# Sockets

CSE 384

Spring 2020

Based on Dr. Fawcett's Socket Presentation

https://ecs.syr.edu/faculty/fawcett/handouts/CSE687/presentations/Win32Sockets.pdf

# 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.

- Background Resources On TCP/IP
  - https://www.guru99.com/tcp-ip-model.html
  - https://www.guru99.com/tcp-3-way-handshake.html
  - https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html
  - https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

# 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
  - REST
  - Network applications

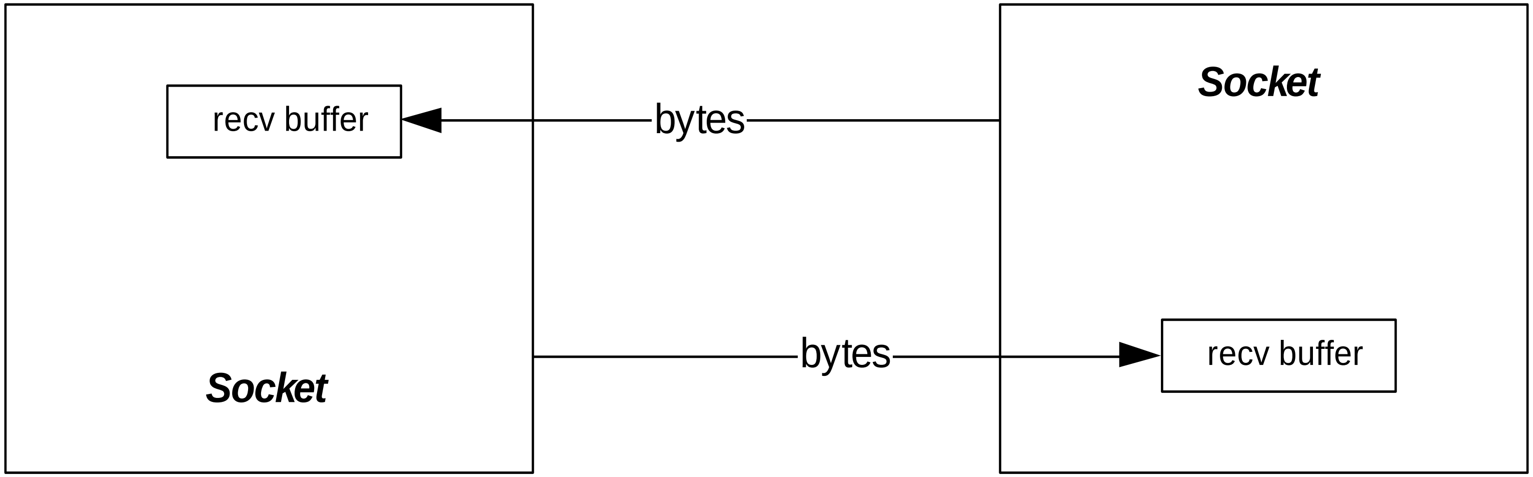
# TCP/IP based Sockets

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

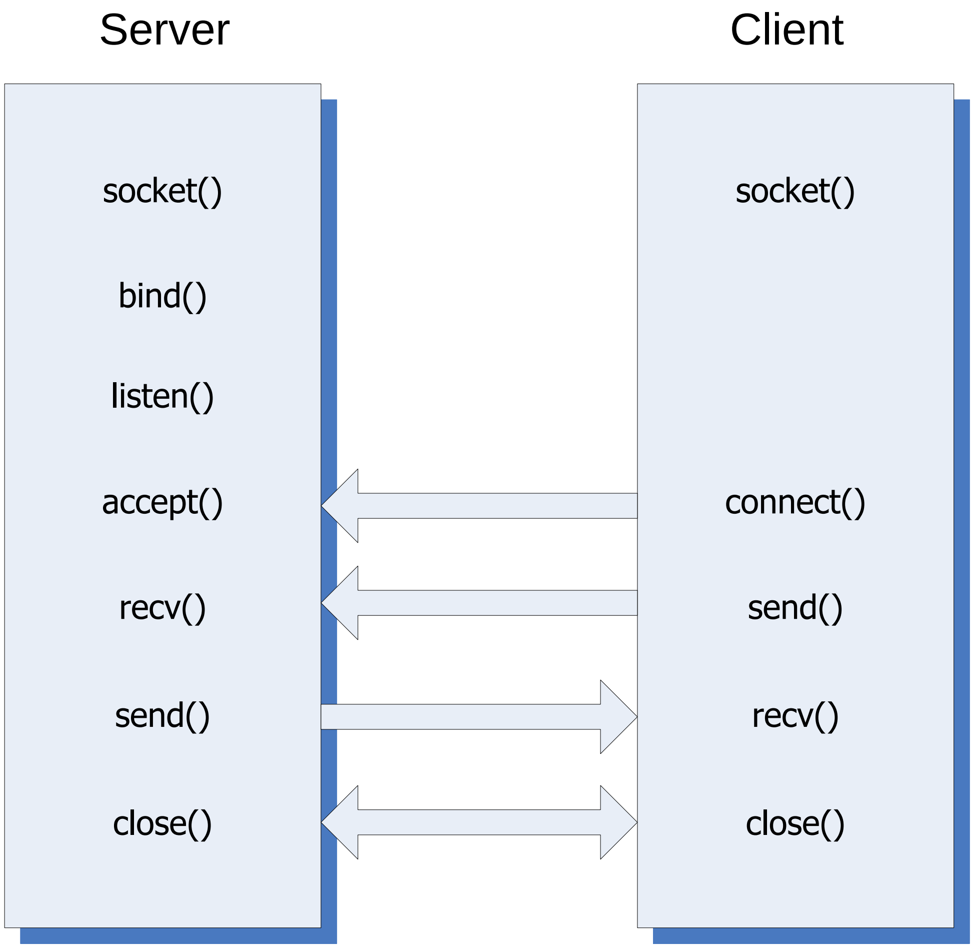
- All data sent will be received in the same order as sent.
  - Note that IP packets may arrive in a different order than that sent.
  - This occurs because all packets in a communication do not necessarily travel the same route between sender and receiver.

