Sockets

Sockets provide two-way *interprocess communications* between processes on one or more machines.

Sockets

Sockets provide two-way *interprocess communications* between processes on one or more machines.

connectionless each packet contains full addresses. (*datagrams*)

Sockets

Sockets provide two-way *interprocess communications* between processes on one or more machines.

connectionless each packet contains full addresses. (*datagrams*)

connection-based sender/receiver (ie. server/client) addresses used to establish a connection which simplifies (ie. shortens) packets' headers; a **virtual circuit** is established.
(data appear to be transmitted in an unbroken stream; reliable)

Sockets

Sockets provide two-way *interprocess communications* between processes on one or more machines.

connectionless each packet contains full addresses. (*datagrams*)

connection-based sender/receiver (ie. server/client) addresses used to establish a connection which simplifies (ie. shortens) packets' headers; a **virtual circuit** is established.
(data appear to be transmitted in an unbroken stream; reliable)

Sockets of both types must specify a **5-tuple**:

Sockets

Sockets provide two-way *interprocess communications* between processes on one or more machines.

connectionless each packet contains full addresses. (*datagrams*)

connection-based sender/receiver (ie. server/client) addresses used to establish a connection which simplifies (ie. shortens) packets' headers; a **virtual circuit** is established.
(data appear to be transmitted in an unbroken stream; reliable)

Sockets of both types must specify a **5-tuple**:

Type	*Protocol*	*Local-Addr* *Local-Process*	*Foreign-Addr* *Foreign-Process*
connection oriented server	socket()	bind()	listen(), accept()
connection oriented client	socket()	connect()	
connectionless server	socket()	bind()	recvfrom()
connectionless client	socket()	bind()	sendto()

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process
	socket()	creates socket: family, domain, protocol *(1-tuple)*

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process
	socket()	creates socket: family, domain, protocol *(1-tuple)*
	connect()	establishes connection to server *(5-tuple!)*

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process
	socket()	creates socket: family, domain, protocol *(1-tuple)*
	connect()	establishes connection to server *(5-tuple!)*
write()/read() send()/recv()	write()/read() send()/recv()	communications

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process
	socket()	creates socket: family, domain, protocol *(1-tuple)*
	connect()	establishes connection to server *(5-tuple!)*
write()/read() send()/recv()	write()/read() send()/recv()	communications
shutdown()	shutdown()	shuts down all or part of connection

Connection-Oriented Setup

Server	*Client*	*Description*
socket()		creates socket: family, domain, protocol *(1-tuple)*
bind()		fills in local addr, local process *(3-tuple)*
listen()		server specifies “calls waiting”: ie. qty clients that may queue for connection
accept() *blocks until connect*		creates a new 5-tuple with foreign-addr,process
	socket()	creates socket: family, domain, protocol *(1-tuple)*
	connect()	establishes connection to server *(5-tuple!)*
write()/read() send()/recv() ⟺	write()/read() send()/recv()	communications
shutdown()	shutdown()	shuts down all or part of connection
close()	close()	closes associated file descriptor

Connectionless Setup

socket() socket() specifies protocol *(1-tuple)*

Connectionless Setup

socket()	socket()	specifies protocol *(1-tuple)*
bind()	bind()	specifies local addr,process *(3-tuple)*

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	$\Longleftrightarrow$	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	⟺	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

- Connectionless packets may get routed different ways and so may arrive out of sequence

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	⟺	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

- Connectionless packets may get routed different ways and so may arrive out of sequence
- Packets compete for network bandwidth and may be dropped due to “collisions”

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	⟺	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

- Connectionless packets may get routed different ways and so may arrive out of sequence
- Packets compete for network bandwidth and may be dropped due to “collisions”
- Connection-oriented communications are built from connectionless services

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	⟺	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

- Connectionless packets may get routed different ways and so may arrive out of sequence
- Packets compete for network bandwidth and may be dropped due to “collisions”
- Connection-oriented communications are built from connectionless services
- Considerable effort has gone to eliminate these issues from connection oriented packets

Connectionless Setup

socket()		socket()	specifies protocol *(1-tuple)*
bind()		bind()	specifies local addr,process *(3-tuple)*
recvfrom()/sendto()	⟺	recvfrom()/sendto()	specifies *(5-tuple)* foreign addr,process

- Connectionless packets may get routed different ways and so may arrive out of sequence
- Packets compete for network bandwidth and may be dropped due to “collisions”
- Connection-oriented communications are built from connectionless services
- Considerable effort has gone to eliminate these issues from connection oriented packets
- If you want reliable, properly sequenced packets, don’t re-invent the wheel: use connection-oriented communications

socket()

#include <sys/types.h>

#include <sys/types.h>

#include <sys/socket.h>

int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor

socket()

#include <sys/types.h>

#include <sys/types.h>

#include <sys/socket.h>

int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- “AF” stands for “***Address Family*** ”:

socket()

#include <sys/types.h>

#include <sys/types.h>

#include <sys/socket.h>

int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer
types	SOCK_STREAM	stream (connection oriented)

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer
types	SOCK_STREAM	stream (connection oriented)
	SOCK_DGRAM	datagram (connectionless)

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family*** ":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer
types	SOCK_STREAM	stream (connection oriented)
	SOCK_DGRAM	datagram (connectionless)
	SOCK_RAW	raw network protocol

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family***":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer
types	SOCK_STREAM	stream (connection oriented)
	SOCK_DGRAM	datagram (connectionless)
	SOCK_RAW	raw network protocol
	SOCK_SEQPACKET	sequenced reliable 2-way packet, fixed max length

socket()

#include <sys/types.h>
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol) *(1-tuple)*

- Returns -1 on failure, + for socket descriptor
- "AF" stands for "***Address Family***":

domains	AF_UNIX	unix internal
	AF_INET	internet
	AF_NS	Novell/Xerox NS Protocol
	AF_IMPLINK	IMP link layer
types	SOCK_STREAM	stream (connection oriented)
	SOCK_DGRAM	datagram (connectionless)
	SOCK_RAW	raw network protocol
	SOCK_SEQPACKET	sequenced reliable 2-way packet, fixed max length
	SOCK_RDM	reliably delivered packet, (but not sequencing)

socket(), con’t.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM			
SOCK_DGRAM			
SOCK_RAW			
SOCK_SEQPACKET			

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√		
SOCK_DGRAM	√		
SOCK_RAW			
SOCK_SEQPACKET			

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	
SOCK_DGRAM	√	UDP	
SOCK_RAW		IP	
SOCK_SEQPACKET			

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	SPP
SOCK_DGRAM	√	UDP	IDP
SOCK_RAW		IP	√
SOCK_SEQPACKET			SPP

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	SPP
SOCK_DGRAM	√	UDP	IDP
SOCK_RAW		IP	√
SOCK_SEQPACKET			SPP

- Sockets of type SOCK_STREAM are full-duplex byte streams and resemble pipes.

