LEC03.
SOCKET API INTRODUCTION

Bui Trong Tung, SoICT, HUST

1

1

Content

- Socket
- Stream Socket
- Datagram Socket
- APIs for managing names and IP addresses
- Socket Address Structures

2

2

Socket

- What is a socket ?
- *Sockets* (in plural) are an application programming interface (API) application program and the TCP/IP stack
- A *socket* is an abstraction through which an application may send and receive data
- A socket allows an application to plug in to the network and communicate with other applications that are plugged into the same network.

3

3

Socket (cont)

- The main types of sockets in TCP/IP are
  - *stream sockets :* use TCP as the end-to-end protocol (with IP underneath) and thus provide a reliable byte-stream service
  - *datagram sockets :* use UDP (again, with IP underneath) and thus provide a **best-effort** datagram service
  - *raw socket:* allows direct sending and receiving of Internet Protocol packets without any protocol-specific transport layer formatting.
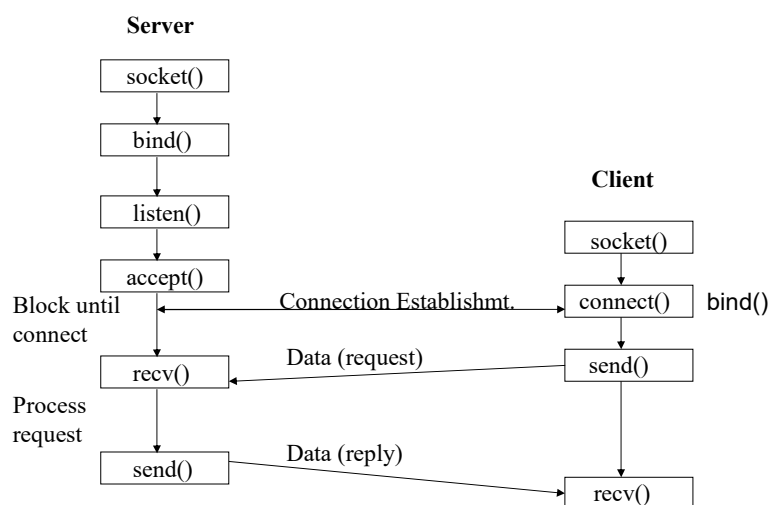- Socket Address(excluding raw socket) : include host name and port

4

4

Stream sockets (TCP)

- TCP provides connections between clients and servers
- TCP also provides reliability : When TCP sends data to the other end, it requires an acknowledgment in return
- TCP provides flow control
- TCP connection is full-duplex

5

5

Stream sockets(TCP)

Server

```
socket()
  ↓
bind()
  ↓
listen()
  ↓
accept()
```

Block until connect

Process request

```
recv()
  ↓
send()
```

Client

```
socket()
  ↓
connect()    bind()
  ↓
send()
  ↓
recv()
```

Connection Establishmt.

Data (request)

Data (reply)

6

6

Stream Socket APIs

- socket()
  - creates a socket of a given domain, type, protocol (buy a phone)
  - Returns a file descriptor (called a socket ID)
- bind()
  - Assigns a name to the socket (get a telephone number)
  - Associate a socket with an IP address and port number (Eg : 192.168.1.1:80)
- connect()
  - Client requests a connection request to a server
  - This is the first of the client calls

7

7

Stream Socket APIs (cont)

- accept() :
  - Server accept an incoming connection on a listening socket (request from a client)
  - There are basically three styles of using accept:
    - *Iterating server*: Only one socket is opened at a time.
    - *Forking server*: After an accept, a child process is forked off to handle the connection.
    - *Concurrent single server*: use select to simultaneously wait on all open socketIds, and waking up the process only when new data arrives

8

8

Stream Socket APIs (cont)

