ESPRIT / HPCN
PROJECT 29737 – HPGIN
High Performance Gigabit I_2O Networking Software

Final Software
HPGIN-Linux / Task D4

University of Helsinki
Department of Computer Science

Date: 15.01.2001
Document-Id: UHEL.01.01.01-DR-D4

ESPRIT / HPCN
PROJECT 29737 – HPGIN
High Performance Gigabit I_2O Networking Software

Final Software
HPGIN-Linux / Task D4

Written by:	Juha Sievänen, Auvo Häkkinen
Organization:	University of Helsinki, Department of Computer Science
Date:	15.01.2001
Delivered by:	**Frank Hohmann**
Organization:	SysKonnect GmbH
Date:	15.01.2001

Document Id: UHEL.01.01.01-DR-D4

CHANGE HISTORY

This document has been submitted in draft to all contractors and has been approved. This is the updated version of the previous documents describing HPGIN-Linux Software prototypes.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

1 PURPOSE

This document contains the Deliverable Report of Task D4 of the Esprit Project 29737. The report is addressed to the EC Project Officer. A complete list of the planned deliverables, sorted by months, is included in the Project Programme (Annex I to the *Cost Reimbursement Contract* [1] , page 27, form 5.1).

2 SUMMARY

Task deliverable of Task D4 consists of the HPGIN-Linux software. The purpose of the deliverable report is to explain how to use this software. The technical solution is explained in *HPGIN-Linux: Specification of the Software Package D* [3].

3 VALIDITY

The contents of this document are applicable to the partners of the consortium of the ESPRIT/HPCN project 29737, HPGIN.

4 REFERENCES

[1] EP 29737 – ESPRIT/HPGIN. *Cost Reimbursement Contract*. Annex I – "Project Programme" - Adjustment. March 1999.

[2] I_2O Special Interest Group. *Intelligent I/O (I_2O) Architecture Specification* Version 1.5, March 1997.

[3] University of Helsinki, HPGIN-Linux. *Specification of the Software Package D, Task D1*, March 2000.

[4] EP 29737 – ESPRIT/HPGIN. *Test Reports, Task E3*, January 2001.

5 TERMS AND DEFINITIONS

For terms and definitions used in this document, see *HPGIN-Linux: Specification of the Software Package D* [3].

6 DESCRIPTION OF DELIVERABLE

Task D4 covers the implementation of the HPGIN-Linux software. It includes the I_2O subsystem initialization, message passing, resource management, OSM for LAN class networking and a configuration interface to configure the IOPs and DDMs. The deliverable consist of two separate software packages. The I_2O extension is embedded into kernel version 2.4 and is distributed together with the kernel. The Configuration Utility is a separate web-based software.

The kernel is a part of every Linux distribution and it can also be obtained separately e.g. from ftp://ftp.kernel.org/pub/linux/kernel/v2.4/. The web-based Configuration Utility can be loaded from page http://www.cs.helsinki.fi/reasearch/hpgin/ at the University of Helsinki. The technical solutions are defined in *HPGIN-Linux: Specification of the Software Package D* [3] and the test results are reported in *HPGIN-Linux: Test Report* [4].

The following sections give an introduction to how to use the Linux I_2O environment.

7 TEST INSTALLATION

The installation used in examples throughout this report is presented below in Figure 1.

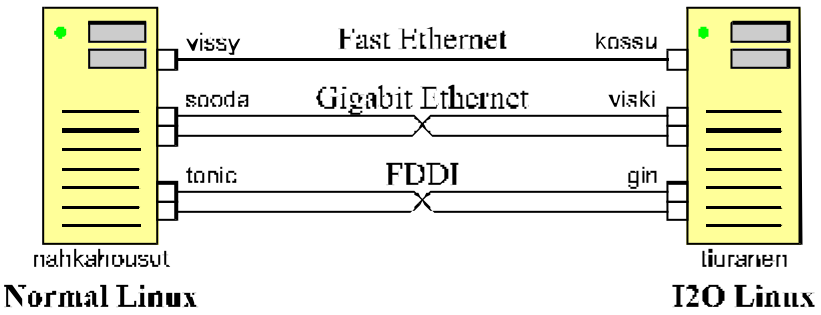

Figure 1. Test installation used during the HPGIN-project.

