Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

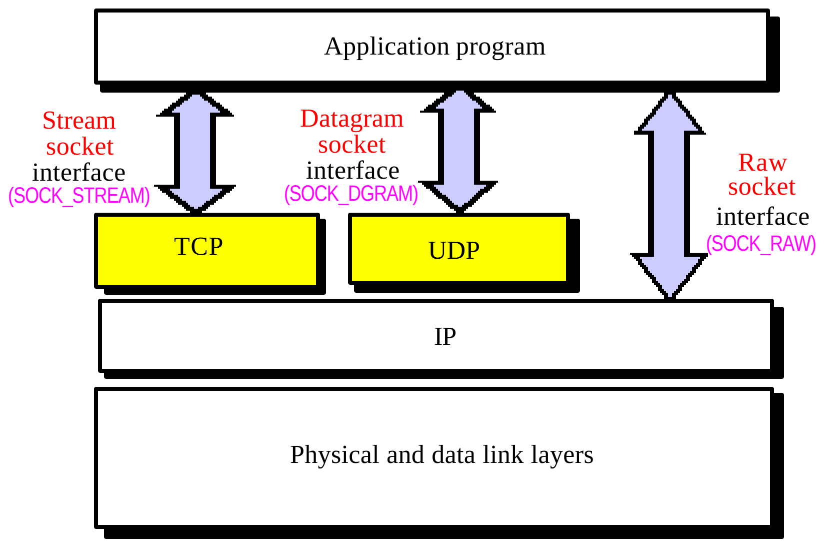
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

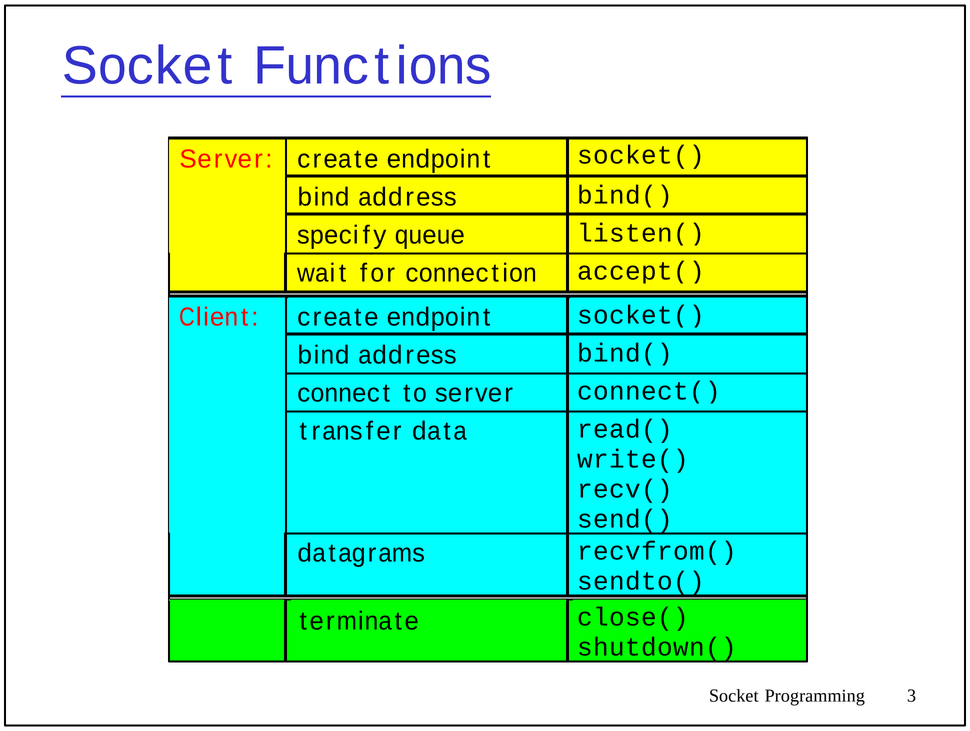
a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Socket Programming 1

Socket types


Application program
Stream socket interface (SOCK_STREAM)
Datagram socket interface (SOCK_DGRAM)
Raw socket interface (SOCK_RAW)
TCP
UDP
IP
Physical and data link layers

