IA240/241 Linux User's Manual

Sixth Edition, March 2015

www.moxa.com/product

IA240/241 Linux User's Manual

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Trademarks

MOXA is a registered trademark of Moxa Inc.
All other trademarks or registered marks in this manual belong to their respective manufacturers.

Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document "as is," without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Technical Support Contact Information
www.moxa.com/support

Moxa Americas:
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

Moxa China (Shanghai office):
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-10-6872-3958

Moxa Europe:
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

Moxa Asia-Pacific:
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

Table of Contents

1

Introduction

Moxa IA240 and IA241 are RISC-based ready-to-run embedded computers. Available features include four RS-232/422/485 serial ports, dual 10/100 Mbps Ethernet ports, PCMCIA, SD socket for storage expansion and USB 2.0 host making IA240/241 ideal for your embedded applications.

The following topics are covered in this chapter:

- ❑ **Overview**
- ❑ **Software Architecture**
    - ➢ Journaling Flash File System (JFFS2)
    - ➢ Software Package

Overview

The IA240/IA241 embedded computers, which are designed for industrial automation applications, feature 4 RS-232/422/485 serial ports, dual Ethernet ports, 4 digital input channels, 4 digital output channels, and a PCMCIA cardbus and SD socket. The computers come in a compact, IP30 protected, industrial-strength rugged chassis. The DIN-Rail vertical form factor makes it easy to install the IA240/241 embedded computers in small cabinets. This space-saving feature also facilitates easy wiring, and makes the IA240/241 the best choice as front-end embedded controllers for industrial applications.

In addition to the standard models, the IA240/IA241 also come in wide temperature models. The IA240-T and IA241-T have an operating temperature range of -40 to 75°C, and are appropriate for harsh industrial automation environments. The industrial mechanism of the IA240/IA241 design provides robust, reliable computing. Due to the RISC-based architecture, the IA240/IA241 will not generate a lot of heat when in use. The high communication performance and fanless design make the IA240/IA241 ideal for industrial automation environments.

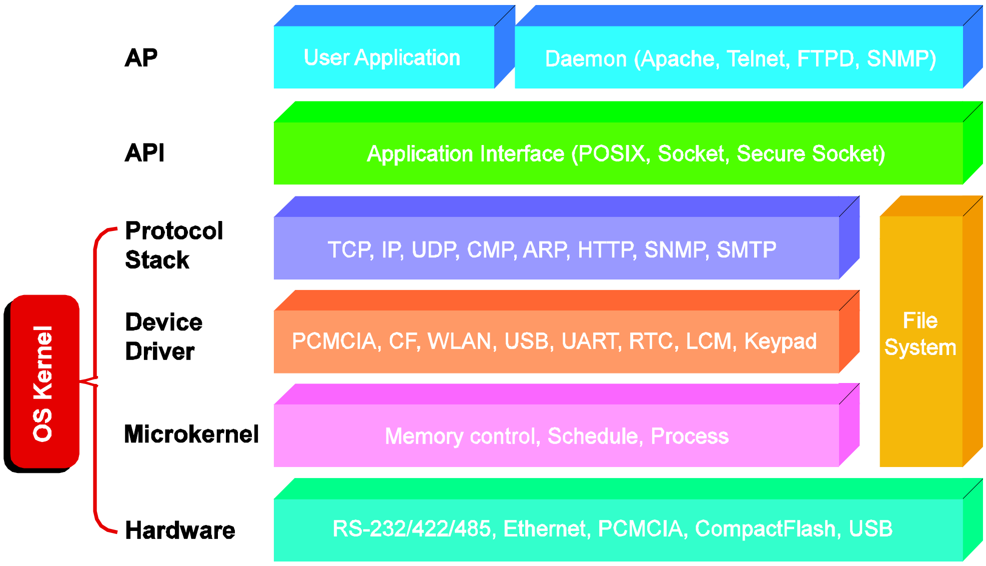
The IA240/241 computers use a Moxa ART 192 Mhz RISC CPU. Unlike the X86 CPU, which uses a CISC design, the RISC architecture and modern semiconductor technology provide these embedded computers with a powerful computing engine and communication functions, but without generating a lot of heat. A 16 MB NOR Flash ROM and a 64 MB SDRAM give you enough memory to install your application software directly on the embedded computer. In addition, dual LAN ports are built right into the RISC CPU. This network capability, in combination with the ability to control serial devices, makes the IA240/241 ideal communication platforms for data acquisition and industrial control applications.

The IA240/241's pre-installed Linux operating system (OS) provides an open software operating system for your software program development. Software written for desktop PCs can be easily ported to the computer with a GNU cross compiler, without needing to modify the source code. The OS, device drivers (e.g., serial and buzzer control), and your own applications, can all be stored in the NOR Flash memory.

The IA240/241 Linux Series (referred to here as IA240/241, or as the target computer) consists of two models: IA241-LX with CardBus, and IA240-LX (which doesn't support CardBus). Both models have exactly the same hardware and software features, except for the PCMCIA CardBus provided by the IA241-LX.

Software Architecture

The Linux operating system that is pre-installed in the IA240/241 follows the standard Linux architecture, making it easy to accept programs that follow the POSIX standard. Program porting is done with the GNU Tool Chain provided by Moxa. In addition to Standard POSIX APIs, device drivers for the USB storage, buzzer and Network controls, and UART are also included in the Linux OS.



The IA240/241's built-in Flash ROM is partitioned into **Boot Loader**, **Linux Kernel, Root File System**, and **User directory** partitions.

In order to prevent user applications from crashing the Root File System, the IA240/241 uses a specially designed **Root File System with Protected Configuration** for emergency use. This **Root File System** comes with serial and Ethernet communication capability for users to load the **Factory Default Image** file. The user directory saves the user's settings and application.

To improve system reliability, the IA240/241 has a built-in mechanism that prevents the system from crashing. When the Linux kernel boots up, the kernel will mount the root file system for read only, and then enable services and daemons. During this time, the kernel will start searching for system configuration parameters via ***rc*** or ***inittab***.

Normally, the kernel uses the Root File System to boot up the system. The Root File System is protected, and cannot be changed by the user. This type of setup creates a "safe" zone.

For more information about the memory map and programming, refer to Chapter 5, *Programmer's Guide*.

Journaling Flash File System (JFFS2)

The Root File System and User directory in the flash memory is formatted with the **Journaling Flash File System (JFFS2)**. The formatting process places a compressed file system in the flash memory. This operation is transparent to the user.

The Journaling Flash File System (JFFS2), which was developed by Axis Communications in Sweden, puts a file system directly on the flash, instead of emulating a block device. It is designed for use on flash-ROM chips and recognizes the special write requirements of a flash-ROM chip. JFFS2 implements wear-leveling to extend the life of the flash disk, and stores the flash directory structure in the RAM. A log-structured file system is maintained at all times. The system is always consistent, even if it encounters crashes or improper power-downs, and does not require ***fsck*** (file system check) on boot-up.

JFFS2 is the newest version of JFFS. It provides improved wear-leveling and garbage-collection performance, improved RAM footprint and response to system-memory pressure, improved concurrency and support for suspending flash erases, marking of bad sectors with continued use of the remaining good sectors (enhancing the write-life of the devices), native data compression inside the file system design, and support for hard links.

The key features of JFFS2 are:

- Targets the Flash ROM Directly
- Robustness
- Consistency across power failures
- No integrity scan (fsck) is required at boot time after normal or abnormal shutdown
- Explicit wear leveling
- Transparent compression

Although JFFS2 is a journaling file system, this does not preclude the loss of data. The file system will remain in a consistent state across power failures and will always be mountable. However, if the board is powered down during a write then the incomplete write will be rolled back on the next boot, but writes that have already been completed will not be affected.

Additional information about JFFS2 is available at:
http://sources.redhat.com/jffs2/jffs2.pdf
http://developer.axis.com/software/jffs/
http://www.linux-mtd.infradead.org/

Software Package

Boot Loader	Moxa private (V1.2)
Kernel	Linux 2.6.9
Protocol Stack	ARP, PPP, CHAP, PAP, IPv4, ICMP, TCP, UDP, DHCP, FTP, SNMP V1/V3, HTTP, NTP, NFS, SMTP, SSH 1.0/2.0, SSL, Telnet, PPPoE, OpenVPN
File System	JFFS2, NFS, Ext2, Ext3, VFAT/FAT
OS shell command	Bash
Busybox	Linux normal command utility collection
Utilities	
tinylogin	login and user manager utility
telnet	telnet client program
ftp	FTP client program
smtpclient	email utility
scp	Secure file transfer Client Program
Daemons	
pppd	dial in/out over serial port daemon
snmpd	snmpd agent daemon
telnetd	telnet server daemon
inetd	TCP server manager program
ftpd	ftp server daemon
apache	web server daemon
sshd	secure shell server
openvpn	virtual private network
openssl	open SSL
Linux Tool Chain	
Gcc (V3.3.2)	C/C++ PC Cross Compiler
GDB (V5.3)	Source Level Debug Server
Glibc (V2.2.5)	POSIX standard C library

2

Getting Started

In this chapter, we explain how to connect the IA240/241, how to turn on the power, how to get started programming, and how to use the IA240/241's other functions.

The following topics are covered in this chapter:

- **Powering on the IA240/241**
- **Connecting the IA240/241 to a PC**
  - Serial Console
  - Telnet Console
  - SSH Console
- **Configuring the Ethernet Interface**
  - Modifying Network Settings with the Serial Console
  - Modifying Network Settings over the Network
- **Configuring the WLAN via the PCMCIA Interface**
  - IEEE802.11g
- **SD Socket and USB for Storage Expansion**
- **Test Program—Developing Hello.c**
  - Installing the Tool Chain (Linux)
  - Checking the Flash Memory Space
  - Compiling Hello.c
  - Uploading and Running the "Hello" Program
- **Developing Your First Application**
  - Testing Environment
  - Compiling tcps2.c
  - Uploading and Running the "tcps2-release" Program
  - Testing Procedure Summary

Powering on the IA240/241

Connect the SG wire to the shielded contact located in the upper left corner of the IA240/241, and then power on the computer by connecting it to the power adaptor. It takes about 30 to 60 seconds for the system to boot up. Once the system is ready, the Ready LED will light up.

NOTE After connecting the IA240/241 to the power supply, it will take about 30 to 60 seconds for the operating system to boot up. The green Ready LED will not turn on until the operating system is ready.

ATTENTION

This product is intended to be supplied by a Listed Power Unit and output marked with "LPS" and rated 12-48 VDC, 580 mA (minimum requirements).

Connecting the IA240/241 to a PC

There are two ways to connect the IA240/241 to a PC: through the Serial Console port or via Telnet over the network.

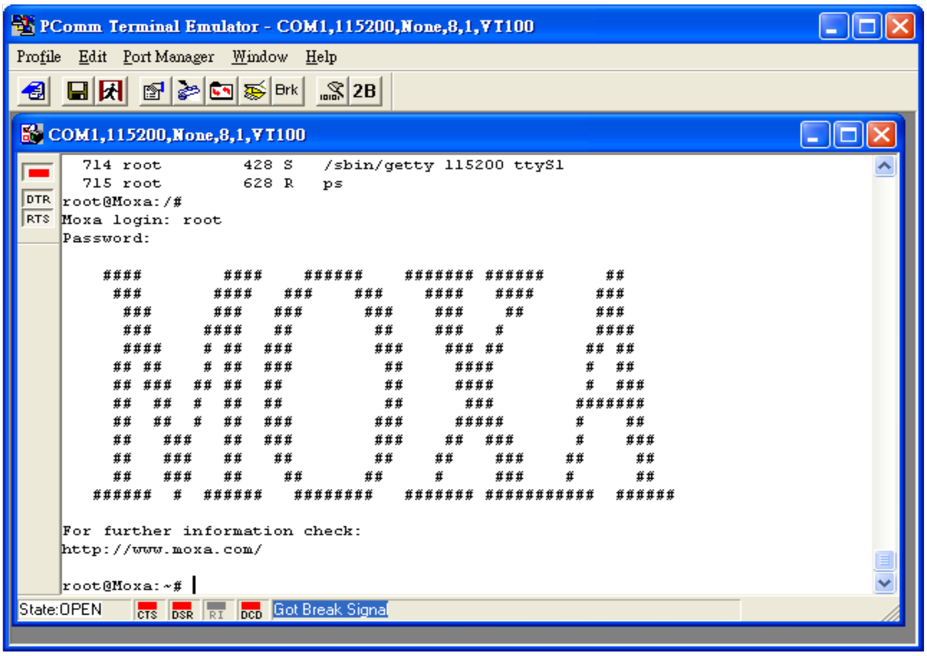
Serial Console

The serial console port gives users a convenient way of connecting to the IA240/241's console utility. This method is particularly useful when using the computer for the first time. The signal is transmitted over a direct serial connection, so you do not need to know either of its two IP addresses in order to connect to the serial console utility.

Use the serial console port settings shown below.

Baudrate	115200 bps
Parity	None
Data bits	8
Stop bits:	1
Flow Control	None
Terminal	VT100

Once the connection is established, the following window will open.



To log in, type the Login name and password as requested. The default values are both **root**:

```
Login:  root
Password: root
```

Telnet Console

If you know at least one of the two IP addresses and netmasks, then you can use Telnet to connect to the IA240/241's console utility. The default IP address and Netmask for each of the two ports are given below:

	Default IP Address	**Netmask**
LAN 1	192.168.3.127	255.255.255.0
LAN 2	192.168.4.127	255.255.255.0

Use a cross-over Ethernet cable to connect directly from your PC to the IA240/241. You should first modify your PC's IP address and netmask so that your PC is on the same subnet as one of IA240/241's two LAN ports. For example, if you connect to LAN 1, you can set your PC's IP address to 192.168.3.126 and netmask to 255.255.255.0. If you connect to LAN 2, you can set your PC's IP address to 192.168.4.126 and netmask to 255.255.255.0.

To connect to a hub or switch connected to your local LAN, use a straight-through Ethernet cable. The default IP addresses and netmasks are shown above. To login, type the Login name and password as requested. The default values are both **root**:

```
Login:  root
Password: root
```

You can proceed with configuring the network settings of the target computer when you reach the bash command shell. Configuration instructions are given in the next section.

ATTENTION

Serial Console Reminder

Remember to choose VT100 as the terminal type. Use the cable CBL-RJ45F9-150, which comes with the IA240/241, to connect to the serial console port.

Telnet Reminder

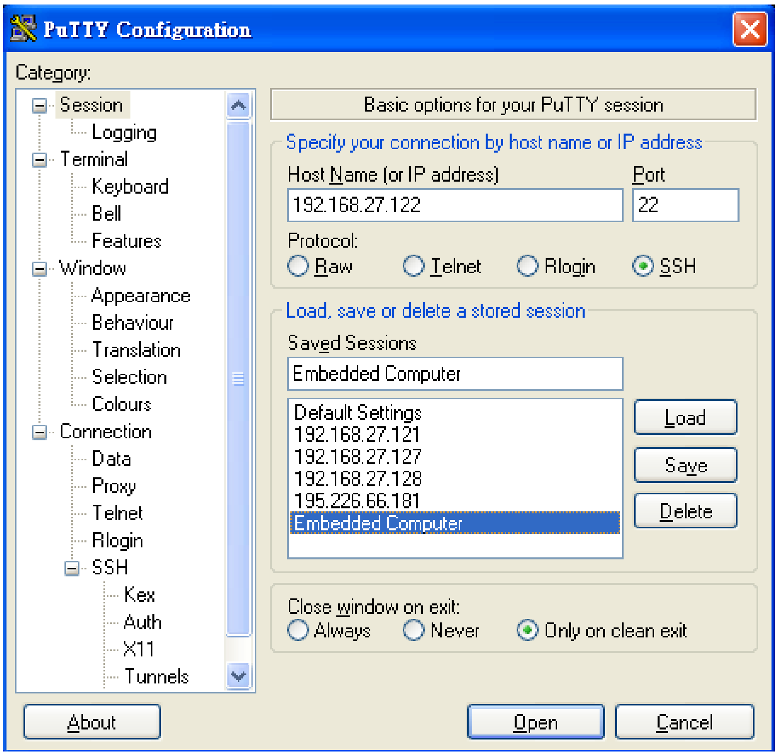
When connecting to the IA240/241 over a LAN, you must configure your PC's Ethernet IP address to be on the same subnet as the IA240/241 that you wish to contact. If you do not get connected on the first try, re-check the serial and IP settings, and then unplug and re-plug the IA240/241's power cord.

SSH Console

The IA240/241 supports an SSH Console to provide users with better security options.

Windows Users

Click on the link http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html to download PuTTY (free software) to set up an SSH console for the IA240/241 in a Windows environment. The following figure shows a simple example of the configuration that is required.



Linux Users

From a Linux machine, use the "ssh" command to access the IA240/241's console utility via SSH.

#ssh 192.168.3.127

Select **yes** to complete the connection.

```
 [root@bee_notebook root]# ssh 192.168.3.127
The authenticity of host '192.168.3.127 (192.168.3.127)' can't be established.
RSA key fingerprint is 8b:ee:ff:84:41:25:fc:cd:2a:f2:92:8f:cb:1f:6b:2f.
Are you sure you want to continue connection (yes/no)? yes_
```

NOTE	SSH provides better security compared to Telnet for accessing the IA240/241's console utility over the network.

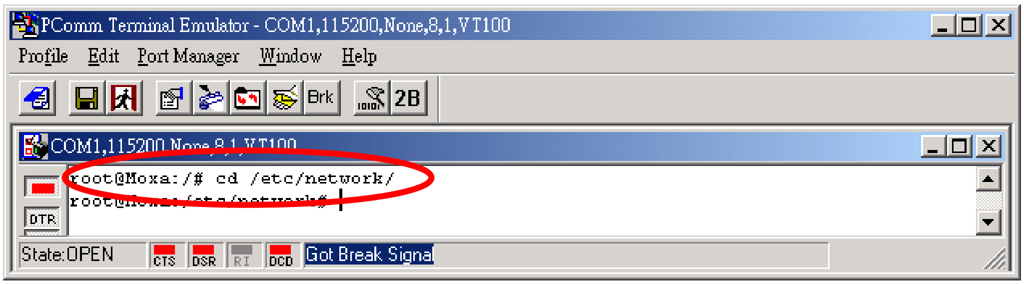
Configuring the Ethernet Interface

The network settings of the IA240/241 can be modified with the serial console, or online over the network.

Modifying Network Settings with the Serial Console

In this section, we use the serial console to configure the network settings of the target computer.

1. Follow the instructions given in a previous section to access the Console Utility of the target computer via the serial console port, and then type **`#cd /etc/network`** to change directories.



2. Type **`#vi interfaces`** to edit the network configuration file with vi editor. You can configure the Ethernet ports of the IA240/241 for **static** or **dynamic** (DHCP) IP addresses.

<u>Static IP addresses</u>:

As shown below, 4 network addresses must be modified: **address**, **network**, **netmask**, and **broadcast**. The default IP addresses are 192.168.3.127 for LAN1 and 192.168.4.127 for LAN2, with default netmask of 255.255.255.0.

```
# We always want the loopback interface.

auto eth0 eth1 lo
iface lo inet loopback

# embedded ethernet LAN1
iface eth0 inet static
        address 192.168.3.127
        network 192.168.3.0
        netmask 255.255.255.0
        broadcast 192.168.3.255

# embedded ethernet LAN2
iface eth1 inet static
        address 192.168.4.127
        network 192.168.4.0
        netmask 255.255.255.0
        broadcast 192.168.4.255

# 802.11g Gigabyte Cardbus wireless card
#iface eth2 inet static
#       address 192.168.5.127
#       network 192.168.5.0
"/etc/network/interfaces" line 1 of 162 --0%--
```

Dynamic IP addresses:

By default, the IA240/241 is configured for "static" IP addresses. To configure one or both LAN ports to request an IP address dynamically, replace **static** with **dhcp** and then delete the address, network, netmask, and broadcast lines.

Default Setting for LAN1	Dynamic Setting using DHCP
iface eth0 inet **static** address 192.168.3.127 network: 192.168.3.0 netmask 255.255.255.0 broadcast 192.168.3.255	iface eth0 inet **dhcp**

```
auto eth0 eth1 lo
iface lo inet loopback

iface eth0 inet dhcp

iface eth1 inet dhcp
```

3. After the boot settings of the LAN interface have been modified, issue the following command to activate the LAN settings immediately:

   #/etc/init.d/networking restart

NOTE After changing the IP settings, use the **networking restart** command to activate the new IP address.

Modifying Network Settings over the Network

IP settings can be activated over the network, but the new settings will not be saved to the flash ROM without modifying the file **/etc/network/interfaces**.

For example, type the command **#ifconfig eth0 192.168.1.1** to change the IP address of LAN1 to 192.168.1.1.

```
root@Moxa:~# ifconfig eth0 192.168.27.125
root@Moxa:~# _
```

Configuring the WLAN via the PCMCIA Interface

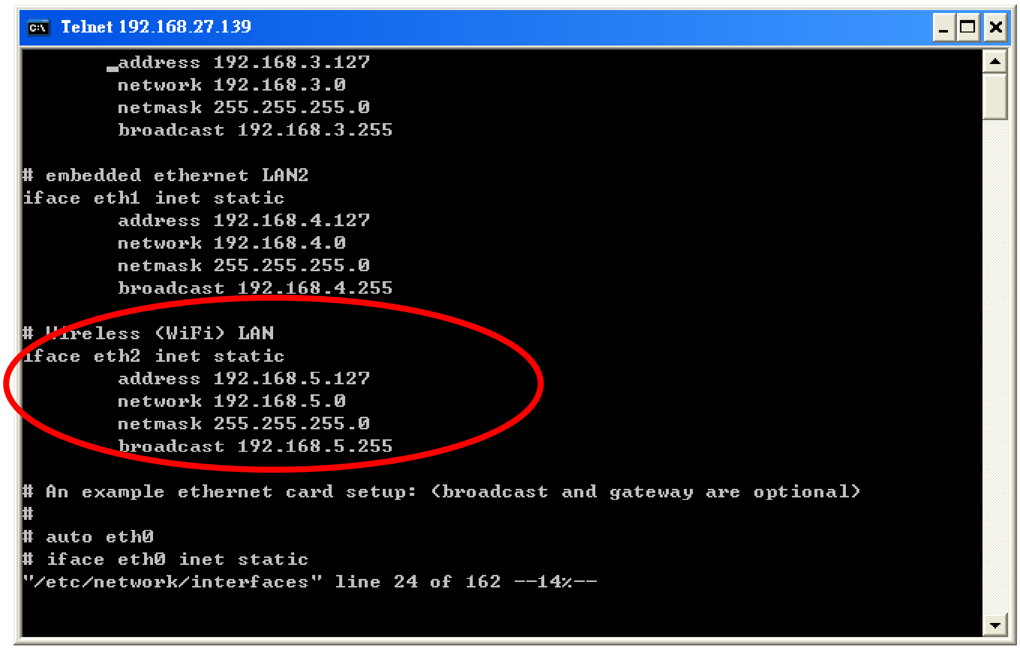
IEEE802.11g

The following IEEE802.11g wireless card modules are supported:

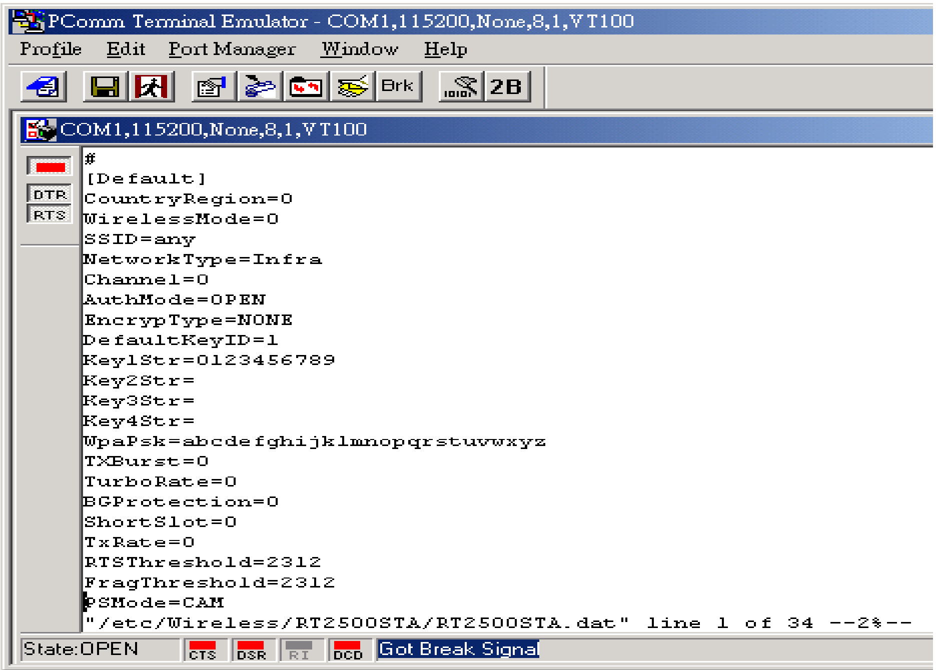
- ASUS—WL-107g
- CNET—CWC-854 (181D version)
- Edmiax—EW-7108PCg
- Amigo—AWP-914W
- GigaByte—GN-WMKG
- Other brands that use the Ralink RT2500 series chip set

To configure the WLAN for IEEE802.11g:

1. Unplug the CardBus Wireless LAN card first.
2. Use the command **`#vi /etc/networking/interfaces`** to open the "interfaces" configuration file with vi editor, and then edit the 802.11g network settings (circled in red in the following figure).