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	SPP
SOCK_DGRAM	√	UDP	IDP
SOCK_RAW		IP	√
SOCK_SEQPACKET			SPP

- Sockets of type SOCK_STREAM are full-duplex byte streams and resemble pipes.
- SOCK_STREAM communications do not preserve record boundaries

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	SPP
SOCK_DGRAM	√	UDP	IDP
SOCK_RAW		IP	√
SOCK_SEQPACKET			SPP

- Sockets of type SOCK_STREAM are full-duplex byte streams and resemble pipes.
- SOCK_STREAM communications do not preserve record boundaries
- SOCK_DGRAMs do preserve record boundaries; they are not reliable nor sequenced

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

Socket types are not implemented by all protocol families:

Valid Type/Domain/Protocol Combinations

type	*AF_UNIX*	*AF_INET*	*AF_NS*
SOCK_STREAM	√	TCP	SPP
SOCK_DGRAM	√	UDP	IDP
SOCK_RAW		IP	√
SOCK_SEQPACKET			SPP

- Sockets of type SOCK_STREAM are full-duplex byte streams and resemble pipes.
- SOCK_STREAM communications do not preserve record boundaries
- SOCK_DGRAMs do preserve record boundaries; they are not reliable nor sequenced
- SOCK_RAW sockets resemble SOCK_DGRAMs; they send datagrams via sendto()/recvfrom() (common use: ping)

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

protocol: usually zero (socket() will pick an appropriate one)

Family	*Type*	*Protocol*	*Actual Protocol*

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

protocol: usually zero (socket() will pick an appropriate one)

Family	*Type*	*Protocol*	*Actual Protocol*
AF_INET			
AF_NS			

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

protocol: usually zero (socket() will pick an appropriate one)

Family	*Type*	*Protocol*	*Actual Protocol*
AF_INET	SOCK_DGRAM		
	SOCK_STREAM		
	SOCK_RAW		
	SOCK_RAW		
AF_NS	SOCK_STREAM		
	SOCK_SEQPACKET		
	SOCK_RAW		
	SOCK_RAW		

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

protocol: usually zero (socket() will pick an appropriate one)

Family	*Type*	*Protocol*	*Actual Protocol*
AF_INET	SOCK_DGRAM	IPPROTO_UDP	
	SOCK_STREAM	IPPROTO_TCP	
	SOCK_RAW	IPPROTO_ICMP	
	SOCK_RAW	IPPROTO_RAW	
AF_NS	SOCK_STREAM	NSPROTO_SPP	
	SOCK_SEQPACKET	NSPROTO_SPP	
	SOCK_RAW	NSPROTO_ERROR	
	SOCK_RAW	NSPROTO_RAW	

socket(), con't.

#include <sys/types.h>

#include <sys/socket.h>

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

protocol: usually zero (socket() will pick an appropriate one)

Family	*Type*	*Protocol*	*Actual Protocol*
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
	SOCK_STREAM	IPPROTO_TCP	TCP
	SOCK_RAW	IPPROTO_ICMP	ICMP
	SOCK_RAW	IPPROTO_RAW	(raw)
AF_NS	SOCK_STREAM	NSPROTO_SPP	SPP
	SOCK_SEQPACKET	NSPROTO_SPP	SPP
	SOCK_RAW	NSPROTO_ERROR	(error)
	SOCK_RAW	NSPROTO_RAW	(raw)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)
ICMP	Internet Control Message Protocol (handles error and control between gateways and hosts, uses IP)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)
ICMP	Internet Control Message Protocol (handles error and control between gateways and hosts, uses IP)
IP	Internet Protocol. Provides packet delivery for TCP, UDP, ICMP.

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)
ICMP	Internet Control Message Protocol (handles error and control between gateways and hosts, uses IP)
IP	Internet Protocol. Provides packet delivery for TCP, UDP, ICMP.
ARP	Address Resolution Protocol (maps internet address to hardware address)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)
ICMP	Internet Control Message Protocol (handles error and control between gateways and hosts, uses IP)
IP	Internet Protocol. Provides packet delivery for TCP, UDP, ICMP.
ARP	Address Resolution Protocol (maps internet address to hardware address)
RARP	Reverse Address Resolution Protocol (maps hardware address to internet address)

Socket Definitions

UDP	User Data Protocol (connectionless, no guarantee of delivery, uses IP)
TCP	Transmission Control Protocol (connection, reliable, full duplex, byte stream, uses IP)
ICMP	Internet Control Message Protocol (handles error and control between gateways and hosts, uses IP)
IP	Internet Protocol. Provides packet delivery for TCP, UDP, ICMP.
ARP	Address Resolution Protocol (maps internet address to hardware address)
RARP	Reverse Address Resolution Protocol (maps hardware address to internet address)
SPP	Sequenced Packet Protocol (a Xerox Network Systems protocol for reliable flow-controlled packet delivery)

Open Systems Interconnection Model

- The International Organization for Standardization *(ISO)* developed an *abstract model* of networking (called the *Basic Reference Model*)

Open Systems Interconnection Model

- The International Organization for Standardization *(ISO)* developed an *abstract model* of networking (called the *Basic Reference Model*)
- The model consistes of seven layers *(see next slide)*

Open Systems Interconnection Model

- The International Organization for Standardization *(ISO)* developed an *abstract model* of networking (called the *Basic Reference Model*)
- The model consistes of seven layers *(see next slide)*
- Each layer only interacts directly with the layer directly beneath it

OSI Layers

Open Systems Interconnection Model

Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application

OSI Layers

Open Systems Interconnection Model

Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data

OSI Layers

Open Systems Interconnection Model

Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data
Data	5. Session	Interhost communication, managing sessions between applications

OSI Layers

Open Systems Interconnection Model

Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data
Data	5. Session	Interhost communication, managing sessions between applications
Segments	4. Transport	End-to-end connections, reliability, and flow control

OSI Layers

Open Systems Interconnection Model

Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data
Data	5. Session	Interhost communication, managing sessions between applications
Segments	4. Transport	End-to-end connections, reliability, and flow control
Packet Datagram	3. Network	Path determination and logical addressing

