

[PATCH 1/1] IPN: Inter Process Networking

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-12/msg05676.html>

- *From:* renzo@xxxxxxxxxxx (Renzo Davoli)
 - *Date:* Mon, 17 Dec 2007 10:27:47 +0100
-

Inter Process Networking Patch.

It applies to 2.6.24-rc5, include documentation, the new kernel option (experimental), kernel include file include/net/af_ipn.h and the protocol directory net/ipn.

renzo

Signed-off-by: Renzo Davoli <renzo@xxxxxxxxxxx>

```
diff -Naur linux-2.6.24-rc5/Documentation/networking/ipn.txt
linux-2.6.24-rc5-ipn/Documentation/networking/ipn.txt
--- linux-2.6.24-rc5/Documentation/networking/ipn.txt 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.24-rc5-ipn/Documentation/networking/ipn.txt 2007-12-16 16:30:01.000000000 +0100
@@ -0,0 +1,326 @@
```

+Inter Process Networking (IPN)

+

+IPN is an Inter Process Communication service. It uses the same programming interface and protocols used for networking. Processes using IPN are connected to a "network" (many to many communication). The messages or packets sent by a process on an IPN network can be delivered to many other processes connected to the same IPN network, potentially to all the other processes. Different protocols can be defined on the IPN service. The basic one is the broadcast (level 1) protocol: all the packets get received by all the processes but the sender. It is also possible to define more sophisticated protocols. For example it is possible to have IPN sockets dispatching packets using the Ethernet protocol (like a Virtual Distributed Ethernet - VDE switch), or Internet Protocol (like a layer 3 switch). These are just examples, several other policies can be defined.

+

+Description:

+-----

+

+The Berkeley socket Application Programming Interface (API) was designed for client server applications and for point-to-point communications. There is not a support for broadcasting/multicasting domains.

+

+IPN updates the interface by introducing a new protocol family (PF_IPN or AF_IPN). PF_IPN is similar to PF_UNIX but for IPN the Socket API calls have a

[PATCH 1/1] IPN: Inter Process Networking

+different (extended) behavior.

+

+ #include <sys/socket.h>

+ #include <sys/un.h>

+ #include <sys/ipn.h>

+

+ sockfd = socket(AF_IPN, int socket_type, int protocol);

+

+creates a communication socket. The only socket_type defined is SOCK_RAW, other
+socket_types can be used for future extensions. A socket cannot be used to send
+or receive data until it gets connected (using the "connect" call). The
+protocol argument defines the policy used by the socket. Protocol IPN_BROADCAST
+(1) is the basic policy: a packet is sent to all the recipients but the sender
+itself. The policy IPN_ANY (0) can be used to connect or bind a pre-existing
+IPN network regardless of the policy used. (2 will be IPN_VDESWITCH and 3
+IPN_VDESWITCHL3).

+

+The address format is the same of PF_UNIX (a.k.a PF_LOCAL), see unix(7) manual.

+

+ int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);

+

+This call creates an IPN network if it does not exist, or join an existing
+network (just for management) if it already exists. The policy of the network
+must be consistent with the protocol argument of the "socket" call. A new
+network has the policy defined for the socket. "bind" or "connect" operations
+on existing networks fail if the policy of the socket is neither IPN_ANY nor
+the same of the network. (A network should not be created by a IPN_ANY socket).
+An IPN network appears in the file system as a unix socket. The execution
+permission (x) on this file is required for "bind" to succeed (otherwise -EPERM
+is returned). Similarly the read/write permissions (rw) permits the "connect"
+operation for reading (receiving) or writing (sending) packets respectively.
+When a socket is bound (but not connected) to a IPN network the process does
+not receive or send any data but it can call "ioctl" or "setsockopt" to
+configure the network.

+

+ int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

+

+This call connects a socket to an existing IPN network. The socket can be
+already bound (through the "bind" call) or unbound. Unbound connected sockets
+receive and send data but they cannot configure the network. The read or write
+permission on the socket (rw) is required to "connect" the channel and
+read/write respectively. When "connect" succeeds and provided the socket has
+appropriate permissions, the process can send packets and receives all the
+packets sent by other processes and delivered to it by the network policy. The
+socket can receive data at any time (like a network interface) so the process
+must be able to handle incoming data (using select/poll or multithreading).
+Obviously higher level protocols can also prevent the reception of unexpected
+messages by design. It is the case of networks used with exactly one
+sender, all the other processes can simply receive the data and the sender will
+never receive any packet. It is also possible to have sockets with different
+roles assigning reading permission to some and writing permissions to others.

[PATCH 1/1] IPN: Inter Process Networking

+If data overrun occurs there can be data loss or the sender can be blocked
+depending on the policy of the socket (LOSSY or LOSSLESS, see over). Bind must
+be called before connect. The correct sequences are: socket+bind: just for
+management, socket+bind+connect: management and communication. socket+connect:
+communication without management).

+
+The calls "accept" and "listen" are not defined for AF_IPN, as there is not any
+server. All the communication takes place among peers.

+
+Data can be sent and received using read, write, send, recv, sendto, recvfrom, sendmsg, recvmsg.

+
+Socket options and flags.
+-----

+
+These options can be set by getsockopt and setsockopt.

+
+There are two different kinds of options: network options and node options. The
+formers define the structure of the network and must be set prior to bind. It
+is not currently possible to change this flag of an existing network. When a
+socket is bound and/or connected to an existing network getsockopt gives the
+current value of the options. Node options define parameters of the node. These
+must be set prior to connect.

+
+***Network Options (These options can be set prior to bind/connec

+
+IPN_SO_FLAGS: This tag permits to set/get the network flags.

+
+IPN_FLAG_LOSSLESS: this flag defines the behavior in case of network
+overloading or data overrun, i.e. when some process are too slow in consuming
+the packets for the network buffers. When the network is LOSSY (the flag is
+cleared) packets get dropped in case of buffer overflow. A LOSSLESS (flag set)
+IPN network blocks the sender if the buffer is full. LOSSY is the default
+behavior.

+
+IPN_SO_NUMNODES: max number of connected sockets (default value 32)

+
+IPN_SO_MTU: maximum transfer unit: maximum size of packets (default value 1514,
+Ethernet frame, including VLAN).

+
+IPN_SO_MSGPOOLSIZE: size of the buffer (#of pending packets, default value 8).
+This option has two different meanings depending on the LOSSY/LOSSLESS behavior
+of the network. For LOSSY networks, this is the maximum number of pending
+packets of each node. For LOSSLESS network this is the global number of the
+pending packets in the network. When the same packet is sent to many
+destinations it is counted just once.

+
+IPN_SO_MODE: this option specifies the permission to use when the socket gets
+created on the file system. It is modified by the process' umask in the usual
+way. The created socket permission are (mode & ~umask).

+
+***Network Options (Options for bound/connected sockets)

[PATCH 1/1] IPN: Inter Process Networking

+
+IPN_SO_CHANGE_NUMNODES: (runtime) change of the number of ipn network ports.
+
+***Node Options
+
+IPN_SO_PORT: (default value IPN_PORTNO_ANY) This option specify the port number
+where the socket must be connected. When IPN_PORTNO_ANY the port number is
+decided by the service. There can be network services where different ports
+have different definitions (e.g. different VLANs for ports of virtual Ethernet
+switches).
+
+IPN_SO_DESCR: This is the description of the node. It is a string, having
+maxlength IPN_DESCRLEN. It is just used by debugging tools.
+
+IPN_SO_HANDLE_OOB: The node is able to manage Out Of Band protocol messages
+
+IPN_SO_WANT_OOB_NUMNODES: The socket wants OOB messages to notify the change
+of #writers #readers (requires IPN_SO_HANDLE_OOB)
+
+TAP and GRAB nodes for IPN networks
+-----
+
+It is possible to connect IPN sockets to virtual and real network interfaces
+using specific ioctl and provided the user has the permission to configure the
+network (e.g. the CAP_NET_ADMIN Posix capability). A virtual interface
+connected to an IPN network is similar to a tap interface (provided by the
+tuntap module). A tap interface appears as an ethernet interface to the hosting
+operating system, all the packets sent and received through the tap interface
+get received and sent by the application which created the tap interface. IPN
+virtual network interface appears in the same way but the packets are received
+and sent through the IPN network and delivered consistently with the policy
+(BROADCAST acts as a basic HUB for the connected processes). It is also
+possible to *grab* a real interface. In this case the closest example is the
+Linux kernel ethernet bridge. When a real interface is connected to a IPN all
+the packets received from the real network are injected also into the IPN and
+all the packets sent by the IPN through the real network 'port' get sent on the
+real network.
+
+ioctl is used for creation or control of TAP or GRAB interfaces.
+
+ int ioctl(int d, int request, .../* arg */);
+
+A list of the request values currently supported follows.
+
+IPN_CONN_NETDEV: (struct ifreq *arg). This call creates a TAP interface or
+implements a GRAB on an existing interface and connects it to a bound IPN
+socket. The field ifr_flags can be IPN_NODEFLAG_TAP for a TAP interface,
+IPN_NODEFLAG_GRAB to grab an existing interface. The field ifr_name is the
+desired name for the new TAP interface or is the name of the interface to grab
+(e.g. eth0). For TAP interfaces, ifr_name can be an empty string. The interface
+in this latter case is named ipn followed by a number (e.g. ipn0, ipn1, ...).