3. Additional WLAN parameters are contained in the file **RT2500STA.dat**. To open the file, navigate to the RT2500STA folder and invoke vi, or type the command **`#vi /etc/Wireless/RT2500STA/RT2500STA.dat`** to edit the file with vi editor. Options for the various parameters are listed below the figure.



CountryRegion—Sets the channels for your particular country / region

Setting	Explanation
0	use channels 1 to 11
1	use channels 1 to 11
2	use channels 1 to 13
3	use channels 10, 11
4	use channels 10 to 13
5	use channel 14
6	use channels 1 to 14
7	use channels 3 to 9

***WirelessMode*—**Sets the wireless mode

Setting	Explanation
0	11b/g mixed
1	11b only
2	11g only

***SSID*—**Sets the softAP SSID

Setting
Any 32-byte string

***NetworkType*—**Sets the wireless operation mode

Setting	Explanation
Infra	Infrastructure mode (uses access points to transmit data)
Adhoc	Adhoc mode (transmits data from host to host)

***Channel*—**Sets the channel

Setting	Explanation
0	Auto
1 to 14	the channel you want to use

***AuthMode*—**Sets the authentication mode

Setting
OPEN
SHARED
WPAPSK
WPANONE

EncrypType—*Sets encryption type*

Setting
NONE
WEP
TKIP
AES

DefaultKeyID—Sets default key ID

Setting
1 to 4

Key1Str, Key2Str, Key3Str, Key4Str—Sets strings Key1 to Key4

Setting
The keys can be input as 5 ascii characters, 10 hex numbers, 13 ascii characters, or 26 hex numbers

TxBurst—WPA pre-shared key

Setting
8 to 64 ascii characters

WpaPsk—Enables or disables TxBurst

Setting	**Explanation**
0	disable
1	enable

TurboRate—Enables or disables TurboRate

Setting	**Explanation**
0	disable
1	enable

BGProtection—Sets 11b/11g protection (this function is for engineering testing only)

Setting	**Explanation**
0	auto
1	always on
2	always off

***ShortSlot*—**Enables or disables the short slot time

Setting	Explanation
0	disable
1	enable

***TxRate*—**Sets the TxRate

Setting	Explanation
0	Auto
1	1 Mbps
2	2 Mbps
3	5.5 Mbps
4	11 Mbps
5	6 Mbps
6	9 Mbps
7	12 Mbps
8	18 Mbps
9	24 Mbps
10	36 Mbps
11	48 Mbps
12	54 Mbps

***RTSThreshold*—**Sets the RTS threshold

Setting
1 to 2347

***FragThreshold*—**Sets the fragment threshold

Setting
256 to 2346

SD Socket and USB for Storage Expansion

Both the IA240 and IA241 provide an SD socket for storage expansion. Moxa provides an SD flash disk for plug & play expansion that allows users to plug in a Secure Digital (SD) memory card compliant with the SD 1.0 standard for up to 1 GB of additional memory space, or a Secure Digital High Capacity (SDHC) memory card compliant with the SD 2.0 standard for up to 16 GB of additional memory space.The SD socket is located on the front panel of the IA240/241. To install an SD card, you must first remove the SD protection cover to access the socket, and then plug the SD card directly into the socket. Remember to press on the SD card first if you want to remove it.

The SD card will be mounted at **`/mnt/sd`**.

In addition to the SD socket, a USB 2.0 host is located on the front panel. The USB host is also designed for storage expansion. To expand the storage by USB flash disk, you just need to plug the USB flash disk into this USB port. The flash disk will be detected automatically, and its file partition will be mounted into the OS. The USB storage will be mounted at **`/mnt/usbstorage`**.

Test Program—Developing Hello.c

In this section, we use the standard "Hello" programming example to illustrate how to develop a program for the IA240/241. In general, program development involves the following seven steps.

Step 1:
Connect the IA240/241 to a Linux PC.
Step 2:
Install Tool Chain (GNU Cross Compiler & glibc).
Step 3:
Set the cross compiler and glibc environment variables.
Step 4:
Code and compile the program.
Step 5:
Download the program to the IA240/241 Via FTP or NFS.
Step 6:
Debug the program
→ If bugs are found, return to Step 4.
→ If no bugs are found, continue with Step 7.
Step 7:
Back up the user directory (distribute the program to additional IA240/241 units if needed).

Installing the Tool Chain (Linux)

The Linux Operating System must be pre-installed in the PC before installing the IA240/241 GNU Tool Chain. Fedora core or compatible versions are recommended. The Tool Chain requires approximately 100 MB of hard disk space on your PC. The IA240/241 Tool Chain software is located on the IA240/241 CD. To install the Tool Chain, insert the CD into your PC and then issue the following commands:

```
#mount /dev/cdrom /mnt/cdrom
#sh /mnt/cdrom/tool-chain/linux/install.sh
```

The Tool Chain will be installed automatically on your Linux PC within a few minutes. Before compiling the program, be sure to set the following path first, since the Tool Chain files, including the compiler, link, library, and include files are located in this directory.

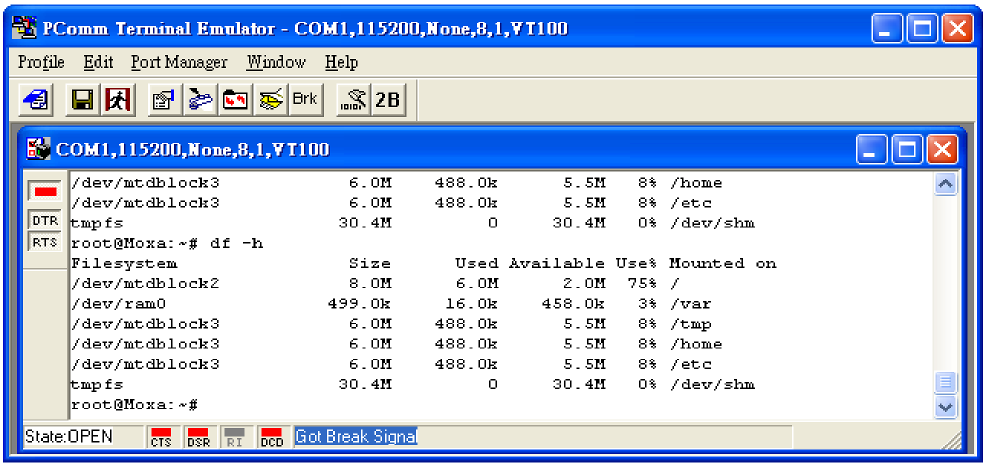
```
PATH=/usr/local/arm-linux/bin:$PATH
```

Setting the path allows you to run the compiler from any directory.

Checking the Flash Memory Space

If the flash memory is full, you will not be able to save data to the Flash ROM. Use the following command to calculate the amount of "Available" flash memory:

```
/>df -h
```



If there isn't enough "Available" space for your application, you will need to delete some existing files. To do this, connect your PC to the IA240/241 with the console cable, and then use the console utility to delete the files from the IA240/241's flash memory. To check the amount of free space available, look at the directories in the read/write directory **/dev/mtdblock3**. Note that the directories **/home** and **/etc** are both mounted on the directory **/dev/mtdblock3**.

NOTE If the flash memory is full, you will need to free up some memory space before saving files to the Flash ROM.

Compiling Hello.c

The package CD contains several example programs. Here we use **Hello.c** as an example to show you how to compile and run your applications. Type the following commands from your PC to copy the files used for this example from the CD to your computer's hard drive:

```
# cd /tmp/
# mkdir example
```

```
# cp –r /mnt/cdrom/example/* /tmp/example
```

To compile the program, go to the **Hello** subdirectory and issue the following commands:

```
#cd example/hello
#make
```

You should receive the following response:

```
[root@localhost hello]# make
/usr/local/arm-linux/bin/arm-linux-gcc -o hello-release hello.c
/usr/local/arm-linux/bin/arm-linux-strip -s hello-release
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o hello-debug hello.c
[root@localhost hello]# _
```

Next, execute hello.exe to generate **hello-release** and **hello-debug**, which are described below:

hello-release—an ARM platform execution file (created specifically to run on the IA240/241)

hello-debug—an ARM platform GDB debug server execution file (see Chapter 5 for details about the GDB debug tool).

NOTE Since Moxa's tool chain places a specially designed **Makefile** in the directory **/tmp/example/hello**, be sure to type the **#make** command from within that directory. This special Makefile uses the mxscale-gcc compiler to compile the hello.c source code for the Xscale environment. If you type the **#make** command from within any other directory, Linux will use the x86 compiler (for example, cc or gcc).

Refer to Chapter 5 to see a Makefile example.

Uploading and Running the "Hello" Program

Use the following commands to upload **hello-release** to the IA240/241 via FTP.

1. From the PC, type:

```
#ftp 192.168.3.127
```

2. Use the bin command to set the transfer mode to Binary mode, and then use the put command to initiate the file transfer:

```
ftp> bin
ftp> put hello-release
```

3. From the IA240/241, type:

```
# chmod +x hello-release
# ./hello-release
```

The word **Hello** will be printed on the screen.

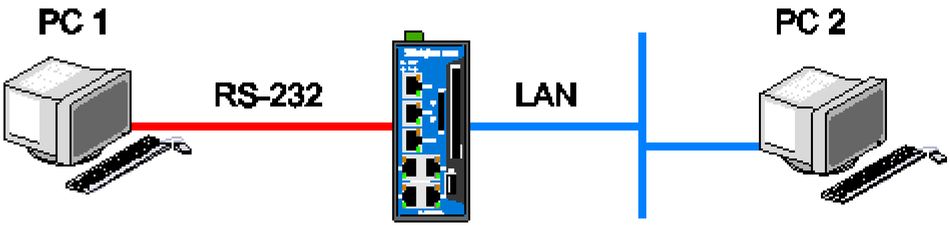
```
root@Moxa:~# ./hello-release
Hello
```

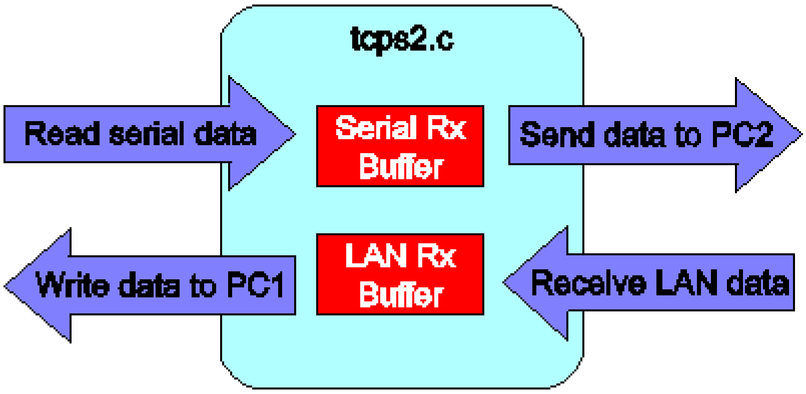
Developing Your First Application

We use the tcps2 example to illustrate how to build an application. The procedure outlined in the following subsections will show you how to build a TCP server program plus serial port communication that runs on the IA240/241.

Testing Environment

The tcps2 example demonstrates a simple application program that delivers transparent, bi-directional data transmission between the IA240/241's serial and Ethernet ports. As illustrated in the following figure, the purpose of this application is to transfer data between PC 1 and the IA240/241 via an RS-232 connection. At the remote site, data can be transferred between the IA240/241's Ethernet port and PC 2 over an Ethernet connection.





Compiling tcps2.c

The source code for the tcps2 example is located on the CD-ROM at **CD-ROM://example/TCPServer2/tcps2.c**. Use the following commands to copy the file to a specific directory on your PC. We use the directory **/home/ia240241/1st_application/**. Note that you need to copy 3 files—Makefile, tcps2.c, tcpsp.c—from the CD-ROM to the target directory.

```
#mount -t iso9660 /dev/cdrom /mnt/cdrom
#cp /mnt/cdrom/example/TCPServer2/tcps2.c/home/ia240241/1st_application/tcps2.c
#cp /mnt/cdrom/example/TCPServer2/tcpsp.c/home/ia240241/1st_application/tcpsp.c
#cp /mnt/cdrom/example/TCPServer2/Makefile.c/home/ia240241/1st_application/Makefile
```

Type **`#make`** to compile the example code:

You will get the following response, indicating that the example program was compiled successfully.

```
root@server11:/home/ia240241/1st_application
```

```
[root@server11 1st_application]# pwd
/home/ia240241/1st_application
[root@server11 1st_application]# ll
total 20
-rw-r—r--  1 root root  514 Nov 27 11:52 Makefile
-rw-r—r--  1 root root 4554 Nov 27 11:52 tcps2.c
-rw-r—r--  1 root root 6164 Nov 27 11:55 tcps2.c
[root@server11 1st_application]# make_
/usr/local/arm-linux/bin/arm-linux-gcc -o tcps2-release tcps2.c
/usr/local/arm-linux/bin/arm-linux-strip -s tcps2-release
/usr/local/arm-linux/bin/arm-linux-gcc -o tcpsp-release tcpsp.c
/usr/local/arm-linux/bin/arm-linux-strip -s tcpsp-release
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o tcps2-debug tcps2.c
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o tcpsp-debug tcpsp.c
[root@server11 1st_application]# ll
total 92
-rw-r—-r--  1 root root   514  Nov 27 11:52 Makefile
-rwxr-xr—x  1 root root 25843 Nov 27 12:03 tcps2-debug
-rwxr—xr-x  1 root root  4996  Nov 27 12:03 tcps2-release
-rw-r—-r--  1 root root  4554  Nov 27 11:52 tcps2.c
-rwxr—xr-x  1 root root 26823 Nov 27 12:03 tcpsp-debug
-rwxr—xr-x  1 root root  5396  Nov 27 12:03 tcpsp-release
-rw-r—-r--  1 root root  6164  Nov 27 11:55 tcpsp.c
[root@server11 1st_application]#
```

Two executable files, tcps2-release and tcps2-debug, are created.

tcps2-release—an ARM platform execution file (created specifically to run on the IA240/241)

tcps2-debug—an ARM platform GDB debug server execution file (see Chapter 5 for details about the GDB debug tool).

NOTE If you get an error message at this point, it could be because you neglected to put tcps2.c and tcpsp.c in the same directory. The example Makefile we provide is set up to compile both tcps2 and tcpsp into the same project Makefile. Alternatively, you could modify the Makefile to suit your particular requirements.

Uploading and Running the "tcps2-release" Program

Use the following commands to use FTP to upload **tcps2-release** to the IA240/241.

1. From the PC, type:

```
#ftp 192.168.3.127
```

2. Next, use the **bin** command to set the transfer mode to **Binary**, and the **put** command to initiate the file transfer:

```
ftp> bin
ftp> put tcps2-release
```

```
root@server11:/home/ia240241/1st_application
[root@server11 1st_application]# ftp 192.168.3.127
Connected to 192.168.3.127
220 Moxa FTP server (Version wu-2.6.1(2) Mon Nov 24 12:17:04 CST 2003) ready.
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (192.168.3.127:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bin
200 Type set to I.
ftp> put tcps2-release
local: tcps2-release remote: tcps2-release
277 Entering Passive Mode (192.168.3.127.82.253)
150 Opening BINARY mode data connection for tcps2-release.
226 Transfer complete
4996 bytes sent in 0.00013 seconds (3.9e+04 Kbytes/s)
ftp> ls
227 Entering Passive Mode (192.168.3.127.106.196)
150 Opening ASCII mode data connection for /bin/ls.
-rw-------    1 root     root          899 Jun 10 08:11  bash_history
-rw-r--r--    1 root     root         4996 Jun 12 02:15 tcps2-release
226 Transfer complete
ftp>
```

3. From the IA240/241, type:

```
# chmod +x tcps2-release
# ./tcps2-release &
```

```
192.168.3.127 - PuTTY
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rw-r--r--  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# chmod +x tcps2-release
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rwxr-xr-x  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~#
```

4. The program should start running in the background. Use the **#ps -ef** command to check if the tcps2 program is actually running in the background.

#ps // use this command to check if the program is running

```
192.168.3.127 - PuTTY
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1  1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rw-r--r--  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# chmod +x tcps2-release
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1  1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rwxr-xr-x  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# ./tcps2-release &
[1] 187
start
root@Moxa:~# ps
[1]+  Running     ./tcps2-release &
root@Moxa:~#
```

NOTE Use the **kill** command for job number 1 to terminate this program: **#kill %1**

#ps -ef // use this command to check if the program is running

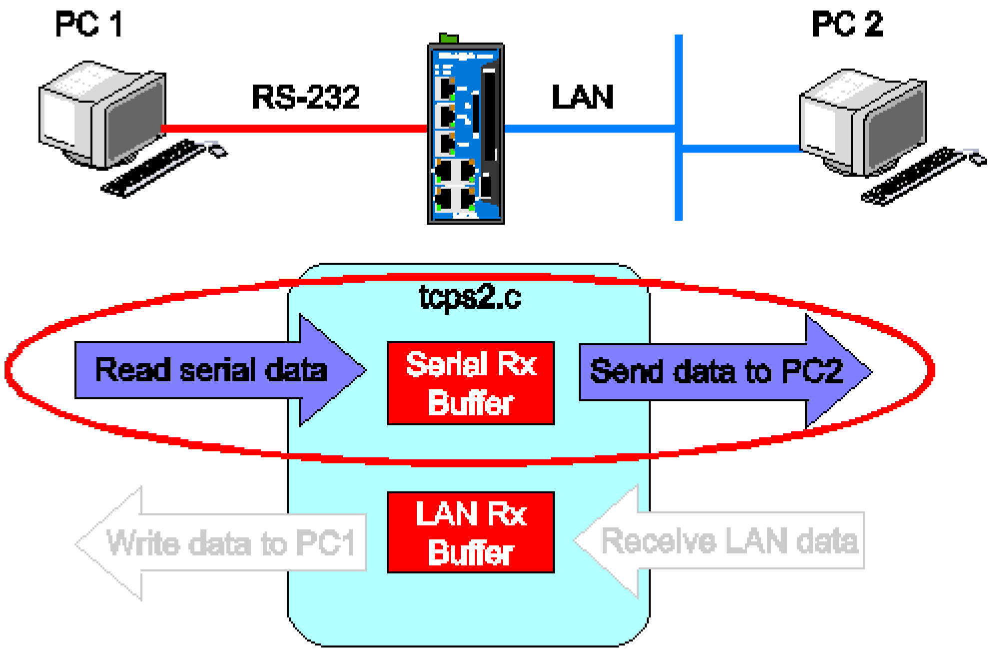
```
192.168.3.127 - PuTTY
[1]+  Running     ./tcps2-release &
root@Moxa:~# ps -ef
  PID  Uid     VmSize Stat Command
    1 root        532 S   init [3]
    2 root            SWN [ksoftirqd/0]
    3 root            SW< [events/0]
    4 root            SW< [khelper]
   13 root            SW< [kblockd/0]
   14 root            SW  [khubd]
   24 root            SW  [pdflush]
   25 root            SW  [pdflush]
   27 root            SW< [aio/0]
   26 root            SW  [kswapd0]
  604 root            SW  [mtdblockd]
  609 root            SW  [pccardd]
  611 root            SW  [pccardd]
  625 root            SWN [jffs2_gcd_mtd3]
  673 root        500 S   /bin/inetd
  679 root       3004 S   /usr/bin/httpd -k start -d /etc/apache
  682 bin         380 S   /bin/portmap
  685 root       1176 S   /bin/sh --login
  690 root        464 S   /bin/snmpd
  694 nobody     3012 S   /usr/bin/httpd -k start -d /etc/apache
  695 nobody     3012 S   /usr/bin/httpd -k start -d /etc/apache
  696 nobody     3012 S   /usr/bin/httpd -k start -d /etc/apache
  697 nobody     3012 S   /usr/bin/httpd -k start -d /etc/apache
  698 nobody     3012 S   /usr/bin/httpd -k start -d /etc/apache
  701 root        352 S   /bin/reportip
  714 root       1176 S   -bash
  726 root        436 S   /bin/telnetd
  727 root       1164 S   -bash
  728 root       1264 S ./tcps2-release
  729 root       1592 S ps -ef
root@Moxa:~#
```

NOTE Use the `kill -9` command for PID 187 to terminate this program: `#kill -9 %187`

Testing Procedure Summary

1. Compile **tcps2.c** (`#make`).
2. Upload and run **tcps2-release** in the background (`#./tcps2-release &`).
3. Check that the process is running (`#jobs` or `#ps -ef`).
4. Use a serial cable to connect PC1 to the IA240/241's serial port 1.
5. Use an Ethernet cable to connect PC2 to the IA240/241.
6. On PC1: If running Windows, use HyperTerminal (**38400, n, 8, 1**) to open COMn.
7. On PC2: Type `#telnet 192.168.3.127 4001`.
8. On PC1: Type some text on the keyboard and then press **Enter**.
9. On PC2: The text you typed on PC1 will appear on PC2's screen.

The testing environment is illustrated in the following figure. However, note that there are limitations to the example program **tcps2.c**.



NOTE The **tcps2.c** application is a simple example designed to give users a basic understanding of the concepts involved in combining Ethernet communication and serial port communication. However, the example program has some limitations that make it unsuitable for real-life applications.

1. The serial port is in canonical mode and block mode, making it impossible to send data from the Ethernet side to the serial side (i.e., from PC 2 to PC 1 in the above example).
2. The Ethernet side will not accept multiple connections.

3

Managing Embedded Linux

This chapter includes information about version control, deployment, updates, and peripherals. The information in this chapter will be particularly useful when you need to run the same application on several IA240/241 units.

