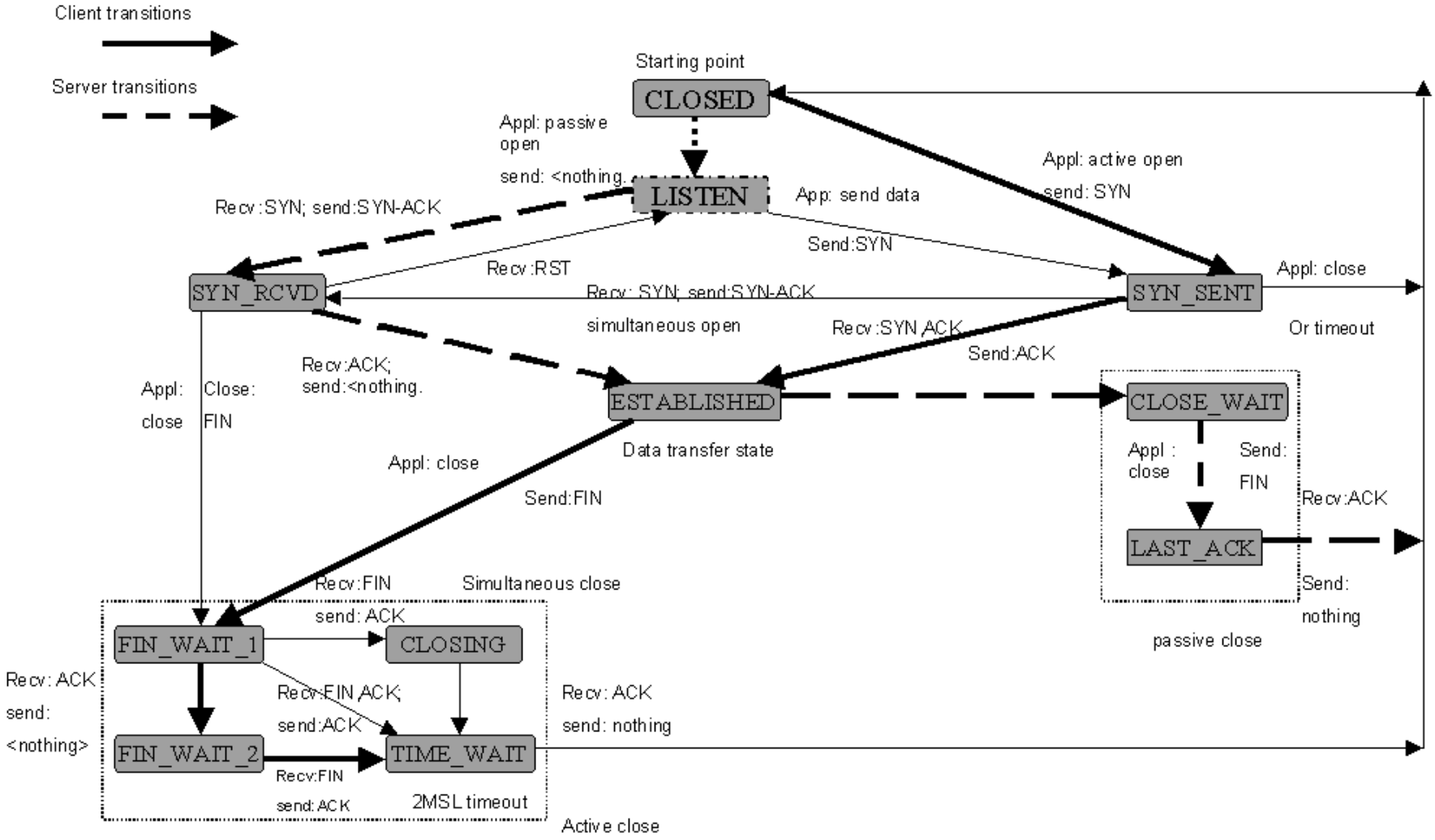
Project 3 - TCP

Original slides - Aditya Ganjam
Rampaged through by – Dave Eckhardt
Modified by – Vinay Chaudhary

