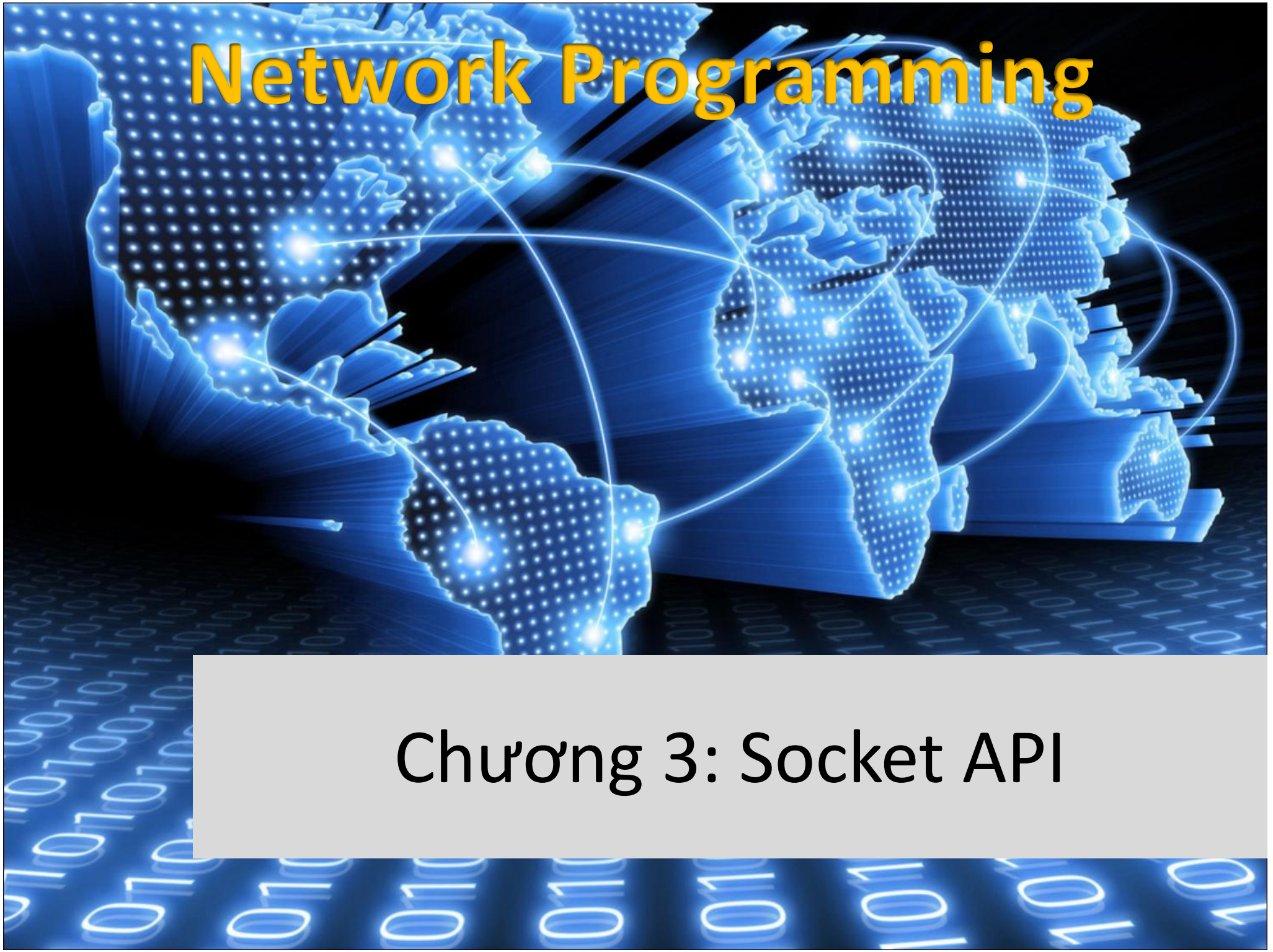


Chương 3: Socket API

Nội dung

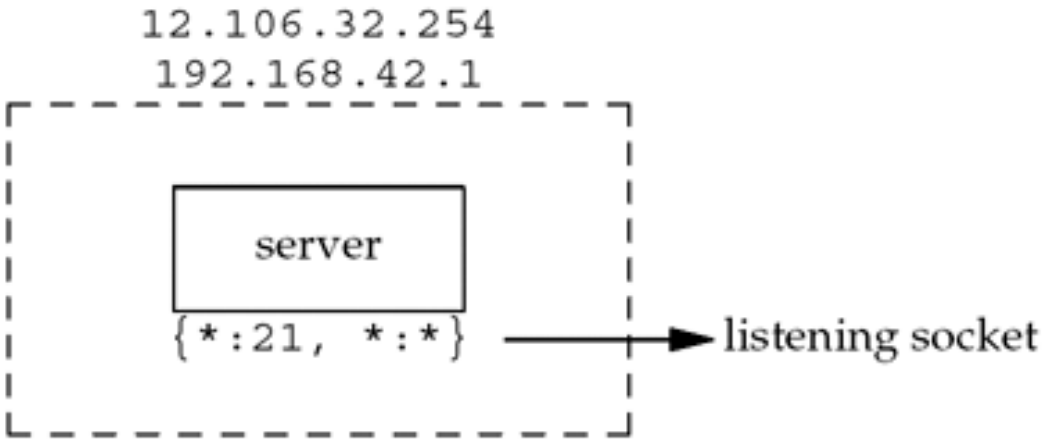
- Basics of socket