Socket Programming 2



Socket Functions

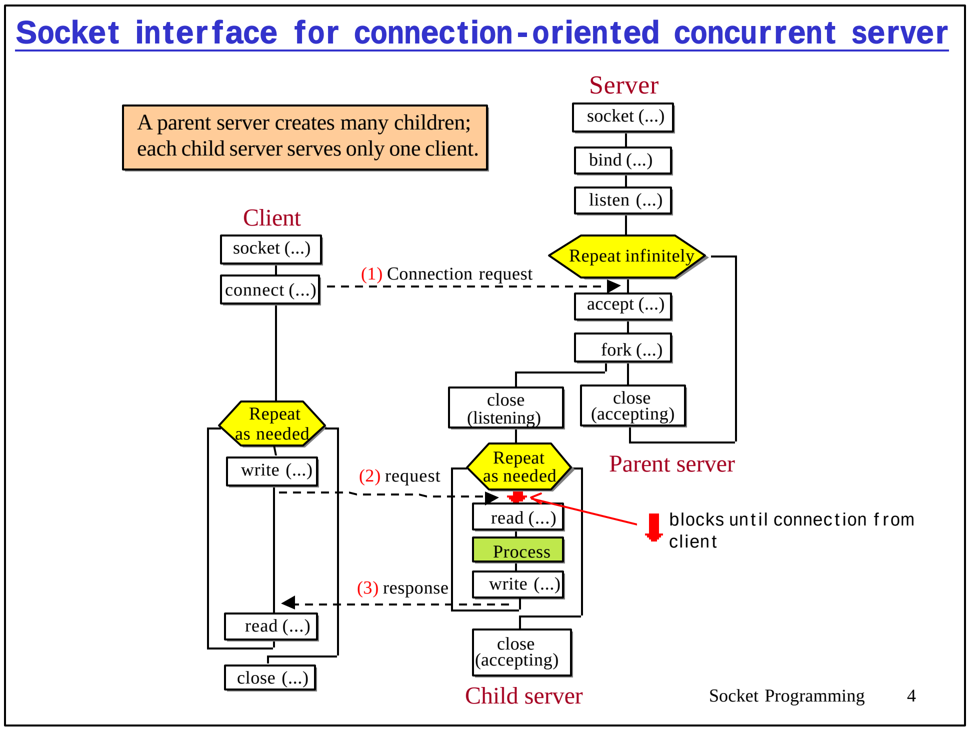
Server:	create endpoint	socket()
	bind address	bind()
	specify queue	listen()
	wait for connection	accept()
Client:	create endpoint	socket()
	bind address	bind()
	connect to server	connect()

	transfer data	read()
		write()
		recv()
		send()

	datagrams	recvfrom()
		sendto()

	terminate	close()
		shutdown()

Socket Programming 3



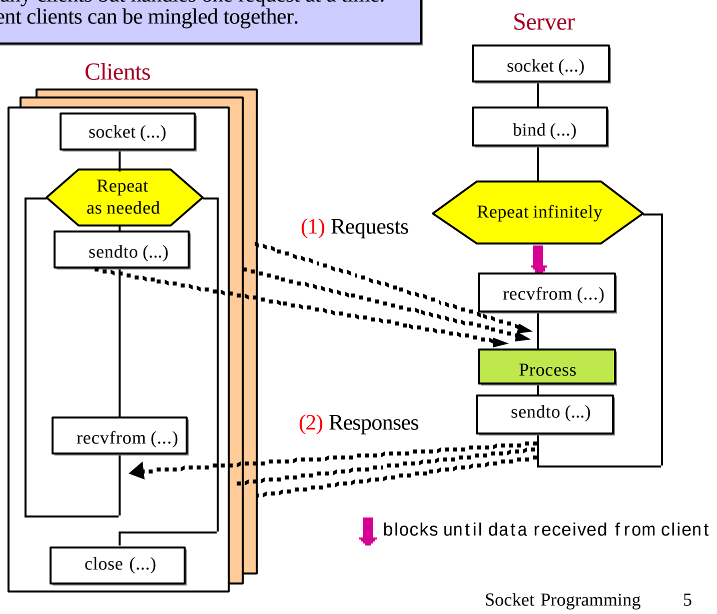
Socket interface for connection-oriented concurrent server

A parent server creates many children; each child server serves only one client.

Socket Programming 4

Socket interface for connectionless iterative server

Each server serves many clients but handles one request at a time. Requests from different clients can be mingled together.



