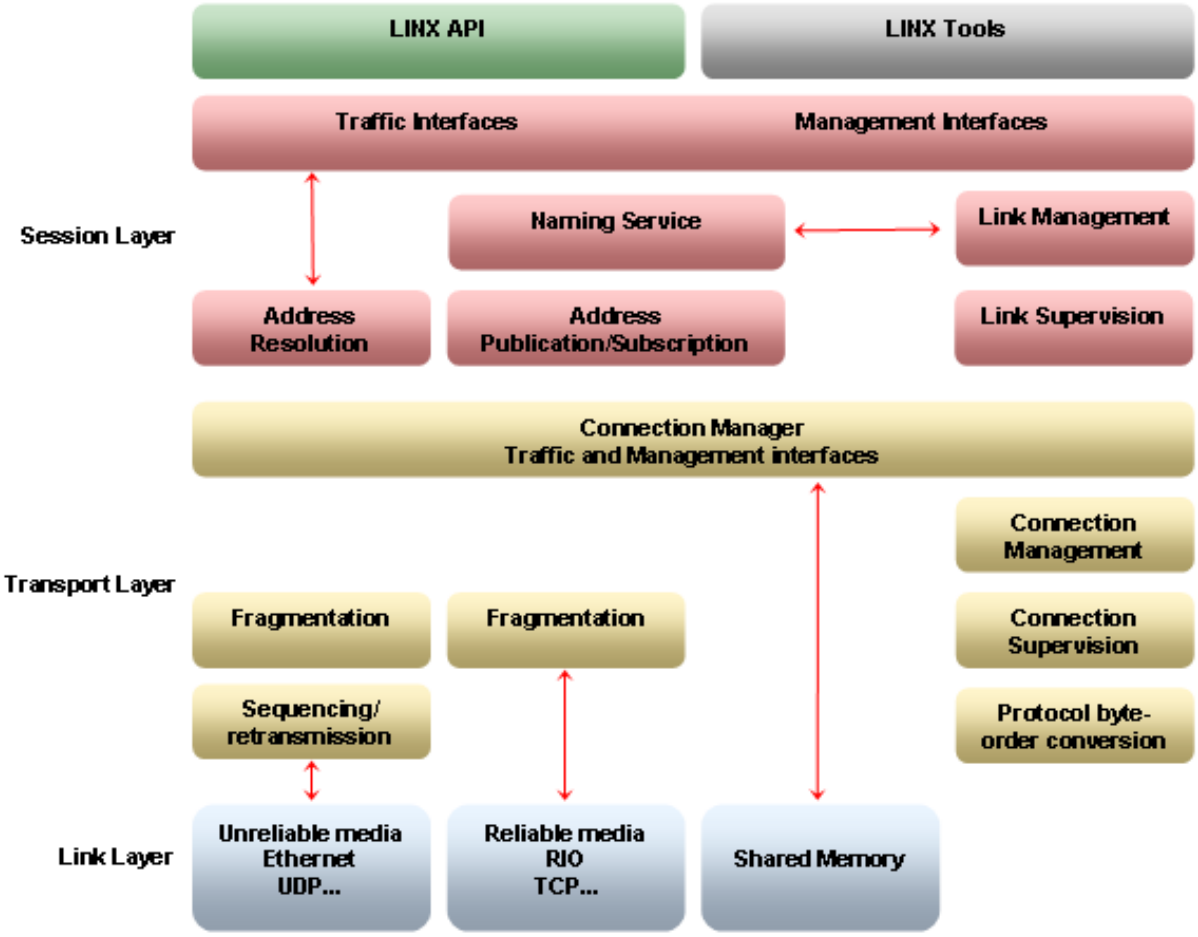
LINX for Linux User's Guide

1. Overview of LINX Communication

Enea LINX is an open technology for location transparent inter-process and intra-node message passing communication for distributed systems, e.g. clusters and multi-core real-time environments. It is platform and interconnect independent with high performance, scaling well to large systems. It is based on a well known transparent message passing method, used for many years by Enea Software AB in the OSE family of real time operating systems.



2. Description and Main Features of LINX

The LINX for Linux is an open source implementation of the LINX inter-process communication protocol for distributed heterogeneous systems. LINX for Linux currently contains support for the Ethernet inter-node transport media and other media will be added.

LINX for Linux includes the loadable **LINX kernel module**, the user-space **LINX API** library and command **tools for configuring and supervising** inter-node communication using LINX.

