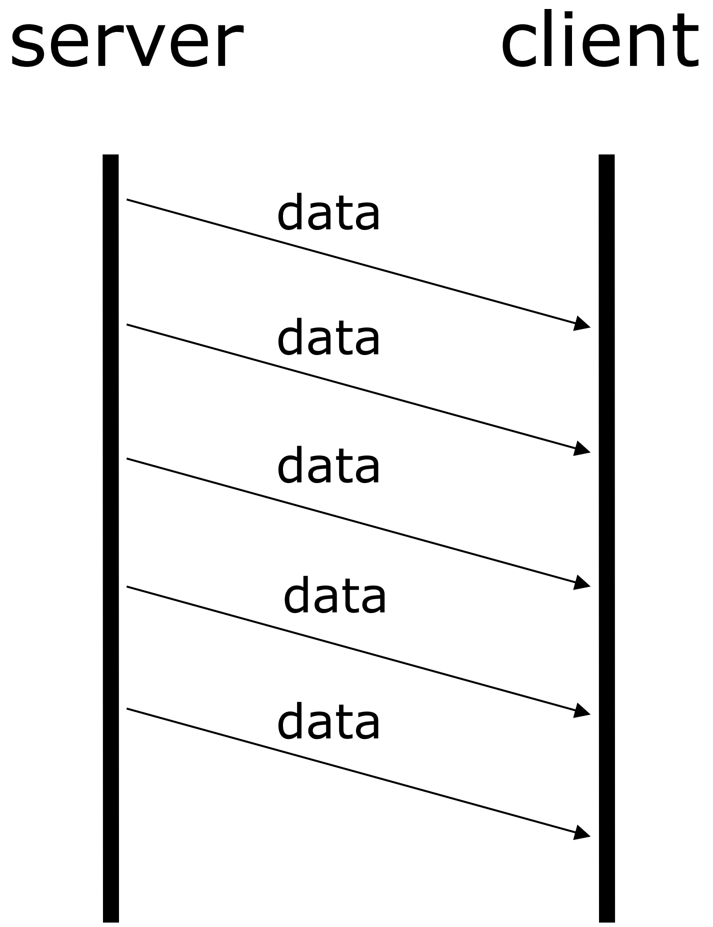
# UDP CLIENT AND SERVER

## Content

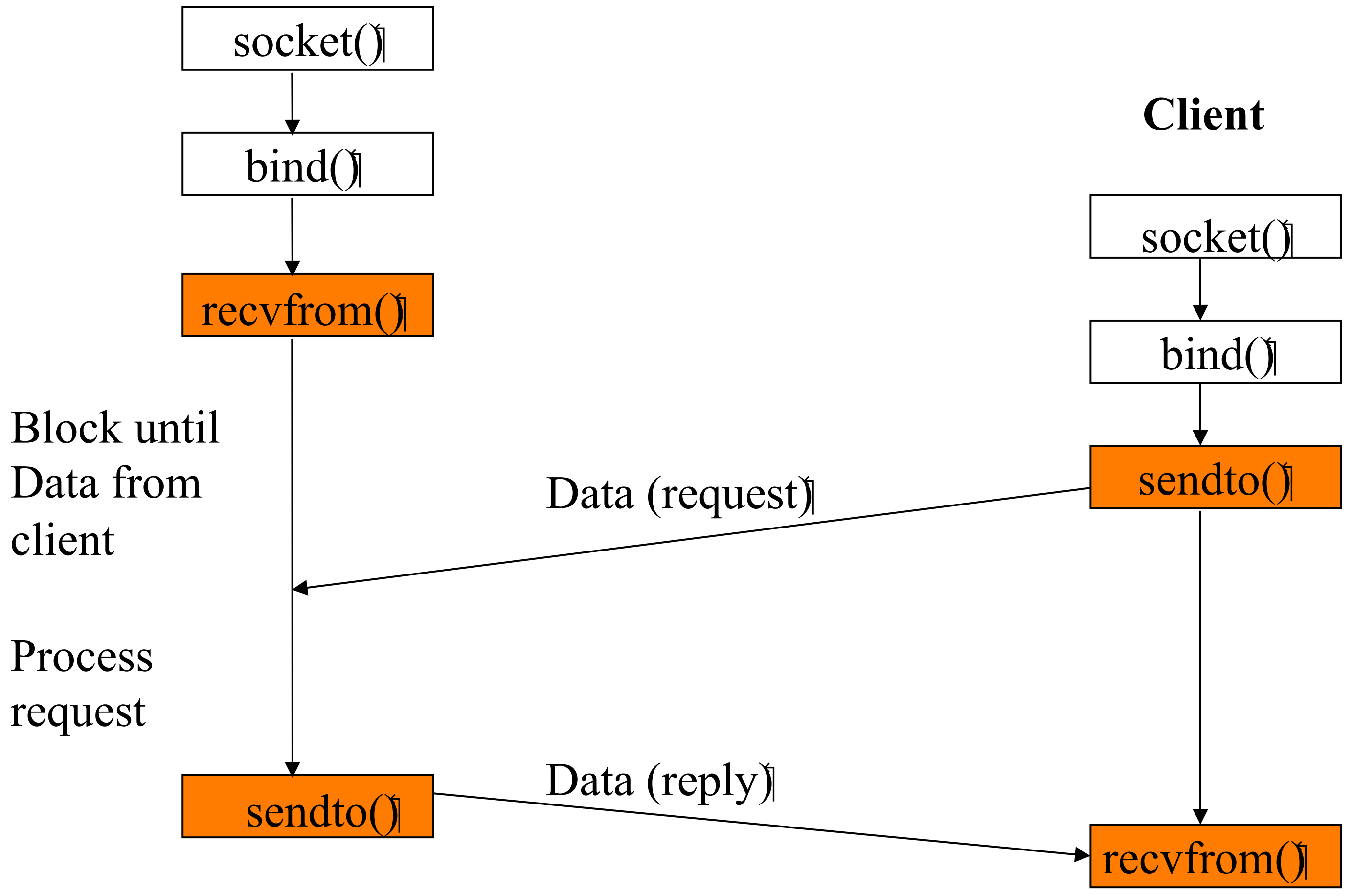
- What is UDP ?
- UDP Socket APIs
- Sample and Exercise