The following topics are covered in this chapter:

- **System Version Information**
- **System Image Backup**
  - Upgrading the Firmware
  - Loading Factory Defaults
- **Enabling and Disabling Daemons**
- **Setting the Run-Level**
- **Adjusting the System Time**
  - Setting the Time Manually
  - NTP Client
  - Updating the Time Automatically
- **Cron—Daemon to Execute Scheduled Commands**

System Version Information

To determine the hardware capability of your IA240/241, and what kind of software functions are supported, check the version numbers of your IA240/241's hardware, kernel, and user file system. Contact Moxa to determine the hardware version. You will need the **Production S/N** (Serial number), which is located on the IA240/241's bottom label.

To check the kernel version, type:
#kversion

```
192.168.3.127 - PuTTY
root@Moxa:~# kversion
Version 1.0
root@Moxa:~#
```

NOTE The kernel version number is for the factory default configuration, and if you download the latest firmware version from Moxa's website and then upgrade the IA240/241's hardware.

System Image Backup

Upgrading the Firmware

The IA240/241's bios, kernel, and root file system are combined into one firmware file, which can be downloaded from Moxa's website (www.moxa.com). The name of the file has the form **ia240-x.x.x.frm or ia241-x.x.x.frm**, with "x.x.x" indicating the firmware version. To upgrade the firmware, download the firmware file to a PC, and then transfer the file to the IA240/241 via a serial Console or Telnet Console connection.

ATTENTION

Upgrading the firmware will erase all data on the Flash ROM

If you are using the **ramdisk** to store code for your applications, beware that updating the firmware will erase all of the data on the Flash ROM. You should back up your application files and data before updating the firmware.

Since different Flash disks have different sizes, it's a good idea to check the size of your Flash disk before upgrading the firmware, or before using the disk to store your application and data files. Use the **#df -h** command to list the size of each memory block and how much free space is available in each block.

```
192.168.3.127 - PuTTY
 root@Moxa:~# df -h
 Filesystem   Size   Used Available Use% Mounted on
/dev/mtdblock2 8.0M     6.0M      2.0M    75% /
/dev/ram0      499.0k  16.0k    458.0k    3% /var
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /tmp
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /home
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /etc
tmpfs          30.4M       0     30.4M    0% /dev/shm
 root@Moxa:~# upramdisk
 root@Moxa:~# df -h
 Filesystem   Size   Used Available Use% Mounted on
/dev/mtdblock2 8.0M     6.0M      2.0M    75% /
/dev/ram0      499.0k  16.0k    458.0k    3% /var
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /tmp
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /home
/dev/mtdblock3 6.0M    488.0k     5.5M    8% /etc
tmpfs          30.4M       0     30.4M    0% /dev/shm
/dev/ram1     16.0M    1.0k  15.1M    0%  /mnt/ramdisk
 root@Moxa:~# cd /mnt/ramdisk
 root@Moxa:/mnt/ramdisk#
```

The following instructions give the steps required to save the firmware file to the IA240/241's RAM disk and how to upgrade the firmware.

1. Type the following commands to enable the RAM disk:

```
#upramdisk
#cd /mnt/ramdisk
```

2. Type the following commands to use the IA240/241's built-in FTP client to transfer the firmware file (**ia240-x.x.x.frm or ia241-x.x.x.frm**) from the PC to the IA240/241:

```
/mnt/ramdisk> ftp <destination PC's IP>
Login Name: xxxx
Login Password: xxxx
ftp> bin
ftp> get ia240-x.x.x.frm
```

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# ftp 192.168.3.193
Connected to 192.168.3.193 (192.168.3.193).
220 TYPSoft FTP Server 1.10 ready…
Name (192.168.3.193:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd newsw
250 CWD command successful. "/C:/ftproot/newsw/" is current directory.
ftp> bin
200 Type set to I.
ftp> ls
200 Port command successful.
150 Opening data connection for directory list.
drw-rw-rw-   1 ftp  ftp     0 Nov 30 10:03 .
drw-rw-rw-   1 ftp  ftp     0 Nov 30 10:03 .
-rw-rw-rw-   1 ftp  ftp 13167772 Nov 29 10:24 ia240-1.0.frm
226 Transfer complete.
```

```
ftp> get ia240-1.0.frm
local: ia240-1.0.frm remote: ia240-1.0.frm
200 Port command successful.
150 Opening data connection for ia240-1.0.frm
226 Transfer complete.
13167772 bytes received in 2.17 secs (5925.8 kB/s)
ftp>
```

3. Next, use the **upfirm** command to upgrade the kernel and root file system:

#upfirm ia240-x.x.x.frm

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# upfirm ia240-1.0.frm
Moxa  IA240 upgrade firmware utility version 1.0.
To check source firmware file context.
The source firmware file conext is OK.
This step will destroy all your firmware.
Continue ? (Y/N) : Y
Now upgrade the file [kernel].
Format MTD device [/dev/mtd1] . . .
MTD device [/dev/mtd1] erase 128 Kibyte @ 1C0000 - 100% complete.
Wait to write file . . .
Compeleted 100%
Now upgrade the file [usrdisk].
Format MTD device [/dev/mtd2] . . .
MTD device [/dev/mtd2] erase 128 Kibyte @ 800000 - 100% complete.
Wait to write file . . .
Compeleted 100%
Upgrade the firmware is OK.
```

ATTENTION

The upfirm utility will reboot your target after the upgrade is OK.

Loading Factory Defaults

To load the the factory default settings, you must press the reset-to-default button for more than 5 seconds. All files in the /home & /etc directories will be destroyed. Note that while pressing the reset-to-default button, the Ready LED will blink once every second for the first 5 seconds. The Ready LED will turn off after 5 seconds, and the factory defaults will be loaded.

Enabling and Disabling Daemons

The following daemons are enabled when the IA240/241 boots up for the first time.

snmpd ....SNMP Agent daemon
telnetd ....Telnet Server / Client daemon
inetd ....Internet Daemons
ftpd...............FTP Server / Client daemon
sshd ....Secure Shell Server daemon
httpd.............Apache WWW Server daemon

Type the command "ps –ef" to list all processes currently running.

```
192.168.3.127 - PuTTY
root@Moxa:~# cd /etc
root@Moxa:/etc# ps -ef
  PID  Uid     VmSize Stat Command
    1 root       532 S   init [3]
    2 root           SWN [ksoftirqd/0]
    3 root           SW< [events/0]
    4 root           SW< [khelper]
   13 root           SW< [kblockd/0]
   14 root           SW  [khubd]
   24 root           SW  [pdflush]
   25 root           SW  [pdflush]
   27 root           SW< [aio/0]
   26 root           SW  [kswapd0]
  604 root           SW  [mtdblockd]
  609 root           SW  [pccardd]
  611 root           SW  [pccardd]
  625 root           SWN [jffs2_gcd_mtd3]
  673 root       500 S   /bin/inetd
  679 root      3004 S   /usr/bin/httpd -k start -d /etc/apache
  682 bin        380 S   /bin/portmap
  685 root      1176 S   /bin/sh --login
  690 root       464 S   /bin/snmpd
  694 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  695 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  696 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  697 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  698 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  701 root       352 S   /bin/reportip
  714 root      1176 S   -bash
  726 root       436 S   /bin/telnetd
  727 root      1180 S   -bash
  783 root       628 R   ps -ef
root@Moxa:/ect#
```

To run a private daemon, you can edit the file rc.local, as follows:
#cd /etc/rc.d
#vi rc.local

```
192.168.3.127 - PuTTY
root@Moxa:~# cd /etc/rc.d
root@Moxa:/etc/rc.d# vi rc.local
```

Next, use vi to open your application program. We use the example program **tcps2-release**, and put it to run in the background.

```
192.168.3.127 - PuTTY
# !/bin/sh
# Add you want to run daemon
/root/tcps2-release &~
```

The enabled daemons will be available after you reboot the system.

```
192.168.3.127 - PuTTY
root@Moxa:~# ps -ef
  PID  Uid     VmSize Stat Command
    1 root       532 S   init [3]
    2 root           SWN [ksoftirqd/0]
    3 root           SW< [events/0]
    4 root           SW< [khelper]
   13 root           SW< [kblockd/0]
   14 root           SW  [khubd]
   24 root           SW  [pdflush]
   25 root           SW  [pdflush]
   27 root           SW< [aio/0]
   26 root           SW  [kswapd0]
  604 root           SW  [mtdblockd]
  609 root           SW  [pccardd]
  611 root           SW  [pccardd]
  625 root           SWN [jffs2_gcd_mtd3]
  673 root       500 S   /bin/inetd
  674 root      1264 S   /root/tcps2-release
  679 root      3004 S   /usr/bin/httpd -k start -d /etc/apache
  682 bin        380 S   /bin/portmap
  685 root      1176 S   /bin/sh --login
  690 root       464 S   /bin/snmpd
  694 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  695 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  696 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  697 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  698 nobody    3012 S   /usr/bin/httpd -k start -d /etc/apache
  701 root       352 S   /bin/reportip
  714 root      1176 S   -bash
  726 root       436 S   /bin/telnetd
  727 root      1180 S   -bash
  783 root       628 R   ps -ef
root@Moxa:~#
```

Setting the Run-Level

In this section, we outline the steps you should take to set the Linux run-level and execute requests. Use the following command to enable or disable settings:

```
192.168.3.127 - PuTTY
root@Moxa:/ect/rc.d/rc3.d# ls
S19nfs-common  S25nfs-user-serverS99showreadyled
S20snmpd    S55ssh
S24pcmcia   S99rmnologin
root@Moxa:/etc/rc.d/rc3.d#
```

```
#cd /etc/rc.d/init.d
```

Edit a shell script to execute **/root/tcps2-release** and save to **tcps2** as an example.

```
#cd /etc/rc.d/rc3.d
#ln –s /etc/rc.d/init.d/tcps2 S60tcps2
```

SxxRUNFILE stands for
S: start the run file while linux boots up.
xx: a number between 00-99. Smaller numbers have a higher priority.
RUNFILE: the file name.

```
192.168.3.127 - PuTTY
root@Moxa:/ect/rc.d/rc3.d# ls
S19nfs-common  S25nfs-user-serverS99showreadyled
S20snmpd    S55ssh
S24pcmcia   S99rmnologin
root@Moxa:/ect/rc.d/rc3.d# ln -s /root/tcps2-release S60tcps2
root@Moxa:/ect/rc.d/rc3.d# ls
S19nfs-common  S25nfs-user-serverS99rmnologin
S20snmpd    S55ssh      S99showreadyled
S24pcmcia   S60tcps2
root@Moxa:/etc/rc.d/rc3.d#
```

KxxRUNFILE stands for
K: start the run file while linux shuts down or halts.
xx: a number between 00-99. Smaller numbers have a higher priority.
RUNFILE: the file name.

To remove the daemon, remove the run file from the **/etc/rc.d/rc3.d** directory by using the following command:

#rm –f /etc/rc.d/rc3.d/S60tcps2

Adjusting the System Time

Setting the Time Manually

The IA240/241 has two time settings. One is the system time, and the other is the RTC (Real Time Clock) time kept by the IA240/241's hardware. Use the **#date** command to query the current system time or set a new system time. Use **#hwclock** to query the current RTC time or set a new RTC time.

Use the following command to query the system time:
#date

Use the following command to query the RTC time:
#hwclock

Use the following command to set the system time:
#date MMDDhhmmYYYY

MM = Month
DD = Date
hhmm = hour and minute
YYYY = Year

Use the following command to set the RTC time:
#hwclock –w

Write current system time to RTC

The following figure illustrates how to update thesystem time and set the RTC time.

```
192.168.3.127 - PuTTY
root@Moxa:~# date
Fri Jun 23 23:30:31 CST 2000
root@Moxa:~# hwclock
Fri Jun 23 23:30:35 2000  -0.557748 seconds
root@Moxa:~# date 120910002004
Thu Dec  9 10:00:00 CST 2004
root@Moxa:~# hwclock -w
root@Moxa:~# date ; hwclock
Thu Dec  9 10:01:07 CST 2004
Thu Dec  9 10:01:08 2004  -0.933547 seconds
root@Moxa:~#
```

NTP Client

The IA240/241 has a built-in NTP (Network Time Protocol) client that is used to initialize a time request to a remote NTP server. Use **#ntpdate <*this client utility*>** to update the system time.

```
#ntpdate time.stdtime.gov.tw
#hwclock -w
```

Visit http://www.ntp.org for more information about NTP and NTP server addresses.

```
10.120.53.100 - PuTTY
root@Moxa:~# date ; hwclock
Sat Jan  1 00:00:36 CST 2000
Sat Jan  1 00:00:37 2000  -0.772941 seconds
root@Moxa:~# ntpdate time.stdtion.gov.tw
 9 Dec 10:58:53 ntpdate[207]: step time server 220.130.158.52 offset 155905087.9
84256 sec
root@Moxa:~# hwclock -w
root@Moxa:~# date ; hwclock
Thu Dec  9 10:59:11 CST 2004
Thu Dec  9 10:59:12 2004  -0.844076 seconds
root@Moxa:~#
```

NOTE Before using the NTP client utility, check your IP and DNS settings to make sure that an Internet connection is available. Refer to Chapter 2 for instructions on how to configure the Ethernet interface, and see Chapter 4 for DNS setting information.

Updating the Time Automatically

In this subsection, we show how to use a shell script to update the time automatically.

Example shell script to update the system time periodically

```
#!/bin/sh
ntpdate time.nist.gov # You can use the time server's ip address or domain
        # name directly. If you use domain name, you must
        # enable the domain client on the system by updating
        # /etc/resolv.conf file.
hwclock -systohc
sleep 100  # Updates every 100 seconds. The min. time is 100 seconds. Change
    # 100 to a larger number to update RTC less often.
```

Save the shell script using any file name. E.g., **fixtime**

How to run the shell script automatically when the kernel boots up

Copy the example shell script `fixtime` to directory `/etc/init.d`, and then use `chmod 755 fixtime` to change the shell script mode. Next, use vi editor to edit the file `/etc/inittab`. Add the following line to the bottom of the file:

```
ntp : 2345 : respawn : /etc/init.d/fixtime
```

Use the command `#init q` to re-init the kernel.

Cron—Daemon to Execute Scheduled Commands

Start Cron from the directory `/etc/rc.d/rc.local`. It will return immediately, so you don't need to start it with '&' to run in the background.

The Cron daemon will search `/etc/cron.d/crontab` for crontab files, which are named after accounts in /etc/passwd.

Cron wakes up every minute, and checks each command to see if it should be run in the current minute.

Modify the file `/etc/cron.d/crontab` to set up your scheduled applications. Crontab files have the following format:

mm	h	dom	mon	dow	user	command
min	hour	date	month	week	user	command
0-59	0-23	1-31	1-12	0-6 (0 is Sunday)		

The following example demonstrates how to use Cron.

How to use cron to update the system time and RTC time every day at 8:00.

STEP1: Write a shell script named fixtime.sh and save it to /home/.

```
#!/bin/sh
ntpdate time.nist.gov
hwclock –systohc
exit 0
```

STEP2: Change mode of fixtime.sh

```
#chmod 755 fixtime.sh
```

STEP3: Modify /etc/cron.d/crontab file to run fixtime.sh at 8:00 every day.

Add the following line to the end of crontab:

```
* 8 * * * root /home/fixtime.sh
```

STEP4: Enable the cron daemon manually.

```
#/etc/init.d/cron start
```

STEP5: Enable cron when the system boots up.

Add the following line in the file /etc/init.d/rc.local

```
#/etc/init.d/cron start
```

4

Managing Communications

In this chapter, we explain how to configure the IA240/241's various communication functions.

The following topics are covered in this chapter:

- **Telnet / FTP**
- **DNS**
- **Web Service—Apache**
- **Install PHP for Apache Web Server**
- **IPTABLES**
- **NAT**
  - NAT Example
  - Enabling NAT at Bootup
- **Dial-up Service—PPP**
- **PPPoE**
- **NFS (Network File System)**
  - Setting up the IA240/241 as an NFS Client
- **Mail**
- **SNMP**
- **OpenVPN**

Telnet / FTP

In addition to supporting Telnet client/server and FTP client/server, the IA240/241 also supports SSH and sftp client/server. To enable or disable the Telnet/ftp server, you first need to edit the file **/etc/inetd.conf**.

Enabling the Telnet/ftp server

The following example shows the default content of the file /etc/inetd.conf. The default is to enable the Telnet/ftp server:

```
discard dgram udp wait root /bin/discard
discard stream tcp nowait root /bin/discard
telnet stream tcp nowait root /bin/telnetd
ftp stream tcp nowait root /bin/ftpd -l
```

Disabling the Telnet/ftp server

Disable the daemon by typing '#' in front of the first character of the row to comment out the line.

DNS

The IA240/241 supports DNS client (but not DNS server). To set up DNS client, you need to edit three configuration files: **/etc/hosts, /etc/resolv.conf, and /etc/nsswitch.conf.**

/etc/hosts
This is the first file that the Linux system reads to resolve the host name and IP address.

/etc/resolv.conf
This is the most important file that you need to edit when using DNS for the other programs. For example, before you use #ntpdate time.nist.goc to update the system time, you will need to add the DNS server address to the file. Ask your network administrator which DNS server address you should use. The DNS server's IP address is specified with the "nameserver" command. For example, add the following line to /etc/resolv.conf if the DNS server's IP address is 168.95.1.1:

```
nameserver 168.95.1.1
```

```
10.120.53.100 - PuTTY
root@Moxa:/etc# cat resolv.conf
#
# resolv.conf  This file is the resolver configuration file
# See resolver(5).
#
#nameserver 192.168.1.16
nameserver 168.95.1.1
nameserver 140.115.1.31
nameserver 140.115.236.10
root@Moxa:/etc#
```

/etc/nsswitch.conf
This file defines the sequence to resolve the IP address by using /etc/hosts file or /etc/resolv.conf.

Web Service—Apache

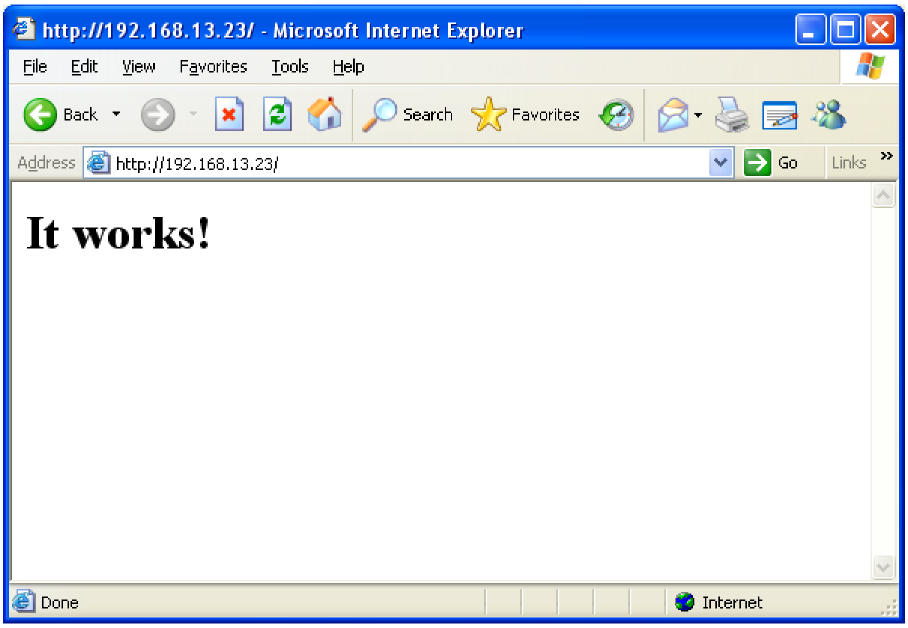
The Apache web server's main configuration file is **`/etc/apache/conf/httpd.conf`**, with the default homepage located at **`/home/httpd/htdocs/index.html`**. Save your own homepage to the following directory:

/home/httpd/htdocs/

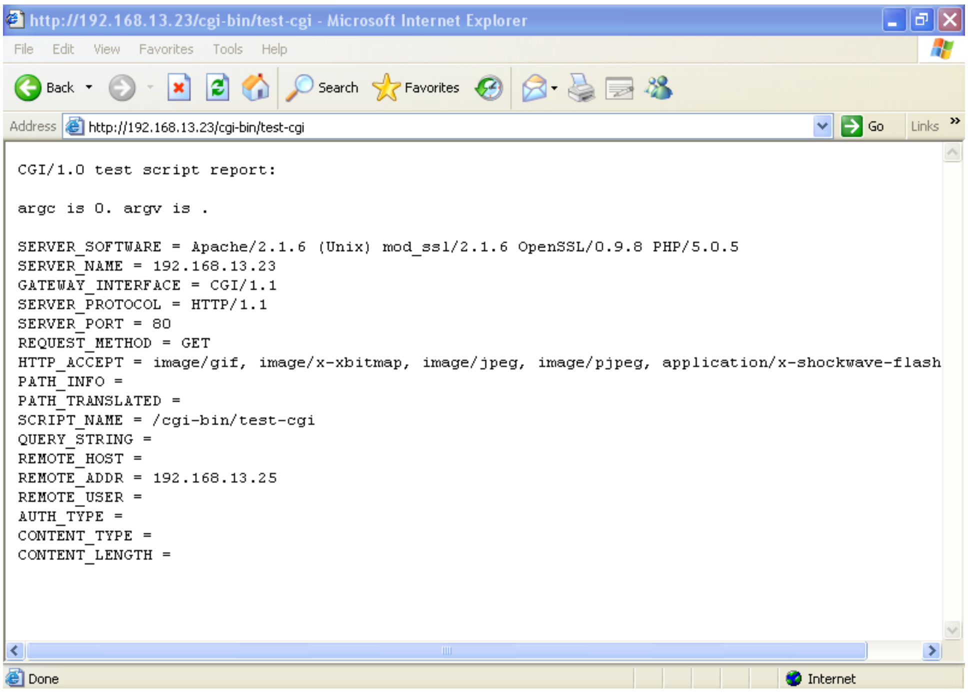
Save your CGI page to the following directory:

/home/httpd/cgi-bin/

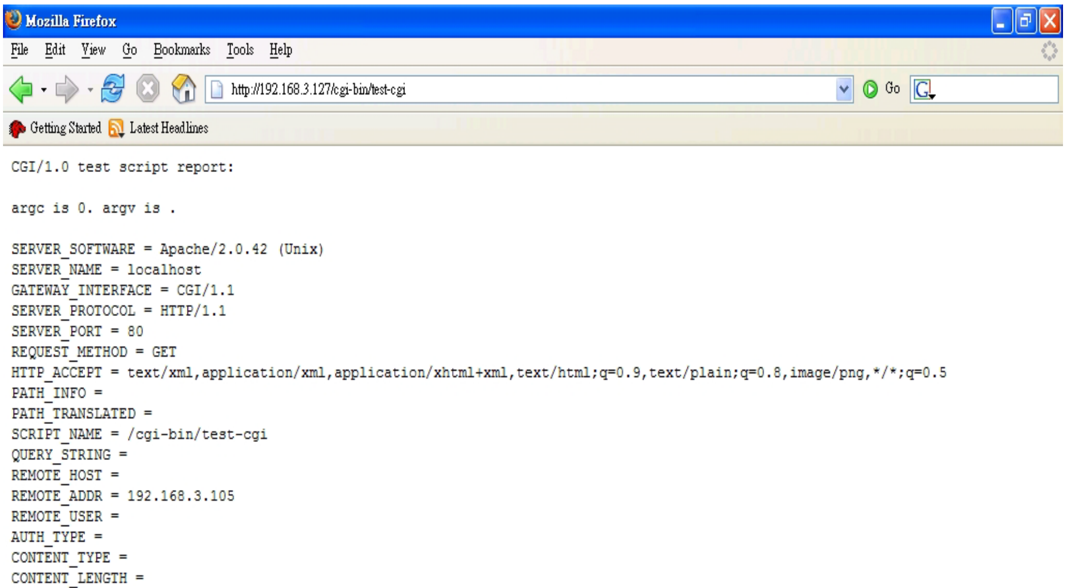
Before you modify the homepage, use a browser (such as Microsoft Internet Explore or Mozilla Firefox) from your PC to test if the Apache Web Server is working. Type the LAN1 IP address in the browser's address box to open the homepage. E.g., if the default IP address is still active, type **http://192.168.3.127** in the address box.