- listen()
  - Specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)
- send()
  - Write to connection (speak)
  - Send a message
- recv()
  - read from connection (listen)
  - Receive data on a socket
- close()
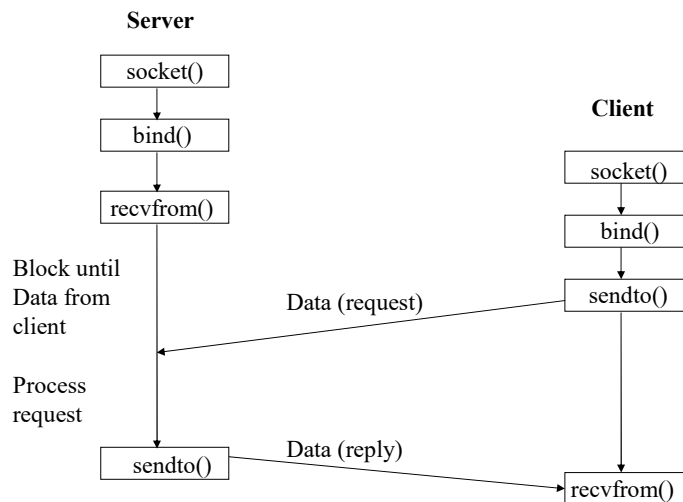 - close a socket (end the call)

9

9

Datagram Socket (UDP)

- UDP is a simple transport-layer protocol
- If a datagram is errored or lost, it won't be automatically retransmitted (can process in application)
- UDP provides a *connectionless* service, as there need not be any long-term relationship between a UDP client and server

10

10

Datagram Socket (UDP)

Server

socket()

bind()

recvfrom()

Block until
Data from
client

Process
request

sendto()

Client

socket()

bind()

sendto()

recvfrom()

Data (request)

Data (reply)

11

11

APIs for managing names and IP addresses

- `gethostname()`: Returns the name of the system
- `gethostbyname()` : Get an IP address for a hostname, or vice-versa
- `htons(), htonl(), ntohs(), ntohl()`: *byte ordering*
- `inet_ntoa(), inet_aton()` : Convert IPv4 addresses from a dots-and-number string (eg : 192.168.1.1) to a struct in_addr and back
- `inet_pton(), inet_ntop()`: conversion of IPv4 or IPv6 numbers between presentation and strings

12

12

Byte Ordering

- There are two ways to store the two bytes in memory
  - little-endian byte order
  - big-endian byte order
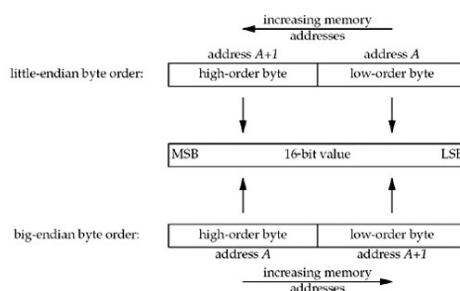
13

13

Byte Ordering (cont)

- There is no standard between these two byte orderings
- A variety of systems that can change between little-endian and big-endian byte ordering
- Problem : Converting between
  - *host byte order*
  - *network byte order* (The Internet protocols use big-endian byte ordering)
- Four functions to convert between these two byte orders.

14

14

htons(), htonl(), ntohs(), ntohl()

- Convert multi-byte integer types from host byte order to network byte order

```
#include <netinet/in.h>
uint32_t htonl(u_long hostlong);  /* host to network long */
uint16_t htons(u_short hostshort);/* host to network short */
uint32_t ntohl(u_long netlong);   /* network to host long */
uint16_t ntohs(u_short netshort); /* network to host short */
```

- Each function returns the converted value.

