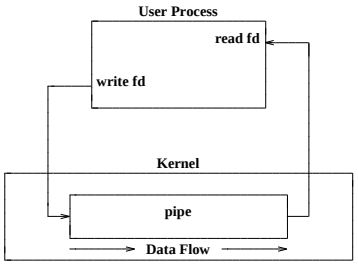
Network Programming
Connectivity Interfaces

- Early interfaces, limited and low-level programming (MS-Net)
  - Software maintenance and portability is at best arduous.
  - May necessitate to train programmers to be specialized network programmers.
  - Backward compatibility requires continuous support for these low-level applications.
  - High data transmission throughput, however, was achieved based on low-overhead associated with the transport mechanism.

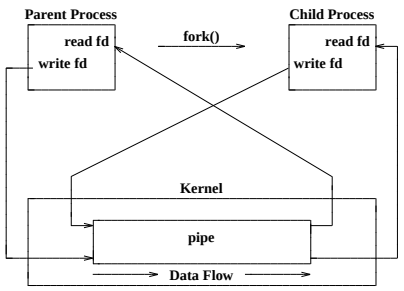
Connectivity Interfaces
Pipes

- Pipes provide a one-way flow of data between processes
  - A pipe is created by a pipe() system call
    - The call returns two file descriptors, one for reading and one for writing

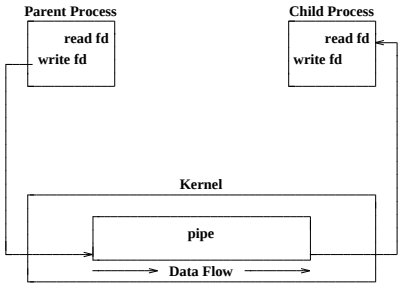
Connectivity Interfaces
Pipes, Single Process



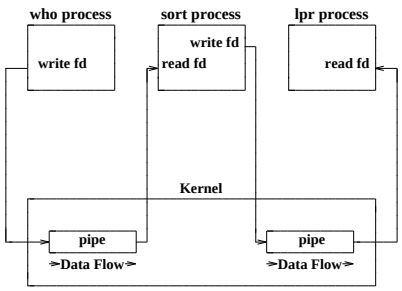
Connectivity Interfaces
Pipes, Related Processes



Connectivity Interfaces
Pipes, Related Processes



Connectivity Interfaces
Pipes

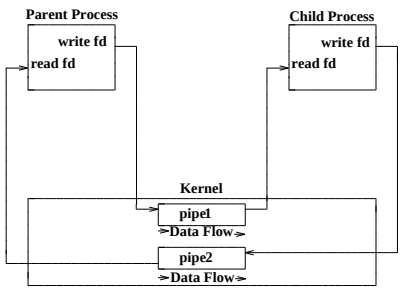


who | sort | lpr

Connectivity Interfaces
Bidirectional Pipes

- create pipe 1, create pipe 2
- fork()
- parent closes read end of pipe 1
- parent closes write end of pipe 2
- child closes write end of pipe 1
- child closes read end of pipe 2

Connectivity Interfaces
Bidirectional Pipes



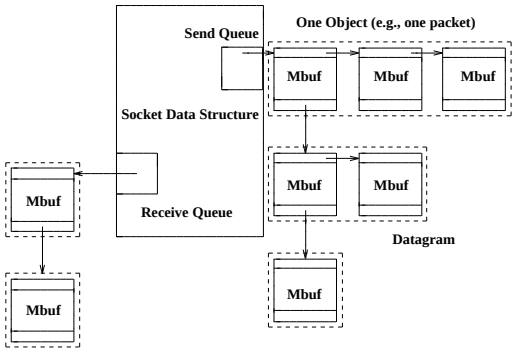
Connectivity Interfaces
FIFOs (Named Pipes)

- A Unix FIFO is similar to a pipe
  - It is a one-way flow of data, with the first byte written to it being the first byte read from it
- Unlike pipes, FIFOs have names associated with them
  - Consequently, unrelated processes can have access to them by their names

Connectivity Interfaces
Sockets

- Socket is an abstraction that provides endpoint for communication
  - A socket is created
  - Used for communication
  - Terminated

Socket Structure

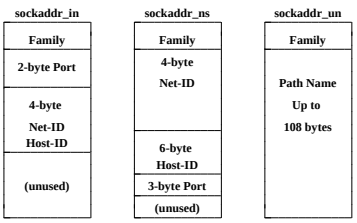


Mbuf :
- Resides in Memory
- Up to 128 bytes

The stream socket maintains only one dashed box

Sockets

- int socket(int family, int type, int protocol)
  - AF_Unix family, Unix Internal Protocol
  - AF_INET family, Internet Protocol
  - AF_NS family, Xerox NS protocols



Socket Types

- SOCK_STREAM, for a reliable stream delivery
- SOCK_DGRAM, for a datagram service
- SOCK_RAW, usually for the development of new protocols
- SOCK_SEQ_PACKET for sequenced packet delivery
- SOCK_RDM, reliably delivered message socket
  - Not implemented yet

Socket Types

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	Yes	TCP	SPP
SOCK_DGRAM	Yes	UDP	IDP
SOCK_RAW		IP	Yes
SOCK_SEQPACKET			SPP

SPP = Sequenced Packet Protocol

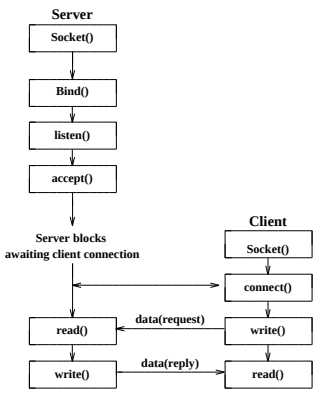
IDP = Internetworking Datagram Protocol

Socket Arguments
Protocol

- The protocol argument refers to the protocol that supports the semantics of the socket

Family	Type	Protocol	Actual Protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPROTO_RAW	(Raw)
AF_NS	SOCK_DGRAM	NSPROTO_SPP	SPP
AF_NS	SOCK_SEQPACKET	NSPROTO_SPP	SPP
AF_NS	SOCK_RAW	NSPROTO_ERROR	Error
AF_NS	SOCK_RAW	NSPROTO_RAW	RAW

Typical Scenario
Connection-Oriented Protocol



Socket
Bind()

- Servers use bind() to register their ‘‘well-known’’ address with the system
  - ‘‘This is my address and any message received for this address is for me’’
  - Both connection oriented and connection-less services execute bind() before accepting messages

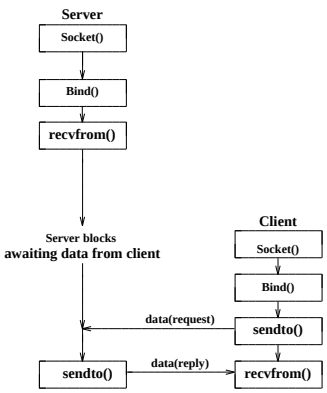
Socket
Bind()

- A client uses the command to register itself with a specific address
- The command assures a connectionless client that the system has assigned to it some unique address, so that the other end has an address to return replies to.

Socket Commands

- The client establishes a connection with the server
  - connect()
- The server has to accept a connection from the client
  - accept()

Typical Scenario
Connectionless Protocol



Connectionless Sockets Commands

- sendto() requires the address of the destination (the server) as a parameter
- The server does not have to accept a connection a client
  - The server uses the recvfrom() system call to receive data
    - The call causes the server to block until data is available
    - recvfrom() returns the network address of the client process along with the datagram
  - The server uses the sendto() system call to send data