Modem-HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Modem-HOWTO

David S.Lawyer mailto:dave@lafn.org

v0.39, January 2007

Help with selecting, connecting, configuring, trouble-shooting, and understanding analog modems for a PC.

1. Introduction

1.1 DSL, Cable, and ISDN Modems in other HOWTOs

This HOWTO covers conventional analog modems for PCs on the PCI, USB, LPC, and ISA buses. USB and ISDN coverage is weak. For other types of modems see:

- DSL-HOWTO
- ADSL-Bandwidth-Management-HOWTO
- Cable-Modems-HOWTO (same as Cable Modem Providers HOWTO)
- SuSE ISDN Howto (not a LDP Howto) http://brenner.chemietechnik.uni-dortmund.de/doc/sdb/en/html/isdn.html
- http://public.swbell.net/ISDN/overview.html tutorial on ISDN
- ISDN docs in the kernel documentation subdirectory: "isdn".
- http://www.isdn4linux.de
- Appendix D: Other Types of Modems

1.2 Also not well covered: PCMCIA Modems, PPP

For modems on the PCMCIA bus see the PCMCIA-HOWTO: PCMCIA serial and modem devices. This HOWTO also doesn't cover the details of PPP (used to connect to the Internet via a modem) or communication programs. If you want to use a modem to connect to the Internet then you need to use a program that will automatically set up PPP for you (such as wvdial). More documentation on ppp should be found in /usr/doc/ppp, /usr/share/doc/ppp or the like.

1.3 Copyright, Disclaimer, Trademarks, & Credits

Copyright

3. Give due credit to previous authors and major contributors.

If you're considering making a derived work other than a translation, it's requested that you discuss your plans with the current maintainer.

Disclaimer

While I haven't intentionally tried to mislead you, there are likely a number of errors in this document. Please let me know about them. Since this is free documentation, it should be obvious that I cannot be held legally responsible for any errors.

Trademarks.

Any brand names (starts with a capital letter such as MS Windows) should be assumed to be a trademark). Such trademarks belong to their respective owners.

"Hayes" is a trademark of Microcomputer Products Inc. I use "winmodem" to mean any modem which originally required MS-Windows and not in the trademark sense. All other trademarks belong to their respective owners.

Credits

The following is only a rough approximation of how this document was created in the year 2000: About 1/4 of the material here was lifted directly from Serial-HOWTO v. 1.11 (1997) by Greg Hankins. mailto:gregh@twoguys.org (with his permission). About another 1/4 was taken from that Serial-HOWTO and revised. The remaining 1/2 is newly created by the new author: David S. Lawyer mailto:dave@lafn.org. Since 2000 much more has been added by the current author so that little remains of the modem coverage in the old Serial-HOWTO.

1.4 Contacting the Author

Since I don't follow the many different brands/models of modems please don't email me with questions about them (or suggestions of which one to buy). If you are interested in a certain model (to find out if it works under Linux, etc.) see the huge list at Web Sites. Also, please don't ask me how to configure a modem unless you've looked over this HOWTO and still can't do it. I've no personal experience with software-based modems.

Please let me know of any errors in facts, opinions, logic, spelling, grammar, clarity, links, etc. But first, if the date is over a month or two old, check to see that you have the latest version. Please send me any other info that you think belongs in this document.

1.5 New Versions of this HOWTO

New versions of this Modem-HOWTO should come out every few months. Your problem might be solved in the latest version. It will be available to browse and/or download at LDP mirror sites. For a list of such sites see: http://www.tldp.org/mirrors.html If you only want to quickly compare the date of this the version v0.39, January 2007 with the date of the latest version go to: http://www.tldp.org/HOWTO/Modem-HOWTO.html

1.6 New in Recent Versions

For a full revision history going back to the first version see the source file (in linuxdoc format) at http://cvsview.tldp.org/index.cgi/LDP/howto/linuxdoc/Modem-HOWTO.sgml.

- v0.39 Jan. 2007 gromitkc url (modem list) seems to be no longer maintained. Redefined "antique modems" as all under 56k speed. setpci can't set IRQs. devfs is obsolete. vgetty supports ITU v.253
- v0.38 May 2005: Eliminated section on Digital Modems in appendix since it's already covered elsewhere. More on cable modems. ISDN serial modems. Troubleshooting: Can't find winmodems if no driver.
- v0.37 Feb. 2005: For AMR, codec is on motherboard. Fixed a few typos. Better clarity for Dial-In. "NO CARRIER" likely not displayed when remote hangs up.
- v0.36 Feb. 2005 Rewrote "Quick Install" oriented towards PCI. Some external RS-232 modems are winmodems. /dev/modem.

1.7 What is a Modem ?

A modem (or analog modem) is a device that lets one send digital signals over an ordinary telephone line not designed for digital signals. If telephone lines were all digital then you wouldn't need a modem. But sometimes, a substitute for an analog modem, connected to a digital phone line, is imprecisely called a "digital modem". A modem permits your computer to connect to and communicate with the rest of the world. When you use a modem, you normally use a communication program or web browser to utilize the modem and dial-out on a telephone line. Advanced modem users can set things up so that others may phone in to them and use the computer remotely. This is called "dial-in".

Oversimplified, there are four basic types of analog modems for a PC: external serial (RS-232), USB (= external USB), internal, and built-in. The external serial and USB set on your desk outside the PC while the other two types are not visible since they're inside the PC. The external serial modem plugs into a connector on the back of the PC known as a "serial port". The USB modem plugs into a USB cable. See USB Modems. The internal modem is a card that is inserted inside the computer. The built-in modem is a chip on the motherboard used primarily in laptops. What is said in this HOWTO regarding internal modems will generally apply also to built-in modems. Internal modems are further subdivided into PCI, ISA, and AMR, depending on whether they are designed for the PCI or ISA bus, or for an AMR slot.

For an external vs internal comparison see External vs. Internal. When you get an internal or built-in modem, you also get a dedicated serial port (which can only be used with the modem and not with anything else such as an external modem or console terminal). In Linux, the common serial ports are named ttyS0, ttyS1, etc. These ports usually corresponding respectively to COM1, COM2, etc. in Dos/Windows). But in special cases, the names are longer such as: ttySHCF0 is the 0th serial port for a type of winmodem (HCF = Host Controlled Family). New types of serial ports just add some more letters to ttyS.

See Modem & Serial Port Basics for more details on how modems and serial ports work. With a USB modem, the driver simulates a serial port at for example /dev/ttySHCFUSB.

Modems usually include the ability to send Faxes (Fax Modems). See Fax for a list of fax software. "Voice" modems can work like an automatic answering machine and handle voicemail. See Voicemail Software.

The v.92 protocol can put the modem "on hold" when someone makes an ordinary voice call to your telephone, provided that you have "call waiting" from your telephone company. Thus you can get a phone call while online. As of Jan. 2003 Linux doesn't seem to support it. If this is the latest version of this HOWTO, let

me know about any Linux support for it. Some linmodem drivers may support it (but what if you have a hardware modem that doesn't use any linmodem driver?).

1.8 Does My Computer Contain an Internal Modem ?

Internal modems usually have a pair of modular telephone jacks on the back of the computer. They should be right next to each other and each one looks like a jack on the interior wall of a building where a telephone plugs in. One of the pair should be labeled "line" (or the like) which is where you plug in the telephone line.

Network cards also have modular jacks, but they are seldom in pairs and are slightly wider since they normally have 8 pins. Internal DSL "modems" exist and also have modular telephone jacks, but I think they are not very common (most DSL modems are external) as of 2002.

1.9 Quick Install

Very Quick Install

If you think your modem will work under Linux and needs no special driver, then just physically install/connect it. Start you computer, watch the boot-time messages for Linux to find the modem. Note it's the serial port number such as ttyS2 (/dev/ttyS2). Connect a phone line to it and dial out with say wvdial (after configuring wvdial). If the above doesn't work, read on.

Will my modem work under Linux?

So called "winmodems" will work under Linux only if a driver for it exists and gets installed. In this case it's called a "linmodem" since it can be made to work under Linux. If it's made prior to 2004 see old modem list and Software-based Modems (winmodems). There's no point of installing a modem that will not work with Linux.

External Serial Modem Install

At one time (2002 ?) no external serial modem was a winmodem but that's no longer the case. With a straight-thru or modem cable, connect the modem to an unused serial port on the PC. Make sure you know the name of the serial port: in most cases COM1 is ttyS0, COM2 is ttyS1, etc. You may need to check the BIOS setup menu to determine this. Plug in the power cord to provide power to the modem. See All Modems for further instructions.

Internal Modems (ISA, PCI and AMR)

If the modem is both PnP and directly supported by the serial driver (kernel 2.4 +) or by a winmodem driver that you've installed, then there is no configuring for you to do since the driver should configure it.