- Sockets API

BASICS OF SOCKET

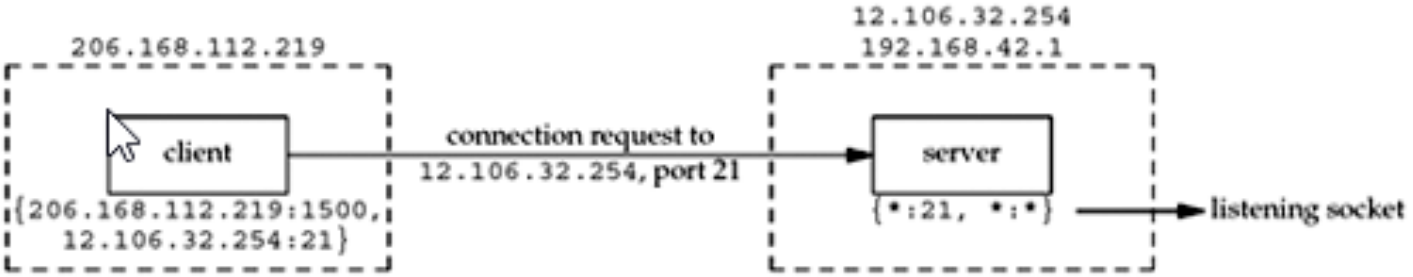
Socket Pair

- 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*.

