

# Introduction to Computer Networks

Polly Huang  
EE NTU

# Unix Network Programming

The socket  
struct and data handling  
System calls

Based on **Beej's Guide to Network Programming**

## The Unix Socket

- A file descriptor really
- The Unix fact
  - When Unix programs do any sort of I/O, they do it by reading or writing to a file descriptor
  - A file descriptor is simply an integer associated with an open file

## A File Descriptor

- A file in Unix can be
  - A network connection
  - A FIFO queue
  - A pipe
  - A terminal
  - A real on-the-disk file
  - Or just about anything else

Jeez, *everything* in Unix is a file!

Well, we know how to handle files!

- In theory
  - The `read()` and `write()` calls allows to communicate through a socket
- In practice
  - The `send()` and `recv()` offer much greater control over your data transmission

## The structs

- `int`
  - For the file descriptor
- `struct sockaddr`
  - Space holder for “types” of addresses
- `struct sockaddr_in`
  - Specific for the “Internet” type
  - `_in` for Internet
- `struct in_addr`
  - 4 byte IP address

## `struct sockaddr`

```
struct sockaddr {
    unsigned short sa_family;
        // address family, AF_XXX
    char sa_data[14];
        // 14 bytes of protocol address
};
```

## struct sockaddr\_in

```
struct sockaddr_in {
    short int sin_family;
        // Address family unsigned, AF_INET
    short int sin_port;
        // Port number, in network byte order
    struct in_addr sin_addr;
        // Internet address, in network byte order
    unsigned char sin_zero[8];
        // Same size as struct sockaddr
};
```

Polly Huang, NTU EE

9

## Struct in\_addr

```
struct in_addr {
    // Internet address (a structure for historical reasons)
    unsigned long s_addr;
        // that's a 32-bit long, or 4 bytes
};
```

Polly Huang, NTU EE

10

## Reference

- Let *ina* be of type `struct sockaddr_in`
- *ina.sin\_addr.s\_addr* references the 4-byte IP address in network byte order

## Types of Byte Ordering

- Network Byte Order
  - Most significant byte first
  - Need conversion from the app program to the network
- Host Byte Order
  - Least significant byte first
  - Usually no need in app program
  - But need conversion if data coming from the network

## Functions to Convert

- `htons()`
  - Host to Network Short
- `htonl()`
  - Host to Network Long
- `ntohs()`
  - Network to Host Short
- `ntohl()`
  - Network to Host Long

## Storing the IP address

```
ina.sin_addr.s_addr = inet_addr("10.12.110.57");
```

- Returns “-1” on error
- For unsigned short it’s 255.255.255.255
- A broadcast address

## A Cleaner Interface

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

## An Example

```
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
```



## Things to Note

- `inet_addr()` and `inet_aton()` both convert IP addresses into the network byte order
- Not all platforms implement `inet_aton()`

## Get the IP Address Back

- `printf("%s", inet_ntoa(ina.sin_addr));`
- `inet_ntoa()` returns a pointer to a `char*`
- And...

## Use strcpy()

```
char *a1, *a2; . .
a1 = inet_ntoa(ina1.sin_addr);
    // this is 192.168.4.14
a2 = inet_ntoa(ina2.sin_addr);
    // this is 10.12.110.57
printf("address 1: %s\n",a1);
printf("address 2: %s\n",a2);
```

This program will print:

```
address 1: 10.12.110.57
address 2: 10.12.110.57
```

Polly Huang, NTU EE

19

## Quiz Time!

20

# System Calls

## socket()

### Creating the File Descriptor

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

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

domain: AF\_INET

type: SOCK\_STREAM or SOCK\_DGRAM

protocol: 0 or getprotobyname()

## bind()

### Associating Port with the FD

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

## Example (Typical Server)

```
#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); // do some error checking!
    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));
    ...
}
```

## connect()

### Making a Connection

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

### Example (Typical Client)

```
#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));
    ...
}
```

## listen()

### Waiting for Connection

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

On the server side, you see typically this:

```
socket();
bind();
listen();
/* accept() goes here */
```

## accept()

### Getting a Connection

```
#include <sys/socket.h>
int accept(int sockfd, void *addr, int *addrlen);
```

## The Server Example

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

#define MYPORT 3490 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold
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); // do some error checking!
    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);

```

Polly Huang, NTU EE

29

## send() and recv() Data Transmission

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

Polly Huang, NTU EE

30

## Example

```
char *msg = "Hello World!";  
int len, bytes_sent;  
..  
len = strlen(msg);  
bytes_sent = send(sockfd, msg, len, 0);  
...
```

Quiz Time!



## sendto() and recvfrom() Transmission the Datagram Style

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

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

Or if transmitting over **TCP socket**,  
one can simply use **send()** and **recv()**.

## close() and shutdown() Closing the Communication

```
close(sockfd);
```

```
int shutdown(int sockfd, int how);
```

- 0 – Further receives are disallowed
- 1 – Further sends are disallowed
- 2 – Further sends and receives are disallowed (like close())

## Reference

- **Beej's Guide to Network Programming**
  - <https://beej.us/guide/bgnet/>
- Additional system calls
- TCP stream client, server example
- UDP datagram example