To open the default CGI page, type **http://192.168.3.127/cgi-bin/test-cgi** in your browser's address box.



To open the default CGI test script report page, type **http://192.168.3.127/cgi-bin/test-cgi** in your browser's address box.



NOTE The CGI function is enabled by default. If you want to disable the function, modify the file **/etc/apache/conf/httpd.conf**. When you develop your own CGI application, make sure your CGI file is executable.

```
192.168.3.127 - PuTTY
root@Moxa:/home/httpd/cgi-bin# ls -al
drwxr-xr-x   2 root  root       0 Aug 24 1999
drwxr-xr-x   5 root  root       0 Nov  5 16:16
-rwxr-xr-x   1 root  root     757 Aug 24 1999 test-cgi
root@Moxa:/home/httpd/cgi-bin#
```

Install PHP for Apache Web Server

This embedded computer supports the PHP option. However, since the PHP file is 3 MB, it is not installed by default. To install it yourself, first make sure there is enough free space (at least 3 MB) on your embedded flash ROM).

Step 1: Check that you have enough free space

```
192.168.3.127 - PuTTY
root@Moxa:/bin# df -h
Filesystem              Size     Used Available Use% Mounted on
/dev/mtdblock2           8.0M      6.0M      2.0M  75% /
/dev/ram0              499.0k    17.0k    457.0k   4% /var
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /tmp
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /home
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /etc
tmpfs                   30.4M        0     30.4M   0% /dev/shm
root@Moxa:/bin#
```

To check that the /dev/mtdblock3 free space is greater than 3 MB.

Step 2: Type 'upramdisk' to get the free space ram disk to save the package.

```
192.168.3.127 - PuTTY
root@Moxa:/bin# upramdisk
root@Moxa:/bin# df -h
Filesystem              Size     Used Available Use% Mounted on
/dev/mtdblock2           8.0M      6.0M      2.0M  75% /
/dev/ram0              499.0k    18.0k    456.0k   4% /var
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /tmp
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /home
/dev/mtdblock3           6.0M   488.0k      5.5M   8% /etc
tmpfs                   30.4M        0     30.4M   0% /dev/shm
/dev/ram1               16.0M     1.0k     15.1M   0% /var/ramdisk
root@Moxa:/bin#
```

Step 3: Download the PHP package from the CD-ROM. You can find the package in CD-ROM/target/php/php.tar.gz

```
192.168.3.127 - PuTTY
root@Moxa:/bin# cd /mnt/ramdisk
root@Moxa:/mnt/ramdisk# ftp 192.168.27.130
Connected to 192.168.27.130.
220 (vsFTPd 2.0.1)
Name (192.168.27.130:root): root
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /tmp
250 Directory successfully changed.
ftp> bin
200 Switching to Binary mode.
ftp> get php.tar.gz
local: php.tar.gz remote: php.tar.gz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for php.tar.gz (1789032 bytes).
226 File send OK.
1789032 bytes received in 0.66 secs (2.6e+03 Kbytes/sec)
ftp>
```

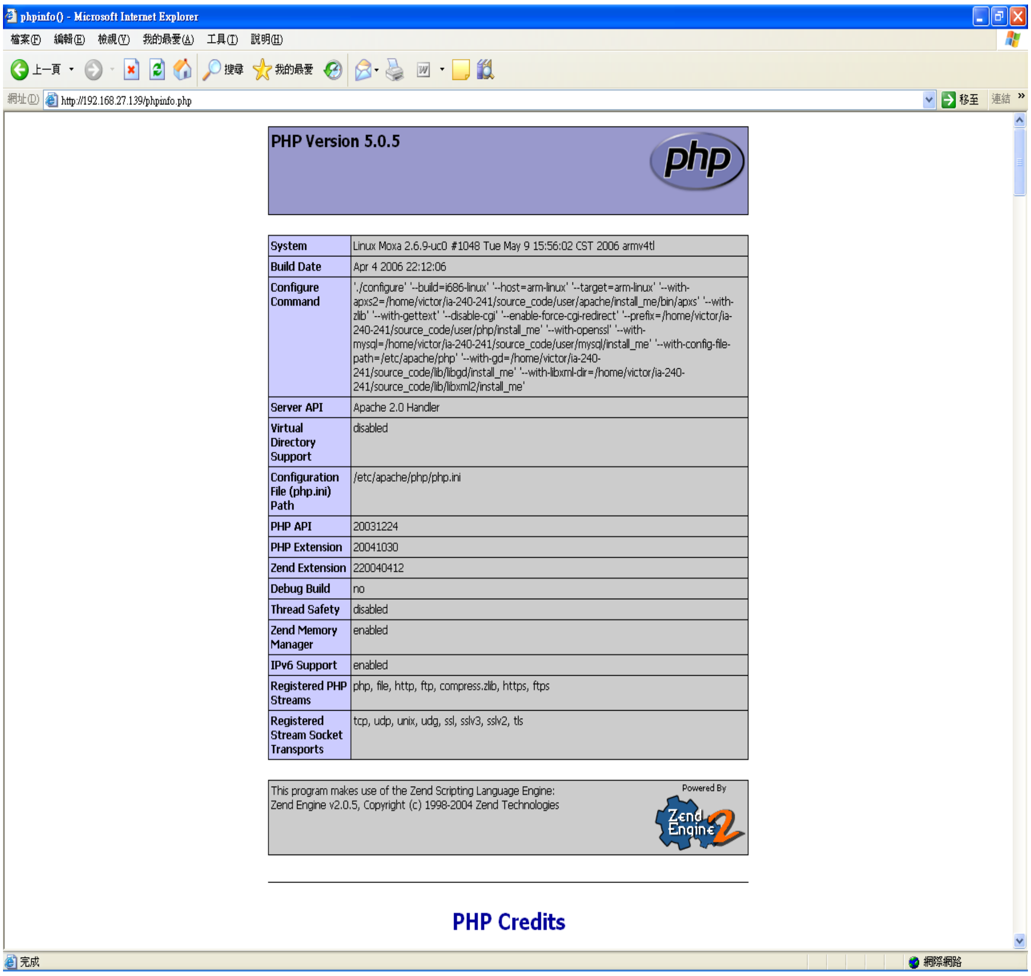
Step 4: utar the package. To do this, type the command 'tar xvzf php.tar.gz'

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# tar xvzf php.tar.gz
envvars
envvars.old
httpd.conf
httpd.conf.old
install.sh
lib
lib/libmysqlclient.so.15
lib/libpng.so.2
lib/libphp5.so
lib/libmysqlclient.so.15.0.0
lib/libgd.so
lib/libxml2.so.2.6.22
lib/libgd.so.2.0.0
lib/libjpeg.so
lib/libxml2.so.2
lib/libgd.so.2
php
php/php.ini
phpinfo.php
root@Moxa:/mnt/ramdisk#
```

Step 5: Run 'install.sh' and select to install php

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# ./install.sh
Press the number:
1. Install PHP package
2. Uninstall PHP package
3. Exit.
1
Start to install PHP. Please wait ...
Starting web server: apache.
PHP install sucess.
root@Moxa:/mnt/ramdisk#
```

Step 6: Test it. Use the browser to access http://192.168.3.127/phpinfo.php



If you want to uninstall PHP, follow steps 2 to 5 but select the uninstall option.

IPTABLES

IPTABLES is an administrative tool for setting up, maintaining, and inspecting the Linux kernel's IP packet filter rule tables. Several different tables are defined, with each table containing built-in chains and user-defined chains.

Each chain is a list of rules that apply to a certain type of packet. Each rule specifies what to do with a matching packet. A rule (such as a jump to a user-defined chain in the same table) is called a "target."

The IA240/241 supports 3 types of IPTABLES table: **Filter** tables, **NAT** tables, and **Mangle** tables:

A. **Filter Table**—includes three chains:

INPUT chain
OUTPUT chain
FORWARD chain

B. **NAT Table**—includes three chains:

PREROUTING chain—transfers the destination IP address (DNAT)

POSTROUTING chain—works after the routing process and before the Ethernet device process to transfer the source IP address (SNAT)

OUTPUT chain—produces local packets

sub-tables

Source NAT (SNAT)—changes the first source packet IP address

Destination NAT (DNAT)—changes the first destination packet IP address

MASQUERADE—a special form for SNAT. If one host can connect to Internet, then other computers that connect to this host can connect to the Internet when the computer does not have an actual IP address.

REDIRECT—a special form of DNAT that re-sends packets to a local host independent of the destination IP address.

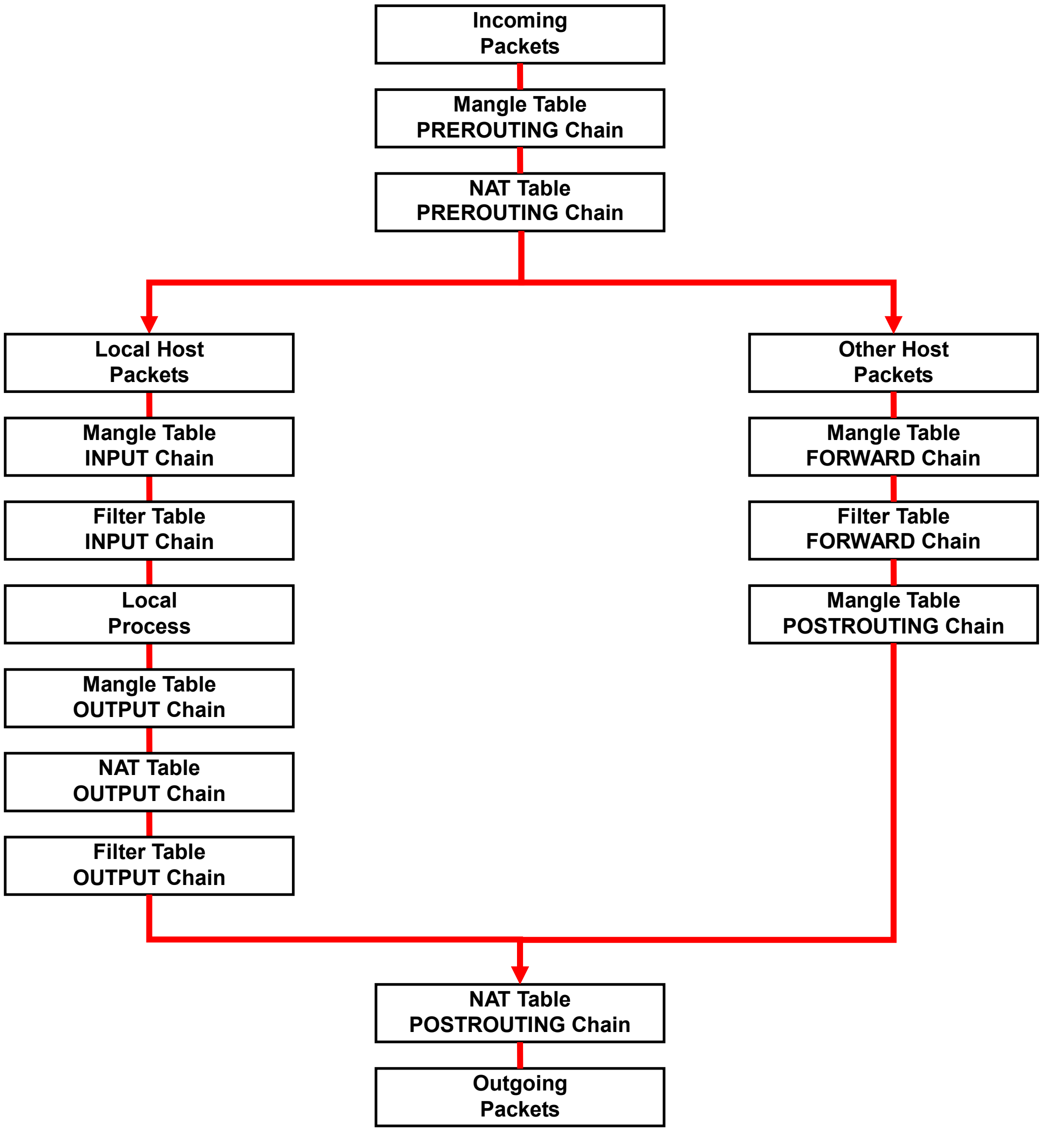
C. **Mangle Table**—includes two chains

PREROUTING chain—pre-processes packets before the routing process.

OUTPUT chain—processes packets after the routing process.

It has three extensions—TTL, MARK, TOS.

The following figure shows the IPTABLES hierarchy.



The IA240/241 supports the following sub-modules. Be sure to use the module that matches your application.

ip_conntrack	ipt_MARK	ipt_ah	ipt_state
ip_conntrack_ftp	ipt_MASQUERADE	ipt_esp	ipt_tcpmss
ipt_conntrack_irc	ipt_MIRROT	ipt_length	ipt_tos
ip_nat_ftp	ipt_REDIRECT	ipt_limit	ipt_ttl
ip_nat_irc	ipt_REJECT	ipt_mac	ipt_unclean
ip_nat_snmp_basic	ipt_TCPMSS	ipt_mark	
ip_queue	ipt_TOS	ipt_multiport	
ipt_LOG	ipt_ULOG	ipt_owner	

NOTE The IA240/241 does NOT support IPV6 and ipchains.

The basic syntax to enable and load an IPTABLES module is as follows:

```
#lsmod
#modprobe ip_tables
#modprobe iptable_filter
```

Use **lsmod** to check if the ip_tables module has already been loaded in the IA240/241. Use **modprobe** to insert and enable the module.

Use the following command to load the modules (iptable_filter, iptable_mangle, iptable_nat):

```
#modprobe iptable_filter
```

NOTE IPTABLES plays the role of packet filtering or NAT. Take care when setting up the IPTABLES rules. If the rules are not correct, remote hosts that connect via a LAN or PPP may be denied access. We recommend using the Serial Console to set up the IPTABLES.

Click on the following links for more information about iptables.

http://www.linuxguruz.com/iptables/
http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html

Since the IPTABLES command is very complex, to illustrate the IPTABLES syntax we have divided our discussion of the various rules into three categories: **Observe and erase chain rules**, **Define policy rules,** and **Append or delete rules**.

Observe and erase chain rules

Usage:

```
# iptables [-t tables] [-L] [-n]
```

-t tables: Table to manipulate (default: 'filter'); example: nat or filter.
-L [chain]: ListList all rules in selected chains. If no chain is selected, all chains are listed.
-n: Numeric output of addresses and ports.

```
# iptables [-t tables] [-FXZ]
```

-F: Flush the selected chain (all the chains in the table if none is listed).
-X: Delete the specified user-defined chain.
-Z: Set the packet and byte counters in all chains to zero.

Examples:

```
# iptables -L -n
```

In this example, since we do not use the -t parameter, the system uses the default 'filter' table. Three chains are included: INPUT, OUTPUT, and FORWARD. INPUT chains are accepted automatically, and all connections are accepted without being filtered.

```
#iptables –F
#iptables –X
#iptables -Z
```

Define policy for chain rules

Usage:

```
# iptables [-t tables] [-P] [INPUT, OUTPUT, FORWARD, PREROUTING, OUTPUT, POSTROUTING]
[ACCEPT, DROP]
```

-P: Set the policy for the chain to the given target.
INPUT: For packets coming into the IA240/241.
OUTPUT: For locally-generated packets.
FORWARD: For packets routed out through the IA240/241.
PREROUTING: To alter packets as soon as they come in.
POSTROUTING: To alter packets as they are about to be sent out.

Examples:

```
#iptables –P INPUT DROP
#iptables –P OUTPUT ACCEPT
#iptables –P FORWARD ACCEPT
#iptables –t nat –P PREROUTING ACCEPT
#iptables –t nat –P OUTPUT ACCEPT
#iptables -t nat –P POSTROUTING ACCEPT
```

In this example, the policy accepts outgoing packets and denies incoming packets.

Append or delete rules:

Usage:

```
# iptables [-t table] [-AI] [INPUT, OUTPUT, FORWARD] [-io interface] [-p tcp, udp, icmp,
all] [-s IP/network] [--sport ports] [-d IP/network] [--dport ports] –j [ACCEPT. DROP]
```

-A: Append one or more rules to the end of the selected chain.
-I: Insert one or more rules in the selected chain as the given rule number.
-i: Name of an interface via which a packet is going to be received.
-o: Name of an interface via which a packet is going to be sent.
-p: The protocol of the rule or of the packet to check.
-s: Source address (network name, host name, network IP address, or plain IP address).
--sport: Source port number.
-d: Destination address.
--dport: Destination port number.
-j: Jump target. Specifies the target of the rules; i.e., how to handle matched packets. For example, ACCEPT the packet, DROP the packet, or LOG the packet.

Examples:

Example 1: Accept all packets from lo interface.

```
# iptables –A INPUT –i lo –j ACCEPT
```

Example 2: Accept TCP packets from 192.168.0.1.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.0.1 –j ACCEPT
```

Example 3: Accept TCP packets from Class C network 192.168.1.0/24.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.1.0/24 –j ACCEPT
```

Example 4: Drop TCP packets from 192.168.1.25.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.1.25 –j DROP
```

Example 5: Drop TCP packets addressed for port 21.

```
# iptables –A INPUT –i eth0 –p tcp --dport 21 –j DROP
```

Example 6: Accept TCP packets from 192.168.0.24 to IA240/241's port 137, 138, 139

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.0.24 --dport 137:139 –j ACCEPT
```

Example 7: Log TCP packets that visit IA240/241's port 25.

```
# iptables -A INPUT -i eth0 -p tcp --dport 25 -j LOG
```

Example 8: Drop all packets from MAC address 01:02:03:04:05:06.

```
# iptables -A INPUT -i eth0 -p all -m mac -mac-source 01:02:03:04:05:06 -j DROP
```

NOTE: In Example 8, remember to issue the command **#modprobe ipt_mac** first to load module **ipt_mac**.

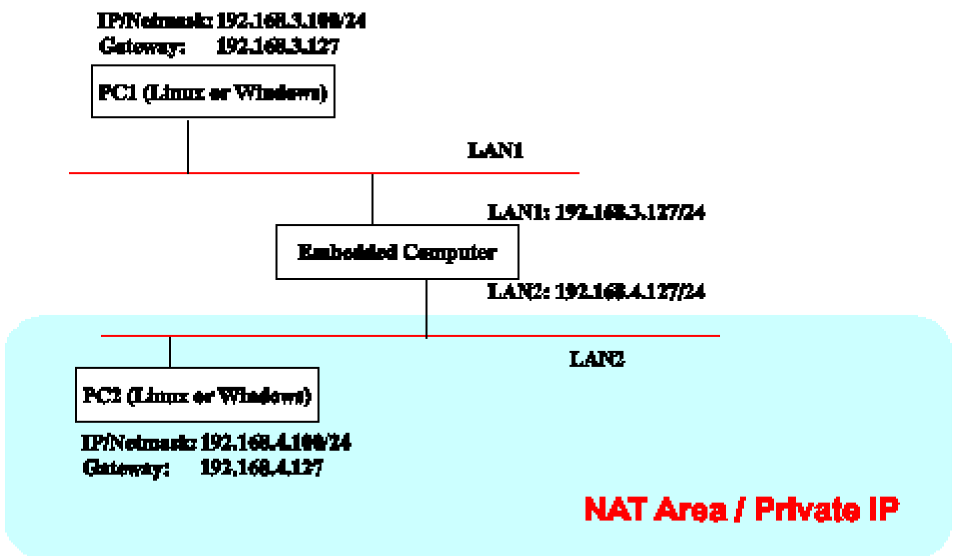
NAT

NAT (Network Address Translation) protocol translates IP addresses used on one network to different IP addresses used on another network. One network is designated the inside network and the other is the outside network. Typically, the IA240/241 connects several devices on a network and maps local inside network addresses to one or more global outside IP addresses, and un-maps the global IP addresses on incoming packets back into local IP addresses.

NOTE Click on the following link for more information about iptables and NAT:
http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html

NAT Example

The IP address of LAN1 is changed to 192.168.3.127 (you will need to load the module ipt_MASQUERADE):




```
1.  #echo 1 > /proc/sys/net/ipv4/ip_forward
2.  #modprobe  ip_tables
3.  #modprobe  iptable_ filter
4.  #modprobe  ip_conntrack
5.  #modprobe  iptable_nat
6.  #modprobe  ipt_MASQUERADE
7.  #iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.168.3.127
8.  #iptables -t nat -A POSTROUTING -o eth0 -s 192.168.3.0/24 -j MASQUERADE
```

Enabling NAT at Bootup

In most real world situations, you will want to use a simple shell script to enable NAT when the IA240/241 boots up. The following script is an example.

```
#!/bin/bash
# If you put this shell script in the /home/nat.sh
# Remember to chmod 744 /home/nat.sh
# Edit the rc.local file to make this shell startup automatically.
# vi /etc/rc.d/rc.local
# Add a line in the end of rc.local /home/nat.sh
EXIF='eth0'  #This is an external interface for setting up a valid IP address.
EXNET='192.168.4.0/24'  #This is an internal network address.
# Step 1. Insert modules.
# Here 2> /dev/null means the standard error messages will be dump to null device.
modprobe ip_tables  2> /dev/null
modprobe ip_nat_ftp  2> /dev/null
modprobe ip_nat_irc  2> /dev/null
modprobe ip_conntrack  2> /dev/null
modprobe ip_conntrack_ftp  2> /dev/null
modprobe ip_conntrack_irc  2> /dev/null
# Step 2. Define variables, enable routing and erase default rules.
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
export PATH
echo "1" > /proc/sys/net/ipv4/ip_forward
/bin/iptables -F
/bin/iptables -X
/bin/iptables -Z
/bin/iptables -F -t nat
/bin/iptables -X -t nat
/bin/iptables -Z -t nat
/bin/iptables -P INPUT   ACCEPT
/bin/iptables -P OUTPUT  ACCEPT
/bin/iptables -P FORWARD ACCEPT
/bin/iptables -t nat -P PREROUTING  ACCEPT
/bin/iptables -t nat -P POSTROUTING ACCEPT
/bin/iptables -t nat -P OUTPUT     ACCEPT
# Step 3. Enable IP masquerade.
```

Dial-up Service—PPP

PPP (Point to Point Protocol) is used to run IP (Internet Protocol) and other network protocols over a serial link. PPP can be used for direct serial connections (using a null-modem cable) over a Telnet link, and links established using a modem over a telephone line.

Modem / PPP access is almost identical to connecting directly to a network through the IA240/241's Ethernet port. Since PPP is a peer-to-peer system, the IA240/241 can also use PPP to link two networks (or a local network to the Internet) to create a Wide Area Network (WAN).

