ARM和Linux下 nanomsg 编译与使用

原创 尘埃zza 2020-03-06 14:50:02 421 收藏 4 版权

分类专栏： C/C++ ARM Linux

简介

首页：https://nanomsg.org/index.html

nanomsg是一个套接字库，提供了几种常见的通信模式。它旨在使网络层快速，可扩展且易于使用。它以C语言实现，可在多种操作系统上运行，而无需进一步依赖。(该项目已在很大程度上被*nng*项目取代 。鼓励用户使用nng)

通信模式，也称为“可伸缩性协议”，是构建分布式系统的基本模块。通过组合它们，可以创建大量的分布式应用程序。

以下是C中每种模式类型的示例：

- Pipeline (A One-Way Pipe)
- Request/Reply (I ask, you answer)
- Pair (Two Way Radio)
- Pub/Sub (Topics & Broadcast)
- Survey (Everybody Votes)
- Bus (Routing)

支持以下传输机制：

- INPROC：在进程内传输（线程，模块等之间）
- IPC：在一台机器上的进程之间传输
- TCP：通过TCP进行网络传输
- WS：TCP上的WebSockets

安装cmake

```
sudo apt-get install cmake
```

Linux下编译

解压nanomsg源码文件，进入目录，修改CMakeLists.txt

```
...
# User-defined options.

option (NN_STATIC_LIB "Build static library instead of shared library." OFF)
option (NN_ENABLE_DOC "Enable building documentation." OFF)
option (NN_ENABLE_COVERAGE "Enable coverage reporting." OFF)
option (NN_ENABLE_GETADDRINFO_A "Enable/disable use of getaddrinfo_a in place of getaddrinfo." OFF)
option (NN_TESTS "Build and run nanomsg tests" OFF)
option (NN_TOOLS "Build nanomsg tools" OFF)
option (NN_ENABLE_NANOCAT "Enable building nanocat utility." ${NN_TOOLS})
set (NN_MAX_SOCKETS 512 CACHE STRING "max number of nanomsg sockets that can be created")

#  Platform checks.
...
```

只编译生成静态库或者动态库，其他模块关掉。

执行：

```
./configure
```

然后：

```
make
```

在当前目录下生成：

```
libnanomsg.so
libnanomsg.so.5
libnanomsg.so.5.1.0
```

用file命令查看：

```
libnanomsg.so.5.1.0: ELF 64-bit LSB  shared object, x86-64, version 1 (SYSV), dy
namically linked, BuildID[sha1]=be8533fe8a4f4425f1f6067412b7306ea42e979e, not st
ripped
```

ARM下编译

解压nanomsg源码文件，进入目录，修改CMakeLists.txt

```
cmake_minimum_required (VERSION 2.8.12)
#添加配置
SET(CMAKE_C_COMPILER   arm-none-linux-gnueabi-gcc)
SET(CMAKE_CXX_COMPILER arm-none-linux-gnueabi-g++)
#完成配置
project (nanomsg C)
...
# User-defined options.

option (NN_STATIC_LIB "Build static library instead of shared library." OFF)
option (NN_ENABLE_DOC "Enable building documentation." OFF)
option (NN_ENABLE_COVERAGE "Enable coverage reporting." OFF)
option (NN_ENABLE_GETADDRINFO_A "Enable/disable use of getaddrinfo_a in place of getaddrinfo." OFF)
option (NN_TESTS "Build and run nanomsg tests" OFF)
option (NN_TOOLS "Build nanomsg tools" OFF)
option (NN_ENABLE_NANOCAT "Enable building nanocat utility." ${NN_TOOLS})
set (NN_MAX_SOCKETS 512 CACHE STRING "max number of nanomsg sockets that can be created")

#  Platform checks.
...
```

添加指定编译器。

只编译生成静态库或者动态库，其他模块关掉。

执行：

```
./configure
```

然后：

```
make
```

在当前目录下生成：

```
libnanomsg.so
libnanomsg.so.5
libnanomsg.so.5.1.0
```

用file命令查看：

```
libnanomsg.so.5.1.0: ELF 32-bit LSB  shared object, ARM, EABI5 version 1 (SYSV),
 dynamically linked, not stripped
```