8 LINUX I$_2$O SUPPORT

The following sections show how to configure I$_2$O support into Linux, and how to use I$_2$O modules.

8.1 Configuring I$_2$O Support into Kernel

The I$_2$O support is configured into kernel by a Linux configurating command, e.g by `make xconfig` (Figure 2). The I$_2$O support can be based on dynamically loadable modules (**m**) or it can be compiled into the kernel (**y**). In the following sections we present the module approach.

The Ethernet part of the LAN OSM is always compiled into the LAN OSM. If you want to use FDDI or Token Ring I$_2$O cards, you should configure the kernel to support FDDI devices and Token Ring devices, respectively.

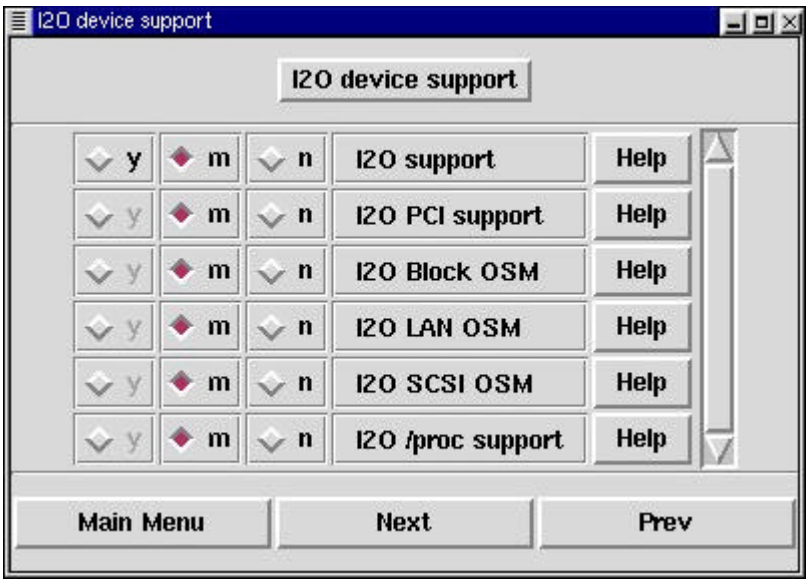

Figure 2. Kernel compilation options for I$_2$O modules

After configuring I$_2$O support into the kernel, the installation is continued normally. Refer e.g to *Linux Kernel Documentation* (file `linux/README`) for details of kernel compilation and installation.

8.2 Loading and Removing I$_2$O Modules

The initialization of the I$_2$O environment is fully automated. For example, you just activate the I$_2$O LAN interface (e.g `ifup eth2`) or start the Configuration Utility in a Web browser (`http://<server>/cgi-bin/I2O`), and all modules needed are loaded automatically. You can also set up the I$_2$O environment by loading modules separately with the `modprobe <module>` command. The dependencies are defined so that all needed modules are loaded.

To make it easier to use I$_2$O environment, you might want to set aliases to modules and module parameters in file `/etc/modules.conf`. In the following example there are three I$_2$O network interfaces (fddi0, eth1, and eth2)

```
gin$ cat /etc/modules.conf
alias char-major-10-166 i2o_config

alias fddi0     i2o_lan         # I2O LAN OSM - FDDI
alias eth1      i2o_lan         # I2O LAN OSM - FE
alias eth2      i2o_lan         # I2O LAN OSM - GE

options i2o_lan max_buckets_out=64 bucketthresh=9 rx_copybreak=128
```

You can get a list of loaded modules by using command `lsmod`.

```
gin$ ifup eth2
gin$ lsmod
Module                  Size  Used by
i2o_lan                13176   1 (autoclean)
i2o_core               19196   0 (autoclean) [i2o_lan]
i2o_pci                 2284   1 (autoclean) [i2o_core]
nfs                    34636   2 (autoclean)
lockd                  37620   0 (autoclean) [nfs]
sunrpc                 68768   1 (autoclean) [nfs lockd]
```

You can remove modules from the kernel by hand with command `modprobe –r <module>`. Unused modules might be removed also automatically by using a cron job.