# UDP (User Datagram Protocol)

- Not reliable
- Familiar example of application using UDP
  - DNS
  - SNMP
  - Video streaming



## UDP


socket()
bind()
recvfrom()
Block until
Data from
client
Process
request
sendto()
Client
socket()
bind()
sendto()
Data (request)
Data (reply)
recvfrom()

# recvfrom()

- Received data from a socket

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct
                                        sockaddr *from, socklen_t *fromlen);
```

- Parameters :
  - s : a socket we use to receive data
  - buf : a buffer uses to receive data
  - len : the length of buf
  - flag : how to control recvfrom function work
  - from : an address structure that will tell you where the data came from → need to use *sockaddr_in* or *sockaddr_in6* type and cast it
  - fromlen : the size of struct sockaddr.

## recvfrom()

- Return value
  - the number of bytes actually received (less than len)
  - -1 on error
- Differences between recv() and recvfrom()
  - Recv() : do not need address parameter (because two host have connected already)
  - Recvfrom() : need address parameter – no need connection, no reliable

## sendto()

- Send data to a socket

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t sendto(int s, const void *buf, size_t len, int flags, const
                              struct sockaddr *to, socklen_t tolen);
```

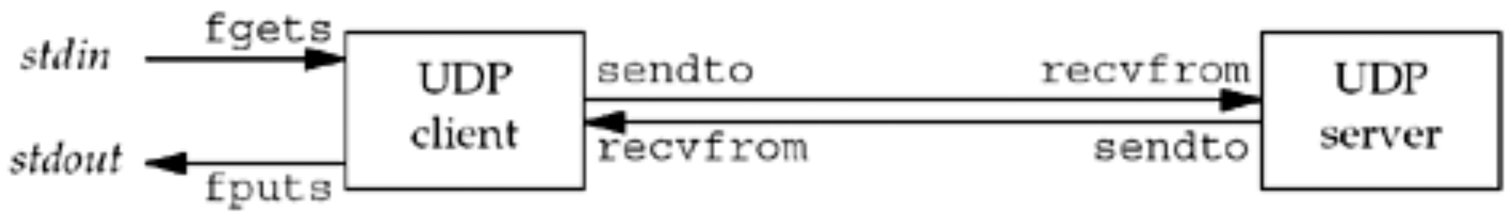
- Parameters :
  - s : a socket we use to send data
  - buf : a buffer contains data to send
  - len : the length of buf
  - flag : how to control sendto function work
  - to : a address structure where you specify the remote socket to send data to
  - tolen : the size of struct sockaddr.

## sendto()

- Return value :
  - the number of bytes actually sent
  - -1 on error
- Note :
  - Number of bytes sent must be less than len parameters

## Example

- A simple UDP client and server
  - Server receives data from client
  - Server sends back data to client
  - It present in udpserv01.c and dg_echo.c

