Win32 Sockets

Jim Fawcett

CSE 687-OnLine – Object Oriented Design

Summer 2017

What are Sockets?

- Sockets provide a common interface to the various protocols supported by networks.
- They allow you to establish connections between machines to send and receive data.
- Sockets support the simultaneous connection of multiple clients to a single server machine.

TCP Protocol

- ***TCP/IP stands for "Transmission Control Protocol / Internet Protocol*.**
  TCP/IP is the most important of several protocols used on the internet. Some others are: HyperText Transport Protocol (HTTP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and Telnet, a protocol for logging into a remote computer. Sockets provide a standard interface for a variety of network protocols. TCP/IP is, by far, the most commonly used protocol for sockets. Here are the main features of TCP/IP:

  - ***IP is a routable protocol.***
    That means that TCP/IP messages can be passed between networks in a Wide Area Network (WAN) cluster.

  - ***Each device using TCP/IP must have an IP address*.**
    This address is a 32 bit word, organized into four 8-bit fields, called octets. Part of the IP address identifies the network and the rest identifies a specific host on the network.

  - ***IP addresses are organized into three classes*.**
    Each class has a different allocation of octets to these two identifiers. This allows the internet to define many networks, each containing up to 256 devices (mostly computers), and a few networks, each containing many more devices.

  - ***A single machine can run mulitple communictaions sessions using TCP/IP.***
    That is, you can run a web browser while using Telnet and FTP, simultaneously.

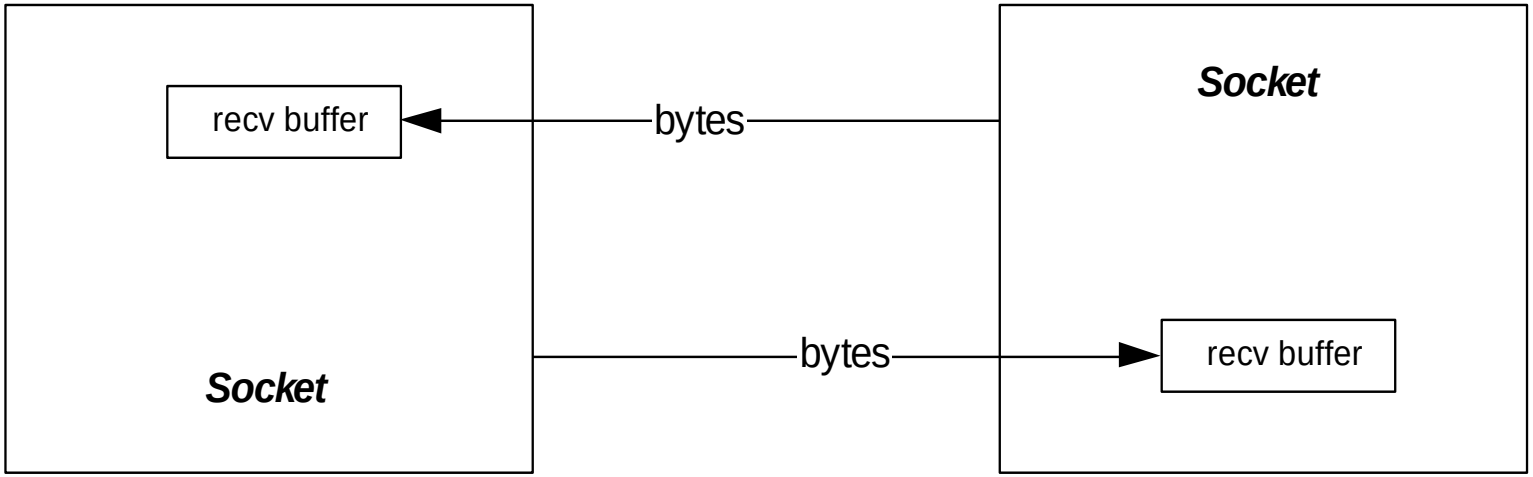
TCP/IP based Sockets

- Connection-oriented means that two communicating machines must first connect.

- All data sent will be received in the same order as sent.
  - Note that IP packets may arrive in a different order than that sent.
  - This occurs because all packets in a communication do not necessarily travel the same route between sender and receiver.

