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
Timers (tcp_timer.c)

- Initial connect timer
- Retransmit timer
- Close timer

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

`untimout(timeout ftn, void *arg);`
  - Cancel a timer
Interface with Socket Layer (Setup and Send)

Application (1.. N)
Socket Layer
UDP | TCP

Connection Setup
Socket()
tcp_socket()

Create some connection state
Create some connection state (tcpcb)

Send Buffer
Send Buffer

Sending Packets
timer
Receive ack
Write()
tcp_write()
tcp_send()
ip_output()
Interface with Socket Layer (Receive)

Read()

→

Socket Layer

→

tcp_read(..., buf, len, ...)

→

Receive Buffer

→

tcp_input(pbuf *pkt, ...)

→

ip_input()
Synchronization Fundamentals

Two Fundamental operations

⇒ 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

<table>
<thead>
<tr>
<th></th>
<th>Customer 0</th>
<th>Customer 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td><code>cash = store-&gt;cash;</code></td>
<td><code>cash = store-&gt;cash;</code></td>
</tr>
<tr>
<td></td>
<td><code>cash += 50;</code></td>
<td><code>cash += 20;</code></td>
</tr>
<tr>
<td>Wallet</td>
<td><code>wallet -= 50;</code></td>
<td><code>wallet -= 20;</code></td>
</tr>
<tr>
<td>Store Cash</td>
<td><code>store-&gt;cash = cash;</code></td>
<td><code>store-&gt;cash = cash;</code></td>
</tr>
</tbody>
</table>

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

⇒ Voluntary de-scheduling
Voluntary de-scheduling

• Problem domain
  – “Are we there yet?”
  – “Waiting for Godot”

• Example - "Sim City" disaster daemon
  
```c
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

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

• Your partner-competitor will

```c
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

Write()

Lock(socket)
While (send window is full)
  Wait(out_avail, socket)
Copy data...
Enqueue...
Unlock(socket)
Trigger transmit

ACK → ip_input() → tcp_input()

Lock(socket)
ACK ⇒ delete 1 pbuf
Signal(out_avail)
Unlock(socket)
Trigger transmit
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!