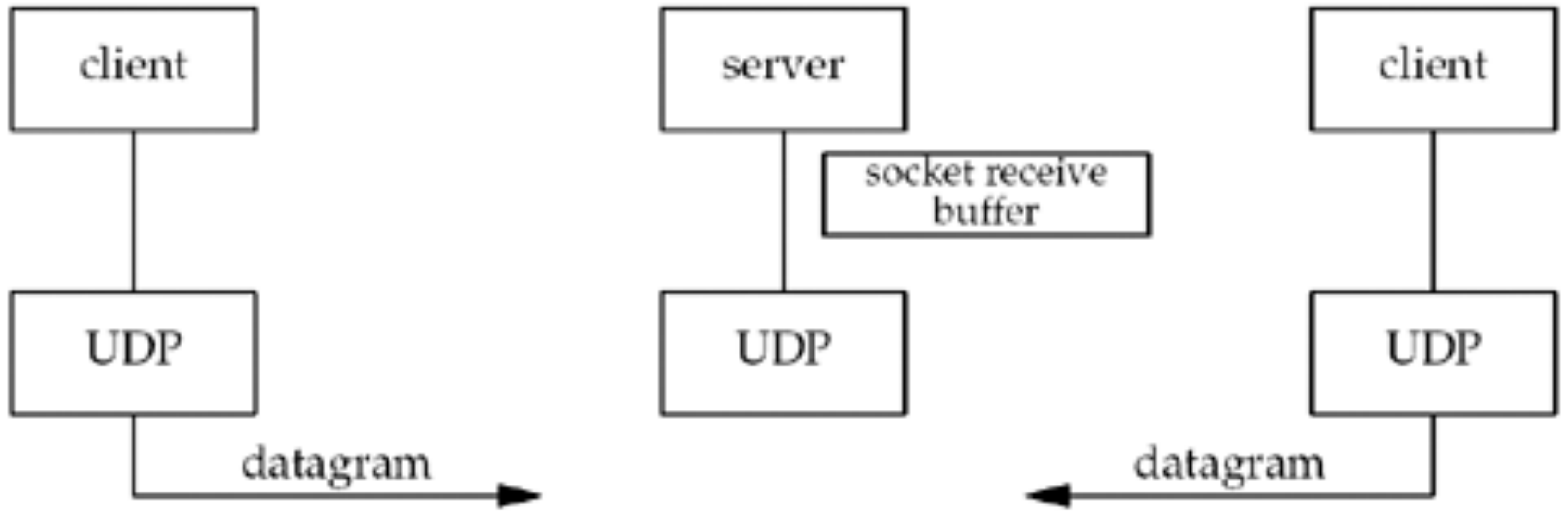


## Example – UDP Server

```
int sockfd, n;
socklen_t len;
char mesg[MAXLINE];
struct sockaddr_in servaddr, cliaddr;

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
for ( ; ; ) {
     len = sizeof(cliaddr);
     n = recvfrom(sockfd, mesg, MAXLINE, 0, (struct sockaddr *)
                                          &cliaddr, &len);
     sendto(sockfd, mesg, n, 0, (struct sockaddr *) &cliaddr,
                                                  len);
}
```

## Two clients connect to a server



## Example - updClient.c

```
int sockfd, n;
struct sockaddr_in servaddr;
char sendline[MAXLINE], recvline[MAXLINE + 1];
….
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(SERV_PORT);
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
while (fgets(sendline, MAXLINE, stdin) != NULL) {
   sendto(sockfd, sendline, strlen(sendline), 0,
       (struct sockaddr *) &servaddr, sizeof(servaddr));

   n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
   recvline[n] = 0; /* null terminate */
   printf("%s", recvline);
}
..
```

## Exercise

1) Test the existing UDP server and client.

- Use the existing UDP server, write your client
- Use the existing UDP client, write your server.

2) Revise server and client so that they print the IP address of the other (sender) each time they receive a message from the sender.

- Syntax: Receive from: IP address : message
- Ex:
  - Receive from: 192.168.0.1: Hello
  - Receive from: 192.168.0.1: world
- 3) Test 1 server with 2 clients

## Exercise

- Revise client và server so that
  - One server will work with 2 clients
  - Whenever the server receives a message from one client, it sends the message to the other client

## connect() with UDP

- If server isn't running, the client blocks forever in the call to recvfrom → asynchronous error
- Use connect() for a UDP socket
  - But it's different from calling connect() on a TCP socket
  - Calling connect() on a UDP socket doesn't create a connection
  - The kernel just checks for any immediate errors and returns immediately to the calling process

## connected UDP socket vs unconnected UDP socket

- We do not use sendto(), but write() or send() instead
- We do not need to use recvfrom() to learn the sender of a datagram, but read(), recv() instead
- Asynchronous errors are returned to the process for connected UDP sockets

## Example

```
int n;
char sendline[MAXLINE], recvline[MAXLINE + 1];
struct sockaddr_in servaddr;
connect(sockfd, (struct sockaddr *) &servaddr,
                                              servlen);
while (fgets(sendline, MAXLINE, fp) != NULL) {
    send(sockfd, sendline, strlen(sendline));
    n = recv(sockfd, recvline, MAXLINE);
    recvline[n] = 0;        /* null terminate */
    printf("%s", recvline);
}
```

## Broadcasting

- Broadcast address: All of bits in hostID are 1
  - Local: 255.255.255.255
- Broadcasting: send data with destination address as a broadcast address
- Only broadcast on UDP
- SocketAPI: SO_BROADCAST Socket Option

## Example

```
int      n;
const int on = 1;
char     sendline[MAXLINE], recvline[MAXLINE + 1];
socklen_t servlen;
struct sockaddr *preply_addr;
preply_addr = malloc(servlen);
setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &on,

    sizeof(on));
sendto(sockfd, sendline, strlen(sendline), 0,
                 servaddr, servlen);
```

## Example(cont)

```
for ( ; ; ) {
    len = servlen;
    n = recvfrom(sockfd, recvline, MAXLINE, 0,
                                    preply_addr, &len);
    if (n < 0) {
        printf("recvfrom error");
    } else {
        recvline[n] = 0;
        printf("%s",recvline);
    }
}
```