CSCI4430 Data Communication and Computer Networks

Socket Programming for TCP

HAN Shujie

sjhan@cse.cuhk.edu.hk

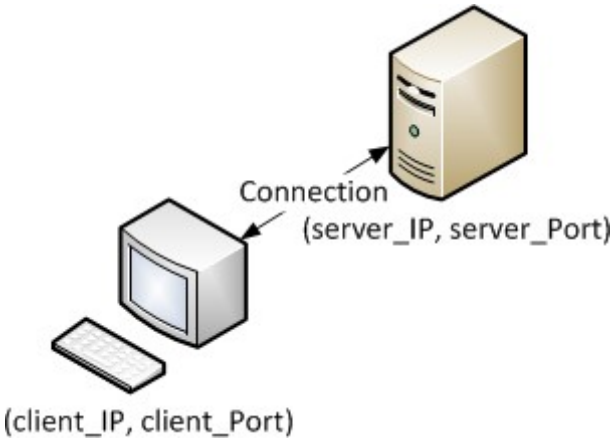
Jan. 16, 2020

Outline

- What is TCP?
- What is socket?
- Overview of workflow
- Multiple clients
- Basic socket programming
  - Network byte ordering

What is TCP?

- A connection-oriented transport layer protocol
- How to transfer data via TCP connection
  - Create a connection (port to port connection)
  - Transfer data
- Reliable delivery of data

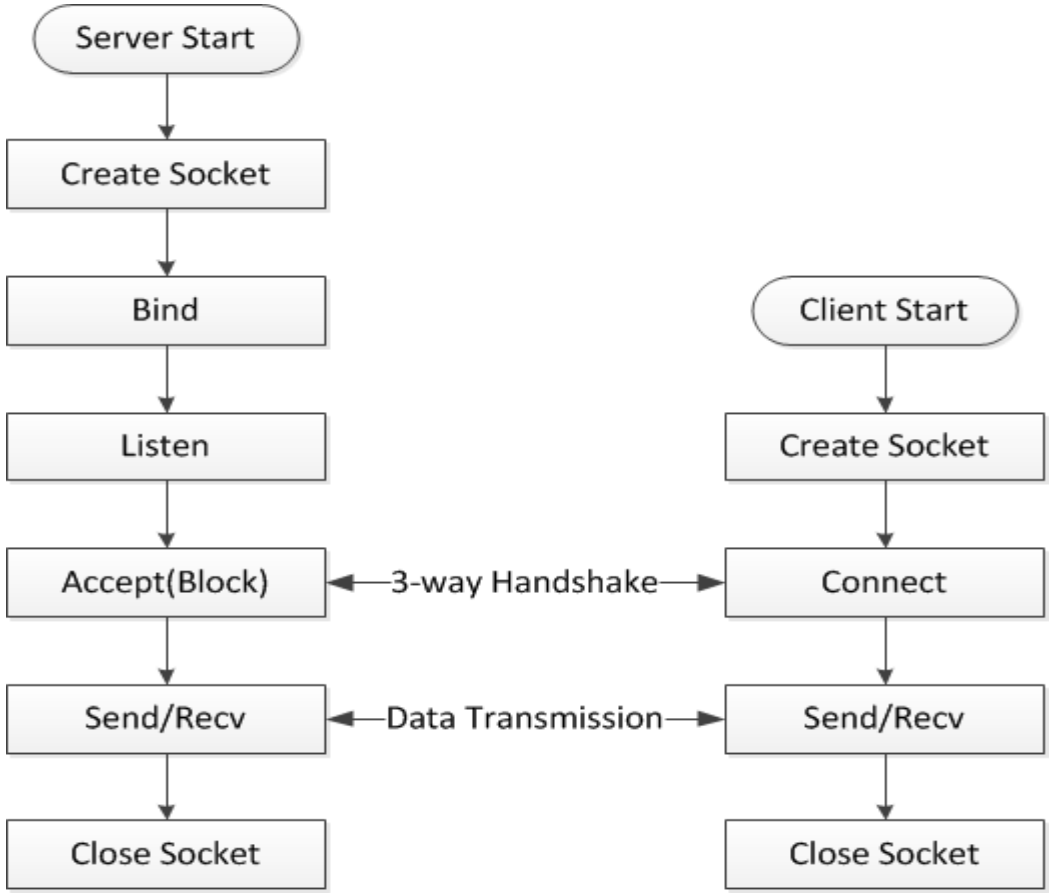


What is Socket?