[PATCH 1/1] IPN: Inter Process Networking

+This ioctl must be used on a bound but unconnected socket. When the call
+succeeds, the socket gets the connected status, but the packets are sent and
+received through the interface. Persistence apply only to interface nodes (TAP
+or GRAB).

+
+IPN_SETPERSIST (int arg). If (arg != 0) it gives the interface the persistent
+status: the network interface survives and stay connected to the IPN network
+when the socket is closed. When (arg == 0) the standard behavior is resumed:
+the interface is deleted or the grabbing is terminated when the socket is
+closed.

+
+IPN_JOIN_NETDEV: (struct ifreq *arg). This call reconnects a socket to an
+existing persistent node. The interface can be defined either by name
+(ifr_name) or by index (ifr_index). If there is already a socket controlling
+the interface this call fails (EADDRNOTAVAIL).

+
+There are also some ioctl that can be used by a sysadm to give/clear
+persistence on existing IPN interfaces. These calls apply to unbound sockets.

+
+IPN_SETPERSIST_NETDEV: (struct ifreq *arg). This call sets the persistence
+status of an IPN interface. The interface can be defined either by name
+(ifr_name) or by index (ifr_index).

+
+IPN_CLRERSIST_NETDEV: (struct ifreq *arg). This call clears the persistence
+status of an IPN interface. The interface is specified as in the opposite call
+above. The interface is deleted (TAP) or the grabbing is terminated when the
+socket is closed, or immediately if the interface is not controlled by a
+socket. If the IPN network had the interface as its sole node, the IPN network
+is terminated, too.

+
+When unloading the ipn kernel module, all the persistent flags of interfaces
+are cleared.

+
+Related Work.

+-----

+
+IPN is able to give a unifying solution to several problems and creates new
+opportunities for applications.

+
+Several existing tools can be implemented using IPN sockets:

- + * VDE. Level 2 service implements a VDE switch in the kernel, providing a
+ considerable speedup.
- + * Tap (tuntap) networking for virtual machines
- + * Kernel ethernet bridge
- + * All the applications which need multicasting of data streams, like tee

+
+A continuous stream of data (like audio/video/midi etc) can be sent on an IPN
+network and several application can receive the broadcast just by joining the
+channel.

+
+

[PATCH 1/1] IPN: Inter Process Networking

+It is possible to write programs that forward packets between different IPN
+networks running on the same or on different systems extending the IPN in the
+same way as cables extend ethernet networks connecting switches or hubs
+together. (VDE cables are examples of such a kind of programs).

+

+IPN interface to protocol modules

+-----

+

```
+struct ipn_protocol {
+ int refcnt;
+ int (*ipn_p_newport)(struct ipn_node *newport);
+ int (*ipn_p_handlemsg)(struct ipn_node *from,struct msgpool_item *msgitem, int depth);
+ void (*ipn_p_delport)(struct ipn_node *oldport);
+ void (*ipn_p_postnewport)(struct ipn_node *newport);
+ void (*ipn_p_predelport)(struct ipn_node *oldport);
+ int (*ipn_p_newnet)(struct ipn_network *newnet);
+ int (*ipn_p_resizenet)(struct ipn_network *net,int oldsize,int newsize);
+ void (*ipn_p_delnet)(struct ipn_network *oldnet);
+ int (*ipn_p_setsockopt)(struct ipn_node *port,int optname,
+ char __user *optval, int optlen);
+ int (*ipn_p_getsockopt)(struct ipn_node *port,int optname,
+ char __user *optval, int *optlen);
+ int (*ipn_p_ioctl)(struct ipn_node *port,unsigned int request,
+ unsigned long arg);
+};
+
+int ipn_proto_register(int protocol,struct ipn_protocol *ipn_service);
+int ipn_proto_deregister(int protocol);
+
+void ipn_proto_sendmsg(struct ipn_node *to, struct msgpool_item *msg, int depth);
```

+

+

+A protocol (sub) module must define its own ipn_protocol structure (maybe a
+global static variable).

+

+ipn_proto_register must be called in the module init to register the protocol
+to the IPN core module. ipn_proto_deregister must be called in the destructor
+of the module. It fails if there are already running networks based on this
+protocol.

+

+Only two fields must be initialized in any case: ipn_p_newport and
+ipn_p_handlemsg.

+

+ipn_p_newport is the new network node notification. The return value is the
+port number of the new node. This call can be used to allocate and set private
+data used by the protocol (the field proto_private of the struct ipn_node has
+been defined for this purpose).

+

+ipn_p_handlemsg is the notification of a message that must be dispatched. This
+function should call ipn_proto_sendmsg for each recipient. It is possible for
+the protocol to change the message (provided the global length of the packet

[PATCH 1/1] IPN: Inter Process Networking

+does not exceed the MTU of the network). Depth is for loop control. Two IPN can
+be interconnected by kernel cables (not implemented yet). Cycles of cables
+would generate infinite loops of packets. After a pre-defined number of hops
+the packet gets dropped (it is like EMLINK for symbolic links). Depth value
+must be copied to all ipn_proto_sendmsg calls. Usually the handlemsg function
+has the following structure:

```
+  
+static int ipn_xxxxx_handlemsg(struct ipn_node *from, struct msgpool_item *msgitem, int depth)  
+{  
+ /* compute the set of recipients */  
+ for (/*each recipient "to"*/)  
+ ipn_proto_sendmsg(to,msgitem,depth);  
+}
```

+
+It is also possible to send different packets to different recipients.

```
+  
+struct msgpool_item *newitem=ipn_msgpool_alloc(from->ipn);  
+/* create a new contents for the packet by filling in newitem->len and newitem->data */  
+ipn_proto_sendmsg(recipient1,newitem,depth);  
+ipn_proto_sendmsg(recipient2,newitem,depth);
```

+....

```
+ipn_msgpool_put(newitem);
```

+
+

+(please remember to call ipn_msgpool_put after the sendmsg of packets allocated
+by the protocol submodule).

+
+

+ipn_p_delpport is used to deallocate port related data structures.

+
+

+ipn_p_postnewport and ipn_p_predelpport are used to notify new nodes or deleted
+nodes. newport and delpport get called before activating the port and after
+disactivating it respectively, therefore it is not possible to use the new port
+or deleted port to signal the change on the net itself. ipn_p_postnewport and
+ipn_p_predelpport get called just after the activation and just before the
+deactivation thus the protocols can already send packets on the network.

+
+

+ipn_p_newnet and ipn_p_delnet notify the creation/deletion of a IPN network
+using the given protocol.

+
+

+ipn_p_resizenet notifies a number of ports change

+
+

+ipn_p_setsockopt and ipn_p_getsockopt can be used to provide specific socket
+options.

+
+

+ipn_p_ioctl protocols can implement also specific ioctl services.

+
+

+Further documentation and examples can be found in the Virtual Square project
+web site: wiki.virtualsquare.org

```
diff -Naur linux-2.6.24-rc5/MAINTAINERS linux-2.6.24-rc5-ipn/MAINTAINERS
```

```
--- linux-2.6.24-rc5/MAINTAINERS 2007-12-11 04:48:43.000000000 +0100
```

```
+++ linux-2.6.24-rc5-ipn/MAINTAINERS 2007-12-16 16:30:01.000000000 +0100
```

```
@@ -2094,6 +2094,15 @@
```

[PATCH 1/1] IPN: Inter Process Networking

W: <http://openipmi.sourceforge.net/>

S: Supported

+IPN INTER PROCESS NETWORKING

+P: Renzo Davoli

+M: renzo@xxxxxxxxxxxx

+P: Ludovico Gardenghi

+M: garden@xxxxxxxxxxxx

+L: netdev@xxxxxxxxxxxxxxxx

+W: <http://wiki.virtualsquare.org>

+S: Maintained

+

IPX NETWORK LAYER

P: Arnaldo Carvalho de Melo

M: acme@xxxxxxxxxxxxxxxxxxxx

diff -Naur linux-2.6.24-rc5/include/linux/net.h linux-2.6.24-rc5-ipn/include/linux/net.h

--- linux-2.6.24-rc5/include/linux/net.h 2007-12-11 04:48:43.000000000 +0100

+++ linux-2.6.24-rc5-ipn/include/linux/net.h 2007-12-16 16:30:03.000000000 +0100

@@ -25,7 +25,7 @@

struct inode;

struct net;

