Short introduction to ZeroMQ

Michal Hrušecký
Writing network application

Sockets

• low-level
• quite hard to use
• you have to do everything

AMQ

• message based with various patterns
• mainly used in Java world
• broker based
  ⇒ little complicated to get to started
ZeroMQ

- easy to use
- fast and asynchronous
- message based
- broker is not required
- plenty of ready made scenarios
- active community
- additional extra features
- not only network - ipc as well
- bindings for various languages
General recommendations

- distributed systems are hard
- having central component means having single point of failure
  - if you have broker, make it as simple as possible
  - better to know how to recover than to avoid inevitable
- use CZMQ - high-level API
  - other nice features - zactor, zauth, zbeacon
- use simplest pattern possible
Sockets example

```c
struct hostent *he;
struct sockaddr_in addr;

he = gethostbyname(hostname);
sockfd = socket(AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(addr.sin_zero), '\0', 8);

connect(sockfd, (struct sockaddr *)&addr, sizeof(struct sockaddr));

numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0);
buf[numbytes] = '\0';
close(sockfd);
```
ZeroMQ example

```c
void *context = zmq_ctx_new ();

// Socket to talk to server
void *requester = zmq_socket (context, ZMQ_REQ);
zmq_connect (requester, "tcp://localhost:5559");

for (int i = 0; i != 10; i++) {
    s_send (requester, "Hello");
    char *string = s_recv (requester);
    printf ("Received reply %d [%s]\n", i, string);
    free (string);
}

zmq_close (requester);
zmq_ctx_destroy (context);
```
Request-Reply

- simplest form of communication
- reliable
Publish-Subscribe

- fast
- asynchronous
- delivery not guaranteed
Divide and Conquer

- distributing work between workers
- based on PUSH/PULL
Useful stuff

- zactor - thread abstraction
- zpoller - handling multiple sockets
- zhash, zlist, zring
- zbroker
- zbeacon & zgossip
Going deeper - Zproto

- code generator for ZeroMQ
- xml based templates
- generates readable czmq code
- can be used to describe protocol
- can be used to describe state machine

⇒ Helps you get started much much faster!
<message name = "DIRECT">
    Client sends a message to a specific client
    <field name = "address" type = "string">
        Client identifier</field>
    <field name = "headers" type = "dictionary">
        Content header fields</field>
    <field name = "content" type = "msg">
        Content, as multipart message</field>
</message>
#define ZCCP_MSG_DIRECT 6

// Create a new zccp_msg
zccp_msg_t * zccp_msg_new (int id);

// Get/set the address field
const char * zccp_msg_address (zccp_msg_t *self);
void zccp_msg_set_address (zccp_msg_t *self,
                           const char *format, ...);

// Encode the DIRECT
zmsg_t * zccp_msg_encode_direct (const char *address,
                                 zhash_t *headers, zmsg_t *content);

// Send the DIRECT to the output in one step
int zccp_msg_send_direct (void *output,
                          const char *address, zhash_t *headers,
                          zmsg_t *content);
<state name = "connected" inherit = "external">
    <event name = "SUBSCRIBE">
        <action name = "store new subscription" />
        <action name = "send" message = "SUBSCRIBE OK" />
    </event>
</state>

<state name = "external">
    <event name = "*">
        <action name = "send" message = "INVALID" />
        <action name = "terminate" />
    </event>
    <event name = "expired">
        <action name = "terminate" />
    </event>
</state>
Zproto protocol - server c

• generated C file with server actor

```c
zactor_t *server = zactor_new(zccp_server, "server");
if(verbose)
    zstr_send(server, "VERBOSE");
zstr_sendx(server, "BIND", "ipc://@/server", NULL);
```

• generated stubs for unknown actions

```c
static void store_new_subscription(client_t *self) {
}
```
What next?

ZGuide:
http://zguide.zeromq.org/

Github:
https://github.com/zeromq/