BSD Sockets API

Constantinos Dovrolis

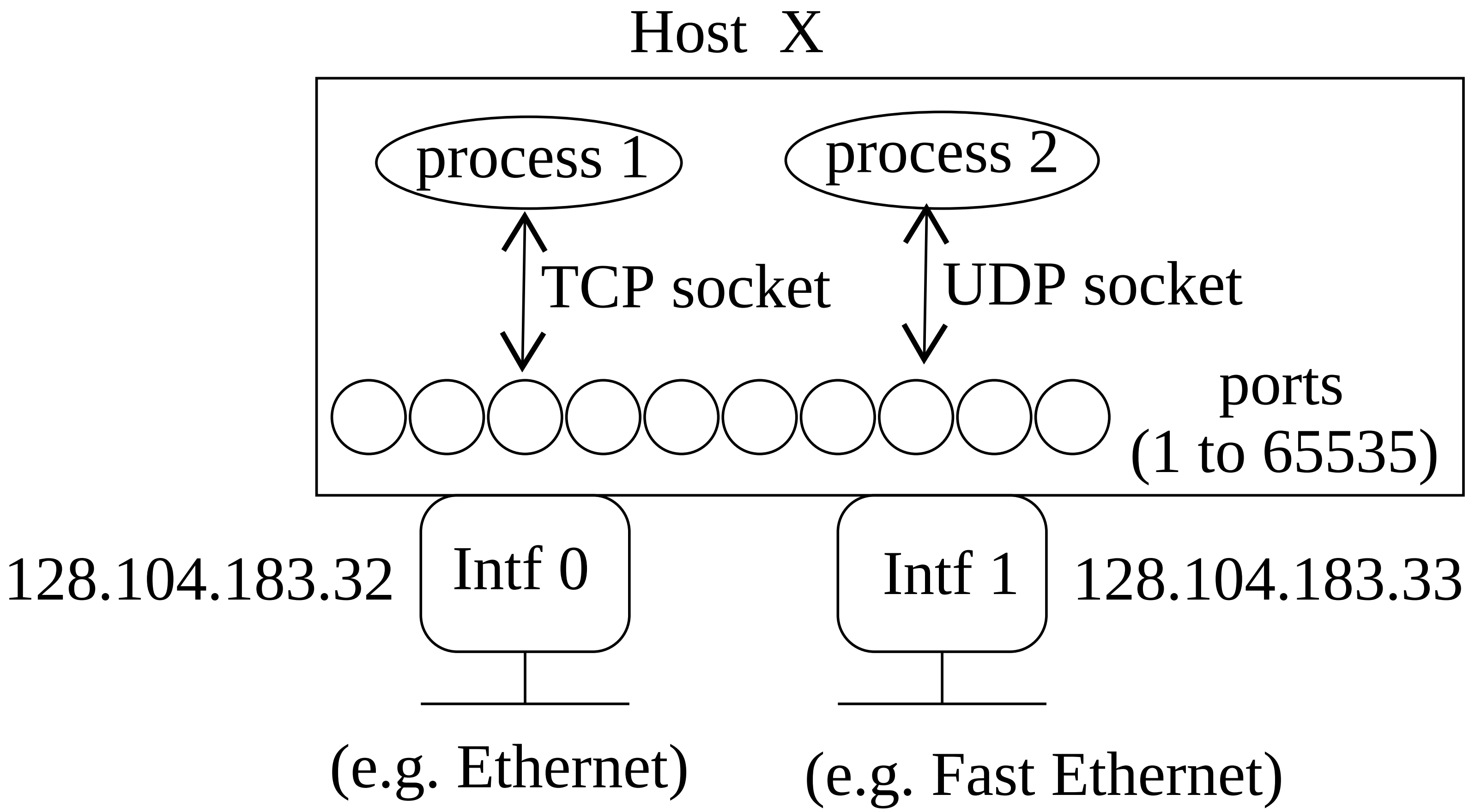
Office hours:

M: 4-6 W: 4-5

dovrolis@cs.wisc.edu

2-5130, CS 5364

Hosts, Ports, Interfaces, Sockets



Host and Network Byte Order

- How to deal with big-endian vs little-endian machines?
- **in_addr_t htonl(in_addr_t hostlong)**
- **in_port_t htons(in_port_t hostshort)**
- The reverse conversion → **ntohl()** and **ntohs()**
- These conversions apply to **all multi-byte numbers** (including addresses and ports)