---#define NPROTO 34 /* should be enough for now.. */

+++#define NPROTO 35 /* should be enough for now.. */

#define SYS_SOCKET 1 /* sys_socket(2) */

#define SYS_BIND 2 /* sys_bind(2) */

diff -Naur linux-2.6.24-rc5/include/linux/netdevice.h linux-2.6.24-rc5-ipn/include/linux/netdevice.h

--- linux-2.6.24-rc5/include/linux/netdevice.h 2007-12-11 04:48:43.000000000 +0100

+++ linux-2.6.24-rc5-ipn/include/linux/netdevice.h 2007-12-16 16:30:03.000000000 +0100

@@ -705,6 +705,8 @@

struct net_bridge_port *br_port;

/* macvlan */

struct macvlan_port *macvlan_port;

+ /* ipn */

+ struct ipn_node *ipn_port;

/* class/net/name entry */

struct device dev;

diff -Naur linux-2.6.24-rc5/include/linux/socket.h linux-2.6.24-rc5-ipn/include/linux/socket.h

--- linux-2.6.24-rc5/include/linux/socket.h 2007-12-11 04:48:43.000000000 +0100

+++ linux-2.6.24-rc5-ipn/include/linux/socket.h 2007-12-16 16:30:03.000000000 +0100

@@ -189,7 +189,8 @@

#define AF_BLUETOOTH 31 /* Bluetooth sockets */

#define AF_IUCV 32 /* IUCV sockets */

#define AF_RXRPC 33 /* RxRPC sockets */

---#define AF_MAX 34 /* For now.. */

+++#define AF_IPN 34 /* IPN sockets */

+++#define AF_MAX 35 /* For now.. */

/* Protocol families, same as address families. */

[PATCH 1/1] IPN: Inter Process Networking

```
#define PF_UNSPEC AF_UNSPEC
@@ -224,6 +225,7 @@
#define PF_BLUETOOTH AF_BLUETOOTH
#define PF_IUCV AF_IUCV
#define PF_RXRPC AF_RXRPC
+#define PF_IPN AF_IPN
#define PF_MAX AF_MAX

/* Maximum queue length specifiable by listen. */
diff -Naur linux-2.6.24-rc5/include/net/af_ipn.h linux-2.6.24-rc5-ipn/include/net/af_ipn.h
--- linux-2.6.24-rc5/include/net/af_ipn.h 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.24-rc5-ipn/include/net/af_ipn.h 2007-12-16 16:30:03.000000000 +0100
@@ -0,0 +1,233 @@
+#ifndef __LINUX_NET_AFIPN_H
+#define __LINUX_NET_AFIPN_H
+
+#define IPN_ANY 0
+#define IPN_BROADCAST 1
+#define IPN_HUB 1
+#define IPN_VDESWITCH 2
+#define IPN_VDESWITCH_L3 3
+
+#define IPN_SO_PREBIND 0x80
+#define IPN_SO_PORT 0
+#define IPN_SO_DESCR 1
+#define IPN_SO_CHANGE_NUMNODES 2
+#define IPN_SO_HANDLE_OOB 3
+#define IPN_SO_WANT_OOB_NUMNODES 4
+#define IPN_SO_MTU (IPN_SO_PREBIND | 0)
+#define IPN_SO_NUMNODES (IPN_SO_PREBIND | 1)
+#define IPN_SO_MSGPOOLSIZE (IPN_SO_PREBIND | 2)
+#define IPN_SO_FLAGS (IPN_SO_PREBIND | 3)
+#define IPN_SO_MODE (IPN_SO_PREBIND | 4)
+
+#define IPN_PORTNO_ANY -1
+
+#define IPN_DESCLLEN 128
+
+#define IPN_FLAG_LOSSLESS 1
+#define IPN_FLAG_TERMINATED 0x1000
+
+/* Ioctl defines */
+#define IPN_SETPERSIST_NETDEV_IOW('I', 200, int)
+#define IPN_CLRERSIST_NETDEV_IOW('I', 201, int)
+#define IPN_CONN_NETDEV_IOW('I', 202, int)
+#define IPN_JOIN_NETDEV_IOW('I', 203, int)
+#define IPN_SETPERSIST_IOW('I', 204, int)
+
+#define IPN_OOB_NUMNODE_TAG 0
+
+/* OOB message for change of numnodes
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ * Common fields for oob IPN signaling:
+ * @level=level of the service who generated the oob
+ * @tag=tag of the message
+ * Specific fields:
+ * @numreaders=number of readers
+ * @numwriters=number of writers
+ */
+struct numnode_oob {
+ int level;
+ int tag;
+ int numreaders;
+ int numwriters;
+};
+
+#ifdef __KERNEL__
+#include <linux/socket.h>
+#include <linux/mutex.h>
+#include <linux/un.h>
+#include <net/sock.h>
+#include <linux/netdevice.h>
+
+#define IPN_HASH_SIZE 256
+
+/* The AF_IPN socket */
+struct msgpool_item;
+struct ipn_network;
+struct pre_bind_parms;
+
+/*
+ * ipn_node
+ *
+ * @nodelist=pointers for connectqueue or unconnectqueue (see network)
+ * @protocol=kind of service 0->standard broadcast
+ * @flags= see IPN_NODEFLAG_xxx
+ * @shutdown= SEND_SHUTDOWN/RCV_SHUTDOWN and OOBRCV_SHUTDOWN
+ * @descr=description of this port
+ * @portno=when connected: port of the netowrk (<0 means unconnected)
+ * @msglock=mux on the msg queue
+ * @totmsgcount=total # of pending msgs
+ * @oobmsgcount=# of pending oob msgs
+ * @msgqueue=queue of messages
+ * @oobmsgqueue=queue of messages
+ * @read_wait=waitqueue for reading
+ * @net=current network
+ * @dev=device (TAP or GRAB)
+ * @ipn=network we are connected to
+ * @pbp=temporary storage for parms that must be set prior to bind
+ * @proto_private=handle for protocol private data
+ */
+struct ipn_node {
+ struct list_head nodelist;
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ int protocol;
+ volatile unsigned char flags;
+ unsigned char shutdown;
+ char descr[IPN_DESCRLEN];
+ int portno;
+ spinlock_t msglock;
+ unsigned short totmsgcount;
+ unsigned short oobmsgcount;
+ struct list_head msgqueue;
+ struct list_head oobmsgqueue;
+ wait_queue_head_t read_wait;
+ struct net *net;
+ struct net_device *dev;
+ struct ipn_network *ipn;
+ struct pre_bind_parms *pbp;
+ void *proto_private;
+};
+#define IPN_NODEFLAG_BOUND 0x1 /* bind succeeded */
+#define IPN_NODEFLAG_INUSE 0x2 /* is currently "used" (0 for persistent, unbound interfaces) */
+#define IPN_NODEFLAG_PERSIST 0x4 /* if persist does not disappear on close (net interfaces) */
+#define IPN_NODEFLAG_TAP 0x10 /* This is a tap interface */
+#define IPN_NODEFLAG_GRAB 0x20 /* This is a grab of a real interface */
+#define IPN_NODEFLAG_DEVMASK 0x30 /* True if this is a device */
+#define IPN_NODEFLAG_OOB_NUMNODES 0x40 /* Node wants OOB for NNODES */
+
+/*
+ * ipn_sock
+ *
+ * unfortunately we must use a struct sock (most of the fields are useless) as
+ * this is the standard "agnostic" structure for socket implementation.
+ * This proves that it is not "agnostic" enough!
+ */
+
+struct ipn_sock {
+ struct sock sk;
+ struct ipn_node *node;
+};
+
+/*
+ * ipn_network network descriptor
+ *
+ * @hnode=hash to find this entry (looking for i-node)
+ * @unconnectqueue=queue of unconnected (bound) nodes
+ * @connectqueue=queue of connected nodes (faster for broadcasting)
+ * @refcnt=reference count (bound or connected sockets)
+ * @dentry/@mnt=to keep the file system descriptor into memory
+ * @ipnn_lock=lock for protocol functions
+ * @protocol=kind of service
+ * @flags=flags (IPN_FLAG_LOSSLESS)
+ * @maxports=number of ports available in this network
+ * @msgpool_nelem=number of pending messages
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ * @msgpool_size=max number of pending messages *per net* when IPN_FLAG_LOSSLESS
+ * @msgpool_size=max number of pending messages *per port*when LOSSY
+ * @mtu=MTU
+ * @send_wait=wait queue waiting for a message in the msgpool (IPN_FLAG_LOSSLESS)
+ * @msgpool_cache=slab for msgpool (unused yet)
+ * @proto_private=handle for protocol private data
+ * @connports=array of connected sockets
+ */
+struct ipn_network {
+ struct hlist_node hnode;
+ struct list_head unconnectqueue;
+ struct list_head connectqueue;
+ atomic_t refcnt;
+ struct dentry *dentry;
+ struct vfsmount *mnt;
+ struct semaphore ipnn_mutex;
+ int sunaddr_len;
+ struct sockaddr_un sunaddr;
+ unsigned int protocol;
+ unsigned int flags;
+ int numreaders;
+ int numwriters;
+ atomic_t msgpool_nelem;
+ unsigned short maxports;
+ unsigned short msgpool_size;
+ unsigned short mtu;
+ wait_queue_head_t send_wait;
+ struct kmem_cache *msgpool_cache;
+ void *proto_private;
+ struct ipn_node **connport;
+};
+
+/* struct msgpool_item
+ * the local copy of the message for dispatching
+ * @count refcount
+ * @len packet len
+ * @data payload
+ */
+struct msgpool_item {
+ atomic_t count;
+ int len;
+ unsigned char data[0];
+};
+
+struct msgpool_item *ipn_msgpool_alloc(struct ipn_network *ipnn);
+void ipn_msgpool_put(struct msgpool_item *old, struct ipn_network *ipnn);
+
+/*
+ * protocol service:
+ *
+ * @refcnt: number of networks using this protocol
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ * @newport=upcall for reporting a new port. returns the portno, -1=error
+ * @handlemsg=dispatch a message.
+ * should call ipn_proto_sendmsg for each destination
+ * can allocate other msgitems using ipn_msgpool_alloc to send
+ * different messages to different destinations;
+ * @delport=(may be null) reports the terminatio of a port
+ * @postnewport,@predelport: similar to newport/delport but during these calls
+ * the node is (still) connected. Useful when protocols need
+ * welcome and goodbye messages.
+ * @ipn_p_setsockopt
+ * @ipn_p_getsockopt
+ * @ipn_p_ioctl=(may be null) upcall to manage specific options or ctls.
+ */
+struct ipn_protocol {
+ int refcnt;
+ int (*ipn_p_newport)(struct ipn_node *newport);
+ int (*ipn_p_handlemsg)(struct ipn_node *from,struct msgpool_item *msgitem);
+ void (*ipn_p_delport)(struct ipn_node *oldport);
+ void (*ipn_p_postnewport)(struct ipn_node *newport);
+ void (*ipn_p_predelport)(struct ipn_node *oldport);
+ int (*ipn_p_newnet)(struct ipn_network *newnet);
+ int (*ipn_p_resizenet)(struct ipn_network *net,int oldsize,int newsize);
+ void (*ipn_p_delnet)(struct ipn_network *oldnet);
+ int (*ipn_p_setsockopt)(struct ipn_node *port,int optname,
+ char __user *optval, int optlen);
+ int (*ipn_p_getsockopt)(struct ipn_node *port,int optname,
+ char __user *optval, int *optlen);
+ int (*ipn_p_ioctl)(struct ipn_node *port,unsigned int request,
+ unsigned long arg);
+};
+
+int ipn_proto_register(int protocol,struct ipn_protocol *ipn_service);
+int ipn_proto_deregister(int protocol);
+
+int ipn_proto_injectmsg(struct ipn_node *from, struct msgpool_item *msg);
+void ipn_proto_sendmsg(struct ipn_node *to, struct msgpool_item *msg);
+void ipn_proto_oobsendmsg(struct ipn_node *to, struct msgpool_item *msg);
+
+extern struct sk_buff *(*ipn_handle_frame_hook)(struct ipn_node *p,
+ struct sk_buff *skb);
+
+#endif
+#endif
diff -Naur linux-2.6.24-rc5/net/Kconfig linux-2.6.24-rc5-ipn/net/Kconfig
--- linux-2.6.24-rc5/net/Kconfig 2007-12-11 04:48:43.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/Kconfig 2007-12-16 16:30:04.000000000 +0100
@@ -37,6 +37,7 @@
source "net/packet/Kconfig"
source "net/unix/Kconfig"
+source "net/ipn/Kconfig"
source "net/xfrm/Kconfig"
```