- Streams mean that, as far as sockets are concerned, the only recognized structure is bytes of data.

Socket Logical Structure



Creating Sockets

- Socket connections are based on:
  - Domains – network connection or IPC pipe
    - AF_INET for IPv4 protocol
    - AF_INET6 for IPv6 protocol
  - Type – stream, datagram, raw IP packets, …
    - SOCK_STREAM → TCP packets
    - SOCK_DGRAM → UDP packets
  - Protocol – TCP, UDP, …
    - 0 → default, e.g., TCP for SOCK_STREAM
  - Example:

```
HANDLE sock = socket(AF_INET,SOCK_STREAM,0);
```

Connecting Sockets

- Socket addresses

```
struct SOCKADDR_IN {
    sin_family              // AF_INET
    sin_address.s_addr      // inet_addr(“127.0.0.1”);
    sin_port                // htons(8000);
} addr;
```

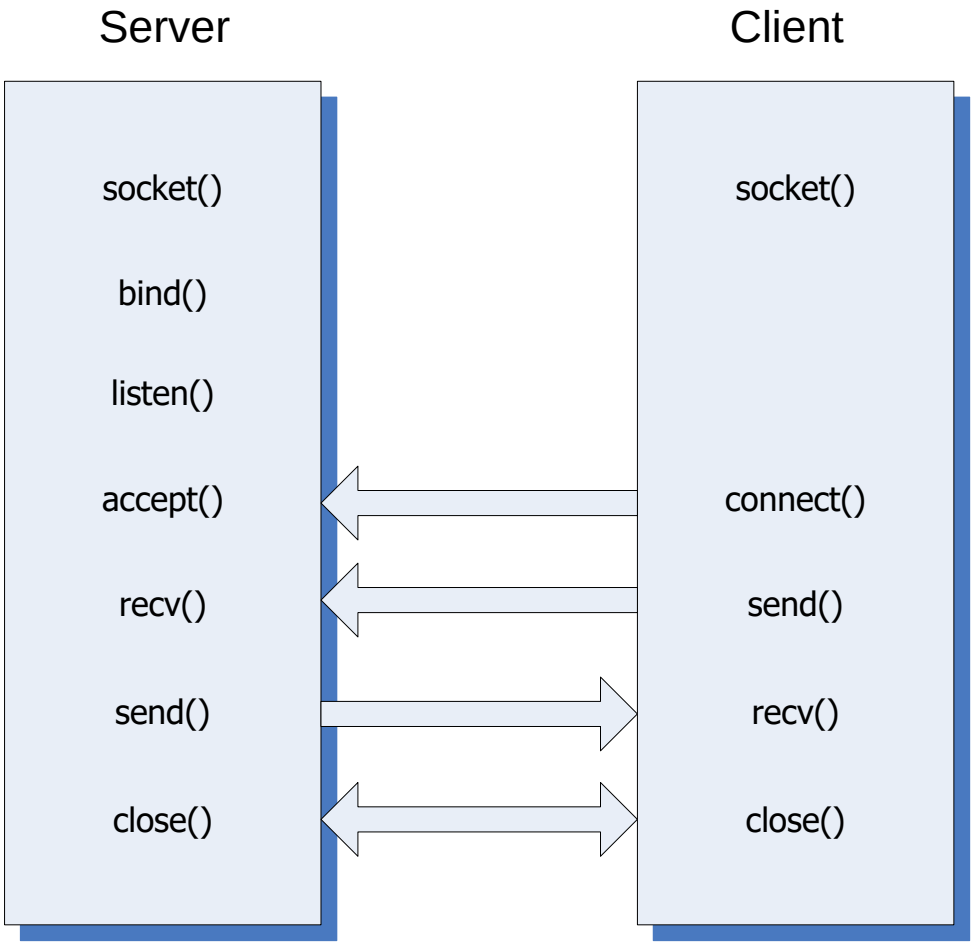
- Bind server listener to port:

```
int err = bind(sock, (SOCKADDR_IN*)&addr,sizeof(addr));
```

- Connect client to server:

```
HANDLE connect(sock, (SOCKADDR_IN*)&addr,sizeof(addr))
```

