Use of IO::Select

use IO::Select;
$sel = IO::Select->new();
$sel->add(STDIN);
@handles = $sel->can_read();

see perldoc IO::Select

IO::Select and Sending/Receiving of Data

- Possible Solutions:
  1. `sysread()`/`syswrite()`, with selfwritten function to isolate lines.
  2. Switch to non-blocking I/O

Management of Multiple Connections

- Problem:
  - how to check if there is data ready to read on a connection?
  - how to additionally handle user input and incoming connections?
- Solution: IO::Select / select()

IO::Select and Sending/Receiving of Data

- IO::Select->can_read() looks for data in system buffers
- `<$handle>` reads data from system buffer into a PERL buffer
- Problem: can_read() cannot see data in PERL buffers
  - blocks with unread data in PERL buffer!

Non-Blocking I/O

- Enabling by:
  `$handle->blocking(0);`
- Effect:
  - `<$handle>` returns empty string if no complete line is available or end-of-file or error!
  - No implicit detection of connection end, has to be checked explicitly with `$handle->eof()`
Non-Blocking I/O

- Sending data:
  print $handle "protocol message\n";
- Wait for I/O event:
  @handles = $sel->can_read();
  foreach $h ( @handles ) { ... }
- Distinguishing between events:
  - by comparison with handles,
  - by checking for end-of-file (end of connection, connection died)

Sending data:
print $handle "protocol message\n";

Wait for I/O event:
@handles = $sel->can_read();
foreach $h ( @handles ) { ... }

Distinguishing between events:
- by comparison with handles,
- by checking for end-of-file (end of connection, connection died)

Non-Blocking I/O

Need loop for reading entire PERL buffer:
while($line=$handle->getline()) {
  # process line
}

Loop terminates when getline() cannot return entire line. Rest stays in PERL buffer, but that's OK.

# listen socket
$listen_sock = IO::Socket::INET->new();

# tastatur/STDIN
$stdin = IO::Handle->new();
$stdin->blocking(0);

# select
$sel = IO::Select->new();
$sel->add($listen_sock);
$sel->add($stdin);

Program-Kernel - Initialization

Program-Kernel - Dispatcher I

Main loop
while (defined @handles = $sel->can_read() ) {
  foreach $h ( @handles ) {
    # keyboard
    if($h == $stdin) { ... }
    # listen-Socket
    elsif($h == $listen_sock) {
      $new = accept(...);
      $sel->add($new);
    }
    else ...
  }
}

Program-Kernel - Dispatcher II

Peer-connection
else {
  # Test for end-of-file/error
  if(...) {
    close($h);
    $sel->remove($h)
    ...
  } else {
    # read message
    ...
    # process message
    ...
  }
}

P2P-Protokol, Version 0.1

- Uses TCP
- General message format
  - single lines
  - lines terminated by "\r\n"
  - Requests end with "P2P/0.1" (and "\r\n", of course!)
  - Replies start with "P2P/0.1" and a numeric reply code, e.g., "200"
P2P-Protokol, Version 0.1

- Node connected by direct TCP connection is ‘neighbour’
- Max. 4 active connections, unlimited accept()s
- Needs session setup and teardown handshake to exchange node IDs and neighbour lists
- Multiple connections between same two neighbours is not allowed!
- Never forward handshake messages
- Other messages for now forwarded by flooding

Broadcasting vs. Flooding

- Forwarding messages without knowledge of paths:
  - Broadcasting: forward to all neighbours
  - Flooding: Like broadcasting, but don’t forward into the direction, the message was received from (slight optimization).
P2P-Protokol, Version 0.1

- Request-Reply matching using message ID. Has together with node ID to be globally unique to detect message duplicates.
- Message have maximum travel distance of 3 hops using TTL field (decreased on receiving, forward if >0)

On receipt of a message:
- only HELLO is allowed!
- send reply message
- $state{$h} = 'up';

P2P-Protokol, Version 0.1

- Ending a session:
  - DISCONNECT P2P/0.1
  - P2P/0.1 210 GOODBYE

- After that, close the TCP connection and delete all session related information (e.g., routing paths, ...)

- new session in state 'new' after connect()/accept()
- On receipt of a message:
  - if($state{$h} eq 'new') {
  - $state{$h} = 'up';
  - }
  - elsif ($state{$h} eq 'hello_sent') {
  - $state{$h} = 'up';
  - }
  - elsif ($state{$h} eq 'up') {
  - $state{$h} = 'up';
  - $state{$h} = 'up';
  - }

- Adjust state also when sending messages, e.g., after sending a HELLO message.

P2P-Protokol, Version 0.1

- Special handshake to initialize protocol session in order to exchange node IDs:
  - Request: HELLO NODE-ID viper:2000 P2P/0.1
  - Reply: P2P/0.1 200 NODE-ID boa:3000

- Only after the handshake, other messages are allowed to be sent over a connection!
- Handshake messages are never forwarded to other nodes!

P2P-Protokol, Version 0.1

- Mapping of session setup and teardown onto a state machine
- Edges (state transitions):
  - <event> "/" <action>

Command Input

- Reading short commands from keyboard:
  - connect viper:2100
  - list connections
  - disconnect viper:2100