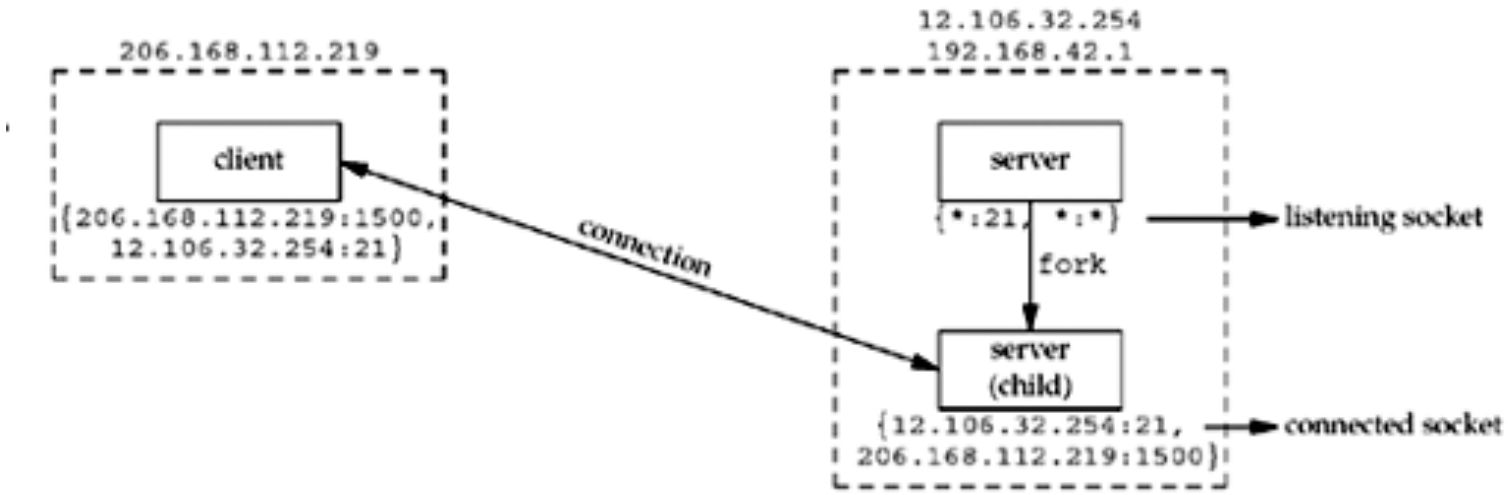
TCP Port Numbers and Concurrent Servers (1)



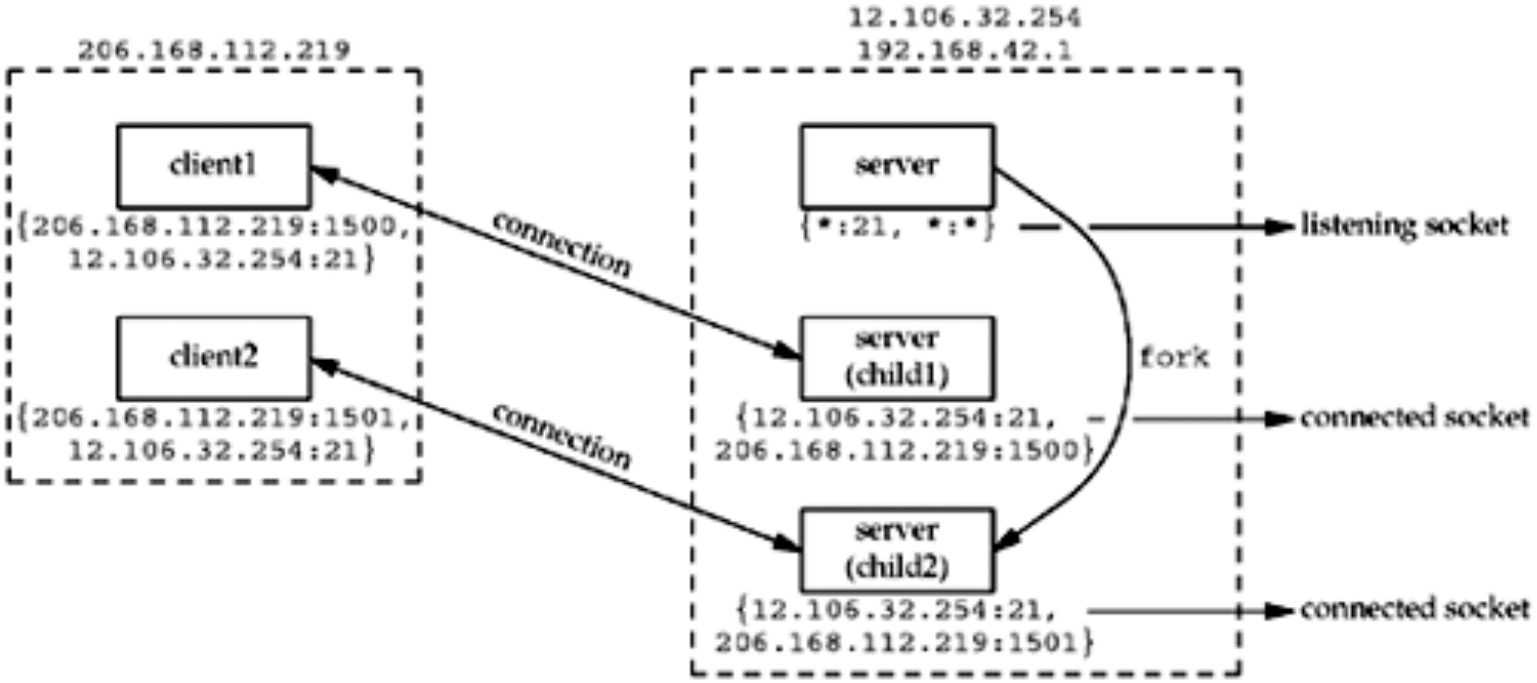
TCP Port Numbers and Concurrent Servers (2)