C API

官方API：https://nanomsg.org/v1.1.5/nanomsg.html

- 创建一个SP套接字

  nn_socket (3)

- 关闭一个SP套接字

  nn_close (3)

- 设置套接字选项

  nn_setsockopt (3)

- 检索套接字选项

  nn_getsockopt (3)

- 将本地端点添加到套接字

  nn_bind (3)

- 将远程端点添加到套接字

  nn_connect (3)

- 从套接字删除端点

  nn_shutdown (3)

- 发送消息

  nn_send (3)

- 收到讯息

  nn_recv (3)

- nn_send的细粒度替代

  nn_sendmsg (3)

- nn_recv的细粒度替代

  nn_recvmsg (3)

- 消息分配

  nn_allocmsg (3) nn_reallocmsg (3) nn_freemsg (3)

- 消息控制数据的处理

  nn_cmsg (3)

- 多路复用

  nn_poll (3)

- 检索当前的errno

  nn_errno (3)

- 将错误号转换为人类可读的字符串

  nn_strerror (3)

- 查询nanomsg符号的名称和值

  nn_symbol (3)

- nanomsg符号的查询属性

  nn_symbol_info (3)

- 查询套接字的统计信息

  nn_get_statistic (3)

- 启动设备

  nn_device (3)

- 通知所有套接字有关进程终止的信息

  nn_term (3)

- 影响nanomsg工作的环境变量

  nn_env (7)

nanomsg提供了以下可伸缩性协议：

- 一对一协议

  nn_pair (7)

- 请求/回复协议

  nn_reqrep (7)

- 发布/订阅协议

  nn_pubsub (7)

- 调查协议

  nn_survey (7)

- 管道协议

  nn_pipeline (7)

- 消息总线协议

  nn_bus (7)

nanomsg提供了以下传输机制：

- 流程中的运输

  nn_inproc (7)

- 进程间运输

  nn_ipc (7)

- TCP传输

  nn_tcp (7)

- WebSocket传输

  nn_ws (7)

库中安装了以下工具：

- nanocat

  nanocat(1)

Pipeline

管道（单向管道）

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-PUSha21i-1583477272542)(https://nanomsg.org/gettingstarted/pipeline.png)]

此模式对于解决生产者/消费者问题（包括负载平衡）很有用。消息从推送侧流向推送侧。如果连接了多个对等方，则该模式将尝试公平分配。

pipeline.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <nanomsg/nn.h>
#include <nanomsg/pipeline.h>

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

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
        exit(1);
}

int
node0(const char *url)
{
        int sock;
        int rv;

        if ((sock = nn_socket(AF_SP, NN_PULL)) < 0) {
                fatal("nn_socket");
        }
        if ((rv = nn_bind(sock, url)) < 0) {
                fatal("nn_bind");
        }
        for (;;) {
                char *buf = NULL;
                int bytes;
                if ((bytes = nn_recv(sock, &buf, NN_MSG, 0)) < 0) {
                        fatal("nn_recv");
                }
                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;
        int rv;
        int bytes;

        if ((sock = nn_socket(AF_SP, NN_PUSH)) < 0) {
                fatal("nn_socket");
        }
        if ((rv = nn_connect(sock, url)) < 0) {
                fatal("nn_connect");
        }
        printf("NODE1: SENDING \"%s\"\n", msg);
        if ((bytes = nn_send(sock, msg, sz_msg, 0)) < 0) {
                fatal("nn_send");
        }
        sleep(1); // wait for messages to flush before shutting down
        return (nn_shutdown(sock, 0));
}

int
main(const int argc, const char **argv)
{
        if ((argc > 1) && (strcmp(NODE0, argv[1]) == 0))
                return (node0(argv[2]));

        if ((argc > 2) && (strcmp(NODE1, argv[1]) == 0))
                return (node1(argv[2], argv[3]));

        fprintf(stderr, "Usage: pipeline %s|%s <URL> <ARG> ...'\n",
                NODE0, NODE1);
        return (1);
}
```

编译：