8.3 I$_2$O Debugging Information

During the I$_2$O subsystem development and testing quite a lot of I$_2$O debug information was written into the system log (e.g. `/var/log/messages`). The printouts are preserved inside the code, and they can be turned on by uncommenting setting the #define DRIVERDEBUG sentence and re-compiling the module.

For example, **i2o_pci** and **i2o_core** modules write the following messages into system log during the initialization. As you can see there is one IOP and three LAN cards in this environment.

```
i2o: Checking for PCI I2O controllers...
i2o: I2O controller on bus 0 at 105.
i2o: PCI I2O controller at 0xE0000000 size=33554432
i2o/iop0: Installed at IRQ19
i2o: 1 I2O controller found and installed.
Activating I2O controllers
This may take a few minutes if there are many devices
i2o/iop0: LCT has 7 entries.
Target ID 0.
     Vendor: Wind River Sys.     Device: IxWorks        Rev: 0201
     Class: Executive            Subclass: 0x0001               Flags: Pm
Target ID 8.
     Vendor: SysKonnect          Device: SK-NET Gigabit         Rev: v1.0
     Class: LAN Interface        Subclass: 0x0030               Flags: Pm
Target ID 10.
     Vendor: SysKonnect          Device: SK-NET FDDI PCI        Rev: 1.05DMEM
     Class: Device Driver Module Subclass: 0x0020               Flags: Pm
Target ID 11.
     Vendor: SysKonnect          Device: SK-NET FDDI PCI        Rev: N/A
     Class: LAN Interface        Subclass: 0x0060               Flags: Pm
Target ID 12.
     Vendor: Intel Corp          Device: EtherExpress PRO       Rev: 0.51
     Class: Device Driver Module Subclass: 0x0020               Flags: Pm
Target ID 13.
     Vendor: SysKonnect          Device: SK-DDM Gigabit         Rev: 1.03 MEM
     Class: Device Driver Module Subclass: 0x0020               Flags: Pm
Target ID 14.
     Vendor: Intel Corp          Device: EtherExpress PRO       Rev: 0.51
     Class: LAN Interface        Subclass: 0x0030               Flags: Pm
```

9 CONFIGURATION UTILITY

The Configuration Utility (Figure 3 and Figure 4) implements the web-based user interface to the configuration dialogue, software management and configuration validation. The Configuration Utility is a collection of CGI programs that are used by a web browser. The main CGI program (I2O) generates an HTML page that has links to each IOP's home page on the server. The CGI programs implement the user space interface to the subsystem. The user's input is translated into I_2O request and is delivered to the IOP. The IOP generates replies in html-format from the user's input. The utility has facilities to use configuration dialogue, to manage software modules on the IOP, and to validate a configuration of an IOP.