[PATCH 1/1] IPN: Inter Process Networking

```
source "net/iucv/Kconfig"
```

```
diff -Naur linux-2.6.24-rc5/net/Makefile linux-2.6.24-rc5-ipn/net/Makefile
--- linux-2.6.24-rc5/net/Makefile 2007-12-11 04:48:43.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/Makefile 2007-12-16 16:30:04.000000000 +0100
@@ -19,6 +19,7 @@
obj-$(CONFIG_INET) += ipv4/
obj-$(CONFIG_XFRM) += xfrm/
obj-$(CONFIG_UNIX) += unix/
+obj-$(CONFIG_IPN) += ipn/
ifneq ($(CONFIG_IPV6),)
obj-y += ipv6/
endif
diff -Naur linux-2.6.24-rc5/net/core/dev.c linux-2.6.24-rc5-ipn/net/core/dev.c
--- linux-2.6.24-rc5/net/core/dev.c 2007-12-11 04:48:43.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/core/dev.c 2007-12-16 16:30:04.000000000 +0100
@@ -1925,7 +1925,7 @@
int *ret,
struct net_device *orig_dev)
{
- if (skb->dev->macvlan_port == NULL)
+ if (!skb || skb->dev->macvlan_port == NULL)
return skb;

if (*pt_prev) {
@@ -1938,6 +1938,32 @@
#define handle_macvlan(skb, pt_prev, ret, orig_dev) (skb)
#endif

+#if defined(CONFIG_IPN) || defined(CONFIG_IPN_MODULE)
+struct sk_buff *(*ipn_handle_frame_hook)(struct ipn_node *port,
+ struct sk_buff *skb) __read_mostly;
+EXPORT_SYMBOL_GPL(ipn_handle_frame_hook);
+
+static inline struct sk_buff *handle_ipn(struct sk_buff *skb,
+ struct packet_type **pt_prev,
+ int *ret,
+ struct net_device *orig_dev)
+{
+ struct ipn_node *port;
+
+ if (!skb || skb->pkt_type == PACKET_LOOPBACK ||
+ (port = rcu_dereference(skb->dev->ipn_port)) == NULL)
+ return skb;
+
+ if (*pt_prev) {
+ *ret = deliver_skb(skb, *pt_prev, orig_dev);
+ *pt_prev = NULL;
+ }
+ return ipn_handle_frame_hook(port, skb);
+}

```

[PATCH 1/1] IPN: Inter Process Networking

```
+#else
+#define handle_ipn(skb, pt_prev, ret, orig_dev) (skb)
+#endif
+
+#ifdef CONFIG_NET_CLS_ACT
+/* TODO: Maybe we should just force sch_ingress to be compiled in
+ * when CONFIG_NET_CLS_ACT is? otherwise some useless instructions
+@@ -2070,9 +2096,8 @@
+#endif

skb = handle_bridge(skb, &pt_prev, &ret, orig_dev);
- if (!skb)
- goto out;
skb = handle_macvlan(skb, &pt_prev, &ret, orig_dev);
+ skb = handle_ipn(skb, &pt_prev, &ret, orig_dev);
if (!skb)
goto out;

diff -Naur linux-2.6.24-rc5/net/ipn/Kconfig linux-2.6.24-rc5-ipn/net/ipn/Kconfig
--- linux-2.6.24-rc5/net/ipn/Kconfig 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/ipn/Kconfig 2007-12-16 16:30:04.000000000 +0100
@@ -0,0 +1,21 @@
+#
+# Unix Domain Sockets
+#
+
+config IPN
+ tristate "IPN domain sockets (EXPERIMENTAL)"
+ depends on EXPERIMENTAL
+ ---help---
+ If you say Y here, you will include support for IPN domain sockets.
+ Inter Process Networking socket are similar to Unix sockets but
+ they support peer-to-peer, one-to-many and many-to-many communication
+ among processes.
+ Sub-Modules can be loaded to provide dispatching protocols.
+ This service include the IPN_BROADCAST policy: all the messages get
+ sent to all the recipients (but the sender itself).
+
+ To compile this driver as a module, choose M here: the module will be
+ called ipn.
+
+ If unsure, say 'N'.
+
diff -Naur linux-2.6.24-rc5/net/ipn/Makefile linux-2.6.24-rc5-ipn/net/ipn/Makefile
--- linux-2.6.24-rc5/net/ipn/Makefile 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/ipn/Makefile 2007-12-16 16:30:04.000000000 +0100
@@ -0,0 +1,8 @@
+#
+### Makefile for the IPN (Inter Process Networking) domain socket layer.
+#
+
```