```
gcc pipeline.c -lnanomsg -o pipeline
```

执行：

```
./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
```

输出：

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

Request/Reply

请求/答复（我问，你回答）

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-z4HvgrMR-1583477272543)(https://nanomsg.org/gettingstarted/reqrep.png)]

请求/答复用于同步通信，其中每个问题都用一个答案来回答，例如远程过程调用（RPC）。像Pipeline一样，它也可以执行负载平衡。这是套件中唯一可信赖的消息传递模式，因为如果请求与响应不匹配，它将自动重试。

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <nanomsg/nn.h>
#include <nanomsg/reqrep.h>

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

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
        exit(1);
}

char *
date(void)
{
        time_t now = time(&now);
        struct tm *info = localtime(&now);
        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;
        int rv;

        if ((sock = nn_socket(AF_SP, NN_REP)) < 0) {
                fatal("nn_socket");
        }
          if ((rv = nn_bind(sock, url)) < 0) {
                fatal("nn_bind");
        }
        for (;;) {
                char *buf = NULL;
                int bytes;
                if ((bytes = nn_recv(sock, &buf, NN_MSG, 0)) < 0) {
                        fatal("nn_recv");
                }
                if ((bytes == (strlen(DATE) + 1)) && (strcmp(DATE, buf) == 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);
                        if ((bytes = nn_send(sock, d, sz_d, 0)) < 0) {
                                fatal("nn_send");
                        }
                }
                nn_freemsg(buf);
        }
}

int
node1(const char *url)
{
        int sz_date = strlen(DATE) + 1; // '\0' too
        char *buf = NULL;
        int bytes = -1;
        int sock;
        int rv;

        if ((sock = nn_socket(AF_SP, NN_REQ)) < 0) {
                fatal("nn_socket");
        }
        if ((rv = nn_connect (sock, url)) < 0) {
                fatal("nn_connect");
        }
        printf("NODE1: SENDING DATE REQUEST %s\n", DATE);
        if ((bytes = nn_send(sock, DATE, sz_date, 0)) < 0) {
                fatal("nn_send");
        }
        if ((bytes = nn_recv(sock, &buf, NN_MSG, 0)) < 0) {
                fatal("nn_recv");
        }
        printf("NODE1: RECEIVED DATE %s\n", buf);
        nn_freemsg(buf);
        return (nn_shutdown(sock, 0));
}

int
main(const int argc, const char **argv)
{
        if ((argc > 1) && (strcmp(NODE0, argv[1]) == 0))
                return (node0(argv[2]));

        if ((argc > 1) && (strcmp(NODE1, argv[1]) == 0))
                return (node1(argv[2]));

      fprintf(stderr, "Usage: reqrep %s|%s <URL> ...\n", NODE0, NODE1);
      return (1);
}
```

编译：

```
gcc reqrep.c -lnanomsg -o reqrep
```

执行：

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

输出：

```
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

配对（双向广播）

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Bvrj9AV7-1583477272543)(https://nanomsg.org/gettingstarted/pair.png)]

当存在一对一的对等关系时，将使用配对模式。一次只能将一个对等方连接到另一个对等方，但是两者都可以自由发言。

pair.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <nanomsg/nn.h>
#include <nanomsg/pair.h>

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

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
        exit(1);
}

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_recv(int sock, const char *name)
{
        int to = 100;
        if (nn_setsockopt(sock, NN_SOL_SOCKET, NN_RCVTIMEO, &to,
            sizeof (to)) < 0) {
                fatal("nn_setsockopt");
        }

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

int
node0(const char *url)
{
        int sock;
        if ((sock = nn_socket(AF_SP, NN_PAIR)) < 0) {
                fatal("nn_socket");
        }
         if (nn_bind(sock, url) < 0) {
                fatal("nn_bind");
        }
        return (send_recv(sock, NODE0));
}

int
node1(const char *url)
{
        int sock;
        if ((sock = nn_socket(AF_SP, NN_PAIR)) < 0) {
                fatal("nn_socket");
        }
        if (nn_connect(sock, url) < 0) {
                fatal("nn_connect");
        }
        return (send_recv(sock, NODE1));
}

int
main(const int argc, const char **argv)
{
        if ((argc > 1) && (strcmp(NODE0, argv[1]) == 0))
                return (node0(argv[2]));

        if ((argc > 1) && (strcmp(NODE1, argv[1]) == 0))
                return (node1(argv[2]));

        fprintf(stderr, "Usage: pair %s|%s <URL> <ARG> ...\n", NODE0, NODE1);
        return 1;
}
```