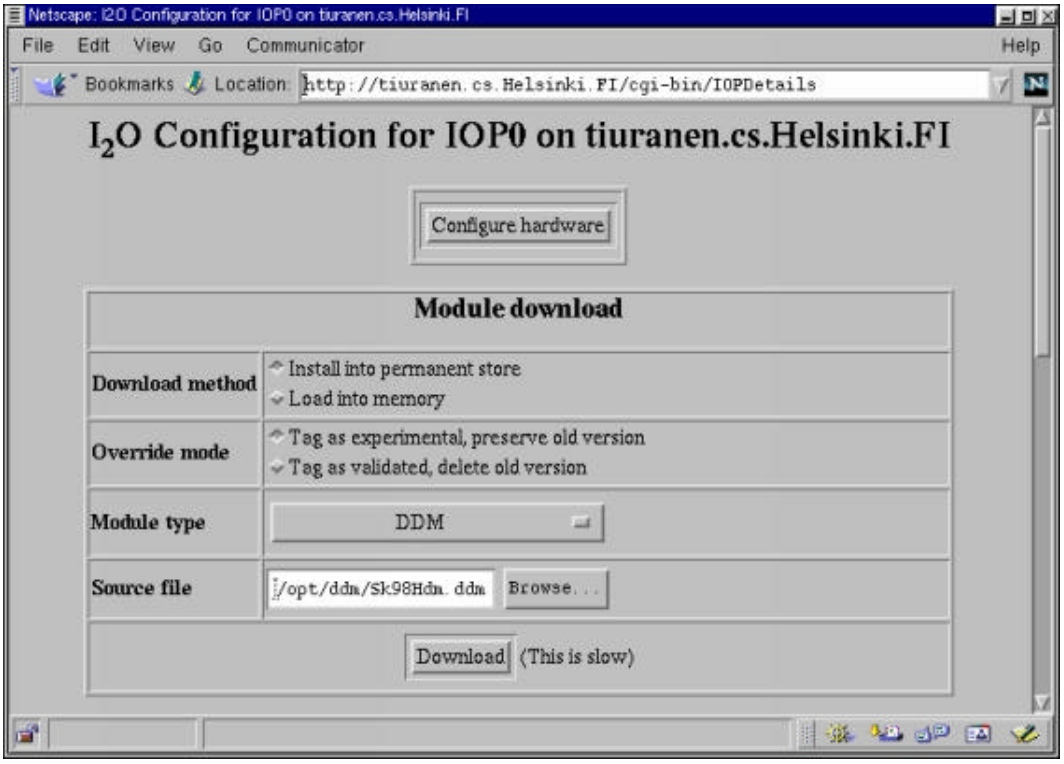

Figure 3. Configuration utility interface – download.

9.1 Configuration Dialogue

The configuration dialogue lets the user to manage the hardware-specific operating parameters of the IOP. You can start the configuration dialoque whenever you need. If there are more than one IOP in the system, you have to choose the IOP from the list. If the IOP is requesting for the configuration (f.g. the DDM is not loaded), the text *"requesting configuration"* will be added next to the IOP's number.

The dialogue constitutes of HTML pages generated by the IOP, and they are just show by the browser. First the IOP's home page (page 0) is fetched and shown in the browser. Theis page contains configuration values and links to pages of all devices on the specific IOP.