- Just something which will make sending/receiving data through network feel like reading/writing files.
- How to use?
  - Create a socket
  - Bind the socket to a connection
  - Send/receive data
  - Close

Overview of Workflow

- Network programming for TCP



Overview of Workflow

- Before client contacts server:
  - server process must first be running
  - server must have created socket (door) that welcomes client's contact
- Client contacts server by:
  - creating client-local TCP socket
  - specifying **IP address, port number** of server process
  - When client creates socket, client TCP establishes connection to server TCP

Multiple Clients

- How to support multiple clients communication for a server?
- When contacted by client, server creates new TCP socket for server process to communicate with client
  - allows server to talk with multiple clients
  - **source port numbers used to distinguish clients**

Basic Socket Programming

- Create a socket(Server & Client)
  - Syntax:
    - int socket(int family, int type, int protocol);
  - It is like open() function for file operations, the return value is a socket descriptor
  - Example:

```
int sd=socket(AF_INET,SOCK_STREAM,0);
```

Basic Socket Programming

- Create a socket(Server & Client)
  - Family: protocol family
    - AF_INET: ipv4 protocols
    - AF_INET6: ipv6 protocols
  - Type:
    - SOCK_STREAM: stream socket (for TCP)
    - SOCK_DGRAM: datagram socket (for UDP)
    - SOCK_PACKET: raw socket for Linux
  - Protocol: set to 0 except raw socket.
  - Return value: non-negative integer for success and -1 for failures.

Basic Socket Programming

- bind() (Server)
  - Bind the socket to a port in the local machine
  - Syntax
    - int bind(int socket, struct sockaddr* addr, int addr_len);
  - Example:

```
struct sockaddr_in server_addr;
memset(&server_addr,0,sizeof(server_addr));
server_addr.sin_family=AF_INET;
server_addr.sin_addr.s_addr=htonl(INADDR_ANY);
server_addr.sin_port=htons(PORT);
if(bind(sd,(struct sockaddr *) &server_addr,sizeof(server_addr))<0){
        printf("bind error: %s (Errno:%d)\n",strerror(errno),errno);
        exit(0);
}
```

Basic Socket Programming

- bind() (Server)
  - struct sockaddr defines a socket address
  - INADDR_ANY the socket accept connections to ANY IP addresses of this host.

```
struct sockaddr_in server_addr;
memset(&server_addr,0,sizeof(server_addr));
server_addr.sin_family=AF_INET;
server_addr.sin_addr.s_addr=htonl(INADDR_ANY);
server_addr.sin_port=htons(PORT);
if(bind(sd,(struct sockaddr *) &server_addr,sizeof(server_addr))<0){
        printf("bind error: %s (Errno:%d)\n",strerror(errno),errno);
        exit(0);
}
```

Basic Socket Programming

- listen() (Server)
  - Defines the maximum number of connections can be pending to this socket
  - Syntax:
    - int listen (int socket, int backlog);
  - Example:

```
if(listen(sd,3)<0){
        printf("listen error: %s (Errno:%d)\n",strerror(errno),errno);
        exit(0);
}
```

Basic Socket Programming

- accept() (Server)
  - Returns socket descriptors for new connection
  - Syntax
    - int accept(int socket, struct sockaddr *addr, int addr_len);
  - Example

```
struct sockaddr_in client_addr;
int addr_len=sizeof(client_addr);
if((client_sd=accept(sd,(struct sockaddr *) &client_addr,&addr_len))<0){
    printf("accept erro: %s (Errno:%d)\n",strerror(errno),errno);
    exit(0);
}
```

Basic Socket Programming

- accept() (Server)
  - The received value is a socket descriptor
  - You may find that there are 2 socket descriptors, *sd* and *client_sd*
    - *sd* is for listening to a port and accepting connections from clients
    - The descriptor *client_sd* returned by accept() is used for transferring data.

```
struct sockaddr_in client_addr;
int addr_len=sizeof(client_addr);
if((client_sd=accept(sd,(struct sockaddr *) &client_addr,&addr_len))<0){
    printf("accept erro: %s (Errno:%d)\n",strerror(errno),errno);
    exit(0);
}
```

Basic Socket Programming

- connect() (Meanwhile on the client side…)
  - Connect a TCP client to a TCP server
  - Syntax:
    - int connect(int socket, struct sockaddr *addr, int *addr_len);
  - Example:

```
int sd=socket(AF_INET,SOCK_STREAM,0);
struct sockaddr_in server_addr;
memset(&server_addr,0,sizeof(server_addr));
server_addr.sin_family=AF_INET;
server_addr.sin_addr.s_addr=inet_addr(argv[1]);
server_addr.sin_port=htons(PORT);
if(connect(sd,(struct sockaddr *)&server_addr,sizeof(server_addr))<0){
        printf("connection error: %s (Errno:%d)\n",strerror(errno),errno);
        exit(0);
}
```