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

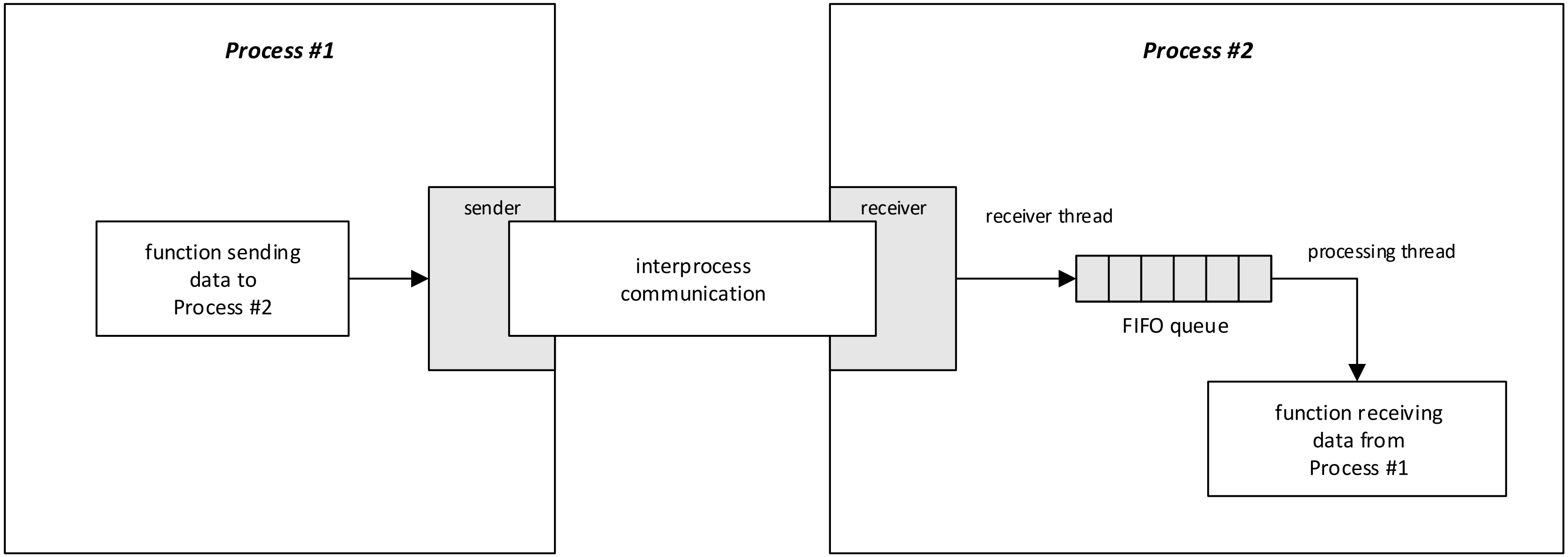
# Socket Logical Structure



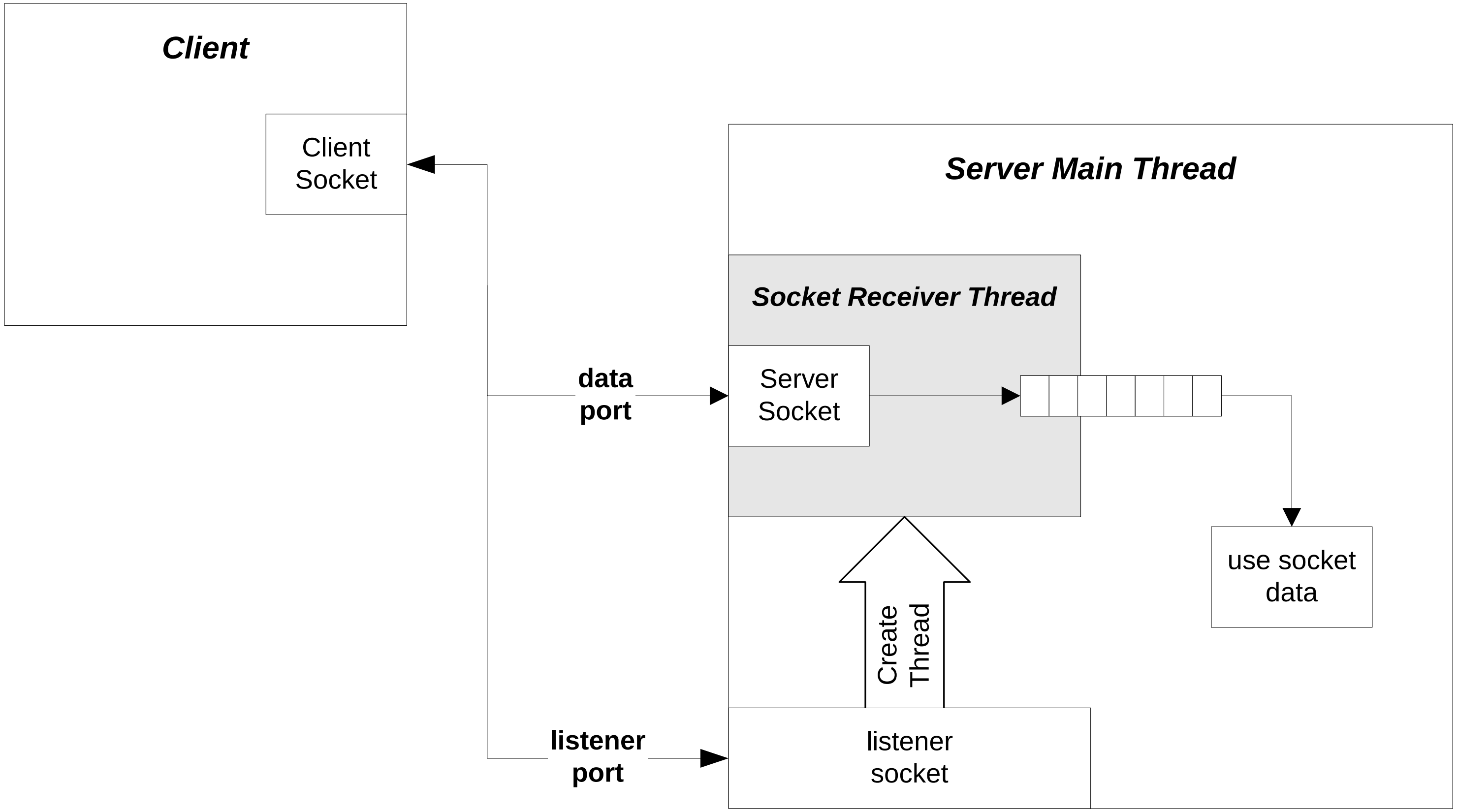
# Client / Server Processing



# Non-Blocking Communication


Process #1
Process #2
sender
receiver
function sending data to Process #2
interprocess communication
receiver thread
processing thread
FIFO queue
function receiving data from Process #1

# Client/Server Configuration



# 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

# 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.
    - Sometimes referred to as a “wire protocol”

# 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

# What we didn’t talk about

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

# 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:

```
int sockfd = 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:

```
int connect(sock, (sockaddr_in*)&addr,sizeof(addr))
```

# TCP/IP socket

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

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

- Bind listener socket to network card and port

# Listen for incoming requests

```
int listen(int 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.

# recv

```
int recv(
  int 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(
  int 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(int 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).

# Close socket

```
int close( s)
```

- Closes socket handle s. Called on the client signals server that connection

# TCP Addresses – IP4

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

# 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(
    int s,
    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

# The End