Sockets Programming

Network Application Programming Interface (API)

- The services provided by the operating system that provide the interface between application and protocol software.

Application		
Network API		
Protocol A	Protocol B	Protocol C

Network API

- Generic Programming Interface.
- Support for message oriented and connection oriented communication.
- Work with existing I/O services (when this makes sense).
- Operating System independence.

Generic Programming Interface

- Support multiple communication protocol suites (families).
- Address (endpoint) representation independence.
- Provide special services for Client and Server?

TCP/IP

- TCP/IP does not include an API definition.
- There are a variety of APIs for use with TCP/IP:
  - Sockets
  - TLI, XTI
  - Winsock
  - MacTCP

Functions needed:

- Specify local and remote communication endpoints
- Initiate a connection
- Wait for incoming connection
- Send and receive data
- Terminate a connection gracefully
- Error handling

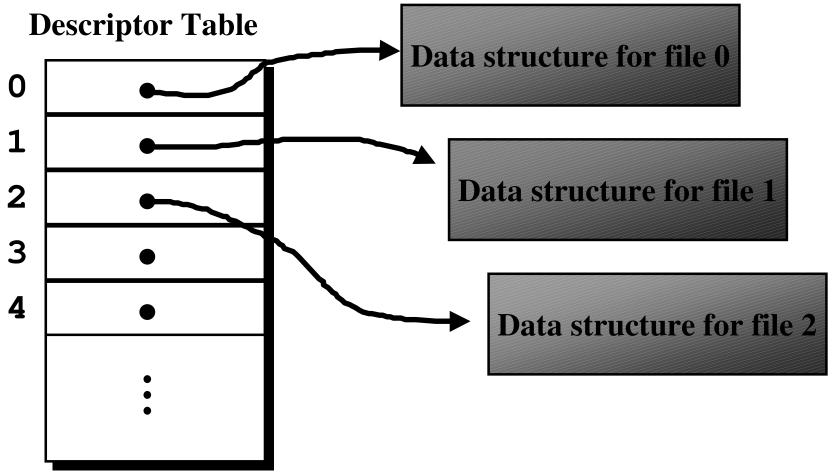
Berkeley Sockets

- Generic:
  - support for multiple protocol families.
  - address representation independence
- Uses existing I/O programming interface as much as possible.

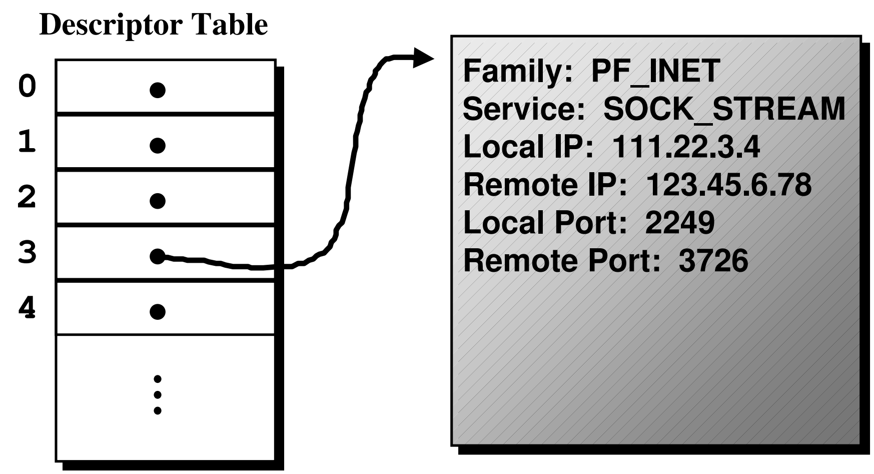
Socket

- A socket is an abstract representation of a communication endpoint.
- Sockets work with Unix I/O services just like files, pipes & FIFOs.
- Sockets (obviously) have special needs:
  - establishing a connection
  - specifying communication endpoint addresses

Unix Descriptor Table



Socket Descriptor Data Structure



Creating a Socket

```
int socket(int family,int type,int proto);
```

- `family` specifies the protocol family (PF_INET for TCP/IP).
- `type` specifies the type of service (SOCK_STREAM, SOCK_DGRAM).
- `protocol` specifies the specific protocol (usually 0 which means the default).

socket()

- The socket() system call returns a socket descriptor (small integer) or a -1 on error.
- socket() allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.

Specifying an Endpoint Address

- Remember that the sockets API is generic.
- There must be a generic way to specify endpoint addresses.
- TCP/IP requires an IP address and a port number for each endpoint address.
- Other protocol suites (families) may use other schemes.