Socket Addresses

Defined in <sys/socket.h>:

```
struct sockaddr {
     u_short    sa_family;        /* address family: AF_xxx value */
     char       sa_data[14];      /* up to 14 bytes of protocol-spec addr */
};
```

Defined in <netinet/in.h>:

```
struct in_addr {
     u_long     s_addr;           /* 32-bit netid/hostid */
};
struct sockaddr_in {
     short      sin_family;       /* AF_INET */
     u_short    sin_port;         /* 16-bit port number */
     struct     in_addr;          /* 32-bit netid/hostid */
     char       sin_zero[8];      /* unused */
};
```

Example: connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));

socket() System Call

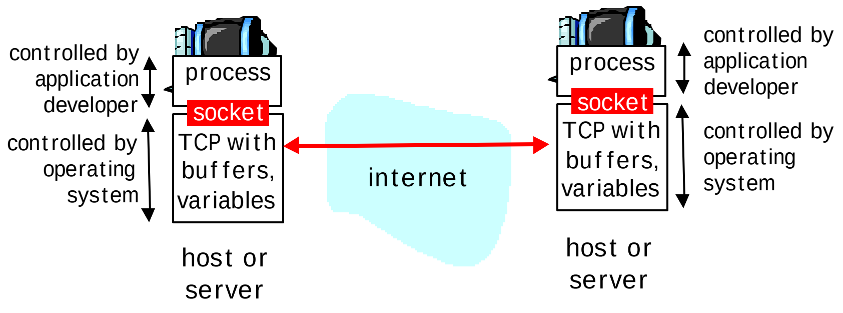
```
int socket (int family,  int type,       int protocol);
                AF_UNIX      SOCK_STREAM
                AF_INET      SOCK_DGRAM
                             SOCK_RAW
```

family	*type*	*protocol*	Actual protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)

Socket-programming with TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

TCP Concurrent Server Program

```
#include <stdio.h>        #include <sys/types.h>   #include <sys/socket.h>
#include <netinet/in.h>   #include <netdb.h>       #include <string.h>

#define PORT        0x1234
#define DIRSIZE     8192

main()
{
        char        dir[DIRSIZE];  /* used for incomming dir name, and outgoing data */
        int         sd, sd_current, cc, fromlen, tolen;
        int         addrlen;
        struct      sockaddr_in sin;
        struct      sockaddr_in pin;

        /* get an internet domain socket */
        if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                    perror("socket");
                    exit(1);
        }

        /* complete the socket structure */
        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(PORT);
```

TCP Concurrent Server Program (cont'd)

```
        /* bind the socket to the port number */
        if (bind(sd, (struct sockaddr *) &sin, sizeof(sin)) == -1) {
                perror("bind");
                exit(1);
        };    printf("After bind.\n");

        /* show that we are willing to listen */
        if (listen(sd, 5) == -1) {
                perror("listen");
                exit(1);
        };    printf("After listen.\n");

        /* wait for a client to talk to us */
        if ((sd_current = accept(sd, (struct sockaddr *)  &pin, &addrlen)) == -1) {
                perror("accept");
                exit(1);
        };    printf("After accept.\n");

        /* get a message from the client */
        if (recv(sd_current, dir, sizeof(dir), 0) == -1) {
                perror("recv");
                exit(1);
        }; printf("After read_dir.\n");
```

TCP Concurrent Server Program (cont'd)

```
        /* acknowledge the message, reply w/ the file names */
        if (send(sd_current, dir, strlen(dir), 0) == -1) {
                perror("send");
                exit(1);
        }
    printf("After send.\n");

    /* close up both sockets */
        close(sd_current); close(sd);
    printf("After close.\n");

    /* give client a chance to properly shutdown */
    sleep(1);
}
```

TCP Concurrent Client Program

```
#include <stdio.h>          #include <sys/types.h>    #include <sys/socket.h>
#include <netinet/in.h>     #include <netdb.h>         #include <string.h>

#define PORT        0x1234    /* REPLACE with your server machine name*/
#define HOST        "redhat21"
#define DIRSIZE     8192

main(argc, argv)
        int argc; char **argv;
{
        char        hostname[100];
        char        dir[DIRSIZE];
        int         sd;
        struct sockaddr_in   sin;
        struct sockaddr_in   pin;
        struct hostent       *hp;

        strcpy(hostname,HOST);
        if (argc>2)  strcpy(hostname,argv[2]);

        /* go find out about the desired host machine */
        if ((hp = gethostbyname(hostname)) == 0) {
                perror("gethostbyname");
                exit(1);
        };    printf("After gethostbyname.\n");
```

Socket Programming 13

TCP Concurrent Client Program (cont'd)

```
        /* fill in the socket structure with host information */
        memset(&pin, 0, sizeof(pin));
        pin.sin_family = AF_INET;
        pin.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;
        pin.sin_port = htons(PORT);
    printf("After fill in the socket struct.\n");

        /* grab an Internet domain socket */
        if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        };
   printf("After socket.\n");

        /* connect to PORT on HOST */
        if (connect(sd,(struct sockaddr *)  &pin, sizeof(pin)) == -1) {
                perror("connect");
                exit(1);
        };
   printf("After connect.\n");
```