OSI Layers

Open Systems Interconnection Model

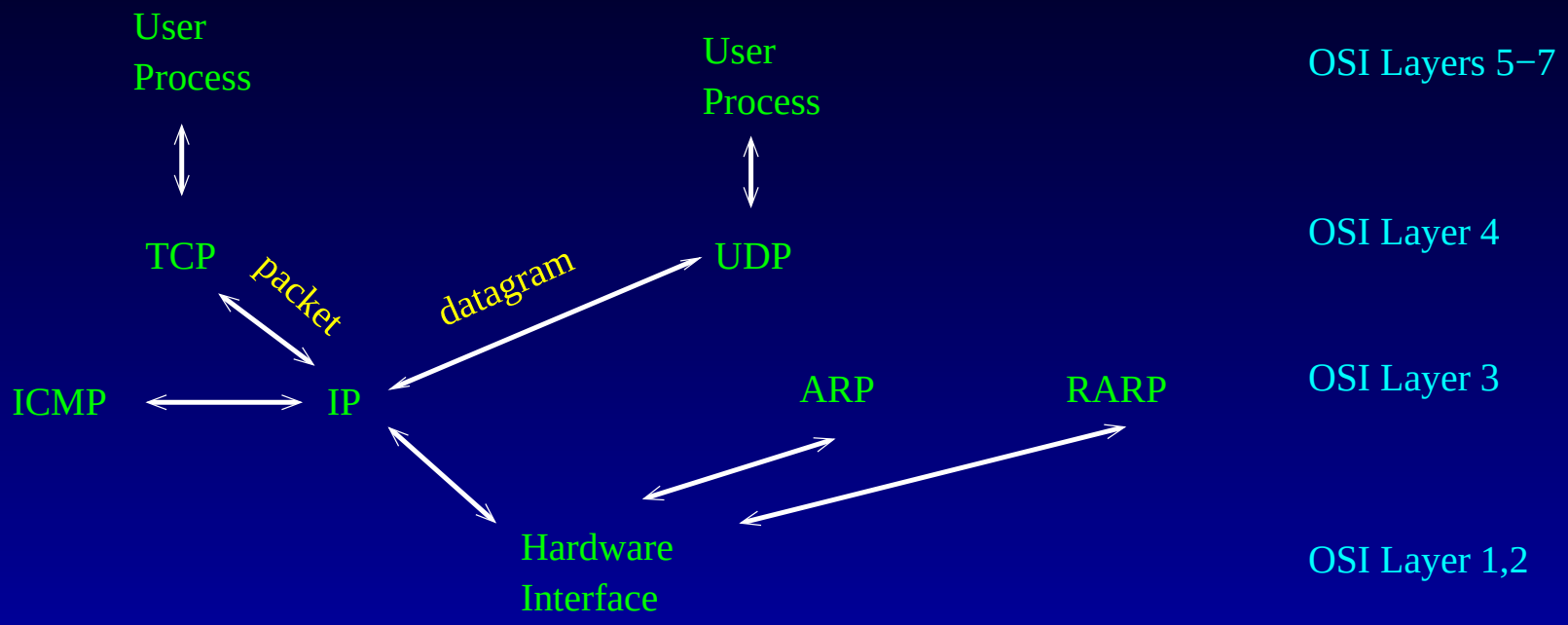
Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data
Data	5. Session	Interhost communication, managing sessions between applications
Segments	4. Transport	End-to-end connections, reliability, and flow control
Packet Datagram	3. Network	Path determination and logical addressing
Frame	2. Data link	Physical addressing

OSI Layers

Open Systems Interconnection Model

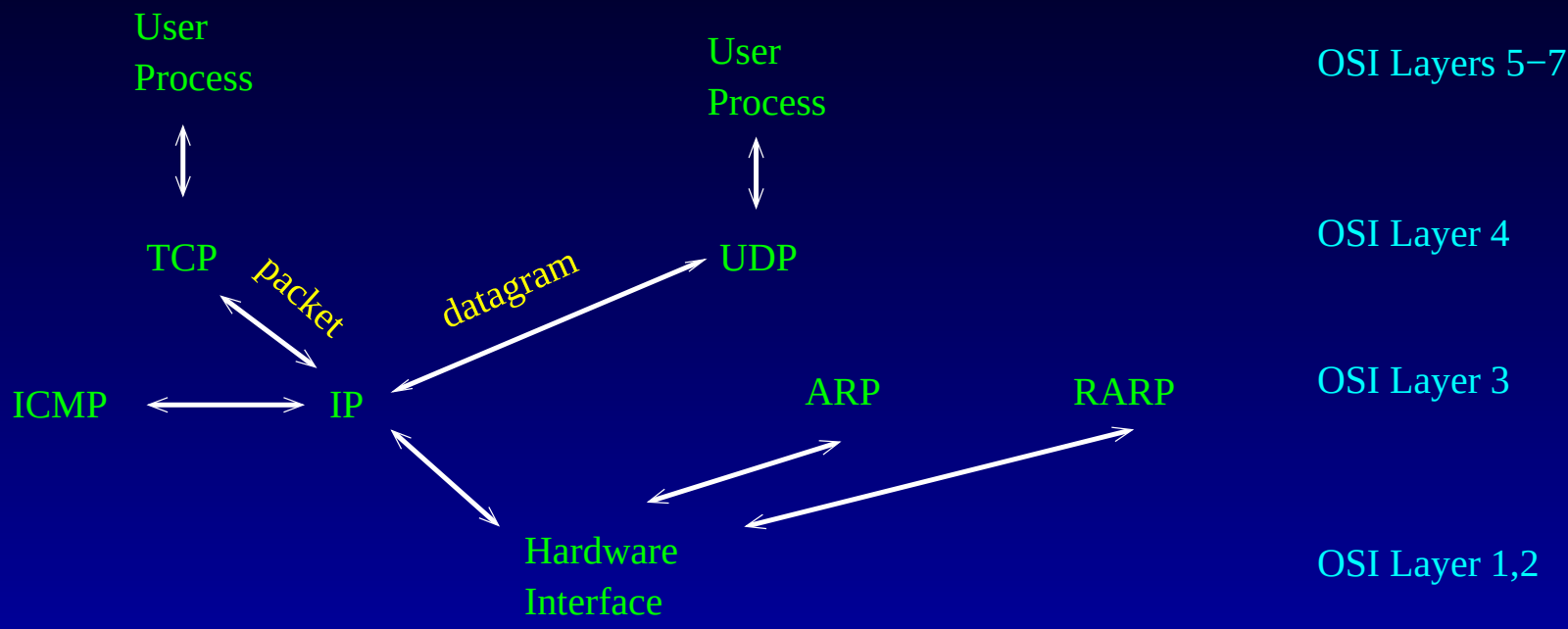
Data Unit	*Layer*	*Function*
Data	7. Application	Network process to application
Data	6. Presentation	Data representation, encryption/decryption, convert dependent data to machine independent data
Data	5. Session	Interhost communication, managing sessions between applications
Segments	4. Transport	End-to-end connections, reliability, and flow control
Packet Datagram	3. Network	Path determination and logical addressing
Frame	2. Data link	Physical addressing
Bit	1. Physical	Media, signal, and binary transmission

Layers and Protocols



- IP handles routing. May fragment a packet into multiple packets if the next network can't handle the packet's size, but it will be re-assembled upon delivery

Layers and Protocols



- IP handles routing. May fragment a packet into multiple packets if the next network can't handle the packet's size, but it will be re-assembled upon delivery
- If IP packets arrive so fast that they are discarded, the IP module will send a quench IPCMP to the source

Transport Layer

Feature	*IP*	*UDP*	*TCP*

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√
timeout to re-transmit			√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√
timeout to re-transmit			√
duplicate detection			√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√
timeout to re-transmit			√
duplicate detection			√
sequencing			√