TCP Port Numbers and Concurrent Servers (3)



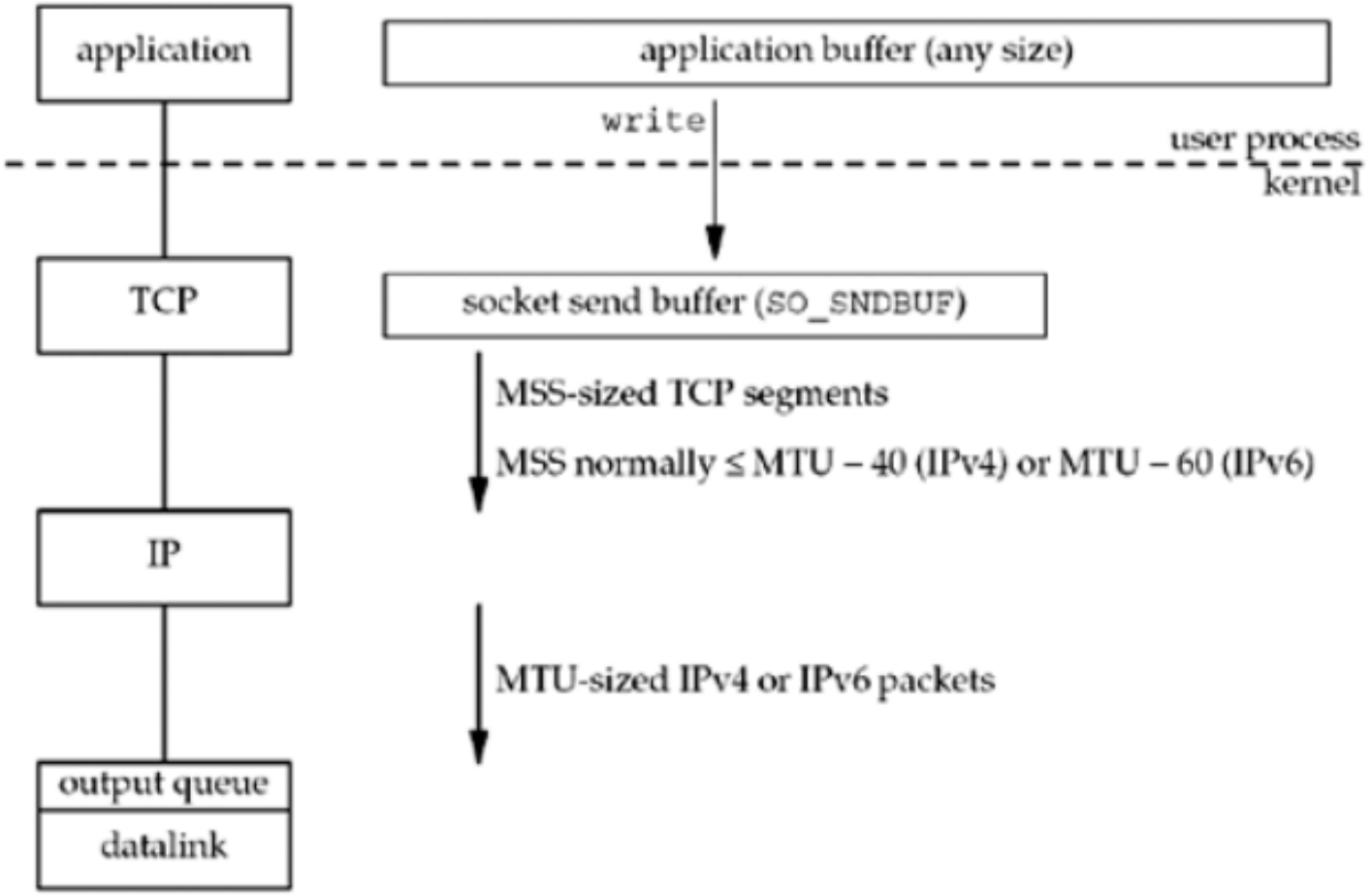
TCP Port Numbers and Concurrent Servers (4)



