

 [dysinger](#) / [nanomsg-examples](#)

Examples of different message types in C with nanomsg

 137 stars  46 forks Star Watch

Code

Issues 7

Pull requests 4

Actions

Projects

Wiki

Security

Insights

Join GitHub today

Dismiss

GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#) master ▾

dysinger ...

on 17 Sep 2013

[View code](#)

README.org

nanomsg

Pipeline

Code

```
#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nn.h>
#include <nanomsg/pipeline.h>

#define NODE0 "node0"
```

```

#define NODE1 "node1"

int node0 (const char *url)
{
    int sock = nn_socket (AF_SP, NN_PULL);
    assert (sock >= 0);
    assert (nn_bind (sock, url) >= 0);
    while (1)
    {
        char *buf = NULL;
        int bytes = nn_recv (sock, &buf, NN_MSG, 0);
        assert (bytes >= 0);
        printf ("NODE0: RECEIVED \"%s\"\n", buf);
        nn_freemsg (buf);
    }
}

int node1 (const char *url, const char *msg)
{
    int sz_msg = strlen (msg) + 1; // '\0' too
    int sock = nn_socket (AF_SP, NN_PUSH);
    assert (sock >= 0);
    assert (nn_connect (sock, url) >= 0);
    printf ("NODE1: SENDING \"%s\"\n", msg);
    int bytes = nn_send (sock, msg, sz_msg, 0);
    assert (bytes == sz_msg);
    return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (strncmp (NODE0, argv[1], strlen (NODE0)) == 0 && argc > 1)
        return node0 (argv[2]);
    else if (strncmp (NODE1, argv[1], strlen (NODE1)) == 0 && argc > 2)
        return node1 (argv[2], argv[3]);
    else
    {
        fprintf (stderr, "Usage: pipeline %s|%s <URL> <ARG> ...'\n",
            NODE0, NODE1);
        return 1;
    }
}

```

Compile

```
gcc pipeline.c /usr/local/lib/libnanomsg.a -o pipeline
```

Run

```
./pipeline node0 ipc:///tmp/pipeline.ipc & node0=$! && sleep 1
./pipeline node1 ipc:///tmp/pipeline.ipc "Hello, World!"
./pipeline node1 ipc:///tmp/pipeline.ipc "Goodbye."
kill $node0
```

Results

```
NODE1: SENDING "Hello, World!"
NODE0: RECEIVED "Hello, World!"
NODE1: SENDING "Goodbye."
NODE0: RECEIVED "Goodbye."
```

Request/Reply

Code

```
#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nn.h>
#include <nanomsg/reqrep.h>

#define NODE0 "node0"
#define NODE1 "node1"
#define DATE "DATE"

char *date ()
{
    time_t raw = time (&raw);
    struct tm *info = localtime (&raw);
    char *text = asctime (info);
    text[strlen(text)-1] = '\0'; // remove '\n'
    return text;
}

int node0 (const char *url)
{
    int sz_date = strlen (DATE) + 1; // '\0' too
    int sock = nn_socket (AF_SP, NN_REP);
    assert (sock >= 0);
    assert (nn_bind (sock, url) >= 0);
    while (1)
    {
        char *buf = NULL;
        int bytes = nn_recv (sock, &buf, NN_MSG, 0);
        assert (bytes >= 0);
        if (strncmp (DATE, buf, sz_date) == 0)
        {

```

```

    printf ("NODE0: RECEIVED DATE REQUEST\n");
    char *d = date();
    int sz_d = strlen(d) + 1; // '\0' too
    printf ("NODE0: SENDING DATE %s\n", d);
    bytes = nn_send (sock, d, sz_d, 0);
    assert (bytes == sz_d);
}
nn_freemsg (buf);
}
return nn_shutdown (sock, 0);
}

int node1 (const char *url)
{
    int sz_date = strlen(DATE) + 1; // '\0' too
    char *buf = NULL;
    int bytes = -1;
    int sock = nn_socket (AF_SP, NN_REQ);
    assert (sock >= 0);
    assert (nn_connect (sock, url) >= 0);
    printf ("NODE1: SENDING DATE REQUEST %s\n", DATE);
    bytes = nn_send (sock, DATE, sz_date, 0);
    assert (bytes == sz_date);
    bytes = nn_recv (sock, &buf, NN_MSG, 0);
    assert (bytes >= 0);
    printf ("NODE1: RECEIVED DATE %s\n", buf, bytes);
    nn_freemsg (buf);
    return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (strncmp (NODE0, argv[1], strlen (NODE0)) == 0 && argc > 1)
        return node0 (argv[2]);
    else if (strncmp (NODE1, argv[1], strlen (NODE1)) == 0 && argc > 1)
        return node1 (argv[2]);
    else
    {
        fprintf (stderr, "Usage: reqrep %s|%s <URL> <ARG> ...\n",
                NODE0, NODE1);
        return 1;
    }
}

