Unix Socket Interface

- Introduced in 4.2 BSD.
- A socket is an endpoint of communication referred to by a descriptor.
- Include files:
  <stdio.h>
  <sys/types.h> /*sys/socket.h uses sys/types.h*/
  <sys/socket.h> /*defines constants such as SOCK_DGRAM used in socket based I/O */
  <netinet/in.h> /*defines structures such as sockaddr_in */
  <netdb.h> /* defines host ip addresses */

Socket system calls

1. A socket is created by

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

   - `domain` specifies the address family (format) that will be used later with the socket. PF_INET: Internet family; PF_UNIX: Unix pipe facility
   - `type` indicates the type of socket: SOCK_DGRAM for connectionless; SOCK_STREAM for connection-oriented; SOCK_RAW: allows privileged users to low-level IP interfaces.
   - `protocol` specifies the protocol to be used: udp: IPPROTO_UDP, tcp: IPPROTO_TCP, UNSPEC...

2. The next step is to bind this socket to an internet address. This is done by

   ```
   int bind(int socket, struct sockaddr *address, int addr_len)
   ```

   `sockaddr_in` is defined in netinet/in.h:

   ```
   struct sockaddr_in {
       short sin_family; /*kind of socket, in our case: AF_INET */
       u_short sin_port; /*port number, allocated by bind */
       struct in_addr sin_addr; /*in_addr defined in netinet/in.h*/
               /* in_addr: internet address structure */
       char sin_zero[8]; /* padding to be consistent with sockaddr in sys/socket.h*/
   };
   ```

3. In a connection-based communication, the process that initiates a connection is termed a *client process*, whereas the process that receives, or responds to, a connection is termed a *server process*. In the client process, a connection is initiated with a *connect* system call:

   ```
   int connect(int socket, struct sockaddr *address, int addr_len)
   ```

4. In the server process, the socket is first marked to indicate incoming connections are to be accepted on it:

   ```
   int listen(int socket, int backlog)
   ```

   - `backlog` specifies an upper bound on the number of pending connections that should be queued for acceptance.

5. Connections are then received, one at a time, with

   ```
   int accept(int socket, struct sockaddr *client_addr, int addr_len)
   ```

   - `accept` returns a new socket descriptor associated with the newly established connection; `client_addr` contains the address of the client that connects to the server.

6. Message may be sent or received by:

   ```
   int send(int socket, char *message, int msg_len, int flags)
   int sendto(int socket, char *message, int msg_len, int flags,
              struct sockaddr *recv_addr, int addr_len)
   int recv(int socket, char *buffer, int buf_len, int flags)
   int recvfrom(int socket, char *buffer, int buf_len, int flags,
              struct sockaddr *from_addr, int addrlen)
   ```

7. Other useful commands: `socketpair`, `select`, `close`, `dup`, `setsockopt`, `getsockopt`, `read`, `write`, `getpeername`, `getsockname`, `gethostname`, `gethostent`, `htons`, `ntohs`, `inet_addr`, ...