NOTE Click on the following links for more information about ppp:
http://tldp.org/HOWTO/PPP-HOWTO/index.html
http://axion.physics.ubc.ca/ppp-linux.html

The pppd daemon is used to connect to a PPP server from a Linux system. For detailed information about pppd see the man page.

Example 1: Connecting to a PPP server over a simple dial-up connection

The following command is used to connect to a PPP server by modem. Use this command for old ppp servers that prompt for a login name (replace *username* with the correct name) and password (replace *password* with the correct password). Note that *debug* and *defaultroute 192.1.1.17* are optional.

```
#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ogin: username word: password'
/dev/ttyM0 115200 debug crtscts modem defaultroute
```

If the PPP server does not prompt for the username and password, the command should be entered as follows. Replace *username* with the correct username and replace *password* with the correct password.

```
#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ' user username password password
/dev/ttyM0 115200 crtscts modem
```

The pppd options are described below:

`connect 'chat etc...'`
This option gives the command to contact the PPP server. The 'chat' program is used to dial a remote computer. The entire command is enclosed in single quotes because pppd expects a one-word argument for the 'connect' option. The options for 'chat' are given below:

`-v`
verbose mode; log what we do to syslog

`" "`
Double quotes—don't wait for a prompt, but instead do ... (note that you must include a space after the second quotation mark)

`ATDT5551212`
Dial the modem, and then ...

`CONNECT`
Wait for an answer.

`" "`
Send a return (null text followed by the usual return)

`ogin: username word: password`
Log in with *username* and *password*.

Refer to the chat man page, chat.8, for more information about the chat utility.

`/dev/`
Specify the callout serial port.

`115200`
The baudrate.

`debug`
Log status in syslog.

`crtscts`
Use hardware flow control between computer and modem (at 115200 this is a must).

`modem`
Indicates that this is a modem device; pppd will hang up the phone before and after making the call.

`defaultroute`
Once the PPP link is established, make it the default route; if you have a PPP link to the Internet, this is probably what you want.

`192.1.1.17`
This is a degenerate case of a general option of the form x.x.x.x:y.y.y.y. Here x.x.x.x is the local IP address and y.y.y.y is the IP address of the remote end of the PPP connection. If this option is not specified, or if just one side is specified, then x.x.x.x defaults to the IP address associated with the local machine's hostname (located in **/etc/hosts**), and y.y.y.y is determined by the remote machine.

Example 2: Connecting to a PPP server over a hard-wired link

If a username and password are not required, use the following command (note that *noipdefault* is optional):

```
#pppd connect 'chat –v" " " " ' noipdefault /dev/ttyM0 19200 crtscts
```

If a username and password is required, use the following command (note that *noipdefault* is optional, and *root* is both the username and password):

```
#pppd connect 'chat –v" " " " ' user root password root noipdefault
/dev/ttyM0 19200 crtscts
```

How to check the connection

Once you've set up a PPP connection, there are some steps you can take to test the connection. First, type:

```
/sbin/ifconfig
```

(The folder **ifconfig** may be located elsewhere, depending on your distribution.) You should be able to see all the network interfaces that are UP. ppp0 should be one of them, and you should recognize the first IP address as your own, and the "P-t-P address" (or point-to-point address) the address of your server. Here's what it looks like on one machine:

```
lo      Link encap Local Loopback
        inet addr 127.0.0.1    Bcast 127.255.255.255    Mask 255.0.0.0
        UP LOOPBACK RUNNING    MTU 2000    Metric 1
        RX packets 0 errors 0 dropped 0 overrun 0

ppp0    Link encap Point-to-Point Protocol
        inet addr 192.76.32.3    P-t-P 129.67.1.165    Mask 255.255.255.0
        UP POINTOPOINT RUNNING    MTU 1500    Metric 1
        RX packets 33 errors 0 dropped 0 overrun 0
        TX packets 42 errors 0 dropped 0 overrun 0
```

Now, type:

```
ping z.z.z.z
```

where z.z.z.z is the address of your name server. This should work. Here's what the response could look like:

```
waddington:~$p ping 129.67.1.165
PING 129.67.1.165 (129.67.1.165): 56 data bytes
64 bytes from 129.67.1.165: icmp_seq=0 ttl=225 time=268 ms
64 bytes from 129.67.1.165: icmp_seq=1 ttl=225 time=247 ms
64 bytes from 129.67.1.165: icmp_seq=2 ttl=225 time=266 ms
^C
--- 129.67.1.165 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 247/260/268 ms
waddington:~$
```

Try typing:

```
netstat -nr
```

This should show three routes, something like this:

Kernel routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	iface
129.67.1.165	0.0.0.0	255.255.255.255	UH	0	0	6	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	129.67.1.165	0.0.0.0	UG	0	0	6298	ppp0

If your output looks similar but doesn’t have the destination 0.0.0.0 line (which refers to the default route used for connections), you may have run pppd without the ‘defaultroute’ option. At this point you can try using Telnet, ftp, or finger, bearing in mind that you’ll have to use numeric IP addresses unless you’ve set up /etc/resolv.conf correctly.

Setting up a Machine for Incoming PPP Connections

This first example applies to using a modem, and requiring authorization with a username and password.

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2 login auth
```

You should also add the following line to the file **/etc/ppp/pap-secrets**:

```
*    *    ""    *
```

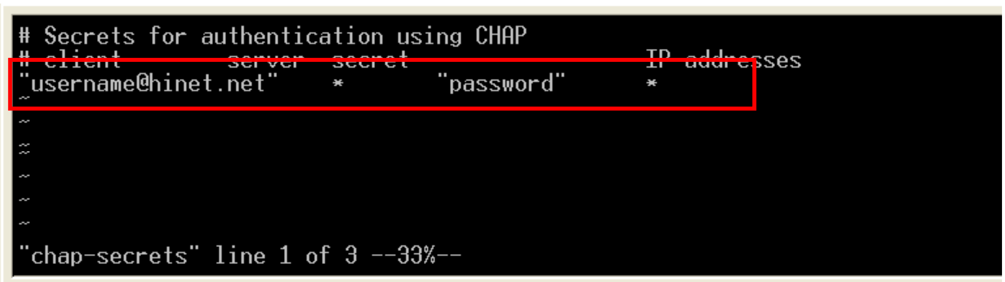
The first star (*) lets everyone login. The second star (*) lets every host connect. The pair of double quotation marks (““) is to use the file **/etc/passwd** to check the password. The last star (*) is to let any IP connect.

The following example does not check the username and password:

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2
```

PPPoE

1. Connect IA240/241’s LAN port to an ADSL modem with a cross-over cable, HUB, or switch.
2. Login to the IA240/241 as the root user.
3. Edit the file **/etc/ppp/chap-secrets** and add the following:
   “username@hinet.net”*“password”*



“username@hinet.net” is the username obtained from the ISP to log in to the ISP account. **“password”** is the corresponding password for the account.

4. Edit the file **/etc/ppp/pap-secrets** and add the following:
   "username@hinet.net"*"password"*

```
# password if you don't use the login option of pppd! The mgetty Debian
# package already provides this option; make sure you don't change that.

# INBOUND connections

# Every regular user can use PPP and has to use passwords from /etc/passwd
*       hostname        ""              *
"username@hinet.net"    *       "password"      *

# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest   hostname        "*"     -
master  hostname        "*"     -
root    hostname        "*"     -
support hostname        "*"     -
stats   hostname        "*"     -

# OUTBOUND connections

# Here you should add your userid password to connect to your providers via
# PAP. The * means that the password is to be used for ANY host you connect
# to. Thus you do not have to worry about the foreign machine name. Just
# replace password with your password.
"pap-secrets" line 1 of 42 --2%--
```

"username@hinet.net" is the username obtained from the ISP to log in to the ISP account. **"password"** is the corresponding password for the account.

5. Edit the file **/etc/ppp/options** and add the following line:
   plugin pppoe

```
# terminated because it was idle.
#holdoff <n>

# Wait for up n milliseconds after the connect script finishes for a valid
# PPP packet from the peer.  At the end of this time, or when a valid PPP
# packet is received from the peer, pppd will commence negotiation by
# sending its first LCP packet.  The default value is 1000 (1 second).
# This wait period only applies if the connect or pty option is used.
#connect-delay <n>
plugin pppoe.so

# ---<End of File>---
~
~
~
~
~
~
~
~
~
~
~
"options" line 1 of 342 --0%--
```

6. Add one of two files: **/etc/ppp/options.eth0** or **/etc/ppp/options.eth1**. The choice depends on which LAN is connected to the ADSL modem. If you use LAN1 to connect to the ADSL modem, then add **/etc/ppp/options.eth0**. If you use LAN2 to connect to the ADSL modem, then add **/etc/ppp/options.eth1**. The file context is shown below:

Type your username (the one you set in the **/etc/ppp/pap-secrets** and **/etc/ppp/chap-secrets** files) after the "name" option. You may add other options as desired.

7. Set up DNS

   If you are using DNS servers supplied by your ISP, edit the file **/etc/resolv.conf** by adding the following lines of code:

   nameserver ip_addr_of_first_dns_server

   nameserver ip_addr_of_second_dns_server

   For example:

   nameserver 168..95.1.1

   nameserver 139.175.10.20

8. Use the following command to create a pppoe connection:

   pppd eth0

   The eth0 is what is connected to the ADSL modem LAN port. The example above uses LAN1. To use LAN2, type:

   pppd eth1

9. Type **ifconfig ppp0** to check if the connection is OK or has failed. If the connection is OK, you will see information about the ppp0 setting for the IP address. Use ping to test the IP.
10. If you want to disconnect it, use the kill command to kill the pppd process.

NFS (Network File System)

The Network File System (NFS) is used to mount a disk partition on a remote machine, as if it were on a local hard drive, allowing fast, seamless sharing of files across a network. NFS allows users to develop applications for the IA240/241, without worrying about the amount of disk space that will be available. The IA240/241 supports NFS protocol for client.

NOTE Click on the following links for more information about NFS:
http://www.tldp.org/HOWTO/NFS-HOWTO/index.html
http://nfs.sourceforge.net/nfs-howto/client.html
http://nfs.sourceforge.net/nfs-howto/server.html

Setting up the IA240/241 as an NFS Client

The following procedure is used to mount a remote NFS Server.

1. To know the NFS Server's shared directory.
2. Establish a mount point on the NFS Client site.
3. Mount the remote directory to a local directory.

```
#mkdir  -p  /home/nfs/public
#mount  -t  nfs  NFS_Server(IP):/directory  /mount/point
```

Example

```
#mount -t nfs 192.168.3.100:/home/public  /home/nfs/public
```

Mail

smtpclient is a minimal SMTP client that takes an email message body and passes it on to an SMTP server. It is suitable for applications that use email to send alert messages or important logs to a specific user.

NOTE Click on the following link for more information about smtpclient:
http://www.engelschall.com/sw/smtpclient/

To send an email message, use the 'smtpclient' utility, which uses SMTP protocol. Type **`#smtpclient -help`** to see the help message.

Example:

```
smtpclient -s test -f sender@company.com -S IP_address receiver@company.com
< mail-body-message
```

- **`-s:`** The mail subject.
- **`-f:`** Sender's mail address
- **`-S:`** SMTP server IP address

The last mail address **receiver@company.com** is the receiver's e-mail address. **mail-body-message** is the mail content. The last line of the body of the message should contain ONLY the period '.' character.

You will need to add your hostname to the file **/etc/hosts**.

SNMP

The IA240/241 has built-in SNMP V1 (Simple Network Management Protocol) agent software. It supports RFC1317 RS-232 like group and RFC 1213 MIB-II.

The following simple example allows you to use an SNMP browser on the host site to query the IA240/241, which is the SNMP agent. The IA240/241 will respond.

```
***** SNMP QUERY STARTED *****
    1: sysDescr.0 (octet string) Version 1.0
    2: sysObjectID.0 (object identifier) enterprises.8691.12.240
    3: sysUpTime.0 (timeticks) 0 days 03h:50m:11s.00th (1381100)
    4: sysContact.0 (octet string) Moxa Systems Co., LDT.
    5: sysName.0 (octet string) Moxa
    6: sysLocation.0 (octet string) Unknown
    7: sysServices.0 (integer) 6
    8: ifNumber.0 (integer) 6
    9: ifIndex.1 (integer) 1
    10: ifIndex.2 (integer) 2
    11: ifIndex.3 (integer) 3
    12: ifIndex.4 (integer) 4
    13: ifIndex.5 (integer) 5
    14: ifIndex.6 (integer) 6
    15: ifDescr.1 (octet string) eth0
    16: ifDescr.2 (octet string) eth1
    17: ifDescr.3 (octet string) Serial port 0
    18: ifDescr.4 (octet string) Serial port 1
    19: ifDescr.5 (octet string) Serial port 2
    20: ifDescr.6 (octet string) Serial port 3
    21: ifType.1 (integer) ethernet-csmacd(6)
    22: ifType.2 (integer) ethernet-csmacd(6)
    23: ifType.3 (integer) other(1)
    24: ifType.4 (integer) other(1)
    25: ifType.5 (integer) other(1)
    26: ifType.6 (integer) other(1)
    27: ifMtu.1 (integer) 1500
    28: ifMtu.2 (integer) 1500
    29: ifMtu.3 (integer) 0
    30: ifMtu.4 (integer) 0
    31: ifMtu.5 (integer) 0
    32: ifMtu.6 (integer) 0
    33: ifSpeed.1 (gauge) 100000000
    34: ifSpeed.2 (gauge) 100000000
    35: ifSpeed.3 (gauge) 38400
    36: ifSpeed.4 (gauge) 38400
    37: ifSpeed.5 (gauge) 38400
    38: ifSpeed.6 (gauge) 38400
    39: ifPhysAddress.1 (octet string) 00.90.E8.10.02.41 (hex)
    40: ifPhysAddress.2 (octet string) 00.90.E8.10.02.40 (hex)
    41: ifPhysAddress.3 (octet string) 00 (hex)
    42: ifPhysAddress.4 (octet string) 00 (hex)
    43: ifPhysAddress.5 (octet string) 00 (hex)
    44: ifPhysAddress.6 (octet string) 00 (hex)
    45: ifAdminStatus.1 (integer) up(1)
    46: ifAdminStatus.2 (integer) up(1)
    47: ifAdminStatus.3 (integer) down(2)
    48: ifAdminStatus.4 (integer) down(2)
    49: ifAdminStatus.5 (integer) down(2)
    50: ifAdminStatus.6 (integer) down(2)
    51: ifOperStatus.1 (integer) up(1)
    52: ifOperStatus.2 (integer) up(1)
    53: ifOperStatus.3 (integer) down(2)
    54: ifOperStatus.4 (integer) down(2)
    55: ifOperStatus.5 (integer) down(2)
    56: ifOperStatus.6 (integer) down(2)
    57: ifLastChange.1 (timeticks) 0 days 00h:00m:00s.00th (0)
```

58: ifLastChange.2 (timeticks) 0 days 00h:00m:00s.00th (0)
59: ifLastChange.3 (timeticks) 0 days 00h:00m:00s.00th (0)
60: ifLastChange.4 (timeticks) 0 days 00h:00m:00s.00th (0)
61: ifLastChange.5 (timeticks) 0 days 00h:00m:00s.00th (0)
62: ifLastChange.6 (timeticks) 0 days 00h:00m:00s.00th (0)
63: ifInOctets.1 (counter) 25511
64: ifInOctets.2 (counter) 2240203
65: ifInOctets.3 (counter) 0
66: ifInOctets.4 (counter) 0
67: ifInOctets.5 (counter) 0
68: ifInOctets.6 (counter) 0
69: ifInUcastPkts.1 (counter) 254
70: ifInUcastPkts.2 (counter) 28224
71: ifInUcastPkts.3 (counter) 0
72: ifInUcastPkts.4 (counter) 0
73: ifInUcastPkts.5 (counter) 0
74: ifInUcastPkts.6 (counter) 0
75: ifInNUcastPkts.1 (counter) 0
76: ifInNUcastPkts.2 (counter) 0
77: ifInNUcastPkts.3 (counter) 0
78: ifInNUcastPkts.4 (counter) 0
79: ifInNUcastPkts.5 (counter) 0
80: ifInNUcastPkts.6 (counter) 0
81: ifInDiscards.1 (counter) 0
82: ifInDiscards.2 (counter) 0
83: ifInDiscards.3 (counter) 0
84: ifInDiscards.4 (counter) 0
85: ifInDiscards.5 (counter) 0
86: ifInDiscards.6 (counter) 0
87: ifInErrors.1 (counter) 0
88: ifInErrors.2 (counter) 0
89: ifInErrors.3 (counter) 0
90: ifInErrors.4 (counter) 0
91: ifInErrors.5 (counter) 0
92: ifInErrors.6 (counter) 0
93: ifInUnknownProtos.1 (counter) 0
94: ifInUnknownProtos.2 (counter) 0
95: ifInUnknownProtos.3 (counter) 0
96: ifInUnknownProtos.4 (counter) 0
97: ifInUnknownProtos.5 (counter) 0
98: ifInUnknownProtos.6 (counter) 0
99: ifOutOctets.1 (counter) 51987
100: ifOutOctets.2 (counter) 3832
101: ifOutOctets.3 (counter) 0
102: ifOutOctets.4 (counter) 0
103: ifOutOctets.5 (counter) 0
104: ifOutOctets.6 (counter) 0
105: ifOutUcastPkts.1 (counter) 506
106: ifOutUcastPkts.2 (counter) 42
107: ifOutUcastPkts.3 (counter) 0
108: ifOutUcastPkts.4 (counter) 0
109: ifOutUcastPkts.5 (counter) 0
110: ifOutUcastPkts.6 (counter) 0
111: ifOutNUcastPkts.1 (counter) 0
112: ifOutNUcastPkts.2 (counter) 0
113: ifOutNUcastPkts.3 (counter) 0
114: ifOutNUcastPkts.4 (counter) 0
115: ifOutNUcastPkts.5 (counter) 0
116: ifOutNUcastPkts.6 (counter) 0
117: ifOutDiscards.1 (counter) 0
118: ifOutDiscards.2 (counter) 0
119: ifOutDiscards.3 (counter) 0
120: ifOutDiscards.4 (counter) 0
121: ifOutDiscards.5 (counter) 0
122: ifOutDiscards.6 (counter) 0
123: ifOutErrors.1 (counter) 0
124: ifOutErrors.2 (counter) 0

125: ifOutErrors.3 (counter) 0
126: ifOutErrors.4 (counter) 0
127: ifOutErrors.5 (counter) 0
128: ifOutErrors.6 (counter) 0
129: ifOutQLen.1 (gauge) 1000
130: ifOutQLen.2 (gauge) 1000
131: ifOutQLen.3 (gauge) 0
132: ifOutQLen.4 (gauge) 0
133: ifOutQLen.5 (gauge) 0
134: ifOutQLen.6 (gauge) 0
135: ifSpecific.1 (object identifier) (null-oid) zeroDotZero
136: ifSpecific.2 (object identifier) (null-oid) zeroDotZero
137: ifSpecific.3 (object identifier) (null-oid) zeroDotZero
138: ifSpecific.4 (object identifier) (null-oid) zeroDotZero
139: ifSpecific.5 (object identifier) (null-oid) zeroDotZero
140: ifSpecific.6 (object identifier) (null-oid) zeroDotZero
141: atIfIndex.1.192.168.27.139 (integer) 1
142: atIfIndex.2.192.168.4.127 (integer) 2
143: atPhysAddress.1.192.168.27.139 (octet string) 00.90.E8.10.02.41 (hex)
144: atPhysAddress.2.192.168.4.127 (octet string) 00.90.E8.10.02.40 (hex)
145: atNetAddress.1.192.168.27.139 (ipaddress) 192.168.27.139
146: atNetAddress.2.192.168.4.127 (ipaddress) 192.168.4.127
147: ipForwarding.0 (integer) forwarding(1)
148: ipDefaultTTL.0 (integer) 64
149: ipInReceives.0 (counter) 1289
150: ipInHdrErrors.0 (counter) 0
151: ipInAddrErrors.0 (counter) 0
152: ipForwDatagrams.0 (counter) 9
153: ipInUnknownProtos.0 (counter) 0
154: ipInDiscards.0 (counter) 0
155: ipInDelivers.0 (counter) 1160
156: ipOutRequests.0 (counter) 858
157: ipOutDiscards.0 (counter) 0
158: ipOutNoRoutes.0 (counter) 0
159: ipReasmTimeout.0 (integer) 0
160: ipReasmReqds.0 (counter) 0
161: ipReasmOKs.0 (counter) 0
162: ipReasmFails.0 (counter) 0
163: ipFragOKs.0 (counter) 0
164: ipFragFails.0 (counter) 0
165: ipFragCreates.0 (counter) 0
166: ipAdEntAddr.192.168.27.139 (ipaddress) 192.168.27.139
167: ipAdEntAddr.192.168.4.127 (ipaddress) 192.168.4.127
168: ipAdEntIfIndex.192.168.27.139 (integer) 1
169: ipAdEntIfIndex.192.168.4.127 (integer) 2
170: ipAdEntNetMask.192.168.27.139 (ipaddress) 255.255.255.0
171: ipAdEntNetMask.192.168.4.127 (ipaddress) 255.255.255.0
172: ipAdEntBcastAddr.192.168.27.139 (integer) 1
173: ipAdEntBcastAddr.192.168.4.127 (integer) 1
174: ipAdEntReasmMaxSize.192.168.27.139 (integer) 65535
175: ipAdEntReasmMaxSize.192.168.4.127 (integer) 65535
176: ipRouteDest.192.168.4.0 (ipaddress) 192.168.4.0
177: ipRouteDest.192.168.27.0 (ipaddress) 192.168.27.0
178: ipRouteIfIndex.192.168.4.0 (integer) 2
179: ipRouteIfIndex.192.168.27.0 (integer) 1
180: ipRouteMetric1.192.168.4.0 (integer) 0
181: ipRouteMetric1.192.168.27.0 (integer) 0
182: ipRouteMetric2.192.168.4.0 (integer) -1
183: ipRouteMetric2.192.168.27.0 (integer) -1
184: ipRouteMetric3.192.168.4.0 (integer) -1
185: ipRouteMetric3.192.168.27.0 (integer) -1
186: ipRouteMetric4.192.168.4.0 (integer) -1
187: ipRouteMetric4.192.168.27.0 (integer) -1
188: ipRouteNextHop.192.168.4.0 (ipaddress) 192.168.4.127
189: ipRouteNextHop.192.168.27.0 (ipaddress) 192.168.27.139
190: ipRouteType.192.168.4.0 (integer) direct(3)
191: ipRouteType.192.168.27.0 (integer) direct(3)