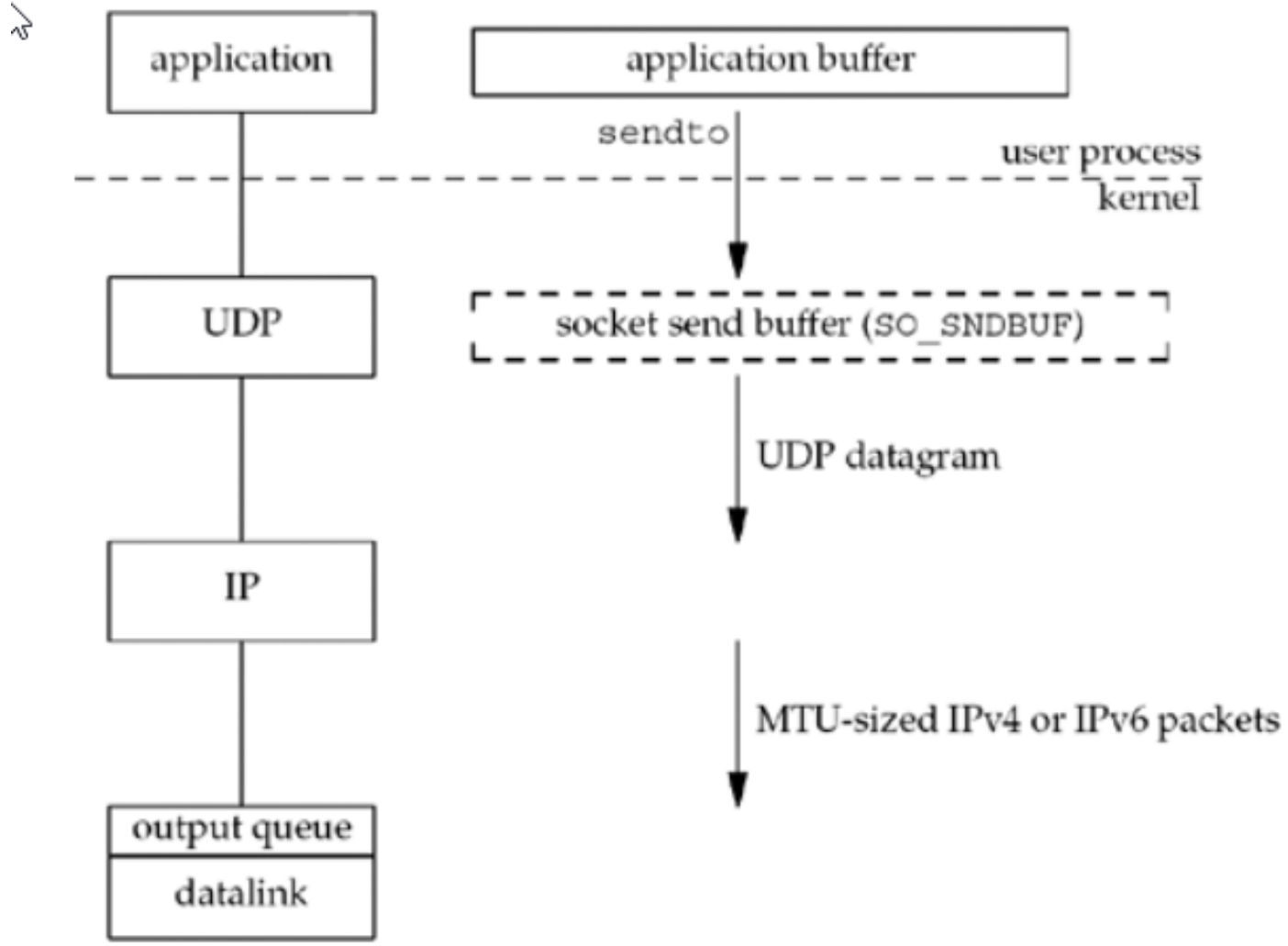
Buffer Sizes and Limitations

- 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 */
};
```

Chương trình ví dụ: **init_sockaddr_in.c**

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 */
```

Value-Result Arguments

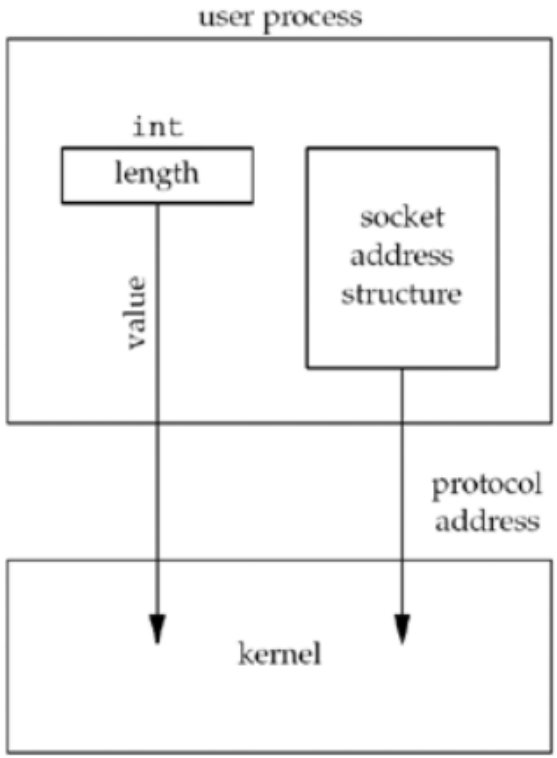
- when a socket address structure is passed to any socket function, it is always passed by reference.
- The length of the structure is also passed as an argument.
- 2 directions: **process → kernel** or **kernel → process**