To physically install a modem card, remove the cover of the PC by /removing some screws. Find a matching vacant slot for the card next to the other adapter cards. Before inserting the card in the slot, remove a small cover plate on the back of the PC so that the telephone jacks on the card will be accessible from the rear of the PC. Then carefully align the card with the slot and push the card all the way down into the slot. Attach the card with a mounting screw (usually 3mm, .5mm pitch --don't use the wrong size).

You may watch the boot-time messages to see if your modem is detected. Use "dmesg" to see them or shift-page-up to scroll the screen back after they have flashed by.

Internal Modems: Manual configuration

Normally, you don't need to do this manual configuration since the modem's serial port may be detected and assigned a port at boot-time. For example: ttyS14 at I/O 0x6450 (IRQ = 10). Otherwise (or if there is some special reason to change the configuration) then you need to configure it yourself (or perhaps update your kernel to increases the likelihood that the modem gets detected). If your modem has no ttyS number assigned to it, it can't be used until it gets a ttyS number (like ttyS10). It thus can't be detected by application programs such as dialers or minicom. But it might be found by using say "lspci -v" if it's on the PCI bus.

Finding a lost modem may not be easy and you may need to read a lot more of this HOWTO. Once found, you need to use the "setserial" program to manually assign it to an available ttyS? port of your choice . For this you need to know both it's IO address (such as 0x6450) and its IRQ (such as 10). In the worst case, the modem has been disabled by a failing to be detected and enabled by the BIOS (or Linux) and doesn't have any IO address nor IRQ number. But you may still be able to find it. Older modems could be disabled by a jumper on the card or in rare cases by MS software.

You may have some choice of IRQs and IO addresses (including the case where you are able to change what the BIOS has set). See Choosing Serial IRQs and Choosing Addresses.

Old ISA Modems

ISA modems normally use ttyS0 - ttyS3. For old modems with jumpers look at the modem manual or look for printing on the modem card that tells you what the jumpers do. They have standard IO addresses corresponding to the ttySx. For example you may find it feasible to use /dev/ttyS2 at IO address 0x3e8 and IRQ 11.

If it has no jumpers then it's likely a Plug-and-Play modem which the BIOS may configure when you power one your PC. Typing "pnpdump --dumpregs" should find it. If you need to set or change them use "isapnp". Use the "pnpdump" program to see what changes are possible.

Both PCI and ISA: Use setserial to tell the serial driver

You must find the file where "setserial" is run at boot-time and add a line something like: "setserial /dev/ttyS2 irq 5 port 0x0b8". For setserial v2.15 and later the results of running "setserial" on the command line may (or may not) be saved to file named serial.conf or autoserial.conf. It might be in say the /etc directory or in the /var/lib/setserial directory (use "locate to find it). it runs each time you boot. See What is Setserial for more info. See the next subsection All Modems for further instructions on quick installation.

Use MS Windows to set the BIOS (A last resort method)

If you are using the BIOS to configure you may attempt to use MS Windows9x to "force" the BIOS to set a certain IRQ and/or IO. It can set them into the PnP BIOS's flash memory where they will be used to configure for Linux as well as Windows. See "Plug-and-Play-HOWTO and search for "forced" (occurs in several places). For Windows3.x you can do the same thing using the ICU under Windows 3.x. A few modems have a way to disable PnP in the modem hardware using software (under Windows) that came with the modem.

All Modems

Plug the modem into a telephone line. Then configure a dialing program. If you have an Internet Service Provider (ISP) you might configure one of these : wvdial, pppconfig, gnome-ppp, modem lights (Gnome) or kppp. They not only dial out but start PPP, which you need to connect to the Internet. Otherwise, you might try configuring the minicom dialer which isn't designed for connecting to the Internet, but is good for testing. Whether it's minicom or a dialer that starts PPP, set the serial port speed to a baud rate a few times higher than the bit rate of your modem. See Speed Table for more details on the "best" speeds to use. Tell it the full name of your serial port such as /dev/ttyS1 (or /dev/ttys/1).

Minicom is one way to set up and test your modem. Set hardware flow control (RTS/CTS). With minicom you may check to see if your modem is there (and ready to dial). Once you've set up minicom, type the command: AT, hit enter and you should see an "OK" response which comes directly from the modem. See Dialing Out with Minicom.

1.10 dev/modem

If your modem is on say /dev/ttyS2, you may want to link that to /dev/modem. It's not really necessary to do this since you can write down (or remember) say ttyS2 and tell it to programs that use the modem. It may be simpler to just link it. To link it, type say `ln -s /dev/ttyS2 /dev/modem` . Note "ttyS2" is just for example. It might actually be ttyS14, etc. Or use Red Hat's modemtool (or the like) to link it. But once you link it, be sure that all programs that use the modem use /dev/modem and not /dev/ttyS2, otherwise two programs may try to use the modem at the same time without knowing they are doing this. System software was written around 2000 to fix this problem but it may not be in recent kernels (like 2.6).

2. Modems for a Linux PC

2.1 Many Winmodems Will Not Work with Linux

Unfortunately, some software modems (winmodems) will not work with Linux due to lack of Linux drivers. Configuring the software modems that can be made to work with Linux ranges from very easy (automatically) to difficult, depending on both the modem, your skills, and how easy it is to find info about your modem --info that is not all in this HOWTO. If you buy a new one that you're not sure will work under Linux, try to get an agreement that you can return it for a refund if it doesn't work out.

Even if your modem works with Linux it can't be used until the serial port it's located on is enabled and made known to the operating system. For a detailed explanation of this (or if boot-time messages don't show your modem's serial port) study this HOWTO or see Plug-and-Play-HOWTO.

2.2 External vs. Internal

A modem for a PC may be either internal, external serial, or external USB. The internal one is installed inside of your PC (you must remove screws, etc. to install it). An external one just plugs in to a cable: USB cable (USB modem) or to the serial port (RS-232 serial modem). As compared to external serial modems, the internal modems are less expensive, are less likely to suffer data loss due to buffer overrun, and usually use less electricity. An internal modem obviously doesn't use up any desk space.

External serial modems are usually easier to install and usually has less configuration problems provided the serial port you'll connect it to is configured OK. External USB modems are more likely to be winmodems and are reportedly usually more difficult to deal with than external serial modems. External modems have lights which may give you a clue as to what is happening and aid in troubleshooting. The fact that the serial port and modem can be physically separated also aids in troubleshooting. External modems are easy to move to another computer. If you need to turn the power off to reset your modem (this is seldom necessary) then with an external you don't have to power down the entire PC.

Unfortunately, most external serial modems have no switch to turn off the power supply when not in use and thus are likely to consume a little electricity even when turned off (unless you unplug the power supply from the wall). Each watt they draw usually costs you over $1/yr. Another possible disadvantage of an external is that you will be forced to use an existing serial port which may not support a speed of over 115,200 bps (although as of late 2000 most new internal modems don't either --but some do). For details Can't Set a High Enough Speed

2.3 Is a Driver Needed ?

Any modem, of course, needs the serial driver that comes with Linux (either built into the kernel or as a module). For PCI, this driver should also detect the modem but it's not really a modem driver since it just detects which serial port the modem is on.

But what about modem drivers? Any software modem (winmodem, linmodem) must have a modem driver (if it exists for Linux). Hardware modems don't really need any modem driver unless you want to use special features such as voice and "modem on hold".

Software modems require software to run them and obviously do need a driver. The drivers for MS Windows are *.exe programs which will not run under Linux. So you must use a Linux driver (if it exists). See Software-based Modems (winmodems, linmodems)

2.4 External Modems

Do they all work under Linux?

At one time (2002 ?) all external modems would work under Linux. But then came the controllerless external modem which wouldn't. If the box says it requires Windows with no mention of Linux it could mean just that. Could it be that Windows software is provided for "modem on hold" and for use as an answering machine, etc., but that otherwise it will work under Linux? Linux may not support these features very well if at all. If this is a recent version of Modem-HOWTO, let me know of your experience with this.

PnP External Modems

Many external modems are labeled "Plug and Play" (PnP). If they are hardware modems, they should all work as non-PnP modems. While the serial port itself may need to be configured (IRQ number and IO address) unless the default configuration is OK, an external modem uses no such IRQ/IO configuration. You just plug the modem into the serial port.

The PnP modem has a special PnP identification built into it that can be read (thru the serial port) by a PnP operating system. Such an operating system would then know that you have a modem on a certain port and would also know the id number. If it's a controllerless modem, it could try to locate a driver for it. It could

also tell application programs what port your modem is on (such as /dev/ttyS2 or COM3). But Linux may not be able to do this. Thus you may need to configure your application program manually by giving it the ttyS number (such as /dev/ttyS2). Some programs like wvdial can probe for a modem on various ports.

Cabling & Installation

Connecting an external modem is simple compared to connecting most other devices to a serial port that require various types of "null modem" cables (which will not work for modems). Modems use a straight through cable, with no pins crossed over. Most computer stores should have one. Make sure you get the correct gender and number of pins. Hook up your modem to one of your serial ports. If you are willing to accept the default IRQ and IO address of the port you connect it to, then you are ready to start your communication program and configure the modem itself.