LINX supports location-transparent and platform-transparent inter-process communication. Applications use the same API, independent of where the communicating peer is located. Peer processes can be in the same node, handled by the same CPU or another CPU in a multi-core system, or processes can be in other nodes, whether part of a cluster or independent, reachable via a suitable transport mechanism. The protocol is specified to allow different operating systems to communication transparently.

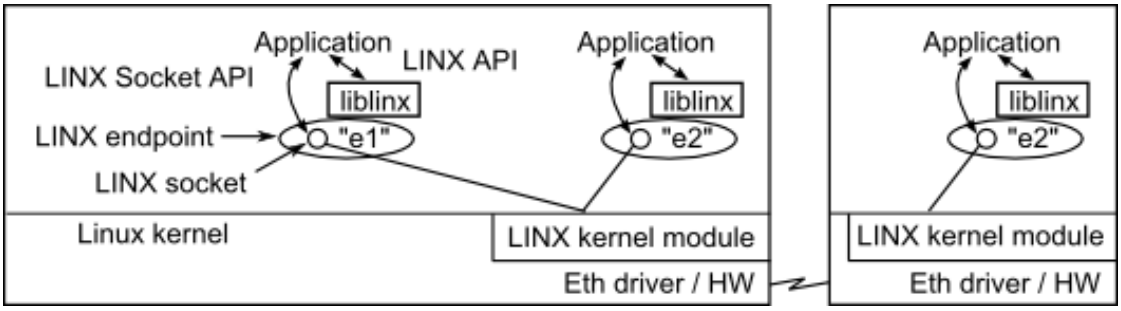
Scalability and real-time performance means that the protocol is designed to be robust and predictable, but still lightweight to avoid overhead and be highly scalable upwards or downwards, while keeping high performance and a small footprint. LINX is used on DSP's with little memory available as well as on large servers in large networks with many nodes. This scaling is easy, as LINX dynamically builds address maps to peers, node links and processes at run-time and only maintains connections and links which are actually used in the node. Clusters can be configured to limit LINX communication to a group of systems.

The LINX protocol is also supported by Enea's OSE / OSEck family of real-time operating systems for small and large systems.

LINX Communication

In LINX for Linux, an application ordinarily uses the **LINX API**. The application is linked with a the liblinx.a LINX library and uses functions calls to communicate. This hides the underlying communication protocols. Each application process or thread creates a **LINX endpoint**, owning a LINX socket and connected to the LINX kernel module. All communication are through this LINX endpoint. This socket is of a special type (PF_LINX). LINX messages and notifications are asynchronous and can arrive at any time.

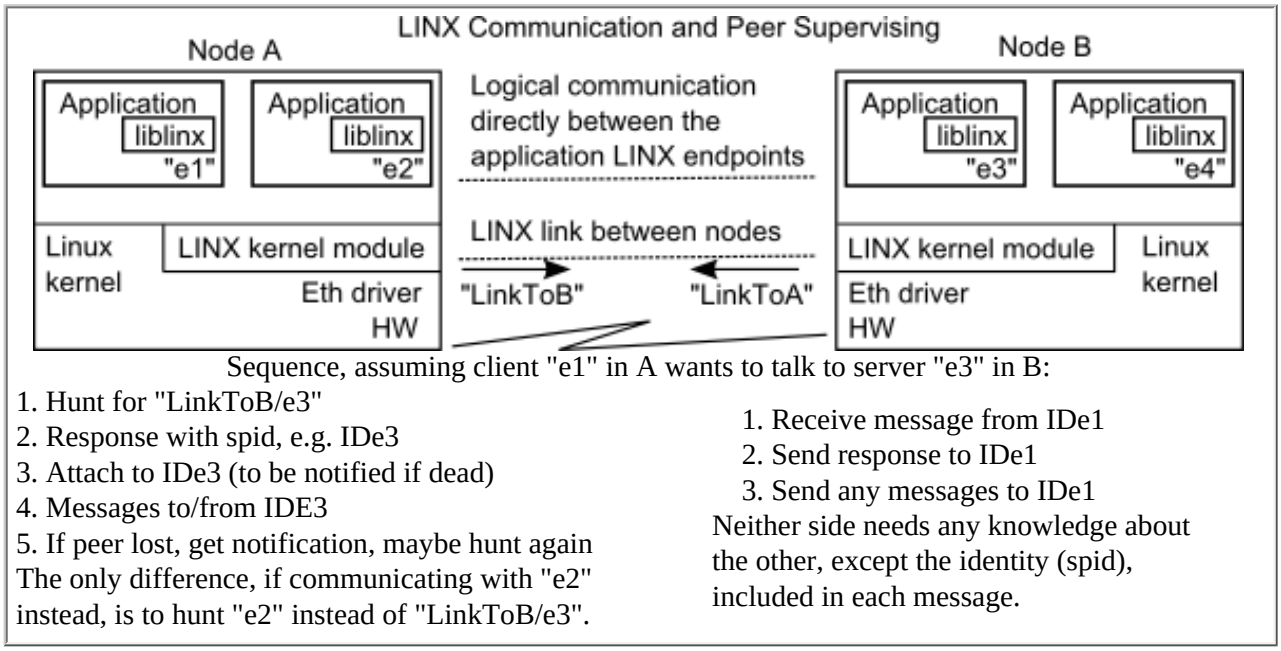
A lower level **LINX Socket API** is also available to be used if needed, e.g. when an application wants to simultaneously wait for multiple events with select(2), without looping and polling.