192: ipRouteProto.192.168.4.0 (integer) local(2)
193: ipRouteProto.192.168.27.0 (integer) local(2)
194: ipRouteAge.192.168.4.0 (integer) 0
195: ipRouteAge.192.168.27.0 (integer) 0
196: ipRouteMask.192.168.4.0 (ipaddress) 255.255.255.0
197: ipRouteMask.192.168.27.0 (ipaddress) 255.255.255.0
198: ipRouteMetric5.192.168.4.0 (integer) -1
199: ipRouteMetric5.192.168.27.0 (integer) -1
200: ipRouteInfo.192.168.4.0 (object identifier) (null-oid) zeroDotZero
201: ipRouteInfo.192.168.27.0 (object identifier) (null-oid) zeroDotZero
202: ipNetToMediaIfIndex.1.192.168.27.139 (integer) 1
203: ipNetToMediaIfIndex.2.192.168.4.127 (integer) 2
204: ipNetToMediaPhysAddress.1.192.168.27.139 (octet string) 00.90.E8.10.02.41 (hex)
205: ipNetToMediaPhysAddress.2.192.168.4.127 (octet string) 00.90.E8.10.02.40 (hex)
206: ipNetToMediaNetAddress.1.192.168.27.139 (ipaddress) 192.168.27.139
207: ipNetToMediaNetAddress.2.192.168.4.127 (ipaddress) 192.168.4.127
208: ipNetToMediaType.1.192.168.27.139 (integer) static(4)
209: ipNetToMediaType.2.192.168.4.127 (integer) static(4)
210: ipRoutingDiscards.0 (integer) 0
211: icmpInMsgs.0 (counter) 130
212: icmpInErrors.0 (counter) 3
213: icmpInDestUnreachs.0 (counter) 128
214: icmpInTimeExcds.0 (counter) 0
215: icmpInParmProbs.0 (counter) 0
216: icmpInSrcQuenchs.0 (counter) 0
217: icmpInRedirects.0 (counter) 0
218: icmpInEchos.0 (counter) 2
219: icmpInEchoReps.0 (counter) 0
220: icmpInTimestamps.0 (counter) 0
221: icmpInTimestampReps.0 (counter) 0
222: icmpInAddrMasks.0 (counter) 0
223: icmpInAddrMaskReps.0 (counter) 0
224: icmpOutMsgs.0 (counter) 144
225: icmpOutErrors.0 (counter) 0
226: icmpOutDestUnreachs.0 (counter) 135
227: icmpOutTimeExcds.0 (counter) 0
228: icmpOutParmProbs.0 (counter) 0
229: icmpOutSrcQuenchs.0 (counter) 0
230: icmpOutRedirects.0 (counter) 7
231: icmpOutEchos.0 (counter) 0
232: icmpOutEchoReps.0 (counter) 2
233: icmpOutTimestamps.0 (counter) 0
234: icmpOutTimestampReps.0 (counter) 0
235: icmpOutAddrMasks.0 (counter) 0
236: icmpOutAddrMaskReps.0 (counter) 0
237: tcpRtoAlgorithm.0 (integer) other(1)
238: tcpRtoMin.0 (integer) 200
239: tcpRtoMax.0 (integer) 120000
240: tcpMaxConn.0 (integer) -1
241: tcpActiveOpens.0 (counter) 0
242: tcpPassiveOpens.0 (counter) 0
243: tcpAttemptFails.0 (counter) 0
244: tcpEstabResets.0 (counter) 0
245: tcpCurrEstab.0 (gauge) 0
246: tcpInSegs.0 (counter) 0
247: tcpOutSegs.0 (counter) 0
248: tcpRetransSegs.0 (counter) 0
249: tcpConnState.192.168.27.139.1024.0.0.0.0.0 (integer) listen(2)
250: tcpConnState.192.168.4.127.1024.0.0.0.0.0 (integer) listen(2)
251: tcpConnState.192.168.27.139.1025.0.0.0.0.0 (integer) listen(2)
252: tcpConnState.192.168.4.127.1025.0.0.0.0.0 (integer) listen(2)
253: tcpConnState.192.168.27.139.2049.0.0.0.0.0 (integer) listen(2)
254: tcpConnState.192.168.4.127.2049.0.0.0.0.0 (integer) listen(2)
255: tcpConnState.192.168.27.139.1026.0.0.0.0.0 (integer) listen(2)
256: tcpConnState.192.168.4.127.1026.0.0.0.0.0 (integer) listen(2)
257: tcpConnState.192.168.27.139.9.0.0.0.0.0 (integer) listen(2)
258: tcpConnState.192.168.4.127.9.0.0.0.0.0 (integer) listen(2)

259: tcpConnState.192.168.27.139.111.0.0.0.0.0 (integer) listen(2)
260: tcpConnState.192.168.4.127.111.0.0.0.0.0 (integer) listen(2)
261: tcpConnState.192.168.27.139.80.0.0.0.0.0 (integer) listen(2)
262: tcpConnState.192.168.4.127.80.0.0.0.0.0 (integer) listen(2)
263: tcpConnState.192.168.27.139.21.0.0.0.0.0 (integer) listen(2)
264: tcpConnState.192.168.4.127.21.0.0.0.0.0 (integer) listen(2)
265: tcpConnState.192.168.27.139.22.0.0.0.0.0 (integer) listen(2)
266: tcpConnState.192.168.4.127.22.0.0.0.0.0 (integer) listen(2)
267: tcpConnState.192.168.27.139.23.0.0.0.0.0 (integer) listen(2)
268: tcpConnState.192.168.4.127.23.0.0.0.0.0 (integer) listen(2)
269: tcpConnLocalAddress.192.168.27.139.1024.0.0.0.0.0 (ipaddress) 192.168.27.139
270: tcpConnLocalAddress.192.168.4.127.1024.0.0.0.0.0 (ipaddress) 192.168.4.127
271: tcpConnLocalAddress.192.168.27.139.1025.0.0.0.0.0 (ipaddress) 192.168.27.139
272: tcpConnLocalAddress.192.168.4.127.1025.0.0.0.0.0 (ipaddress) 192.168.4.127
273: tcpConnLocalAddress.192.168.27.139.2049.0.0.0.0.0 (ipaddress) 192.168.27.139
274: tcpConnLocalAddress.192.168.4.127.2049.0.0.0.0.0 (ipaddress) 192.168.4.127
275: tcpConnLocalAddress.192.168.27.139.1026.0.0.0.0.0 (ipaddress) 192.168.27.139
276: tcpConnLocalAddress.192.168.4.127.1026.0.0.0.0.0 (ipaddress) 192.168.4.127
277: tcpConnLocalAddress.192.168.27.139.9.0.0.0.0.0 (ipaddress) 192.168.27.139
278: tcpConnLocalAddress.192.168.4.127.9.0.0.0.0.0 (ipaddress) 192.168.4.127
279: tcpConnLocalAddress.192.168.27.139.111.0.0.0.0.0 (ipaddress) 192.168.27.139
280: tcpConnLocalAddress.192.168.4.127.111.0.0.0.0.0 (ipaddress) 192.168.4.127
281: tcpConnLocalAddress.192.168.27.139.80.0.0.0.0.0 (ipaddress) 192.168.27.139
282: tcpConnLocalAddress.192.168.4.127.80.0.0.0.0.0 (ipaddress) 192.168.4.127
283: tcpConnLocalAddress.192.168.27.139.21.0.0.0.0.0 (ipaddress) 192.168.27.139
284: tcpConnLocalAddress.192.168.4.127.21.0.0.0.0.0 (ipaddress) 192.168.4.127
285: tcpConnLocalAddress.192.168.27.139.22.0.0.0.0.0 (ipaddress) 192.168.27.139
286: tcpConnLocalAddress.192.168.4.127.22.0.0.0.0.0 (ipaddress) 192.168.4.127
287: tcpConnLocalAddress.192.168.27.139.23.0.0.0.0.0 (ipaddress) 192.168.27.139
288: tcpConnLocalAddress.192.168.4.127.23.0.0.0.0.0 (ipaddress) 192.168.4.127
289: tcpConnLocalPort.192.168.27.139.1024.0.0.0.0.0 (integer) 1024
290: tcpConnLocalPort.192.168.4.127.1024.0.0.0.0.0 (integer) 1024
291: tcpConnLocalPort.192.168.27.139.1025.0.0.0.0.0 (integer) 1025
292: tcpConnLocalPort.192.168.4.127.1025.0.0.0.0.0 (integer) 1025
293: tcpConnLocalPort.192.168.27.139.2049.0.0.0.0.0 (integer) 2049
294: tcpConnLocalPort.192.168.4.127.2049.0.0.0.0.0 (integer) 2049
295: tcpConnLocalPort.192.168.27.139.1026.0.0.0.0.0 (integer) 1026
296: tcpConnLocalPort.192.168.4.127.1026.0.0.0.0.0 (integer) 1026
297: tcpConnLocalPort.192.168.27.139.9.0.0.0.0.0 (integer) 9
298: tcpConnLocalPort.192.168.4.127.9.0.0.0.0.0 (integer) 9
299: tcpConnLocalPort.192.168.27.139.111.0.0.0.0.0 (integer) 111
300: tcpConnLocalPort.192.168.4.127.111.0.0.0.0.0 (integer) 111
301: tcpConnLocalPort.192.168.27.139.80.0.0.0.0.0 (integer) 80
302: tcpConnLocalPort.192.168.4.127.80.0.0.0.0.0 (integer) 80
303: tcpConnLocalPort.192.168.27.139.21.0.0.0.0.0 (integer) 21
304: tcpConnLocalPort.192.168.4.127.21.0.0.0.0.0 (integer) 21
305: tcpConnLocalPort.192.168.27.139.22.0.0.0.0.0 (integer) 22
306: tcpConnLocalPort.192.168.4.127.22.0.0.0.0.0 (integer) 22
307: tcpConnLocalPort.192.168.27.139.23.0.0.0.0.0 (integer) 23
308: tcpConnLocalPort.192.168.4.127.23.0.0.0.0.0 (integer) 23
309: tcpConnRemAddress.192.168.27.139.1024.0.0.0.0.0 (ipaddress) 0.0.0.0
310: tcpConnRemAddress.192.168.4.127.1024.0.0.0.0.0 (ipaddress) 0.0.0.0
311: tcpConnRemAddress.192.168.27.139.1025.0.0.0.0.0 (ipaddress) 0.0.0.0
312: tcpConnRemAddress.192.168.4.127.1025.0.0.0.0.0 (ipaddress) 0.0.0.0
313: tcpConnRemAddress.192.168.27.139.2049.0.0.0.0.0 (ipaddress) 0.0.0.0
314: tcpConnRemAddress.192.168.4.127.2049.0.0.0.0.0 (ipaddress) 0.0.0.0
315: tcpConnRemAddress.192.168.27.139.1026.0.0.0.0.0 (ipaddress) 0.0.0.0
316: tcpConnRemAddress.192.168.4.127.1026.0.0.0.0.0 (ipaddress) 0.0.0.0
317: tcpConnRemAddress.192.168.27.139.9.0.0.0.0.0 (ipaddress) 0.0.0.0
318: tcpConnRemAddress.192.168.4.127.9.0.0.0.0.0 (ipaddress) 0.0.0.0
319: tcpConnRemAddress.192.168.27.139.111.0.0.0.0.0 (ipaddress) 0.0.0.0
320: tcpConnRemAddress.192.168.4.127.111.0.0.0.0.0 (ipaddress) 0.0.0.0
321: tcpConnRemAddress.192.168.27.139.80.0.0.0.0.0 (ipaddress) 0.0.0.0
322: tcpConnRemAddress.192.168.4.127.80.0.0.0.0.0 (ipaddress) 0.0.0.0
323: tcpConnRemAddress.192.168.27.139.21.0.0.0.0.0 (ipaddress) 0.0.0.0
324: tcpConnRemAddress.192.168.4.127.21.0.0.0.0.0 (ipaddress) 0.0.0.0
325: tcpConnRemAddress.192.168.27.139.22.0.0.0.0.0 (ipaddress) 0.0.0.0

326: tcpConnRemAddress.192.168.4.127.22.0.0.0.0.0 (ipaddress) 0.0.0.0
327: tcpConnRemAddress.192.168.27.139.23.0.0.0.0.0 (ipaddress) 0.0.0.0
328: tcpConnRemAddress.192.168.4.127.23.0.0.0.0.0 (ipaddress) 0.0.0.0
329: tcpConnRemPort.192.168.27.139.1024.0.0.0.0.0 (integer) 0
330: tcpConnRemPort.192.168.4.127.1024.0.0.0.0.0 (integer) 0
331: tcpConnRemPort.192.168.27.139.1025.0.0.0.0.0 (integer) 0
332: tcpConnRemPort.192.168.4.127.1025.0.0.0.0.0 (integer) 0
333: tcpConnRemPort.192.168.27.139.2049.0.0.0.0.0 (integer) 0
334: tcpConnRemPort.192.168.4.127.2049.0.0.0.0.0 (integer) 0
335: tcpConnRemPort.192.168.27.139.1026.0.0.0.0.0 (integer) 0
336: tcpConnRemPort.192.168.4.127.1026.0.0.0.0.0 (integer) 0
337: tcpConnRemPort.192.168.27.139.9.0.0.0.0.0 (integer) 0
338: tcpConnRemPort.192.168.4.127.9.0.0.0.0.0 (integer) 0
339: tcpConnRemPort.192.168.27.139.111.0.0.0.0.0 (integer) 0
340: tcpConnRemPort.192.168.4.127.111.0.0.0.0.0 (integer) 0
341: tcpConnRemPort.192.168.27.139.80.0.0.0.0.0 (integer) 0
342: tcpConnRemPort.192.168.4.127.80.0.0.0.0.0 (integer) 0
343: tcpConnRemPort.192.168.27.139.21.0.0.0.0.0 (integer) 0
344: tcpConnRemPort.192.168.4.127.21.0.0.0.0.0 (integer) 0
345: tcpConnRemPort.192.168.27.139.22.0.0.0.0.0 (integer) 0
346: tcpConnRemPort.192.168.4.127.22.0.0.0.0.0 (integer) 0
347: tcpConnRemPort.192.168.27.139.23.0.0.0.0.0 (integer) 0
348: tcpConnRemPort.192.168.4.127.23.0.0.0.0.0 (integer) 0
349: tcpInErrs.0 (counter) 6
350: tcpOutRsts.0 (counter) 37224
351: udpInDatagrams.0 (counter) 434
352: udpNoPorts.0 (counter) 8
353: udpInErrors.0 (counter) 0
354: udpOutDatagrams.0 (counter) 903
355: udpLocalAddress.192.168.27.139.1024 (ipaddress) 192.168.27.139
356: udpLocalAddress.192.168.4.127.1024 (ipaddress) 192.168.4.127
357: udpLocalAddress.192.168.27.139.2049 (ipaddress) 192.168.27.139
358: udpLocalAddress.192.168.4.127.2049 (ipaddress) 192.168.4.127
359: udpLocalAddress.192.168.27.139.1026 (ipaddress) 192.168.27.139
360: udpLocalAddress.192.168.4.127.1026 (ipaddress) 192.168.4.127
361: udpLocalAddress.192.168.27.139.1027 (ipaddress) 192.168.27.139
362: udpLocalAddress.192.168.4.127.1027 (ipaddress) 192.168.4.127
363: udpLocalAddress.192.168.27.139.9 (ipaddress) 192.168.27.139
364: udpLocalAddress.192.168.4.127.9 (ipaddress) 192.168.4.127
365: udpLocalAddress.192.168.27.139.161 (ipaddress) 192.168.27.139
366: udpLocalAddress.192.168.4.127.161 (ipaddress) 192.168.4.127
367: udpLocalAddress.192.168.27.139.4800 (ipaddress) 192.168.27.139
368: udpLocalAddress.192.168.4.127.4800 (ipaddress) 192.168.4.127
369: udpLocalAddress.192.168.27.139.854 (ipaddress) 192.168.27.139
370: udpLocalAddress.192.168.4.127.854 (ipaddress) 192.168.4.127
371: udpLocalAddress.192.168.27.139.111 (ipaddress) 192.168.27.139
372: udpLocalAddress.192.168.4.127.111 (ipaddress) 192.168.4.127
373: udpLocalPort.192.168.27.139.1024 (integer) 1024
374: udpLocalPort.192.168.4.127.1024 (integer) 1024
375: udpLocalPort.192.168.27.139.2049 (integer) 2049
376: udpLocalPort.192.168.4.127.2049 (integer) 2049
377: udpLocalPort.192.168.27.139.1026 (integer) 1026
378: udpLocalPort.192.168.4.127.1026 (integer) 1026
379: udpLocalPort.192.168.27.139.1027 (integer) 1027
380: udpLocalPort.192.168.4.127.1027 (integer) 1027
381: udpLocalPort.192.168.27.139.9 (integer) 9
382: udpLocalPort.192.168.4.127.9 (integer) 9
383: udpLocalPort.192.168.27.139.161 (integer) 161
384: udpLocalPort.192.168.4.127.161 (integer) 161
385: udpLocalPort.192.168.27.139.4800 (integer) 4800
386: udpLocalPort.192.168.4.127.4800 (integer) 4800
387: udpLocalPort.192.168.27.139.854 (integer) 854
388: udpLocalPort.192.168.4.127.854 (integer) 854
389: udpLocalPort.192.168.27.139.111 (integer) 111
390: udpLocalPort.192.168.4.127.111 (integer) 111
391: rs232Number.0 (integer) 4
392: rs232PortIndex.1 (integer) 1 [1]

393: rs232PortIndex.2 (integer) 2 [2]
394: rs232PortIndex.3 (integer) 3 [3]
395: rs232PortIndex.4 (integer) 4 [4]
396: rs232PortType.1 (integer) rs232(2)
397: rs232PortType.2 (integer) rs232(2)
398: rs232PortType.3 (integer) rs232(2)
399: rs232PortType.4 (integer) rs232(2)
400: rs232PortInSigNumber.1 (integer) 3
401: rs232PortInSigNumber.2 (integer) 3
402: rs232PortInSigNumber.3 (integer) 3
403: rs232PortInSigNumber.4 (integer) 3
404: rs232PortOutSigNumber.1 (integer) 2
405: rs232PortOutSigNumber.2 (integer) 2
406: rs232PortOutSigNumber.3 (integer) 2
407: rs232PortOutSigNumber.4 (integer) 2
408: rs232PortInSpeed.1 (integer) 38400
409: rs232PortInSpeed.2 (integer) 38400
410: rs232PortInSpeed.3 (integer) 38400
411: rs232PortInSpeed.4 (integer) 38400
412: rs232PortOutSpeed.1 (integer) 38400
413: rs232PortOutSpeed.2 (integer) 38400
414: rs232PortOutSpeed.3 (integer) 38400
415: rs232PortOutSpeed.4 (integer) 38400
416: rs232AsyncPortIndex.1 (integer) 1 [1]
417: rs232AsyncPortIndex.2 (integer) 2 [2]
418: rs232AsyncPortIndex.3 (integer) 3 [3]
419: rs232AsyncPortIndex.4 (integer) 4 [4]
420: rs232AsyncPortBits.1 (integer) 8
421: rs232AsyncPortBits.2 (integer) 8
422: rs232AsyncPortBits.3 (integer) 8
423: rs232AsyncPortBits.4 (integer) 8
424: rs232AsyncPortStopBits.1 (integer) one(1)
425: rs232AsyncPortStopBits.2 (integer) one(1)
426: rs232AsyncPortStopBits.3 (integer) one(1)
427: rs232AsyncPortStopBits.4 (integer) one(1)
428: rs232AsyncPortParity.1 (integer) none(1)
429: rs232AsyncPortParity.2 (integer) none(1)
430: rs232AsyncPortParity.3 (integer) none(1)
431: rs232AsyncPortParity.4 (integer) none(1)
432: rs232InSigPortIndex.1.2 (integer) 1 [1]
433: rs232InSigPortIndex.2.2 (integer) 2 [2]
434: rs232InSigPortIndex.3.2 (integer) 3 [3]
435: rs232InSigPortIndex.4.2 (integer) 4 [4]
436: rs232InSigPortIndex.1.3 (integer) 1 [1]
437: rs232InSigPortIndex.2.3 (integer) 2 [2]
438: rs232InSigPortIndex.3.3 (integer) 3 [3]
439: rs232InSigPortIndex.4.3 (integer) 4 [4]
440: rs232InSigPortIndex.1.6 (integer) 1 [1]
441: rs232InSigPortIndex.2.6 (integer) 2 [2]
442: rs232InSigPortIndex.3.6 (integer) 3 [3]
443: rs232InSigPortIndex.4.6 (integer) 4 [4]
444: rs232InSigName.1.2 (integer) cts(2)
445: rs232InSigName.2.2 (integer) cts(2)
446: rs232InSigName.3.2 (integer) cts(2)
447: rs232InSigName.4.2 (integer) cts(2)
448: rs232InSigName.1.3 (integer) dsr(3)
449: rs232InSigName.2.3 (integer) dsr(3)
450: rs232InSigName.3.3 (integer) dsr(3)
451: rs232InSigName.4.3 (integer) dsr(3)
452: rs232InSigName.1.6 (integer) dcd(6)
453: rs232InSigName.2.6 (integer) dcd(6)
454: rs232InSigName.3.6 (integer) dcd(6)
455: rs232InSigName.4.6 (integer) dcd(6)
456: rs232InSigState.1.2 (integer) off(3)
457: rs232InSigState.2.2 (integer) off(3)
458: rs232InSigState.3.2 (integer) off(3)
459: rs232InSigState.4.2 (integer) off(3)

460: rs232InSigState.1.3 (integer) off(3)
461: rs232InSigState.2.3 (integer) off(3)
462: rs232InSigState.3.3 (integer) off(3)
463: rs232InSigState.4.3 (integer) off(3)
464: rs232InSigState.1.6 (integer) off(3)
465: rs232InSigState.2.6 (integer) off(3)
466: rs232InSigState.3.6 (integer) off(3)
467: rs232InSigState.4.6 (integer) off(3)
468: rs232OutSigPortIndex.1.1 (integer) 1 [1]
469: rs232OutSigPortIndex.2.1 (integer) 2 [2]
470: rs232OutSigPortIndex.3.1 (integer) 3 [3]
471: rs232OutSigPortIndex.4.1 (integer) 4 [4]
472: rs232OutSigPortIndex.1.4 (integer) 1 [1]
473: rs232OutSigPortIndex.2.4 (integer) 2 [2]
474: rs232OutSigPortIndex.3.4 (integer) 3 [3]
475: rs232OutSigPortIndex.4.4 (integer) 4 [4]
476: rs232OutSigName.1.1 (integer) rts(1)
477: rs232OutSigName.2.1 (integer) rts(1)
478: rs232OutSigName.3.1 (integer) rts(1)
479: rs232OutSigName.4.1 (integer) rts(1)
480: rs232OutSigName.1.4 (integer) dtr(4)
481: rs232OutSigName.2.4 (integer) dtr(4)
482: rs232OutSigName.3.4 (integer) dtr(4)
483: rs232OutSigName.4.4 (integer) dtr(4)
484: rs232OutSigState.1.1 (integer) off(3)
485: rs232OutSigState.2.1 (integer) off(3)
486: rs232OutSigState.3.1 (integer) off(3)
487: rs232OutSigState.4.1 (integer) off(3)
488: rs232OutSigState.1.4 (integer) off(3)
489: rs232OutSigState.2.4 (integer) off(3)
490: rs232OutSigState.3.4 (integer) off(3)
491: rs232OutSigState.4.4 (integer) off(3)
492: snmpInPkts.0 (counter) 493
493: snmpOutPkts.0 (counter) 493
494: snmpInBadVersions.0 (counter) 0
495: snmpInBadCommunityNames.0 (counter) 0
496: snmpInBadCommunityUses.0 (counter) 0
497: snmpInASNParseErrs.0 (counter) 0
498: snmpInTooBigs.0 (counter) 0
499: snmpInNoSuchNames.0 (counter) 0
500: snmpInBadValues.0 (counter) 0
501: snmpInReadOnlys.0 (counter) 0
502: snmpInGenErrs.0 (counter) 0
503: snmpInTotalReqVars.0 (counter) 503
504: snmpInTotalSetVars.0 (counter) 0
505: snmpInGetRequests.0 (counter) 0
506: snmpInGetNexts.0 (counter) 506
507: snmpInSetRequests.0 (counter) 0
508: snmpInGetResponses.0 (counter) 0
509: snmpInTraps.0 (counter) 0
510: snmpOutTooBigs.0 (counter) 0
511: snmpOutNoSuchNames.0 (counter) 0
512: snmpOutBadValues.0 (counter) 0
513: snmpOutGenErrs.0 (counter) 0
514: snmpOutGetRequests.0 (counter) 0
515: snmpOutGetNexts.0 (counter) 0
516: snmpOutSetRequests.0 (counter) 0
517: snmpOutGetResponses.0 (counter) 517
518: snmpOutTraps.0 (counter) 0
519: snmpEnableAuthenTraps.0 (integer) disabled(2)
***** SNMP QUERY FINISHED *****

NOTE Click on the following links for more information about MIB II and RS-232 like groups:
http://www.faqs.org/rfcs/rfc1213.html
http://www.faqs.org/rfcs/rfc1317.html

→ IA240/241 does NOT support SNMP trap.