Value-Result Arguments

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

```
struct sockaddr_in serv;

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



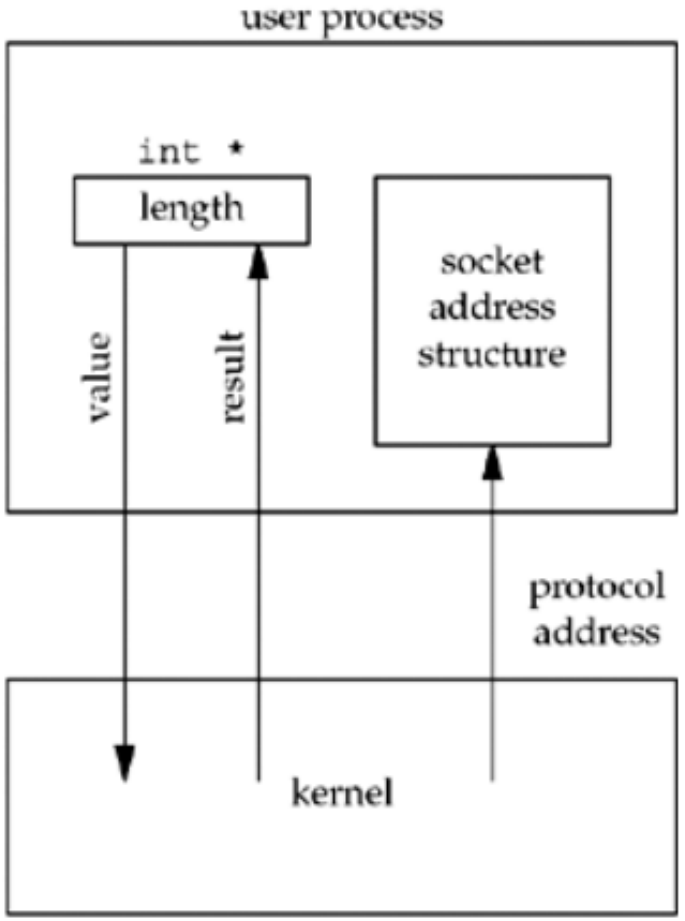
Chương trình ví dụ: **ex_connect.c**

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 */
```