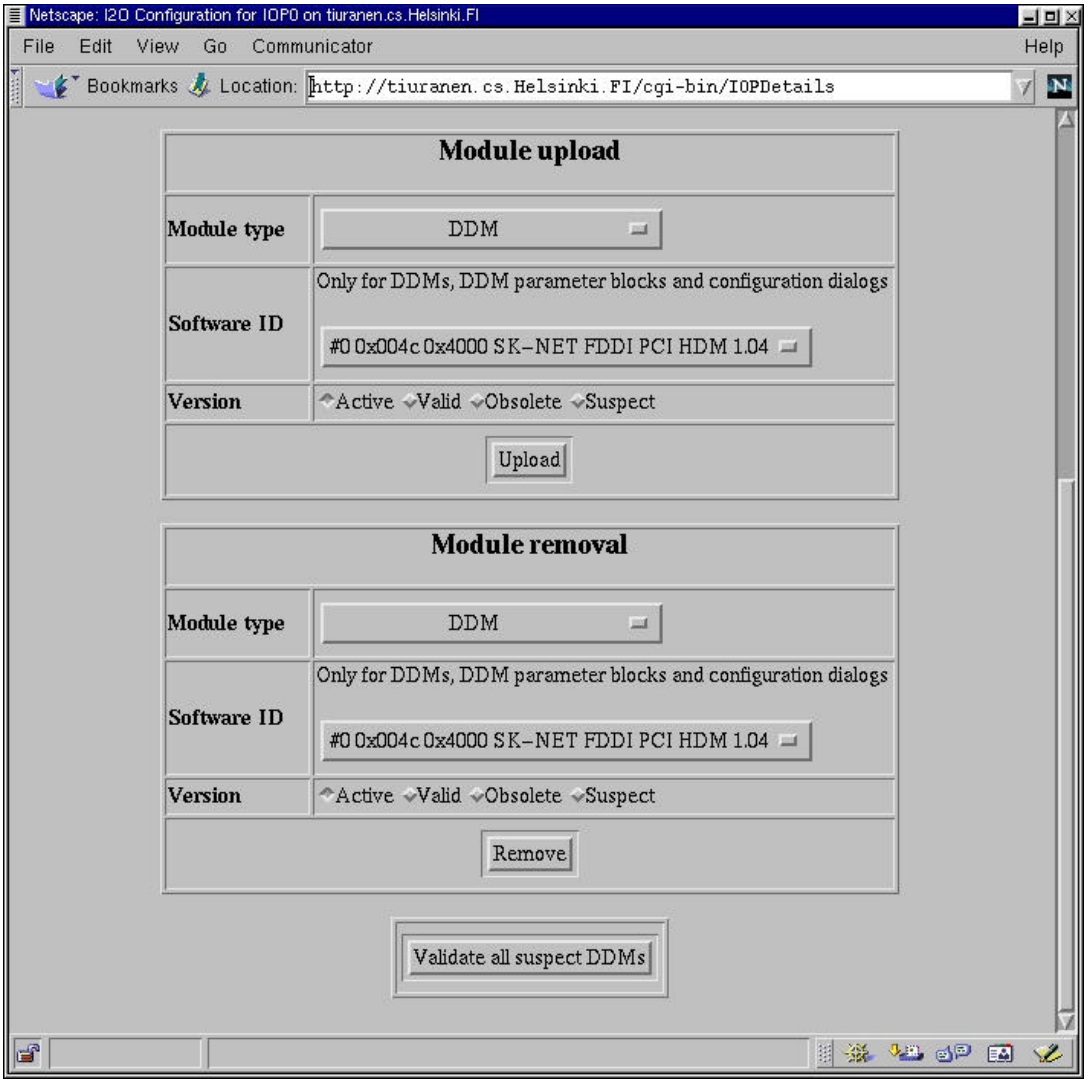

Figure 4. Configuration utility – upload, removal, and validation.

9.2 Software Management

Downloading (installing) a DDM stores the module's executable code into IOP's permanent store so that it can be loaded next time the IOP initializes. Downloading includes placing the module's executable code into IOP's main memory and invoking it. The user has to select the module type, give the path to the file consisting the module, select the download method (Install/Load) and select the override mode (Experimental/Validated) and finally push the Download button. If the override mode is *validated*, the module replaces immediately the previous version of the module in memory. In that case, the module is tagged as validated and the previous version is deleted from the permanent store. If the module is loaded as *experimental*, and seems to work properly it should be validated before the IOP is rebooted. When the DDM is downloaded, the IOP creates an empty parameter block in caseit is not supplied within the module. The IOP calls the module's initialization routine with a pointer to the module parameter block. If the parameter block is empty, the module can request configuration dialogue to setup its operating parameters.

Uploading a module from the IOP permanent store to local disk constitutes of selecting the type and version of the module to be uploaded and selecting the correct module. There are four different versions of modules: Active, Valid, Obsolete, and Suspect. The default is to upload currectly active module.

To **remove** a module from the permanent store, the user needs to do same selections as in module upload.

9.3 Configuration Validation

When a module is downloaded it is tagged as experimental. When the module is loaded into memory for the first time, the IOP changes the tag to suspect. The configuration validation tags the module (and its parameter block) to be valid. When a replacement module is installed, the existing module is tagged old and is not loaded any more. If the experimental module is validated, then the old module is removed. Otherwise, the IOP loads and tags the old module as validated next time IOP boots. The suspect module is tagged as rejected and not loaded. The IOP may delete rejected modules from its permanent store at any time.

To validate the configuration the user needs to select the IOP (if there are more than one) and to push the *Validate all suspect DDMs* button.

10 LAN OSM

The LAN OSM implements the I$_2$O interface to the local area networks. Currently the LAN OSM is able to handle Ethernet, FDDI, Token Ring, and Fibre Channel subclasses (although we have been able to test only FDDI and Ethernet). AnyLAN subclass has not been implemented due to lack of existing AnyLAN I$_2$O hardware.

When the LAN OSM (i.e. the **i2o_lan** module) is loaded, it assigns kernel interfaces to all I$_2$O LAN cards. The Linux network interface has not been changed and therefore all conventional interfaces and drivers will work as well.