What the Lights (LED's) Mean (for some external modems)

- TM Test Modem
- AA Auto Answer (If on, your modem will answer an incoming call)
- RD Receive Data line = RxD
- SD Send Data line = TxD
- TR data Terminal Ready = DTR (set by your PC)
- RI Ring Indicator (If on, someone is "ringing" your modem)
- OH Off Hook (If off, your modem has hung up the phone line)
- MR Modem Ready = DSR ??
- EC Error Correction
- DC Data Compression
- HS High Speed (for this modem)

2.5 Internal Modems

An internal modem is installed in a PC by taking off the cover of the PC and inserting the modem card into a vacant slot on the motherboard. There are modems for PCI slots, other modems for the older ISA slots, and ARM software "modems" for the new small AMR slot. Only some newer PCs will have ARM slots. While external modems plug into the serial port (via a short cable) the internal modems have the serial port built into the modem. In other words, the modem card is both a serial port and a modem.

Setting the IO address and IRQ for a serial port was formerly done by jumpers on the card. These are little black rectangular "cubes" about 5x4x2 mm in size which push in over pins on the card. Plug-and-Play modems (actually the serial port part of the modems) don't use jumpers for setting these but instead are configured by sending configuration commands to them over the bus inside the computer. Such configuration commands can be sent by a PnP BIOS, by the isapnp program (for the ISA bus only), by setpci (PCI bus: can't set IRQs), or by newer serial of how to configure the ones that don't get io-irq configured by the serial driver.

1. ISA bus: Use "isapnp" which may be run automatically at every boot-time
2. Let a PnP BIOS do it, and then maybe tell setserial the IO and IRQ
3. PCI bus: Use lspci -vv to look at it and setpci to configure the IO only (can't set the IRQ).

See Quick Install for more details, especially for the PCI bus.

2.6 Software-based Modems (winmodems, linmodems)

Introduction to software modems (winmodems)

Software modems turn over some (or even almost all) of the work of the modem to the main processor (CPU) chip of your computer (such as a Pentium chip). This requires special software (a modem driver) to do the job. Until late 1999, such software was released only for MS Windows and wouldn't work with Linux. Even worse was that the maker of the modem kept the interface to the modem secret so that no one could write a Linux driver for it (even though a few volunteers were willing to write Linux drivers).

But things have improved some since then so that today (late 2001) many such modems do have a linux driver. There is no standard interface so that different brands/models of software-modems need different drivers (unless the different brands/models happen to use the same chipset internally). But some drivers may not work perfectly nor have all the features that a MS Windows driver has.

Another name for a software modem (used by MS) is "driver-based modem". The conventional hardware-based modem (that works with Linux) doesn't need a modem driver (but does use the Linux serial driver) After about mid-1998 most new internal modems were software modems.

Software modems fall into 2 categories: linmodems and winmodems. Winmodems will only work under MS Windows. Linmodems will work under Linux. They formerly were mostly winmodems so some also call them "winmodems". The term "Winmodem" is also a trademark for a certain model of "winmodem" but that's not the meaning of it in this document.

Linmodems

In late 1999, two software-based modems appeared that could work under Linux and were thus called "linmodems". Lucent Technologies (LT) unofficially released a Linux binary-only code to support most of its PCI modems. PC-TEL (includes "Zoltrix") introduced a new software-based modem for Linux. After that, interest increased for getting winmodems to work under linux. There is a GPL'ed driver for Intel's (Modem Silicon Operations) MD563x HaM chipset (nee Ambient division of Cirrus Logic). As of mid-2001 there are also drivers for: Conexant HSF and HCF, Motorola SM56 (support terminated), ESS (ISA only), and IBM's Mwave for Thinkpads 600+. See http://linmodems.org.

What percent of software modems now (2001) work under Linux? Well, there's a number of modem chips not supported: Lucent/Agere ARM (Scorpio), 3COM/US Robotics, some SmartLink (3 different chipsets), Ambient HSP, and possibly others. So it seems that over half the software modem chips were supported as of late 2001. As of 2005 it seems that the situation has gotten worse. Why? Well, Linux on the Desktop didn't grow as fast as expected and many PC users went for higher speed cable modems and DSL.

Another reason is that many of the drivers were written years ago and will only work for older versions of the Linux kernels. The driver code is secret and the companies don't want to update drivers for hardware they are no longer selling.

Be warned in advance that determining if your modem is a linmodem may not be very easy. You may need to first find out what chipset you have and who makes it. Just knowing the brand and model number of your modem may not be sufficient. One method is to download the scanModem tool from http://linmodems.org but the results may be hard to decipher and you may need to ask for help from the linmodems mailing list. Another way to find this out using say "lspci -v" and then looking up the chip maker using the long modem number. This requires checking a database or searching the Internet. Still another way is to look at the fine

print on the chips on the modem card. All this is not always simple. It could happen that you will put a lot of effort into this only to get the bad news that your modem isn't supported. But even if it is supported, support may only be for an old version of the kernel. See Linmodem-HOWTO for more details.

Linmodem sites and documentation

- Linmodem-HOWTO
- Winmodems-and-Linux-HOWTO (not as well written as Linmodem-HOWTO)
- http://linmodems.org is a project to turn winmodems into linmodems. Has a mailing list.
- Conexant+Rockwell-modem-HOWTO
- old modem list Has links to linmodem info, but not maintained after 2003.
- PCTel-HSP-MicroModem-Configuration-mini-HOWTO

Software-based modem types

There are two basic types of software modems. In one type the software does almost all of the work. The other is where the software only does the "control" operations (which is everything except processing the digital waveshapes --to be explained later). Since the hardware doesn't do the control it's called a "controllerless" modem. The first type is an all-software modem (sometimes just called a software modem).

For both of these types there must be analog hardware in the modem (or on the motherboard) to generate an electrical waveshape to send out the phone line. It's generated from a digital signal (which is sort of a "digital waveshape"). It's something like the digital electronics creates a lot of discrete points on graph paper and then the modem draws a smooth voltage curve thru them. There must also be hardware to convert the incoming waveshape to digital. This is just analog-to-digital conversion (and conversely). It's done by a codec (coder-decoder).

The incoming digital waveshape must be converted to a data byte stream. This is part of the demodulation process. Recall that these data bytes have likely been compressed, so they are not at all like the original message. Turning data bytes into a digital waveshape is part of the modulation process. Even after demodulation is done, the modem can't just send the resulting incoming data byte stream to the serial port input buffers, but must first do decompression, error correction, and convert from serial to the parallel bus of the computer. But the modem may get the CPU to do the actual work. It's the reverse sequence for an outgoing data byte stream.

The difference between the two types of software-based modems is where the digital modulation takes place. In the all-software modem this modulation is done in the CPU and it's called a Host Signal Processor (HSP). In the controllerless modem it's done in the modem but all other digital work is done by the CPU. This other digital work consists of dealing with AT-commands, data compression, error correction, and simulating a serial port. In the all-software modem, there are still two items handled by hardware: the A/D conversion of waveshapes by the codec and echo cancellation.

Is this modem a software modem?

How do you determine if an internal modem is a software modem? First see if the name, description of it, or even the name of the MS Windows driver for it indicates it's a software modem: HSP (Host Signal Processor) , HCF (Host Controlled Family), HSF (Host Signal Family), controllerless, host-controlled, host-based, and soft-... modem. If it's one of these modem it will only work for the cases where a Linux driver is available. Since software modems cost less, a low price is a clue that it's a software modem.

If you don't know the model of the modem and you also have Windows on your Linux PC, click on the "Modem" icon in the "Control Panel". Then see the modem list (not maintained after 2003). If the above doesn't work (or isn't feasible), you can look at the package the modem came in (or a manual). Read the section on the package that says something like "Minimum System Requirements" or just "System Requirements".

A hardware modem will work fine on old CPUs (such as the 386 or better). So if it requires a modern CPU (such as a Pentium or other "high speed" CPU of say over 150 MHz) this is a clue that it's a all-software modem. If it only requires a 486 CPU (or better) then it's likely a host-controlled software modem. Saying that it only works with Windows is also bad news. However, even in this case there may be a Linux driver for it or it could be a mistake in labeling.

Otherwise, it may be a hardware modem if it fails to state explicitly that you must have Windows. By saying it's "designed for Windows" it may only mean that it fully supports Microsoft's plug-and-play which is OK since Linux uses the same plug-and-play specs (but it's harder to configure under Linux). Being "designed for Windows" thus gives no clue as to whether or not it will work under Linux. You might check the Website of the manufacturer or inquire via email. Some manufacturers are specifically stating that certain models work under Linux. Sometimes they are linmodems that require you to obtain and install a certain linmodem driver.

Should I get a software modem?