Socket Programming 14

TCP Concurrent Client Program (cont'd)

```
        /* send a message to the server PORT on machine HOST */
        if (send(sd, argv[1], strlen(argv[1]), 0) == -1) {
                perror("send");
                exit(1);
        };
  printf("After send.\n");

    /* wait for a message to come back from the server */
    if (recv(sd, dir, DIRSIZE, 0) == -1) {
        perror("recv");
        exit(1);
    }
  printf("After recv.");

    /* spew-out the results and bail out of here! */
  printf("%s\n", dir);

        close(sd);
}  /* main */
```

Socket programming with UDP

UDP: no "connection" between client and server

r no handshaking

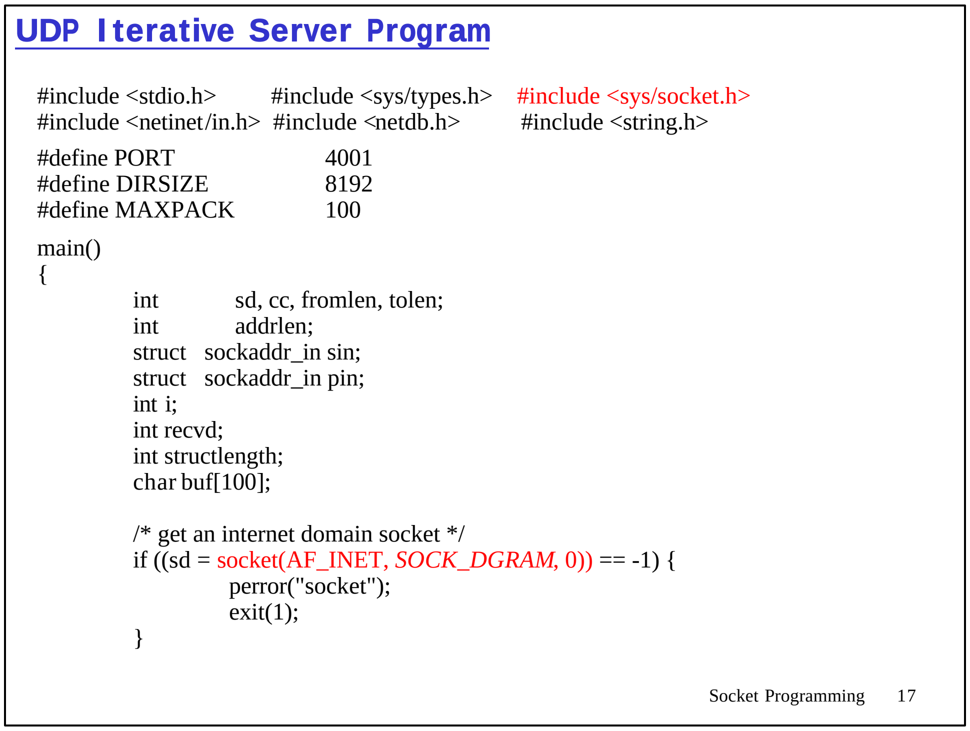
r sender explicitly attaches IP address and port of destination