15

15

IP Number translation

- IP address strings to 32 bit number
- Hence, these routines translate between the address as a string and the address as the number.
- Hence, we have 4 representations:
  - IP number in host order
  - IP number in network order
  - Presentation (eg. dotted decimal)
  - Fully qualified domain name

16

16

Socket Address Structures

- Most socket functions require a pointer to a socket address structure as an argument.
- Each supported protocol suite defines its own socket address structure.
- A Socket Address Structure is a structure which has information of a socket to create or connect with it
- There are three types of socket address structures
  - IPv4
  - IPv6

17

17

IPv4 socket address structure

```
#include <netinet/in.h>
struct in_addr {
    in_addr_t s_addr;           /* 32-bit IPv4 address */
                                /* network byte ordered */
};

struct sockaddr_in {
    uint8_t sin_len;            /* length of structure */
    sa_family_t sin_family;     /* AF_INET */
    in_port_t sin_port;         /* 16-bit TCP or UDP port number
                                network byte ordered */
    struct in_addr sin_addr;    /* 32-bit IPv4 address network byte
                                ordered */
    char sin_zero[8];           /* unused */
};
```

18

18

IPv6 socket address structure

```c
#include <netinet/in.h>
struct in6_addr {
   uint8_t s6_addr[16];        /* 128-bit IPv6 address */
                               /* network byte ordered */
};

#define SIN6_LEN                /* required for compile-time tests  */

struct sockaddr_in6 {
   uint8_t sin6_len;           /* length of this struct */
   sa_family_t sin6_family;    /* AF_INET6 */
   in_port_t sin6_port;        /* transport layer port# network byte
                                  /* ordered */
   uint32_t sin6_flowinfo;     /* flow information, undefined */
   struct in6_addr sin6_addr;  /* IPv6 address network byte ordered */
   uint32_t sin6_scope_id;     /* set of interfaces for a scope */
};
```

19

19

inet_aton()

```c
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp)
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr`
- Return:
  - The value non-zero if the address is valid
  - The value 0 if the address is invalid

```c
struct in_addr someAddr;
if(inet_aton("10.0.0.1", &someAddr))
   printf("The address is valid");
else printf ("The address is invalid");
```

20

20

[[deprecated]]`inet_ntoa()`

```
#include <arpa/inet.h>
char *inet_ntoa(struct in_addr in);
```

- Convert IP addresses from a struct `in_addr` to a dots-and-number string
- Return: the dots-and-numbers string

```
struct in_addr someAddr;
if(inet_aton("10.0.0.1", someAddr))
    printf("The address is valid");
else printf ("The address is invalid");
char *addrStr;
addrStr = inet_ntoa(someAddr);
```

21

21

`inet_addr()`

```
#include <arpa/inet.h>
in_addr_t inet_addr(const char *cp);
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr_t`
- Return:
  - The value -1 if there's an error
  - The address as an `in_addr_t`

```
struct in_addr someAddr;
someAddr.s_addr = inet_addr("10.0.0.1");
```

22

22

inet_pton()

```
#include <arpa/inet.h>
int inet_pton(int family, const char *cp, void *addr)
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr` or `in6_addr`
- `family` is `AF_INET` or `AF_INET6`
- Return:
    - The value non-zero if the address is valid
    - The value 0 if the address is invalid

23

23

inet_ntop()

```
#include <arpa/inet.h>
const char *inet_ntop(int family, const void *addr,
                                    char *cp, size_t len);
```

- Convert IP addresses from a struct `in_addr` to a dots-and-number string
- Return: the dots-and-numbers string

```
struct sockaddr_in sa;
char str[INET_ADDRSTRLEN];

// store this IP address in sa:
inet_pton(AF_INET, "192.0.2.33", &(sa.sin_addr));