Chương trình ví dụ: **ex_getpeername.c**

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

Chương trình ví dụ: **convert_IP.c**

Address Conversion Functions (2)

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

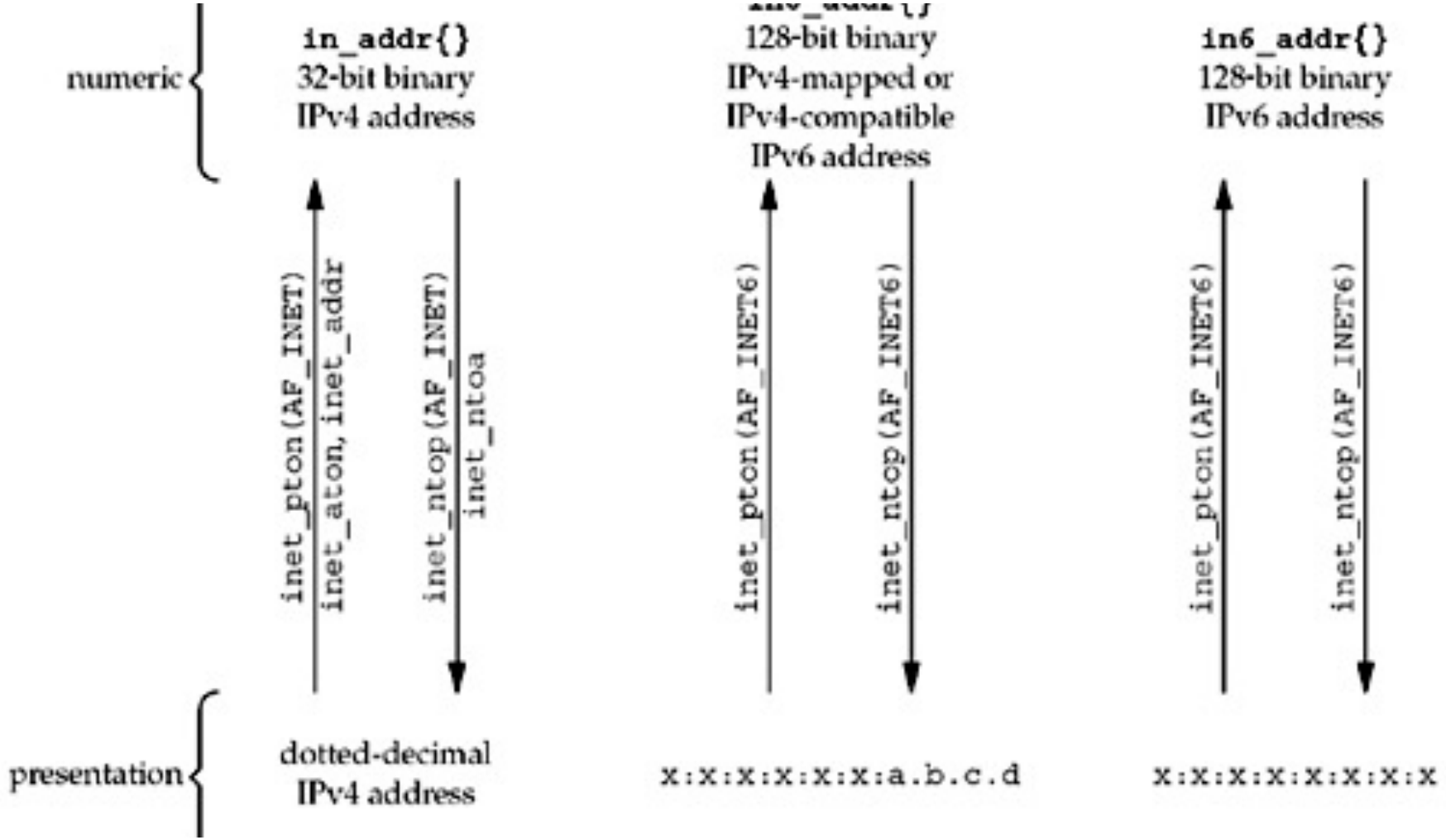
Returns: 1 if OK, 0 if input not a valid presentation format, -1 on error

