Socket Abstraction and Inter-process Communication

B.Ramamurthy

CSE421

Communication Mechanisms

- Pipes (process level)
- Sockets (OS level)
- Distributed System Methods
  - Remote Procedure Call- RPC
  - Remote Method Invocation (Programming language level)
- Other communication paradigms

Introduction

- Typical applications today consist of many cooperating processes either on the same host or on different hosts.
- For example, consider a client-server application. How to share (large amounts of) data?
- Share files? How to avoid contention? What kind of system support is available?
- We want a general mechanism that will work for processes irrespective of their location and across address spaces.

Purposes of communication

- Data transfer
- Sharing data
- Event notification
- Process control (invoke methods, send messages)

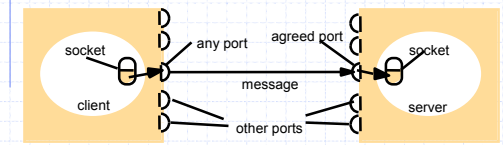
Communication using sockets

- What if wanted to communicate between processes that have no common ancestor?
- IPC for processes that are not necessarily on the same host?
- Means of network IO?
- Answer: sockets
- Lets study socket: architecture, API, usage
- Architecture: names, types, ports, addresses and domains

Sockets

- Socket is an abstraction for an end point of communication that can be manipulated with a file descriptor.
- It is an abstract object from which messages are sent and received.
- Sockets are created within a communication domain just as files are created within a file system.
- A communication domain is an abstraction introduced to bundle common properties of processes communicating through sockets. Example: UNIX domain, internet domain.

Sockets and ports



Internet address = 138.37.94.248 Internet address = 138.37.88.249

Sockets and Addresses

- IP address and port number. About 2^{16} ports are available for use by user processes.
- UDP and TCP abstraction of the above is a socket.
- Socket is associated with a protocol.
- IPC is transmitting a message between a socket in one process to a socket in another process.
- Messages sent to particular IP and port# can be received by the process whose socket is associated with that IP and port#.
- Processes cannot share ports with other processes within the computer. Can receive messages on diff ports.

Socket Names

- Applications refer to sockets by name.
- But within the communication domain sockets are referred by addresses.
- Name to address translation is usually done outside the operating system.

Socket types

- The format in which an address is specified is according to a domain:
  - AF_UNIX (address format of UNIX) - a path name within the file system,
  - AF_INET (internet format) : network address, port number etc.
- Communication style: stream , datagram, raw or sequenced packets
- Stream : reliable, error-free, connection-oriented comm.
- Datagram: Connectionless, unreliable, message boundaries preserved.
- Why use datagram?

Socket API

- Creation: name, type, domain are specified
- Binding: bound to an address, port
- Close: release data structures and port
- Send and Receive: several versions
- Listen: Specify number of connections that can be accepted
- Accept and Connect: Wait for connection and connect to a socket respectively
- Other miscellaneous functions such as getsockname, gethostbyname etc.

Functions : creation

- Socket creation : socket system call creates sockets on demand.

sockid = socket (af, type, protocol);

where sockid is an int,

af - address family , AF_INET, AF_UNIX, AF_AAPLETALK etc.

type - communication type:
SOCK_STREAM, SOCK_DGRAM, SOCK_RAW etc.

protocol - some domains have multiple protocol, use a 0 for your appl.

Example: door1 = socket(AF_UNIX, SOCK_DGRAM,0);

Functions - bind

- Socket binding: A socket is created without any association to local or destination address. It is possible to bind the socket to a specific host and in it a specific port number.

socerr = bind (sockid, localaddr, addrlength)

localaddr - a struct of a specific format for each address domain;

addrlength - is the length of this struct; obtained usually by *sizeof* function.

Functions -bind (contd.)

