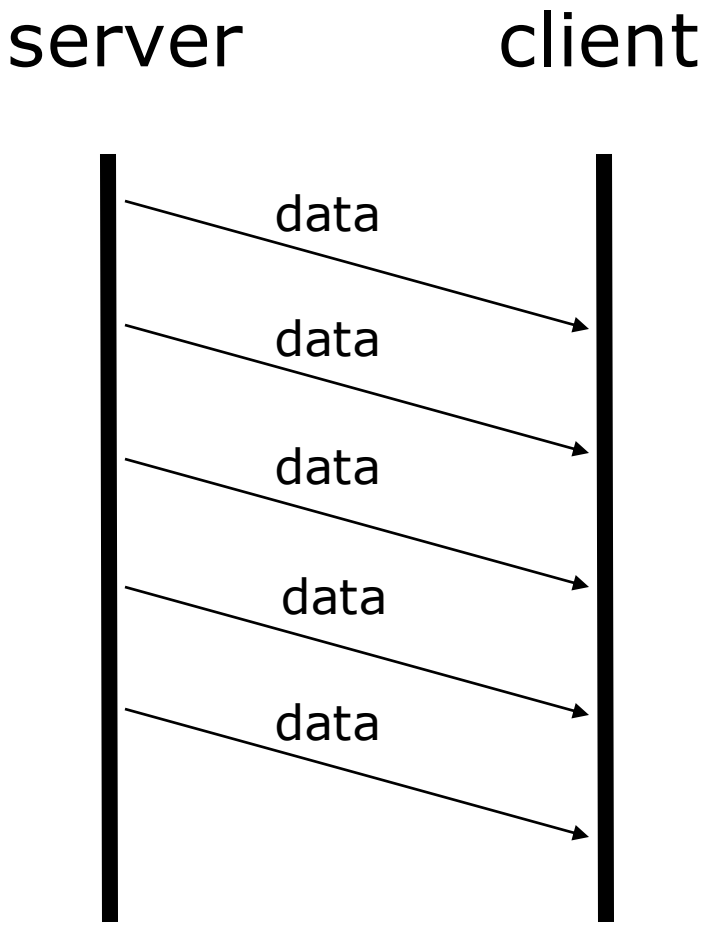
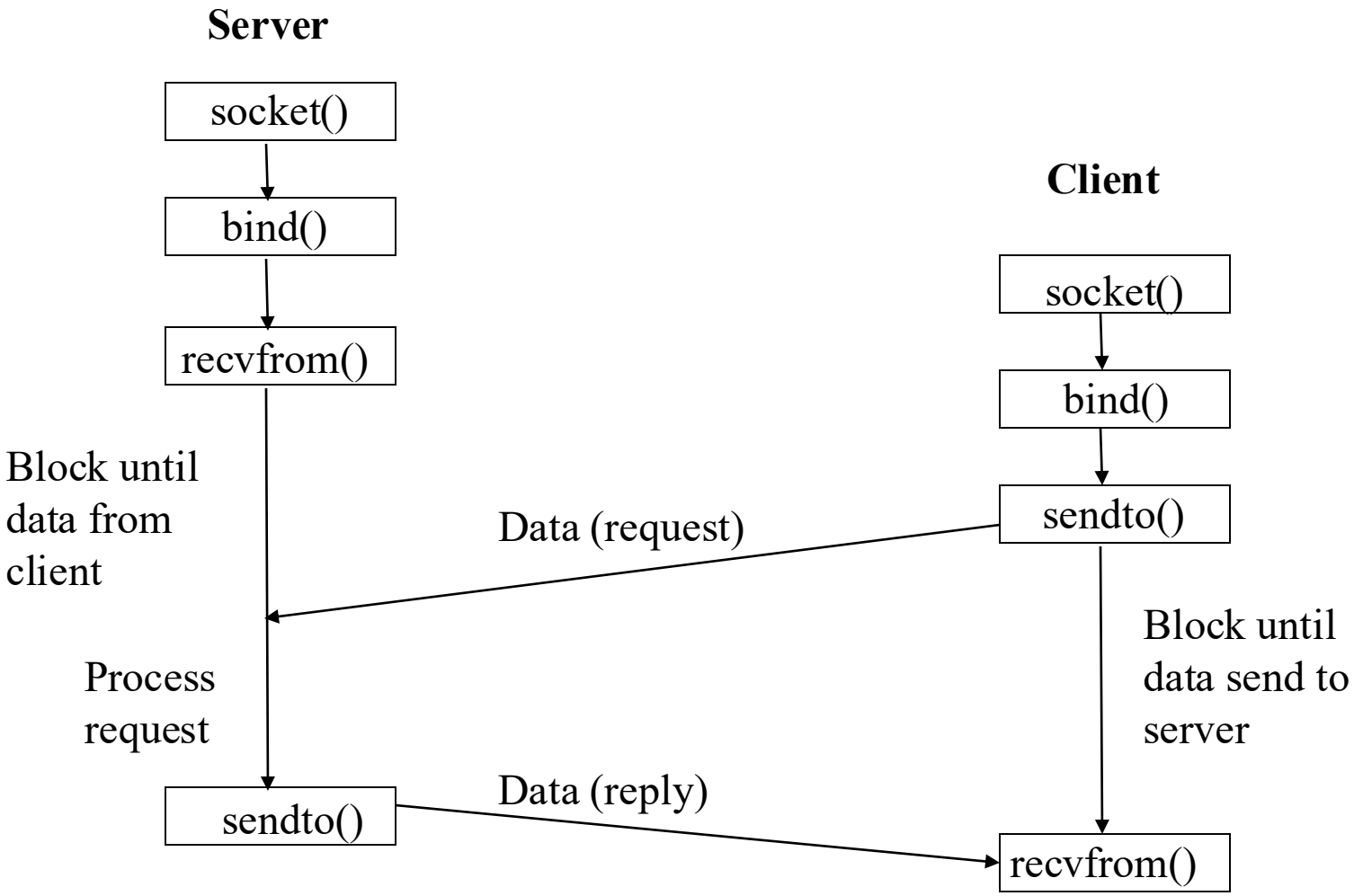
UDP SOCKET

