ECE453 – Introduction to Computer Networks
Lecture 15 – Transport Layer (II)

Transport Services
- Connection-oriented Service
  - TCP
    - Reliable data transfer
    - TCP flow control
    - TCP congestion control
    - TCP connection management
- Connectionless Service
  - UDP

Through socket programming

Socket Programming
TCP Socket Programming Flow

Server

- `socket()`
- `listen()` (Block until connection from client)
- `accept()` (Process requests)
- `read()`
- `write()`
- `close()`

Client

- `socket()`
- `connect()`
- `read()`
- `write()`
- `close()`

UDP Socket Programming Flow

Server

- `socket()`
- `bind()`
- `recvfrom()`

Client

- `socket()`
- `sendto()`
- `recvfrom()`

Socket APIs

- `result = socket(pf, type, protocol)`
- `close(socket)`
- `bind(socket, localaddr, addrlen)`
Create A Socket

\[
\text{result = socket(pf, type, protocol)}
\]

- **PF**: protocol family, example, PF_INET (TCP/IP), PF_UNIX (IPC, local)
- **Type**: specify the type of communication (service model) desired, example, SOCKET_STREAM (TCP), SOCKET_DGRAM (UDP), SOCKET_RAW (bypass TCP/UDP, use IP directly)
- **Protocol**: select a specific protocol within the family. It is needed when there are more than one protocol to support the type of service desired. Usually is 0

Close A Socket

\[
\text{close(socket)}
\]

- **Socket**: specify the descriptor of socket to be closed
- Internally, a call to close() decrements the reference count for a socket and destroys the socket if the count reaches zero
- Socket can be used by multiple applications

Specify Local Address

\[
\text{bind(socket, localaddr, addrlen)}
\]

- **localaddr**: a structure that specifies the local address to which the socket should be bound
- **addrlen**: an integer that specifies the length of address measured in bytes
Structure of Local Address

```
struct sockaddr_in {
    short sin_family; /* must be AF_INET */
    u_short sin_port; /* protocol port, can ignore */
    struct in_addr sin_addr; /* IP address */
    char sin_zero[8]; /* Not used, must be zero */
};
```

```
struct in_addr {
    in_addr_t s_addr; /* 32 bit ipv4 address, network byte ordered */
};
```

Connect Sockets to Destination

```
connect(socket, destaddr, addrlen)
```

- `destaddr`: a socket address structure that specifies the destination address to which a socket should be bound.

Specify A Queue Length for A Server

```
listen(socket, qlength)
```

- `listen()` allows server to prepare a socket for incoming connections, and it puts the socket in a passive mode ready to accept connections.
- It also tells server to en-queue up to “qlength” requests for connections, when full, discard new requests.
- `listen()` only applies to sockets that have selected reliable data delivery services.
Accepts Connections

\[ \text{newsock=accept(socket, addr, addrlen)} \]

- A call to accept() blocks until a connection request arrives.
- When the request arrives, the system fills in argument “addr” with the address of the client that has placed the request, and “addrlen” to the length of the address.
- Return a new socket with its destination connected to the requesting client.
- The original socket still has a wildcard foreign destination address, and it still remains open.

Accepts Connection: Illustration

Send Data Through Socket

- `write(socket, buffer, length)`
- `writev(socket, iovector, vectorlen)`
- `send(socket, message, length, flags)`
- `sendto(socket, message, length, flags, Destaddr, addrlen)`
- `sendmsg(socket, messagestruct, flags)`
Receive Data Through Socket

- `read(socket, buffer, length)`
- `readv(socket, iovector, vectorlen)`
- `recv(socket, message, length, flags)`
- `recvfrom(socket, message, length, flags, Destaddr, addrlen)`
- `recvmsg(socket, messagestruct, flags)`

Byte Ordering

- Little-endian byte order: High-order byte < Low-order byte
- Big-endian byte order: Old-order byte > High-order byte

Increasing memory address

Address A

High-order byte

Low-order byte

MSB 16-bit value LSB

Implications of Byte Order

- Unfortunately there is no standard between these two byte orderings and we encounter systems that use both formats.
- We refer to the byte ordering used by a given system as host byte order.
- The sender and the receiver must agree on the order in which the bytes of these multi-byte field transmitted: specify network byte order, which is big-endian byte ordering.
Byte Order Functions

```c
#include <netinet.h>

uint16_t htons(uint16_t host16bitvalue)
Converts a 16-bit integer from host to network byte order

uint32_t htonl(uint32_t host32bitvalue)
Converts a 32-bit integer from host to network byte order

Both return: value in network byte order

uint16_t ntohs(uint16_t net16bitvalue)
uint32_t ntohl(uint32_t net32bitvalue)
Both return: value in host byte order
```

Byte Manipulation Functions

```c
#include <strings.h>
/* Berkeley-derived functions */

void bzero(void *dest, size_t nbytes)
Set the first part of an object to null bytes

void bcopy(const void *src, void *dest, size_t nbytes);

int bcmp(const void *ptr1, const void *ptr2, size_t nbytes)
/* return:0 if equal, nonzero if unequal */
```

```c
#include <string.h>
/* ANSI C defined functions */

void *memset(void *dest, int c, size_t len)
Sets the first len bytes in memory dest to the value of c

void *memcpy(void *dest, const void *src, size_t nbytes)

void memcmp(const void *ptr1, const void *ptr2, size_t nbytes)
```

Address Conversion Functions

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

int inet_aton(const char *strptr, struct in_addr *addrptr); /* return 1 if string was valid, 0 error */
Convert an IP address in string format (x.x.x.x) to the 32-bit packed binary format used in low-level network functions

in_addr_t inet_addr(const char *strptr);
/* return 32-bit binary network byte ordered IPv4 address; INADDR_NONE if error, deprecated and replaced by inet_aton() */

char *inet_ntoa(struct in_addr inaddr);
/* returns: pointer to dotted-decimal string */
```
/* A simple server in the internet domain using TCP */

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

int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
    
    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    bzero((char *) &serv_addr,sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    bzero(buffer,256);
    n = read(newsockfd,buffer,255);
    printf("Here is the message: %s\n",buffer);
    n = write(newsockfd,"I got your msg",18);
    return 0;
}

/* Client program */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];
    portno = atoi(argv[2]);
    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    server = gethostbyname(argv[1]);
    bzero((char *) &serv_addr,sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,
    server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    printf("%s\n",buffer);
    return 0;
}