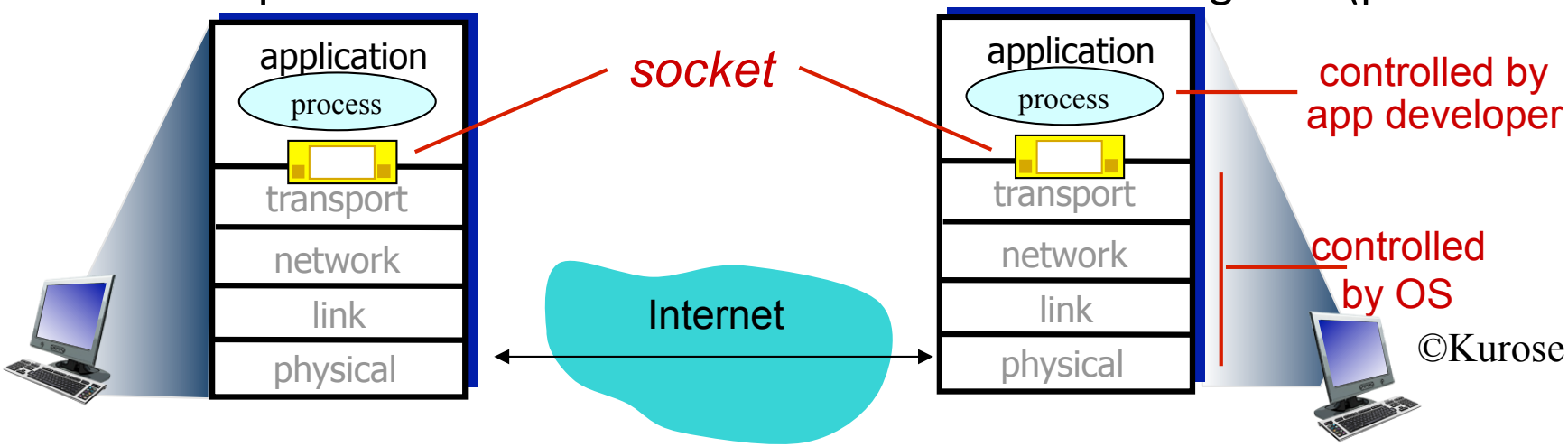
How processes communicate

- **Sockets** provide the application programmers' interface (API) between a **process** and the transport layer (sys/socket.h, java.net).
- User application code runs on end-systems - not network core
- The application programmer needs to specify
  - which transport protocol to use
  - what host to send messages to (e.g. IP address or hostname)
  - what process on the destination host to send messages to (port number)



Internet Transport Services

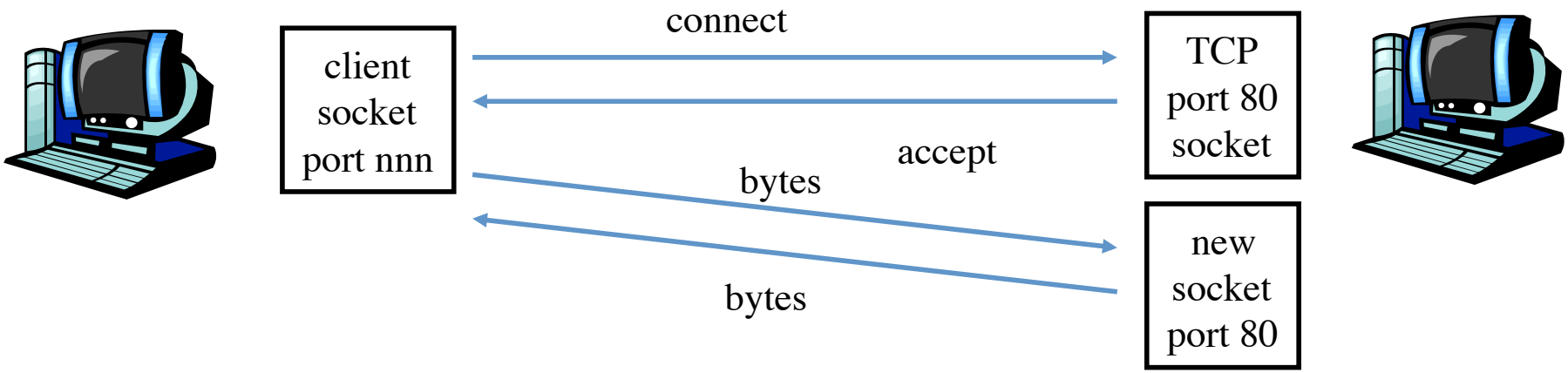
- What services do applications need?
  - Reliable data transfer, Minimum throughput guarantees, Bounded delays, Security
- What do the Internet protocols provide?
  - Reliable data transfer with transmission control protocol TCP
  - Minimal overhead, available bandwidth/delays, no delivery guarantee with user datagram protocol UDP
  - emerging protocols for providing timing and bandwidth guarantees
- Current choices in Internet are TCP or UDP. How does a network application designer decide?