UDP (User Datagram Protocol)

- No reliable
- No flow control
- Familiar example
  - DNS
  - Streaming
- Image
  - Postcard exchange



UDP client/server



recvfrom()

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

- Received data from a socket
- Parameters:
  - [IN] `sockfd`: the socket file descriptor
  - [OUT] `buf`: the buffer where the message should be stored
  - [IN] `len`: the size of the buffer
  - [IN] `flags`: how to control recvfrom function work
  - [OUT] `from`: the address of the sender
  - [OUT] `fromlen`: the size of sender's address
- Return:
  - Success: return the length of the received data in bytes. If the incoming message is too long to fit in the supplied buffer, the excess bytes shall be discarded.
  - Error: −1 and set **errno** to indicate the error.

recvfrom()

- 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

`recvfrom()` - Flags

- `MSG_PEEK`: Peeks at an incoming message. The data is treated as unread and the next *recvfrom*() or similar function shall still return this data.
- `MSG_OOB`: Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
- `MSG_WAITALL`: On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned, excepting:
  - the connection is terminated
  - MSG_PEEK was specified
  - an error is pending for the socket
  - a signal is caught
- Use bitwise OR operator (|) to combine more than one flag

sendto()

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

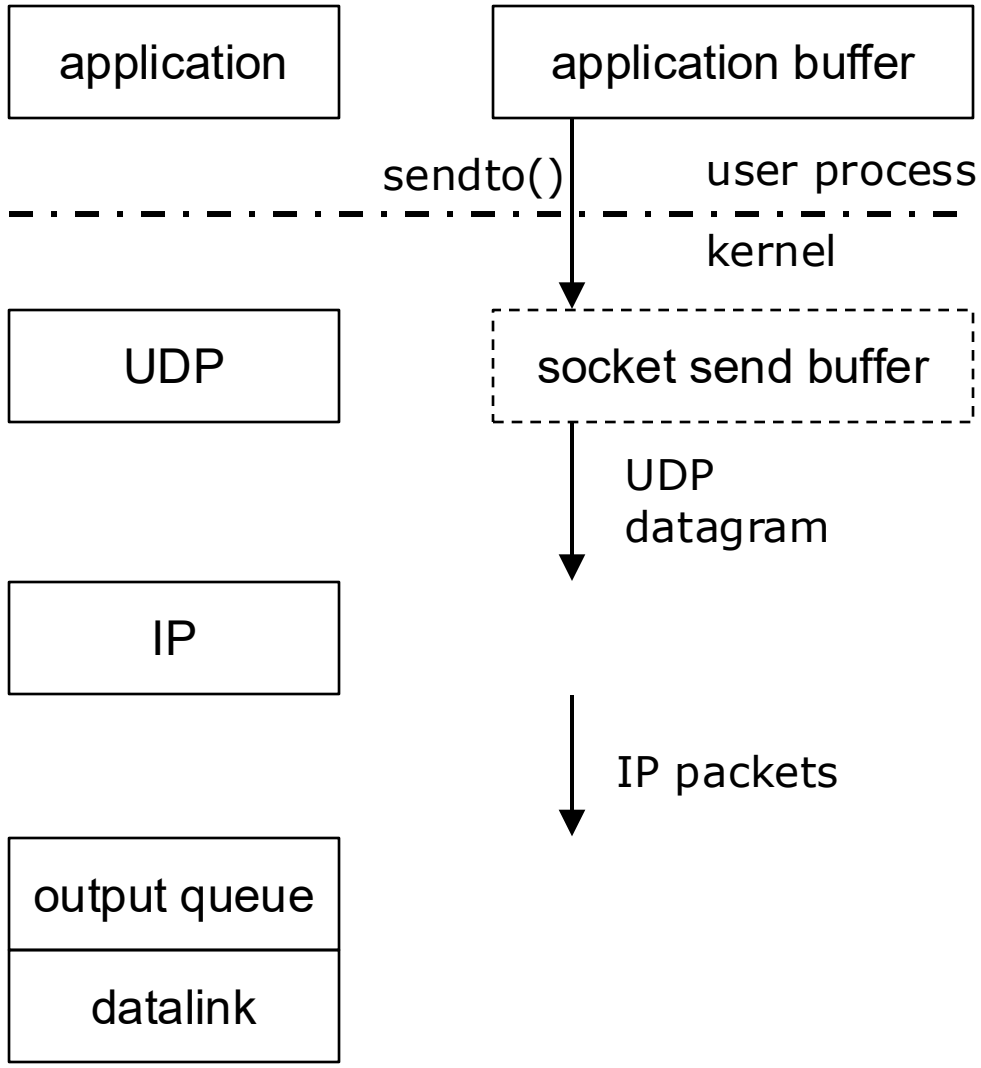
- Received data from a socket
- Parameters:
  - [IN] `sockfd`: the socket file descriptor
  - [IN] `buf`: points to a buffer containing the message to be sent
  - [IN] `len`: the size of the message
  - [IN] `flags`: how to control sendto function work
  - [IN] `to`: the address of the receiver
  - [IN] `tolen`: the length of the `sockaddr` structure pointed to by the `to` argument
- Return:
  - Success: shall return the length of the sent message in bytes
  - Error: −1 and set **errno** to indicate the error.