- Example: type *sockaddr_un* defines localaddr format for unix family.

its definition, (you don't have to define it... it is in *un.h* file - include this file)

```
struct sockaddr_un {
        short sun_family;
        char sun_path[108]; };
in your program:
#define SocName "testsock"
sockaddr_un  mysoc;
mysoc.sun_family = AF_UNIX;
strcpy(mysoc.sun_path, SocName);
binderr = bind(sockid, &mysoc, sizeof(mysoc));
```


Functions (contd.) - close

close (socid); closes the specified socket. This is done by a process or thread when it no longer needs the socket connection. Premature closing results in "broken pipe" error.

- When a socket is created it is represented by a special file ('s' in the place where d appears for directory files when you ls -l).The name of the file is the name assigned in the socket bind command.

Functions (contd.) - connect

- A socket is created in an unconnected state, which means that the socket is not associated with any destination.
- An application program should call connect to establish a connection before it can transfer data thru' reliable stream socket. For datagrams connect is not required but recommended.

connect (sockid, destaddr, addlength);

Example: if (connect(sock, &server, sizeof(server)) < 0)
...

- ("sendto" commend does not need "connect")

Functions (contd.) -sending

- Five different system calls : **send, sendto, sendmsg, write, writev**
- **send, write and writev** work only with connected sockets. No parameter for destination address. Prior connect should be present for communication.
- Example : write (sock, DATA, sizeof(DATA));
- sendto (socket, message, length, flags, destaddr, addlen);
- flags allow for special processing of messages.
- sendmsg is same as sento except that it allows for different message structure.

Functions(contd.) - receiving

- Five different calls are available: read, readv, recv, recvfrom, recvmsg
- read, readv, and recv are for connection-oriented comm.
- **read(socdescriptor, buffer, length); Example: read(sock, buf, 1024);**
- For your application (project) you may use read.
- For connectionless, datagram-kind :
- recvfrom(same set of params as sendto); except that message length and addr length return values.

Functions (contd.) -listen

- accept and listen are connection-oriented communication.
- These are for AF_INET, SOCK_STREAM type of sockets.
- listen: To avoid having protocols reject incoming request, a server may have to specify how many messages need to be queued until it has time to process them. Example: listen(socket,length);
- accept: wait for the call. Example: accept(sockid, sockaddr, sizeof sockaddr);

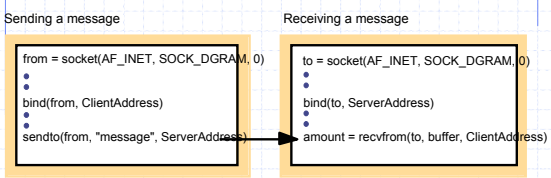
Functions (contd.) - accept

- **accept** : blocks until a connect calls the socket associated with this connection. socket -- bind --(listen) -- accept is the sequence. "connect" from calling process will complete the connection and unblock the process or thread that is blocked on "accept"
- Now read, write or writev can be executed to carry out the actual communication over the connection established.

Functions (contd.) - getsockname

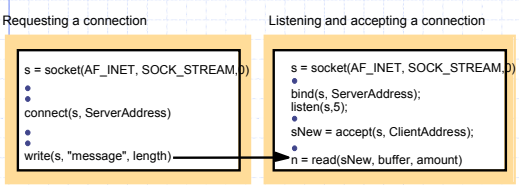
- **getsockname (sockid, sockaddr, sizeof sockaddr);** : given a sockid returns the address of the socket identified by sockid.
- This address may be needed, for instance, by an accept function call.
- There are other functions such as gethostbyname may be needed for internet domain sockets. See man pages for the details.

Sockets used for datagrams



ServerAddress and *ClientAddress* are socket addresses

Sockets used for streams



ServerAddress and *ClientAddress* are socket addresses

Sockets on Unix

- When a new process is created, the newly created process inherits access to all open sockets from its parent.
- For threads socket identifiers should be defined in the common address space or passed as parameters.
- include files : <socket.h>, <un.h> or <in.h>, and other related header files.
- When linking add a **-lsocket –lnsl** options besides.