Project 3 - TCP

Original slides - Aditya Ganjam

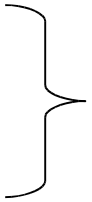
Rampaged through by – Dave Eckhardt

What you will implement …

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

The Functions

- tcp_socket
- tcp_bind
- tcp_connect
- tcp_accept
- tcp_write
- tcp_read
- tcp_close
- tcp_input – packet acceptor



Connection Setup

Connection tear down

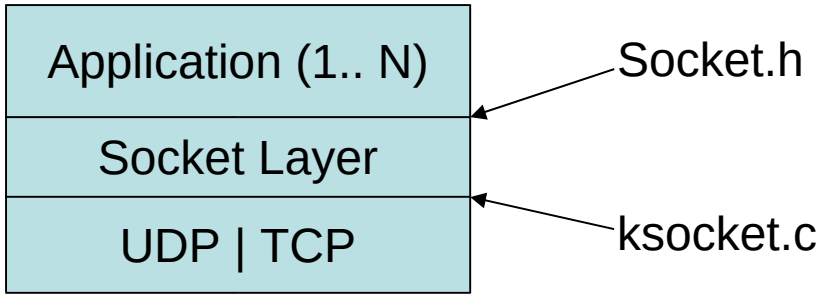
Timers (tcp_timer.c)

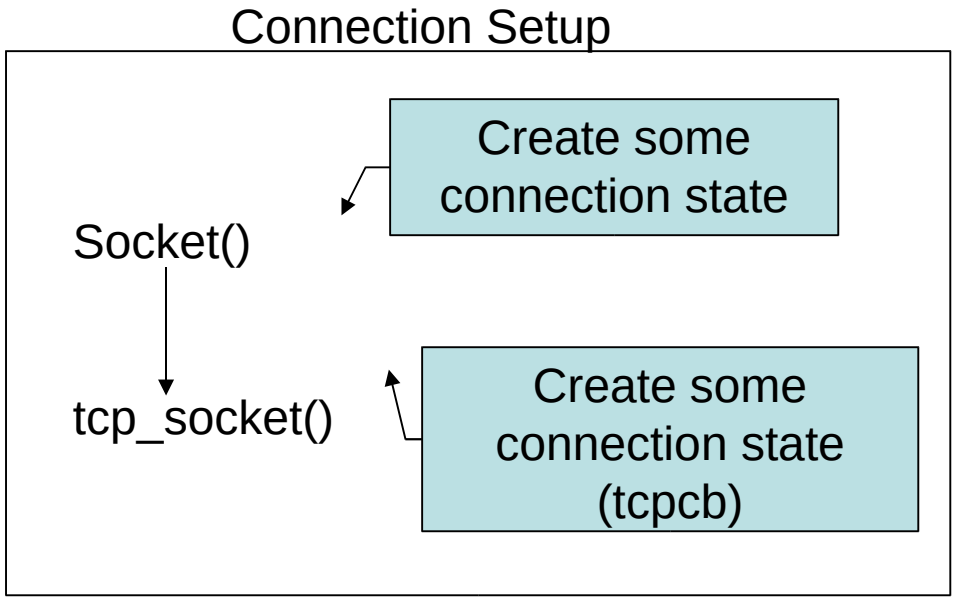
- Initial connect timer
- Retransmit timer
- Close timer

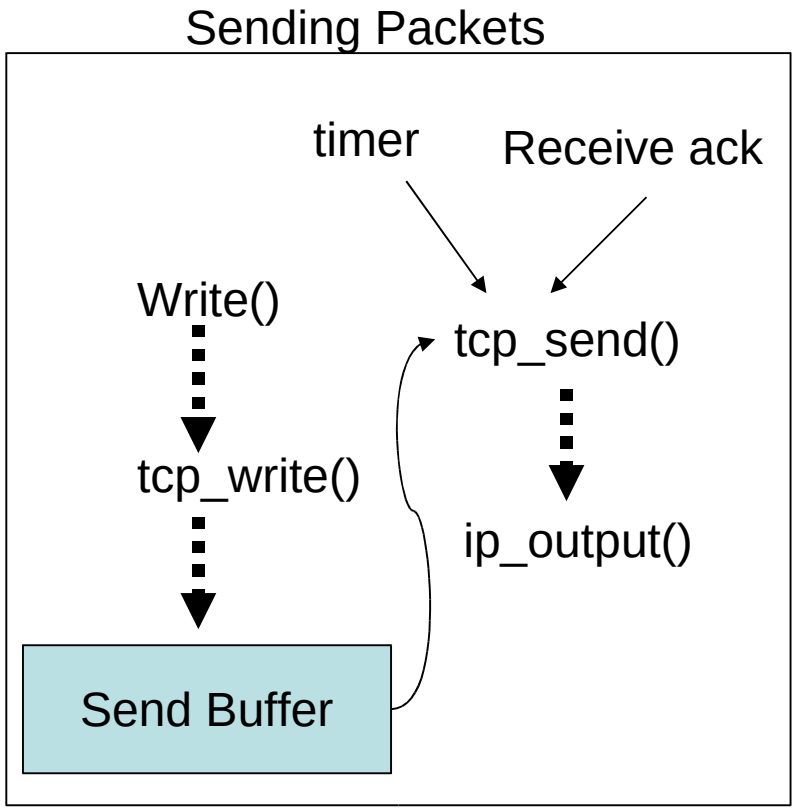
- timeout(timeout ftn, void *arg, int ticks);
  - Setup a timer