What you will implement …

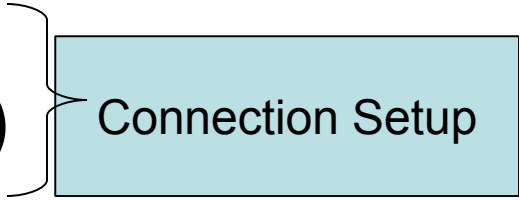
- TCP state machine (connection setup / teardown)
- Reliability
- In order-delivery
- Flow control

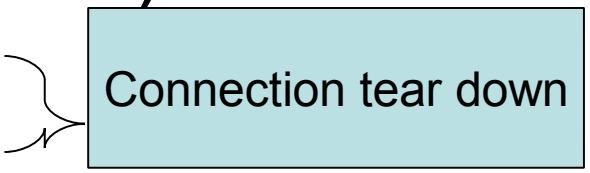
TCP State Machine (TCP/IP Illustrated vol. 1) W. Richard Stevens



The Functions

- tcp_socket (socket *)
- tcp_bind (socket *, src addr)
- tcp_connect(socket *, dest addr)
- tcp_accept(socket *, from addr)
- tcp_write(socket *, buf, buflen)
- tcp_read(socket *, buf, buflen)
- tcp_close(socket *)
- tcp_receive (pbuf, src, dest)– packet acceptor