[PATCH 1/1] IPN: Inter Process Networking

```
+obj-$(CONFIG_IPN) += ipn.o
+
+ipn-y := af_ipn.o ipn_netdev.o
+
diff -Naur linux-2.6.24-rc5/net/ipn/af_ipn.c linux-2.6.24-rc5-ipn/net/ipn/af_ipn.c
--- linux-2.6.24-rc5/net/ipn/af_ipn.c 1970-01-01 01:00:00.000000000 +0100
+++ linux-2.6.24-rc5-ipn/net/ipn/af_ipn.c 2007-12-16 18:53:13.000000000 +0100
@@ -0,0 +1,1540 @@
+/*
+ * Main inter process networking (virtual distributed ethernet) module
+ * (part of the View-OS project: wiki.virtualsquare.org)
+ *
+ * Copyright (C) 2007 Renzo Davoli (renzo@xxxxxxxxxxxx)
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * Due to this file being licensed under the GPL there is controversy over
+ * whether this permits you to write a module that #includes this file
+ * without placing your module under the GPL. Please consult a lawyer for
+ * advice before doing this.
+ *
+ * WARNING: THIS CODE IS ALREADY EXPERIMENTAL
+ *
+ */
+
+#include <linux/init.h>
+#include <linux/module.h>
+#include <linux/socket.h>
+#include <linux/poll.h>
+#include <linux/un.h>
+#include <linux/list.h>
+#include <linux/mount.h>
+#include <net/sock.h>
+#include <net/af_ipn.h>
+#include "ipn_netdev.h"
+
+MODULE_LICENSE("GPL");
+MODULE_AUTHOR("VIEW-OS TEAM");
+MODULE_DESCRIPTION("IPN Kernel Module");
+
+#define IPN_MAX_PROTO 4
+
+/*extension of RCV_SHUTDOWN defined in include/net/sock.h
+ * when the bit is set recv fails */
+/* NO_OOB: do not send OOB */
+#define RCV_SHUTDOWN_NO_OOB 4
+/* EXTENDED MASK including OOB */
+#define SHUTDOWN_XMASK (SHUTDOWN_MASK | RCV_SHUTDOWN_NO_OOB)
```

[PATCH 1/1] IPN: Inter Process Networking

```
+/* if XRCV_SHUTDOWN is all set recv fails */
+#define XRCV_SHUTDOWN (RCV_SHUTDOWN | RCV_SHUTDOWN_NO_OOB)
+
+/* Network table and hash */
+struct hlist_head ipn_network_table[IPN_HASH_SIZE + 1];
+#DEFINE_SPINLOCK(ipn_table_lock);
+static struct kmem_cache *ipn_network_cache;
+static struct kmem_cache *ipn_node_cache;
+static struct kmem_cache *ipn_msgitem_cache;
+static DECLARE_MUTEX(ipn_glob_mutex);
+
+/* Protocol 1: HUB/Broadcast default protocol. Function Prototypes */
+static int ipn_bcast_newport(struct ipn_node *newport);
+static int ipn_bcast_handlemsg(struct ipn_node *from,
+ struct msgpool_item *msgitem);
+
+/* default protocol IPN_BROADCAST (0) */
+static struct ipn_protocol ipn_bcast = {
+ .refcnt=0,
+ .ipn_p_newport=ipn_bcast_newport,
+ .ipn_p_handlemsg=ipn_bcast_handlemsg};
+/* Protocol table */
+static struct ipn_protocol *ipn_protocol_table[IPN_MAX_PROTO]={ &ipn_bcast};
+
+/* Socket call function prototypes */
+static int ipn_release(struct socket *);
+static int ipn_bind(struct socket *, struct sockaddr *, int);
+static int ipn_connect(struct socket *, struct sockaddr *,
+ int addr_len, int flags);
+static int ipn_getname(struct socket *, struct sockaddr *, int *, int);
+static unsigned int ipn_poll(struct file *, struct socket *, poll_table *);
+static int ipn_ioctl(struct socket *, unsigned int, unsigned long);
+static int ipn_shutdown(struct socket *, int);
+static int ipn_sendmsg(struct kiocb *, struct socket *,
+ struct msghdr *, size_t);
+static int ipn_recvmmsg(struct kiocb *, struct socket *,
+ struct msghdr *, size_t, int);
+static int ipn_setsockopt(struct socket *sock, int level, int optname,
+ char __user *optval, int optlen);
+static int ipn_getsockopt(struct socket *sock, int level, int optname,
+ char __user *optval, int __user *optlen);
+
+/* Network table Management
+ * inode->ipn_network hash table */
+static inline void ipn_insert_network(struct hlist_head *list, struct ipn_network *ipnn)
+{
+ spin_lock(&ipn_table_lock);
+ hlist_add_head(&ipnn->hnode, list);
+ spin_unlock(&ipn_table_lock);
+}
+
```