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Socket Programming with TCP

- Recall that TCP provides a reliable byte stream. All of our data will be going to the same host and port (ie to the same process).
- Assume we want to get a web page. We want to talk to www.foo.com on port 80. If we stay connected to the socket on port 80, how will www.foo.com service other requests?
- port 80 is used to establish a connection on a **second server socket**.

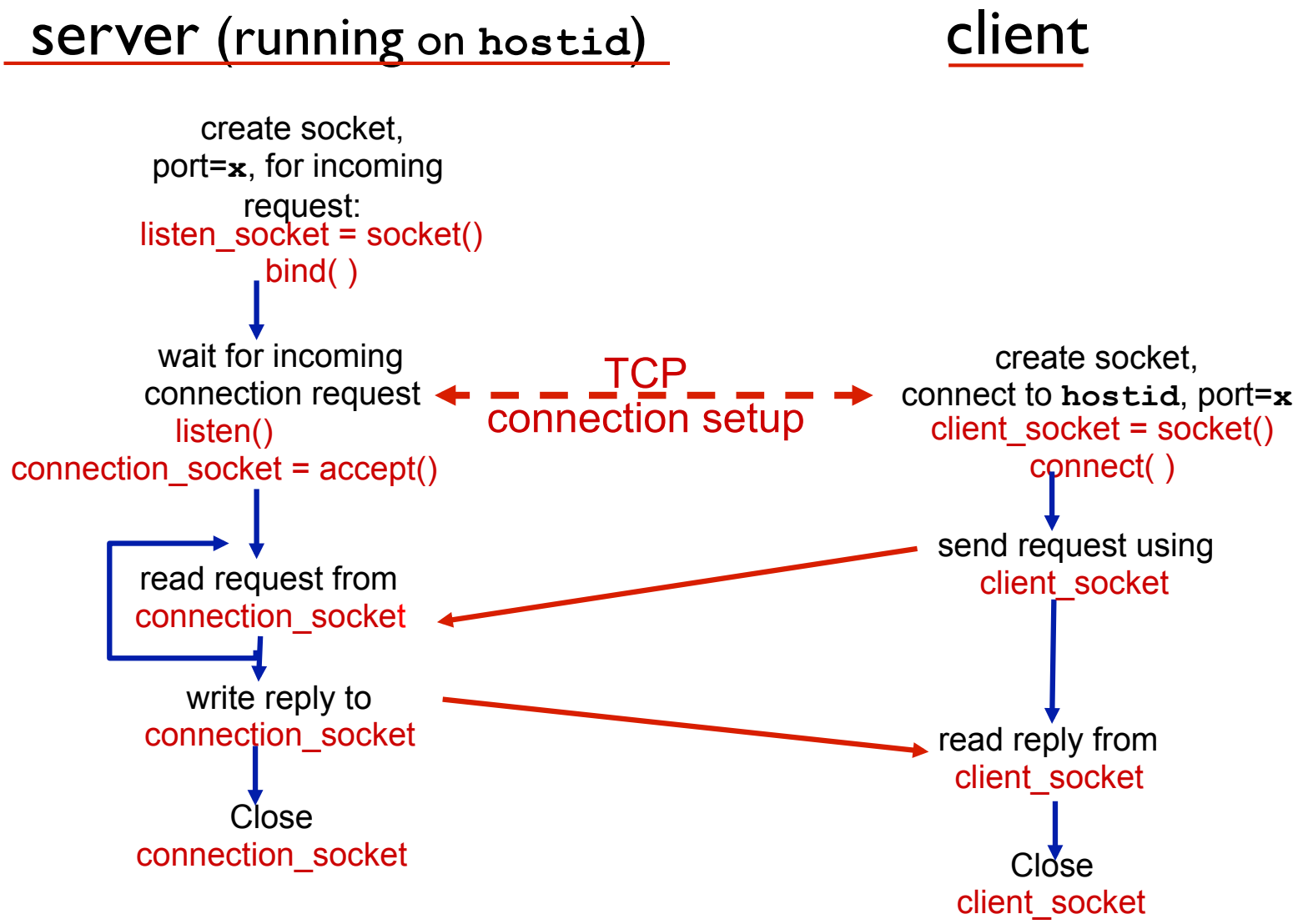


Socket programming with TCP

Application Example:

1. Client reads a character (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts character to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the character on its screen.

Client/server socket interaction: TCP



C TCP Client

Libraries we will use

```
#include <sys/types.h>  // socket, recv, send, close
#include <sys/socket.h>
#include <netdb.h>  //  getaddrinfo
#include <unistd.h> // close
#include <string.h> // memset
#include <stdio.h> // fgets, fputs, puts
#include <stdbool.h> // true false

int main(int argc, char *argv[]) {

  char * SERVER_NAME = "localhost";
  char * SERVER_PORT = "6789";
```

Set these to the hostname and port of the ***SERVER***

"localhost" works if the server is running on the same computer as the client. Effectively "this computer"

The port must be the port the server is listening on.