```

Compile

```
gcc reqrep.c /usr/local/lib/libnanomsg.a -o reqrep
```

Run

```
./reqrep node0 ipc:///tmp/reqrep.ipc & node0=$! && sleep 1
./reqrep node1 ipc:///tmp/reqrep.ipc
kill $node0
```

Results

```
NODE1: SENDING DATE REQUEST DATE
NODE0: RECEIVED DATE REQUEST
NODE0: SENDING DATE Sat Sep  7 17:39:01 2013
NODE1: RECEIVED DATE Sat Sep  7 17:39:01 2013
```

Pair

Code

```
#include <assert.h>
#include <libc.h>
#include <nanomsg/nn.h>
#include <nanomsg/pair.h>
#include <stdio.h>

#define NODE0 "node0"
#define NODE1 "node1"

int send_name(int sock, const char *name)
{
    printf ("%s: SENDING \"%s\"\n", name, name);
    int sz_n = strlen (name) + 1; // '\0' too
    return nn_send (sock, name, sz_n, 0);
}

int recv_name(int sock, const char *name)
{
    char *buf = NULL;
    int result = nn_recv (sock, &buf, NN_MSG, 0);
    if (result > 0)
    {
        printf ("%s: RECEIVED \"%s\"\n", name, buf);
        nn_freemsg (buf);
    }
    return result;
}

int send_rcv(int sock, const char *name)
{
    int to = 100;
    assert (nn_setsockopt (sock, NN_SOL_SOCKET, NN_RCVTIMEO, &to, sizeof (to)) >= 0);
    while(1)
```

```

    {
        recv_name(sock, name);
        sleep(1);
        send_name(sock, name);
    }
}

int node0 (const char *url)
{
    int sock = nn_socket (AF_SP, NN_PAIR);
    assert (sock >= 0);
    assert (nn_bind (sock, url) >= 0);
    send_recv(sock, NODE0);
    return nn_shutdown (sock, 0);
}

int node1 (const char *url)
{
    int sock = nn_socket (AF_SP, NN_PAIR);
    assert (sock >= 0);
    assert (nn_connect (sock, url) >= 0);
    send_recv(sock, NODE1);
    return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (strncmp (NODE0, argv[1], strlen (NODE0)) == 0 && argc > 1)
        return node0 (argv[2]);
    else if (strncmp (NODE1, argv[1], strlen (NODE1)) == 0 && argc > 1)
        return node1 (argv[2]);
    else
    {
        fprintf (stderr, "Usage: pair %s|%s <URL> <ARG> ...\n",
                 NODE0, NODE1);
        return 1;
    }
}

```

Compile

```
gcc pair.c /usr/local/lib/libnanomsg.a -o pair
```

Run

```

./pair node0 ipc:///tmp/pair.ipc & node0=$!
./pair node1 ipc:///tmp/pair.ipc & node1=$!
sleep 3
kill $node0 $node1

```

Results

```

NODE1: SENDING HELLO
NODE0: RECEIVED HELLO
NODE0: SENDING HELLO
NODE1: RECEIVED HELLO
NODE1: SENDING HELLO
NODE0: RECEIVED HELLO
NODE0: SENDING HELLO
NODE1: RECEIVED HELLO

```

Pub/Sub

Code

```

#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nn.h>
#include <nanomsg/pubsub.h>

#define SERVER "server"
#define CLIENT "client"

char *date ()
{
    time_t raw = time (&raw);
    struct tm *info = localtime (&raw);
    char *text = asctime (info);
    text[strlen(text)-1] = '\\0'; // remove '\\n'
    return text;
}

int server (const char *url)
{
    int sock = nn_socket (AF_SP, NN_PUB);
    assert (sock >= 0);
    assert (nn_bind (sock, url) >= 0);
    while (1)
    {
        char *d = date();
        int sz_d = strlen(d) + 1; // '\\0' too
        printf ("SERVER: PUBLISHING DATE %s\\n", d);
        int bytes = nn_send (sock, d, sz_d, 0);
        assert (bytes == sz_d);
        sleep(1);
    }
    return nn_shutdown (sock, 0);
}