Transport Layer

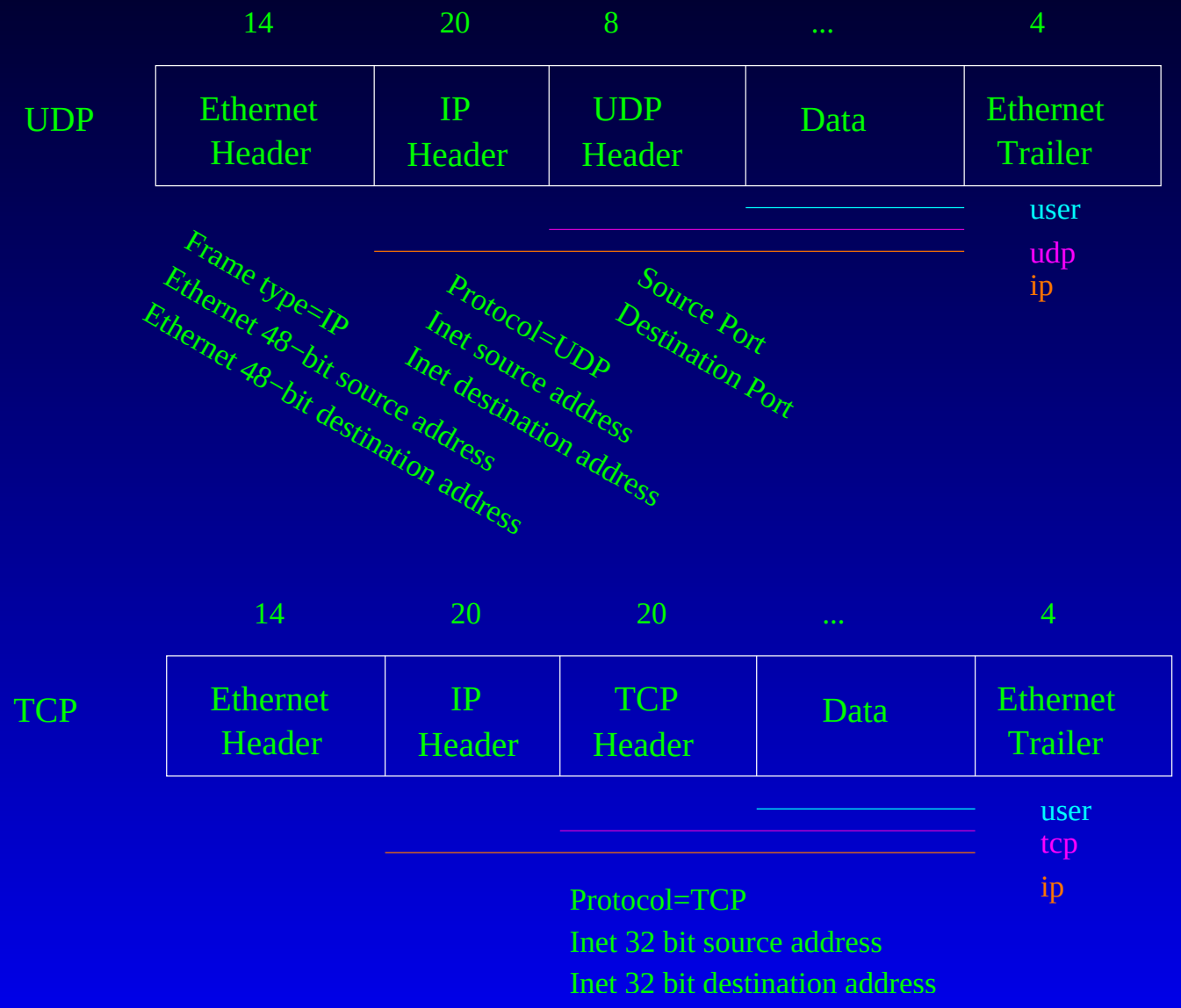
Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√
timeout to re-transmit			√
duplicate detection			√
sequencing			√
flow control			√

Transport Layer

Feature	*IP*	*UDP*	*TCP*
connection-oriented			√
message boundaries	√	√	
data checksum		optional	√
+ acknowledge			√
timeout to re-transmit			√
duplicate detection			√
sequencing			√
flow control			√

UDP provides port numbers and an optional checksum over IP

Data Packaging



Clearly, transmission of user data is most efficient when the data section is as large as possible.

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.
- Bind assigns the address specified by addr

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.
- Bind assigns the address specified by addr
- You'll need to use accept() to assign a local address for a full 5-tuple *(for servers)*

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.
- Bind assigns the address specified by addr
- You'll need to use accept() to assign a local address for a full 5-tuple *(for servers)*
- For addrlen: use sizeof()

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.
- Bind assigns the address specified by addr
- You'll need to use accept() to assign a local address for a full 5-tuple *(for servers)*
- For addrlen: use sizeof()
- server: sin_addr.s_addr= htonl(INADDR_ANY);

Sockets: bind()

#include <sys/types.h>

#include <sys/socket.h> *(1-tuple → 3-tuple)*

int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

- When a socket is created with socket(), it has an address family (name space) but no address assigned to it.
- Bind assigns the address specified by addr
- You'll need to use accept() to assign a local address for a full 5-tuple *(for servers)*
- For addrlen: use sizeof()
- server: sin_addr.s_addr= htonl(INADDR_ANY);
- client: sin_addr= *(((struct in_addr *) gethostbyname(...))->h_addr); *(obsolete - see getaddrinfo())*

Sockets: bind(), con't.

- For Unix-style *(local)* addr:

```
struct sockaddr {
  short sun_family; // AF_UNIX
  char  sun_path[]; // unix pathname
  }
```

- For Internet-style addr:

```
struct sockaddr_in {
  short          sin_family; // AF_INET
  u_short        port;       // port number
  struct in_addr sin_addr;
  }
```

Use a cast to pass in_addr structure pointers to bind():

```
(struct sockaddr *)&sockaddrin
```

Sockets: listen()

int listen(int sockfd, int backlog);

- sockfd: socket descriptor as returned by socket()

Sockets: listen()

int listen(int sockfd, int backlog);

- sockfd: socket descriptor as returned by socket()
- backlog: qty of clients allowed to be awaiting connection. Usually maximum is 5. Often ignored.

Sockets: listen()

int listen(int sockfd, int backlog);

- sockfd: socket descriptor as returned by socket()
- backlog: qty of clients allowed to be awaiting connection.
  Usually maximum is 5. Often ignored.
- Establishes a connection-oriented socket (SOCK_STREAM or SEQPACKET)

Sockets: listen()

int listen(int sockfd, int backlog);

- sockfd: socket descriptor as returned by socket()
- backlog: qty of clients allowed to be awaiting connection.
  Usually maximum is 5. Often ignored.
- Establishes a connection-oriented socket (SOCK_STREAM or SEQPACKET)
- The socket referred to by sockfd is a passive socket; ie. it needs accept() to complete the 5-tuple

Sockets: listen()

int listen(int sockfd, int backlog);

- sockfd: socket descriptor as returned by socket()
- backlog: qty of clients allowed to be awaiting connection.
  Usually maximum is 5. Often ignored.
- Establishes a connection-oriented socket (SOCK_STREAM or SEQPACKET)
- The socket referred to by sockfd is a passive socket; ie. it needs accept() to complete the 5-tuple
- If a connection request arrives, but the backlog queue is full, the client may receive a -1/ECONNREFUSED error.

sockaddr

```
struct sockaddr {
  unsigned short sa_family;     // address family, AF_xxx
  char           sa_data[14];   // 14 bytes of protocol address
  }
struct in_addr {                // IPv4 sockets
  unsigned long  s_addr;        // load this with inet_pton()
  }
struct sockaddr_in6 {           // IPv6 sockets
  u_int16_t      sin6_family;   // address family, AF_INET6
  u_int16_t      sin6_port;     // port number, in network byte order
  u_int32_t      sin6_scope_id; // scope id
  }
struct in6_addr {               // IPv6 address
  unsigned char  s6_addr[16];   // load with inet_pton()
  }
```

Many socket functions use one of the above structures cast to a pointer to `struct sockaddr`.