OpenVPN

OpenVPN provides two types of tunnels for users to implement VPNS: **Routed IP Tunnels** and **Bridged Ethernet Tunnels**. To begin with, check to make sure that the system has a virtual device **/dev/net/tun**. If not, issue the following command:

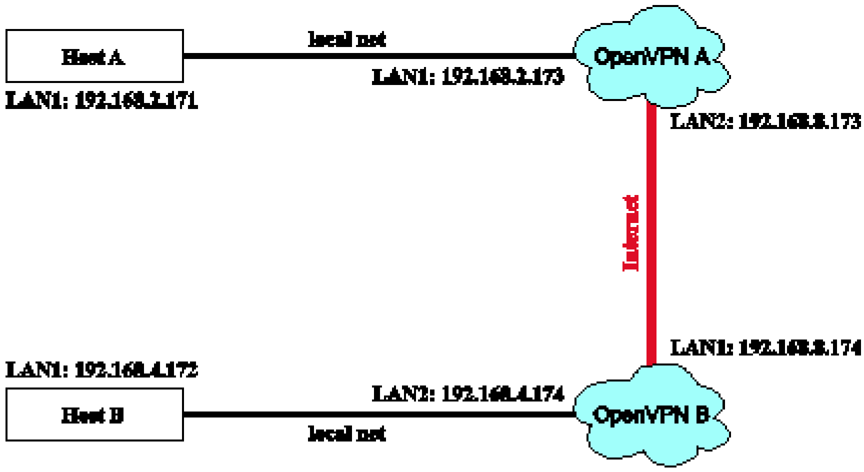
```
# mknod /dev/net/tun c 10 200
```

An Ethernet bridge is used to connect different Ethernet networks together. The Ethernets are bundled into one bigger, "logical" Ethernet. Each Ethernet corresponds to one physical interface (or port) that is connected to the bridge.

On each OpenVPN machine, you should generate a working directory, such as **/etc/openvpn**, where script files and key files reside. Once established, all operations will be performed in that directory.

Setup 1: Ethernet Bridging for Private Networks on Different Subnets

1. Set up four machines, as shown in the following diagram.



Host A (B) represents one of the machines that belongs to OpenVPN A (B). The two remote subnets are configured for a different range of IP addresses. When this setup is moved to a public network, the external interfaces of the OpenVPN machines should be configured for static IPs, or connect to another device (such as a firewall or DSL box) first.

```
# openvpn --genkey --secret secrouter.key
```

Copy the file that is generated to the OpenVPN machine.

2. Generate a script file named **openvpn-bridge** on each OpenVPN machine. This script reconfigures interface "eth1" as IP-less, creates logical bridge(s) and TAP interfaces, loads

modules, enables IP forwarding, etc.

```
#--------------------------------Start----------------------------

#!/bin/sh

iface=eth1  # defines the internal interface
maxtap=`expr 1`# defines the number of tap devices. I.e., # of tunnels

IPADDR=
NETMASK=
BROADCAST=

# it is not a great idea but this system doesn't support
# /etc/sysconfig/network-scripts/ifcfg-eth1
ifcfg_vpn()
{
 while read f1 f2 f3 f4 r3
 do
  if [ "$f1" = "iface" -a "$f2" = "$iface" -a "$f3" = "inet" -a "$f4" = "static" ];then
   i=`expr 0`
   while :
   do
    if [ $i -gt 5 ]; then
     break
    fi
    i=`expr $i + 1`
    read f1 f2
    case "$f1" in
     address ) IPADDR=$f2
      ;;
     netmask ) NETMASK=$f2
      ;;
     broadcast ) BROADCAST=$f2
      ;;
    esac
   done
      break
  fi
 done < /etc/network/interfaces
}

# get the ip address of the specified interface
mname=
module_up()
{
 oIFS=$IFS
 IFS='
 '
 FOUND="no"
 for LINE in `lsmod`
 do
  TOK=`echo $LINE | cut -d' ' -f1`
  if [ "$TOK" = "$mname" ]; then
   FOUND="yes";
   break;
  fi
 done
 IFS=$oIFS

 if [ "$FOUND" = "no" ]; then
  modprobe $mname
 fi
}

start()
{
 ifcfg_vpn
```

```
if [ ! \( -d "/dev/net" \) ]; then
 mkdir /dev/net
fi

if [ ! \( -r "/dev/net/tun" \) ]; then
 # create a device file if there is none
 mknod /dev/net/tun c 10 200
fi

# load modules "tun" and "bridge"
mname=tun
module_up
mname=bridge
module_up
# create an ethernet bridge to connect tap devices, internal interface
brctl addbr br0
brctl addif br0 $iface
# the bridge receives data from any port and forwards it to other ports.

i=`expr 0`
while :
do
 # generate a tap0 interface on tun
 openvpn --mktun --dev tap${i}

 # connect tap device to the bridge
 brctl addif br0 tap${i}

 # null ip address of tap device
 ifconfig tap${i} 0.0.0.0 promisc up

 i=`expr $i + 1`
 if [ $i -ge $maxtap ]; then
  break
 fi
done

# null ip address of internal interface
ifconfig $iface 0.0.0.0 promisc up

# enable bridge ip
ifconfig br0 $IPADDR netmask $NETMASK broadcast $BROADCAST

ipf=/proc/sys/net/ipv4/ip_forward
# enable IP forwarding
echo 1 > $ipf
echo "ip forwarding enabled to"
cat $ipf
}

stop() {
echo "shutdown openvpn bridge."
ifcfg_vpn
i=`expr 0`
while :
do
 # disconnect tap device from the bridge
 brctl delif br0 tap${i}
 openvpn --rmtun --dev tap${i}

 i=`expr $i + 1`
 if [ $i -ge $maxtap ]; then
  break
 fi
done
brctl delif br0 $iface
brctl delbr br0
ifconfig br0 down
```

```
 ifconfig $iface $IPADDR netmask $NETMASK broadcast $BROADCAST
 killall -TERM openvpn
}

case "$1" in
 start)
  start
  ;;
 stop)
  stop
  ;;
 restart)
  stop
  start
  ;;
 *)
  echo "Usage: $0 [start|stop|restart]"
  exit 1
esac
exit 0
#---------------------------------- end -----------------------------
```

Create link symbols to enable this script at boot time:

```
# ln -s /etc/openvpn/openvpn-bridge /etc/rc.d/rc3.d/S32vpn-br # for example
# ln -s /etc/openvpn/openvpn-bridge /etc/rc.d/rc6.d/K32vpn-br # for example
```

3. Create a configuration file named **A-tap0-br.conf** and an executable script file named **A-tap0-br.sh** on OpenVPN A.

```
# point to the peer
remote 192.168.8.174
dev tap0
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
up /etc/openvpn/A-tap0-br.sh
```

```
#----------------------------------Start------------------------------
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.4.0 netmask 255.255.255.0 dev br0
#---------------------------------- end ------------------------------
```

Create a configuration file named **B-tap0-br.conf** and an executable script file named **B-tap0-br.sh** on OpenVPN B.

```
# point to the peer
remote 192.168.8.173
dev tap0
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
up /etc/openvpn/B-tap0-br.sh
```

```
#---------------------------------- Start----------------------------
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.2.0 netmask 255.255.255.0 dev br0
#---------------------------------- end -----------------------------
```

Note: Select cipher and authentication algorithms by specifying "cipher" and "auth". To see with algorithms are available, type:

```
# openvpn --show-ciphers
# openvpn --show—auths
```

4. Start both of OpenVPN peers,

```
# openvpn --config A-tap0-br.conf&
# openvpn --config B-tap0-br.conf&
```

If you see the line "Peer Connection Initiated with 192.168.8.173:5000" on each machine, the connection between OpenVPN machines has been established successfully on UDP port 5000.

5. On each OpenVPN machine, check the routing table by typing the command:

```
# route
```

Destination	Gateway	Genmsk	Flags	Metric	Ref	Use	Iface
192.168.4.0	*	255.255.255.0	U	0	0	0	br0
192.168.2.0	*	255.255.255.0	U	0	0	0	br0
192.168.8.0	*	255.255.255.0	U	0	0	0	eth0

Interface **eth1** is connected to the bridging interface **br0**, to which device **tap0** also connects, whereas the virtual device **tun** sits on top of **tap0**. This ensures that all traffic from internal networks connected to interface **eth1** that come to this bridge write to the TAP/TUN device that the OpenVPN program monitors. Once the OpenVPN program detects traffic on the virtual device, it sends the traffic to its peer.

6. To create an indirect connection to Host B from Host A, you need to add the following routing item:

```
route add –net 192.168.4.0 netmask 255.255.255.0 dev eth0
```

To create an indirect connection to Host A from Host B, you need to add the following routing item:

```
route add –net 192.168.2.0 netmask 255.255.255.0 dev eth0
```

Now ping Host B from Host A by typing:

```
ping 192.168.4.174
```

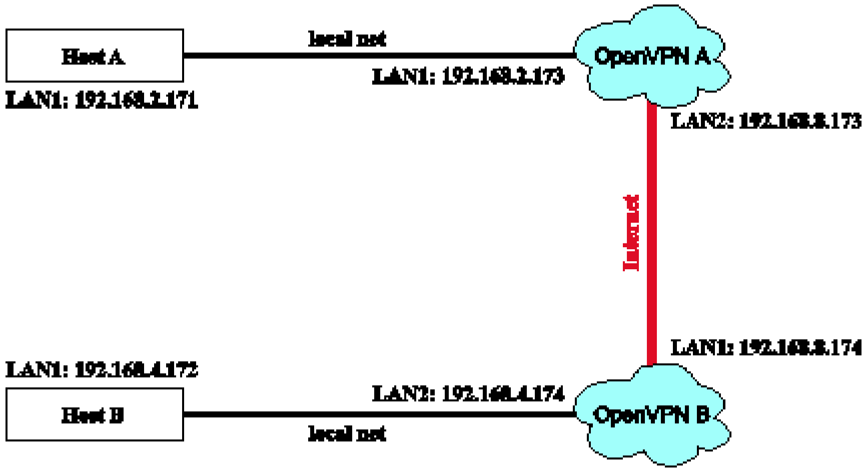
A successful ping indicates that you have created a VPN system that only allows authorized users from one internal network to access users at the remote site. For this system, all data is transmitted by UDP packets on port 5000 between OpenVPN peers.

7. To shut down OpenVPN programs, type the command:

```
# killall -TERM openvpn
```

Setup 2: Ethernet Bridging for Private Networks on the Same Subnet

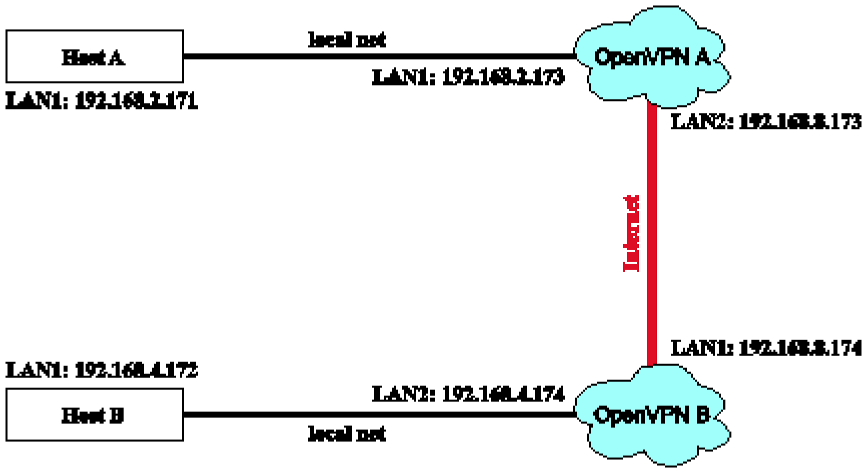
1. Set up four machines as shown in the following diagram:



2. The configuration procedure is almost the same as for the previous example. The only difference is that you will need to comment out the parameter "up" in "/etc/openvpn/A-tap0-br.conf" and "/etc/openvpn/B-tap0-br.conf".

Setup 3: Routed IP

1. Set up four machines as shown in the following diagram:



2. Create a configuration file named "A-tun.conf" and an executable script file named "A-tun.sh".

```
# point to the peer
remote 192.168.8.174
dev tun
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
ifconfig 192.168.2.173 192.168.4.174
up /etc/openvpn/A-tun.sh
```

```
#-------------------------------- Start-----------------------------
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.4.0 netmask 255.255.255.0 gw $5
#-------------------------------- end ------------------------------
```

Create a configuration file named **B-tun.conf** and an executable script file named **B-tun.sh** on OpenVPN B:

```
remote 192.168.8.173
dev tun
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
ifconfig 192.168.4.174 192.168.2.173
up /etc/openvpn/B-tun.sh
```

```
#-------------------------------- Start----------------------------
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.2.0 netmask 255.255.255.0 gw $5
#-------------------------------- end -----------------------------
```

Note that the parameter "ifconfig" defines the first argument as the local internal interface and the second argument as the internal interface at the remote peer.

Note that **$5** is the argument that the OpenVPN program passes to the script file. Its value is the second argument of **ifconfig** in the configuration file.

3. Check the routing table after you run the OpenVPN programs, by typing the command:

```
# route
```

Destination	Gateway	Genmsk	Flags	Metric	Ref	Use	Iface
192.168.4.174	*	255.255.255.255	UH	0	0	0	tun0
192.168.4.0	192.168.4.174	255.255.255.0	UG	0	0	0	tun0
192.168.2.0	*	255.255.255.0	U	0	0	0	eth1
192.168.8.0	*	255.255.255.0	U	0	0	0	eth0

5

Development Tool Chains

This chapter describes how to install a tool chain in the host computer that you use to develop your applications. In addition, the process of performing cross-platform development and debugging are also introduced. For clarity, the IA240/241 embedded computer is called a target computer.

The following functions are covered in this chapter:

- **Linux Tool Chain**
  - Steps for Installing the Linux Tool Chain
  - Compilation for Applications
  - On-Line Debugging with GDB

Linux Tool Chain

The Linux tool chain contains a suite of cross compilers and other tools, as well as the libraries and header files that are necessary to compile your applications. These tool chain components must be installed in your host computer (PC) running Linux. We have confirmed that the following Linux distributions can be used to install the tool chain.

Fefora core 1 & 2.

Steps for Installing the Linux Tool Chain

The tool chain needs about 485 MB of hard disk space. To install it, follow the steps.

1. Insert the package CD into your PC and then issue the following commands:
```
#mount /dev/cdrom /mnt/cdrom
#sh /mnt/cdrom/tool-chain/linux/install.sh
```
2. Wait for the installation process to complete. This should take a few minutes.
3. Add the directory **/usr/local/arm-linux/bin** to your path. You can do this for the current login by issuing the following commands:
```
#export PATH="/usr/local/arm-linux/bin:$PATH"
```
Alternatively, you can add the same commands to **$HOME/.bash_profile** to make it effective for all login sessions.

Compilation for Applications

To compile a simple C application, use the cross compiler instead of the regular compiler:

```
#arm-linux-gcc  -o example -Wall -g -O2 example.c
#arm-linux-strip  -s example
#arm-linux-gcc  -ggdb -o example-debug example.c
```

Most of the cross compiler tools are the same as their native compiler counterparts, but with an additional prefix that specifies the target system. In the case of x86 environments, the prefix is **`i386-linux-`** and in the case of IA204/241 ARM boards, it is **`arm-linux-`**.

For example, the native C compiler is **`gcc`** and the cross C compiler for ARM in the IA240/241 is **`arm-linux-gcc.`**

The following cross compiler tools are provided:

ar	Manages archives (static libraries)
as	Assembler
c++, g++	C++ compiler
cpp	C preprocessor
gcc	C compiler
gdb	Debugger
ld	Linker
nm	Lists symbols from object files
objcopy	Copies and translates object files
objdump	Displays information about object files
ranlib	Generates indexes to archives (static libraries)
readelf	Displays information about ELF files

size	Lists object file section sizes
strings	Prints strings of printable characters from files (usually object files)
strip	Removes symbols and sections from object files (usually debugging information)

On-Line Debugging with GDB

The tool chain also provides an on-line debugging mechanism to help you develop your program. Before performing a debugging session, add the option **-ggdb** to compile the program. A debugging session runs on a client-server architecture on which the server **`gdbserver`** is installed int the targe computer and the client **ddd** is installed in the host computer. We'll asuumne that you have uploaded a program named **hello-debug** to the target computer and strat to debug the program.

1. Log on to the target computer and run the debugging server program.

   ```
   #gdbserver 192.168.4.142:2000 hello-debug
   Process hello-debug created; pid=38
   ```

   The debugging server listens for connections at network port 2000 from the network interface 192.168.4.142. The name of the program to be debugged follows these parameters. For a program requiring arguments, add the arguments behind the program name.

2. In the host computer, change the directory to where the program source resides.

   ```
   cd /my_work_directory/myfilesystem/testprograms
   ```

3. Execute the client program.

   ```
   #ddd --debugger arm-linux-gdb hello-debug &
   ```

4. Enter the following command at the GDB, DDD command prompt.

   ```
   Target remote 192.168.4.99:2000
   ```

   The command produces a line of output on the target console, similar to the following.

   ```
   Remote debugging using 192.168.4.99:2000
   ```

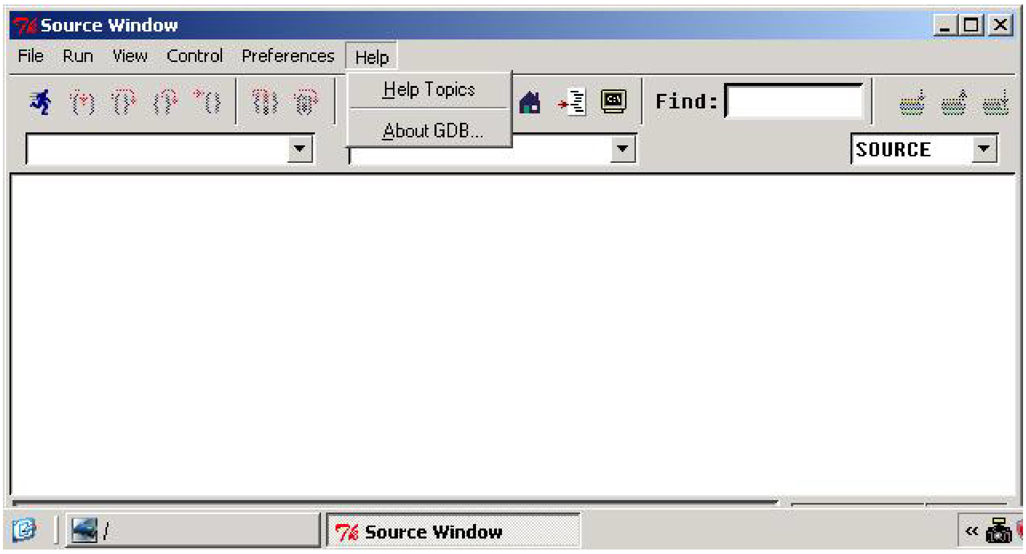
   192.168.4.99 is the machine's IP address, and 2000 is the port number. You can now begin debugging in the host environment using the interface provided by DDD.

5. Set a break point on main by double clicking, or by entering **`b main`** on the command line.
6. Click the **cont** button.

On-Line Debugging with Insight

Insight is a graphical user interface that accompanies **GDB**, the GNU Debugger was written in Tcl/Tk by people working at Red Hat, Inc., and Cygnus Solutions. Red Hat was generous enough to make Insight available for public use, and continues to maintain the program.

Click on http://sources.redhat.com/insight/ for more information about using Insight, or click on **Help Topics** under the **Help** menu to read the user manual.



6

Programmer's Guide

This chapter includes important information for programmers.

The following functions are covered in this chapter:

- **Flash Memory Map**
- **Device API**
- **RTC (Real Time Clock)**
- **Buzzer**
- **WDT (Watch Dog Timer)**
- **UART**
- **DI/DO**
- **Make File Example**

Flash Memory Map

Partition sizes are hard coded into the kernel binary. To change the partition sizes, you will need to rebuild the kernel. The flash memory map is shown in the following table.

Address	Size	Contents
0x00000000 – 0x0003FFFF	256 KB	Boot Loader—Read ONLY
0x00040000 – 0x001FFFFF	1.8 MB	Kernel object code—Read ONLY
0x00200000 – 0x009FFFFF	8 MB	Root file system (JFFS2) —Read ONLY
0x00A00000 – 0x00FFFFFF	6 MB	User directory (JFFS2) —Read/Write

Mount the user file system to **/mnt/usrdisk** with the root file system.

NOTE

1. The default Moxa file system only enables the network and CF. It lets users recover the user file system when it fails.
2. The user file system is a complete file system. Users can create and delete directories and files (including source code and executable files) as needed.
3. Users can create the user file system on the PC host or target platform, and then copy it to the IA240/241.

Device API

The IA240/241 supports control devices with the **ioctl** system API. You will need to **`include <moxadevice.h>`**, and use the following **ioctl** function.

```
int ioctl(int d, int request,…);
 Input: int d - open device node return file handle
      int request – argument in or out
```

Use the desktop Linux's man page for detailed documentation:

```
#man ioctl
```

RTC (Real Time Clock)

The device node is located at **/dev/rtc**. The IA240/241 supports Linux standard simple RTC control. You must **`include <linux/rtc.h>`**.

1. Function: RTC_RD_TIME

   ```
   int ioctl(fd, RTC_RD_TIME, struct rtc_time *time);
   ```

   Description: read time information from RTC. It will return the value on argument 3.

2. Function: RTC_SET_TIME

   ```
   int ioctl(fd, RTC_SET_TIME, struct rtc_time *time);
   ```

   Description: set RTC time. Argument 3 will be passed to RTC.

Buzzer

The device node is located at **/dev/console**. The IA240/241 supports Linux standard buzzer control, with The IA240/241's buzzer running at a fixed frequency of 100 Hz. You must **`include <sys/kd.h>`**.

Function: KDMKTONE

```
ioctl(fd, KDMKTONE, unsigned int arg);
```

Description: The buzzer's behavior is determined by the argument **`arg`**. The "high word" part of arg gives the length of time the buzzer will sound, and the "low word" part gives the frequency.

The buzzer's on / off behavior is controlled by software. If you call the "ioctl" function, you MUST set the frequency at 100 Hz. If you use a different frequency, the system could crash.

WDT (Watch Dog Timer)

1. **Introduction**

The WDT works like a watch dog function. You can enable it or disable it. When the user enables WDT but the application does not acknowledge it, the system will reboot. You can set the ack time from a minimum of 50 msec to a maximum of 60 seconds.

2. **How the WDT works**

The sWatchDog is disabled when the system boots up. The user application can also enable ack. When the user does not ack, it will let the system reboot.

Kernel boot
…..
….
User application running and enable user ack
….
….