[PATCH 1/1] IPN: Inter Process Networking

```
+static inline void ipn_remove_network(struct ipn_network *ipnn)
+{
+ spin_lock(&ipn_table_lock);
+ hlist_del(&ipnn->hnode);
+ spin_unlock(&ipn_table_lock);
+}
+
+static struct ipn_network *ipn_find_network_byinode(struct inode *i)
+{
+ struct ipn_network *ipnn;
+ struct hlist_node *node;
+
+ spin_lock(&ipn_table_lock);
+ hlist_for_each_entry(ipnn, node,
+ &ipn_network_table[i->i_ino & (IPN_HASH_SIZE - 1)], hnode) {
+ struct dentry *dentry = ipnn->dentry;
+
+ if(atomic_read(&ipnn->refcnt) > 0 && dentry && dentry->d_inode == i)
+ goto found;
+ }
+ ipnn = NULL;
+found:
+ spin_unlock(&ipn_table_lock);
+ return ipnn;
+}
+
+/* msgpool management
+ * msgpool_item are ipn_network dependent (each net has its own MTU)
+ * for each message sent there is one msgpool_item and many struct msgitem
+ * one for each receiptent.
+ * msgitem are connected to the node's msgqueue or oobmsgqueue.
+ * when a message is delivered to a process the msgitem is deleted and
+ * the count of the msgpool_item is decreased.
+ * msgpool_item elements gets deleted automatically when count is 0*/
+
+struct msgitem {
+ struct list_head list;
+ struct msgpool_item *msg;
+};
+
+/* alloc a fresh msgpool item. count is set to 1.
+ * the typical use is
+ * ipn_msgpool_alloc
+ * for each receiptent
+ * enqueue messages to the process (using msgitem), ipn_msgpool_hold
+ * ipn_msgpool_put
+ * The message can be delivered concurrently. init count to 1 guarantees
+ * that it survives at least until is has been enqueued to all
+ * receivers */
+struct msgpool_item *ipn_msgpool_alloc(struct ipn_network *ipnn)
+{
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ struct msgpool_item *new;
+ new=kmem_cache_alloc(ipnn->msgpool_cache,GFP_KERNEL);
+ atomic_set(&new->count,1);
+ atomic_inc(&ipnn->msgpool_nelem);
+ return new;
+ }
+
+ /* If the service is LOSSLESS, this msgpool call waits for an
+  * available msgpool item */
+static struct msgpool_item *ipn_msgpool_alloc_locking(struct ipn_network *ipnn)
+{
+ if (ipnn->flags & IPN_FLAG_LOSSLESS) {
+ while (atomic_read(&ipnn->msgpool_nelem) >= ipnn->msgpool_size) {
+ if (wait_event_interruptible_exclusive(ipnn->send_wait,
+ atomic_read(&ipnn->msgpool_nelem) < ipnn->msgpool_size))
+ return NULL;
+ }
+ }
+ return ipn_msgpool_alloc(ipnn);
+ }
+
+static inline void ipn_msgpool_hold(struct msgpool_item *msg)
+{
+ atomic_inc(&msg->count);
+ }
+
+ /* decrease count and delete msgpool_item if count == 0 */
+void ipn_msgpool_put(struct msgpool_item *old,
+ struct ipn_network *ipnn)
+{
+ if (atomic_dec_and_test(&old->count)) {
+ kmem_cache_free(ipnn->msgpool_cache,old);
+ atomic_dec(&ipnn->msgpool_nelem);
+ if (ipnn->flags & IPN_FLAG_LOSSLESS) /* could be done anyway */
+ wake_up_interruptible(&ipnn->send_wait);
+ }
+ }
+
+ /* socket calls */
+static const struct proto_ops ipn_ops = {
+ .family = PF_IPN,
+ .owner = THIS_MODULE,
+ .release = ipn_release,
+ .bind = ipn_bind,
+ .connect = ipn_connect,
+ .socketpair = sock_no_socketpair,
+ .accept = sock_no_accept,
+ .getname = ipn_getname,
+ .poll = ipn_poll,
+ .ioctl = ipn_ioctl,
+ .listen = sock_no_listen,
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ .shutdown = ipn_shutdown,
+ .setsockopt = ipn_setsockopt,
+ .getsockopt = ipn_getsockopt,
+ .sendmsg = ipn_sendmsg,
+ .recvmsg = ipn_recvmsg,
+ .mmap = sock_no_mmap,
+ .sendpage = sock_no_sendpage,
+};
+
+static struct proto ipn_proto = {
+ .name = "IPN",
+ .owner = THIS_MODULE,
+ .obj_size = sizeof(struct ipn_sock),
+};
+
+/* create a socket
+ * ipn_node is a separate structure, pointed by ipn_sock -> node
+ * when a node is "persistent", ipn_node survives while ipn_sock gets released*/
+static int ipn_create(struct net *net, struct socket *sock, int protocol)
+{
+ struct ipn_sock *ipn_sk;
+ struct ipn_node *ipn_node;
+
+ if (net != &init_net)
+ return -EAFNOSUPPORT;
+
+ if (sock->type != SOCK_RAW)
+ return -EPROTOTYPE;
+ if (protocol > 0)
+ protocol=protocol-1;
+ else
+ protocol=IPN_BROADCAST-1;
+ if (protocol < 0 || protocol >= IPN_MAX_PROTO ||
+ ipn_protocol_table[protocol] == NULL)
+ return -EPROTONOSUPPORT;
+ ipn_sk = (struct ipn_sock *) sk_alloc(net, PF_IPN, GFP_KERNEL, &ipn_proto);
+
+ if (!ipn_sk)
+ return -ENOMEM;
+ ipn_sk->node=ipn_node=kmem_cache_alloc(ipn_node_cache,GFP_KERNEL);
+ if (!ipn_node) {
+ sock_put((struct sock *) ipn_sk);
+ return -ENOMEM;
+ }
+ sock_init_data(sock,(struct sock *) ipn_sk);
+ sock->state = SS_UNCONNECTED;
+ sock->ops = &ipn_ops;
+ sock->sk=(struct sock *)ipn_sk;
+ INIT_LIST_HEAD(&ipn_node->nodelist);
+ ipn_node->protocol=protocol;
+ ipn_node->flags=IPN_NODEFLAG_INUSE;
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ ipn_node->shutdown=RCV_SHUTDOWN_NO_OOB;
+ ipn_node->descr[0]=0;
+ ipn_node->portno=IPN_PORTNO_ANY;
+ ipn_node->net=net;
+ ipn_node->dev=NULL;
+ ipn_node->proto_private=NULL;
+ ipn_node->totmsgcount=0;
+ ipn_node->oobmsgcount=0;
+ spin_lock_init(&ipn_node->msglock);
+ INIT_LIST_HEAD(&ipn_node->msgqueue);
+ INIT_LIST_HEAD(&ipn_node->oobmsgqueue);
+ ipn_node->ipn=NULL;
+ init_waitqueue_head(&ipn_node->read_wait);
+ ipn_node->pbp=NULL;
+ return 0;
+}
+
+/* update # of readers and # of writers counters for an ipn network.
+ * This function sends oob messages to nodes requesting the service */
+static void ipn_net_update_counters(struct ipn_network *ipnn,
+ int chg_readers, int chg_writers) {
+ ipnn->numreaders += chg_readers;
+ ipnn->numwriters += chg_writers;
+ if (ipnn->mtu >= sizeof(struct numnode_oob))
+ {
+ struct msgpool_item *ipn_msg=ipn_msgpool_alloc(ipnn);
+ if (ipn_msg) {
+ struct numnode_oob *oob_msg=(struct numnode_oob *) (ipn_msg->data);
+ struct ipn_node *ipn_node;
+ ipn_msg->len=sizeof(struct numnode_oob);
+ oob_msg->level=IPN_ANY;
+ oob_msg->tag=IPN_OOB_NUMNODE_TAG;
+ oob_msg->numreaders=ipnn->numreaders;
+ oob_msg->numwriters=ipnn->numwriters;
+ list_for_each_entry(ipn_node, &ipnn->connectqueue, nodelist) {
+ if (ipn_node->flags & IPN_NODEFLAG_OOB_NUMNODES)
+ ipn_proto_oobsendmsg(ipn_node, ipn_msg);
+ }
+ ipn_msgpool_put(ipn_msg, ipnn);
+ }
+ }
+
+/* flush pending messages (for close and shutdown RCV) */
+static void ipn_flush_rcvqueue(struct ipn_node *ipn_node)
+{
+ struct ipn_network *ipnn=ipn_node->ipn;
+ spin_lock(&ipn_node->msglock);
+ while (!list_empty(&ipn_node->msgqueue)) {
+ struct msgitem *msgitem=
+ list_first_entry(&ipn_node->msgqueue, struct msgitem, list);
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ list_del(&msgitem->list);
+ ipn_node->totmsgcount--;
+ ipn_msgpool_put(msgitem->msg,ipnn);
+ kmem_cache_free(ipn_msgitem_cache,msgitem);
+ }
+ spin_unlock(&ipn_node->msglock);
+}
+
+/* flush pending oob messages (for socket close) */
+static void ipn_flush_oobrecvqueue(struct ipn_node *ipn_node)
+{
+ struct ipn_network *ipnn=ipn_node->ipn;
+ spin_lock(&ipn_node->msglock);
+ while (!list_empty(&ipn_node->oobmsgqueue)) {
+ struct msgitem *msgitem=
+ list_first_entry(&ipn_node->oobmsgqueue, struct msgitem, list);
+ list_del(&msgitem->list);
+ ipn_node->totmsgcount--;
+ ipn_node->oobmsgcount--;
+ ipn_msgpool_put(msgitem->msg,ipnn);
+ kmem_cache_free(ipn_msgitem_cache,msgitem);
+ }
+ spin_unlock(&ipn_node->msglock);
+}
+
+/* Terminate node. The node is "logically" terminated. */
+static int ipn_terminate_node(struct ipn_node *ipn_node)
+{
+ struct ipn_network *ipnn=ipn_node->ipn;
+ if (ipnn) {
+ if (down_interruptible(&ipnn->ipnn_mutex))
+ return -ERESTARTSYS;
+ if (ipn_node->portno >= 0) {
+ ipn_protocol_table[ipnn->protocol]->ipn_p_predelport(ipn_node);
+ ipnn->connport[ipn_node->portno]=NULL;
+ }
+ list_del(&ipn_node->nodelist);
+ ipn_flush_recvqueue(ipn_node);
+ ipn_flush_oobrecvqueue(ipn_node);
+ if (ipn_node->portno >= 0) {
+ ipn_protocol_table[ipnn->protocol]->ipn_p_delport(ipn_node);
+ ipn_node->ipn=NULL;
+ ipn_net_update_counters(ipnn,
+ (ipn_node->shutdown & RCV_SHUTDOWN)?0:-1,
+ (ipn_node->shutdown & SEND_SHUTDOWN)?0:-1);
+ up(&ipnn->ipnn_mutex);
+ if (ipn_node->dev)
+ ipn_netdev_close(ipn_node);
+ }
+ /* No more network elements */
+ if (atomic_dec_and_test(&ipnn->refcnt))
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ {
+ ipn_protocol_table[ipnn->protocol]->ipn_p_delnet(ipnn);
+ ipn_remove_network(ipnn);
+ ipn_protocol_table[ipnn->protocol]->refcnt--;
+ if (ipnn->dentry) {
+ dput(ipnn->dentry);
+ mntput(ipnn->mnt);
+ }
+ module_put(THIS_MODULE);
+ if (ipnn->msgpool_cache)
+ kmem_cache_destroy(ipnn->msgpool_cache);
+ if (ipnn->connport)
+ kfree(ipnn->connport);
+ kmem_cache_free(ipn_network_cache, ipnn);
+ }
+ }
+ if (ipn_node->pbp) {
+ kfree(ipn_node->pbp);
+ ipn_node->pbp=NULL;
+ }
+ ipn_node->shutdown = SHUTDOWN_XMASK;
+ return 0;
+ }
+
+ /* release of a socket */
+ static int ipn_release (struct socket *sock)
+ {
+ struct ipn_sock *ipn_sk=(struct ipn_sock *)sock->sk;
+ struct ipn_node *ipn_node=ipn_sk->node;
+ int rv;
+ if (down_interruptible(&ipn_glob_mutex))
+ return -ERESTARTSYS;
+ if (ipn_node->flags & IPN_NODEFLAG_PERSIST) {
+ ipn_node->flags &= ~IPN_NODEFLAG_INUSE;
+ rv=0;
+ } else {
+ rv=ipn_terminate_node(ipn_node);
+ if (rv==0)
+ kmem_cache_free(ipn_node_cache,ipn_node);
+ }
+ if (rv==0)
+ sock_put((struct sock *) ipn_sk);
+ up(&ipn_glob_mutex);
+ return rv;
+ }
+
+ /* _set persist, change the persistence of a node,
+ * when persistence gets cleared and the node is no longer used
+ * the node is terminated and freed.
+ * ipn_glob_mutex must be locked */
+ static int _ipn_setpersist(struct ipn_node *ipn_node, int persist)
```