r server must extract IP address, port of sender from received datagram

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server



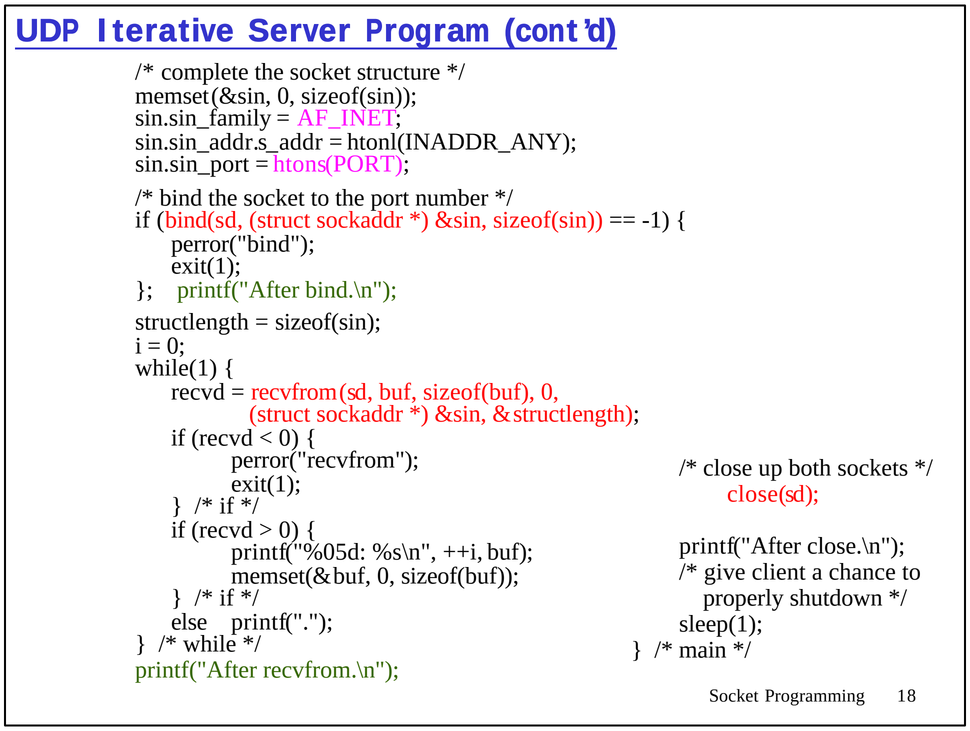
UDP Iterative Server Program

```
#include <stdio.h>        #include <sys/types.h>    #include <sys/socket.h>
#include <netinet/in.h>   #include <netdb.h>        #include <string.h>

#define PORT          4001
#define DIRSIZE       8192
#define MAXPACK       100

main()
{
        int        sd, cc, fromlen, tolen;
        int        addrlen;
        struct   sockaddr_in sin;
        struct   sockaddr_in pin;
        int i;
        int recvd;
        int structlength;
        char buf[100];

        /* get an internet domain socket */
        if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }
```



UDP Iterative Server Program (cont'd)

```
        /* complete the socket structure */
        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = htonl(INADDR_ANY);
        sin.sin_port = htons(PORT);

        /* bind the socket to the port number */
        if (bind(sd, (struct sockaddr *) &sin, sizeof(sin)) == -1) {
           perror("bind");
           exit(1);
        };   printf("After bind.\n");

        structlength = sizeof(sin);
        i = 0;
        while(1) {
           recvd = recvfrom(sd, buf, sizeof(buf), 0,
                      (struct sockaddr *) &sin, &structlength);
           if (recvd < 0) {
                perror("recvfrom");
                exit(1);
           }  /* if */
           if (recvd > 0) {
                printf("%05d: %s\n", ++i, buf);
                memset(&buf, 0, sizeof(buf));
           }  /* if */
           else   printf(".");
        }  /* while */
        printf("After recvfrom.\n");

        /* close up both sockets */
           close(sd);

        printf("After close.\n");
        /* give client a chance to
           properly shutdown */
        sleep(1);
}  /* main */
```

UDP Iterative Client Program

```
#include <stdio.h>        #include <sys/types.h>    #include <sys/socket.h>
#include <netinet/in.h>  #include <netdb.h>        #include <string.h>

#define PORT                 4001      /* REPLACE with your server machine name*/
#define HOST                 "redhat21"
#define MAXPACK              100

main(argc, argv)
int argc; char **argv;
{
        char                 hostname[100];
        int                  port_no, sd, i, j, x;
        struct sockaddr_in   sin;
        struct sockaddr_in   pin;
        struct hostent       *hp;
        char                 buff[10];

        strcpy(hostname,HOST);
        port_no = PORT;
        if (argc == 3)  {    strcpy(hostname,argv[1]);
                             port_no = atoi(argv[2]);        }

        /* go find out about the desired host machine */
        if ((hp = gethostbyname(hostname)) == 0) {
                perror("gethostbyname");
                exit(1);
        };      printf("After gethostbyname.\n");
```

UDP Iterative Client Program (Cont'd)

```
        /* fill in the socket structure with host information */
        memset(&pin, 0, sizeof(pin));
        pin.sin_family = AF_INET;
        pin.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;
        pin.sin_port = htons(port_no);
       printf("After fill in the socket struct.\n");

        /* grab an Internet domain socket */
        if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }
       printf("After socket.\n");

        for (i = 1; i <= MAXPACK; i++) {
            memset(buff, 0, sizeof(buff));
            sprintf(buff, "[%03d]", i);
            printf(">> Sending %s\n", buff);
            if (sendto(sd, &buff, sizeof(buff), 0, (struct sockaddr *) &pin, sizeof(pin)) < 0)
                   perror("sendto");

            // delay between sending two messages
            // sleep(1);
        }  /* for */      printf("After sendto\n");

        close(sd);
  }  /* main */
```