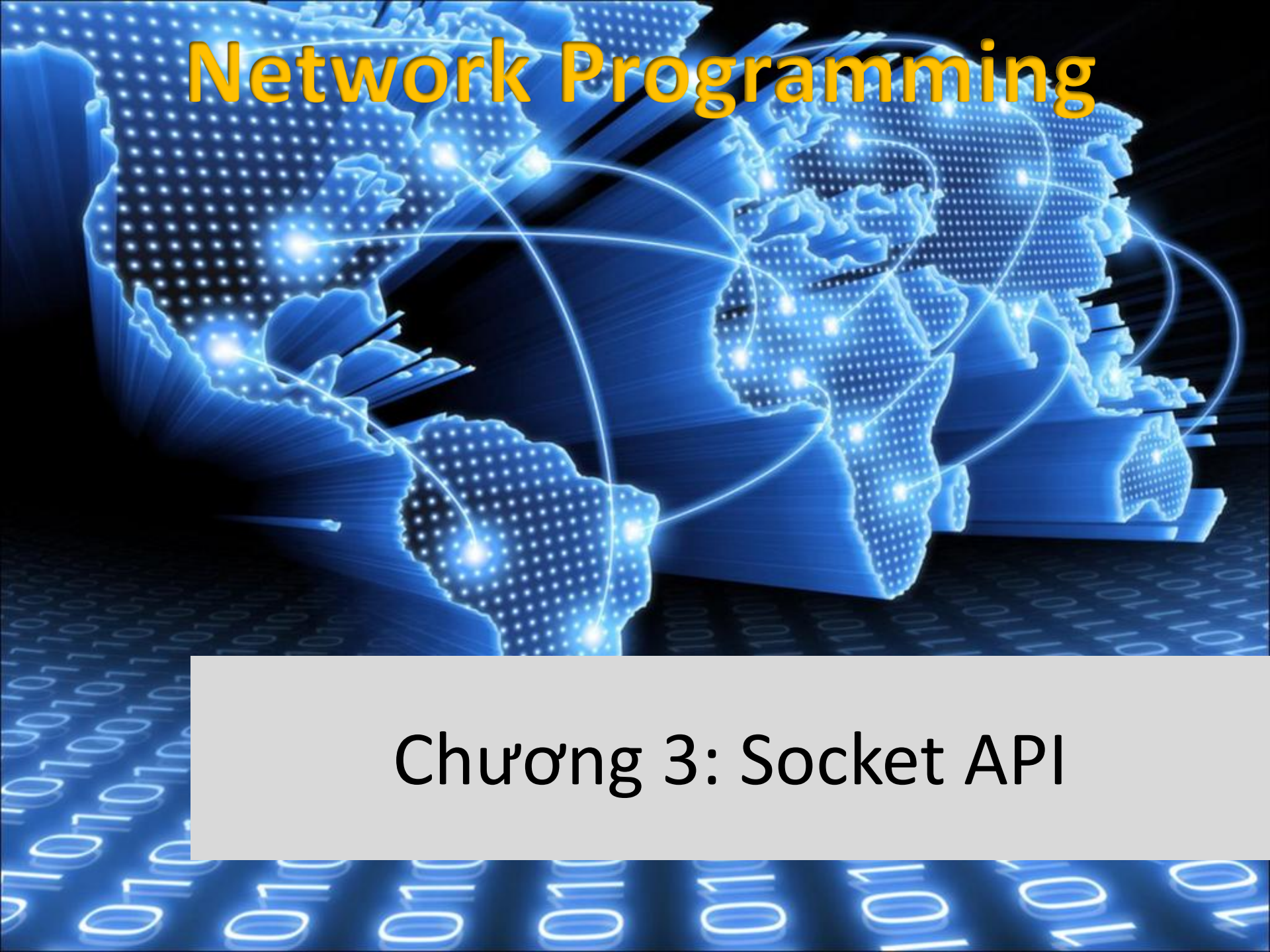


Chương 3: Socket API

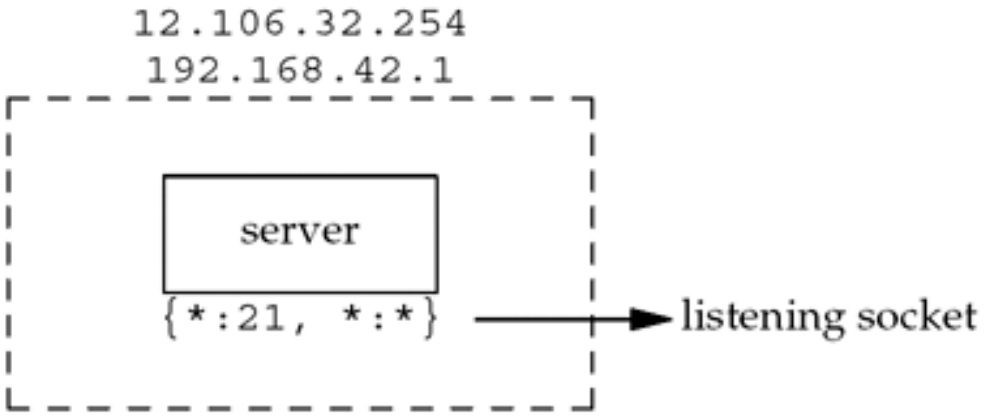
Nội dung

BASICS OF SOCKET

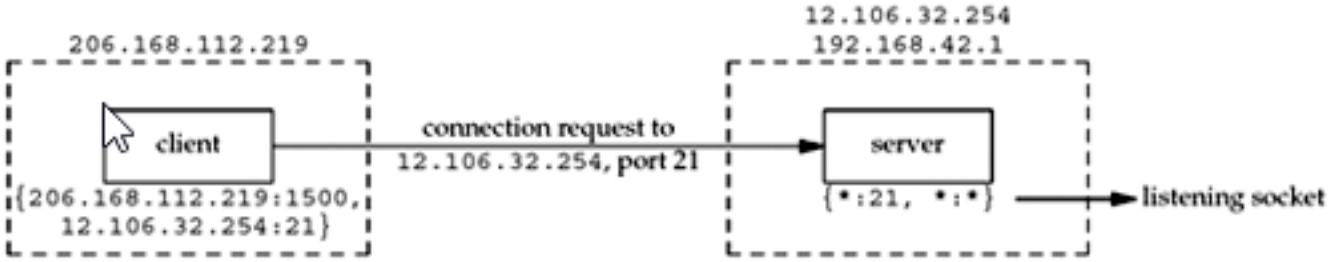
Socket Pair

- four-tuple that defines the two endpoints of the connection
  - the local IP address
  - local port
  - foreign IP address