Only if you know there is a Linux driver for it that works OK. But there may be a problem if the driver isn't being maintained and as a result doesn't work with future versions of the kernel. Also, the driver may not have full functionality. Besides the problems of getting a satisfactory driver, what are the pros and cons of software modems? Since the software modem uses the CPU to do some (or all) of its work, the software modem requires less on-board electronics on the modem card and thus costs less. At the same time, the CPU work load is increased by the modem which may result in slower operation.

The percentage of loading of the CPU by the modem depends on both what CPU you have and whether or not it's an all-software modem. For a modern CPU and a modem that only uses the CPU as a controller, there's little loss of performance. Even if it's an all-software modem, you will not suffer a loss of performance if there are no other CPU-intensive tasks are running at the same time. Of course, when you're not using the software modem there is no degradation in performance at all.

Is the modem cost savings worth it? In many cases yes, especially if you don't use the modem much and/or are not running any other CPU intensive tasks when the modem is in use. The savings in modem cost could be used for a better CPU which would speed things up a little. But the on-board electronics of a hardware modem can do the job more efficiently than a general purpose CPU (except that it's not efficient at all when it's not in use). So if you use the modem a lot it's probably better to avoid all-software modems.

2.7 PCI Modems

A PCI modem card is one which inserts into a PCI-bus slot on the motherboard of a PC. While many PCI winmodems will not work under Linux (no driver available) other PCI modems will work under Linux. The Linux serial driver has been modified to support certain PCI hardware modem cards (but not winmodems/linmodems). If it's a linmodem, it will work only if you install a certain linmodem driver. If the Linux serial driver supports your hardware modem then the driver will set up the PnP configuration for you. See PCI Bus Support Underway. If no special support for your PCI hardware modem is in the Linux serial driver it may still work OK but you have to do some work to configure it.

2.8 AMR Modems

These are mainly used in laptops. They are all winmodems that insert into a special AMR (Audio Modem Riser) slot on the motherboard. Audio cards or combined audio-modem cards are sometimes used in this slot. The slot's main use is for HSF type modems where the CPU does almost all of the work. This results in a small "modem" card and thus a short AMR slot. The motherboard has a codec which takes digital output from the CPU and generates analog voltage waves at the ARM slot (and conversely). Thus the "modem" that plugs into the slot has little to do except to interface the telephone line with the codec. Linux supports at least one AMR modem. lspci -v should display it.

2.9 USB Modems

USB = Universal Serial Bus. Most USB modems are winmodems, so many will not work with Linux. Linux has support for modems that conform to the USB Communication Device Class Abstract Control Model (= USB CDC ACM). There's a module for ACM named acm.o. See the /usb/acm.txt document in the kernel documentation directory (/usr/share/doc/kernel-doc-2.6.x in Debian, perhaps /usr/doc/kernel... in some distributions). The ACM "serial port" for the first (0th) such modem is: /dev/usb/acm/0 or possibly /dev/usb/ttyACM0. This should be the case regardless of whether or not you use the new "device file system". It's not really a serial port, but the driver makes it look like a serial port to software which uses the modem.

Since the bandwidth on the USB is high it's possible to send a lot more that just data to a USB modem. This means that it's feasible to create a USB winmodem where the driver does most of the modem's work on the CPU and sends the results to the modem. So beware of USB winmodems (unless they have Linux support).

2.10 Which Internal Modems might not work with Linux

- Software-based Modems (winmodems, linmodems) Only about half have a Linux driver available.
- MWave and DSP Modems might work, but only if you first start Windows/Dos each time you power on your PC.
- Modems with Old Rockwell (RPI) Drivers work but with reduced performance.

MWave and some DSP Modems

Note that there's now a Linux driver for the ACP (Mwave) modem used in IBM Thinkpads 600+. See the mini-HOWTO: ACP-Modem.

While hardware modems used use DSPs (Digital Signal Processors) some of these DSPs are programmed by a driver which must be downloaded from the hard disk to the DSPs memory just before using the modem. Unfortunately, such downloading is normally done by Dos/Windows programs (which doesn't work for Linux). But there has been substantial success in getting some of these modems to work with Linux. For example, there is a Linux driver available to run a Lucent (DSP) modem.

Ordinary modems that work fine with Linux (without needing a driver for the modem) often have a DSP too (and may mention this on the packaging), but the program that runs the DSP is stored inside the modem. These work fine under Linux. An example of a DSP modem that has problems working under Linux is the old IBM's Aptiva MWAVE.

One way to get some DSP modems to work with Linux is to boot from DOS (if you have it on your Linux PC). You first install the driver under DOS (using DOS and not Window drivers). Then start Dos/Windows

and start the driver for the modem so as to program the DSP. Then without turning off the computer, start Linux.

One may write a "batch" file (actually a script) to do this. Here is an example but you must modify it to suit your situation.

```
rem mwave is a batch file supplied by the modem maker
call c:\mww\dll\mwave start
rem loadlin.exe is a DOS program that will boot Linux from DOS (See
rem Config-HOWTO).
c:\linux\loadlin f:\vmlinuz root=/dev/hda3 ro
```

One may create an icon for the Window's desktop which points to such a batch file and set the icon properties to "Run in MSDOS Mode". Then by clicking on this icon one sets up the modem and goes to Linux. Another possible way to boot Linux from DOS is to press CTRL-ALT-DEL and tell it to reboot (assuming that you have set things up so that you can boot directly into Linux). The modem remains on the same com port (same IO address) that it used under DOS.

The Newcom ifx modem needs a small kernel patch to work correctly since its simulation of a serial port is non-standard. The patch and other info for using this modem with Linux is at http://quinine.pharmacy.ohio-state.edu/~ejolson/linux/newcom.html.

Old Rockwell (RPI) Drivers

Some older Rockwell chips need Rockwell RPI (Rockwell Protocol Interface) drivers for compression and error correction. They can still be used with Linux even though the driver software works only under MS Windows. This is because the MS Windows software (which you don't have) does only compression and error correction. If you are willing to operate the modem without compression and error correction then it's feasible to use it with Linux. To do this you will need to disable RPI by sending the modem (via the initialization string) a "RPI disable" command each time you power on your modem. On my old modem this command was +H0. Not having data compression available makes it slower to get webpages but is just as fast when downloading files that are already compressed.

3. Modem Pools, RAS

3.1 Introduction

A "modem pool" is a group of modems which are normally used to receive incoming calls. Today, many such modems may be on a single card. ISPs once used modem pools so that customers could call in to the ISP, but today, the RAS (Remote Access Server) has replaced modem pools for ISPs. RAS works for incoming calls at near 56k (V.90 and V.92) and uses what amounts to "digital modems". Modem pools use the older analog modems and can only go to 33.6 kbps for incoming calls. Thus analog modem pools are more likely to be used by small organizations that don't want to use the more expensive RAS. A RAS is in a sense a digital modem pool.

An analog modem pool may be provided by an analog multi-port modem card. In olden days it was many modems in an external chassis (something like an external modem). Such modems could be analog modems similar to modems used for home/office PCs (can't send above 33.6k even if they are labeled "56k modems"). A RAS will use "digital modems" which can send at nearly 56k (if you have a good line). The "digital modems" require a digital connection to the telephone line and don't use any serial ports at all. All of these

modem pools (or RAS's) will require that you install special drivers for them.

3.2 Analog Modem Pools, Multi-modem Cards

A "multimodem card" is short for "multiport modem card". Some put a hyphen after "multi": multi-modem or multi-port. An analog modem pool is just many analog modems (the common home/office modem) provided either on an internal plug-in card or in an external chassis. Each modem comes with a built-in serial port. There is usually a system of sharing interrupts or of handling interrupts by their own electronics, thus removing much of this burden from the CPU. Note that these modems are not "digital modems" and will thus not be able to use 56k for people who dial-in.

Here is a list of some companies that make analog multiport modem cards which plug into slots in a PC. 8 modems/card is common. The cards listed claim to work with Linux and the websites should point you to a driver for them.