10.1 Activating the interface

The interface can be activated and deactivated with `ifconfig` or `ifup/ifdown` scripts. In the following listing `ifconfig` is used to check statistics, and `ping` is used to check that the packets sent reach their destination.

```
gin$ ifconfig fddi0
fddi0     Link encap:UNSPEC  HWaddr 00-00-5A-40-E3-24-00-FA-00-00-00-00-00-00-
00-00
          inet addr:192.168.110.1  Bcast:192.168.110.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:4470  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100

gin$ cat /etc/hosts
# I2O FDDI (tiuranen <-> nahkahousut)
192.168.110.1   gin.hpgin.cs.Helsinki.FI   gin
192.168.110.3   tonic.hpgin.cs.Helsinki.FI tonic

# I2O Gigabit Ethernet (tiuranen <-> nahkahousut)
192.168.111.1   kossu.hpgin.cs.Helsinki.FI kossu
192.168.111.3   sooda.hpgin.cs.Helsinki.FI sooda

# I2O Fast Ethernet (tiuranen <-> nahkahousut)
192.168.112.1   viski.hpgin.cs.Helsinki.FI viski
192.168.112.3   vissy.hpgin.cs.Helsinki.FI vissy

gin$ ping -c5 tonic
PING tonic.hpgin.cs.Helsinki.FI (192.168.110.3) from 192.168.110.1 : 56(84)
bytes of data.
64 bytes from 192.168.110.3: icmp_seq=0 ttl=255 time=52.3 ms
64 bytes from 192.168.110.3: icmp_seq=1 ttl=255 time=0.3 ms
64 bytes from 192.168.110.3: icmp_seq=2 ttl=255 time=0.3 ms
64 bytes from 192.168.110.3: icmp_seq=3 ttl=255 time=0.3 ms
64 bytes from 192.168.110.3: icmp_seq=4 ttl=255 time=0.3 ms

--- tonic.hpgin.cs.Helsinki.FI ping statistics ---
```

```
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.3/10.7/52.3 ms

gin$ ping -f -c2000 tonic
ping -f -c2000 tonic
PING tonic.hpgin.cs.Helsinki.FI (192.168.110.3) from 192.168.110.1 : 56(84)
bytes of data.
.........
--- tonic.hpgin.cs.Helsinki.FI ping statistics ---
2009 packets transmitted, 2000 packets received, 0% packet loss
round-trip min/avg/max = 0.3/1.1/2.3 ms
```

10.2 Debugging information

Normally, as shown in the following listing, only registering of LAN devices is written into the log.

```
I2O LAN OSM (c) 1999 University of Helsinki
eth1: I2O LAN device registered, subclass = 0x00000030, unit = 0, tid = 14.
fddi0: I2O LAN device registered, subclass = 0x00000060, unit = 1, tid = 11.
eth2: I2O LAN device registered, subclass = 0x00000030, unit = 2, tid = 8.
```

When the **i2o_lan** module is compiled to produce more debugging information, you will got a lot more messages into log:

```
fddi0: LAN_RECEIVE_POST, SUCCESS / SUCCESS.
fddi0: Buckets_remaining = 9, bucket_count = 9
fddi0: Incoming packet (105 bytes) delivered to upper level.
fddi0: Packet (105 bytes) sent to network.
fddi0: LAN_PACKET_SEND, SUCCESS / SUCCESS.
fddi0: Request skb freed (trl_count=1).
fddi0: LAN_RECEIVE_POST, SUCCESS / SUCCESS.
fddi0: Buckets_remaining = 8, bucket_count = 8
fddi0: Incoming packet (105 bytes) delivered to upper level.
fddi0: Sending 9 buckets (size 4494) to LAN HDM.
...
fddi0: LAN_RECEIVE_POST, ABORT_NO_DATA_TRANSFER / CANCELED.
fddi0: Releasing unused sk_buff c753e0e0
fddi0: LAN_SUSPEND, SUCCESS / SUCCESS.
```

11 PROC FILESYSTEM

The proc filesystem can be used to read the configuration information and to set parameter values for the LAN OSM, DDMs and IOPs. Directory `/proc/i2o` contains subdirectories for all IOPs in the system. The directory for each IOP contains directories for all its DDMs, and files for its parameter groups.

In this section we present how to get and set parameters for IOPs, DDMs and LAN OSM using plain shell commands `ls`, `cat` and `echo`.