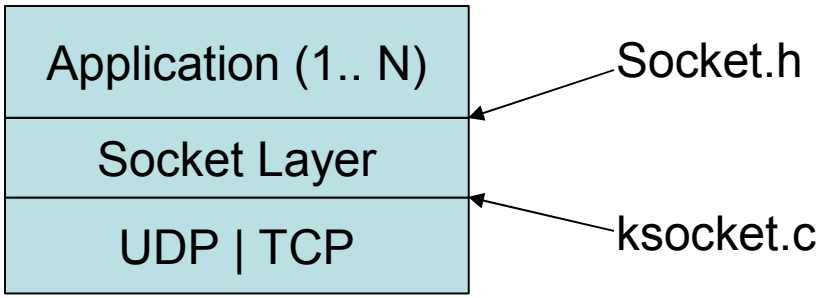


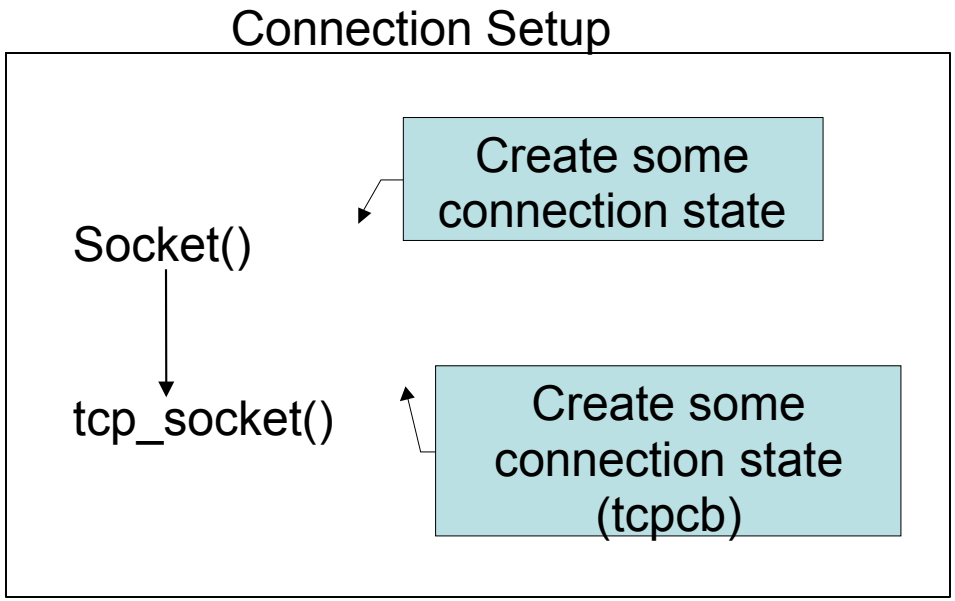


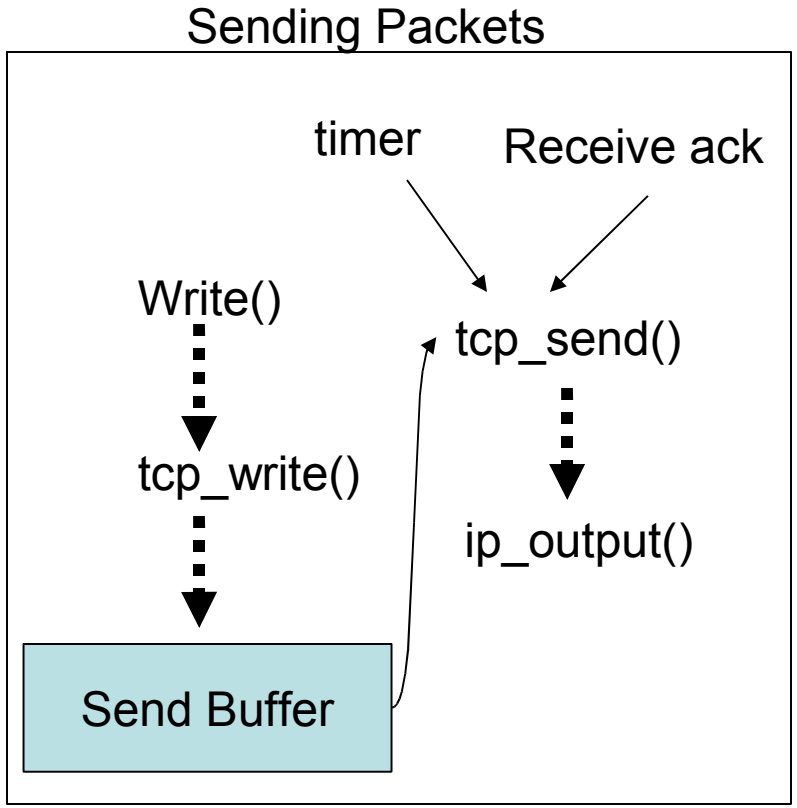
Timers (tcp_timer.c)

- Initial connect timer (Time to wait for established state after connect is called)
- Retransmit timer (If a packet hasn't been acked for this long, retransmit)
- Close timer (Time to wait between TIME_WAIT and closed to retransmit lost ACK
- timeout(timeout ftn, void *arg, int ticks);
  - Setup a timer
- untimeout(timeout ftn, void *arg);
  - Cancel a timer