POSIX data types

`int8_t`	signed 8bit int
`uint8_t`	unsigned 8 bit int
`int16_t`	signed 16 bit int
`uint16_t`	unsigned 16 bit int
`int32_t`	signed 32 bit int
`uint32_t`	unsigned 32 bit int

`u_char, u_short, u_int, u_long`

More POSIX data types

`sa_family_t`	address family
`socklen_t`	length of struct
`in_addr_t`	IPv4 address
`in_port_t`	IP port number

Generic socket addresses

Used by kernel

```
struct sockaddr {
  uint8_t       sa_len;
  sa_family_t   sa_family;
  char          sa_data[14];
};
```

- `sa_family` specifies the address type.
- `sa_data` specifies the address value.

sockaddr

- An address that will allow me to use sockets to communicate with my kids.
- address type `AF_DAVESKIDS`
- address values:

```
Andrea  1        Mom    5
Jeff    2        Dad    6
Robert  3        Dog    7
Emily   4
```


AF_DAVESKIDS

- Initializing a sockaddr structure to point to Robert:

```
struct sockaddr robster;

robster.sa_family = AF_DAVESKIDS;
robster.sa_data[0] = 3;
```


AF_INET

- For AF_DAVESKIDS we only needed 1 byte to specify the address.

- For AF_INET we need:
  - 16 bit port number
  - 32 bit IP address

IPv4 only!

struct sockaddr_in (IPv4)

```
struct sockaddr_in {
    uint8_t           sin_len;
    sa_family_t       sin_family;
    in_port_t         sin_port;
    struct in_addr    sin_addr;
    char              sin_zero[8];
 };
```

struct in_addr

```
struct in_addr {
   in_addr_t      s_addr;
};
```

`in_addr` just provides a name for the 'C' type associated with IP addresses.

Network Byte Order

- All values stored in a `sockaddr_in` must be in network byte order.
  - `sin_port` a TCP/IP port number.
  - `sin_addr` an IP address.

Network Byte Order Functions

'h' : host byte order 'n' : network byte order
's' : short (16bit) 'l' : long (32bit)

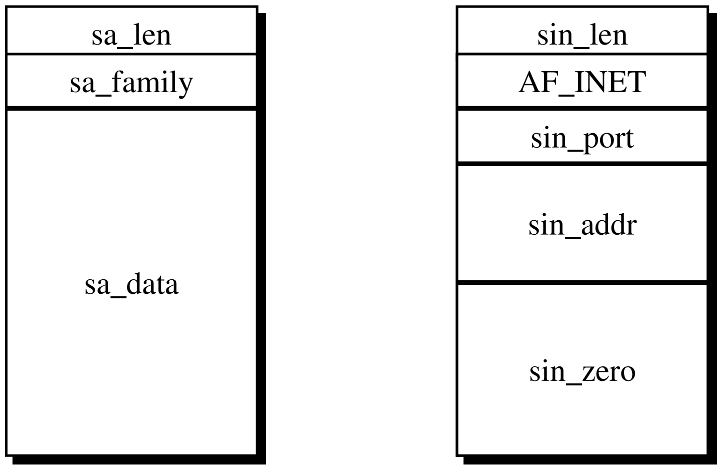
```
uint16_t htons(uint16_t);
uint16_t ntohs(uint_16_t);

uint32_t htonl(uint32_t);
uint32_t ntohl(uint32_t);
```


TCP/IP Addresses

- We don't need to deal with `sockaddr` structures since we will only deal with a real protocol family.
- We can use `sockaddr_in` structures.

BUT: The C functions that make up the sockets API expect structures of type `sockaddr`.

`sockaddr` `sockaddr_in`



Assigning an address to a socket

- The bind() system call is used to assign an address to an existing socket.

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

const!

- bind returns 0 if successful or -1 on error.

`bind()`

- calling `bind()` assigns the address specified by the `sockaddr` structure to the socket descriptor.
- You can give `bind()` a `sockaddr_in` structure:

```
bind( mysock,
      (struct sockaddr*) &myaddr,
      sizeof(myaddr) );
```


bind() Example

```
int mysock;
struct sockaddr_in myaddr;

  mysock = socket(PF_INET,SOCK_STREAM,0);
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons( portnum );
  myaddr.sin_addr = htonl( ipaddress);

  bind(mysock, &myaddr, sizeof(myaddr));
```

Uses for `bind()`

- There are a number of uses for `bind()`:
  - Server would like to bind to a well known address (port number).
  - Client can bind to a specific port.
  - Client can ask the O.S. to assign any available port number.

Port schmort - who cares ?

- Clients typically don't care what port they are assigned.
- When you call bind you can tell it to assign you any available port:

```
myaddr.port = htons(0);
```


What is my IP address ?

- How can you find out what your IP address is so you can tell `bind()` ?
- There is no realistic way for you to know the right IP address to give bind() - what if the computer has multiple network interfaces?
- specify the IP address as: `INADDR_ANY`, this tells the OS to take care of things.

IPv4 Address Conversion

`int inet_aton( char *, struct in_addr *);`

Convert ASCII dotted-decimal IP address to network byte order 32 bit value. Returns 1 on success, 0 on failure.

`char *inet_ntoa(struct in_addr);`

Convert network byte ordered value to ASCII dotted-decimal.

Other socket system calls

- General Use
  - read()
  - write()
  - close()

- Connection-oriented (TCP)
  - connect()
  - listen()
  - accept()

- Connectionless (UDP)
  - send()
  - recv()

ERROR: undefined
OFFENDING COMMAND: 0

STACK: