# Sockets

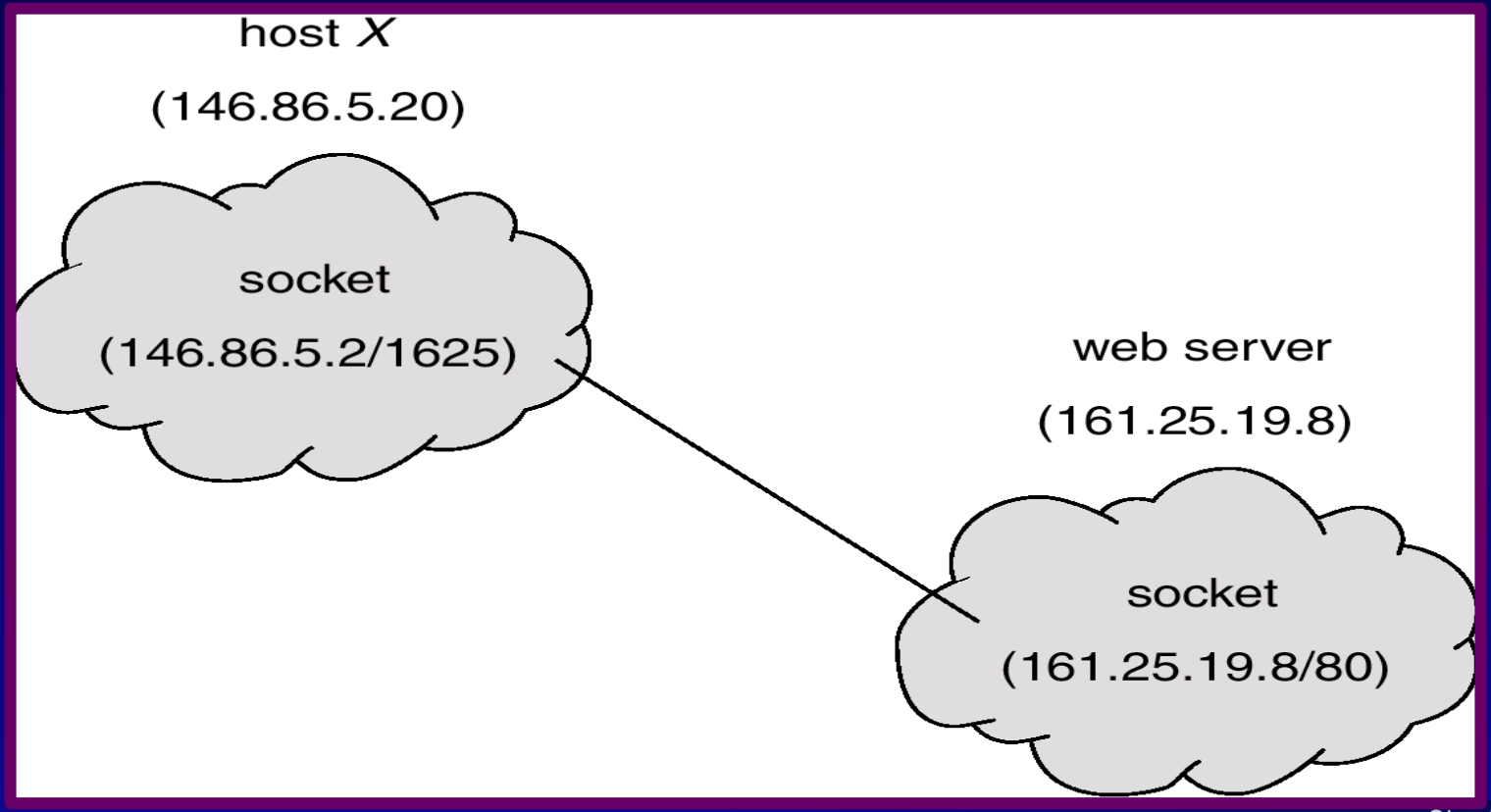
- A socket is an *endpoint* of communication.
- An *in-use* socket it usually bound with an *address*
- The nature of the address depends on the *communication domain* of the socket.
  - Unix, Internet, XEROX (historical)
  - e.g. 161.25.19.8:1625 is an address in the Internet domain referring to:

    **Host IPv4**: 161.25.19.8

    **Port**: 1625

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Sockets




Giorgio Richelli
giorgio.richelli@uniroma1.it

# Sockets

- Communication can be estabilished between *pairs* of sockets.
- Each active socket has:
  - Address
  - Protocol
- Processes communication in the same domain use the same address format

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Sockets

- A single socket can communicate in only one domain
- Commonly implemented domains:
  - UNIX (AF_LOCAL, PF_LOCAL)
  - Internet (AF_INET, PF_INET)

- Note:
  - Originally it was thought that an address family might support several protocols.
  - So the most correct thing would be:

    `AF_INET` in `struct sockaddr_in`

    `PF_INET` in calls to `socket()`.
  - But in practice, you can always use `AF_INET`.

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket Types

- Stream
  - Reliable, duplex, sequenced data streams.
  - Supported in Internet domain by the TCP protocol.
  - In UNIX domain, pipes are implemented as a pair of communicating stream sockets.
- Sequenced packet
  - Provide similar data streams, except that record boundaries are provided.

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket Types

- Datagram:
  - Transfer messages of variable size in either direction.
  - Supported in Internet domain by UDP protocol
- Reliably delivered message:
  - Connectionless, message-oriented, preserving message boundaries
  - Guaranteed to arrive
  - Almost  unsupported

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket Types

- Raw:
  - Allow direct access by processes to the protocols that support the other socket types.
  - In the Internet domain, it is possible to reach:
    - TCP
    - IP
    - Ethernet
- Useful for developing new protocols or for *sniffers.*

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

- `socket()`
  - creates a socket
- `bind()`
  - Assigns name and address to a socket
- `server:` `listen()/accept()`
- client: connect()
  - Initiate the connection

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls (Cont.)

- `close()`
  - terminates a connection and destroys the associated socket
- `select()`
  - multiplexes data transfers on several file descriptors (and /or socket descriptors).

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

A **server** process usually calls:

- `socket()`
  - to create a socket
- `bind()`
  - to bind an address
- `listen()`
  - to indicate willingness to accept connections from clients

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

A **server** process then calls (cont):

- `accept()`
  - to accept an individual connection
  - eventually, `fork()` a new process after the accept()
- `send() & recv()`
  - to move data
- `close()`
  - when all is done on the connection

Giorgio Richelli
giorgio.richelli@uniroma1.it

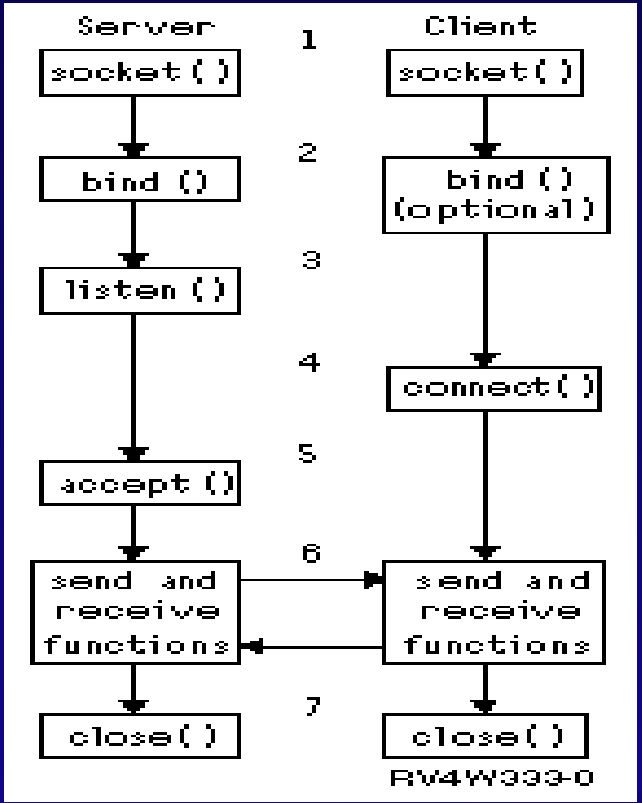
# Socket System Calls

A **client** process usually calls:

- `socket()`
  - to create a socket
- `connect()`
  - to establish a connection with server
- `send(), recv()`
  - to move data
- `close()`
  - to close the connection

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket Calls Flow




Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int socket(int domain, int type, int protocol);
```

- creates an endpoint for communication and returns a descriptor
- `domain`: PF_UNIX, PF_INET, ...
- `type`: SOCK_STREAM, SOCK_DGRAM, ...
- `protocol`: 0, IPPROTO_TCP, IPPROTO_UDP, ...

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int bind(int sockfd, struct sockaddr *my_addr,
socklen_t addrlen);
```

- gives the socket `sockfd` the local address `my_addr` (`addrlen` bytes long)

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int listen(int s, int backlog);
```

- specify willingness to accept incoming connections and a queue limit (for pending connections)
- `s`: socket
- `backlog`: maximum length for the queue

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int accept(int s, struct sockaddr *addr, socklen_t
*addrlen);
```

- extracts the first connection request on the queue of pending connections on s and creates a new connected socket with (mostly) the same properties
- `s`: socket
- `addr`: will contain the *from* address
- `addrlen`: bytes available in addr

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
ssize_t send(int s, const void *buf, size_t len, int
flags);
```

- transmits a message to another socket
- `s` must be *connected*
- almost identical to `write()`, except for `flags`
- `flags`: MSG_DONTWAIT, MSG_DONTROUTE, ....

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
ssize_t recv(int s, void *buf, size_t len, int flags);
```

- receives messages from a (connected) socket
- almost identical to a `read()`, except for flags
- `flags`: MSG_PEEK, MSG_TRUNC, MSG_WAITALL, ...

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int connect(int sockfd, const struct sockaddr
*serv_addr, socklen_t addrlen);
```

- attempts to connect `sockfd` to another socket, specified by `serv_addr`, which is an address (of length `addrlen`) in the communications space of the socket.
- returns: 0 / -1

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Socket System Calls

```
int select(int n, fd_set *readfds, fd_set *writefds,
fd_set  *exceptfds,  struct timeval *timeout);
```

- waits on a number of file descriptors (until, eventually, a timeout occurs)
- three sets of descriptors are watched:
  - `readfds` see if characters become available for reading
  - `writefds` will be watched to see if a write will not block
  - `exceptfds` will be watched for exceptions
- macros to manipulate the sets:
  - FD_ZERO, FD_SET, FD_CLR, FD_ISSET

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Getting Host Name & Address(es)

```
struct hostent *gethostbyname(const char *name);
```

- returns a structure of type `hostent` for the given host name (either an host name, or an IP address)

```
struct hostent {
  char    *h_name;        /* official name of host */
  char    **h_aliases;    /* alias list */
  int     h_addrtype;      /* host address type */
  int     h_length;          /* length of address */
  char    **h_addr_list; /* list of addresses */
}
#define h_addr  h_addr_list[0]  /* backw. compatibility */
```

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Getting Host Name & Address(es)

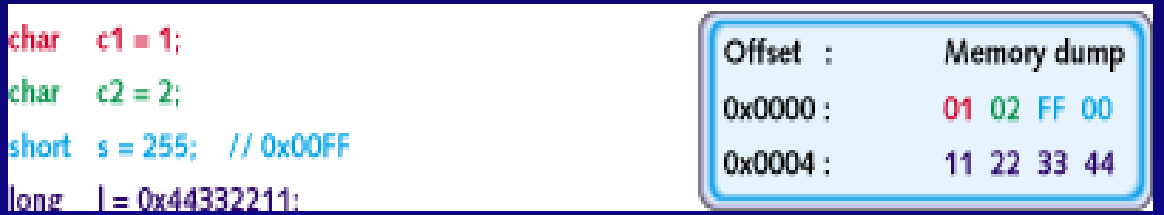
```
struct hostent *gethostbyaddr(const char *addr, int
len, int type);
```

- returns a structure of type `hostent` for the given host address addr of length `len` and address type `type`.
- type: AF_INET, AF_INET6

Giorgio Richelli
giorgio.richelli@uniroma1.it

# Endiannes

- Big Endian
  - the most significant byte of any multibyte data field is stored at the lowest memory address
- Little Endian
  - the least significant byte of any multibyte data field is stored at the lowest memory address




Giorgio Richelli
giorgio.richelli@uniroma1.it

# Host Independent Formats

x86 CPUs are LittleEndian, the network byte order is BigEndian

- from host byte order to network byte order:
  - `uint32_t htonl(uint32_t hostlong);`
  - `uint16_t htons(uint16_t hostshort);`
- from network byte order to host byte order:
  - `uint32_t ntohl(uint32_t netlong);`
  - `uint16_t ntohs(uint16_t netshort);`

Giorgio Richelli
giorgio.richelli@uniroma1.it