[PATCH 1/1] IPN: Inter Process Networking

```
+{
+ int rv=0;
+ if (persist)
+ ipn_node->flags |= IPN_NODEFLAG_PERSIST;
+ else {
+ ipn_node->flags &= ~IPN_NODEFLAG_PERSIST;
+ if (!(ipn_node->flags & IPN_NODEFLAG_INUSE)) {
+ rv=ipn_terminate_node(ipn_node);
+ if (rv==0)
+ kmem_cache_free(ipn_node_cache,ipn_node);
+ }
+ }
+ return rv;
+}
+
+/* ipn_setpersist
+ * lock ipn_glob_mutex and call __ipn_setpersist above */
+static int ipn_setpersist(struct ipn_node *ipn_node, int persist)
+{
+ int rv=0;
+ if (ipn_node->dev == NULL)
+ return -ENODEV;
+ if (down_interruptible(&ipn_glob_mutex))
+ return -ERESTARTSYS;
+ rv=__ipn_setpersist(ipn_node,persist);
+ up(&ipn_glob_mutex);
+ return rv;
+}
+
+/* several network parameters can be set by setsockopt prior to bind */
+/* struct pre_bind_parms is a temporary structure connected to ipn_node->pbp
+ * to keep the parameter values. */
+struct pre_bind_parms {
+ unsigned short maxports;
+ unsigned short flags;
+ unsigned short msgpoolsize;
+ unsigned short mtu;
+ unsigned short mode;
+};
+
+/* STD_PARAMS: BITS_PER_LONG nodes, no flags, BITS_PER_BYTE pending msgs,
+ * Ethernet + VLAN MTU*/
+#define STD_BIND_PARAMS {BITS_PER_LONG, 0, BITS_PER_BYTE, 1514, 0x777};
+
+static int ipn_mkname(struct sockaddr_un * sunaddr, int len)
+{
+ if (len <= sizeof(short) || len > sizeof(*sunaddr))
+ return -EINVAL;
+ if (!sunaddr || sunaddr->sun_family != AF_IPN)
+ return -EINVAL;
+ /*
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ * This may look like an off by one error but it is a bit more
+ * subtle. 108 is the longest valid AF_IPN path for a binding.
+ * sun_path[108] doesnt as such exist. However in kernel space
+ * we are guaranteed that it is a valid memory location in our
+ * kernel address buffer.
+ */
+ ((char *)sunaddr)[len]=0;
+ len = strlen(sunaddr->sun_path)+1+sizeof(short);
+ return len;
+ }
+
+
+ /* IPN BIND */
+ static int ipn_bind(struct socket *sock, struct sockaddr *uaddr, int addr_len)
+ {
+     struct sockaddr_un *sunaddr=(struct sockaddr_un *)uaddr;
+     struct ipn_node *ipn_node=((struct ipn_sock *)sock->sk)->node;
+     struct nameidata nd;
+     struct ipn_network *ipnn;
+     struct dentry * dentry = NULL;
+     int err;
+     struct pre_bind_parms parms=STD_BIND_PARMS;
+
+     //printk("IPN bind\n");
+
+     if (down_interruptible(&ipn_glob_mutex))
+         return -ERESTARTSYS;
+     if (sock->state != SS_UNCONNECTED ||
+         ipn_node->ipn != NULL) {
+         err= -EISCONN;
+         goto out;
+     }
+
+     if (ipn_node->protocol >= 0 &&
+         (ipn_node->protocol >= IPN_MAX_PROTO ||
+         ipn_protocol_table[ipn_node->protocol] == NULL)) {
+         err= -EPROTONOSUPPORT;
+         goto out;
+     }
+
+     addr_len = ipn_mkname(sunaddr, addr_len);
+     if (addr_len < 0) {
+         err=addr_len;
+         goto out;
+     }
+
+     /* check if there is already a socket with that name */
+     err = path_lookup(sunaddr->sun_path, LOOKUP_FOLLOW, &nd);
+     if (err) { /* it does not exist, NEW IPN socket! */
+         unsigned int mode;
+         /* Is it everything okay with the parent? */
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ err = path_lookup(sunaddr->sun_path, LOOKUP_PARENT, &nd);
+ if (err)
+ goto out_mknod_parent;
+ /* Do I have the permission to create a file? */
+ dentry = lookup_create(&nd, 0);
+ err = PTR_ERR(dentry);
+ if (IS_ERR(dentry))
+ goto out_mknod_unlock;
+ /*
+ * All right, let's create it.
+ */
+ if (ipn_node->pbp)
+ mode = ipn_node->pbp->mode;
+ else
+ mode = SOCK_INODE(sock)->i_mode;
+ mode = S_IFSOCK | (mode & ~current->fs->umask);
+ err = vfs_mknod(nd.dentry->d_inode, dentry, mode, 0);
+ if (err)
+ goto out_mknod_dput;
+ mutex_unlock(&nd.dentry->d_inode->i_mutex);
+ dput(nd.dentry);
+ nd.dentry = dentry;
+ /* create a new ipn_network item */
+ if (ipn_node->pbp)
+ parms=*ipn_node->pbp;
+ ipnn=kmem_cache_zalloc(ipn_network_cache,GFP_KERNEL);
+ if (!ipnn) {
+ err=-ENOMEM;
+ goto out_mknod_dput_ipnn;
+ }
+ ipnn->connport=kzalloc(parms.maxports * sizeof(struct ipn_node *),GFP_KERNEL);
+ if (!ipnn->connport) {
+ err=-ENOMEM;
+ goto out_mknod_dput_ipnn2;
+ }
+
+ /* module refcnt is incremented for each network, thus
+ * rmmmod is forbidden if there are persistent node */
+ if (!try_module_get(THIS_MODULE)) {
+ err = -EINVAL;
+ goto out_mknod_dput_ipnn2;
+ }
+ memcpy(&ipnn->sunaddr,sunaddr,addr_len);
+ ipnn->mtu=parms.mtu;
+ ipnn->msgpool_cache=kmem_cache_create(ipnn->sunaddr.sun_path,sizeof(struct
msgpool_item)+ipnn->mtu,0,0,NULL);
+ if (!ipnn->msgpool_cache) {
+ err=-ENOMEM;
+ goto out_mknod_dput_putmodule;
+ }
+ INIT_LIST_HEAD(&ipnn->unconnectqueue);
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ INIT_LIST_HEAD(&ipnn->connectqueue);
+ atomic_set(&ipnn->refcnt,1);
+ ipnn->dentry=nd.dentry;
+ ipnn->mnt=nd.mnt;
+ init_MUTEX(&ipnn->ipnn_mutex);
+ ipnn->sunaddr_len=addr_len;
+ ipnn->protocol=ipn_node->protocol;
+ if (ipnn->protocol < 0) ipnn->protocol = 0;
+ ipn_protocol_table[ipnn->protocol]->refcnt++;
+ ipnn->flags=parms.flags;
+ ipnn->numreaders=0;
+ ipnn->numwriters=0;
+ ipnn->maxports=parms.maxports;
+ atomic_set(&ipnn->msgpool_nelem,0);
+ ipnn->msgpool_size=parms.msgpoolsize;
+ ipnn->proto_private=NULL;
+ init_waitqueue_head(&ipnn->send_wait);
+ err=ipn_protocol_table[ipnn->protocol]->ipn_p_newnet(ipnn);
+ if (err)
+ goto out_mknod_dput_putmodule;
+ ipn_insert_network(&ipn_network_table[nd.dentry->d_inode->i_ino & (IPN_HASH_SIZE-1)],ipnn);
+ } else {
+ /* join an existing network */
+ err = vfs_permission(&nd, MAY_EXEC);
+ if (err)
+ goto put_fail;
+ err = -ECONNREFUSED;
+ if (!S_ISSOCK(nd.dentry->d_inode->i_mode))
+ goto put_fail;
+ ipnn=ipn_find_network_byinode(nd.dentry->d_inode);
+ if (!ipnn || (ipnn->flags & IPN_FLAG_TERMINATED))
+ goto put_fail;
+ list_add_tail(&ipn_node->nodelist,&ipnn->unconnectqueue);
+ atomic_inc(&ipnn->refcnt);
+ }
+ if (ipn_node->pbp) {
+ kfree(ipn_node->pbp);
+ ipn_node->pbp=NULL;
+ }
+ ipn_node->ipn=ipnn;
+ ipn_node->flags |= IPN_NODEFLAG_BOUND;
+ up(&ipn_glob_mutex);
+ return 0;
+
+put_fail:
+ path_release(&nd);
+out:
+ up(&ipn_glob_mutex);
+ return err;
+
+out_mknod_dput_putmodule:
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ module_put(THIS_MODULE);
+out_mknod_dput_ipnn2:
+ kfree(ipnn->connport);
+out_mknod_dput_ipnn:
+ kmem_cache_free(ipn_network_cache,ipnn);
+out_mknod_dput:
+ dput(dentry);
+out_mknod_unlock:
+ mutex_unlock(&nd.dentry->d_inode->i_mutex);
+ path_release(&nd);
+out_mknod_parent:
+ if (err== -EEXIST)
+ err=-EADDRINUSE;
+ up(&ipn_glob_mutex);
+ return err;
+}
+
+/* IPN CONNECT */
+static int ipn_connect(struct socket *sock, struct sockaddr *addr,
+ int addr_len, int flags){
+ struct sockaddr_un *sunaddr=(struct sockaddr_un*)addr;
+ struct ipn_node *ipn_node=((struct ipn_sock *)sock->sk)->node;
+ struct nameidata nd;
+ struct ipn_network *ipnn,*previousipnn;
+ int err=0;
+ int portno;
+
+ /* the socket cannot be connected twice */
+ if (sock->state != SS_UNCONNECTED)
+ return EISCONN;
+
+ if (down_interruptible(&ipn_glob_mutex))
+ return -ERESTARTSYS;
+
+ if ((previousipnn=ipn_node->ipn) == NULL) { /* unbound */
+ unsigned char mustshutdown=0;
+ err = ipn_mkname(sunaddr, addr_len);
+ if (err < 0)
+ goto out;
+ addr_len=err;
+ err = path_lookup(sunaddr->sun_path, LOOKUP_FOLLOW, &nd);
+ if (err)
+ goto out;
+ err = vfs_permission(&nd, MAY_READ);
+ if (err) {
+ if (err == -EACCES || err == -EROFS)
+ mustshutdown|=RCV_SHUTDOWN;
+ else
+ goto put_fail;
+ }
+ err = vfs_permission(&nd, MAY_WRITE);
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ if (err) {
+ if (err == -EACCES)
+ mustshutdown|=SEND_SHUTDOWN;
+ else
+ goto put_fail;
+ }
+ mustshutdown |= ipn_node->shutdown;
+ /* if the combination of shutdown and permissions leaves
+ * no abilities, connect returns EACCES */
+ if (mustshutdown == SHUTDOWN_XMASK) {
+ err=-EACCES;
+ goto put_fail;
+ } else {
+ err=0;
+ ipn_node->shutdown=mustshutdown;
+ }
+ if (!S_ISSOCK(nd.dentry->d_inode->i_mode)) {
+ err = -ECONNREFUSED;
+ goto put_fail;
+ }
+ ipnn=ipn_find_network_byinode(nd.dentry->d_inode);
+ if (!ipnn || (ipnn->flags & IPN_FLAG_TERMINATED)) {
+ err = -ECONNREFUSED;
+ goto put_fail;
+ }
+ if (ipn_node->protocol == IPN_ANY)
+ ipn_node->protocol=ipnn->protocol;
+ else if (ipnn->protocol != ipn_node->protocol) {
+ err = -EPROTO;
+ goto put_fail;
+ }
+ path_release(&nd);
+ ipn_node->ipn=ipnn;
+ } else
+ ipnn=ipn_node->ipn;
+
+ if (down_interruptible(&ipnn->ipnn_mutex)) {
+ err=-ERESTARTSYS;
+ goto out;
+ }
+ portno = ipn_protocol_table[ipnn->protocol]->ipn_p_newport(ipn_node);
+ if (portno >= 0 && portno<ipnn->maxports) {
+ sock->state = SS_CONNECTED;
+ ipn_node->portno=portno;
+ ipnn->connport[portno]=ipn_node;
+ if (!(ipn_node->flags & IPN_NODEFLAG_BOUND)) {
+ atomic_inc(&ipnn->refcnt);
+ list_del(&ipn_node->nodelist);
+ }
+ list_add_tail(&ipn_node->nodelist,&ipnn->connectqueue);
+ ipn_net_update_counters(ipnn,
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ (ipn_node->shutdown & RCV_SHUTDOWN)?0:1,
+ (ipn_node->shutdown & SEND_SHUTDOWN)?0:1);
+ } else {
+ ipn_node->ipn=previousipnn; /* undo changes on ipn_node->ipn */
+ err=-EADDRNOTAVAIL;
+ }
+ up(&ipnn->ipnn_mutex);
+ up(&ipn_glob_mutex);
+ return err;
+
+put_fail:
+ path_release(&nd);
+out:
+ up(&ipn_glob_mutex);
+ return err;
+}
+
+static int ipn_getname(struct socket *sock, struct sockaddr *uaddr,
+ int *uaddr_len, int peer) {
+ struct ipn_node *ipn_node=((struct ipn_sock *)sock->sk)->node;
+ struct ipn_network *ipnn=ipn_node->ipn;
+ struct sockaddr_un *sunaddr=(struct sockaddr_un *)uaddr;
+ int err=0;
+
+ if (down_interruptible(&ipn_glob_mutex))
+ return -ERESTARTSYS;
+ if (ipnn) {
+ *uaddr_len = ipnn->sunaddr_len;
+ memcpy(sunaddr,&ipnn->sunaddr,*uaddr_len);
+ } else
+ err = -ENOTCONN;
+ up(&ipn_glob_mutex);
+ return err;
+}
+
+/* IPN POLL */
+static unsigned int ipn_poll(struct file *file, struct socket *sock,
+ poll_table *wait) {
+ struct ipn_node *ipn_node=((struct ipn_sock *)sock->sk)->node;
+ struct ipn_network *ipnn=ipn_node->ipn;
+ unsigned int mask=0;
+
+ if (ipnn) {
+ poll_wait(file,&ipn_node->read_wait,wait);
+ if (ipnn->flags & IPN_FLAG_LOSSLESS)
+ poll_wait(file,&ipnn->send_wait,wait);
+ /* POLLIN if recv succeeds,
+ * POLL{PRI,RDNORM} if there are {oob,non-oob} messages */
+ if (ipn_node->totmsgcount > 0) mask |= POLLIN;
+ if (!(list_empty(&ipn_node->msgqueue))) mask |= POLLRDNORM;
+ if (!(list_empty(&ipn_node->oobmsgqueue))) mask |= POLLPRI;
```