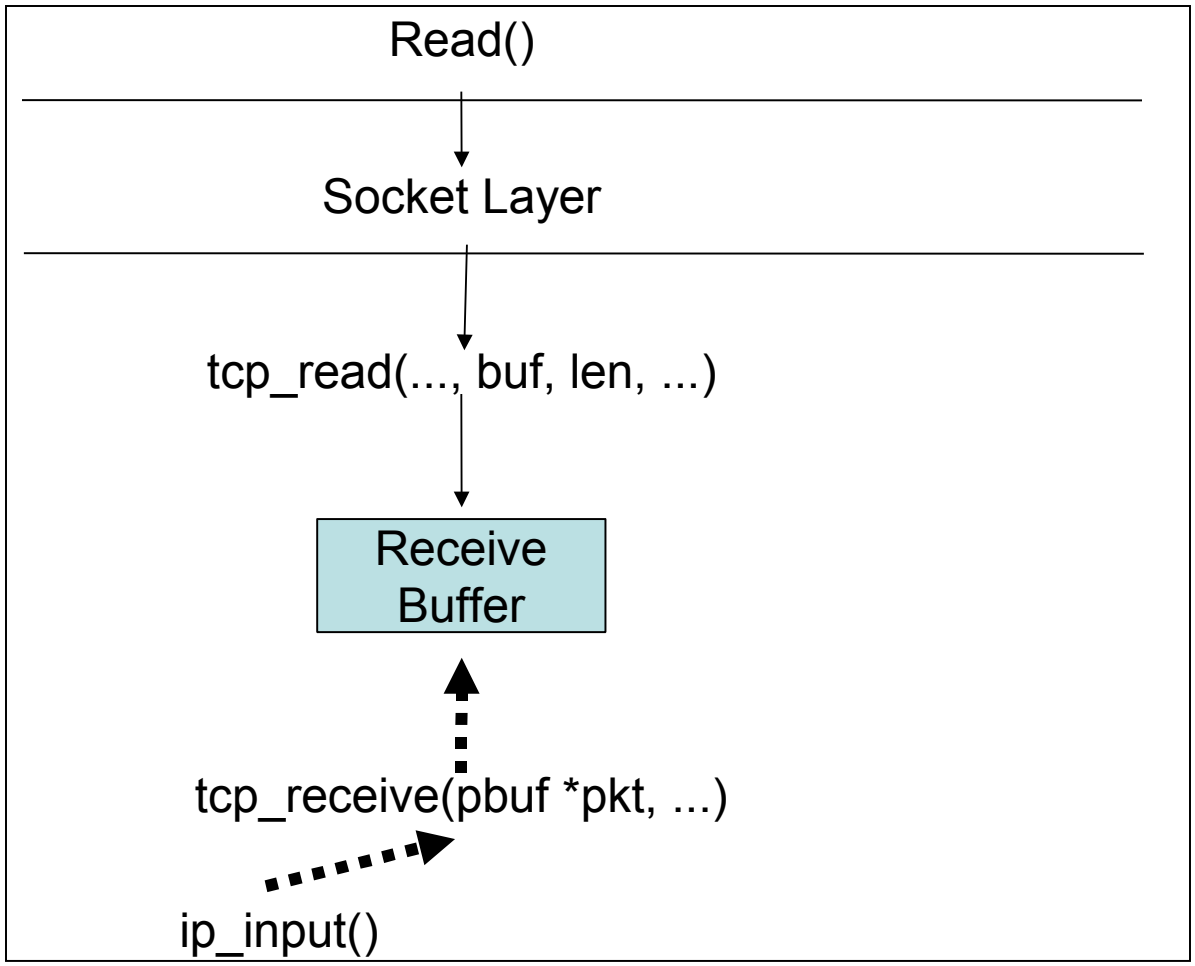
Interface with Socket Layer (Setup and Send)