编译：

```
gcc pair.c -lnanomsg -o pair
```

执行：

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

输出：

```
node0: SENDING "node0"
node1: SENDING "node1"
node1: RECEIVED"node0"
node0: SENDING "node0"
node0: RECEIVED"node1"
```

Pub/Sub

发布/订阅（主题和广播）

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-qtiS1suw-1583477272544)(https://nanomsg.org/gettingstarted/pubsub.png)]

此模式用于允许单个广播者将消息发布给许多订阅者，订阅者可以选择限制它们接收的消息。

pubsub.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include <nanomsg/nn.h>
#include <nanomsg/pubsub.h>

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

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
}

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

int
server(const char *url)
{
        int sock;

        if ((sock = nn_socket(AF_SP, NN_PUB)) < 0) {
                fatal("nn_socket");
        }
          if (nn_bind(sock, url) < 0) {
                fatal("nn_bind");
        }
        for (;;) {
                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);
                if (bytes < 0) {
                        fatal("nn_send");
                }
                sleep(1);
        }
}

int
client(const char *url, const char *name)
{
        int sock;

        if ((sock = nn_socket(AF_SP, NN_SUB)) < 0) {
                fatal("nn_socket");
        }

        // subscribe to everything ("" means all topics)
        if (nn_setsockopt(sock, NN_SUB, NN_SUB_SUBSCRIBE, "", 0) < 0) {
                fatal("nn_setsockopt");
        }
        if (nn_connect(sock, url) < 0) {
                fatal("nn_connet");
        }
        for (;;) {
                char *buf = NULL;
                int bytes = nn_recv(sock, &buf, NN_MSG, 0);
                if (bytes < 0) {
                        fatal("nn_recv");
                }
                printf("CLIENT (%s): RECEIVED %s\n", name, buf);
                nn_freemsg(buf);
        }
}

int
main(const int argc, const char **argv)
{
        if ((argc >= 2) && (strcmp(SERVER, argv[1]) == 0))
                return (server(argv[2]));

          if ((argc >= 3) && (strcmp(CLIENT, argv[1]) == 0))
                return (client (argv[2], argv[3]));

        fprintf(stderr, "Usage: pubsub %s|%s <URL> <ARG> ...\n",
            SERVER, CLIENT);
        return 1;
}
```

编译：

```
gcc pubsub.c -lnanomsg -o pubsub
```

执行：

```
./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
```

输出：

```
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

调查（所有人投票）

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-kpqMDG6J-1583477272544)(https://nanomsg.org/gettingstarted/survey.png)]

用于发送定时调查，调查被单独返回直到调查到期。此模式对于服务发现和投票算法很有用。

survey.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include <nanomsg/nn.h>
#include <nanomsg/survey.h>

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

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
        exit(1);
}

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

int
server(const char *url)
{
        int sock;

        if ((sock = nn_socket (AF_SP, NN_SURVEYOR)) < 0) {
                fatal("nn_socket");
        }
        if (nn_bind(sock, url)  < 0) {
                fatal("nn_bind");
        }
        for (;;) {
                printf("SERVER: SENDING DATE SURVEY REQUEST\n");
                int bytes = nn_send(sock, DATE, strlen(DATE) + 1, 0);
                if (bytes < 0) {
                        fatal("nn_send");
                }