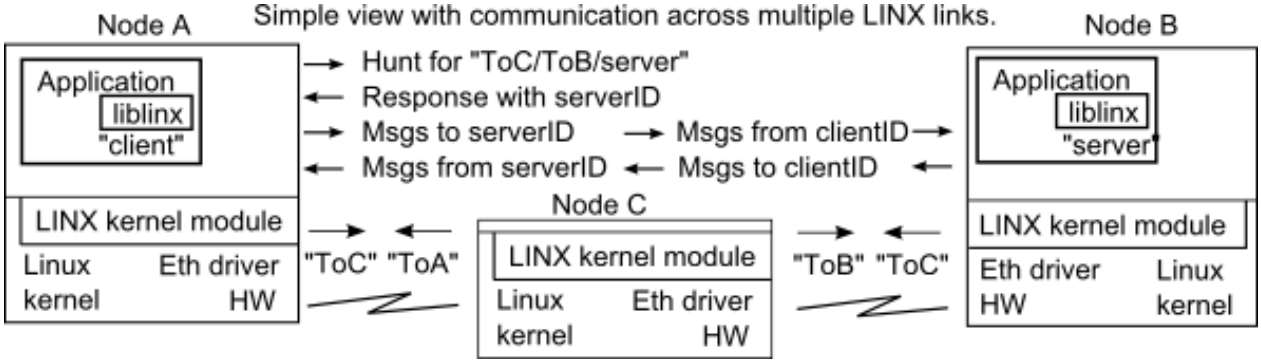
The LINX protocol, used by the LINX kernel module, has two main levels, RLNH and CM. RLNH is generic and handles logical LINX connections and supervision within and between nodes. CM provides a reliable transport. If the underlying transport is non-reliable, e.g. Ethernet, CM needs to provide reliability.

Dynamic Discovery of Peers

Dynamic discovery allows applications to find peers by name and be asynchronously notified when the peers appear; this is called **hunting** for a peer. The notification arrives with a local identifier, often called **spid**, to be used by the application as a handle to the peer, independent of name or location or any ID used in a remote node. Peers can be on the same node or on remote nodes. To hunt for remote peers, the name shall include the name of a LINX link to reach the remote node.



Communication is transparently routed across a series of nodes, if a series of link names are given before the endpoint name, e.g. "ToC/ToB/server".

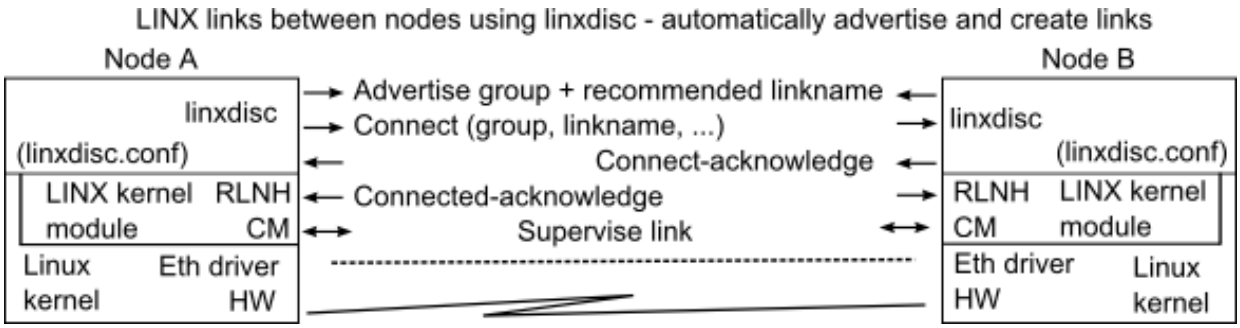


Supervision of Peers

Supervision allows applications to get notified if a peer is lost, whether it dies or a communication link to it is closed/lost. This is called **attaching** to the peer. An application can again hunt for the peer, getting a new notification as soon as it is available again, to continue the communication.