  - foreign port
- The 2 values that identify each endpoint, an IP address and a port number, are often called a *socket*.

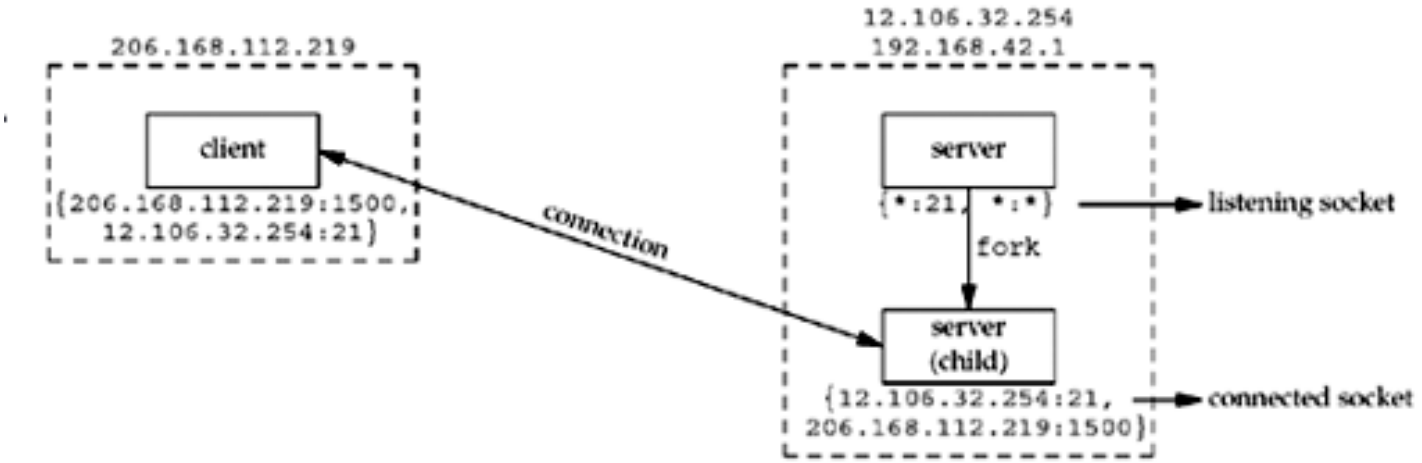
TCP Port Numbers and Concurrent Servers (1)



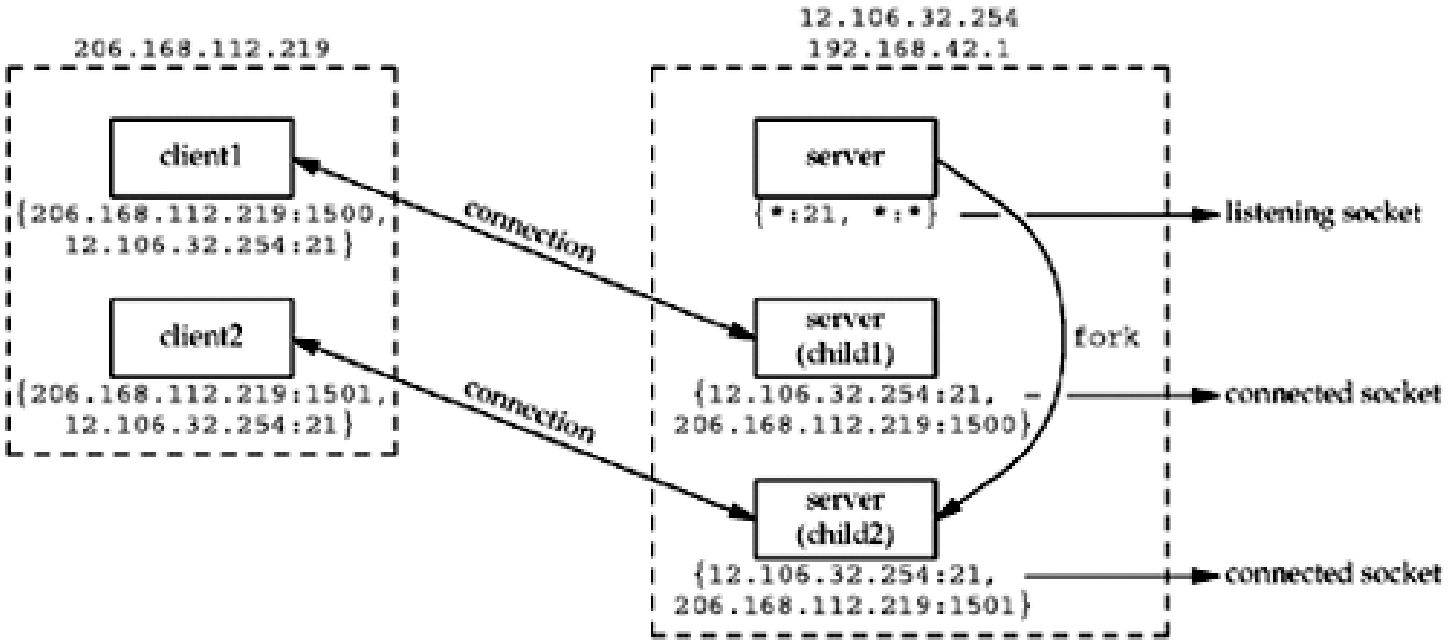
TCP Port Numbers and Concurrent Servers (2)



