Lab 5

CS-335a

Fall 2012
Computer Science Department

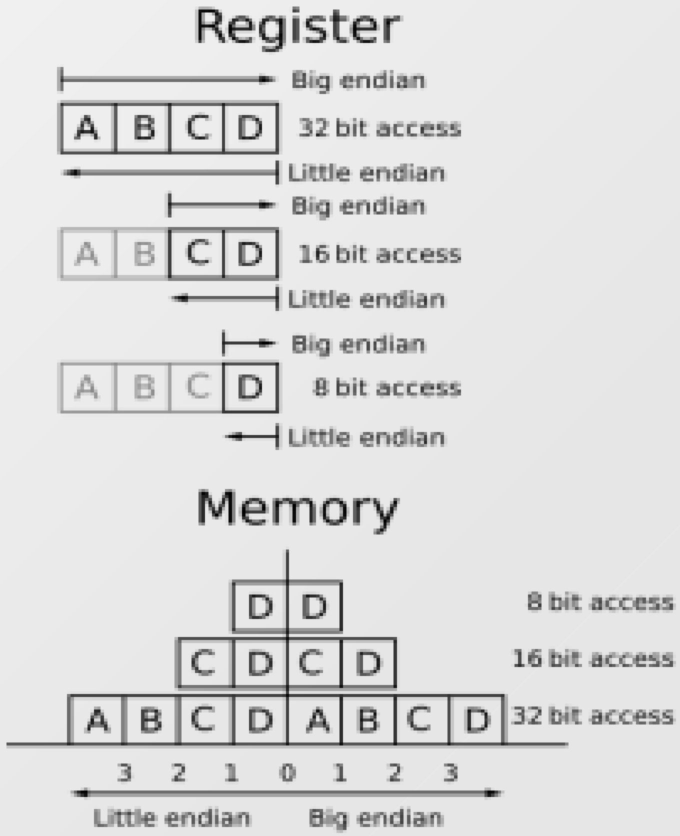
Manolis Surligas
surligas@csd.uoc.gr

Summary

- Endianess – Network Byte Order
- Create UDP sockets
- Send and receive data from a UDP socket

Endianess and Network Byte Order

- A big-endian machine stores the most significant byte
- A little-endian machine stores the least significant byte first



Endianess and Network Byte Order

- Why do we care about Endianess?
- Internet is an heterogenous network with different types of machines
- The architecture of the host at the other side, is not known
- A lazy programmer sais: '*Ok I am sure that the other host is Little-endian, so I do not care about endianness*'
- **WRONG!** The standard network byte order is big endian and many Socket API functions follow that convension

Endianess and Network Byte Order

- To convert 16 and 32 bits numbers from host byte order (little or big endian) to network byte order you can use:
  - htons()
  - htonl()

- For the inverse operation, you can use:
  - ntohs()
  - ntohls()

- Note that you have to care about endianess when you send entities with size greater than one byte, like shorts, integers etc

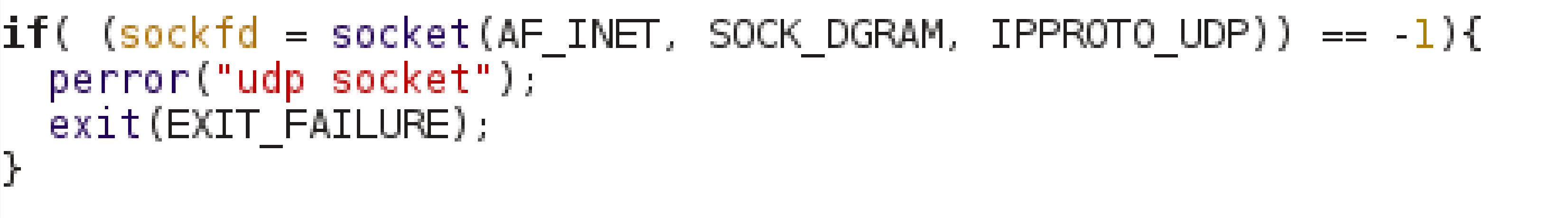
- Sending characters for example is not a problem

