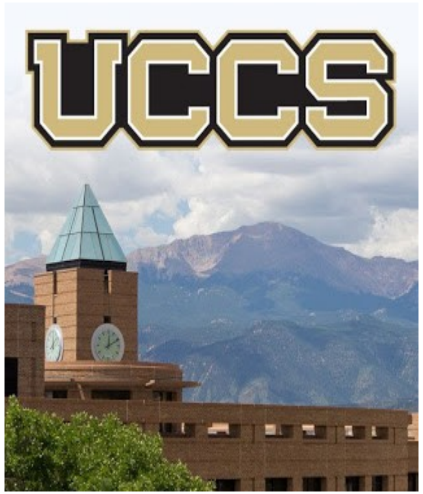
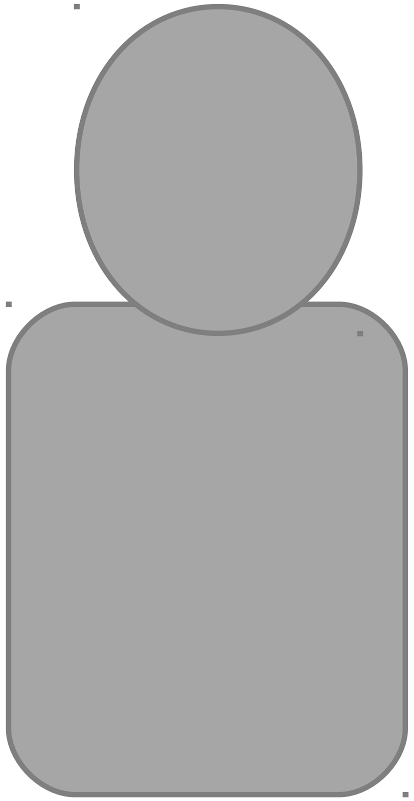


Unit 01.03.02 CS 5220: COMPUTER COMMUNICATIONS

Berkeley Socket API - II

XIAOBO ZHOU, Ph.D.

Professor, Department of Computer Science



Stream Mode of Service

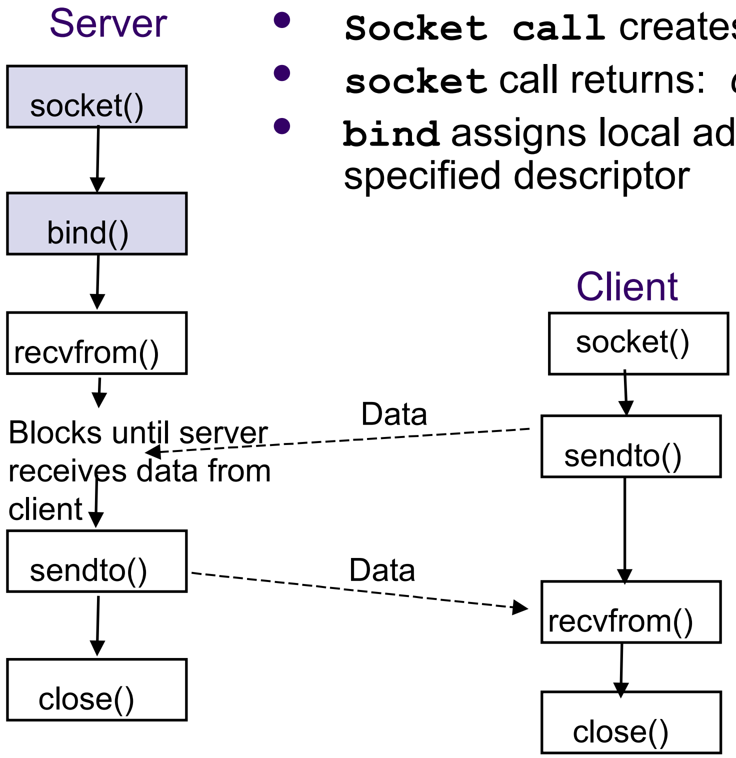
Connectionless (UDP)

- Immediate transfer of one block of information (boundaries preserved)
- No setup overhead & delay
- Destination address with each block
- Send/receive to/from multiple peer processes
- Best-effort service only
  - Possible out-of-order
  - Possible loss