// now get it back and print it
inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);
printf("%s\n", str);
```

24

24

ADDRESS RESOLUTION

25

Content

- IPv4 and IPv6
- DNS
- Address and Name APIs

26

IPv4

- Developed in APRANET (1960s)
- 32-bit number
- Divided into classes that describe the portion of the address assigned to the network (netID) and the portion assigned to endpoints (hosten)
  - A : netID – 8 bit
  - B : netID – 16 bit
  - C : netID – 24 bit
  - D : use for multicast
  - E : use for experiments

27

27

IPv4 problem

- IPv4 addresses is being exhausted
- Have to map multiple private addresses to a single public IP addresses (NATs)
  - Connect 2 PCs use private address space ?
  - NAT must be aware of the underlying protocols
- IPv4 addressing is not entirely hierarchical → router must maintain routing table to deliver packets to right locations
- → Develope a new version of IP Address : IPv6

28

28

IPv6

- IPv6 address is 128 bits
  - To subdivide the available addresses into a hierarchy of routing domains that reflect the Internet's topology
- IPv6 address is typically expressed in 16-bit chunks displayed as hexadecimal numbers separated by colons
  Example : 21DA:00D3:0000:2F3B:02AA:00FF:FE28:9C5A

  or : 21DA:D3:0:2F3B:2AA:FF:FE28:9C5A

29

29

DNS (Domain Name System)

- Computers use IP Addresses to connect hosts
  - What about humans ? – IP Addresses are very complex and hard to remember (for people)
- Use name instead of IP Address → Domain Name System
- Problem of DNS
  - People use names, Computers use IP Addresses → translate between two spaces
  - Domain name system must be hierarchical (for management and maintain)
- Domain name space : divide to zones

30

30

DNS (cont)

- How to translate between domain name-IP Address and reverse ?
  - DNS Resolver
  - DNS Server
- A DNS query
  - A *non-recursive query* : DNS server provides a record for a domain for which it is authoritative itself, or it provides a partial result without querying other servers
  - A *recursive query* : DNS server will fully answer the query by querying other name servers
- DNS primarily uses User Datagram Protocol (UDP) on port number 53 to serve requests

31

31

DNS (cont)

- Address resolution mechanism
  - Local system is pre-configured with the known addresses of the root server in a file of *root hints*
  - Query one of the root servers to find the server authoritative for the next level down
  - Querying level down server for the address of a DNS server with detailed knowledge of the lower level domain until reach the DNS
  - Server return final address
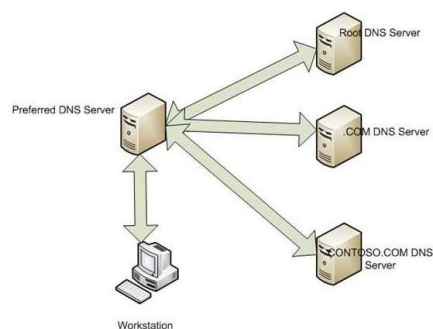
32

32

DNS (cont)

- A *Resource Record* (RR) is the basic data element in the domain name system
- All records use the common format specified in RFC 1035 (in IP networks)
- **RR (Resource record) fields**
  - NAME (variable)
    - Name of the node to which this record pertains.
  - TYPE (2)
    - Type of RR. For example, MX is type 15
  - CLASS (2)
    - Class code
  - TTL (4)
    - Unsigned time in seconds that RR stays valid
  - RDLENGTH (2)
    - Length of RDATA field
  - RDATA (variable)
    - Additional RR-specific data

33

33

List of Address and Name APIs

#include <sys/socket.h>
- **gethostbyaddr()**
  - Retrieve the name(s) and address corresponding to a network address.
- **gethostname()**
  - Retrieve the name of the local host.
- **gethostbyname()**
  - Retrieve the name(s) and address corresponding to a host name.
- **getprotobyname()**
  - Retrieve the protocol name and number corresponding to a protocol name.
- **getprotobynumber()**
  - Retrieve the protocol name and number corresponding to a protocol number.
- **getservbyname()**
  - Retrieve the service name and port corresponding to a service name.
- **getservbyport()**
  - Retrieve the service name and port corresponding to a port.

34

34

New APIs for IPv6

- Those APIs only supports IPv4 but IPv6 will be replace IPv4 in the future, so we need APIs support IPv6
- They are
  - getaddrinfo
  - getnameinfo
- These APIs have replaced the IPv4 specific routines

35

[[deprecated]] gethostbyaddr()