[PATCH 1/1] IPN: Inter Process Networking

```
+ if (!(ipnn->flags & IPN_FLAG_LOSSLESS)) |
+ (atomic_read(&ipnn->msgpool_nelem) < ipnn->msgpool_size))
+ mask |= POLLOUT | POLLWRNORM;
+ }
+ return mask;
+}
+
+/* connect netdev (from ioctl). connect a bound socket to a
+ * network device TAP or GRAB */
+static int ipn_connect_netdev(struct socket *sock,struct ifreq *ifr)
+{
+ int err=0;
+ struct ipn_node *ipn_node=((struct ipn_sock *)sock->sk)->node;
+ struct ipn_network *ipnn=ipn_node->ipn;
+ if (!capable(CAP_NET_ADMIN))
+ return -EPERM;
+ if (sock->state != SS_UNCONNECTED)
+ return -EISCONN;
+ if (!ipnn)
+ return -ENOTCONN; /* Maybe we need a different error for "NOT BOUND" */
+ if (down_interruptible(&ipn_glob_mutex))
+ return -ERESTARTSYS;
+ if (down_interruptible(&ipnn->ipnn_mutex)) {
+ up(&ipn_glob_mutex);
+ return -ERESTARTSYS;
+ }
+ ipn_node->dev=ipn_netdev_alloc(ipn_node->net,ifr->ifr_flags,ifr->ifr_name,&err);
+ if (ipn_node->dev) {
+ int portno;
+ portno = ipn_protocol_table[ipnn->protocol]->ipn_p_newport(ipn_node);
+ if (portno >= 0 && portno<ipnn->maxports) {
+ sock->state = SS_CONNECTED;
+ ipn_node->portno=portno;
+ ipn_node->flags |= ifr->ifr_flags & IPN_NODEFLAG_DEVMASK;
+ ipnn->connport[portno]=ipn_node;
+ err=ipn_netdev_activate(ipn_node);
+ if (err) {
+ sock->state = SS_UNCONNECTED;
+ ipn_protocol_table[ipnn->protocol]->ipn_p_delport(ipn_node);
+ ipn_node->dev=NULL;
+ ipn_node->portno= -1;
+ ipn_node->flags &= ~IPN_NODEFLAG_DEVMASK;
+ ipnn->connport[portno]=NULL;
+ } else {
+ ipn_protocol_table[ipnn->protocol]->ipn_p_postnewport(ipn_node);
+ list_del(&ipn_node->nodelist);
+ list_add_tail(&ipn_node->nodelist,&ipnn->connectqueue);
+ }
+ } e
```