Inter-node Links - Dynamic Discovery and Supervision

Inter-node links are dynamically discovered and supervised by the LINX discovery daemon, linxdisc. Advertisements are broadcast to allow dynamic discovery of other nodes and automatic creation of inter-node links. The configuration can provide limitations to restrict which nodes are allowed, e.g. to form clusters of LINX nodes or to allow/deny specific nodes. Suitable link names are also configured. When a link to another node is created with a link name, and any application has a pending hunt for a peer via this remote link name, LINX gets information from the other node and notifies the application when this peer appears. LINX also supervises the remote peer via the link, as well as supervising the link itself, to notify those applications which have requested it, if a remote peer, or the link to it, is lost.



LINX API and Programming Paradigm

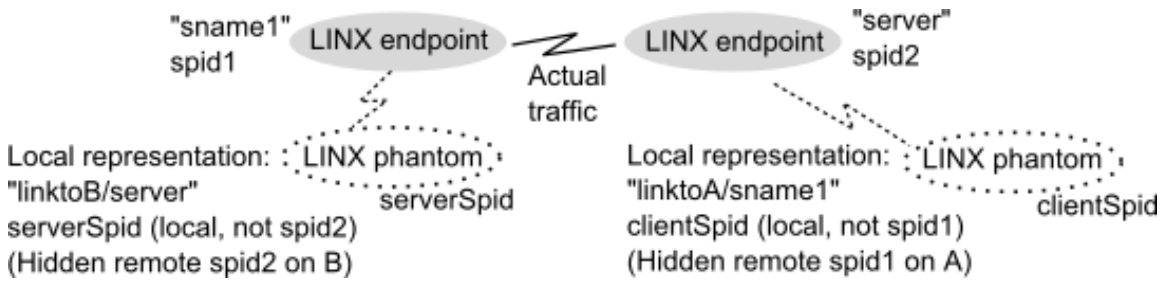
The **LINX-API** provides a powerful but easy-to-use programming paradigm, to transparently communicate with any peer at any location. The following illustration shows the main statements used to communicate. First both applications create **LINX endpoints**, owned by the applications, giving them names ("sname1" and "server" in the example below). The application in node A hunts for the server, using the name "linktoB/server". It knows the name and it know that the server shall be on the remote side of this LINX link. When LINX have discovered node B and they have created a link between each other **and** the process in node B has created it's LINX endpoint (named "server"), a notification is sent to the application in node A with an identity (serverSpid) which the local LINX in A have created. After this step, only this identity is used, transparently and independently of the location. The application may now optionally attach itself to the server process, to be notified if it is lost, and send messages to the server. Messages are always sent in **signal buffers**, allocated at a LINX endpoint, and the ownership of the buffers are transparently transferred to the receiving endpoint. The server needs no knowledge at all about the client, it just receives a message, with an identity (clientSpid) locally created by LINX in node B, and can send a response to this clientSpid.

Linux process in node A

```
LINX lh1 = linx_open("sname1", ....)
linx_hunt(lh1, "linktoB/server", ....)
linx_receive(lh1, &sig, &sigselecthunt)
serverSpid = linx_sender(lh1, &SIG)
linx_free_buf(lh1,&SIG);
linx_attach(lh1, LINX_NIL, serverSpid);
sig2=linx_alloc(lh1,...); // and fill with data
linx_send(lh1, &sig2, serverSpid)
linx_receive(lh1, &SIG, &sigselect)
```

Linux process in node B

```
LINX lh2 = linx_open("server", ....)
linx_receive_w_tmo(lh2, &SIG, ......),
clientSpid = linx_sender(lh2, &SIG)
//    clientSpid is local, not spid1
//    Reuse SIG for response
... Store response in SIG and...
linx_send(lh2, &SIG, clientSpid)
```



A phantom with a local spid represents each remote LINX endpoint, which has been resolved via a hunt call. Phantoms are created on both sides, when a hunt is resolved. Lower level connection links between nodes must previously have been created. Applications only uses local identifiers (spid) to send/receive, thus addressing local endpoints or phantom endpoints.