`sendto()` - Flags

- `MSG_OOB`: Sends out-of-band data on sockets that support out-of-band data.
- `MSG_DONTROUTE`: Don't use a gateway to send out the packet, only send to hosts on directly connected networks
- Use bitwise OR operator (|) to combine more than one flag

sendto()

- UDP socket buffer doesn't really exist
- UDP socket buffer has a send buffer size
- If an application writes a datagram larger than the socket send buffer size, `EMSGSIZE` is returned



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 Echo Server

```
int sockfd, rcvBytes, sendBytes;
socklen_t len;
char buff[BUFF_SIZE+1];
struct sockaddr_in servaddr, cliaddr;

//Step 1: Construct socket
if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
   perror("Error: ");
   return 0;
}

//Step 2: Bind address to socket
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
if(bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))){
   perror("Error: ");
   return 0;
}

printf("Server started.");
```

Example – UDP Echo Server(cont)

```
 //Step 3: Communicate with client
for ( ; ; ) {
    len = sizeof(cliaddr);
    rcvBytes = recvfrom(sockfd, buff, BUFF_SIZE, 0,
                        (struct sockaddr *) &cliaddr, &len);
    if(rcvBytes < 0){
         perror("Error: ");
         return 0;
    }
    buff[recvBytes] = '\0';
    printf("[%s:%d]: %s", inet_ntoa(cliaddr.sin_addr),
                               ntohs(cliaddr.sin_port), buff);

    sendBytes = sendto(sockfd, buff, rcvBytes, 0,
                        (struct sockaddr *) &cliaddr, len);
    if(sendBytes < 0){
         perror("Error: ");
         return 0;
    }
}
```

Example – UDP Echo Client

```
int sockfd, rcvBytes, sendBytes;
socklen_t len;
char buff[BUFF_SIZE+1];
struct sockaddr_in servaddr;

//Step 1: Construct socket
if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
   perror("Error: ");
   return 0;
}

//Step 2: Define the address of the server
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr = inet_aton(SERV_ADDR, &servaddr.sin_addr);
servaddr.sin_port = htons(SERV_PORT);
```

Example – UDP Echo Client(cont)

```
//Step 3: Communicate with server
printf("Send to server: ");
gets_s(buff, BUFF_SIZE);

len = sizeof(servaddr);
sendBytes = sendto(sockfd, buff, strlen(buff), 0,
                        (struct sockaddr *) &seraddr, len);
if(sendBytes < 0){
   perror("Error: ");
   return 0;
}

rcvBytes = recvfrom(sockfd, buff, BUFF_SIZE, 0,
                        (struct sockaddr *) &seraddr, &len);
if(rcvBytes < 0){
   perror("Error: ");
   return 0;
}
buff[recvBytes] = '\0';
printf("Reply from server: %s", buff);
```

`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
- 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);
}
```