```
#include <netdb.h>
#include <sys/socket.h>
struct hostent *gethostbyaddr (in_addr *addr, socklen_t len,
                                int family);
```

- Get host information corresponding to an address.
- Parameters:
  - [IN] addr: A pointer to an address in network byte order.
  - [IN] len: The length of the address, which must be 4 for AF_INET addresses.
  - [IN] family: The type of the address, which must be AF_INET.
- Return value
  - If no error occurs, returns a pointer to the hostent structure
  - Otherwise it returns a NULL pointer and a specific error number

36

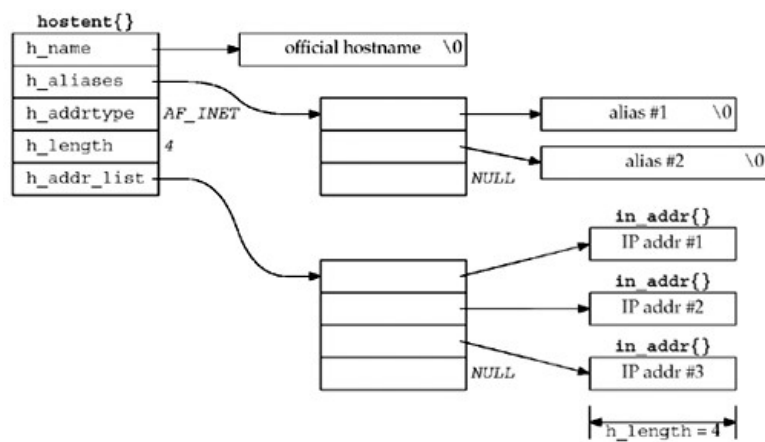## struct hostent

```
struct hostent {
    char    *h_name;        /* official (canonical) name of host*/
    char    **h_aliases;    /* pointer to array of pointers to
                               alias names  */
    int      h_addrtype;    /* host address type: AF_INET  */
    int      h_length;      /* length of address: 4 */
    char    **h_addr_list;  /* ptr to array of ptrs with IPv4
                               addrs */
};
```

37

37

struct hostent

38

38

19

gethostname()

```
#include <sys/unistd.h>
#include <sys/socket.h>
int gethostname(char *name, size_t len);
```

- Return the standard host name for the local machine.
- Parameters:
  - [OUT] name: points to a buffer that will receive the host name.
  - [IN] len: the length of the buffer
- Return value
  - If no error occurs, returns 0
  - Otherwise it returns SOCKET_ERROR and a specific error code

39

39

[[deprecated]] gethostbyname()

```
#include <netdb.h>
#include <sys/socket.h>
struct hostent *gethostbyname (const char *hostname);
```

- Get host information corresponding to a hostname.
- [IN] name: Points to the name of the host
- Returns a pointer to a hostent structure
- Return value
  - If no error occurs, returns a pointer to the hostent structure described above.
  - Otherwise it returns a NULL pointer and a specific error number

40

40

getaddrinfo()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(const char *node, const char *service, const
struct addrinfo *hints, struct addrinfo **res);
```

- Get host information corresponding to a `node` and `service`.
- [IN] `node`: name of the host
- [IN] `service`: name of the service
- [IN] `hint`: limit the set of uddresses returned by `getaddrinfo()`
- [OUT] `res`: points to linked-list containing host information
- Return value
  - If no error occurs, returns 0
  - Otherwise it returns error code

41

41

freeaddrinfo()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int freeaddrinfo(struct addrinfo *res);
```

- Frees the memory that was allocated for the dynamically allocated linked list `res`.

gai_strerror()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
const char* gai_strerror(int errcode);
```