11.1 Getting LAN parameter group information

The contents of a subdirectory of an IOP depend on the class of the device. In this example the IOP has seven devices assigned for TIDs from 0x000 to 0x00e. The directory has also five entries for executive parameter groups. Table 1 explains the information given in these files.

```
gin$ ls -F /proc/i2o/iop0/
0x000/     0x00a/     0x00c/     0x00e/      driver_store   hrt  lct
0x008/     0x00b/     0x00d/     ddm_table   drivers_stored hw   status
```

File	Meaning
`ddm_table`	Executive group 0003h – Executing DDM List
`driver_store`	Executive group 0004h – Driver Store
`drivers_stored`	Executive group 0005h – Driver Store Table
`Hrt`	Hardware Resource Table
`Hw`	Executive group 0000h – IOP Hardware
`Lct`	Local Configuration Table
`status`	Status block of the IOP

Table 1. proc entries for Executive Parameter Groups.

For example, the status block of the first IOP is obtained by using cat command:

```
gin$ cat /proc/i2o/iop0/status
Organization ID        : 0xaaaa
IOP ID                 : 0x000
Host Unit ID           : 0x0000
Segment Number         : 0x000
I2O version            : 1.5
IOP State              : OPERATIONAL
Messenger Type         : Memory mapped
Inbound Frame Size     : 128 bytes
Max Inbound Frames     : 4096
Current Inbound Frames : 2048
```

```
Max Outbound Frames    : 4096
Product ID             : ASUS P2B-D2
Expected LCT Size      : 300 bytes
IOP Capabilities
    Context Field Size Support : Supports only 32-bit context fields
    Current Context Field Size : Supports only 32-bit context fields
    Inbound Peer Support       : Supported
    Outbound Peer Support      : Supported
    Peer to Peer Support       : Not supported
Desired private memory size   : 2048 kB
Allocated private memory size : 2048 kB
Private memory base address   : 0xffe00000
Desired private I/O size      : 0 kB
Allocated private I/O size    : 0 kB
Private I/O base address      : 0x00000000
```

The directory `/proc/i2o/iop0/0x000` contains information about Executive DDM (IRTOS) of the first IOP.

```
gin$ ls -F /proc/i2o/iop0/0x000/
authorized_users  ddm_identity  groups    priv_msgs  sgl_limits  users
claimed           dev_identity  phys_dev  sensors    user_info

gin$ cat /proc/i2o/iop0/0x000/dev_identity
Device Class  : Executive
Owner TID     : 0xfff
Parent TID    : 0x000
Vendor info   : Wind River Sys.
Product info  : IxWorks
Description   : 1.1 FCS
Product rev.  : 0201
Serial number : 0x04
```

Other DDMs are given a permanent TID by the IRTOS when they are installed and loaded for the first time. DDM directories have always some generic parameter group entries, which are the same for every device and DDM. These files are presented in Table 2.

File	Meaning
`groups`	Generic group F000h – Params Descriptor
`phys_dev`	Generic group F001h – Physical Device Table
`claimed`	Generic group F002h – Claimed Table
`users`	Generic group F003h – User Table
`priv_msgs`	Generic group F005h – Private message extensions
`authorized_users`	Generic group F006h – Authorized User Table
`dev_identity`	Generic group F100h - Device Identity
`ddm_identity`	Generic group F101h - DDM Identity

user_info	Generic group F102h - User Information
sgl_limits	Generic group F103h - SGL Operating Limits
sensors	Generic group F200h – Sensors

Table 2. Entries for Generic parameter Groups.

In our environment the directory /proc/i2o/iop0/0x00d contains information about the SysKonnect Gigabit DDM (TID=0x00d) and the actual LAN interface (TID=0x008) information for that DDM can be found in directory /proc/i2o/iop0/0x008. The contents of these directories and device indentification information are shown below.

```
gin$ ls -F /proc/i2o/iop0/0x00d/
authorized_users  ddm_identity  groups    priv_msgs  sgl_limits  users
claimed           dev_identity  phys_dev  sensors    user_info

gin$ cat /proc/i2o/iop0/0x00d/dev_identity
Device Class  : Device Driver Module
Owner TID     : 0xfff
Parent TID    : 0x000
Vendor info   : SysKonnect
Product info  : SK-DDM Gigabit
Description   : I2O LAN HDM
Product rev.  : 1.03 MEM
Serial number : LAN-48 MAC address @ 04:C0:A4:00:A2:C0

gin$ ls -F /proc/i2o/iop0/0x008/
authorized_users  lan_alt_addr    lan_mac_addr        lan_tx_info sgl_limits
claimed           lan_batch_ctrl  lan_mcast_addr      phys_dev    user_info
ddm_identity      lan_dev_info    lan_media_operation priv_msgs   users
dev_identity      lan_eth_stats   lan_operation       sensors
groups            lan_hist_stats  lan_rx_info         settings

gin$ cat /proc/i2o/iop0/0x008/dev_identity
Device Class  : LAN Interface
Owner TID     : 0x001
Parent TID    : 0x00d
Vendor info   : SysKonnect
Product info  : SK-NET Gigabit
Description   : LAN Adapter
Product rev.  : v1.0
Serial number : LAN-48 MAC address @ 04:00:00:5A:98:82
```