TCP Port Numbers and Concurrent Servers (3)



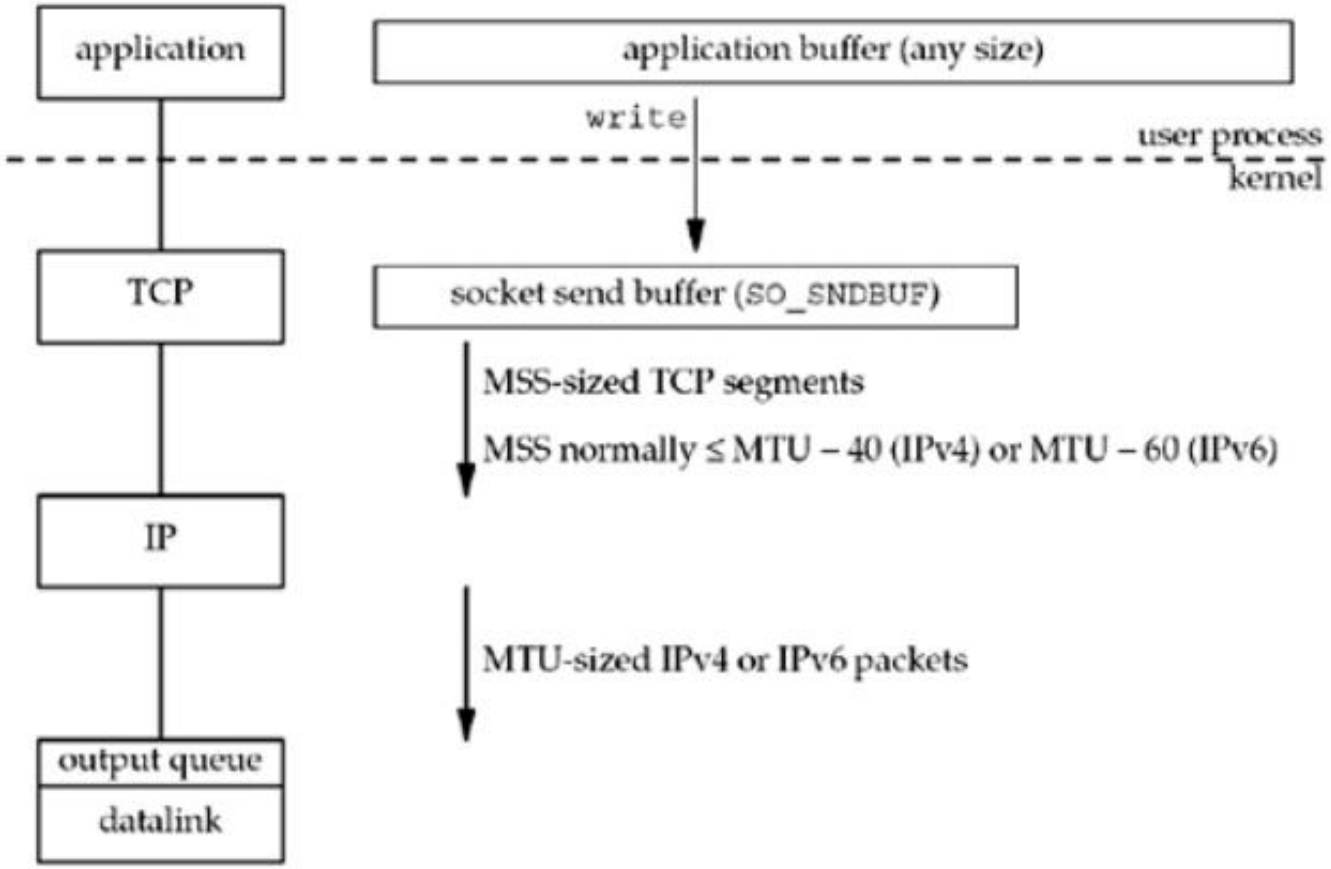
TCP Port Numbers and Concurrent Servers (4)



