Ateneo de Manila University

Network Programming with Sockets

Department of Information Systems and Computer Science
S.Y. 2001-2002
http://sysads.ateneo.net/wyu/
wyy@admu.edu.ph
Socket

★ provide point-to-point, two-way communication between two processes
★ are very versatile and are a basic component of interprocess and inter-system communication
★ is an endpoint of communication to which a name can be bound
★ exist in communication domains called socket domains
★ can only connect to sockets in the same domain
★ there are twenty three sockets types defined in socket.h
★ of these domain only the Unix and Internet domains are used
Socket Types

* define the communication properties visible to the application

* processes communicate only between sockets of the same type

* there are five types of socket:
  - Stream Socket
  - Datagram Socket
  - Sequential Packet Socket
  - Raw Socket
Stream Socket

★ provides two-way, sequenced, reliable, and unduplicated flow of data with no record boundaries
★ operates much like a telephone conversation
★ socket type is SOCK_STREAM
★ in the Internet domain, uses Transmission Control Protocol (TCP)
Datagram Socket

★ supports a two-way flow of messages

★ a datagram socket may receive messages in a different order from the sequence in which the messages were sent

★ record boundaries in the data are preserved

★ operate much like passing letters back and forth in the mail

★ socket type is SOCK_DGRAM

★ in the Internet domain, uses User Datagram Protocol (UDP).
Sequential Packet Socket

* provides a two-way, sequenced, reliable, connection, for datagrams of a fixed maximum length

* socket type is SOCK_SEQPACKET

* no protocol for this type has been implemented for any protocol family
Raw Socket

★ provides access to the underlying communication protocols

★ bypasses the layers in the TCP/IP stack

★ only root (superuser) can open a raw socket
Socket Programming

★ using sockets to communicate between different processes
★ these processes can be remote or local
★ sockets are manipulated like unbuffered I/O files and use socket descriptors
★ most popular of APIs for socket programming is Berkeley Sockets
★ another popular socket API is the System V Transport Layer Interface (TLI)
Socket Programming Considerations

★ client-server functionality is not symmetric and thus they must be delineated

★ network connections are either connection oriented or connectionless
  – for connection oriented, the connection must be opened to allow handshaking
  – for connectionless, the connection need not be opened

★ names are important
  – for file I/O, simply using the file descriptor is enough for accessing files
  – for network communications, names are used typically for authentication and verification

★ network interface can support different protocols which will need different parameters to initialize connections
BSD Sockets System Call Summary

Server Calls

⋆ socket() create connection points
⋆ bind() bind socket to an address
⋆ listen() indicates willingness to receive communications
⋆ accept() takes a connection request

Client Calls

⋆ socket() create connection points
⋆ bind() bind socket to an address
⋆ connect() creates a connection request
⋆ close() terminates connection
Connectionless Network I/O

★ `sendto` sent a datagram
★ `recvfrom` receive a datagram

Connection Oriented Network I/O

★ `read` read data from socket
★ `write` write data to socket
★ `send` send a message to a socket
★ `recv` receive a message from a socket
Figure 1: Socket System Calls for Connection Oriented Protocols
Figure 2: Socket System Calls for Connectionless Protocols
BSD Socket API

⋆ int socket(int domain, int type, int protocol);
− is called to create a socket in the specified domain and of the specified type
− the system defaults to a protocol that supports the specified socket type
− a socket handle (a descriptor) is returned

⋆ int bind(int s, const struct sockaddr *name, int namelen);
− is called to bind a path or internet address to a socket
− there are three different ways to call bind():
  * bind (sd, (struct sockaddr *) &addr, length); - UDP (short paths)
  * bind (sd, (struct sockaddr_un *) &addr, length); - UDP (long paths)
* bind (sd, (struct sockaddr_in *) &addr, length); - TCP

* int listen(int s, int backlog);
  - specifies how many connection requests can be queued
  - backlog is the max number of pending connection

* int connect(int s, struct sockaddr *name, int namelen);
  - initiates a connection to the server’s socket
  - this is initiated by the client

* int accept(int s, struct sockaddr *addr, int *addrlen);
  - returns a new socket descriptor which is valid only for the particular connection
  - this is called by the server
⋆ int send(int sock, const char *buf, int len, int flags);
⋆ int recv(int sock, char *buf, int len, int flags);

– send and receive data from a socket connection defined by sock
– can be used in place of the regular file descriptor functions
– the data is stored or retrieved from buf
– flags can be:
  * MSG_OOB send "out-of-band" data on sockets
  * MSG_DONTROUTE used only by diagnostic or routing programs
  * MSG_PEEK peek at the data present on the socket

⋆ int sendto(int sock, const char *buf, int len, int flags, const struct sockaddr *to, socklen_t tolen);
⋆ int recvfrom(int sock, char *buf, int len, int
flags, const struct sockaddr *to, socklen_t *tolen);

- point to point equivalent of send and recv
- to is the destination or the source of the message
Copyright © 2000-2001 by William Emmanuel S. Yu. This material may be distributed only subject to the terms and conditions set forth in the Open Content License, v1.0 or later (the latest version is presently available at http://opencontent.org/opl.shtml).