Signal Buffers

Applications communicate via LINX using **signal buffers**, containing a signal number, followed by any type of data structure. These are allocated via and owned by LINX endpoints. When a signal buffer is sent, it's ownership is transferred to the receiving endpoint. As endpoints are resources owned by the application, these are automatically freed, if the application dies. Each signal buffer includes a signal number, a 32-bit value, which is used to indicate the type of buffer, i.e. the actual data structure after the signal, in the signal buffer. In the LINX API, any signal buffer structure is cast to the generic struct pointer **`LINX_SIGNAL *`**. Transferred signal buffers are queued at the receiving endpoint and can be selectively received by the application. By specifying a list of signal numbers in the linx_receive(3) call, the application will only receive buffers with the selected signal numbers, leaving others in the queue, for later retrieval.

3. LINX Protocol

The LINX protocol is used for transparent communication with remote and local applications. Applications communicate through their **LINX endpoints** and hunt for other applications by their endpoint name. The LINX discovery daemon can automatically create remote inter-node connections (**LINX links**) and supervise these, to allow transparent communication across distributed systems. Applications get notifications when other remote or local applications (with LINX endpoints) become available or are disconnected or die.

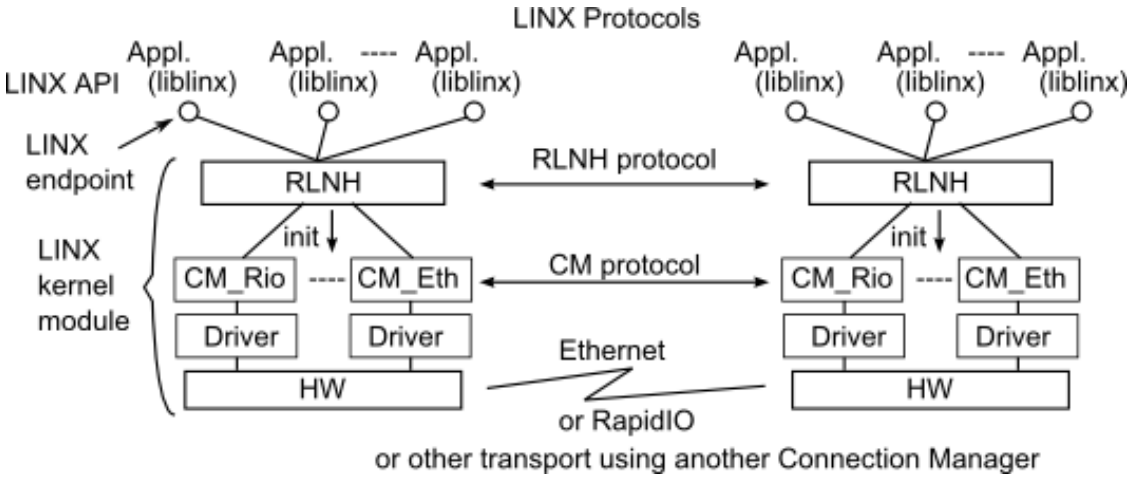
In addition to the LINX protocol, there is a LINX API, with function calls hiding the protocol details from the application.

The LINX protocol is described in a separate document (see Other Links and Documents), but here is a very short summary.

The protocol upper layer, the **RLNH** (rapid link handler) layer, will allow applications to look up LINX endpoints by name, and will actively supervise local and remote endpoints and send notifications when these are disconnected or closed. RLNH relies on the lower LINX level, the CM connection manager, to provide reliable in-order transmission of arbitrary sized messages over any media between nodes. The RLNH is also responsible for creating and supervising remote LINX links to other nodes. This means that it supervises only those endpoints, local and on the remote nodes, that are known to this RLNH and the established links.

The **CM** connection manager is the lower layer and needs to be aware of and support the underlying transport mechanism. If the underlying media is reliable, e.g. RapidIO or shared memory, the CM can be rather simple, but with unreliable media like Ethernet, the CM needs to provide many features, like flow control, peer supervision, retransmission etc.

The LINX for Linux currently contains an implementation of RLNH and CM for Ethernet.



4. Install LINX in Linux

Fetch the LINX distribution as a compressed tar file and unpack the package at a suitable place in your Linux system. When unpacking, a folder with the name **linx-n.n.n/** is created, where n.n.n is the LINX version.

Open the file **doc/index.html** with a browser and read README, RELEASE_NOTES and Changelog for the latest news about this version. Follow the directions in this user's guide to get started using LINX and read reference documentation. Optionally read the MAN page versions of the reference documentation.