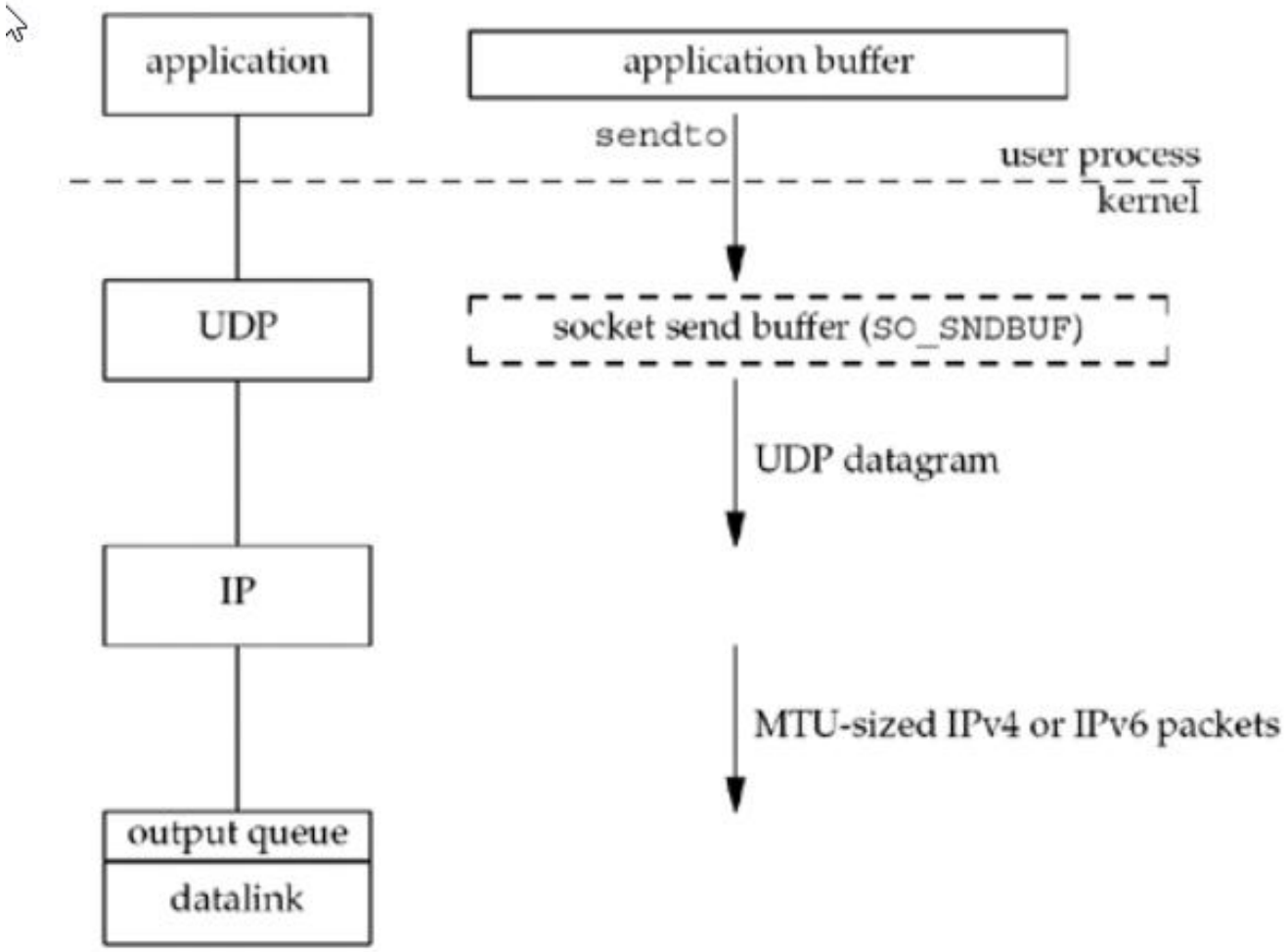
Buffer Sizes and Limitations

- Maximum size of an IPv4 datagram: 65,538 bytes
- MTU (Maximum transmission unit)
- Fragmentation when the size of the datagrram exceeds the link MTU.
  - DF bit (don’t fragment)
- MSS (maximum segment size): that announces to the peer TCP the maximum amount of TCP data that the peer can send per segment.
- MSS = MTU – fixed size of headers of IP and TCP

TCP output


application
application buffer (any size)
write
user process
kernel
TCP
socket send buffer (SO_SNDBUF)
MSS-sized TCP segments
MSS normally ≤ MTU − 40 (IPv4) or MTU − 60 (IPv6)
IP
MTU-sized IPv4 or IPv6 packets
output queue
datalink

UDP output


application
application buffer
sendto
user process
kernel
UDP
socket send buffer (SO_SNDBUF)
UDP datagram
IP
MTU-sized IPv4 or IPv6 packets
output queue
datalink

Protocol Usage by Common Internet Applications

Application	IP	ICMP	UDP	TCP	SCTP
ping		•			
traceroute		•	•		
OSPF (routing protocol)	•				
RIP (routing protocol)			•		
BGP (routing protocol)				•	
BOOTP (bootstrap protocol)			•		
DHCP (bootstrap protocol)			•		
NTP (time protocol)			•		
TFTP			•		
SNMP (network management)			•		
SMTP (electronic mail)				•	
Telnet (remote login)				•	
SSH (secure remote login)				•	
FTP				•	
HTTP (the Web)				•	
NNTP (network news)				•	
LPR (remote printing)				•	
DNS			•	•	
NFS (network filesystem)			•	•	
Sun RPC			•	•	
DCE RPC			•	•	
IUA (ISDN over IP)					•
M2UA,M3UA (SS7 telephony signaling)					•
H.248 (media gateway control)			•	•	•
H.323 (IP telephony)			•	•	•
SIP (IP telephony)			•	•	•