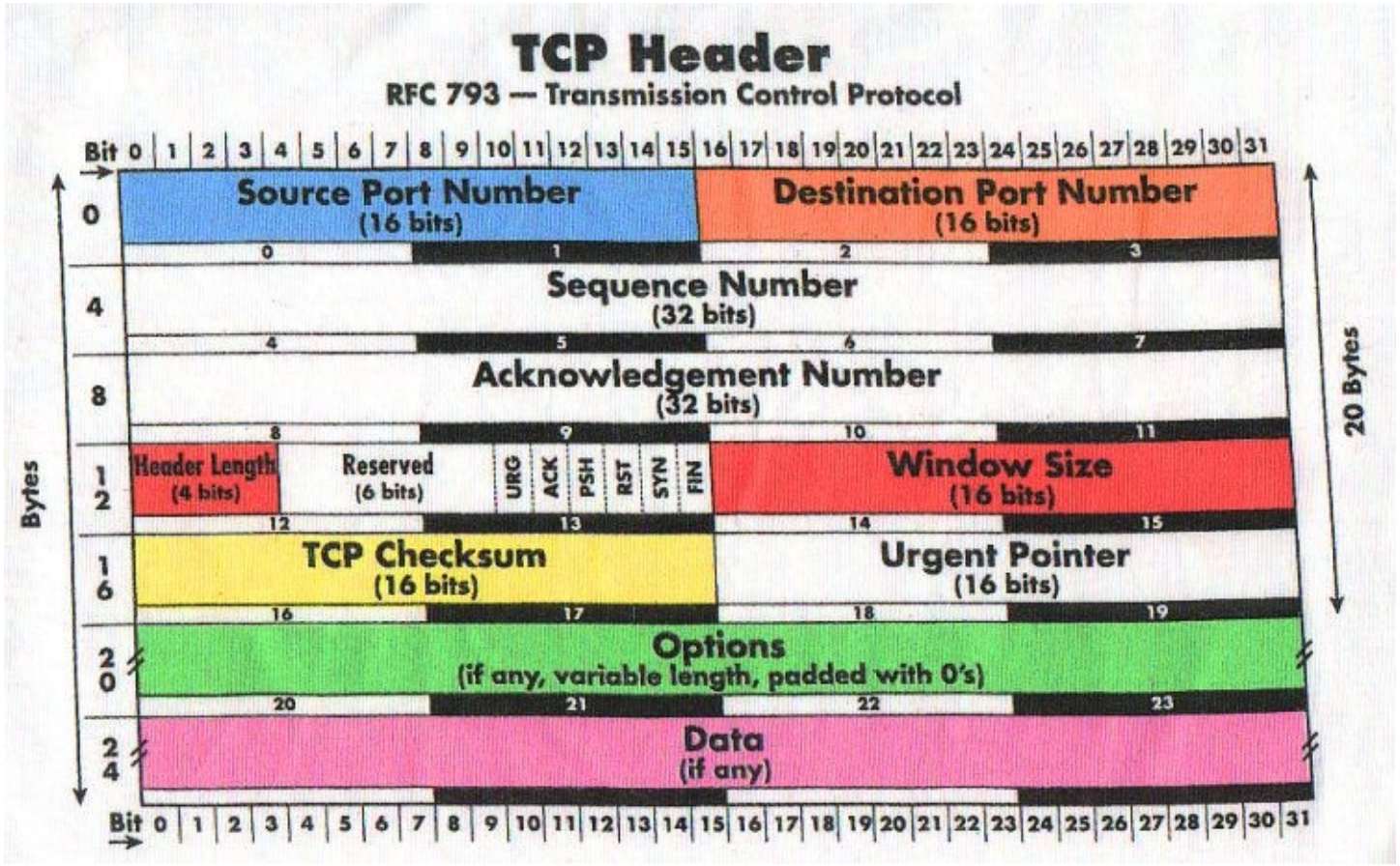




Interface with Socket Layer (Receive)



TCP-Header Structure



TCP Header

- Source port - This field identifies the sending port.
- Destination port -This field identifies the receiving port.
- Sequence number - The sequence number has a dual role. If the SYN flag is present then this is the initial sequence number. The first data packet will have this sequence number plus 1. Otherwise if the SYN flag is not present then the sequence number is the sequence number of the data in the packet.
- Acknowledgement number - If the ACK flag is set then the value of this field is the sequence number the sender expects next.
- Data offset - This 4-bit field specifies the size of the TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes. This field gets its name from the fact that it is also the offset from the start of the TCP packet to the data.

TCP-Header, More

- Reserved - 6-bit reserved field for future use and should be set to zero.
- Window - The number of packets the sender is willing to receive starting from the acknowledgement field value
- Checksum - The 16-bit checksum field is used for error-checking of the header and data.