                for (;;) {
                        char *buf = NULL;
                        int bytes = nn_recv(sock, &buf, NN_MSG, 0);
                        if (bytes < 0) {
                                if (nn_errno() == ETIMEDOUT) {
                                        break;
                                }
                                fatal("nn_recv");
                        }
                        printf("SERVER: RECEIVED \"%s\" SURVEY RESPONSE\n",
                            buf);
                        nn_freemsg(buf);
                }

                printf("SERVER: SURVEY COMPLETE\n");
                sleep(1); // Start another survey in a second
        }
}

int
client(const char *url, const char *name)
{
        int sock;

        if ((sock = nn_socket(AF_SP, NN_RESPONDENT)) < 0) {
                fatal("nn_socket");
        }
        if (nn_connect (sock, url) < 0) {
                fatal("nn_connect");
        }
            for (;;) {
                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);
                        if (nn_send(sock, d, sz_d, 0) < 0) {
                                fatal("nn_send");
                        }
                }
        }
}

int
main(const int argc, const char **argv)
{
        if ((argc >= 2) && (strcmp(SERVER, argv[1]) == 0))
                return (server(argv[2]));

        if ((argc >= 3) && (strcmp(CLIENT, argv[1]) == 0))
                return (client(argv[2], argv[3]));

        fprintf(stderr, "Usage: survey %s|%s <URL> <ARG> ...\n",
            SERVER, CLIENT);
        return 1;
}
```

编译：

```
gcc survey.c -lnanomsg -o survey
```

执行：

```
./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 4
kill $server $client0 $client1 $client2
```

输出：

```
SERVER: SENDING DATE SURVEY REQUEST
SERVER: SURVEY COMPLETE
SERVER: SENDING DATE SURVEY REQUEST
CLIENT (client2): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client1): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client0): RECEIVED "DATE" SURVEY REQUEST
CLIENT (client1): SENDING DATE SURVEY RESPONSE
CLIENT (client0): SENDING DATE SURVEY RESPONSE
CLIENT (client2): SENDING DATE SURVEY RESPONSE
SERVER: RECEIVED "Mon Jan  8 13:10:43 2018" SURVEY RESPONSE
SERVER: RECEIVED "Mon Jan  8 13:10:43 2018" SURVEY RESPONSE
SERVER: RECEIVED "Mon Jan  8 13:10:43 2018" SURVEY RESPONSE
SERVER: SURVEY COMPLETE
```

Bus

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-0eYuYynI-1583477272545)(https://nanomsg.org/gettingstarted/bus.png)]

总线协议对于路由应用程序或构建完全互连的网状网络很有用。在这种模式下，消息被发送到每个直接连接的对等方。

bus.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <nanomsg/nn.h>
#include <nanomsg/bus.h>

void
fatal(const char *func)
{
        fprintf(stderr, "%s: %s\n", func, nn_strerror(nn_errno()));
        exit(1);
}

int
node(const int argc, const char **argv)
{
        int sock;

        if ((sock = nn_socket (AF_SP, NN_BUS)) < 0) {
                fatal("nn_socket");
        }
          if (nn_bind(sock, argv[2]) < 0) {
                fatal("nn_bind");
        }

        sleep(1); // wait for peers to bind
        if (argc >= 3) {
                for (int x = 3; x < argc; x++) {
                        if (nn_connect(sock, argv[x]) < 0) {
                                fatal("nn_connect");
                        }
                }
        }
        sleep(1); // wait for connections
        int to = 100;
        if (nn_setsockopt(sock, NN_SOL_SOCKET, NN_RCVTIMEO, &to,
            sizeof (to)) < 0) {
                fatal("nn_setsockopt");
        }

        // SEND
        int sz_n = strlen(argv[1]) + 1; // '\0' too
        printf("%s: SENDING '%s' ONTO BUS\n", argv[1], argv[1]);
        if (nn_send(sock, argv[1], sz_n, 0) < 0) {
                fatal("nn_send");
        }

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

Compilation

```
gcc bus.c -lnanomsg -o bus
```

Execution

```
./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
```

Output

```
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
```