```

```

int client (const char *url, const char *name)
{
    int sock = nn_socket (AF_SP, NN_SUB);
    assert (sock >= 0);
    // TODO learn more about publishing/subscribe keys
    assert (nn_setsockopt (sock, NN_SUB, NN_SUB_SUBSCRIBE, "", 0) >= 0);
    assert (nn_connect (sock, url) >= 0);
    while (1)
    {
        char *buf = NULL;
        int bytes = nn_recv (sock, &buf, NN_MSG, 0);
        assert (bytes >= 0);
        printf ("CLIENT (%s): RECEIVED %s\n", name, buf);
        nn_freemsg (buf);
    }
    return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (strncmp (SERVER, argv[1], strlen (SERVER)) == 0 && argc >= 2)
        return server (argv[2]);
    else if (strncmp (CLIENT, argv[1], strlen (CLIENT)) == 0 && argc >= 3)
        return client (argv[2], argv[3]);
    else
    {
        fprintf (stderr, "Usage: pubsub %s|%s <URL> <ARG> ...\n",
                SERVER, CLIENT);
        return 1;
    }
}

```

Compile

```
gcc pubsub.c /usr/local/lib/libnanomsg.a -o pubsub
```

Run

```

./pubsub server ipc:///tmp/pubsub.ipc & server=$! && sleep 1
./pubsub client ipc:///tmp/pubsub.ipc client0 & client0=$!
./pubsub client ipc:///tmp/pubsub.ipc client1 & client1=$!
./pubsub client ipc:///tmp/pubsub.ipc client2 & client2=$!
sleep 5
kill $server $client0 $client1 $client2

```

Results


```

SERVER: PUBLISHING DATE Sat Sep  7 17:40:11 2013
SERVER: PUBLISHING DATE Sat Sep  7 17:40:12 2013
SERVER: PUBLISHING DATE Sat Sep  7 17:40:13 2013
CLIENT (client2): RECEIVED Sat Sep  7 17:40:13 2013
CLIENT (client0): RECEIVED Sat Sep  7 17:40:13 2013
CLIENT (client1): RECEIVED Sat Sep  7 17:40:13 2013
SERVER: PUBLISHING DATE Sat Sep  7 17:40:14 2013
CLIENT (client2): RECEIVED Sat Sep  7 17:40:14 2013
CLIENT (client1): RECEIVED Sat Sep  7 17:40:14 2013
CLIENT (client0): RECEIVED Sat Sep  7 17:40:14 2013
SERVER: PUBLISHING DATE Sat Sep  7 17:40:15 2013
CLIENT (client1): RECEIVED Sat Sep  7 17:40:15 2013
CLIENT (client2): RECEIVED Sat Sep  7 17:40:15 2013
CLIENT (client0): RECEIVED Sat Sep  7 17:40:15 2013
SERVER: PUBLISHING DATE Sat Sep  7 17:40:16 2013
CLIENT (client1): RECEIVED Sat Sep  7 17:40:16 2013
CLIENT (client2): RECEIVED Sat Sep  7 17:40:16 2013
CLIENT (client0): RECEIVED Sat Sep  7 17:40:16 2013

```

Survey

Code

```

#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nn.h>
#include <nanomsg/survey.h>

#define SERVER "server"
#define CLIENT "client"
#define DATE   "DATE"

char *date ()
{
    time_t raw = time (&raw);
    struct tm *info = localtime (&raw);
    char *text = asctime (info);
    text[strlen(text)-1] = '\0'; // remove '\n'
    return text;
}

int server (const char *url)
{
    int sock = nn_socket (AF_SP, NN_SURVEYOR);
    assert (sock >= 0);
    assert (nn_bind (sock, url) >= 0);
    sleep(1); // wait for connections
    int sz_d = strlen(DATE) + 1; // '\0' too
    printf ("SERVER: SENDING DATE SURVEY REQUEST\n");

```

```

int bytes = nn_send (sock, DATE, sz_d, 0);
assert (bytes == sz_d);
while (1)
{
    char *buf = NULL;
    int bytes = nn_recv (sock, &buf, NN_MSG, 0);
    if (bytes == ETIMEDOUT) break;
    if (bytes >= 0)
    {
        printf ("SERVER: RECEIVED \"%s\" SURVEY RESPONSE\n", buf);
        nn_freemsg (buf);
    }
}
return nn_shutdown (sock, 0);
}

int client (const char *url, const char *name)
{
    int sock = nn_socket (AF_SP, NN_RESPONDENT);
    assert (sock >= 0);
    assert (nn_connect (sock, url) >= 0);
    while (1)
    {
        char *buf = NULL;
        int bytes = nn_recv (sock, &buf, NN_MSG, 0);
        if (bytes >= 0)
        {
            printf ("CLIENT (%s): RECEIVED \"%s\" SURVEY REQUEST\n", name, buf);
            nn_freemsg (buf);
            char *d = date();
            int sz_d = strlen(d) + 1; // '\0' too
            printf ("CLIENT (%s): SENDING DATE SURVEY RESPONSE\n", name);
            int bytes = nn_send (sock, d, sz_d, 0);
            assert (bytes == sz_d);
        }
    }
    return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (strncmp (SERVER, argv[1], strlen (SERVER)) == 0 && argc >= 2)
        return server (argv[2]);
    else if (strncmp (CLIENT, argv[1], strlen (CLIENT)) == 0 && argc >= 3)
        return client (argv[2], argv[3]);
    else
    {
        fprintf (stderr, "Usage: survey %s|s <URL> <ARG> ... \n",
                SERVER, CLIENT);
        return 1;
    }
}