Socket Calls for Connection-Less Mode

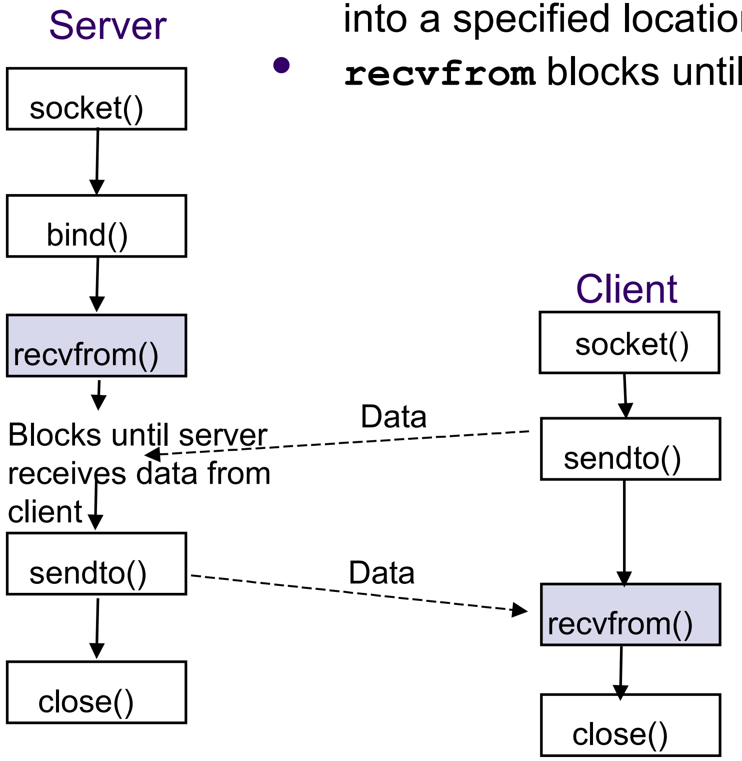
Server starts first

- **`Socket call`** creates socket of type UDP (datagram)
- **`socket`** call returns: *descriptor*; or -1 if unsuccessful
- **`bind`** assigns local address & port # to socket with specified descriptor



Socket Calls for Connection-Less Mode

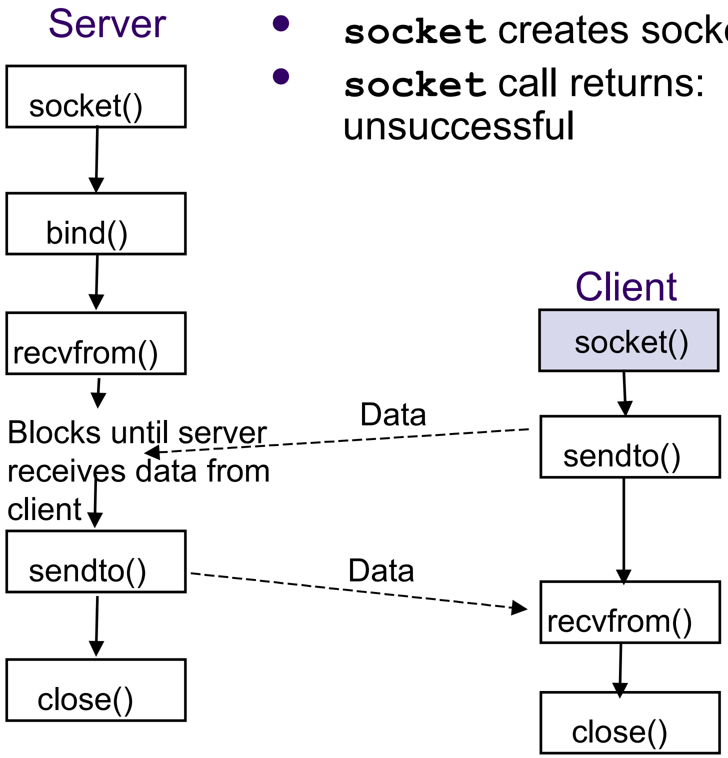
- **`recvfrom`** copies bytes received in specified socket into a specified location
- **`recvfrom`** blocks until data arrives