TCP Flags

- Flags (aka Control bits) - 6 Bit flags
  - URG - Urgent pointer field is significant
  - ACK - Acknowledgement field is significant
  - PSH - Push function
  - RST - Reset the connection
  - SYN - Synchronize sequence numbers
  - FIN - No more data from sender
- Don't need to worry about URG,PSH,RST.
- Current TCP actually has 2 more bit flags, you don't care about them either
- ACK,SYN,FIN are very important

Flow Control – Advertised Window

- You need to make sure you don't send more data then the receiver can handle.
- Each ACK packet has valid Acknowledgment# and WindowSize fields.
- When sender gets an ACK, he determines how many more packets to send with the following equation:
  - NumberToSend = Window – (LastPacketSent – LastPacketAcked)
  - LastPackedAcked = Acknowledgment# -1

Flow Control – Sender Buffer

- Application may want to send data faster than sender can send.
- Have to prevent this. Limit the size of the send buffer
- LastPacketWritten – LastPacketAcked <= SendBufferSize.

TCP Control Block

- What might go in a TCP control block?
  - Src,Dest address
  - Src,Dest port
  - Pointer to corresponding socket
  - Mutexs and Condition variables
  - Send/Receive Queue (pbufs) – Includes out of order receive
  - Current State
  - Sequence Number of last packet sent/recv
  - Last received ack
  - Timers
  - ...

Synchronization Fundamentals

Two Fundamental operations

$\Longrightarrow$ Atomic instruction sequence

Voluntary de-scheduling

Atomic instruction sequence

- Problem domain
  - ***Short*** sequence of instructions
  - Nobody else may interleave same sequence
    - or a "related" sequence
  - “Typically” nobody is competing

Non-interference in P3

- What you've already seen
  - Can't queue two packets to a device at the same time
- Other issues
  - Can't allow two processes to bind port 99 at the same time
    - Would scramble your port $\Leftrightarrow$ socket data structure

Non-Interference – Observations

- Instruction sequences are “short”
  - Ok to force competitors to wait
- Probability of collision is "low"

Synchronization Fundamentals

Two Fundamental operations

Atomic instruction sequence

$\Longrightarrow$ Voluntary de-scheduling

Voluntary de-scheduling

- Anti-atomic
  - We *want* to be “interrupted”
- Making others wait is *wrong*
  - Wrong for them – we won't be ready for a while
  - Wrong for us – we can't be ready until *they* progress
- We don't ***want*** exclusion
- We ***want*** others to run - they ***enable*** us

Brief Mutual Exclusion

```
MUTEX_LOCK(sock->mutex);
sock->state = ...
MUTEX_UNLOCK(sock->mutex);
```

Note: sock refers to whatever structure is used to keep track of tcp connection. It is the tcp control block.

Signaling and Waiting

- `COND_WAIT(cond_t * cond, mutex_t * m)`
- `COND_SIGNAL(cond_t * cond)`
- `COND_WAIT()` will drop the mutex, wait until a `COND_SIGNAL()` is called on the condition variable, and then will try to reacquire the mutex. It will not return until the mutex is reacquired
- `COND_SIGNAL()` will signal an arbitrary thread waiting on the condition variable and cause it to wakeup and attempt to reacquire the mutex it is waiting for.