Other files and directories:

- Makefile, config.mk, common.mk are make files for building LINX.
- doc/ contains documentation - start reading index.html.
- include/ contains include files for LINX API and LINX Socket API.
- net/ contains the LINX kernel module.
- example/ contains LINX example applications.
- liblinx/ contains the LINX API.
- linxcfg/, linxdisc/, linxstat/ contain tools for configuring and status.
- scripts/ contains a few build scripts.
- bin/ and example/bin are created when LINX is built via **make all**
- example/bin/ is also created if examples are built via **make example**.

After building LINX, the main files to use are:

- linx.ko is the LINX kernel module.
- liblinx.a is the LINX API library.
- Commands: linxdisc, linxcfg, linxstat.
- Examples can be used to learn LINX.

To build your applications, include the **linx.h** header and link with the **liblinx.a** library.

5. Build LINX from the Source

When building the entire LINX package, the following will be built:

- The LINX kernel module - linx.ko
- The LINX API library liblinx.a
- Command binaries in a bin/ directory, which is created

The examples are not built by default with make. When built using `make example` at the top level, the example programs are created in the example/bin/ directory, which is created.

Note that headers in the target Linux kernel source tree must be available, to be able to compile LINX. This is needed also when compiling for the running Linux kernel.

Optionally, reconfigure the LINX Kernel Module

Before building, it is possible to reconfigure compile time constants for the LINX kernel module. Things like the maximum number of remote links to handle per node, maximum number of sockets per established remote link and similar constants can be changed from their default values. Information about these kernel module constants are found in the README file; limits for configuration values are in the RELEASE_NOTES.

Build LINX to Run on Current Host

It is easy to build LINX self hosted, e.g. for the running kernel. Just go to the top of the LINX directory and do make:

```
cd /Your_LINX_Install_Path
make
```

which will build the LINX files.

Optionally build also the LINX examples with:

```
make example
```

Building LINX for a Selected Target, Cross-compiling

Cross compiling LINX for another target, requires that a few variables are set accordingly, either as environment variables or by changing the file **config.mk**. At least the following are needed:

- ARCH - Target architecture, e.g. `ppc`, `i586`, ...
- CROSS_COMPILE - Cross compiler tool prefix, e.g. `powerpc-linux-`
- KERNEL - Kernel source tree

In addition, your PATH environment variable must be set to reach your cross compiler tool kit. With correct PATH, perform the cross compiling build:

```
cd /Your_LINX_Install_Path
make
```

6. Quick Start Using LINX Remote Links

After installing LINX from the tar file and building all with make, according to the previous section, follow this short description to create a simple LINX link between two nodes, here called NodeA and NodeB, after which your applications can start using LINX to communicate. See next section for more about creating remote links, e.g. automatically. See also the Examples section for test applications with source code.

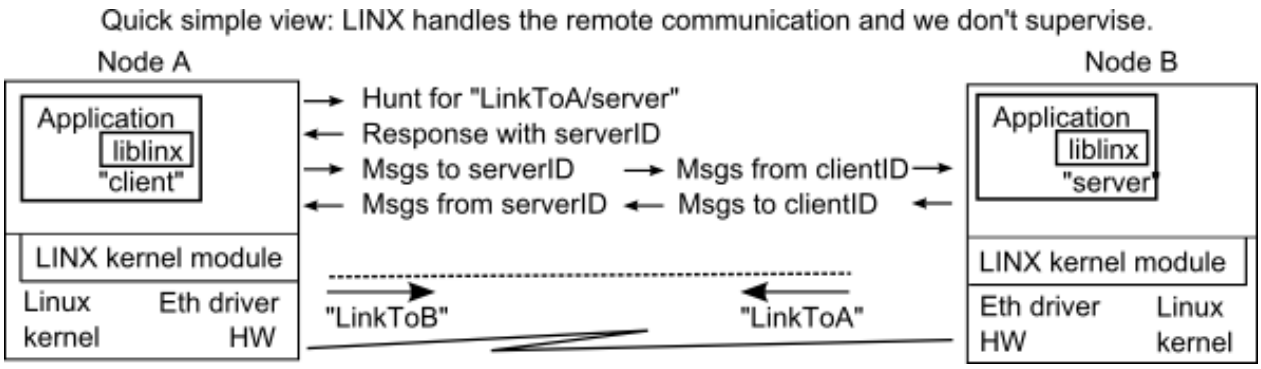
Replace the Ethernet MAC addresses below with the actual values in your two Linux nodes.