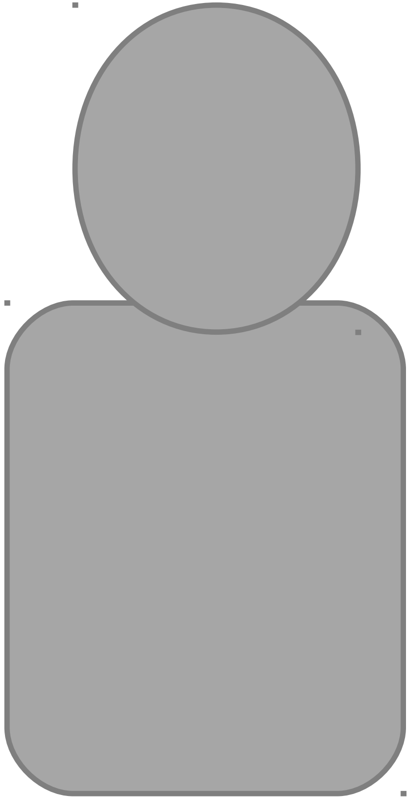
Socket Calls for Connection-Less Mode

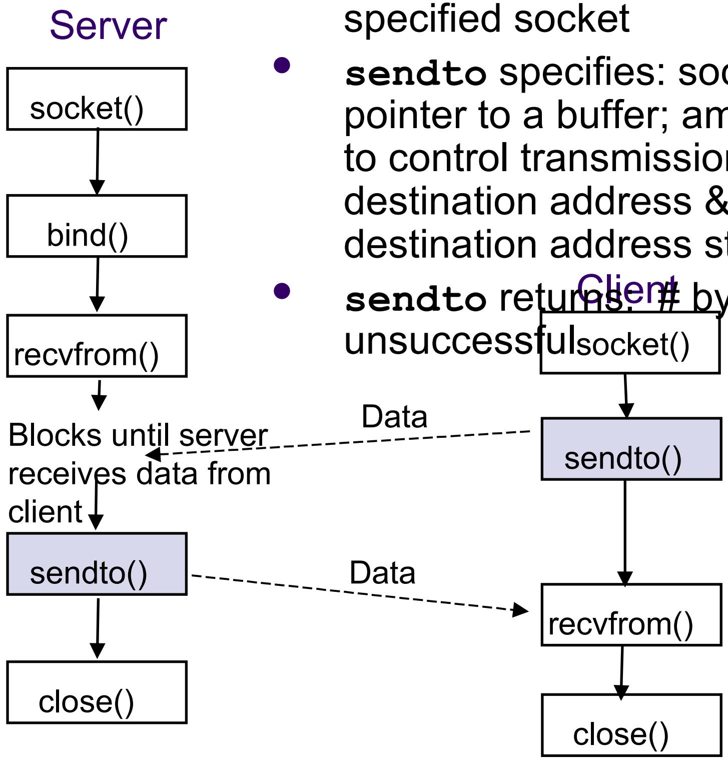
Client started

- `socket` creates socket of type UDP (datagram)
- `socket` call returns: *descriptor*; or -1 if unsuccessful

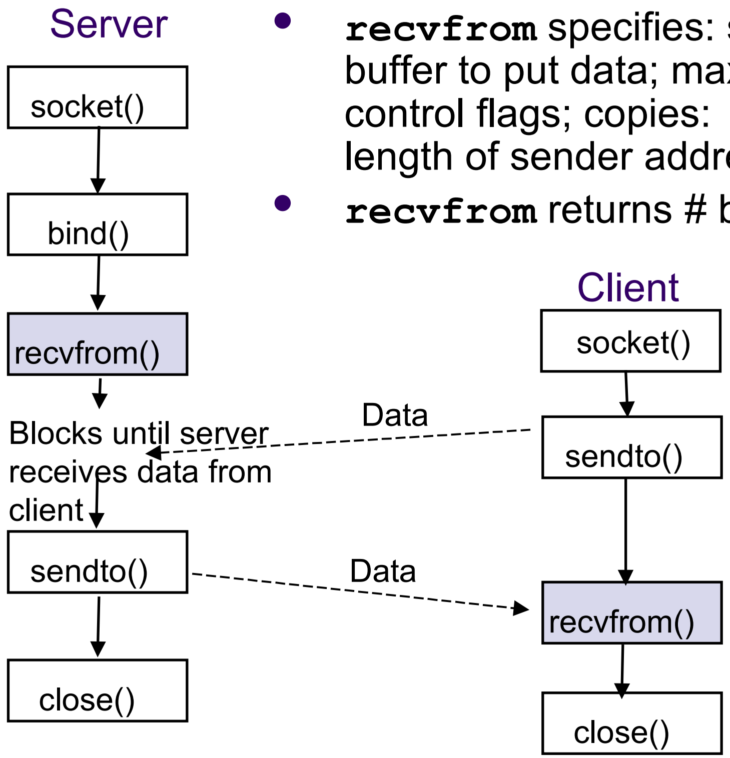


- **`sendto`** transfer bytes in buffer to specified socket
- **`sendto`** specifies: socket descriptor; pointer to a buffer; amount of data; flags to control transmission behavior; destination address & port #; length of destination address structure
- **`sendto`** returns: # bytes sent; or -1 if unsuccessful





- **`recvfrom`** wakes when data arrives
- **`recvfrom`** specifies: socket descriptor; pointer to a buffer to put data; max # bytes to put in buffer; control flags; copies: sender address & port #; length of sender address structure
- **`recvfrom`** returns # bytes received or -1 (failure)