Client / Server Processing



Accessing Sockets Library

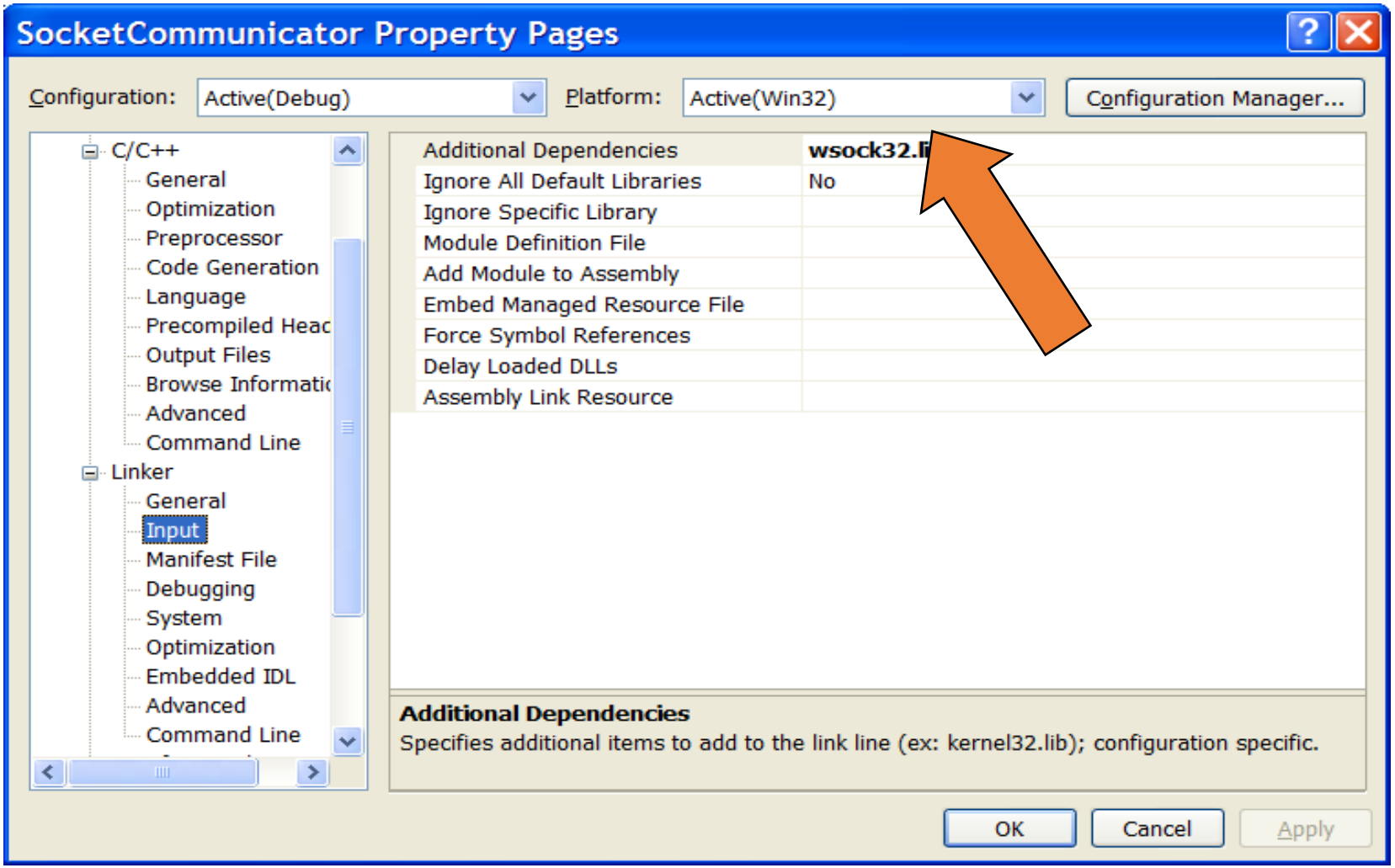
- `#include <winsock2.h>`
- Link with wsock32.lib
- To build a server for multiple clients you will need to use threads, e.g.:

```
#include <process.h> // Win32 threads
    or
#include<threads>    // C++11 threads
```

and use the Project Settings:

C/C++ language\category=code generation\debug multithreaded

Project Settings



Sockets API

- WSAStartup - loads WS2_32.dll
- WSACleanup - unloads dll
- socket - create socket object
- connect - connect client to server
- bind - bind server socket to address/port
- listen - request server to listen for connection requests
- accept - server accepts a client connection
- send - send data to remote socket
- recv - collect data from remote socket
- Shutdown - close connection
- closesocket - closes socket handle

Sequence of Server Calls

- WSAStartup
- socket (create listener socket)
- bind
- listen
- accept
  - create new socket so listener can continue listening
  - create new thread for socket
  - send and recv
  - closesocket (on new socket)
  - terminate thread
- shutdown
- closesocket (on listener socket)
- WSACleanup

WSAStartup

```
wVersionRequested = MAKEWORD(1,1);
WSAData wData;
lpWSAData = &wData

int WSAStartup(
  WORD wVersionRequested,
  LPWSADATA lpWSAData
)
```

- Loads `WS2_32.dll`

TCP/IP socket

```
af       = AF_INET
type     = SOCK_STREAM
protocol = IPPROTO_IP

SOCKET socket(int af, int type, int protocol)
```

- Creates a socket object and returns handle to socket.

Bind socket

```
struct sockaddr_in local;
… define fields of local …
name = (sockaddr*)&local
namelen = sizeof(local)

int bind(
  SOCKET s,
  const struct sockaddr *name,
  int namelen
)
```

- Bind listener socket to network card and port

Listen for incoming requests

```
int listen(SOCKET s, int backlog)
```

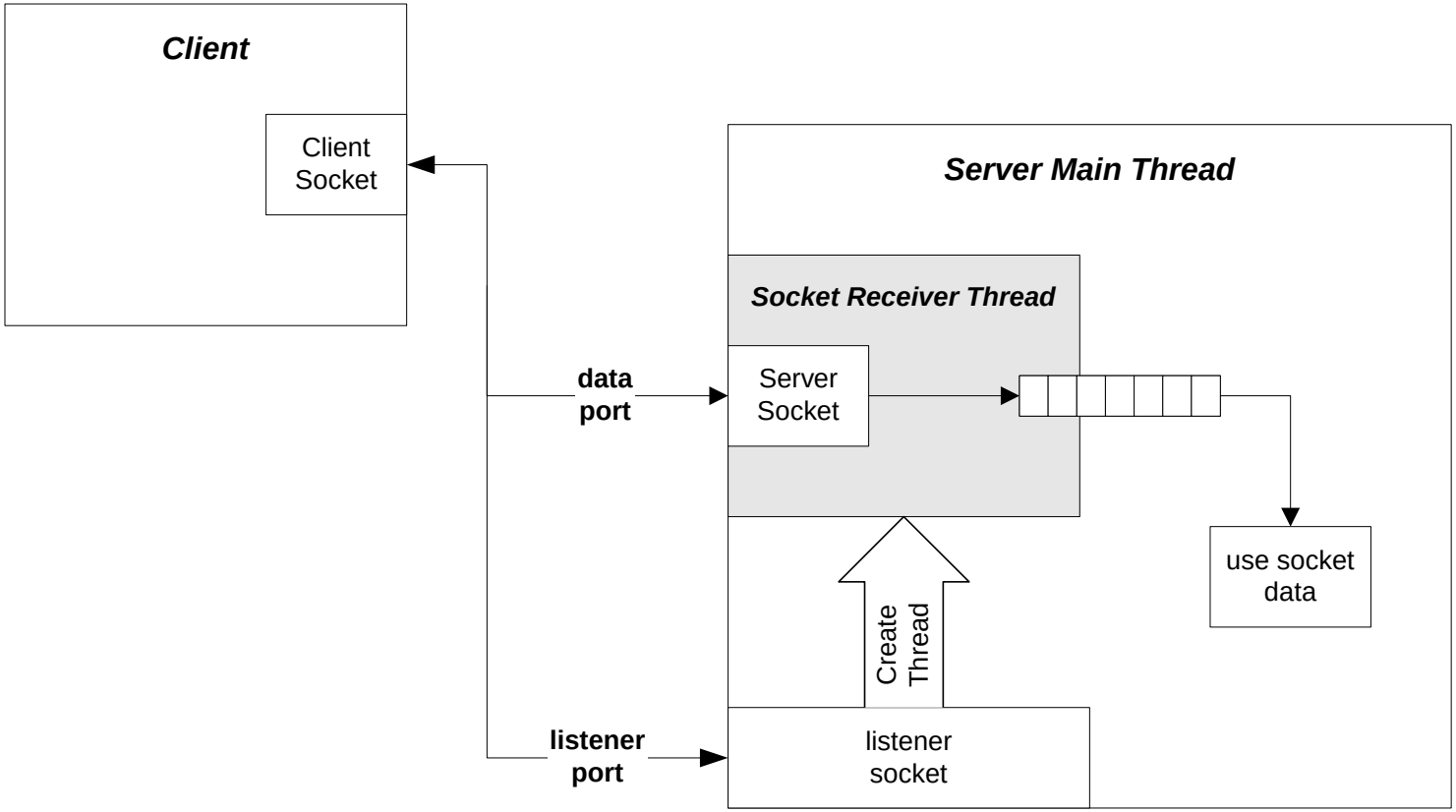
- backlog is the number of incoming connections queued (pending) for acceptance
- Puts socket in listening mode, waiting for requests for service from remote clients.