Create UDP sockets - Server Side

- Since UDP is connection-less, it has a little bit different procedure to create a UDP socket

- However, it follows the client-server approach like TCP

- To create an IPv4 UDP socket takes the following code:



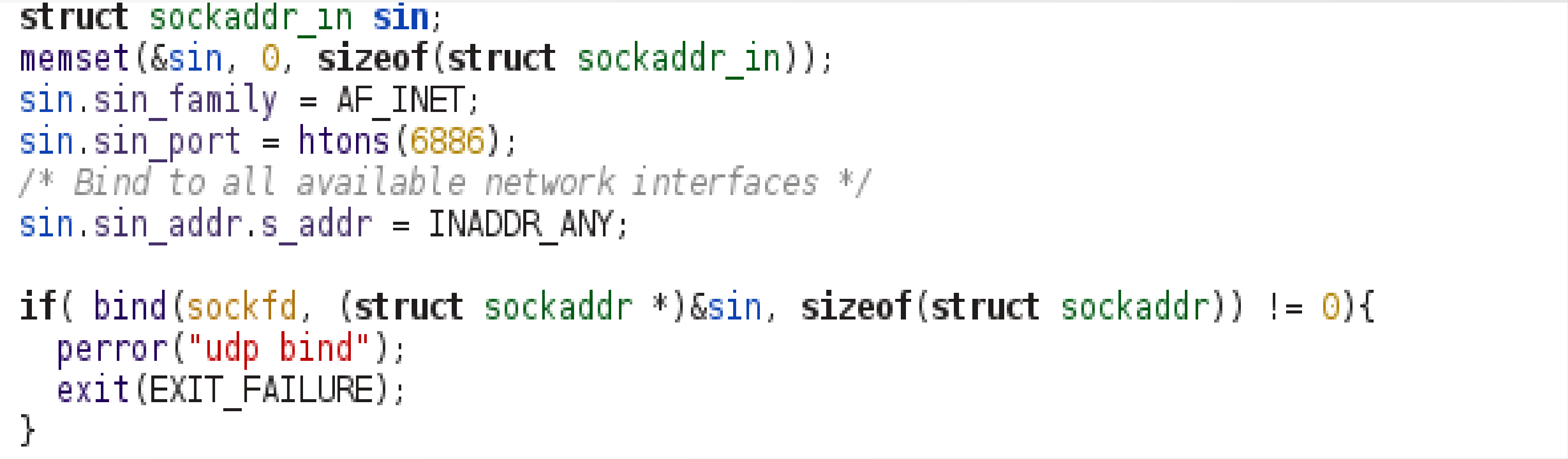
```
if( (sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1){
  perror("udp socket");
  exit(EXIT_FAILURE);
}
```

- Is almost the same with TCP but with different socket parameters

Create UDP sockets - Server Side

- At the server side, is a good practice to bind the socket with a specific port, in which the client will send its UDP packets

- The procedure for bind is the same with TCP



```
struct sockaddr_in sin;
memset(&sin, 0, sizeof(struct sockaddr_in));
sin.sin_family = AF_INET;
sin.sin_port = htons(6886);
/* Bind to all available network interfaces */
sin.sin_addr.s_addr = INADDR_ANY;

if( bind(sockfd, (struct sockaddr *)&sin, sizeof(struct sockaddr)) != 0){
  perror("udp bind");
  exit(EXIT_FAILURE);
}
```