On NodeA, do the following to create a LINX link to NodeB:

1. Unpack LINX and build all with make, according to previous description.
2. Load the LINX kernel module into your kernel. You need root permissions.
   ```
   $ sudo insmod net/linx/linx.ko
   ```
3. Create a connection to the remote node (to NodeB) with:
   ```
   $ bin/linxcfg create 0b:0b:0b:0b:0b:0b eth0 LinkToB
   ```
   The connection will stay in state "CONNECTING" until also the remote side has acknowledged and completed the connection.
4. Build your applications - include linx.h and link with liblinx.a.

On NodeB, do the same, but to NodeA, to complete the bidirectional connection:

1. Unpack LINX and build all with make, according to previous description previous description.
2. Load the LINX kernel module into your kernel. You need root permissions.
   ```
   $ sudo insmod net/linx/linx.ko
   ```
3. Create a connection to the remote node (to NodeA) with:
   ```
   $ bin/linxcfg create 0a:0a:0a:0a:0a:0a eth0 LinkToA
   ```
4. Build your applications - include linx.h and link with liblinx.a.



Now your applications can communicate with each other, using the LINX API. The sequence of functions calls used in the applications are shown in the description section. Simplified, an application in node B creates a LINX endpoint, which it calls "server" with linx_open(3). Another application in node A create a LINX endpoint (it's name is irrelevant here). The A application hunts for "LinkToB/server" with linx_hunt(3). and gets a response from LINX with an identifier for the server, this response it read with linx_receive(3) and contains an identifier for the server, serverID. Finally the A application can send messages to serverID and B will receive these to its endpoint, linx_send(3) and linx_receive(3), transferred through the LINX link. Application B sees only a local identifier (clientID) for application A and can send responses or any messages to it. The identifiers are always local in each node.

7. Create and Supervise LINX Links to Other Nodes

Applications may want to find and communicate with other applications on remote nodes. Each application creates a LINX endpoint with a name, through which they can hunt for any other LINX endpoints in local or remote nodes and communicate with them.

To reach remote nodes, **LINX links** with linknames must be created between the nodes, using the LINX protocol. The name of a link (**linkname**) usually is different on the two sides of the link, often the linkname is the name of the remote node connected via the link. Applications reach LINX endpoints on remote nodes, using hunt paths which include the linkname, e. g. "linkname/endpointname". Multiple linknames can be given in the hunt path to reach an endpoint, automatically routing through a series of LINX links.

LINX links to remote nodes are set up and manages, using the LINX protocol. As soon as a link is active, applications can hunt for remote application, using a hunt name consisting of the link name following by the remote LINX endpoint name, e. g. "linktoA/app2". After finding the remote endpoint, the communication is completely transparent, using the received identity number of the remote endpoint.

Automatic Handling of Remote LINX Links - linxdisc Daemon

Remote links can be automatically created and supervised, using the linxdisc(8) daemon in all nodes. This daemon broadcasts advertisements on the local Ethernet. When receiving advertisements from other nodes, it automatically creates links to them.

Configuration of the remote link handing is done in the /etc/linxdisc.conf(5) file, or other file name if given as the linxdisc start option.

Filters in the configuration file will control which remote nodes to which links are allowed. Some changes in the configuration can be immediately applied, by sending SIGHUP to the linxdisc daemon.

Setting up LINX Links to Remote Nodes - linxcfg Command

The linxcfg(1) command can be used to create or destroy LINX links to remote nodes.

This command must be used on both sides to complete a connection - the status will be "connecting" until also the other side has created its side of the connection. The linxstat(1) command can be used to check the status.

LINX Status Display - linxstat Command

Status information from the LINX kernel module can be displayed, using the linxstat(1) command. Status is shown for local and remote LINX endpoints, as well as LINX links to other nodes. The information includes queued signal buffers, pending attaches and pending hunts.

8. Examples - Using LINX with Applications

Running the Simple LINX Example

This example is included in the LINX distribution and consists of a client/server application. One or many client can be run on one or many different nodes, while one program is started as a server on one node. The server can be terminated and restarted, the clients will receive notifications about the server and can resume operation whenever it sees that the server is available. The actual operation is simple, the client sends a message and the server answers.

