Asynchronous Events in Clojure

@stuartsierra

Clojure NYC
November 17, 2010

ØMQ

```
int main () {
    void *context = zmq_init (1);

    //  Socket to talk to clients
    void *responder = zmq_socket (context, ZMQ_SUB);
    zmq_bind (responder, "tcp://*:5555");

    while (1) {
        //  Wait for next request from client
        zmq_msg_t request;
        zmq_msg_init (&request);
        zmq_recv (responder, &request, 0);
        printf ("Received request: [%s]\n",
            (char *) zmq_msg_data (&request));
```

ØMQ

```
(defn start-receiver []
  (future
    (let [socket (.socket zmq-context ZMQ/SUB)]
      (.bind socket "tcp://*:5555")
      (loop []
        (let [message (.recv socket)]
          ;; do some work
          (recur))))))
```

Erlang

- Fast process creation/destruction
- Ability to support >> 10 000 concurrent processes with largely unchanged characteristics.
- Fast asynchronous message passing.
- Copying message-passing semantics (share-nothing concurrency).
- Process monitoring.
- Selective message reception.

-Ulf Wiger of Erlang Solutions, Ltd.
http://ulf.wiger.net/weblog/2008/02/06/what-is-erlang-style-concurrency/

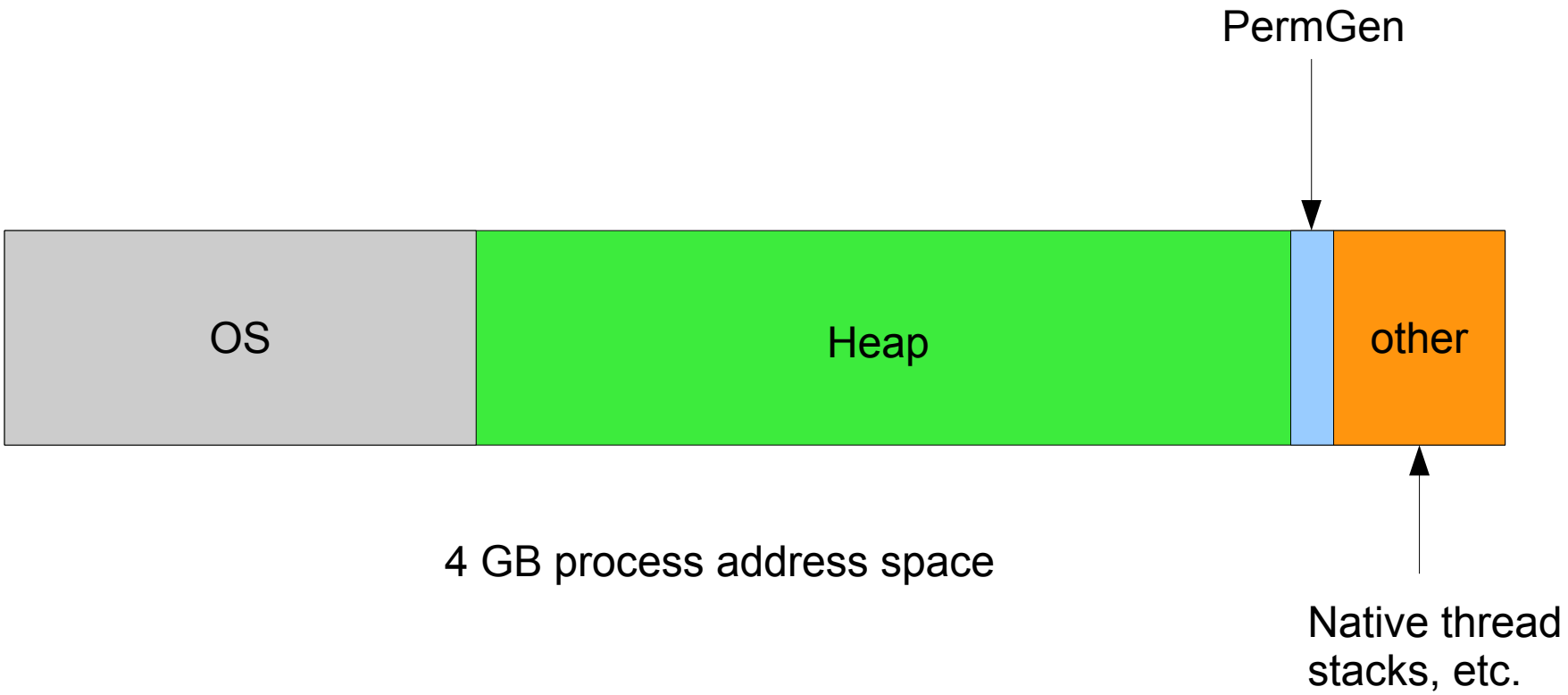
Erlang

```
loop(Users, N) ->
   receive
     {connect, Pid, User, Password} ->
       io:format("connection request from:~p ~p ~p~n",
                 [Pid, User, Password]),
       case member({User, Password}, Users) of
         true ->
           Max = max_connections(),
           if
             N > Max ->
               Pid ! {ftp_server,
                        {error, too_many_connections}},
               loop(Users, N);
             true ->
               New = spawn_link(?MODULE, handler, [Pid]),
               Pid ! {ftp_server, {ok, New}},
               loop(Users,  N + 1)
```

Actors in Clojure?

- Inbox: a LinkedBlockingQueue
  - Has an address of UUID@hostname:port
- Actor: a function looping in a Future
- Node: all actors / inboxes on one JVM
- server-listener actor listens on TCP port
- node-supervisor actor monitors other actors

32-bit JVM Memory



http://java-monitor.com/forum/showthread.php?t=570

Lamina

```
(defprotocol AlephChannel
  (listen- [ch fs])
  (receive-while- [ch callback-predicate-map])
  (receive- [ch fs])
  (receive-all- [ch fs])
  (cancel-callback- [ch fs])
  (enqueue- [ch msgs])
  (enqueue-and-close- [ch msgs])
  (on-zero-callbacks- [ch fs])
  (sealed? [ch]
"Returns true if no further messages can be enqueued.")
  (closed? [ch]
"Returns true if queue is sealed and there are no pending
messages."))
```

Lamina

```
(defn map*
  "Maps 'f' over all messages from 'ch'.  Returns a new
channel which is receive-only."
  [f ch]
  (fork (wrap-channel ch f)))

(defn filter* [f ch]
  "Filters all messages from 'ch'.  Returns a new channel
which is receive-only."
  (fork (wrap-channel ch #(if (f %) % ::ignore))))

(defn take*
  "Returns a receive-only channel which will contain the
first 'n' messages from 'ch'."
```

RX

```
public interface IObservable<out T> {
    IDisposable Subscribe(IObserver<T> observer);
}

public interface IObserver<in T> {
    void OnCompleted();
    void OnError(Exception error);
    void OnNext(T value);
}
```