You need to set these for *your* application.

C TCP Client

```
/* Set socket address structures */
struct addrinfo hints;     // hints for the type of socket wanted
struct addrinfo * server_addr;  // holder for the address information

/* clear memory of  the structures */
memset(&hints,0,sizeof(hints));
memset(&server_addr,0,sizeof(server_addr));

/* set hints on type of connection */
 hints.ai_family = AF_INET;   /* set address family to IPv4 */
 hints.ai_socktype = SOCK_STREAM;  /* use TCP stream */

 getaddrinfo(SERVER_NAME, SERVER_PORT, &hints, &server_addr);
```

The address information is now stored in server_addr. We will use this information to create and connect the socket

C TCP Client

```
/* Create socket */
 int client_socket = socket(server_addr->ai_family, server_addr->ai_socktype, 0);

 /* connect socket */
 if (connect(client_socket,  server_addr->ai_addr, server_addr->ai_addrlen))
   perror("connect failed: ");

 char character;
 while(true) {
   puts("Input lowercase letter: ");
   character = getchar();
   send(client_socket,&character,sizeof(char),0);
```

Note we can use perror( ) to print out the reason for failure. This should be done for all function calls that might fail.

Loop forever getting a character from the user, sending it to the server on the connected socket…

C TCP Client

```
  puts("Server returned: ");
  recv(client_socket, &character, sizeof(char),0);
  putchar(character);

  getchar(); // read the newline character
  putchar('\n');

 }
 close(client_socket);

 freeaddrinfo(server_addr);
}
```

… still in the loop. Read what character the server sends back on the connected socket and print it.

We won't ever reach this code since we have an infinite loop above. But if we were to finish we should close the connected socket so the server knows we are finished and free dynamic memory.

Both will happen automatically (by the operating system) if we kill the client process.

C TCP Server

```
#include <sys/types.h> // socket, recv, send, close bind
#include <sys/socket.h>
#include <netdb.h> // getaddrinfo
#include <unistd.h>  // close
#include <string.h>  // memset
#include <ctype.h> // toupper
#include <stdlib.h> // NULL

int main(int argc, char *argv[])
{
  struct addrinfo hints; // fill this in with the type of address wanted
  struct addrinfo * server_addr; // structure to hold server's address
  int    listen_socket, connection_socket;
  char * DEFAULT_PORT = "6789";
  int    QLEN = 1;
```

Libraries we will use.

Note similarities and differences from client

This is the port the server will listen on and the client will connect to. If it doesn't match, the client will not reach the server application.

QLEN is the number of simultaneous requests. We will only connect to one client at a time.

C TCP Server

```
/* Set up server address structure:  serv_addr */
 /* This sets the port and and IP address that we will bind to */
 /* the socket we just created */

 memset(&hints, 0, sizeof(hints));
 memset(&server_addr,0,sizeof(server_addr)); /* clear sockaddr structure */

 hints.ai_family = AF_INET;
 hints.ai_socktype = SOCK_STREAM;
 hints.ai_flags = AI_PASSIVE; // fill in my IP address

 getaddrinfo(NULL, DEFAULT_PORT, &hints, &server_addr);
```

Address set up is similar to the client address set up

But notice this difference. This hint tells getaddrinfo to fill in my (the servers) address for me.

I don't have to give the server name like I did in the client.

C TCP Server

```
/* Create a socket */
listen_socket = socket(server_addr->ai_family, server_addr->ai_socktype, 0);

/* Bind the socket to the address information set in serv_addr */
bind(listen_socket, server_addr->ai_addr, server_addr->ai_addrlen);

/* Start listening for connections */
listen(listen_socket, QLEN);

/* Accept and handle requests */
connection_socket = accept(listen_socket, NULL, NULL);
```

Different here! The client just connected. The server must bind, listen and accept.

We have two sockets now. connection_socket and listen_socket. Which will we talk to the client on?

C TCP Server

```
/* Receive the characters */
char character;
char capital_character;

while(recv(connection_socket, &character, sizeof(char),0)) {
  /* send capitalised character back */
  capital_character = (char) toupper((int)character);
  send(connection_socket, &capital_character, sizeof(char),0);
}

/* finished, close the socket */
close(connection_socket);
close (listen_socket);
}
```

Receive returns the number of characters read and will receive 0 when the client closes the socket.

Send it back to the client

How would this differ if we wanted to keep running the server and accept another client connection?

Socket programming with UDP

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination
- 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

Client/server socket interaction: **UDP**

server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
bind()

read datagram recvfrom()
serverSocket

write reply sendto()
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and port=x; send datagram via clientSocket

read datagram from
clientSocket

close
clientSocket