Basic Socket Programming

- gethostbyname()
  - convert the hostname into an IP address

```
struct hostent * he;
struct in_addr ** addrList;
he=gethostbyname(argv[1]);
if(he==NULL){
    perror("gethostbyname()");
    exit(2);
}
printf("Host Name:%s\n",he->h_name);
printf("IP Address(es):\n");
addrList=(struct in_addr**)he->h_addr_list;
int i;
for(i=0;addrList[i]!=NULL;i++){
    printf(" %s\n",inet_ntoa(*addrList[i]));
}
```

Basic Socket Programming

- send()/recv() (Server & Client)
  - Send and receive data
  - Syntax:
    - int send (int socket, const void* msg, size_t msg_len, int flags);
    - int recv (int socket, void* buff, size_t msg_len, int flags);
  - Example:

```
if((len=send(sd,buff,strlen(buff),0))<0){
        printf("Send Error: %s (Errno:%d)\n",strerror(errno),errno);
        exit(0);
}
```

Basic Socket Programming

- send()/recv() (Server & Client)
  - Setting *flags* to 0 will be OK for most cases
  - Incomplete recv()/send(), due to the limit of the buffer size, the recv()/send() functions may input/output fewer bytes than requested.
    - Always check the return value, it indicate the bytes sent/received.
    - Use a loop to send/receive

Blocking Operations

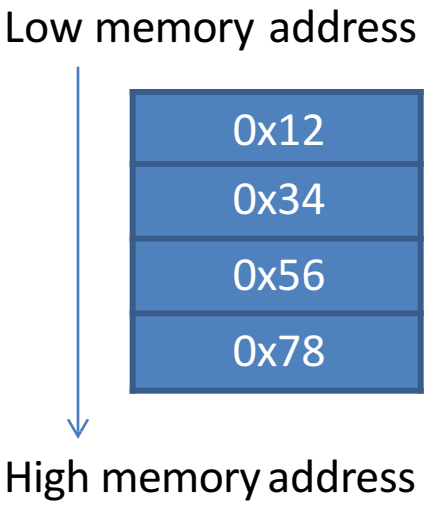
- If the socket is *blocking*, the recv() is a blocking function.
  - It will block the program and wait for data from the other side.
  - Another blocking function is accept()
- How to deal?
  - Multi-thread programming, we will cover this in the next tutorial.

Network-byte Order

- We know that an integer is stored in 4 bytes.
- Suppose an integer 0x12345678 is stored in memory. We know the 4-bytes occupied are storing the following data:
  - 0x12, 0x34, 0x56, 0x78
  - But how are they placed in memory?

Network-byte Order

- How do machines interpret unsigned integers?
- If we simply transfer the 0-1 sequence, we deliver the WRONG message!!
- How to PREVENT this?

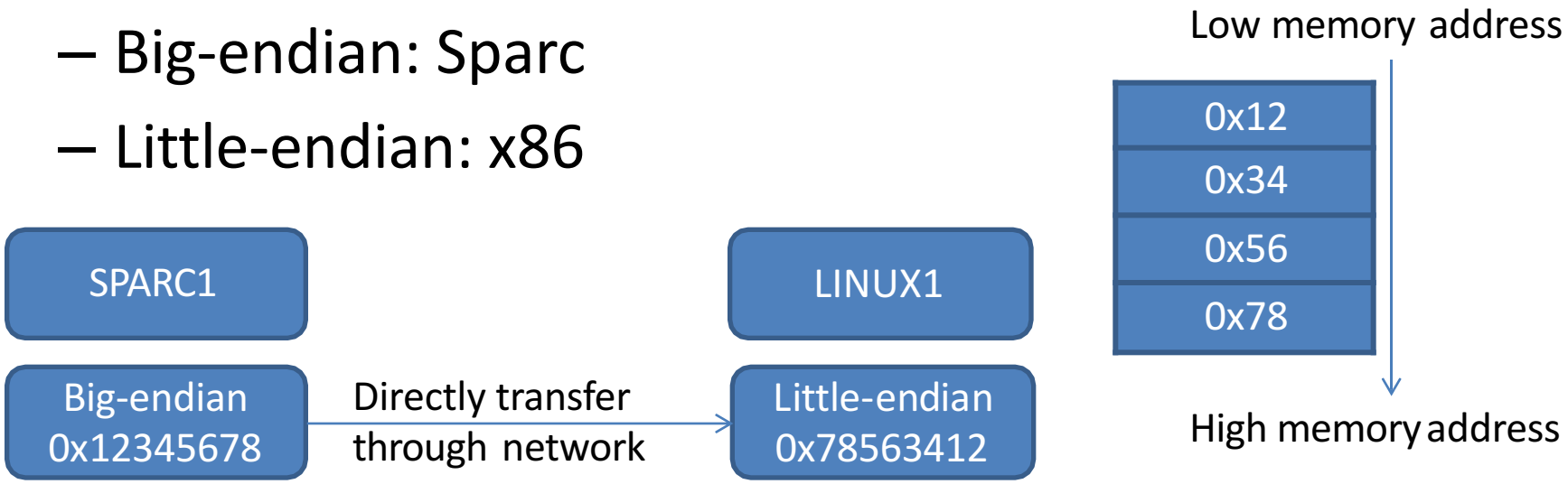


What SPARC Machine (Big Endian) sees is:
0x12345678

What X86 Machine (Little Endian) sees is:
0x78563412

Why Network Byte Ordering?