In addition to the generic parameter groups, device directories have also entries for device class. Table 3. lists the LAN class device entries.

File	Meaning
lan_dev_info	LAN group 0000h - Device info
lan_mac_addr	LAN group 0001h - MAC address table

lan_mcast_addr	LAN group 0002h - Multicast MAC address table
lan_batch_ctrl	LAN group 0003h - Batch Control
lan_operation	LAN group 0004h - LAN Operation
lan_media_operation	LAN group 0005h - Media operation
lan_alt_addr	LAN group 0006h - Alternate address
lan_tx_info	LAN group 0007h - Transmit info
lan_rx_info	LAN group 0008h - Receive info
lan_hist_stats	LAN group 0100h - LAN Historical statistics LAN group 0180h - Supported Optional Historical Statistics LAN group 0182h - Optional Non Media Specific Transmit Historical Statistics LAN group 0183h - Optional Non Media Specific Receive Historical Statistics
lan_opt_chksum_stats	LAN group 0184h - Optional non media specific LAN statistics for checksum generation/validation

Table 3. Parameter group entries for LAN device.

F.g. the lan_dev_info group contains information about the LAN interface.

```
gin$ cat /proc/i2o/iop0/0x008/lan_dev_info
LAN Type            : Ethernet, physical LAN port, simplex
Address format      : IEEE 48bit
State               : Operational
Min packet size     : 60
Max packet size     : 1514
HW address          : 00:00:5A:98:82:74:00:00
Max Tx wire speed   : 1000000000 bps
Max Rx wire speed   : 1000000000 bps
Min SDU packet size : 0x00000000
Max SDU packet size : 0x00000000
```

There are also some specific entries for each device subclass. LAN subclass entries are given in Table 4.

File	Meaning
lan_eth_stats	LAN group 0200h – Required Ethernet Statistics LAN group 0280h – Supported Ethernet Historical Statistics LAN group 0281h - Optional Ethernet Historical Statistics
lan_tr_stats	LAN group 0300h – Required Token Ring Statistics
lan_fddi_stats	LAN group 0400h – Required FDDI Statistics

Table 4. Entries for LAN subclass parameter groups.

11.2 Setting OSM and DDM parameters

LAN OSM parameters can be set on the fly using the proc interface. The file `settings` lists parameters that can be set via the proc interface. The parameter name and the new value are simply echoed to the file. In the following example we show the combined settings for LAN OSM and the SysKonnect Gigabit Ethernet DDM and how batching packets affects throughput.

```
gin$ cat /proc/i2o/iop0/0x008/settings
name                    value           min             max             mode
----                    -----           ---             ---             ----
max_buckets_out         200             1               256             rw
bucket_thresh           10              1               256             rw
rx_copybreak            18              1               256             rw
tx_batch_mode           1               0               2               rw
rx_batch_mode           1               0               2               rw
event_mask               0xFFC00002      0x00000000      0xFFFFFFFF      rw

gin$ echo "tx_batch_mode:0" > /proc/i2o/iop0/0x008/settings

gin$ cat /proc/i2o/iop0/0x008/settings
name                    value           min             max             mode
----                    -----           ---             ---             ----
max_buckets_out         200             1               256             rw
bucket_thresh           10              1               256             rw
rx_copybreak            18              1               256             rw
tx_batch_mode           0               0               2               rw
rx_batch_mode           1               0               2               rw
event_mask               0xFFC00002      0x00000000      0xFFFFFFFF      rw
```