Multi-modem Cards (analog, not digital):

- Equinox SST Multi-modem. PCI, 56k, 4 or 8 ports
  http://www.equinox.com/product/multi-modem.htm
- MultiModemISI by Multi-Tech Systems. 56k or 33.6k, PCI, 4 or 8 ports. ISDN/56k hybrids.
  http://www.multitech.com/PRODUCTS/MultiModemISI/
- PCI-RAS cards by Perle. 56k, 4 or 8 ports.
  http://www.perle.com/products/default.asp?cat=C007
- RocketModem by Comtrol. ISA 33.6k, 4 or 8 port.
  http://www.comtrol.com/sales/specs/rm.htm
- RocketModem II by Comtrol. PCI 56k, 4 or 6 port
  http://www.comtrol.com/sales/specs/rmii.htm
- RockForce. 56k, 2 or 4 port Two port V.92/V.44
  http://www.mainpine.com/ #RockForce+ Two port V.90 (www.mainpine.com/prodrockplus.html) #RockForceDUO Two port V.92/V.44 (www.mainpine.com/prodduo.html) #RockForceQUATRO Four port V.92/V.44 (www.mainpine.com/prodquatro.html) #RockForceDUO+ Two port V.92/V.44/V.34 SuperG3 Fax = #(www.mainpine.com/prodduoplus.html) #RockForceQUATRO+ Four port V.92/V.44/V.34 SuperG3 Fax = #(www.mainpine.com/prodquatroplus.html) #
- Multi-modem communication adapters by Digi.
  http:/www.dgii.com/solutions/mmcommadapters/index.html

3.3 Digital Modems, RAS

"Digital modems" are much different than the analog modems that most people use in their PCs. But they can communicate with analog modems at the other end of the phone line. ISP's use "digital modems" to send out data at almost 56k bps to 56k modems in homes and offices. The "digital modem" requires a digital connection to the telephone line (such as T1, EI, ISDN PRI, etc.). Except for some serial ISDN "modems", they don't use serial ports for the interface to the computer. Instead, they interface directly to a computer bus via a special card(s) (which may also contain the "digital modems"). They are often a component of "remote access servers" (RASs) or "digital modem pools"

You may already know that each time you make a telephone call from an analog device (a telephone or a modem) it gets converted by the telephone company to a digital signal. Then it's transmitted to near its destination as a digital signal and finally converted back into an analog signal just before it reaches it's destination. But it's also possible to receive this digital signal directly from the telephone company if you have

what is called a "T1" line, etc.

The cables from the phone company that carry digital signals have been designed for high bandwidth so that the same cable carries many telephone calls. It's done by what's called "time-division multiplexing". A single phone call in a cable is carried on two different channels, one for each direction. So the RAS must connect each such channel-pair to the appropriate "digital modem" that services that phone call. Such tasks are done by what is sometimes called a "... concentrator".

Now the digital signal received by a "digital modem" may really represent an analog signal which has been sent to it by an analog modem. This is because when you send an analog signal (including ordinary voice) to the telephone company, it gets converted into digital by the phone company. One way for the digital modem to deal with this digital signal would be to convert it to an analog signal and then put that thru an analog modem to get the digital data sent by the analog modem. But why do all this work? Since the signal is already in digital form, why not process it digitally? That's how it's done. The digital signal is processed and converted to another digital stream of bytes which represents data bytes sent by the analog modem. A "digital signal processor" (DSP) is commonly used for this task. A CPU could also handle it but it would be heavily loaded.

Likewise, a "digital modem" must handle sending digital signals in the opposite direction from a RAS to a digital telephone line. Thus it only makes digital-to-digital conversions and doesn't deal in analog at all. It thus is not really a modem at all since it doesn't modulate any analog carrier. So the name "digital modem" is a misnomer but it does do the job formerly done by modems. Thus some "digital modems" call themselves "digital signal processors", or "remote access servers", etc. and may not even mention the word "modem".

Such a RAS system may be a stand-alone proprietary server, a chassis containing digital modems that connects to a PC via a special interface card, or just a card itself. Digi calls one such card a "remote access server concentrator adapter". One incomplete description of what is needed to become an ISP is: See What do I need to be an ISP?. Cyclades promotes their own products here so please do comparison shopping before buying anything.

4. Serial Port and Modem Basics

You don't have to understand the basics to use and install a modem. But understanding it may help to determine what is wrong if you run into problems. After reading this section, if you want to understand it even better you may want to see How Modems Work in this document (not yet complete). More details on the serial port (including much of this section) will be found in Serial-HOWTO.

4.1 Modem Converts Digital to Analog (and conversely)

Most all telephone main lines are digital already but the lines leading to your house (or business) are usually analog which means that they were designed to transmit a voltage wave which is an exact replica of the sound wave coming out of your mouth. Such a voltage wave is called "analog". If viewed on an oscilloscope it looks like a sine wave of varying frequency and amplitude. A digital signal is like a square wave. For example 3 v (volts) might be a 1-bit and 0 v could be a 0-bit. For most serial ports (used by external modems) +12 v is a 0-bit and -12 v is a 1-bit (some are + or - 5 v).

To send data from your computer over the phone line, the modem takes the digital signal from your computer and converts it to "analog". It does this by both creating an analog sine wave and then "MODulating" it. Since the result still represents digital data, it could also be called a digital signal instead of analog. But it looks something like an analog signal and almost everyone calls it analog. At the other end of the phone line another

modem "DEModulates" this signal and the pure digital signal is recovered. Put together the "mod" and "dem" parts of the two words above and you get "modem" (if you drop one of the two d's). A "modem" is thus a MODulator-DEModulator. Just what modulation is may be found in the section Modulation Details.

4.2 What is a Serial Port ?

Intro to Serial

The UART serial port (or just "serial port for short" is an I/O (Input/Output) device. Since modems have a serial port between them and the computer, it's necessary to understand the serial port as well as the modem.

Most PC's have one or two serial ports. Each has a 9-pin connector (sometimes 25-pin) on the back of the computer. Computer programs can send data (bytes) to the transmit pin (output) and receive bytes from the receive pin (input). The other pins are for control purposes and ground.

The serial port is much more than just a connector. It converts the data from parallel to serial and changes the electrical representation of the data. Inside the computer, data bits flow in parallel (using many wires at the same time). Serial flow is a stream of bits over a single wire (such as on the transmit or receive pin of the serial connector). For the serial port to create such a flow, it must convert data from parallel (inside the computer) to serial on the transmit pin (and conversely).

Most of the electronics of the serial port is found in a computer chip (or a part of a chip) known as a UART. For more details on UARTs see the section "What are UARTS" in the Serial-HOWTO. But you may want to finish this section first so that you will hopefully understand how the UART fits into the overall scheme of things.

Pins and Wires

Old PC's used 25 pin connectors but only about 9 pins were actually used so today most connectors are only 9-pin. Each of the 9 pins usually connects to a wire. Besides the two wires used for transmitting and receiving data, another pin (wire) is signal ground. The voltage on any wire is measured with respect to this ground. Thus the minimum number of wires to use for 2-way transmission of data is 3. Except that it has been known to work with no signal ground wire but with degraded performance and sometimes with errors.

There are still more wires which are for control purposes (signalling) only and not for sending bytes. All of these signals could have been shared on a single wire, but instead, there is a separate dedicated wire for every type of signal. Some (or all) of these control wires are called "modem control lines". Modem control wires are either in the asserted state (on) of +12 volts or in the negated state (off) of -12 volts. One of these wires is to signal the computer to stop sending bytes out the serial port cable. Conversely, another wire signals the device attached to the serial port to stop sending bytes to the computer. If the attached device is a modem, other wires may tell the modem to hang up the telephone line or tell the computer that a connection has been made or that the telephone line is ringing (someone is attempting to call in). See the Serial-HOWTO: Pinout and Signals for more details.

Internal Modem Contains Serial Port

For an internal modem there is no 9-pin connector but the behavior is almost exactly as if the above mentioned cable wires existed. Instead of a 12 volt signal in a wire giving the state of a modem control line, the internal modem may just use a status bit in its own memory (a register) to determine the state of this

non-existent "wire". The internal modem's serial port looks just like a real serial port to the computer. It even includes the speed limits that one may set at ordinary serial ports such as 115200 bits/sec.

Unfortunately for Linux, many internal modems today use software (running on the CPU) to do much of the modem's work. Unfortunately, such software may be only available for the MS Windows OS (it hasn't been ported to Linux). Thus you can't use some of these modems with Linux See Software-based Modems (winmodems).

4.3 IO Address & IRQ

Since the computer needs to communicate with each serial port, the operating system must know that each serial port exists and where it is (its I/O address). It also needs to know which wire (IRQ number) the serial port must use to request service from the computer's CPU. It requests service by sending an interrupt voltage on this wire. Thus every serial port device must store in its non-volatile memory both its I/O address and its Interrupt ReQuest number: IRQ. See Interrupts. The PCI bus has its own system of interrupts. But since the PCI-aware BIOS sets up these PCI interrupts to map to IRQs, it seemingly behaves just as described above. Except that sharing of PCI interrupts is allowed (2 or more devices may use the same IRQ number).

I/O addresses are not the same as memory addresses. When an I/O addresses is put onto the computer's address bus, another wire is energized. This both tells main memory to ignore the address and tells all devices which have I/O addresses (such as the serial port) to listen to the address sent on the bus to see if it matches the device's. If the address matches, then the I/O device reads the data on the data bus.

The I/O address of a certain device (such as ttyS2) will actually be a range of addresses. The lower address in this range is the base address. "address" usually means just the "base address".

4.4 Names: ttyS0, ttyS1, etc.

The serial ports are named ttyS0, ttyS1, etc. (and usually correspond respectively to COM1, COM2, etc. in DOS/Windows). The /dev directory has a special file for each port. Type "ls /dev/ttyS*" to see them. Just because there may be (for example) a ttyS3 file, doesn't necessarily mean that there exists a physical serial port there.