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures.

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures. It converts src into a network address structure, and then copies that structure to dst.

af Address family specification:

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures. It converts src into a network address structure, and then copies that structure to dst.

af Address family specification:

AF_INET Converts an IPv4 dotted-decimal format, “ddd.ddd.ddd.ddd” into a struct in_addr.

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures. It converts src into a network address structure, and then copies that structure to dst.

af Address family specification:

AF_INET Converts an IPv4 dotted-decimal format, “ddd.ddd.ddd.ddd” into a struct in_addr.

AF_INET6 Converts an IPv6 colon-hexadecimal format, “x:x:x:x:x:x:x:x” into struct in6_addr.

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures. It converts src into a network address structure, and then copies that structure to dst.

af Address family specification:

AF_INET Converts an IPv4 dotted-decimal format, “ddd.ddd.ddd.ddd” into a struct in_addr.

AF_INET6 Converts an IPv6 colon-hexadecimal format, “x:x:x:x:x:x:x:x” into struct in6_addr.

src A source string describing a numerically-based network address

Utilities: inet_pton()

#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

This function may be used to set up components in various sockaddr family structures. It converts src into a network address structure, and then copies that structure to dst.

af Address family specification:

AF_INET Converts an IPv4 dotted-decimal format, “ddd.ddd.ddd.ddd” into a struct in_addr.

AF_INET6 Converts an IPv6 colon-hexadecimal format, “x:x:x:x:x:x:x:x” into struct in6_addr.

src A source string describing a numerically-based network address

dst Destination structure address; for AF_INET: must be sizeof(struct in_addr) bytes long (4 bytes).

For AF_INET6: must be sizeof(struct in6_addr) bytes long (16 bytes)

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.
- addrlen: use sizeof(*addr)

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.
- addrlen: use sizeof(*addr)
- Connection-oriented clients connect via this call, once. May block.

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.
- addrlen: use sizeof(*addr)
- Connection-oriented clients connect via this call, once. May block.
- If the socket sockfd is of type SOCK_DGRAM, then the addr becomes the default send-to address.

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.
- addrlen: use sizeof(*addr)
- Connection-oriented clients connect via this call, once. May block.
- If the socket sockfd is of type SOCK_DGRAM, then the addr becomes the default send-to address.
- Connectionless sockets may use this call multiple times.

Sockets: connect()

(1-tuple → 5-tuple)

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

- sockfd: socket descriptor
- addr: Use either a struct sockaddr * or a struct sockaddr_in *.
- addrlen: use sizeof(*addr)
- Connection-oriented clients connect via this call, once. May block.
- If the socket sockfd is of type SOCK_DGRAM, then the addr becomes the default send-to address.
- Connectionless sockets may use this call multiple times.
- Connectionless sockets may use sa_family set to AF_UNSPEC to dissolve the association.

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket
- The input sockfd is unaffected by the call, and may be used repeatedly

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket
- The input sockfd is unaffected by the call, and may be used repeatedly
- addr: use struct sockaddr * or struct sockaddr_in *, as appropriate (cast the latter)

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket
- The input sockfd is unaffected by the call, and may be used repeatedly
- addr: use struct sockaddr * or struct sockaddr_in *, as appropriate (cast the latter)
- Presumes that sockaddr had been created with socket(), bound to a local address with bind(), and listen() has been used.

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket
- The input sockfd is unaffected by the call, and may be used repeatedly
- addr: use struct sockaddr * or struct sockaddr_in *, as appropriate (cast the latter)
- Presumes that sockaddr had been created with socket(), bound to a local address with bind(), and listen() has been used.
- If no pending connections are on the queue, and sockaddr is *not* non-blocking, accept() will block.

Sockets: accept()

(3-tuple → 5-tuple)

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

- Used with connection-based sockets (servers)
- Extracts first connection request waiting on the listening queue
- The call creates a new connected socket, returning a file descriptor for that socket
- The input sockfd is unaffected by the call, and may be used repeatedly
- addr: use struct sockaddr * or struct sockaddr_in *, as appropriate (cast the latter)
- Presumes that sockaddr had been created with socket(), bound to a local address with bind(), and listen() has been used.
- If no pending connections are on the queue, and sockaddr is *not* non-blocking, accept() will block.
- If no pending connections are on the queue, but sockaddr is non-blocking, accept() will return a -1/EAGAIN.

Sockets: send()

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

sockfd socket descriptor

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

len qty of bytes in buffer to be sent

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

len qty of bytes in buffer to be sent

flags 0=normal; MSG_OOB: out-of-band (high priority)

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

len qty of bytes in buffer to be sent

flags 0=normal; MSG_OOB: out-of-band (high priority)

- The send() call expects the sockfd to be in an already connected state (ie. the recipient is known).

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

len qty of bytes in buffer to be sent

flags 0=normal; MSG_OOB: out-of-band (high priority)

- The send() call expects the sockfd to be in an already connected state (ie. the recipient is known).
- Returns -1 on failure; otherwise, a positive number is the qty of bytes actually sent. *may be less then len!*

Sockets: send()

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

sockfd socket descriptor

buf buffer of bytes to be sent

len qty of bytes in buffer to be sent

flags 0=normal; MSG_OOB: out-of-band (high priority)

- The send() call expects the sockfd to be in an already connected state (ie. the recipient is known).
- Returns -1 on failure; otherwise, a positive number is the qty of bytes actually sent. *may be less then len!*
- Equivalent to sendto(sockfd,buf,len,flags,NULL,0) (see next slide)

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

dest_addr specifies the destination

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

dest_addr specifies the destination

addrlen specifies the size of dest_addr

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

dest_addr specifies the destination

addrlen specifies the size of dest_addr

- If sendto() is used on a connection-based socket (SOCK_STREAM, SOCK_SEQPACKET), then the last two arguments are ignored.

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

dest_addr specifies the destination

addrlen specifies the size of dest_addr

- If sendto() is used on a connection-based socket (SOCK_STREAM, SOCK_SEQPACKET), then the last two arguments are ignored.
- Otherwise, for connection-less sockets such as SOCK_DGRAM (datagram), the recipient's address is specified by dest_addr and addrlen.