Voluntary Descheduling

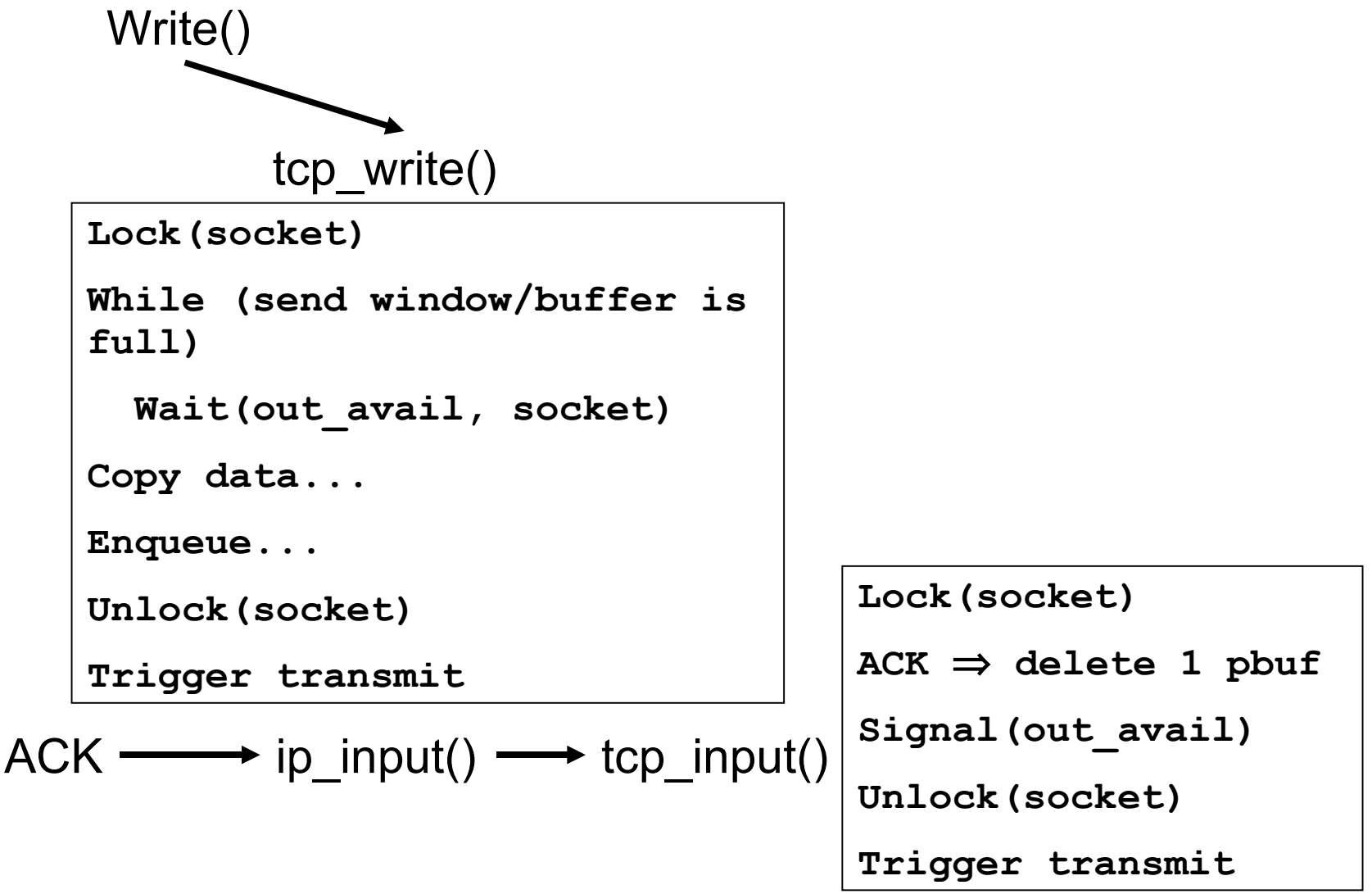
Thread 1:

```
MUTEX_LOCK(sock->mutex);
while (sock->state ...) {
  COND_WAIT(&sock->ready, &sock->mutex)
  }
sock->state = ...
MUTEX_UNLOCK(sock->mutex);
```

Thread 2:

```
MUTEX_LOCK(sock->mutex)
sock->state = ...
COND_SIGNAL(&sock->ready)
MUTEX_UNLOCK(sock->mutex);
```

Blocking Example



Warning: “Deadlock”

- A deadlock is...
  - A group of threads/processes...
  - Each one waiting for something...
  - Held by another one of the threads/processes
- How to get one
  - A: lock(tcp_socket_list); lock(tcp_socket_list[3]);
  - B: lock(tcp_socket_list[3]); lock(tcp_socket_list);
  - Now things get quiet for a while

Strategy

- Project handout includes suggested plan of attack
  - We really think it will help
- You probably haven't written code like this before
  - Asynchronous, state-machine, ...
- Please dive in early!

Questions?

- Questions?
- Examples you want worked out?