Which one of these names (ttyS0, ttyS1, etc.) refers to which physical serial port is determined as follows. The serial driver (software) maintains a table showing which I/O address corresponds to which ttyS. This mapping of names (such as ttyS1) to I/O addresses (and IRQ's) may be both set and viewed by the "setserial" command. See What is Setserial. This does `not` set the I/O address and IRQ in the hardware itself (which is set by jumpers or by plug-and-play software). Thus which physical port corresponds to say ttyS1 depends both on what the serial driver thinks (per setserial) and what is set in the hardware. If a mistake has been made, the physical port may not correspond to any name (such as ttyS2) and thus it can't be used. See Serial Port Devices /dev/ttyS2, etc. for more details.

4.5 Interrupts

Bytes come in over the phone line to the modem, are converted from analog to digital by the modem and passed along to the serial port on their way to their destination inside your computer. When the serial port receives a number of bytes (may be set to 1, 4, 8, or 14) into its FIFO buffer, it signals the CPU to fetch them by sending an electrical signal known as an interrupt on a certain wire normally used only by that port. Thus the FIFO waits until it has received a number of bytes and then issues an interrupt.

However, this interrupt will also be sent if there is an unexpected delay while waiting for the next byte to arrive (known as a timeout). Thus if the bytes are being received slowly (such as from someone typing on a terminal keyboard) there may be an interrupt issued for every byte received. For some UART chips the rule is like this: If 4 bytes in a row could have been received in an interval of time, but none of these 4 show up, then the port gives up waiting for more bytes and issues an interrupt to fetch the bytes currently in the FIFO. Of course, if the FIFO is empty, no interrupt will be issued.

Each interrupt conductor (inside the computer) has a number (IRQ) and the serial port must know which conductor to use to signal on. For example, ttyS0 normally uses IRQ number 4 known as IRQ4 (or IRQ 4). A list of them and more will be found in "man setserial" (search for "Configuring Serial Ports"). Interrupts are issued whenever the serial port needs to get the CPU's attention. It's important to do this in a timely manner since the buffer inside the serial port can hold only 16 incoming bytes. If the CPU fails to remove such received bytes promptly, then there will not be any space left for any more incoming bytes and the small buffer may overflow (overrun) resulting in a loss of data bytes.

For an external modem, there may be no way (such as flow control) to stop the flow rapidly enough to prevent such an overrun. For an internal modem, the 16-byte FIFO buffer is on the same card and most modems will not write to it if it's full. Thus it will not overrun the 16-byte buffers but it may need to use Modem-to-Modem Flow Control to prevent the modem itself from being overrun. This is one advantage of internal modems over an external.

Interrupts are also issued when the serial port has just sent out all of its bytes from its small transmit FIFO buffer out the external cable. It then has space for 16 more outgoing bytes. The interrupt is to notify the CPU of that fact so that it may put more bytes in the small transmit buffer to be transmitted. Also, when a modem control line changes state, an interrupt is issued.

The buffers mentioned above are all hardware buffers. The serial port also has large buffers in main memory. This will be explained later

Interrupts convey a lot of information but only indirectly. The interrupt itself just tells a chip called the interrupt controller that a certain serial port needs attention. The interrupt controller then signals the CPU. The CPU then runs a special program to service the serial port. That program is called an interrupt service routine (part of the serial driver software). It tries to find out what has happened at the serial port and then deals with the problem such a transferring bytes from (or to) the serial port's hardware buffer. This program can easily find out what has happened since the serial port has registers at IO addresses known to the serial driver software. These registers contain status information about the serial port. The software reads these registers and by inspecting the contents, finds out what has happened and takes appropriate action.

4.6 Data Compression (by the Modem)

Before continuing with the basics of the serial port, one needs to understand about something done by the modem: data compression. In some cases this task is actually done by software run on the computer's CPU but unfortunately at present, such software only works for MS Windows. The discussion here will be for the case where the modem itself does the compression since this is what must happen in order for the modem to work under Linux.

In order to send data faster over the phone line, one may compress (encode it) using a custom encoding scheme which itself depends on the data. The encoded data is smaller than the original (less bytes) and can be sent over the Internet in less time. This process is called "data compression".

If you download files from the Internet, they are likely already compressed and it is not feasible for the modem to try to compress them further. Your modem may sense that what is passing thru has already been compressed and refrain from trying a compress it any more. If you are receiving data which has been compressed by the other modem, your modem will decompress it and create many more bytes than were sent over the phone line. Thus the flow of data from your modem into your computer will be higher than the flow over the phone line to you. The ratio of this flow is called the compression ratio. Compression ratios as high as 4 are possible, but not very likely.

4.7 Error Correction

Similar to data compression, modems may be set to do error correction. While there is some overhead cost involved which slows down the byte/sec flow rate, the fact that error correction strips off start and stop bits actually increases the data byte/sec flow rate.

For the serial port's interface with the external world, each 8-bit byte has 2 extra bits added to it: a start-bit and a stop-bit. Without error correction, these extra start and stop bits usually go right thru the modem and out over the phone lines. But when error correction is enabled, these extra bits are stripped off and the 8-bit bytes are put into packets. This is more efficient and results in higher byte/sec flow in spite of the fact that there are a few more bytes added for packet headers and error correction purposes.

4.8 Data Flow (Speeds)

Data (bytes representing letters, pictures, etc.) flows from your computer to your modem and then out on the telephone line (and conversely). Flow rates (such as 56k (56000) bits/sec) are (incorrectly) called "speed". But almost everyone says "speed" instead of "flow rate". If there were no data compression the flow rate from the computer to the modem would be about the same as the flow rate over the telephone line.

Actually there are two different speeds to consider at your end of the phone line:

- The speed on the phone line itself (DCE speed) modem-to-modem
- The speed from your computer's serial port to your modem (DTE speed)

When you dial out and connect to another modem on the other end of the phone line, your modem often sends you a message like "CONNECT 28800" or "CONNECT 115200". What do these mean? Well, its either the DCE speed or the DTE speed. If it's higher than the advertised modem speed it must be the DTE modem-to-computer speed. This is the case for the 115200 speed shown above. The 28800 must be a DCE (modem-to-modem) speed since the serial port has no such speed. One may configure the modem to report either speed. Some modems report both speeds and report the modem-to-modem speed as (for example): CARRIER 28800.

If you have an internal modem you would not expect that there would be any speed limit on the DTE speed from your modem to your computer since you modem is inside your computer and is almost part of your computer. But there usually is since the modem contains a dedicated serial port within it. On some software modems there is no such speed limit.

It's important to understand that the average speed is often less than the specified speed, especially on the short DTE computer-to-modem line. Waits (or idle time) result in a lower average speed. These waits may include long waits of perhaps a second due to Flow Control. At the other extreme there may be very short waits (idle time) of several micro-seconds separating the end of one byte and the start of the next byte. In addition, modems will fallback to lower speeds if the telephone line conditions are less than pristine.

For a discussion of what DTE speed is best to use see section What Speed Should I Use.

4.9 Flow Control

Flow control means the ability to slow down the flow of bytes in a wire. For serial ports this means the ability to stop and then restart the flow without any loss of bytes. Flow control is needed for modems and other hardware to allow a jump in instantaneous flow rates.

Example of Flow Control

For example, consider the case where you connect a 33.6k external modem via a short cable to your serial port. The modem sends and receives bytes over the phone line at 33.6k bits per second (bps). Assume it's not doing any data compression or error correction. You have set the serial port speed to 115,200 bits/sec (bps), and you are sending data from your computer to the phone line. Then the flow from the your computer to your modem over the short cable is at 115.2k bps. However the flow from your modem out the phone line is only 33.6k bps. Since a faster flow (115.2k) is going into your modem than is coming out of it, the modem is storing the excess flow (115.2k -33.6k = 81.6k bps) in one of its buffers. This buffer would soon overrun (run out of free storage space) unless the high 115.2k flow is stopped.

But now flow control comes to the rescue. When the modem's buffer is almost full, the modem sends a stop signal to the serial port. The serial port passes on the stop signal on to the device driver and the 115.2k bps flow is halted. Then the modem continues to send out data at 33.6k bps drawing on the data it previous accumulated in its buffer. Since nothing is coming into this buffer, the number of bytes in it starts to drop. When almost no bytes are left in the buffer, the modem sends a start signal to the serial port and the 115.2k flow from the computer to the modem resumes. In effect, flow control creates an average flow rate in the short cable (in this case 33.6k) which is significantly less than the "on" flow rate of 115.2k bps. This is "start-stop" flow control.