Sockets: sendto()

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

sockfd...flags same as send()'s

dest_addr specifies the destination

addrlen specifies the size of dest_addr

- If sendto() is used on a connection-based socket (SOCK_STREAM, SOCK_SEQPACKET), then the last two arguments are ignored.
- Otherwise, for connection-less sockets such as SOCK_DGRAM (datagram), the recipient's address is specified by dest_addr and addrlen.
- If the message is too long, sendto() will return an error EMSGSIZE; the message is not transmitted.

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

MSG_PEEK allows a peek at the bytes, leaving them for subsequent retrieval

(doesn't remove them from the stream)

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

MSG_PEEK allows a peek at the bytes, leaving them for subsequent retrieval

(doesn't remove them from the stream)

MSG_OOB requests receipt of out-of-band data (which is not normally received)

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

MSG_PEEK allows a peek at the bytes, leaving them for subsequent retrieval (doesn't remove them from the stream)

MSG_OOB requests receipt of out-of-band data (which is not normally received)

- Will block (unless the socket has been set to non-blocking, see fcntl())

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

MSG_PEEK allows a peek at the bytes, leaving them for subsequent retrieval (doesn't remove them from the stream)

MSG_OOB requests receipt of out-of-band data (which is not normally received)

- Will block (unless the socket has been set to non-blocking, see fcntl())
- If nonblocking, a -1 will be returned with errno set to EAGAIN.

Sockets: recv()

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

sockfd the socket descriptor for a connected socket

buf, len receives data in this buffer, up to len bytes

flags

0 normal

MSG_PEEK allows a peek at the bytes, leaving them for subsequent retrieval (doesn't remove them from the stream)

MSG_OOB requests receipt of out-of-band data (which is not normally received)

- Will block (unless the socket has been set to non-blocking, see fcntl())
- If nonblocking, a -1 will be returned with errno set to EAGAIN.
- Returns -1 on failure, or a positive qty of bytes actually received (which may be less than those sent!)

Sockets: recvfrom()

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,struct sockaddr *src_addr, socklen_t *addrlen);

sid...flag just like recv()'s

Sockets: recvfrom()

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,struct sockaddr *src_addr, socklen_t *addrlen);

sid...flag just like recv()'s

src_addr, addrlen specifies the sender's address.

Sockets: recvfrom()

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,struct sockaddr *src_addr, socklen_t *addrlen);

sid...flag just like recv()'s

src_addr, addrlen specifies the sender's address.

- Returns like recv()

Sockets: recvfrom()

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,struct sockaddr *src_addr, socklen_t *addrlen);

sid...flag just like recv()'s

src_addr, addrlen specifies the sender's address.

- Returns like recv()
- Primarily for SOCK_DGRAM (datagrams)

Sockets: shutdown()

int shutdown(int sockfd, int how);

sockfd socket descriptor to be shut down

Sockets: shutdown()

int shutdown(int sockfd, int how);

sockfd socket descriptor to be shut down

how controls partial shutdowns

Sockets: shutdown()

int shutdown(int sockfd, int how);

sockfd socket descriptor to be shut down

how controls partial shutdowns

SHUT_RD further reading (receptions) disallowed

Sockets: shutdown()

int shutdown(int sockfd, int how);

sockfd socket descriptor to be shut down

how controls partial shutdowns

SHUT_RD	further reading (receptions) disallowed
SHUT_WR	further writing (transmissions) disallowed

Sockets: shutdown()

int shutdown(int sockfd, int how);

sockfd socket descriptor to be shut down

how controls partial shutdowns

SHUT_RD	further reading (receptions) disallowed
SHUT_WR	further writing (transmissions) disallowed
SHUT_RDWR	subsequent reading and writing via sockfd is disallowed

Sockets: close()

#include <unistd.h>

int close(int fd);

- This call closes file descriptors, including socket descriptors.

Sockets: close()

#include <unistd.h>

int close(int fd);

- This call closes file descriptors, including socket descriptors.
- The descriptor will no longer refer to any file or socket and may be re-used.

Sockets: close()

#include <unistd.h>

int close(int fd);

- This call closes file descriptors, including socket descriptors.
- The descriptor will no longer refer to any file or socket and may be re-used.
- If fd is the last file descriptor holding open the file, then resources associated with the open file descriptor are freed.

(see connection.c packet.c)

Utilities: gethostbyname()

```
#include <netdb.h>
extern int h_errno;
struct hostent *gethostbyname(const char *name);
#define h_addr h_addr_list[0]
```

name the name of the host, using IPv4 standard dot notation or in IPv6 notation; in either colon or dot notation accepted.

- Returns a struct hostent *(see next slide)*
- A host may have more than one address if it is “multihomed” (ie. the machine has more than one internet interface, each with its own address)
- Considered to be obsolete; use getaddrinfo() instead

Utilities: The hostent Structure

```
struct hostent {
  char  *h_name;       // official name of host
  char **h_aliases;    // alias list
  int    h_length;     // length of address
  char **h_addr_host;  // list of addresses from name server
                       // (a NULL terminated list)
  }
```

Socket Code Fragment

```
struct sockaddr_in  sin; // internet socket addressing
struct hostent     *phostent= NULL;

// open a socket for clients
sktfd= socket(AF_INET,SOCK_STREAM,0);

// initialize the sockaddr_in
memset(&sin,0x00,sizeof(struct sockaddr_in));
sin.sin_family = AF_INET;
sin.sin_addr   = *((struct in_addr *) phostent->h_addr;
sin.sin_port   = htons((u_short) port);

// get a hostent structure
phostent       = gethostbyname(hostname);

// connect
if(connect(sktfd,(struct sockaddr*) &sin,sizeof(sin)) < 0){
  perror("");
  }
```

Utilities: IP and Name Lookups

Transformation	Obsolete Functions	New Functions
name → IP	gethostbyname()	getaddrinfo()
IP → name	gethostbyaddr()	getnameinfo()

The newer functions can use IPv4 or IPv6 transparently, and they can handle services such as “http” as well as port numbers.

Utilities: getaddrinfo()

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

node numerical network address or a hostname string

Utilities: getaddrinfo()

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

node numerical network address or a hostname string

service sets the port; service name (see man 5 services). May be null.

Utilities: getaddrinfo()

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

node numerical network address or a hostname string

service sets the port; service name (see man 5 services). May be null.

hints See struct addrinfo below. Specifies criteria for selecting socket address structure in the list pointed to by res

Utilities: getaddrinfo()

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

node numerical network address or a hostname string

service sets the port; service name (see man 5 services). May be null.

hints See struct addrinfo below. Specifies criteria for selecting socket address structure in the list pointed to by res

res A pointer to the head of a linked list of addrinfo structures. ai_next gives the next link.