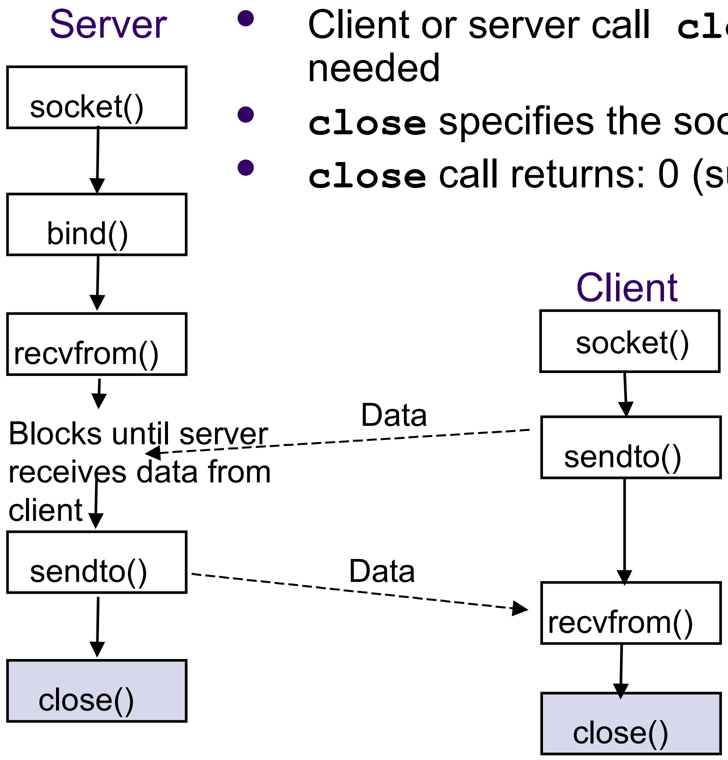


Note: **`recvfrom`** returns data from at most one **`send`**, i.e. from one datagram

Socket Calls for Connection-Less Mode

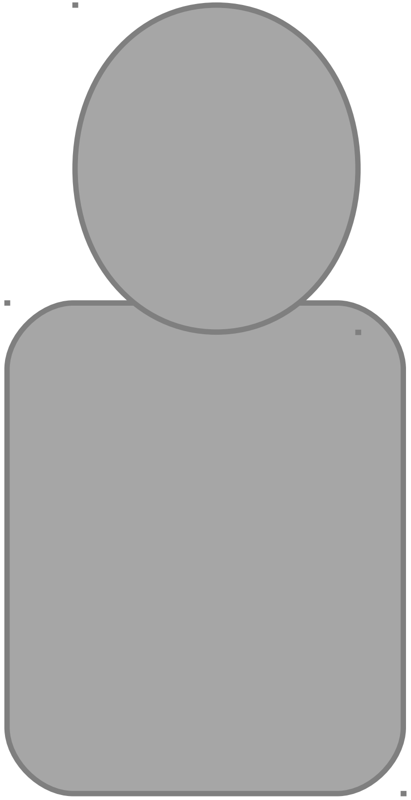
Socket Close

- Client or server call **`close`** when socket is no longer needed
- **`close`** specifies the socket descriptor
- **`close`** call returns: 0 (success); or -1 (failure)



Example-I: TCP Echo Server

- As illustration of the use of system calls and functions, let’s see two programs communicate via TCP.
- The client prompts a user to type a line of text and sends it to the server, and reads the data back from the server.
- The server aces as a simple each server.
- In this example, each program expects a fixed number of bytes from the other end, defined by BUFLEN.
- The example code is given in the Textbook Chapter 2.4



TCP Echo Server - Binding

```
/* Bind an address to the socket */
bzero((char *)&server, sizeof(struct
sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&server,
sizeof(server)) == -1) {
   fprintf(stderr, "Can't bind name to
socket\n");
   exit(1);
}
```

TCP Echo Server - Connections

```
/* queue up to 5 connect requests */
listen(sd, 5);

while (1) {
    client_len = sizeof(client);
    if ((new_sd = accept(sd, (struct sockaddr *)&client,
&client_len)) == -1) {
           fprintf(stderr, "Can't accept client\n");
           exit(1);
    }
```

