accepting connections

4) Accept connection



PennState

- The `accept ()` system call receives the connection from the client:

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Where,

Assignment Project Exam Help

- `sockfd` is the socket (

- `addr` - is the address s

<https://eduassistpro.github.io/>

- `addrlen` - is the size of

- Returns the new socket `sockfd` and `len` if failr

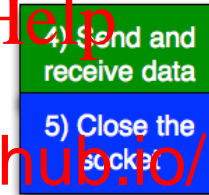
Add WeChat edu_assist_pro

```
inet_len = sizeof(caddr);
if ( (client = accept(server, (struct sockaddr *)&caddr, &inet_len)) == -1 )
{
    printf( "Error on client accept [%s]\n", strerror(errno) );
    close(server);
    return( -1 );
}
printf( "Server new client connection [%s/%d]",
        inet_ntoa(caddr.sin_addr), caddr.sin_port );
```

The rest ...



- From the server perspective, receiving and sending on the newly received socket is the same as if it were a client
 - `read()` and `write()` for sending and receiving
 - `close()` for closing the



Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Putting it all together (client)



PennState

```
int client_operation( void ) {  
  
    int socket_fd;  
    uint32_t value;  
    struct sockaddr_in caddr;  
    char *ip = "127.0.0.1";  
  
    caddr.sin_family = AF_INET;  
    caddr.sin_port = htons(6555);  
    if ( inet_aton(ip, &caddr.sin_addr) == 0 ) {  
        return( -1 );  
    }  
  
    socket_fd = socket(PF_INET,  
    if (socket_fd == -1) {  
        printf( "Error on sock  
        return( -1 );  
    }  
  
    if ( connect(socket_fd, (const struct sockaddr *)&caddr, sizeof(cad  
        printf( "Error on socket connect [%s]\n", strerror(errno) );  
        return( -1 );  
    }  
  
    value = htonl( 1 );  
    if ( write( socket_fd, &value, sizeof(value)) != sizeof(value) ) {  
        printf( "Error writing network data [%s]\n", strerror(errno) );  
        return( -1 );  
    }  
    printf( "Sent a value of [%d]\n", ntohl(value) );  
  
    if ( read( socket_fd, &value, sizeof(value)) != sizeof(value) ) {  
        printf( "Error reading network data [%s]\n", strerror(errno) );  
        return( -1 );  
    }  
    value = ntohl(value);  
    printf( "Receivd a value of [%d]\n", value );  
  
    close(socket_fd); // Close the socket  
    return( 0 );  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Putting it all together (server)



PennState

```
int server_operation( void ) {  
  
    int server, client;  
    uint32_t value, inet_len;  
    struct sockaddr_in saddr, caddr;  
  
    saddr.sin_family = AF_INET;  
    saddr.sin_port  
    saddr.sin_addr.
```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

```
server = socket  
if (server == -1) {  
    printf( "Error on socket creation [%s] ", strerror(errno) );  
    return( -1 );  
}  
  
if ( bind(server, (struct sockaddr *)&saddr, sizeof(saddr)) == -1 ) {  
    printf( "Error on socket bind [%s]\n", strerror(errno) );  
    return( -1 );  
}  
  
if ( listen( server, 5 ) == -1 ) {  
    printf( "Error on socket listen [%s]\n", strerror(errno) );  
    return( -1 );  
}  
}
```

... Together (server, part 2)



PennState

```
while ( 1 ) {
    inet_len = sizeof(caddr);
    if ( (client = accept( server, (struct sockaddr *)&caddr, &inet_len )) == -1 ) {
        printf( "Error on client accept [%s]\n", strerror(errno) );
        close(server);
        return( -1 );
    }
    printf( "Server          in_addr),                caddr.sin_port );
    if ( read( client, &value, sizeof(value)) != sizeof(value) ) {
        printf( "Error reading network data [%s]\n", strerror(errno) );
        close(server);
        return( -1 );
    }
    value = ntohl(value);
    printf( "Received a value of [%d]\n", value );
    value++;
    value = htonl(value);
    if ( write( client, &value, sizeof(value)) != sizeof(value) ) {
        printf( "Error writing network data [%s]\n", strerror(errno) );
        close(server);
        return( -1 );
    }
    printf( "Sent a value of [%d]\n", value );
    close(client); // Close the socket
}
return( 0 );
}
```