Utilities: getaddrinfo()

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

node numerical network address or a hostname string

service sets the port; service name (see man 5 services). May be null.

hints See struct addrinfo below. Specifies criteria for selecting socket address structure in the list pointed to by res

res A pointer to the head of a linked list of addrinfo structures. ai_next gives the next link.

```
struct addrinfo {
  int                 ai_flags;
  int                 ai_family;
  int                 ai_socktype;
  int                 ai_protocol;
  size_t              ai_addrlen;
  struct sockaddr   * ai_addr;
  char              * ai_canonname;
  struct addrinfo   * ai_next;
  }
```

see next slide

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant
- ai_family, ai_socktype, and si_protocol mean the same as with socket()

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant
- ai_family, ai_socktype, and si_protocol mean the same as with socket()
- Pointer to the socket address is placed in ai_addr

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant
- ai_family, ai_socktype, and si_protocol mean the same as with socket()
- Pointer to the socket address is placed in ai_addr
- Length of socket address (in bytes) is placed in ai_addrlen

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant
- ai_family, ai_socktype, and si_protocol mean the same as with socket()
- Pointer to the socket address is placed in ai_addr
- Length of socket address (in bytes) is placed in ai_addrlen
- freeaddrinfo(struct addrinfo *res) frees up the res structure linked list

Utilities: getaddrinfo(), con't.

- Identifies an internet host or service
- Returns one or more addrinfo structures, each of which contains an internet address that can be used in bind() or connect().
- Reentrant
- ai_family, ai_socktype, and si_protocol mean the same as with socket()
- Pointer to the socket address is placed in ai_addr
- Length of socket address (in bytes) is placed in ai_addrlen
- freeaddrinfo(struct addrinfo *res) frees up the res structure linked list

(see using_getaddrinfo.c)

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

serv Pointer to caller-allocated buffer with a null-terminated service name

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

serv Pointer to caller-allocated buffer with a null-terminated service name

servlen Length of serv buffer

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

serv Pointer to caller-allocated buffer with a null-terminated service name

servlen Length of serv buffer

flags Modifies behavior as follows: (see next slide)

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

serv Pointer to caller-allocated buffer with a null-terminated service name

servlen Length of serv buffer

flags Modifies behavior as follows: (see next slide)

- getnameinfo() converts a socket address into a corresponding host and service, depending on the protocol.

Utilities: getnameinfo()

int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);

sa (input) pointer to a generic socket address structure (either sockaddr_in or sockaddr_in6)

salen sa is of size salen.

host Pointer to caller-allocated buffer with a null-terminated host name

hostlen Length of host buffer

serv Pointer to caller-allocated buffer with a null-terminated service name

servlen Length of serv buffer

flags Modifies behavior as follows: (see next slide)

- getnameinfo() converts a socket address into a corresponding host and service, depending on the protocol.
- The function is re-entrant, and allows programs to avoid IPv4 vs IPv6 dependencies

Utilities: getnameinfo(), con't.

flags Modifies behavior of getnameinfo() as follows:

NI_NAMEREQD An error will be returned if the hostname cannot be determined

Utilities: getnameinfo(), con't.

flags Modifies behavior of getnameinfo() as follows:

NI_NAMEREQD An error will be returned if the hostname cannot be determined

NI_DGRAM Service is a datagram (UDP). Required for those ports whose associated serves support both UDP and TCP.

Utilities: getnameinfo(), con't.

flags Modifies behavior of getnameinfo() as follows:

NI_NAMEREQD An error will be returned if the hostname cannot be determined

NI_DGRAM Service is a datagram (UDP). Required for those ports whose associated serves support both UDP and TCP.

NI_NOFQDN Return only hostname portion of fully qualified domain name for local hosts

Utilities: getnameinfo(), con't.

flags Modifies behavior of getnameinfo() as follows:

NI_NAMEREQD An error will be returned if the hostname cannot be determined

NI_DGRAM Service is a datagram (UDP). Required for those ports whose associated serves support both UDP and TCP.

NI_NOFQDN Return only hostname portion of fully qualified domain name for local hosts

NI_NUMERICHOST returns the numeric form of the hostname

Utilities: getnameinfo(), con't.

flags Modifies behavior of getnameinfo() as follows:

NI_NAMEREQD An error will be returned if the hostname cannot be determined

NI_DGRAM Service is a datagram (UDP). Required for those ports whose associated serves support both UDP and TCP.

NI_NOFQDN Return only hostname portion of fully qualified domain name for local hosts

NI_NUMERICHOST returns the numeric form of the hostname

NI_NUMERICSERV returns the numeric form of the service address

Utilities: gethostbyaddr()

struct hostent *gethostbyaddr(const void *addr,socklen_t len, int type);

addr a pointer to a struct depending on the address type (ex. struct in_addr *), which was probably obtained via a call to inet_addr()

Utilities: gethostbyaddr()

struct hostent *gethostbyaddr(const void *addr,socklen_t len, int type);

addr a pointer to a struct depending on the address type (ex. struct in_addr *), which was probably obtained via a call to inet_addr()

len length of addr structure being passed

Utilities: gethostbyaddr()

struct hostent *gethostbyaddr(const void *addr,socklen_t len, int type);

addr a pointer to a struct depending on the address type (ex. struct in_addr *), which was probably obtained via a call to inet_addr()

len length of addr structure being passed

type Valid address types include AF_INET and AF_INET6.

Utilities: gethostbyaddr()

struct hostent *gethostbyaddr(const void *addr,socklen_t len, int type);

addr a pointer to a struct depending on the address type (ex. struct in_addr *), which was probably obtained via a call to inet_addr()

len length of addr structure being passed

type Valid address types include AF_INET and AF_INET6.

- Returns a structure of type hostent (see previous slide)

Utilities: getservbyname()

#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);

- Returns a servent structure using the service name database (ie. /etc/services)

Utilities: getservbyname()

#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);

- Returns a servent structure using the service name database (ie. /etc/services)

```
struct servent {
  char      *    s_name;      official service name
  char      **   s_aliases;   alias list
  int            s_port;      port number, already in network byte order
  char      *    s_proto;     protocol to use
  }
```

Utilities: getservbyname()

#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);

- Returns a servent structure using the service name database (ie. /etc/services)

```
struct servent {
  char      *    s_name;      official service name
  char      **   s_aliases;   alias list
  int            s_port;      port number, already in network byte order
  char      *    s_proto;     protocol to use
  }
```

- To fill in a struct sockaddr_in's short sin_family: use gethostbyname()->h_addrtype

Utilities: getservbyname()

#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);

- Returns a servent structure using the service name database (ie. /etc/services)

```
struct servent {
  char      *    s_name;      official service name
  char      **   s_aliases;   alias list
  int            s_port;      port number, already in network byte order
  char      *    s_proto;     protocol to use
  }
```

- To fill in a struct sockaddr_in's short sin_family: use gethostbyname()->h_addrtype
- To fill in sin_port: gethostbyname(...)->h_addr

Utilities: Byte Ordering

#include <arpa/inet.h>

#include <netinet/in.h> **(some require this header file instead)**

uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);

- **Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.**

Utilities: Byte Ordering

#include <arpa/inet.h>

#include <netinet/in.h> **(some require this header file instead)**

uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);