- Different machines may have different orderings (same bit-stream means different things for different machines)
  - Big-endian: Sparc
  - Little-endian: x86

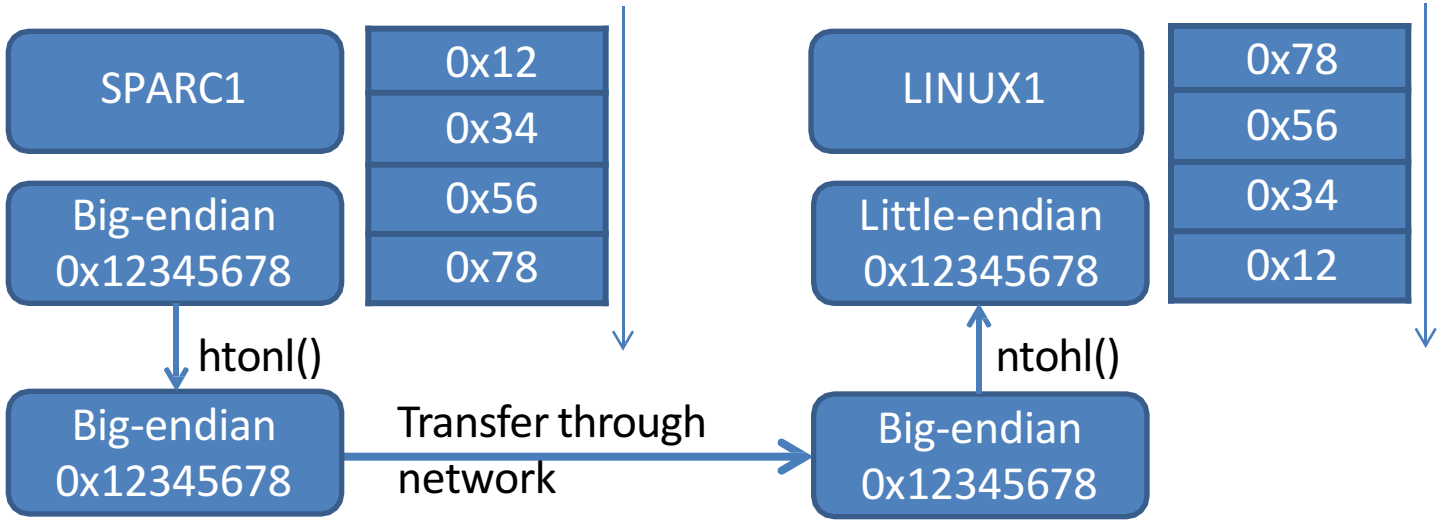


Same 0-1 sequence but different meaning for different machines

We regulate that:
Integers TRANSFERRED IN NETWORK should use Big Endian!!

Why Network Byte Ordering?

- Different machines may have different orderings (same bit-stream means different things for different machines)
  - Big-endian: Sparc
  - little-endian: x86



Network-byte Order

- We use a consistent format when transferring data through the network.
  - Network-byte order (big-endian)
- How to transfer to network-byte order?
  - uint32_t htonl(uint32_t int_type_variable);
  - uint16_t htons(uint16_t short_type_variable);

Data Transmission

- Use network byte ordering for transferring data of type *int* and *unsigned short*
  - In this assignment:
    - IP address
    - Port number
    - Argument length
  - *htonl(), htons(), ntohl(), ntohl()*

Network-byte Order

- Another way to change an IP address into network-byte order
  - Syntax：
    - in_addr_t inet_addr (const char *cp);
  - Example:

```
int sd=socket(AF_INET,SOCK_STREAM,0);
struct sockaddr_in server_addr;
memset(&server_addr,0,sizeof(server_addr));
server_addr.sin_family=AF_INET;
server_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
server_addr.sin_port=htons(PORT);
```

Compile Commands

- Compilation on Linux

```
CC = gcc
LIB =

all: server client

server: server.c
        ${CC} -o server server.c ${LIB}

client: client.c
        ${CC} -o client client.c ${LIB}
```

- Different on Solaris

```
LIB = -lsocket -lnsl
```

Demo of Sample Code

- Basic socket programming
- Data transmission
  - Network byte ordering
- Compilation on Linux and Solaris

Thank you!