- Translates these error codes to a human readable string

42

42

struct `addrinfo`

```c
typedef struct addrinfo {
  int        ai_flags;      /*additional options*/
  int        ai_family;     /*desired address family for the
                             returned addresses*/
  int        ai_socktype;   /*preferred socket type*/
  int        ai_protocol;   /*protocol for the returned socket
                             addresses */
  size_t     ai_addrlen;    /*length of ai_addr */
  char          *ai_canonname; /*canonical name */
  struct sockaddr  *ai_addr;  /*binary address */
  struct addrinfo  *ai_next;  /*next structure in linked list*/
};
```

- `ai_family`: `AF_INET, AF_INET6` or `AF_UNSPEC`
- `ai_socktype`: `SOCK_STREAM, SOCK_DGRAM` or 0 for any type
- `ai_protocol`: 0 for any protocol

43

43

Example

```c
struct addrinfo* result;
struct sockaddr_in  *address;
int error;
char ipStr[INET_ADDRSTRLEN];

/* resolve the domain name into a list of addresses */
error = getaddrinfo("facebook.com", NULL, NULL, &result);
if (error != 0) {
   if (error == EAI_SYSTEM) {
      perror("getaddrinfo");
   }
   else {
      fprintf(stderr, "error: %s\n", gai_strerror(error));
   }
   exit(EXIT_FAILURE);
}
address = (struct sockaddr_in *) result->ai_addr;
inet_ntop(AF_INET, &address->sin_addr, ipStr, sizeof(ipStr));
printf("IPv4 address: %s\n", ipStr);

freeaddrinfo(result);
```

44

44

getnameinfo()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getnameinfo(const struct sockaddr *sa, socklen_t salen, char
*node, socklen_t nodelen, char *service, socklen_t servicelen, int
flag);
```

- Converts a socket address to a corresponding host and service.
- [IN] sa: points to a socket address structure
- [OUT] node: points to a buffer able to contain hostname
  - node = NULL, nodelen = 0: hostname shall not be returned
- [OUT] service: points to a buffer able to contain service name
  - service = NULL, servicelen = 0: service name shall not be returned
- [INT] flag: changes the default actions of the function
- Return value
  - If no error occurs, returns 0
  - Otherwise it returns error code

45

45

Example

```
struct sockaddr_in addr;
int ret;
char hostname[NI_MAXHOST];
char servInfo[NI_MAXSERV];
addr.sin_family = AF_INET;
inet_pton(AF_INET, "8.8.8.8", &addr.sin_addr);

ret = getnameinfo((struct sockaddr *) &addr, sizeof(struct
sockaddr), hostname, NI_MAXHOST, servInfo, NI_MAXSERV, 0);

if (ret != 0)
   printf("Failed: %s\n", gai_strerror(ret));
else
   printf("Hostname: %s\nService: %s\n", hostname, servInfo);
```

46

46

getservbyname()

```
#include <netdb.h>
#include <sys/socket.h>
struct servent *getservbyname (const char *servname,
                                     const char *protoname);
```

- Get service information corresponding to a service name and protocol.
- Parameters:
  - [IN] `servname`: A pointer to a service name.
  - [IN] `protoname`: An optional pointer to a protocol name.
    - If this is NULL, getservbyname() returns the first service entry for which the name matches the `s_name` or one of the `s_aliases`.
    - Otherwise getservbyname() matches both the name and the proto.
- Returns
  - non-null pointer if OK
  - NULL on error

```
struct servent *sptr;
sptr = getservbyname("ftp", "tcp");
```

47

47

struct servent

```
struct servent {
        char *s_name;
        char **s_aliases;
        int s_port;
        char *s_proto;
};
```

- `s_name`
  - Official name of the service.
- `s_aliases`
  - A NULL-terminated array of alternate names.
- `s_port`
  - The port number at which the service may be contacted. Port numbers are returned in network byte order.
- `s_proto`
  - The name of the protocol to use when contacting the service.

48

48