Accept Incoming Connection

```
SOCKET accept(
  SOCKET s,
  struct sockaddr *addr,
  int *addrLen
)
```

- Blocking call, accepts a pending request for service and returns a socket bound to a new port for communication with new client.
- Usually server will spawn a new thread to manage the socket returned by accept, often using a thread pool.

Client/Server Configuration



recv

```
int recv(
  SOCKET s,
  char *buff,
  int len,
  int flags
)
```

- Receive data in buff up to len bytes.
- Returns actual number of bytes read.
- flags variable should normally be zero.

send

```
int send(
  SOCKET s,
  char *buff,
  int len,
  int flags
)
```

- Send data in buff up to len bytes.
- Returns actual number of bytes sent.
- flags variable should normally be zero.

shutdown

```
int shutdown(SOCKET s, int how)
```

- how = SD_SEND or SD_RECEIVE or SD_BOTH

- Disables new sends, receives, or both, respectively. Sends a FIN to server causing thread for this client to terminate (server will continue to listen for new clients).

closesocket

```
int closesocket(SOCKET s)
```

- **Closes socket handle s, returning heap allocation for that data structure back to system.**

WSACleanup

```
int WSACleanup( )
```

- Unloads `W2_32.dll` if no other users. Must call this once for each call to `WSAStartup`.

Sequence of Client Calls

- WSAStartup
- socket
- address resolution - set address and port of intended receiver
- connect - send and recv
- shutdown
- closesocket
- WSACleanup

TCP Addresses – IP4

```
struct sockaddr_in{
    short               sin_family;
    unsigned short      sin_port;
    struct in_addr      sin_addr;
    char                sin_zero[8];
 } SOCKADDR_IN;
```

TCP/IP Address fields - IP4

- sin_family AF_INET
- sin_port at or above 1024
- sin_addr inet_addr(“127.0.0.1”);
- sin_zero padding

  - Setting sin_addr.s_addr = INADDR_ANY allows a server application to listen for client activity on every network interface on a host computer.

connect

```
int connect(
  SOCKET s,
  const struct sockaddr *name,
  int namelen
)
```

- Connects client socket to a specific machine and port.