- untimeout(timeout ftn, void *arg);
  - Cancel a timer

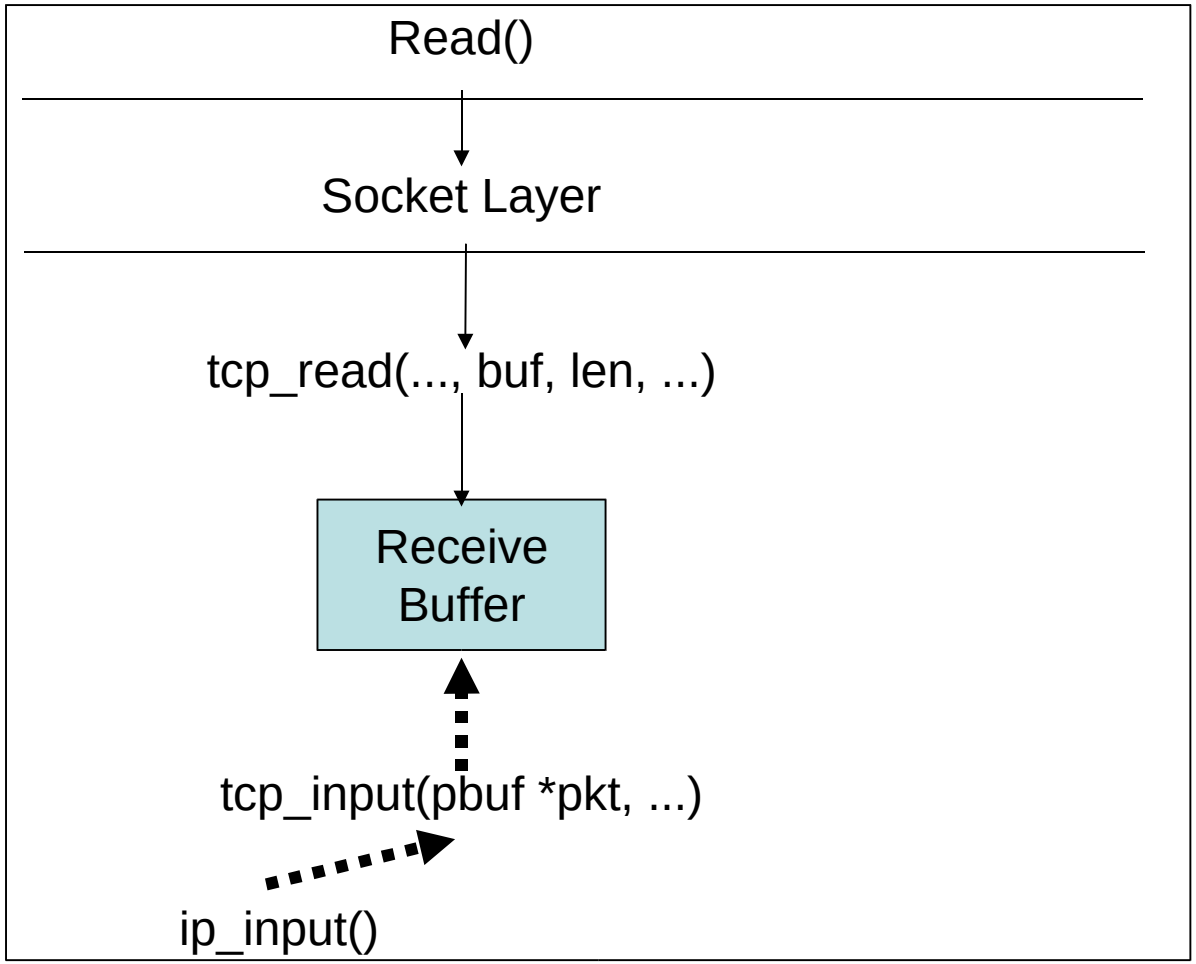
Interface with Socket Layer (Setup and Send)







Interface with Socket Layer (Receive)



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

Commerce

Customer 0	*Customer 1*
`cash = store->cash;`	`cash = store->cash;`
`cash += 50;`	`cash += 20;`
`wallet -= 50;`	`wallet -= 20;`
`store->cash = cash;`	`store->cash = cash;`

Should the store call the police?