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

Returns: pointer to result if OK, NULL on error

Chương trình ví dụ: **convert_IP_v4-6.c**

Summary of address conversion functions



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);
```

Chương trình ví dụ: **example_of_sock_ntop.c**

Compare `inet_ntop` and `sock_ntop`

Aspect	`inet_ntop`	`sock_ntop`
Input	IP address in binary form (`in_addr`, `in6_addr`).	Socket address (`struct sockaddr`).
Output	IP address in string format.	Both IP address and port in string format.
Scope	Focuses on IP address only (IPv4 or IPv6).	Can handle both IP address and port (more general).
Transport Layer	Does not handle port numbers or protocol information.	Includes port numbers and supports transport layers.
Flexibility	Requires the user to separately manage socket info.	Handles entire socket info in one step.

`read()`

- **read() Function:**
- The read() function is used to read data from a file descriptor, which can include socket file descriptors. For socket streams, read() reads data sent by a peer over the network.
- `ssize_t read(int sockfd, void *buf, size_t count);`
- **Parameters:**
  - **sockfd**: The file descriptor of the socket (created using socket()).
  - **buf**: A pointer to a buffer where the data read from the socket will be stored.
  - **count**: The maximum number of bytes to read (size of the buffer).

`read()`

- read() function returns:
  - On success, the number of bytes actually read is returned (this can be less than count if fewer bytes are available).
  - On error, -1 is returned, and errno is set appropriately.
  - If the peer has closed the connection, 0 is returned (end-of-file).

read()

- Example:

```
char buffer[1024];
int n = read(sockfd, buffer, sizeof(buffer));
if (n > 0) {
    printf("Received message: %s\n", buffer);
} else if (n == 0) {
    printf("Connection closed by peer\n");
} else {
    perror("read error");
}
```

`write()`

- **write() Function:**
- The write() function is used to send data to a file descriptor, which can include socket file descriptors. For socket streams, write() sends data to the connected peer.

- `ssize_t write(int sockfd, const void *buf, size_t count);`

`write()`

- **Parameters:**
  - **sockfd**: The file descriptor of the socket (created using socket()).
  - **buf**: A pointer to the buffer containing the data to be sent.
  - **count**: The number of bytes to write (send) from the buffer.
- **Return Value:**
  - On success, the number of bytes actually written is returned.
  - On error, -1 is returned, and errno is set appropriately.

write()

- Example:

```
char *message = "Hello, server!";
int n = write(sockfd, message,
strlen(message));
if (n > 0) {
    printf("Message sent: %s\n",
message);
} else {
    perror("write error");
}
```

Chương trình ví dụ: **read-write-server.c**
read-write-client.c