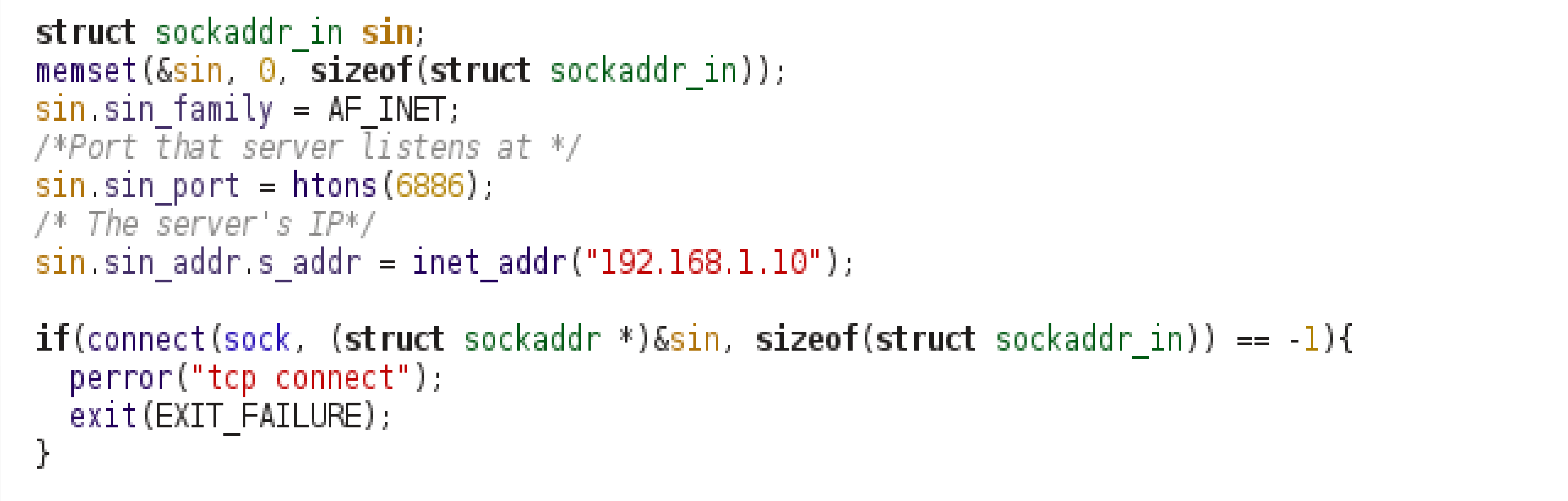
Create UDP sockets – Server Side

- Now your server socket is ready to receive UDP packets
- No need for listen()
- Accept() is useless since UDP is connectionless

Create UDP sockets – Client Side

- At the client side, thinks are also very easy

- After creating a UDP socket, just use connect() in order to be able to send and receive UDP packets from the socket



```
struct sockaddr_in sin;
memset(&sin, 0, sizeof(struct sockaddr_in));
sin.sin_family = AF_INET;
/*Port that server listens at */
sin.sin_port = htons(6886);
/* The server's IP*/
sin.sin_addr.s_addr = inet_addr("192.168.1.10");

if(connect(sock, (struct sockaddr *)&sin, sizeof(struct sockaddr_in)) == -1){
  perror("tcp connect");
  exit(EXIT_FAILURE);
}
```

Sending and receiving data with UDP

- Due to connectionless nature of UDP we can **not** use the send(), recv() system calls
- Use the sendto(), read() instead
- Not that despite TCP in UDP only read() is a blocking operation
- sendto() sends **immediatly** the UDP packet without blocking
- Possible packets lost, due to the unreliable nature of UDP, can not be recovered and is responsibility of the programmer to take care this possibility

Sending and receiving data with UDP

- Due to connectionless nature of UDP we can **not** use the send(), recv() system calls
- Use the sendto(), read() instead
- Not that despite TCP in UDP only read() is a blocking operation
- sendto() sends **immediatly** the UDP packet without blocking
- Possible packets lost, due to the unreliable nature of UDP, can not be recovered and is responsibility of the programmer to take care this possibility

Useful man pages

  - connect(3p)
  - bind(3p)
  - getaddrinfo(3p)
  - setsockopt(3p)
  - sendto(3p)
  - read(3p)
  - inet_ntoa(3p)

- For every man page, take a look at the **SEE ALSO** section. Many other functions that may need are there