Is deflation good for the economy?

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 ⇔ 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

$\Rightarrow$ Voluntary de-scheduling

Voluntary de-scheduling

- Problem domain
  - “Are we there yet?”
  - “Waiting for Godot”
- Example - "Sim City" disaster daemon

```
while (date < 1906-04-18) cwait(date);
while (hour < 5) cwait(hour);
for (i = 0; i < max_x; i++)
  for (j = 0; j < max_y; j++)
    wreak_havoc(i,j);
```

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

Voluntary de-scheduling

- Wait pattern

```
LOCK WORLD
while (!(ready = scan_world())){
  UNLOCK WORLD
  WAIT_FOR(progress_event)
}
```

- Your partner-competitor will

```
SIGNAL(progress_event)
```

Brief Mutual Exclusion

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

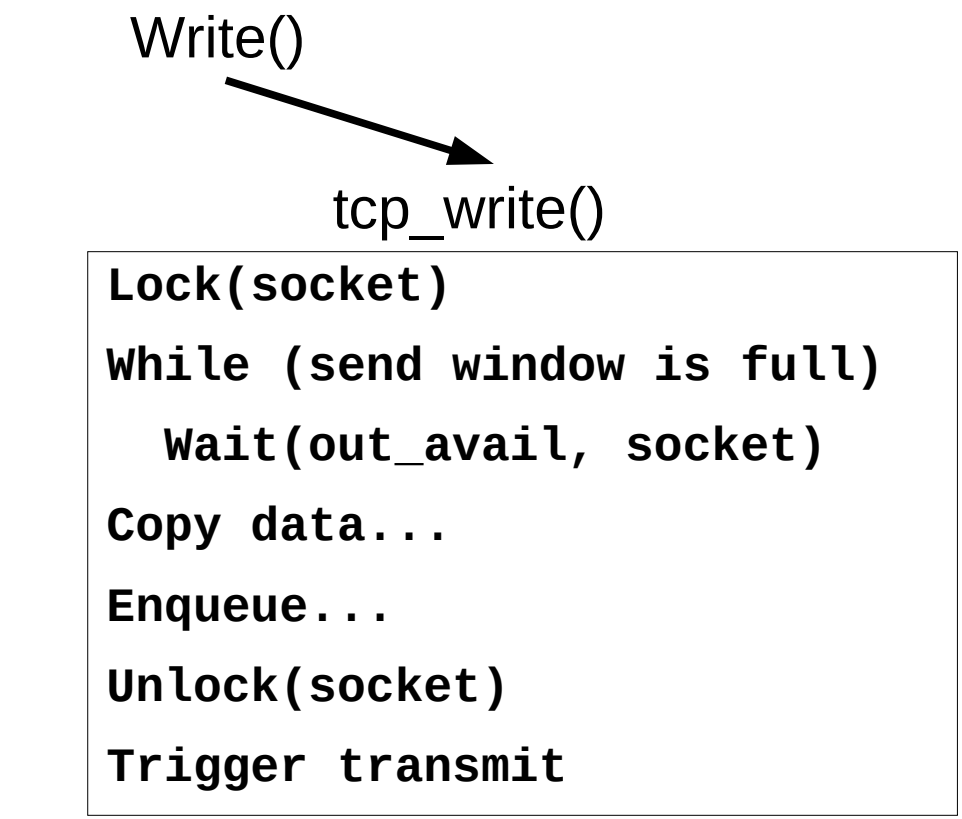
Blocking / Unblocking

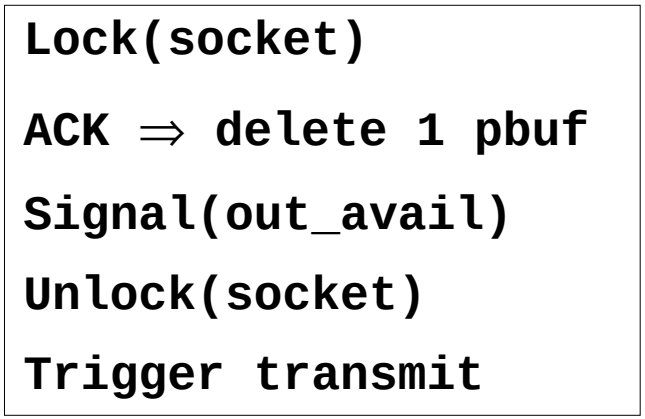
```
MUTEX_LOCK(sock->mutex);
while (sock->state ...) {
  COND_WAIT(&sock->ready, &sock->mutex)
  }
sock->state = ...

MUTEX_UNLOCK(sock->mutex);
```

- `COND_WAIT()` will drop the mutex, wait until a `COND_SIGNAL()` is called on the condition variable, and will re-lock the 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(socket_list); lock(socket_list[3]);
  - B: lock(socket_list[3]); lock(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!