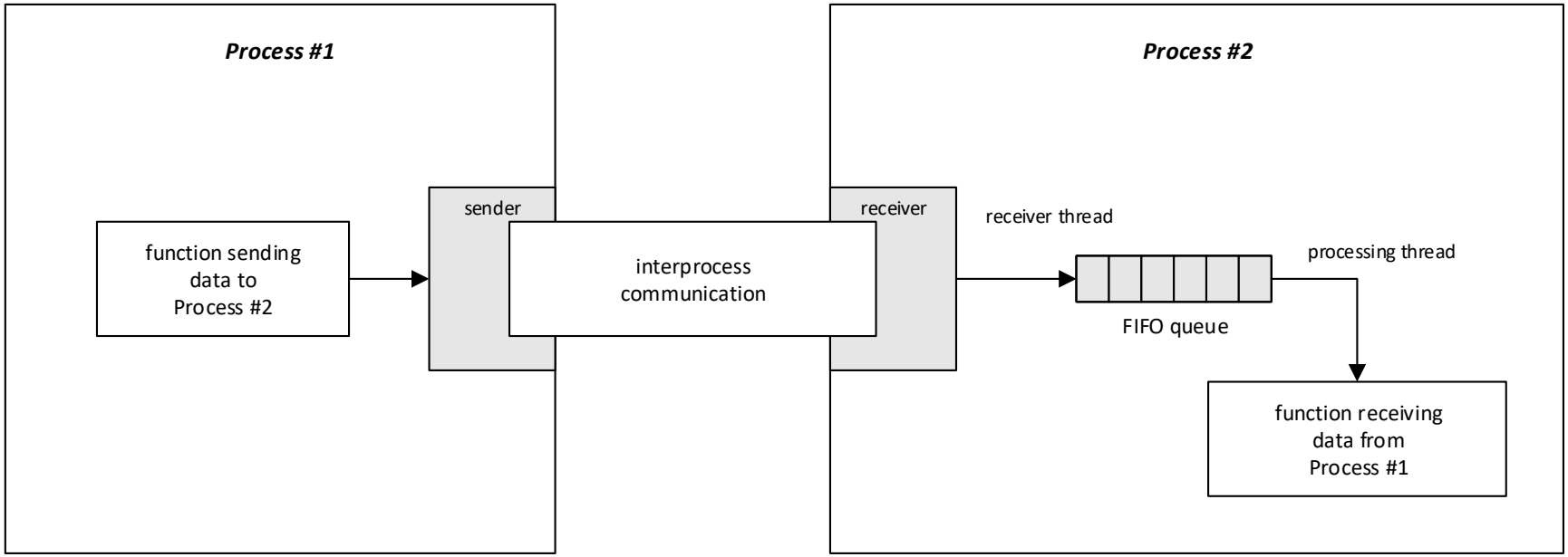
Special Functions

- htons – converts short from host to network byte order
- htonl – converts long from host to network byte order
- ntohs – converts short from network to host byte order
- ntohl – converts long from network to host byte order

A Word of Caution

- With stream oriented sockets, send does not guarantee transfer of all bytes requested in a single call.
- That's why send returns an int, the number of bytes actually send.
- It's up to you to ensure that all the bytes are actually sent
    - See my code example – socks.cpp