Creating Sockets

- **int socket (domain, type, protocol)**

```
/* UDP: unreliable message transfer */
if ((sd_udp=socket(AF_INET,SOCK_DGRAM,0))<0) {
        perror("udp socket:"); exit(1);
}

/* TCP: reliable bi-directional byte-stream */
if ((sd_tcp=socket(AF_INET,SOCK_STREAM,0))<0) {
        perror("tcp socket:"); exit(1);
}
```

- Other possible domain: **AF_UNIX**; **protocol=0** means *use default*

Binding a Socket to a Local Address

- **int bind (int sd, const struct sockaddr *address, size_t namelen)**

```
bzero((char*)&cli_adr, sizeof(cli_adr));
cli_addr.sin_family          = AF_INET;
cli_addr.sin_addr.s_addr     = htonl(INADDR_ANY);
// use an arbitrary available port
cli_addr.sin_port            = htons(0);
if (bind(sd, (struct sockaddr*)&cli_addr,
         sizeof(cli_addr))<0) {
              perror("bind socket:"); exit(1);
}
```

- **INADDR_ANY:** any host address (important for multi-interface hosts)

Binding a Socket to a Specific Port

- Suppose a server listens to a port **SERVER_PORT**, known to the clients

```
bzero((char*)&srv_addr, sizeof(srv_addr));
srv.sin_family          = AF_INET;
srv.sin_addr.s_addr     = htonl(INADDR_ANY);
srv.sin_port            = htons(SERVER_PORT);
if (bind(sd, (struct sockaddr*)&srv_addr,
         sizeof(srv_addr))<0) {
              perror("bind socket:"); exit(1);
}
```

- Well-known ports (<1024): see **/etc/services**

Forming a Remote Address

- **struct hostent *gethostbyaddr (void *addr, size_t len, int type);**

  struct hostent *gethostbyname (const char *name);

```
if ((rmt_addr = gethostbyname(argv[1])) == 0) {
        printf("%s: unknown\n",argv[1]); exit(1);
}
memcpy((void*)&(rcv_addr.sin_addr.s_addr),
        rmt_addr->h_addr, rmt_addr->h_length);
```

- Also see **inet_addr()**, **inet_aton()**, **inet_ntoa()**

Sending UDP packets

- **ssize_t sendto (int socket, void *message, size_t length, int flags, struct sockaddr *dest_addr, size_t dest_len)**

```
if (sendto(sd_udp, msg_buf, msg_sz, 0,
           (struct sockaddr*)&rcv_addr,
           sizeof(rcv_addr)) != msg_sz) {
                perror("sendto:"); exit(1);
}
```

- The sender needs to first form the receiver's address (see previous slide)

Receiving UDP packets

- **ssize_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr *address, size_t *address_len);**

```
snd_addr_len = sizeof(snd_addr);
if ((nread=recvfrom(sd_udp, msg_buf, max_msg_sz,
        0, (struct sockaddr*)snd_addr,
        (size_t*)&snd_addr_len)) < 0) {
            perror("recvfrom:"); exit(1);
}
```

- Blocking (but can be made non-blocking with **O_NONBLOCK** flag, or use **select()** with one socket and timeout)

TCP Connection Establishment: Callee

- We will **not** use TCP in the assignment and project
- After TCP-socket creation and binding:

```
if (listen(sd_tcp, NO_CONNECTIONS) < 0) {
        perror("listen:"); exit(1);
}
client_len = sizeof(client_addr);
if (conn_sd=accept(sd_tcp, (struct sockaddr*)
        &client_addr, &client_len)) < 0) {
         perror("accept:"); exit(1);
}
```

TCP Connection Establishment: Caller

- After TCP-socket creation and binding:

```
if (connect(sd_tcp, (struct sockaddr*)
    &server_addr, sizeof(server_addr))<0){
        perror("connect:"); exit(1);
}
```

TCP Data Transfer

- Sending is like writing to a file:

```
if (write(sd_tcp, msg_buff, msg_len)!=msg_len) {
         perror("write to socket:"); exit(1);
}
```

- Receiving is like reading from a file:

```
if ((nread=read(sd_tcp, msg_buff, msg_len))<0) {
         perror("read from socket:"); exit(1);
}
```

Select

- Check which of an array of sockets are ready for read/write or are in error condition; return when socket-status changes or when timeout expires

```
int select(int nfds, fd_set *readfds,
           fd_set *writefds, fd_set *errorfds,
           struct timeval *timeout);

void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int  FD_ISSET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

- Use timeout argument (with only one socket) to make unblocking **recvfrom()**

Questions?