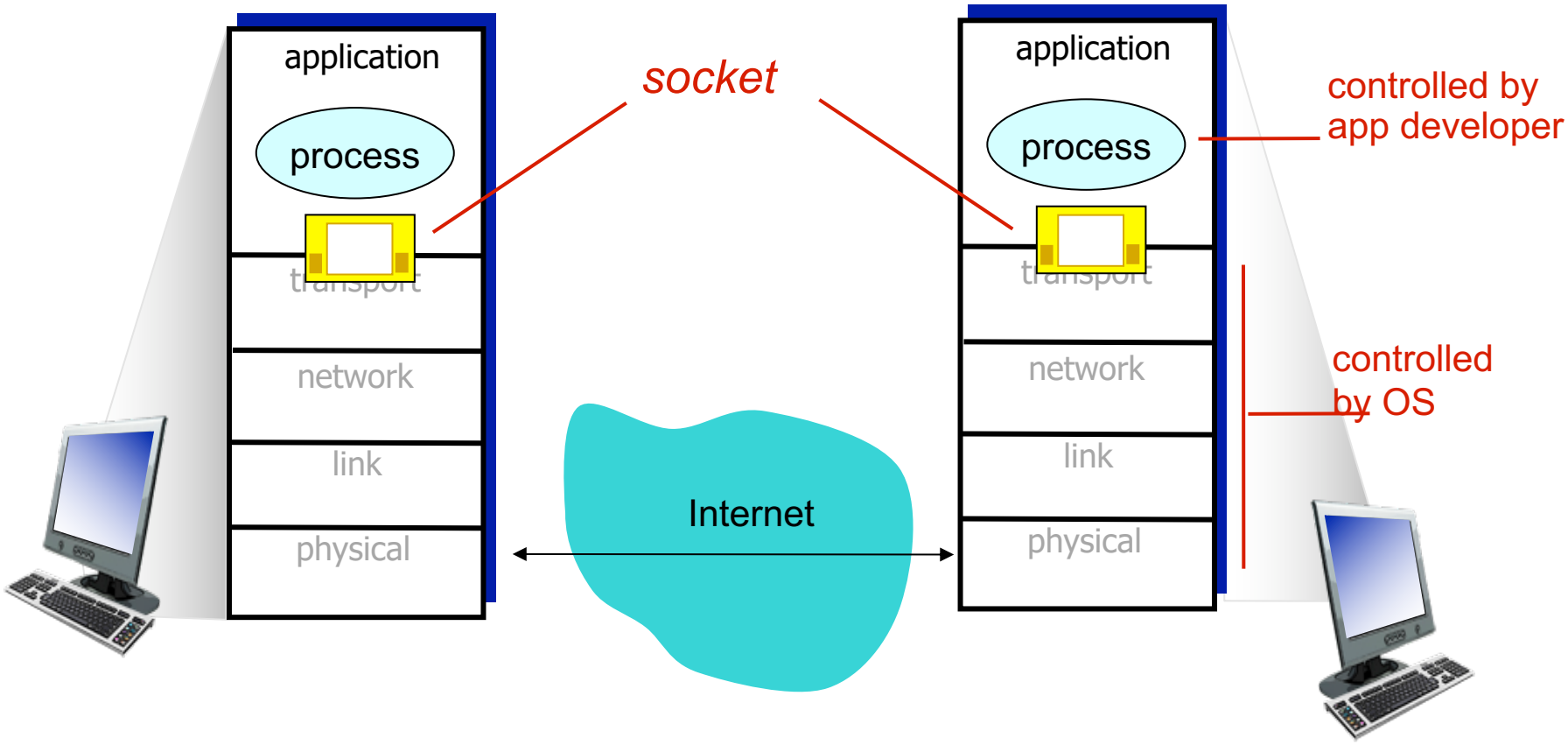
Programing Assignment #1
Echo Server

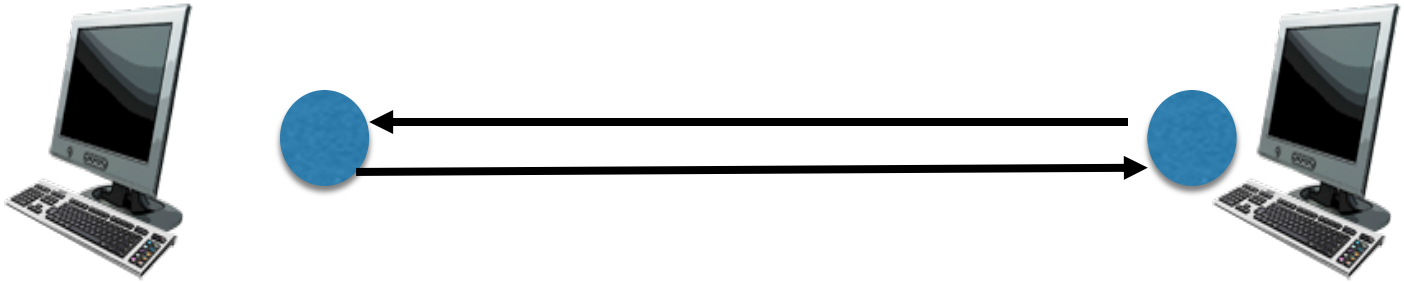
Build a simple ECHO server, using the socket API, that accepts incoming connection requests and then echoes back any string that is sent. We will use **telnet** to connect to and test your program.

Due Sunday, September 12th.

Socket programming



Stream (Connection) based sockets



Server – Waits for a connection request.

Client – Initiates the connection.