In the above simple example it was assumed that the modem did no data compression. This could happen when the modem is sending a file which is already compressed and can't be compressed further. Now let's consider the opposite extreme where the modem is compressing the data with a high compression ratio. In such a case the modem might need an input flow rate of say 115.2k bps to provide an output (to the phone line) of 33.6k bps (compressed data). This compression ratio is 3.43 (115.2/33.6). In this case the modem is able to compress the 115.2 bps PC-to-modem flow and send the same data (in compressed form) out the phone line at 33.6bps. There's no need for flow control here so long as the compression ratio remains higher than 3.43. But the compression ratio varies from second to second and if it should drop below 3.43, flow control will be needed

In the above example, the modem was an external modem. But the same situation exists (as of early 2003) for most internal modems. There is still a speed limit on the PC-to-modem speed even though this flow doesn't take place over an external cable. This makes the internal modems compatible with the external modems.

In the above example of flow control, the flow was from the computer to a modem. But there is also flow control which is used for the opposite direction of flow: from a modem (or other device) to a computer. Each direction of flow involves 3 buffers: 1. in the modem 2. in the UART chip (called FIFOs) and 3. in main memory managed by the serial driver. Flow control protects all buffers (except the FIFOs) from overflowing.

Under Linux, the small UART FIFO buffers are not protected by flow control but instead rely on a fast response to the interrupts they issue. Some UART chips can be set to do hardware flow control to protect their FIFOs but Linux (as of early 2003) doesn't seem to support it. FIFO stand for "First In, First Out" which is the

way it handles bytes in a queue. All the 3 buffers use the FIFO rule but only the one in the UART is named "FIFO". This is the essence of flow control but there are still some more details.

If you have set the highest PC-to-modem speed, you may not need flow control in the direction from the modem to a PC. For a complex example of a case where it's needed see "Complex Flow Control Example" in the Serial-HOWTO. To slow down this incoming flow, your modem must tell the other modem to stop sending. See M-M_flow_c name="Modem-to-Modem Flow Control">.

Hardware vs. Software Flow Control

If feasible, it's best to use "hardware" flow control that uses two dedicated "modem control" wires to send the "stop" and "start" signals. Hardware flow control at the serial port works like this: The two pins, RTS (Request to send) and CTS (Clear to send) are used. When the computer is ready to receive date it asserts RTS by putting a positive voltage on the RTS pin (meaning "Request To Send to me"). When the computer is not able to receive any more bytes, it negates RTS by putting a negative voltage on the pin saying: "stop sending to me". The RTS pin is connected by the serial cable to another pin on the modem, printer, terminal, etc. This other pin's only function is to receive this signal.

For the case of a modem, this "other" pin will be the modem's RTS pin. But for a printer, another PC, or a non-modem device, it's usually a CTS pin so a "crossover" or "null modem" cable is required. This cable connects the CTS pin at one end with the RTS pin at the other end (two wires since each end of the cable has a CTS pin). For a modem, a straight-thru cable is used.

For the opposite direction of flow a similar scheme is used. For a modem, the CTS pin is used to send the flow control signal to the CTS pin on the PC. For a non-modem, the RTS pin sends the signal. Thus modems and non-modems have the roles of their RTS and CTS pins interchanged. Some non-modems such as dumb terminals may use other pins for flow control such as the DTR pin instead of RTS.

Software flow control uses the main receive and transmit data wires to send the start and stop signals. It uses the ASCII control characters DC1 (start) and DC3 (stop) for this purpose. They are just inserted into the regular stream of data. Software flow control is not only slower in reacting but also does not allow the sending of binary data unless special precautions are taken. Since binary data will likely contain DC1 and DC3 characters, special means must be taken to distinguish between a DC3 that means a flow control stop and a DC3 that is part of the binary code. Likewise for DC1. To get software flow control to work for binary data requires both modem (hardware) and software support.

Symptoms of No Flow Control

Understanding flow-control theory can be of practical use. For example I used my modem to access the Internet and it seemed to work fine. But after a few months, I tried to send out long files from my PC and a huge amount of retries and errors resulted but it finally succeeded. Receiving in the other direction (from my ISP to me) worked fine. The problem turned out to be a modem with flow control disabled. My modem's buffer was overflowing (overrunning) on long outgoing files since no "stop" signal was ever sent to my computer to halt sending to the modem. There was no problem in the direction from the modem to my computer since the capacity (say 115.2 kbps) of the serial port was always higher than the flow from the telephone line. But it was a problem in the other direction where the PC would send at 115.2 kbps and the modem couldn't handle this high flow resulting in overruns of the modem's buffer. The fix was to enable flow control by putting into the modem's init string an enable-flow-control command.

Modem-to-Modem Flow Control

This is the flow control of the data sent over the telephone lines between two modems. It works as long as error correction is enabled. Actually, even without error correction it's possible to enable software flow control between modems but it may interfere with sending binary data so it's not often used.

4.10 Data Flow Path; Buffers

It's been mentioned that there are 3 buffers for each direction of flow (3 pairs altogether): 16-byte FIFO buffers (in the UART), a pair of larger buffers inside a device connected to the serial port (such as a modem), and a pair of buffers (say 8k) in main memory. When an application program sends bytes to the serial port they first get stashed in the transmit serial port buffer in main memory. The other member of this pair consists of a receive buffer for the opposite direction of byte-flow. Here's an example diagram for the case of browsing the Internet with a browser. Transmit data flow is left to right while receive flow is right to left. There is a separate buffer for each direction of flow.

```
application     8k-byte         16-byte        1k-byte        tele-
BROWSER ------- MEMORY -------- FIFO --------- MODEM -------- phone
program         buffer          buffer         buffer         line
```

For the transmit case, the serial device driver takes out say 15 bytes from this transmit buffer (in main memory), one byte at a time and puts them into the 16-byte transmit buffer in the serial UART for transmission. Once in that transmit buffer, there is no way to stop them from being transmitted. They are then transmitted to the modem or (other device connected to the serial port) which also has a fair sized (say 1k) buffer. When the device driver (on orders from flow control sent from the modem) stops the flow of outgoing bytes from the computer, what it actually stops is the flow of outgoing bytes from the large transmit buffer in main memory. Even after this has happened and the flow to the modem has stopped, an application program may keep sending bytes to the 8k transmit buffer until it becomes fill. At the same time, the bytes stored in the FIFO and continue to be sent out. Bytes stored in the modem will continue to be sent out the phone line unless the modem has gotten a modem-to-modem flow control stop from the modem at the other end of the phone line.

When the memory buffer gets fill, the application program can't send any more bytes to it (a "write" statement in a C-program blocks) and the application program temporarily stops running and waits until some buffer space becomes available. Thus a flow control "stop" is ultimately able to stop the program that is sending the bytes. Even though this program stops, the computer does not necessarily stop computing since it may switch to running other processes while it's waiting at a flow control stop.

The above was a little oversimplified in three ways. First, some UARTs can do automatic hardware flow control which can stop the transmission out of the FIFO buffers if needed (not yet supported by Linux). Second, while an application process is waiting to write to the transmit buffer, it could possibly perform other tasks. Third, the serial driver (located between the memory buffer and the FIFO) has it's own small buffer (in main memory) used to process characters.

4.11 Modem Commands

Commands to the modem are sent to it from the communication software over the same conductor as used to send data. The commands are short ASCII strings. Examples are "AT&K3" for enabling hardware flow control (RTS/CTS) between your computer and modem; and "ATDT5393401 for Dialing the number 5393401. Note all commands are prefaced by "AT". Some commands such as enabling flow control help

configure the modem. Other commands such as dialing a number actually do something. There are about a hundred or so different possible commands. When your communication software starts running, it first sends an "init" string of commands to the modem to configure it. All commands are sent on the ordinary data line before the modem dials (or receives a call).

Once the modem is connected to another modem (on-line mode), everything that is sent from your computer to your modem goes directly to the other modem and is not interpreted by the modem as a command. There is a way to "escape" from this mode of operation and go back to command mode where everything sent to the modem will be interpreted as a command. The computer just sends "+++" with a specified time spacing before and after it. If this time spacing is correct, the modem reverts to command mode. Another way to do this is by a signal on a certain modem control line.

There are a number of lists of modem commands on the Internet. The section Web Sites has links to a couple of such web-sites. Different models and brands of modems do not use exactly the same set of such commands. So what works for one modem might not work for another. Some common commands (not guaranteed to work on all modems) are listed in this HOWTO in the section Modem Configuration

4.12 Serial Driver Module

The device driver for the serial port is the software that operates the serial port. It is now provided as a serial module. From kernel 2.2 on, this module will normally get loaded automatically if it's needed. In earlier kernels, you had to have `kerneld` running in order to do auto-load modules on demand. Otherwise the serial module needed to be explicitly listed in /etc/modules. Before modules became popular with Linux, the serial driver was usually built into the kernel (and sometimes still is). If it's built-in don't let the serial module load or else you will have two serial drivers running at the same time. With 2 drivers there are all sorts of errors including a possible "I/O error" when attempting to open a serial port. Use "lsmod" to see if the module is loaded.

When the serial module is loaded it displays a message on the screen about the existing serial ports (often showing a wrong IRQ). But once the module is used by `setserial` to tell the device driver the (hopefully) correct IRQ then you should see a second display similar to the first but with the correct IRQ, etc. See "Serial Module" in the Serial-HOWTO. See What is Setserial for more info on `setserial`.