- **Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.**
- **On Intel based machines, the least significant byte is first; network order has most significant byte first.**

Utilities: Byte Ordering

```
#include <arpa/inet.h>
#include <netinet/in.h> (some require this header file instead)
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.
- On Intel based machines, the least significant byte is first; network order has most significant byte first.
- htonl() converts unsigned integers from host format to network byte order

Utilities: Byte Ordering

```
#include <arpa/inet.h>
#include <netinet/in.h> (some require this header file instead)
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.
- On Intel based machines, the least significant byte is first; network order has most significant byte first.
- htonl() converts unsigned integers from host format to network byte order
- htons() converts unsigned short integers from host format to network byte order

Utilities: Byte Ordering

```
#include <arpa/inet.h>
#include <netinet/in.h> (some require this header file instead)
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.
- On Intel based machines, the least significant byte is first; network order has most significant byte first.
- htonl() converts unsigned integers from host format to network byte order
- htons() converts unsigned short integers from host format to network byte order
- ntohl() converts unsigned integers from network byte order to host format

Utilities: Byte Ordering

```
#include <arpa/inet.h>
#include <netinet/in.h> (some require this header file instead)
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Different machines can have different architectures; in particular, integers can be represented with high-byte first or last.
- On Intel based machines, the least significant byte is first; network order has most significant byte first.
- htonl() converts unsigned integers from host format to network byte order
- htons() converts unsigned short integers from host format to network byte order
- ntohl() converts unsigned integers from network byte order to host format
- ntohs() converts unsigned short integers from network byte order to host format

Utilities: Byte Operations

SysV
```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
void *memset(void *s, int c, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
```

BSD
```
#include <strings.h>
* void bcopy(const void *src, void *dest, size_t n);
* void bzero(void *s, size_t n);
* int bcmp(const void *s1, const void *s2, size_t n);
* these functions are deprecated
```

memcpy(), bcopy(), memmove() copies n bytes from src to dest. Overlapping dest/target permitted.

Utilities: Byte Operations

SysV
```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
void *memset(void *s, int c, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
```

BSD
```
#include <strings.h>
* void bcopy(const void *src, void *dest, size_t n);
* void bzero(void *s, size_t n);
* int bcmp(const void *s1, const void *s2, size_t n);
* these functions are deprecated
```

memcpy(), bcopy(), memmove() copies `n` bytes from `src` to `dest`. Overlapping dest/target permitted.

memset(), bzero() Sets `s` to the specified byte (SysV) or to zero (BSD)

Utilities: Byte Operations

SysV
```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
void *memset(void *s, int c, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
```

BSD
```
#include <strings.h>
* void bcopy(const void *src, void *dest, size_t n);
* void bzero(void *s, size_t n);
* int bcmp(const void *s1, const void *s2, size_t n);
* these functions are deprecated
```

memcpy(), bcopy(), memmove() copies `n` bytes from `src` to `dest`. Overlapping dest/target permitted.

memset(), bzero() Sets `s` to the specified byte (SysV) or to zero (BSD)

memcmp(), bcmp() Compares the first `n` bytes of `s1` and `s2`. Returns an integer greater than or less than zero for the first mismatched byte. If n bytes are identical, then zero is returned.

Utilities: Address Conversion

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
in_addr_t inet_network(const char *cp);
```

inet_aton() converts an IPv4 number and dot notation string into a struct in_addr structure.

Returns non-zero if valid, 0 otherwise.

Utilities: Address Conversion

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
in_addr_t inet_network(const char *cp);
```

inet_aton() converts an IPv4 number and dot notation string into a struct in_addr structure.

Returns non-zero if valid, 0 otherwise.

inet_ntoa() reverses inet_aton()

Utilities: Address Conversion

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
in_addr_t inet_network(const char *cp);
```

inet_aton() converts an IPv4 number and dot notation string into a struct in_addr structure.

Returns non-zero if valid, 0 otherwise.

inet_ntoa() reverses inet_aton()

inet_network() converts an IPv4 number and dot notation string into a number in network byte order.

Utilities: Address Conversion

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
in_addr_t inet_network(const char *cp);
```

inet_aton() converts an IPv4 number and dot notation string into a struct in_addr structure.

Returns non-zero if valid, 0 otherwise.

inet_ntoa() reverses inet_aton()

inet_network() converts an IPv4 number and dot notation string into a number in network byte order.

struct in_addr inet_network() returns the following structure:

```
typedef uint32_t in_addr_t;
struct in_addr {
  in_addr_t s_addr;
  };
```

Utilities: getpeername()

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockfd an open socket descriptor

Utilities: getpeername()

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockfd an open socket descriptor

struct sockaddr *addr getpeername() returns the address of the peer connected to the socket sockfd.

Utilities: getpeername()

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockfd an open socket descriptor

struct sockaddr *addr getpeername() returns the address of the peer connected to the socket sockfd.

addrlen User should initialize this to the actual size of the addr buffer being passed. On return it holds the actual size of the name (in bytes). The name will be truncated if the buffer is too small.

Utilities: getpeername()

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockfd an open socket descriptor

struct sockaddr *addr getpeername() returns the address of the peer connected to the socket sockfd.

addrlen User should initialize this to the actual size of the addr buffer being passed. On return it holds the actual size of the name (in bytes). The name will be truncated if the buffer is too small.

The getpeername() function converts a socket file descriptor into a sockaddr-based peername.

Utilities: gethostname()

```
#include <unistd.h>
int gethostname(char *name, size_t len);
```

- This function returns the null-byte terminated hostname in the name buffer, which has a length of len bytes.

Utilities: gethostname()

```
#include <unistd.h>
int gethostname(char *name, size_t len);
```

- This function returns the null-byte terminated hostname in the name buffer, which has a length of len bytes.
- The hostname will be truncated if len isn't large enough.

Utilities: gethostname()

```
#include <unistd.h>
int gethostname(char *name, size_t len);
```

- This function returns the null-byte terminated hostname in the name buffer, which has a length of len bytes.
- The hostname will be truncated if len isn't large enough.
- It may or may not have a terminating null byte in such a case.

Ports

Reserved ports	1 -1023

Ports

Reserved ports	1 -1023
Automatically assignable ports	1024-5000

Ports

Reserved ports	1 -1023
Automatically assignable ports	1024-5000
Ports assigned by rresvport()	512 -1023

Ports

Reserved ports	1 -1023
Automatically assignable ports	1024-5000
Ports assigned by rresvport()	512 -1023

- rresvport(int *port) returns a descriptor to a socket with an address in the privileged port space.
  Only the superuser privileged processes are allowed to bind an address with this sort of port to a socket.

Ports

Reserved ports	1 -1023
Automatically assignable ports	1024-5000
Ports assigned by rresvport()	512 -1023

- rresvport(int *port) returns a descriptor to a socket with an address in the privileged port space.
  Only the superuser privileged processes are allowed to bind an address with this sort of port to a socket.
- See /etc/services for a list of service names, ports, and protocol (ie. whether it uses udp or tcp)