SOCKETS API

Socket Address Structures

- IPv4 Socket Address Structure: sockaddr_in (including <netinet/in.h>)

```
struct in_addr {
  in_addr_t    s_addr;              /* 32-bit IPv4 address */
                                    /* network byte ordered */
};

struct sockaddr_in {
  uint8_t          sin_len;         /* length of structure (16) */
  sa_family_t      sin_family;      /* AF_INET */
  in_port_t        sin_port;        /* 16-bit TCP or UDP port number */
                                    /* network byte ordered */
  struct in_addr   sin_addr;        /* 32-bit IPv4 address */
                                    /* network byte ordered */
  char             sin_zero[8];     /* unused */
};
```

Datatypes required by the POSIX specification

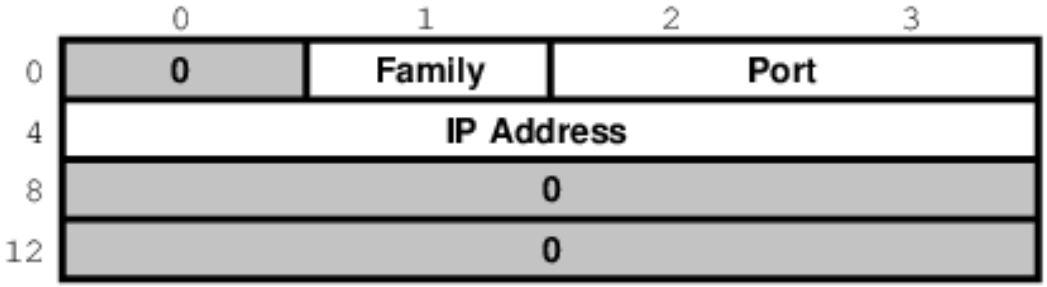
Datatype	Description	Header
int8_t	Signed 8-bit integer	<sys/types.h>
uint8_t	Unsigned 8-bit integer	<sys/types.h>
int16_t	Signed 16-bit integer	<sys/types.h>
uint16_t	Unsigned 16-bit integer	<sys/types.h>
int32_t	Signed 32-bit integer	<sys/types.h>
uint32_t	Unsigned 32-bit integer	<sys/types.h>
sa_family_t	Address family of socket address structure	<sys/socket.h>
socklen_t	Length of socket address structure, normally uint32_t	<sys/socket.h>
in_addr_t	IPv4 address, normally uint32_t	<netinet/in.h>
in_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

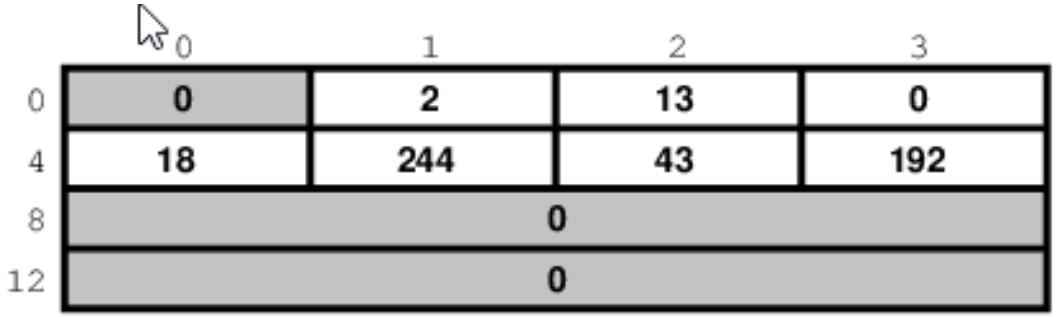
Address families in sys/socket.h

```
/*
 * Address families.
 */
#define AF_UNSPEC       0               /* unspecified */
#define AF_LOCAL        1               /* local to host (pipes, portals) */
#define AF_UNIX         AF_LOCAL        /* backward compatibility */
#define AF_INET         2               /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK      3               /* arpanet imp addresses */
#define AF_PUP          4               /* pup protocols: e.g. BSP */
#define AF_CHAOS        5               /* mit CHAOS protocols */
#define AF_NS           6               /* XEROX NS protocols */
#define AF_ISO          7               /* ISO protocols */
#define AF_OSI          AF_ISO
#define AF_ECMA         8               /* European computer manufacturers */
```

Example

```
sa.sin_family      = AF_INET;
sa.sin_port        = 13;
sa.sin_addr.s_addr = (((((192 << 8) | 43) << 8) | 244) << 8) | 18;
```





Value-Result Arguments

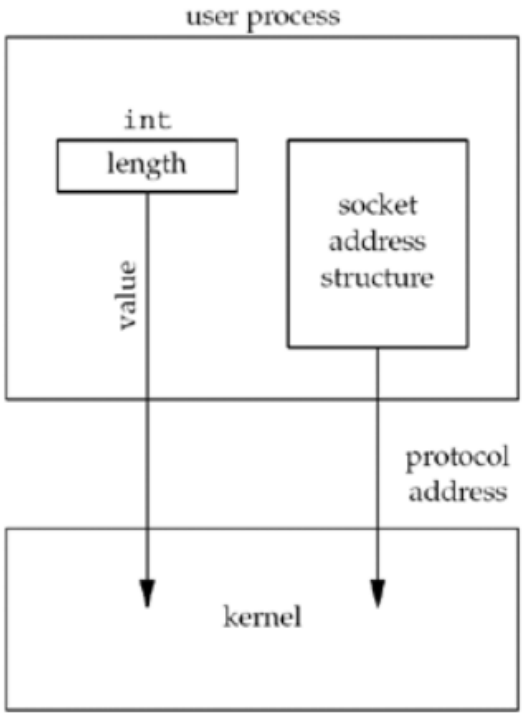
- when a socket address structure is passed to any socket function, it is always passed by reference.
- The length of the structure is also passed as an argument.
- 2 directions: process → kernel or kernel → process