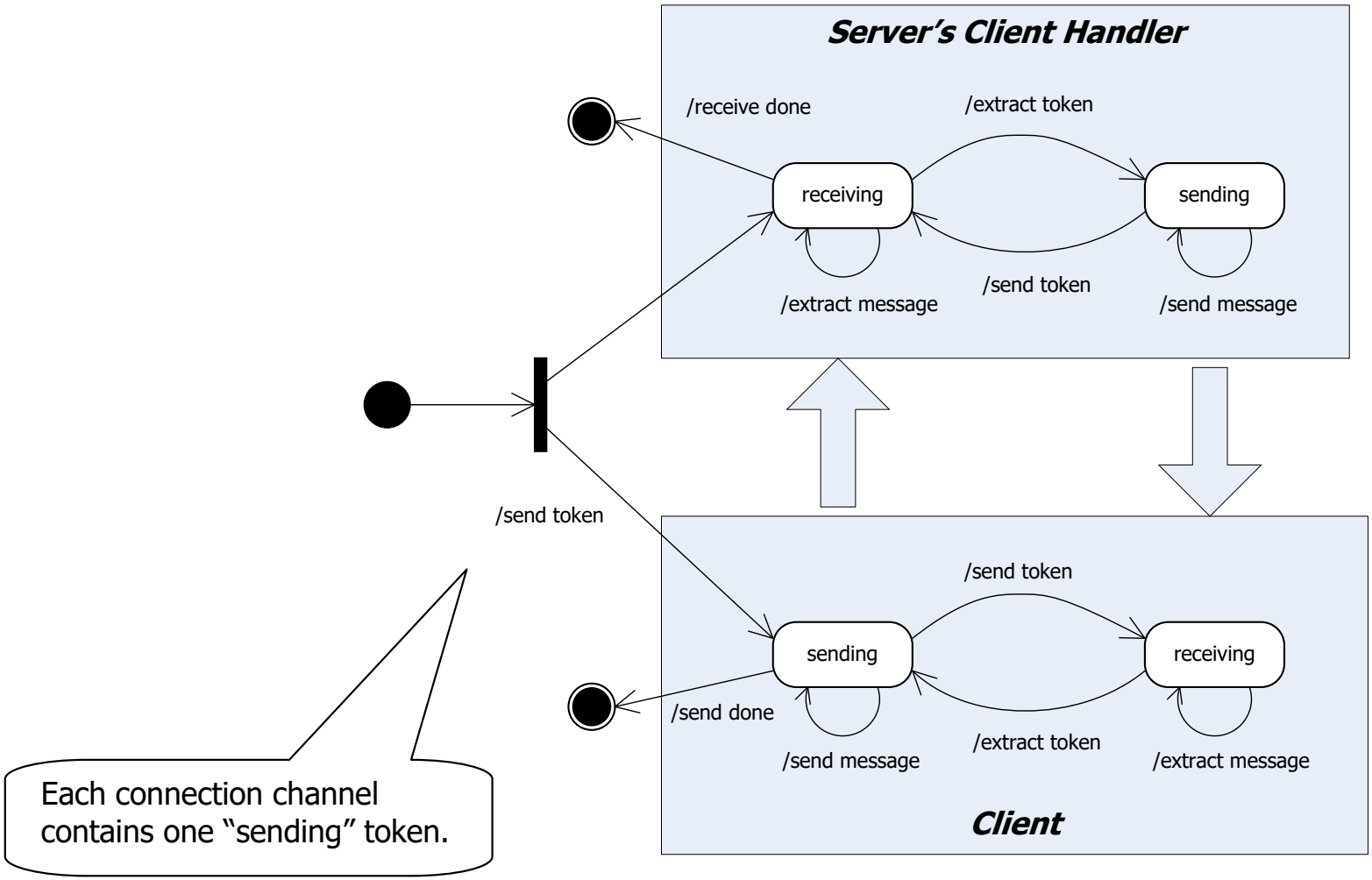
Non-Blocking Communication



Talk Protocol

- The hardest part of a client/server socket communication design is to control the active participant
  - If single-threaded client and server both talk at the same time, their socket buffers will fill up and they both will block, e.g., deadlock.
  - If they both listen at the same time, again there is deadlock.
  - Often the best approach is to use separate send and receive threads

State Chart - Socket
Bilateral Communication Protocol



Message Length

- Another vexing issue is that the receiver may not know how long a sent message is.
  - so the receiver doesn't know how many bytes to pull from the stream to compose a message.
  - Often, the communication design will arrange to use message delimiters, fixed length messages, or message headers that carry the message length as a parameter.
  - For examples see:
    - Repository/CppStringSocketServer // uses string delimiter
    - Repository/CommWithFileXfer // uses messages with headers

Message Framing

- Sockets only understand arrays of bytes
  - Don’t know about strings, messages, or objects
- In order to send messages you simply build the message string, probably with XML
  - string msg = “<msg>message text goes here</msg>”
  - Then send(sock,msg,strlen(msg),flags)
- Receiving messages requires more work
  - Read socket one byte at a time and append to message string:
  - recv(sock,&ch,1,flags); msg.append(ch);
  - Search string msg from the back for </
  - Then collect the msg>
- You will find a more sophisticated approach in the CommWithFileXfer, cited on the previous slide

They’re Everywhere

- Virtually every network and internet communication method uses sockets, often in a way that is invisible to an application designer.
  - Browser/server
  - ftp
  - SOAP
  - Network applications

What we didn’t talk about

- udp protocol
- socket select(…) function
- non-blocking sockets

The End