```

Compile

```
gcc survey.c /usr/local/lib/libnanomsg.a -o survey
```

Run

```
./survey server ipc:///tmp/survey.ipc & server=$!
./survey client ipc:///tmp/survey.ipc client0 & client0=$!
./survey client ipc:///tmp/survey.ipc client1 & client1=$!
./survey client ipc:///tmp/survey.ipc client2 & client2=$!
sleep 3
kill $server $client0 $client1 $client2
```

Results

```
SERVER: SENDING DATE SURVEY REQUEST
CLIENT (client1): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client2): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client0): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client0): SENDING DATE SURVEY RESPONSE
CLIENT (client1): SENDING DATE SURVEY RESPONSE
CLIENT (client2): SENDING DATE SURVEY RESPONSE
SERVER: RECEIVED "Sun Sep 15 13:39:46 2013" SURVEY RESPONSE
SERVER: RECEIVED "Sun Sep 15 13:39:46 2013" SURVEY RESPONSE
SERVER: RECEIVED "Sun Sep 15 13:39:46 2013" SURVEY RESPONSE
```

Bus

Code

```
#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nn.h>
#include <nanomsg/bus.h>

int node (const int argc, const char **argv)
{
    int sock = nn_socket (AF_SP, NN_BUS);
    assert (sock >= 0);
    assert (nn_bind (sock, argv[2]) >= 0);
    sleep (1); // wait for connections
    if (argc >= 3)
    {
        int x=3;
```

```

    for(x; x<argc; x++)
        assert (nn_connect (sock, argv[x]) >= 0);
}
sleep (1); // wait for connections
int to = 100;
assert (nn_setsockopt (sock, NN_SOL_SOCKET, NN_RCVTIMEO, &to, sizeof (to)) >= 0);
// SEND
int sz_n = strlen(argv[1]) + 1; // '\0' too
printf ("%s: SENDING '%s' ONTO BUS\n", argv[1], argv[1]);
int send = nn_send (sock, argv[1], sz_n, 0);
assert (send == sz_n);
while (1)
{
    // RECV
    char *buf = NULL;
    int recv = nn_recv (sock, &buf, NN_MSG, 0);
    if (recv >= 0)
    {
        printf ("%s: RECEIVED '%s' FROM BUS\n", argv[1], buf);
        nn_freemsg (buf);
    }
}
return nn_shutdown (sock, 0);
}

int main (const int argc, const char **argv)
{
    if (argc >= 3) node (argc, argv);
    else
    {
        fprintf (stderr, "Usage: bus <NODE_NAME> <URL> <URL> ...\n");
        return 1;
    }
}

```

Compile

```
gcc bus.c /usr/local/lib/libnanomsg.a -o bus
```

Run

```

gcc bus.c /usr/local/lib/libnanomsg.a -o bus
./bus node0 ipc:///tmp/node0.ipc ipc:///tmp/node1.ipc ipc:///tmp/node2.ipc & node0=$!
./bus node1 ipc:///tmp/node1.ipc ipc:///tmp/node2.ipc ipc:///tmp/node3.ipc & node1=$!
./bus node2 ipc:///tmp/node2.ipc ipc:///tmp/node3.ipc & node2=$!
./bus node3 ipc:///tmp/node3.ipc ipc:///tmp/node0.ipc & node3=$!
sleep 5
kill $node0 $node1 $node2 $node3

```

Results

```
node0: SENDING 'node0' ONTO BUS
node1: SENDING 'node1' ONTO BUS
node2: SENDING 'node2' ONTO BUS
node3: SENDING 'node3' ONTO BUS
node0: RECEIVED 'node1' FROM BUS
node0: RECEIVED 'node2' FROM BUS
node0: RECEIVED 'node3' FROM BUS
node1: RECEIVED 'node0' FROM BUS
node1: RECEIVED 'node2' FROM BUS
node1: RECEIVED 'node3' FROM BUS
node2: RECEIVED 'node0' FROM BUS
node2: RECEIVED 'node1' FROM BUS
node2: RECEIVED 'node3' FROM BUS
node3: RECEIVED 'node0' FROM BUS
node3: RECEIVED 'node1' FROM BUS
node3: RECEIVED 'node2' FROM BUS
```

Releases

No releases published