Value-Result Arguments

- 3 functions: bind, connect & sendto: process → kernel

```
struct sockaddr_in serv;

/* fill in serv{} */
connect (sockfd, (SA *) &serv, sizeof(serv));
```

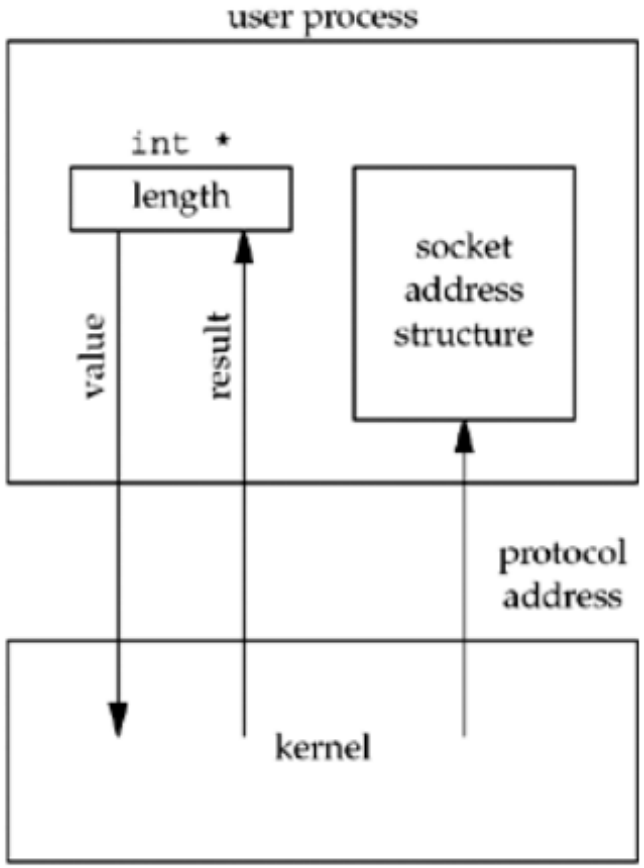


Value-Result Arguments

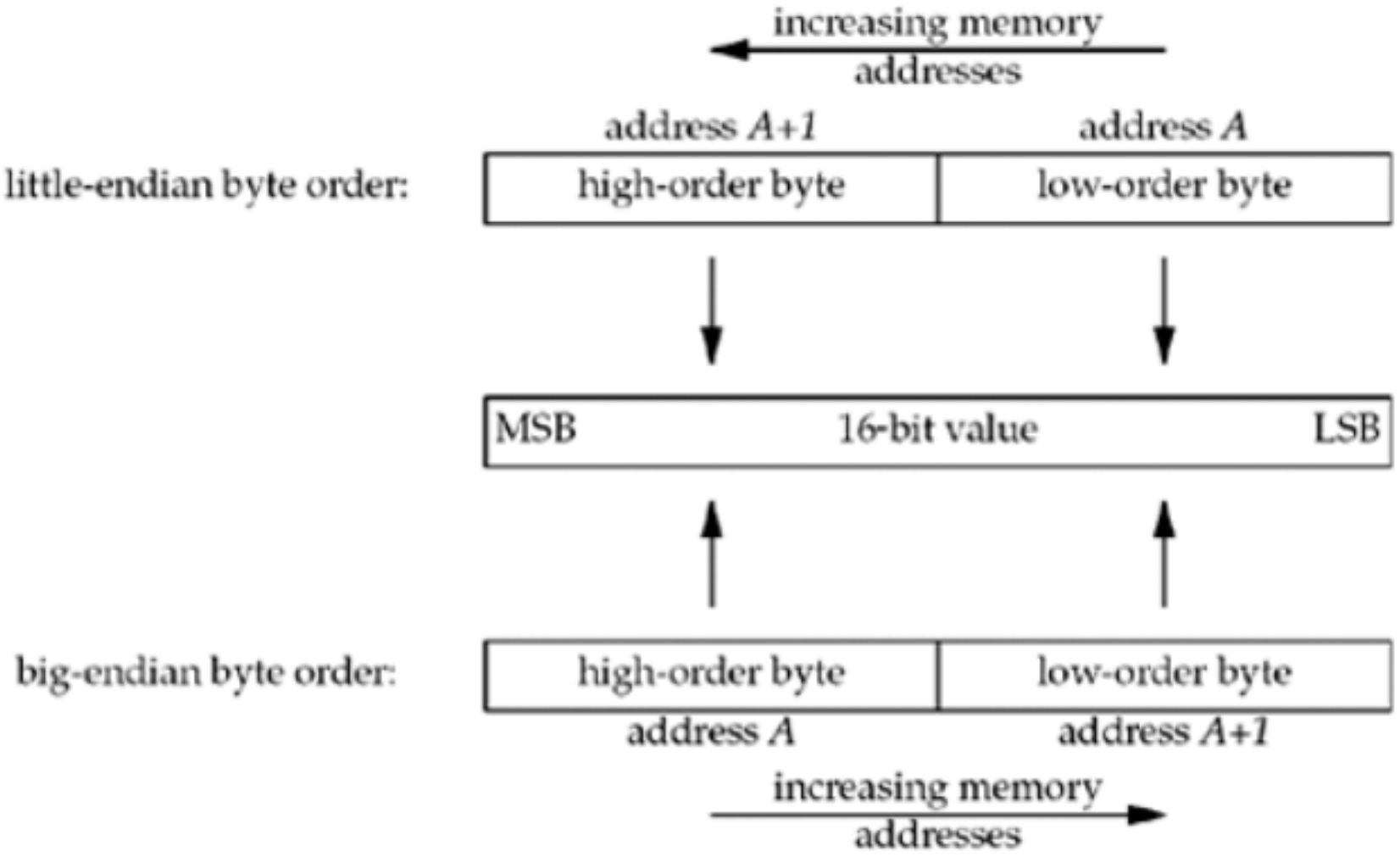
- 4 functions: accept, recvfrom, getsockname, and getpeername: kernel → process

```
struct sockaddr_un  cli;    /* Unix domain */
socklen_t  len;

len = sizeof(cli);          /* len is a value */
getpeername(unixfd, (SA *) &cli, &len);
/* len may have changed */
```