3. **The user API**

The user application must **`include <moxadevic.h>`**, and **`link moxalib.a`**. A makefile example is shown below:

```
all:
  arm-linux-gcc –o xxxx  xxxx.c -lmoxalib
```

```
int swtd_open(void)
```

Description
Open the file handle to control the sWatchDog. If you want to do something you must first to this. And keep the file handle to do other.

Input
None

Output
The return value is file handle. If has some error, it will return < 0 value.

You can get error from errno().

```
int swtd_enable(int fd, unsigned long time)
```

Description
Enable application sWatchDog. And you must do ack after this process.

Input
int fd - the file handle, from the swtd_open() return value.

unsigned long time - The time you wish to ack sWatchDog periodically. You must ack the sWatchDog before timeout. If you do not ack, the system will be reboot automatically. The minimal time is 50 msec, the maximum time is 60 seconds. The time unit is msec.

Output
OK will be zero. The other has some error, to get the error code from errno().

```
int swtd_disable(int fd)
```

Description
Disable the application to ack sWatchDog. And the kernel will be auto ack it. User does not to do it at periodic.

Input
int fd - the file handle from swtd_open() return value.

Output
OK will be zero. The other has some error, to get error code from errno.

```
int swtd_get(int fd, int *mode, unsigned long *time)
```

Description
Get current setting values.

mode –
1 for user application enable sWatchDog: need to do ack.
0 for user application disable sWatchdog: does not need to do ack.

time – The time period to ack sWatchDog.

Input :
int fd - the file handle from swtd_open() return value.
int *mode - the function will be return the status enable or disable user application need to do ack.
unsigned long *time – the function will return the current time period.

Output:
OK will be zero.

The other has some error, to get error code from errno().

```
int swtd_ack(int fd)
```

Description
Acknowledge sWatchDog. When the user application enable sWatchDog. It need to call this function periodically with user predefined time in the application program.

Input
int fd - the file handle from swtd_open() return value.

Output
OK will be zero.

The other has some error, to get error code from errno().

```
int swtd_close(int fd)
```

Description
Close the file handle.

Input
int fd - the file handle from swtd_open() return value.

Output
OK will be zero.

The other has some error, to get error code from errno().

4. **Special Note**

When you "kill the application with -9" or "kill without option" or "Ctrl+c" the kernel will change to auto ack the sWatchDog.

When your application enables the sWatchDog and does not ack, your application may have a logical error, or your application has made a core dump. The kernel will not change to auto ack. This can cause a serious problem, causing your system to reboot again and again.

5. **User application example**

Example 1:

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

int main(int argc, char *argv[])
{
  int fd;

  fd = swtd_open();
  if ( fd < 0 ) {
   printf("Open sWatchDog device fail !\n");
   exit(1);
  }
  swtd_enable(fd, 5000);// enable it and set it 5 seconds
  while ( 1 ) {
   // do user application want to do
   …..
   ….
   swtd_ack(fd);
   …..
   ….
  }
  swtd_close(fd);
  exit(0);
}
```

The makefile is shown below:

```
all:
  arm-linux-gcc –o xxxx xxxx.c –lmoxalib
```

Example 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <moxadevice.h>
```

```
static void mydelay(unsigned long msec)
{
 struct timeval time;

 time.tv_sec = msec / 1000;
 time.tv_usec = (msec % 1000) * 1000;
 select(1, NULL, NULL, NULL, &time);
}

static int swtdfd;
static int stopflag=0;

static void stop_swatchdog()
{
 stopflag = 1;
}

static void do_swatchdog(void)
{
 swtd_enable(swtdfd, 500);
 while ( stopflag == 0 ) {
  mydelay(250);
  swtd_ack(swtdfd);
 }
 swtd_disable(swtdfd);
}

int main(int argc, char *argv[])
{
 pid_t  sonpid;

 signal(SIGUSR1, stop_swatchdog);
 swtdfd = swtd_open();
 if ( swtdfd < 0 ) {
  printf("Open sWatchDog device fail !\n");
  exit(1);
 }
 if ( (sonpid=fork()) == 0 )
  do_swatchdog();
 // do user application main function
 …..
 …..
 …..
 // end user application
 kill(sonpid, SIGUSR1);
 swtd_close(swtdfd);
 exit(1);
}
```

The makefile is shown below:

```
all:
 arm-linux-gcc –o xxxx xxxx.c –lmoxalib
```

UART

The normal tty device node is located at **/dev/ttyM0 … ttyM3**.

The IA240/241 supports Linux standard termios control. The Moxa UART Device API allows you to configure ttyM0 to ttyM3 as RS-232, RS-422, 4-wire RS-485, or 2-wire RS-485. IA240/241 supports RS-232, RS-422, 2-wire RS-485, and 4-wire RS485.

You must **include <moxadevice.h>**.

```
#define RS232_MODE    0
#define RS485_2WIRE_MODE  1
#define RS422_MODE    2
#define RS485_4WIRE_MODE  3
```

1. Function: Moxa_SET_OP_MODE

   ```
   int ioctl(fd, MOXA_SET_OP_MODE, &mode)
   ```

   Description
   Set the interface mode. Argument 3 mode will pass to the UART device driver and change it.

2. Function: MOXA_GET_OP_MODE

   ```
   int ioctl(fd, MOXA_GET_OP_MODE, &mode)
   ```

   Description
   Get the interface mode. Argument 3 mode will return the interface mode.

There are two Moxa private ioctl commands for setting up special baudrates.

Function: MOXA_SET_SPECIAL_BAUD_RATE
Function: MOXA_GET_SPECIAL_BAUD_RATE

If you use this ioctl to set a special baudrate, the termios cflag will be B4000000, in which case the B4000000 define will be different. If the baudrate you get from termios (or from calling tcgetattr()) is B4000000, you must call ioctl with MOXA_GET_SPECIAL_BAUD_RATE to get the actual baudrate.

Example to set the baudrate

```
#include <moxadevice.h>
#include <termios.h>
struct termios term;
int   fd, speed;
fd = open("/dev/ttyM0", O_RDWR);
tcgetattr(fd, &term);
term.c_cflag &= ~(CBAUD | CBAUDEX);
term.c_cflag |= B4000000;
tcsetattr(fd, TCSANOW, &term);
speed = 500000;
ioctl(fd, MOXA_SET_SPECIAL_BAUD_RATE, &speed);
```

Example to get the baudrate

```
 #include <moxadevice.h>
 #include <termios.h>
 struct termios term;
 int   fd, speed;
 fd = open("/dev/ttyM0", O_RDWR);
 tcgetattr(fd, &term);
if ( (term.c_cflag & (CBAUD|CBAUDEX)) != B4000000 ) {
  // follow the standard termios baud rate define
} else {
  ioctl(fd, MOXA_GET_SPECIAL_BAUD_RATE, &speed);
}
```

Baudrate inaccuracy

Divisor = 921600/Target Baud Rate. (Only Integer part)
ENUM = 8 * (921600/Target - Divisor) (Round up or down)
Inaccuracy = ((Target Baud Rate – 921600/(Divisor + (ENUM/8))) / Target Baud Rate)* 100%
E.g.,
To calculate 500000 bps
Divisor = 1, ENUM = 7,
Inaccuracy = 1.7%
*The Inaccuracy should less than 2% for work reliably.

Special Note

1. If the target baudrate is not a special baudrate (e.g. 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600), the termios cflag will be set to the same flag.
2. If you use stty to get the serial information, you will get speed equal to 0.

DI/DO

```
int set_dout_state(int doport, int state)
```

Description
Set the digital output state to high or low.

Input
int doport - the digital output port number. It is 0 to 3.
int state - the output state, high or low. You can use define DIO_HIGH or DIO_LOW.

Output
OK will be zero.

```
int get_din_state(int doport, int *state)
```

Description
Get the digital input current state at now.

Input
int diport - the digital input port number. It is 0 to 3.
int *state - To save the digital input state at now.

Output
OK will be zero.

```
int get_dout_state(int doport, int *state)
```

Description
Get the digital output current state at now.

Input
int doport - the digital output port number. It is 0 to 3.
int *state - To save the digital output state at now.

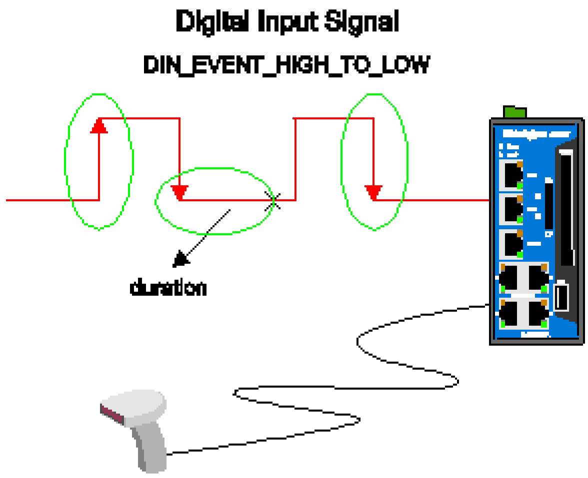
Output
OK will be zero.

```
int set_din_event(int diport, void (*func)(int diport), int mode, long int duration)
```

Description
Set the callback function for digital input port when the state is changed from high to low, low to high or any sate changed.

Input
int diport - the digital output port number. It is 0 to 3.
void (*func)(int diport) - The call back function point. It will be called when the set event happens.
int mode - Set the kind event. High to low, low to high or both.
long int duration - We know the digital signal sometime is not reliable. You can the duration time to except the error signal. If you set to zero, it will not detect the duration time. You can set 40 ms to 3600000 ms by increase 20 ms.

Output
OK will be zero.




```
int get_din_event(int diport, int *mode, long int *duration)
```

Description
Get the set event for digital input port.

Input
int diport - the digital output port number. It is 0 to 3.
int *mode - Save the set event.
long int *duration - Save the set duration time value.

Output
OK will be zero.

Special Note

Don't forget to link the library **libmoxalib & libpthread** for DI/DO programming, and also include the header file **moxadevice.h**. The DI/DO library only can be used by one program at a time.

Example

Example 1

File Name: tdio.c

Description: The program indicates to connect DO1 to DI1, change the digital output state to high or low by manual input, then the detect and count the state changed events from DI1.(OK)

```
#include <stdio.h>
#include <stdlib.h>
#include <moxadevice.h>
#include <fcntl.h>

#ifdef DEBUG
#define dbg_printf(x...) printf(x)
#else
#define dbg_printf(x...)
#endif

#define MIN_DURATION 40

static char *DataString[2]={"Low  ", "High "};
static void hightolowevent(int diport)
{
 printf("\nDIN port %d high to low.\n", diport);
}

static void lowtohighevent(int diport)
{
 printf("\nDIN port %d low to high.\n", diport);
}

int main(int argc, char * argv[])
{
 int i, j, state, retval;
 unsigned long duration;

 while( 1 ) {
  printf("\nSelect a number of menu, other key to exit.\n\
 1. set high to low event  \n\
 2. get now data.       \n\
 3. set low to high event  \n\
 4. clear event         \n\
 5. set high data.      \n\
 6. set low data.       \n\
 7. quit            \n\
 8. show event and duration  \n\
Choose : ");
 retval =0;
  scanf("%d", &i);
  if ( i == 1 ) { // set high to low event
   printf("Please keyin the DIN number : ");
   scanf("%d", &i);
   printf("Please input the DIN duration, this minimun value must be over %d :
",MIN_DURATION);
   scanf("%lu", &duration);
   retval=set_din_event(i, hightolowevent, DIN_EVENT_HIGH_TO_LOW, duration);
  } else if ( i == 2 ) {// get now data
   printf("DIN data  : ");
   for ( j=0; j<MAX_DIN_PORT; j++ ) {
    get_din_state(j, &state);
    printf("%s", DataString[state]);
   }
   printf("\n");
   printf("DOUT data : ");
   for ( j=0; j<MAX_DOUT_PORT; j++ ) {
    get_dout_state(j, &state);
    printf("%s", DataString[state]);
   }
   printf("\n");
```

```
  } else if ( i == 3 ) {// set low to high event
   printf("Please keyin the DIN number : ");
   scanf("%d", &i);
   printf("Please input the DIN duration, this minimun value must be over %d :
",MIN_DURATION);
   scanf("%lu", &duration);
   retval = set_din_event(i, lowtohighevent, DIN_EVENT_LOW_TO_HIGH, duration);
  } else if ( i == 4 ) {// clear event
   printf("Please keyin the DIN number : ");
   scanf("%d", &i);
   retval=set_din_event(i, NULL, DIN_EVENT_CLEAR, 0);
  } else if ( i == 5 ) {// set high data
   printf("Please keyin the DOUT number : ");
   scanf("%d", &i);
   retval=set_dout_state(i, 1);
  } else if ( i == 6 ) {// set low data
   printf("Please keyin the DOUT number : ");
   scanf("%d", &i);
   retval=set_dout_state(i, 0);
  } else if ( i == 7 ) {// quit
   break;
  } else if ( i == 8 ) {// show event and duration
   printf("Event:\n");
   for ( j=0; j<MAX_DOUT_PORT; j++ ) {
    retval=get_din_event(j, &i, &duration);
    switch ( i ) {
    case DIN_EVENT_HIGH_TO_LOW :
     printf("(htl,%lu)", duration);
     break;
    case DIN_EVENT_LOW_TO_HIGH :
     printf("(lth,%lu)", duration);
     break;
    case DIN_EVENT_CLEAR :
     printf("(clr,%lu)", duration);
     break;
    default :
     printf("err " );
     break;
    }
   }
   printf("\n");
  } else {
   printf("Select error, please select again !\n");
  }

  switch(retval) {
    case DIO_ERROR_PORT:
     printf("DIO error port\n");
     break;
    case DIO_ERROR_MODE:
     printf("DIO error mode\n");
     break;
    case DIO_ERROR_CONTROL:
     printf("DIO error control\n");
     break;
    case DIO_ERROR_DURATION:
     printf("DIO error duratoin\n");
    case DIO_ERROR_DURATION_20MS:
     printf("DIO error! The duratoin is not a multiple of 20 ms\n");
     break;
  }
 }

 return 0;
}
```

Example 2

File Name: tduration.c

Description: The program indicates to connect DO1 to DI1 and program will change digital output state automatically at the fixed frequency, then detect event change of the digital input state is high or low in different duration. (OK)

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include  <pthread.h>
#include <moxadevice.h>

#ifdef DEBUG
#define dbg_printf(x...) printf(x)
#else
#define dbg_printf(x...)
#endif

#define DURATION_NUM 7
#define TEST_NUM 10

static int ndin_StateChangeDetected, ndout_StateChangeDetected;
static int nDuration;
static unsigned long duration[2][DURATION_NUM]={  { 50, 40, 35, 30, 25, 20, 15 }, { 160,
140, 120, 100, 80, 60, 40, } };

/*******************************************************************
  When the din state changed form high to low, this function will be invoked
*******************************************************************/
static void low2highevent(int diport)
{
 ndin_StateChangeDetected++;
 dbg_printf("din state changed:%d\n",ndin_StateChangeDetected);
}

/*******************************************************************
  This function is used to exchange the dout state periodically
*******************************************************************/
void dout_control(int signo)
{
 int state;

 get_dout_state(0, &state);
 dbg_printf("dout state changed:%d\n",state);
 if(state) // exchange the dout state periodically
 {
  ndout_StateChangeDetected++;
  set_dout_state(0, 0);
 }
 else
 {
  set_dout_state(0, 1);
 }
}

void dio_test_function(void )
{
 struct itimerval value;
 int j, i, nChoice;
 struct timeval tv;

 do {

  printf("0.Test for Din duration==0.\n");
  printf("1.Test for Din duration!=0.\n");
```

```
  printf("9.Quit.\n" );
  printf("Please select a choice>");
  scanf("%d",&nChoice);

  if( nChoice == 9 ){  // Quit
   break;
  }
  elseif( nChoice == 0 ){  //test for din duration==0

   for ( nDuration=0; nDuration < DURATION_NUM; nDuration++ ) {
    // configure the dout frequency. When the timer timeouts, dout_control() will be
called to change the dout state
    value.it_value.tv_sec = duration[0][nDuration]/1000;
    value.it_value.tv_usec = (duration[0][nDuration]%1000) *1000 ;
    value.it_interval = value.it_value;
    setitimer(ITIMER_REAL,&value,NULL);

    ndin_StateChangeDetected = 0;  // reset these counters
    ndout_StateChangeDetected = 0;

    printf("DI duration,:0, DO duration:%d\n",duration[0][nDuration]);

    set_din_event(0, low2highevent, DIN_EVENT_LOW_TO_HIGH, 0);

    while( ndin_StateChangeDetected < TEST_NUM ) {
     pause();
    }
    printf("ndin_StateChangeDetected:%d, ndout_StateChangeDetected:%d,\n",
ndin_StateChangeDetected, ndout_StateChangeDetected);
    printf("loss detection
probability:%f\%,\n",(ndout_StateChangeDetected-ndin_StateChangeDetected)*100.0/nd
out_StateChangeDetected);
   }
  }//end of if( nChoice ==0 )

  elseif( nChoice == 1 ) { //test for din duration!=0

   for ( nDuration=0; nDuration < DURATION_NUM; nDuration++ ) {
    // configure the dout frequence. when the timer timeout, dout_control() will be
call to change the dout state
    value.it_value.tv_sec = duration[1][nDuration]/1000;
    value.it_value.tv_usec = ( duration[1][nDuration]%1000 ) *1000 ;
    value.it_interval = value.it_value;
    setitimer(ITIMER_REAL,&value,NULL);

    // Test for: dout kept in the same frequency but din set for different duration
    for( i=0; i<DURATION_NUM; i++) {
     if( duration[1][i] <= duration[1][nDuration] ) {
      // reset these counters
      ndin_StateChangeDetected = 0;
      ndout_StateChangeDetected = 0;

      printf("DI duration,:%d, DO duration:%d\n", duration[1][i],
duration[1][nDuration] );

      set_din_event(0, low2highevent, DIN_EVENT_LOW_TO_HIGH, duration[1][i]);

      while( ndout_StateChangeDetected < TEST_NUM ) {
       pause();
      }
      printf("ndin_StateChangeDetected:%d, ndout_StateChangeDetected:%d,\n",
ndin_StateChangeDetected, ndout_StateChangeDetected);
      printf("loss detection
probability:%f\%,\n",(ndout_StateChangeDetected-ndin_StateChangeDetected)*100.0/nd
out_StateChangeDetected);
     }
    } //end of for( i=0; i<DURATION_NUM; i++)
   }
```

```
  }
 } while(1);

 pthread_exit(NULL);
}

void init_sigaction(void)
{
 struct sigaction act;
 act.sa_handler=dout_control;
 act.sa_flags=0;
 sigemptyset(&act.sa_mask);
 sigaction(SIGALRM,&act,NULL);
}

int main(int argc, char * argv[])
{
 pthread_t dio_test;

 init_sigaction();

 set_dout_state(0, 0);  // set the DOUT0 as high

 set_din_event(0, low2highevent, DIN_EVENT_LOW_TO_HIGH, duration[1][0]);

 dio_test_function();

 while( nDuration < DURATION_NUM )
  usleep(100000);

}
```

DIO Program Make File Example

```
FNAME=tdio
FNAME1=tduration
CC=arm-linux-gcc
STRIP=arm-linux-strip

release:
$(CC) -o $(FNAME) $(FNAME).c -lmoxalib -lpthread
$(CC) -o $(FNAME1) $(FNAME1).c -lmoxalib -lpthread
$(STRIP) -s $(FNAME)
$(STRIP) -s $(FNAME1)

debug:
$(CC) -DDEBUG -o $(FNAME)-dbg $(FNAME).cxx -lmoxalib -lpthread
$(CC) -DDEBUG -o $(FNAME1)-dbg $(FNAME1).cxx -lmoxalib -lpthread

clean:
/bin/rm -f $(FNAME) $(FNAME)-dbg $(FNAME1) $(FNAME1)-dbg *.o
```

Make File Example

The following Makefile file example codes are copied from the Hello example on the IA240/241's CD-ROM.

```
CC = /usr/local/arm-linux/bin/arm-linux-gcc
CPP = /usr/local/arm-linux/bin/arm-linux-gcc
SOURCES = hello.c

OBJS =  $(SOURCES:.c=.o)
```

```
all:hello

hello:$(OBJS)
 $(CC) -o $@ $^ $(LDFLAGS) $(LIBS)

clean:
 rm -f $(OBJS) hello core *.gdb
```

7

Software Lock

“Software Lock” is an innovative technology developed by the Moxa engineering force. It can be adopted by a system integrator or developer to protect his applications from being copied. An applicaion is compiled into a binary format bound to the embedded computer and the operating system (OS) that the application runs on. As long as one obtains it from the computer, he/she can install it into the same hardware and the same operating system. The add-on value created by the developer is thus lost.

Moxa engineering force has developed this protection mechanism for your applications via data encryption. The binary file associated with each of your applications needs to undergo an additional encryption process after you have developed it. The process requires you to install an encryption key in the target computer.

1. Choose an encryption key (e.g.,”ABigKey”) and install it in the target computer by a pre-utility program, ‘setkey’.

   #setkey ABigKey

   Note: set an empty string to clear the encryption key in the target computer by:

   #setkey ““

2. Develop and compile your program in the development PC.
3. In the development PC, run the utility program ‘binencryptor’ to encrypt your program with an encryption key.

   #binencryptor yourProgram ABigKey

4. Upload the encrypted program file to the target computerby FTP or NFS and test the program.

The encryption key is a computer-wise key. That is to say, a computer has only one key installed. Running the program ‘setkey’ multiple times causes the key to be overrided.

To prove the effectiveness of this software protection mechanism, prepare a target computer that has not been installed an encryption key or install a key different from that used to encrypt your program. In any case, the encrypted program fails immediately.

This mechanism also allows the computer with an encryption key to bypass programs that are not encrypted. Therefore, in the development phase, you can develop your programs and test them in the target computer cleanly.

A

System Commands

Linux normal command utility collection

File manager

1.	cp	copy file
2.	ls	list file
3.	ln	make symbolic link file
4.	mount	mount and check file system
5.	rm	delete file
6.	chmod	change file owner & group & user
7.	chown	change file owner
8.	chgrp	change file group
9.	sync	sync file system, let system file buffer be saved to hardware
10.	mv	move file
11.	pwd	display now file directly
12.	df	list now file system space
13.	mkdir	make new directory
14.	rmdir	delete directory

Editor

1.	vi	text editor
2.	cat	dump file context
3.	zcat	compress or expand files
4.	grep	search string on file
5.	cut	get string on file
6.	find	find file where are there
7.	more	dump file by one page
8.	test	test file exist or not
9.	sleep	sleep (seconds)
10.	echo	echo string

Network

1.	ping	ping to test network
2.	route	routing table manager
3.	netstat	display network status
4.	ifconfig	set network ip address
5.	tracerout	trace route
6.	tftp	
7.	telnet	
8.	ftp	

Process

1.	kill	kill process
2.	ps	display now running process

Other

1.	dmesg	dump kernel log message
2.	sty	to set serial port
3.	zcat	dump .gz file context
4.	mknod	make device node
5.	free	display system memory usage
6.	date	print or set the system date and time
7.	env	run a program in a modified environment
8.	clear	clear the terminal screen
9.	reboot	reboot / power off/on the server
10.	halt	halt the server
11.	du	estimate file space usage
12.	gzip, gunzip	compress or expand files
13.	hostname	show system's host name

Moxa special utilities

1.	kversion	show kernel version
2.	cat /etc/version	show user directory version
3.	upramdisk	mount ramdisk
4.	downramdisk	unmount ramdisk