Run the Simple Example on one Node - no links between nodes are needed here:

- Compile LINX on the node, as earlier described and install the LINX kernel module
  ```
  make
  sudo insmod net/linx/linx.ko
  ```
- Start the server in the background and start the client in the foreground. Ask the client to send 10 requests:
  ```
  example/bin/linx_example_server &
  example/bin/linx_example_client -n 10
  ```

Run the Simple Example on Two Nodes:

- Compile LINX on each node, or cross compile for the nodes, as earlier described. If not compiled locally, transfer the LINX kernel module, the linxcfg command and the LINX example to the two nodes. Modify the paths below if needed.
  ```
  make
  ```
- On each node, install the LINX kernel module
  ```
  sudo insmod net/linx/linx.ko
  ```
- On each node, start linxcfg, giving the device name and the MAC-address of the other node and any suitable linkname. Below as example we use the MAC-addresses 0a:0a:0a:0a:0a:0a and 0b:0b:0b:0b:0b:0b and the device eth0 in both nodes. We select the linkname **srvnode** for the link to the server node. Any linkname can be used for the link from the server to the client node, as it is not used in this example.
  On server node:
  ```
  sudo bin/linxcfg create 0a:0a:0a:0a:0a:0a eth0 srvnode
  ```
  On client node:
  ```
  sudo bin/linxcfg create 0b:0b:0b:0b:0b:0b eth0 anynode
  ```
- On server node, start the server
  ```
  example/bin/linx_example_server
  ```
- On client node, start the client. Tell the client the linkname for the link to the server node.
  ```
  example/bin/linx_example_client -n 10 srvnode
  ```

9. Reference Manuals - LINX for Linux

The reference manual pages for LINX are under the LINX **doc/** directory, in the man1 - man8 subdirectories. The linx(7) manual page is the top document. The linx.h(3) and linx_types.h(3) are included in you applications. To read these with the **man** command in Linux, you need to add the path to the *doc/* directory to your MANPATH environment variable. Example, reading the linx(7) manual page:

```
MANPATH=/Your_LINX_Install_Path/doc man linx
```

Alternatively instead use index.html in the LINX doc directory to read the HTML version of the reference manual pages, which were created from the man page format files. The MAN pages are the originals and any changes must be done there. You might want to check first, that the HTML files have been recreated after the latest changes of the manual pages.

LINX API

The LINX API is described in the manual pages (index.html).

LINX Socket API

The LINX Socket API is described in the linx(7) manual page.

LINX Commands for Configuration, LINX Links Handling and LINX Status

The linxcfg command, the linxdisc daemon and the linxdisc.conf file, as well as the linxstat command are described in their manual pages.

10. LINX Kernel Module Configuration

Parameters can be passed to the LINX kernel module (linx.ko) at load time. For example:

```
insmod linx.ko MAX_NUM_LINKS=64
```

To list available parameters and types, the modinfo command can be used:

```
modinfo -p linx.ko
```

The following parameters can be passed to the LINX kernel module:

max_num_links

Specify the maximum number of links handled by LINX per node. When the maximum number of links is reached, new links will be refused to be created. The default value is 32 and max is 1024.

max_sock_per_link

Specify the maximum number of sockets/spids per established link. If the maximum number of sockets/spids is reached for a specific link, the link will be disconnected and reconnected. The default value is 1024 and max is 65536.

linx_spid_max

Specify the maximum number of sockets/spids per node. When the maximum number of sockets/spids is reached, new sockets/spids can not be opened. If the maximum number of sockets/spids is reached for a specific link, the link will be disconnected and reconnected. The linx_spid_max value needs to be a power of 2. The default value is 512 and max is 65536.

linx_osattref_max

Specify the maximum number of pending attaches per node. When the maximum is reached, attach calls will fail. If the maximum number of sockets/spids is reached for a specific link, the link will be disconnected and reconnected. The linx_osattref_max value needs to be a power of 2. The default value is 1024 and max is 65536.

linx_sockbuf_max

Specifies the send and receive buffer queue size of a Socket. This value is passed to the socket struct fields socket->sk_sndbuf and socket->sk_rcvbuf. These fields control the total number of memory a socket may use for sending and receiving packages. The default value is 1073741824 and max is 1073741824

---

11. Other Links and Documents

LINX Support and Information

Email: linx@enea.com

Other Documentation

See www.enea.com for general information about LINX. You will find a LINX Datasheet, the LINX protocols described, Questions & Answers about LINX and other information.

---

12. Document History

Current version is also seen in document header when printed (HTML title line).

Revision	Author	Date	Status and Description of purpose for new revision
1.1.0.1	lejo	2007-04-09	Minor doc format adjust
1.1.0	lejo	2006-10-30	For LINX for Linux 1.1.
1.0.3	lejo	2006-09-14	For LINX for Linux 1.0.

To print this document from an HTML browser, it is recommended to use Firefox or IE6, which will print this with reasonable page breaks.