Byte Ordering Functions



intro/byteorder.c

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     union {
 6         short   s;
 7         char    c[sizeof(short)];
 8     } un;

 9     un.s = 0x0102;
10     printf("%s: ", CPU_VENDOR_OS);
11     if (sizeof(short) == 2) {
12         if (un.c[0] == 1 && un.c[1] == 2)
13             printf("big-endian\n");
14         else if (un.c[0] == 2 && un.c[1] == 1)
15             printf("little-endian\n");
16         else
17             printf("unknown\n");
18     } else
19         printf("sizeof(short) = %d\n", sizeof(short));

20     exit(0);
21 }
```

Address Conversion Functions

- They convert Internet addresses between **ASCII strings** (what humans prefer to use) and **network byte ordered binary values** (values that are stored in socket address structures).

```
#include <arpa/inet.h>
int inet_aton(const char *strptr, struct
in_addr *addrptr);
```

Returns: 1 if string was valid, 0 on error

```
in_addr_t inet_addr(const char *strptr);
```

Returns: 32-bit binary network byte ordered IPv4 address; INADDR_NONE if error

```
char *inet_ntoa(struct in_addr inaddr);
```

Returns: pointer to dotted-decimal string

Address Conversion Functions (2)

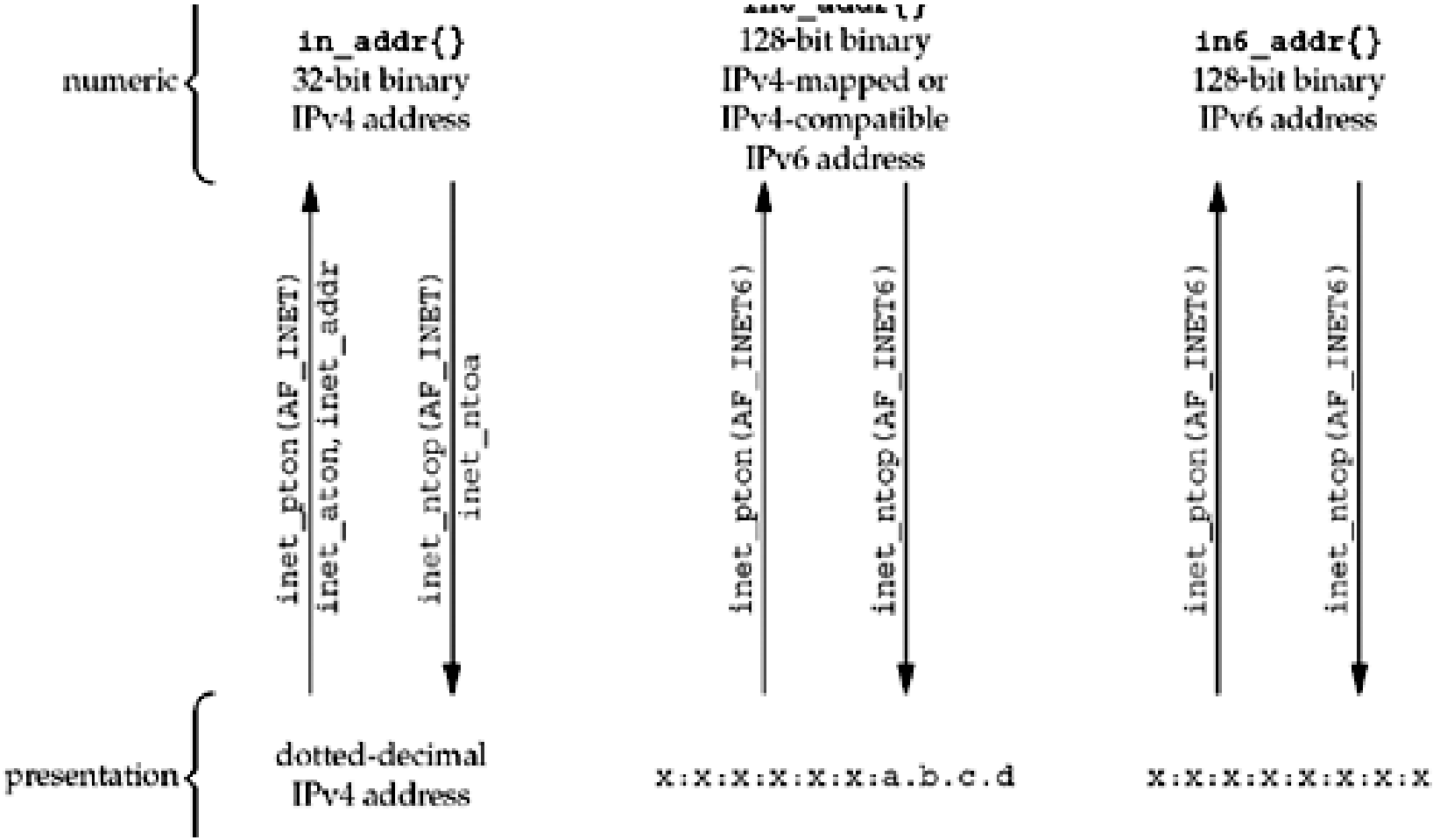
```
#include <arpa/inet.h>
int inet_pton(int family, const char
*strptr, void *addrptr);
```

Returns: 1 if OK, 0 if input not a valid presentation format, -1 on error