Server creates the socket and sets it into "listen" mode, which means it is waiting for someone to call.

Client creates socket, then calls and tells the server.

Server answers accepting call and then either side can send/receive information over the "pipe"

Addresses – where is the server?

Each computer has (at least) one network address (i.e. the IP Address).

138.67.100.1

Server – Waits for a connection request.

138.67.100.2

Client – Initiates the connection.

Addresses – where is the socket?

Each **PROCESS** may have one or more sockets, identified by port number.

138.67.100.1:**80**

Server – Waits for a connection request.

138.67.100.2:**9823**

Client – Initiates the connection.

Do you remember File Descriptors?

```
int fd = open(filename);
char buffer[1024];
read (fd,buffer,10);
write(fd, buffer,10);
close(fd);
```

Do you remember *errno*?

```
int bytesWritten = 0;
if ((bytesWritten = write(fd,buf,10)) < 0) {
   cout << strerr(errno) << endl;
   }
```

Server Side Steps

1. Create the socket

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

int socket(int domain, int type, int protocol);
```

Domain	Type
PF_LOCAL	SOCK_STREAM
PF_UNIX	SOCK_DGRAM
PF_INET	SOCK_RAW
PF_INET6	SOCK_RDM
PF_IPX	SOCK_PACKET
PF_ATMPVC	
PF_PACKET	

```
int lfd = -1;
if ((lfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
   cout << "Failed to create listening socket "
        << strerror(errno) <<  endl;
   exit(-1);
 }
```

Server Side Steps

1. Create the socket.
2. Bind the socket to an address and port number.

```
struct sockaddr_in
  {
    sa_family_t sa_prefix##family;      /* Address family. */
    in_port_t sin_port;                 /* Port number.  */
    struct in_addr sin_addr;            /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) - …];
  };
```

Name	Function
hton()	Convert character string to network format
ntoh()	Convert from network format to character string
htons()	Host to network short
htonl()	Host to network long
ntohs()	Network to host short
ntohl()	Network to host long

```
// Define the structure
struct sockaddr_in  servaddr;

// Zero the whole thing.
bzero(&servaddr, sizeof(servaddr));

// IPv4 Protocol Family
servaddr.sin_family      = PF_INET;

// Let the system pick the IP address.
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

// You pick a random high-numbered port
#define PORT 9873
servaddr.sin_port        = htons(PORT);
```

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

int bind( int sockfd, const struct sockaddr *my_addr,
       socklen_t addrlen);
```

Common Errors	Menaing
EACCES	Don't have access to the fd
EADDRINUSE	Local address in use.
EAFNOSUPPORT	Address family not supported.
EFAULT	Address of my_addr is invalid.
EBADF	Bad file descriptor.
ENOTSOCK	The fd points to a file not a socket.

```
if (bind(lfd, (sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
  cout << "bind() failed: " << strerror(errno) <<  endl;
  exit(-1);
}
```

Server Side Steps

1. Create the socket.
2. Bind the socket to an address and port number.
3. Create the listening queue and associate it with the socket.

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

int listen(int sockfd, int backlog);
```

Common Errors	Menaing
EBADF	Bad file descriptor.
EADDRINUSE	Local address in use.
EAFNOSUPPORT	Address family not supported.
ENOTSOCK	The argument does not point to a socket.
EOPNOTSUPP	The socket type does not support listen().

```
int listenq = 1;
if (listen(listenfd, listenq) < 0) {
  cout << "listen() failed: " << strerror(errno) <<  endl;
   exit(-1);
}
```

Server Side Steps

1. Create the socket.
2. Bind the socket to an address and port number.
3. Create the listening queue and associate it with the socket.
4. Wait for a connection with the accept call.

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

int accept( int sockfd, struct sockaddr *addr,
      socklen_t *addrlen);
```

Common Errors	Menaing
EAGAIN	Non-blocking socket would block
EBADF	File descriptor is bad.
EINTR	System call was interrupted.
EMFILE	File limit reached.
ENOTSOCK	FD points to a file not a socket
EOPNOTSUPP	The referenced socket is not of type SOCK_STREAM

```
while (1) {
  int connfd = -1;
  if ((connfd = accept(listenfd, (sockaddr *) NULL, NULL)) < 0) {
     cout << "accept() failed: " << strerror(errno) <<  endl;
     exit(-1);
  }

  createThreadAndProcess(connfd);
}
```

When accept returns.

1. accept() returns a new file descriptor.
2. Process the transaction with the client.
    1. Typically the server will fork() or create a new thread in which it processes the request.
3. Call accept() again to wait for the next connection.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
```

Common Errors	Type
EAGAIN	Non-blocking I/O with no data available.
EBADF	Bad file descriptor
EFAULT	Buf is a bad address.
EINTR	The call was interrupted before any data was read.
EINVAL	File descriptor is attached to something that can be read from.
EISDIR	

Client Side Steps

1. First 2 steps are the same:
    1. Create the socket.
    2. Bind the socket to an address and port number.
2. Call connect() to connect to the server.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr
            *serv_addr, socklen_t addrlen);
```

Common Errors	Menaing
EACCES	Don't have access to the fd
EADDRINUSE	Local address in use.
EAFNOSUPPORT	Address family not supported.
EALREADY	Socket is non-blocking and a request is already pending.
EBADF	Bad file descriptor.
ECONNREFUSED	No one listening on the other end.

Client Side Steps

1. First 2 steps are the same:
    1. Create the socket.
    2. Bind the socket to an address and port number.
2. Call connect() to connect to the server.
3. Typically the client does not fork, but rather uses the FD created by socket() to communicate with the server.