TCP Echo Server – Repeated Byte Reads

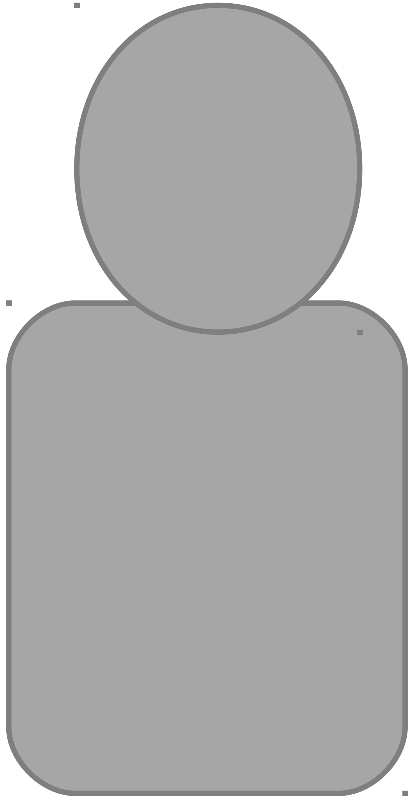
```
/* Repeated calls to read until all data received */
bp = buf;
bytes_to_read = BUFLEN;
while ((n = read(new_sd, bp, bytes_to_read)) > 0) {
      bp += n;
      bytes_to_read -= n;
}
printf("Rec'd: %s\n", buf);

write(new_sd, buf, BUFLEN);
printf("Sent: %s\n", buf);
close(new_sd);
```

TCP Echo Client – Name-to-Address

```
bzero((char *)&server, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
server.sin_port = htons(port);
if ((hp = gethostbyname(host)) == NULL) {
    fprintf(stderr, "Can't get server's address\n");
    exit(1);
}
bcopy(hp->h_addr, (char *)&server.sin_addr, hp->h_length);
```

TCP Echo Client - Connection



```
/* Connecting to the server */
if (connect(sd, (struct sockaddr *)
&server, sizeof(server)) == -1) {
    fprintf(stderr, "Can't connect\n");
    exit(1);
}
printf("Connected: server's address is
%s\n", hp->h_name);
```

TCP Echo Client – Repeated reads

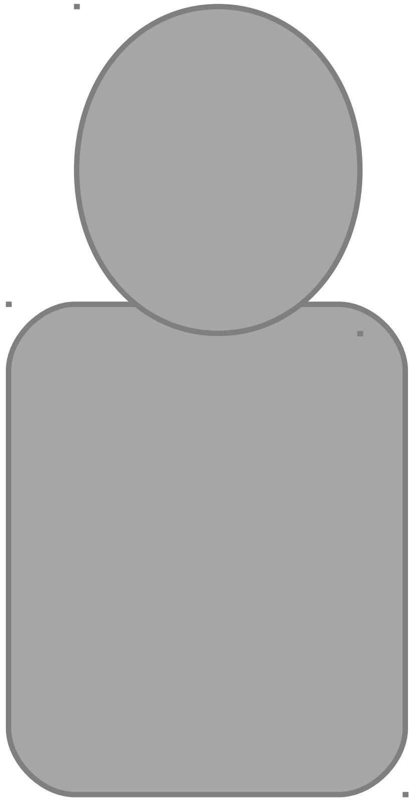
```
printf("Receive:\n");
bp = rbuf;
bytes_to_read = BUFLEN;
while ((n = read(sd, bp, bytes_to_read)) > 0) {
    bp += n;
    bytes_to_read -= n;
}
printf("%s\n", rbuf);
```

Example-II: UDP Echo Server

```
while (1) {
    client_len = sizeof(client);
    if ((n = recvfrom(sd, buf, MAXLEN, 0, (struct
    sockaddr *)&client, &client_len)) < 0) {
        fprintf(stderr, "Can't receive datagram\n");
        exit(1);
    }
    if (sendto(sd, buf, n, 0, (struct sockaddr
    *)&client, client_len) != n) {
         fprintf(stderr, "Can't send datagram\n");
         exit(1);
    }
}
```

Example: UDP Echo Client

```
gettimeofday(&start, NULL); /*start delay measurement*/
server_len = sizeof(server);
if (sendto(sd, sbuf, data_size, 0, (struct sockaddr *)
        &server, server_len) == -1) {
        fprintf(stderr, "sendto error\n")
        exit(1);
}
if (recvfrom(sd, rbuf, MAXLEN, 0, (struct sockaddr *)
        &server, &server_len) < 0) {
        fprintf(stderr, "recvfrom error\n");
        exit(1);
}
gettimeofday(&end, NULL); /* end delay measurement */
```



Summary: UDP Rliability

- As UDP is unreliable, users may have to take care of reliability assurance by themselves.
- LAN vs. WAN
- Timeout mechanism avoids forever wait
- Re-transmission to get a lost message
- Reordering and de-duplication are requiired for reliability