```
const char *inet_ntop(int family, const void
*addrptr, char *strptr, size_t len);
```

Returns: pointer to result if OK, NULL on error

Summary of address conversion functions



Example

```
foo.sin_addr.s_addr = inet_addr(cp);
```

```
inet_pton(AF_INET, cp, &foo.sin_addr);
```

```
ptr = inet_ntoa(foo.sin_addr);
```

```
char str[INET_ADDRSTRLEN];
```

```
ptr = inet_ntop(AF_INET,
&foo.sin_addr, str, sizeof(str));
```

Example

libfree/inet_pton_ipv4.c

```
10 int
11 inet_pton(int family, const char *strptr, void *addrptr)
12 {
13     if (family == AF_INET) {
14         struct in_addr in_val;

15         if (inet_aton(strptr, &in_val)) {
16             memcpy(addrptr, &in_val, sizeof(struct in_addr));
17             return (1);
18         }
19         return (0);
20     }
21     errno = EAFNOSUPPORT;
22     return (-1);
23 }
```

sock_ntop function

- Problem of `inet_ntop`: it requires the caller to pass a pointer to a binary address.

```
struct sockaddr_in addr;
inet_ntop(AF_INET,&addr.sin_addr,
  str, sizeof(str));
```

- function `sock_ntop`

```
#include "unp.h"
char *sock_ntop(const struct sockaddr
  *sockaddr, socklen_t addrlen);
```

`readn`, `writen`, and `readline` functions

```
#include "unp.h"
ssize_t readn(int filedes, void *buff,
  size_t nbytes);
ssize_t writen(int filedes, const void
  *buff, size_t nbytes);
ssize_t readline(int filedes, void *buff,
  size_t maxlen);
```

- All return: number of bytes read or written, –1 on error

`readn` function: Read *n* bytes from a descriptor.

lib/readn.c

```
 1 #include     "unp.h"

 2 ssize_t                           /* Read "n" bytes from a descriptor. */
 3 readn(int fd, void *vptr, size_t n)
 4 {
 5     size_t  nleft;
 6     ssize_t nread;
 7     char   *ptr;

 8     ptr = vptr;
 9     nleft = n;
10     while (nleft > 0) {
11         if ( (nread = read(fd, ptr, nleft)) < 0) {
12             if (errno == EINTR)
13                 nread = 0;      /* and call read() again */
14             else
15                 return (-1);
16         } else if (nread == 0)
17             break;              /* EOF */

18         nleft -= nread;
19         ptr += nread;
20     }
21     return (n - nleft);         /* return >= 0 */
22 }
```

`writen` function: Write n bytes to a descriptor.

lib/writen.c

```
 1 #include    "unp.h"

 2 ssize_t                           /* Write "n" bytes to a descriptor. */
 3 writen(int fd, const void *vptr, size_t n)
 4 {
 5     size_t nleft;
 6     ssize_t nwritten;
 7     const char *ptr;

 8     ptr = vptr;
 9     nleft = n;
10     while (nleft > 0) {
11         if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
12             if (nwritten < 0 && errno == EINTR)
13                 nwritten = 0;   /* and call write() again */
14             else
15                 return (-1);    /* error */
16          }

17          nleft -= nwritten;
18          ptr += nwritten;
19     }
20     return (n);
21 }
```

readline function: Read a text line from a descriptor, one byte at a time.

test/readline1.c

```
 1 #include     "unp.h"

 2 /* PAINFULLY SLOW VERSION -- example only */
 3 ssize_t
 4 readline(int fd, void *vptr, size_t maxlen)
 5 {
 6     ssize_t n, rc;
 7     char    c, *ptr;

 8     ptr = vptr;
 9     for (n = 1; n < maxlen; n++) {
10       again:
11         if ( (rc = read(fd, &c, 1)) == 1) {
12             *ptr++ = c;
13             if (c == '\n')
14                 break;           /* newline is stored, like fgets() */
15         } else if (rc == 0) {
16             *ptr = 0;
17             return (n - 1);      /* EOF, n - 1 bytes were read */
18         } else {
19             if (errno == EINTR)
20                 goto again;
21             return (-1);         /* error, errno set by read() */
22         }
23     }

24     *ptr = 0;                    /* null terminate like fgets() */
25     return (n);
26 }
```