5. Configuring Overview

Since each modem has an associated serial port and the port has both hardware and software, there are three parts to configuring a modem:

- Locate the serial port hardware: IO address, IRQ; Done by PnP methods or jumpers, setserial. See Locating the Serial Port: IO address IRQs What is Setserial
- Configure the serial port driver (high-level): Done by the communication program (stty-like). Sets speed, flow control, etc. See Configuring the Serial Driver (high-level) See What is stty ?
- Configure the modem itself: Done by the communication program See Modem Configuration

The above omits a few other things that "setserial" can do besides locating the serial ports. But normally you don't need to use them. Setserial may be used in the future to enable super-high speed.

Communication programs include `minicom`, `seyon`, or `wvdial` (for PPP) and `mgetty` for dial-in. Such communication programs require that you configure them, although the default configuration they come with may only need a little tweaking.

Unfortunately the communication program doesn't locate the serial port. This "locating" is the low-level PnP configuring of the serial port: setting its IO address and IRQ in both the hardware and the driver. If you are lucky, this will happen automatically when you boot Linux. Setting these in the hardware was formerly done by jumpers and then running "setserial" but today it's done by "Plug-and-Play" software. You may still need "setserial". So if Linux (or the wvdial program, etc.) doesn't report what serial port your modem is on, then you can try to find it yourself per the next section but it may not be easy.

6. Locating the Serial Port: IO address, IRQs

6.1 What Bus is my Serial Port On?

If you need to find a serial port it often helps if you know what bus it's on. If the serial port is on a card, you may know what bus the card inserts into such as a PCI slot. But if the serial port is built into the motherboard it may not be clear what bus it's on. For old motherboards that have ISA bus slots, it's likely on the ISA bus and may not even be Plug-and-Play. But even if all your slots are PCI, the serial port is likely to be on either an ISA bus or LPC bus (also called a "LPC interface"). LPC is common on laptop computers. Type "lspci" to see if it shows "LPC". Unfortunately, the LPC bus has no standard Plug-and-Play method for low-level configuring devices on it. But according to the specs, the BIOS is supposed to do such configuring (using ACPI ??). To see if you have a LPC bus, type "lspci" and look for a LPC bridge or the like.

6.2 IO & IRQ Overview

For a serial port to work properly it first must be given both an IO address and an IRQ. For old hardware (of mid 1990s), jumpers on a card or a saved BIOS setting does it. For newer hardware the BIOS or Linux must set them at boot-time, and the new hardware doesn't remember how it was set once it's powered down. Enabling the hardware (by using software) gives it both an IRQ and an IO address. Without an IO address, it can't be used. Without an IRQ it will need to use inefficient polling methods for which one must set the IRQ to 0 in the serial driver. Using digital signals sent to the hardware by the BIOS or Linux, it all should get configured automatically (provided the BIOS has not been previously set up to disable it). So you only need to read this if you're having problems or if you want to understand how it works.

The driver must of course know both the IO address and IRQ so that it can talk to the serial port chip. Modern serial port drivers (kernel 2.4) try to determine this by PnP methods so one doesn't normally need to tell the driver (by using "setserial"). A driver may also set an IO address or IRQ in the hardware. But unfortunately, there is some PCI serial port hardware that the driver doesn't recognize so you might need to enable the port yourself. See PCI: Enabling a disabled port

For the old ISA bus, the driver also probes likely serial port addresses to see if there are any serial ports there. This works for the case of jumpers and sometimes works for a ISA PnP port when the driver doesn't do ISA PnP (prior to kernel 2.4).

Locating the serial port by giving it an IRQ and IO address is low-level configuring. It's often automatically done by the serial driver but sometimes you have to do it yourself. What follows repeats what was said above but in more detail.

The low-level configuring consists of assigning an IO address, IRQ, and names (such as ttyS2 = tts/2). This IO-IRQ pair must be set in both the hardware and told to the serial driver. And the driver needs to call this pair a name (such as ttyS2). We could call this "io-irq" configuring for short. The modern way to do this is for the driver to use PnP methods to detect/set the IO/IRQ and then remember what it did. An easy way for a

driver to do this is for the driver to ask the kernel to enable the device and then the kernel tells the driver what IO/IRQ it has used. The old way is using the "setserial" program to tell the driver. For jumpers, there is no PnP but the driver might detect the port if the jumpers are set to the usual I0/IRQ. If you need to configure but don't understand certain details it's easy to get into trouble.

When Linux starts, an effort is made to detect and configure (low-level) the serial ports. Exactly what happens depends on your BIOS, hardware, Linux distribution, kernel version, etc. If the serial ports work OK, there may be no need for you to do any more low-level configuring.

If you're having problems with the serial ports, then you may need to do low-level configuring. If you have kernel 2.2 or lower, then you need to do it if you:

- Plan to use more than 2 ISA serial ports
- Are installing a new serial port (such as an internal modem)
- One or more of your serial ports have non-standard IRQs or IO addresses

Starting with kernel 2.2 you may be able to use more that 2 serial ports without doing any low-level configuring by sharing interrupts. All PCI ports should support this but for ISA it only works for some hardware. It may be just as easy to give each port a unique interrupt if they are available. See Interrupt sharing and Kernels 2.2+

The low-level configuring (setting the IRQ and IO address) seems to cause people more trouble than the high-level stuff, although for many it's fully automatic and there is no configuring to be done. Until the port is enabled and the serial driver knows the correct IRQ and IO address, the port will not usually not work at all.

A port may be disabled, either by the BIOS or by failure of Linux to find and enable the port. For modern ports (provided the BIOS hasn't disabled them) manual PnP tools such as lspci should be able to find them. Applications, and utilities such as "setserial" and "scanport" (Debian only ??) only probe I0 addresses, don't use PnP tools, and thus can't detect disabled ports.

Even if an ISA port can be found by the probing done by the serial driver it may work extremely slow if the IRQ is wrong. See Extremely Slow: Text appears on the screen slowly after long delays. PCI ports are less likely to get the IRQ wrong.

IO address, IRQs, etc. are called "resources" and we are thus configuring certain resources. But there are many other types of "resources" so the term has many other meanings. In summary, the low-level configuring consists of enabling the device, giving it a name (ttyS2 for example) and putting two values (an IRQ number and IO address) into two places:

1. The device driver (done by PnP or "`setserial`")
2. Configuration registers of the serial port hardware itself, done by PnP software (or jumpers on legacy hardware).

You may watch the start-up (= boot-time) messages. They are usually correct. But if you're having problems, your serial port may not show up at all or if you do see a message from "setserial" it may not show the true configuration of the hardware (and it is not necessarily supposed to). See I/O Address & IRQ: Boot-time messages.

6.3 PCI Bus Support

Introduction

Although some PCI modems are "winmodems" without a Linux driver (and will not work under Linux), other PCI modems will often work OK under Linux. If it's a software modem, then you need to find a driver for it since support for "winmodems" is not built into their serial driver. See Linmodem-HOWTO.

If you have kernel 2.4 or better, then there should be support for PnP for both the PCI and ISA buses (either built-in or by modules). Some PCI serial ports can be automatically detected and low-level configured by the serial driver. Others may not be.

While kernel 2.2 supported PCI in general, it had no support for PCI serial ports (although some people got them working anyway). Starting with kernel 2.4, the serial driver will read the id number digitally stored in the serial hardware to determine how to support it (if it knows how). It should assign an I/O address to it, determine its IRQ, etc. So you don't need to use "setserial" for it.

There is a possible problem if you don't use the device filesystem. The driver may assign the port to say tt/ttyS4 per a boot-time message (use `dmesg` to see it). But if you don't have a "file" /dev/ttyS4 then the port will not work. So you will then need to create it, using

```
cd /dev; ./MAKEDEV ttyS4
```

For the device filesystem, the driver should create the device `tts/1`

More info on PCI

PCI ports are not well standardized. Some use main memory for communication with the PC. Some require special enabling of the IRQ. The output of "lspci -vv" can help determine if one can be supported. If you see a 4-digit IO port, the port might work by just telling "setserial" the IO port and the IRQ. Some people have gotten a 3COM 3CP5610 PCI Modem to work that way.For example, if lspci shows IRQ 10, I/O at 0xecb8 and you decide to name it ttyS2 then the command is:

setserial /dev/ttyS2 irq 10 port 0xecb8 autoconfig

Note that the boot-time message "Probing PCI hardware" means reading the PnP configuration registers in the PCI devices which detects info about all PCI cards and on-board PCI devices This is different that the probing of IO addresses by the serial driver which means reading certain IO addresses to see if what's read looks like there's a serial port at that address.

6.4 Common mistakes made re low-level configuring

Here are some common mistakes people make:

- setserial command: They run it (without the "autoconfig" and auto_irq options) and think it has checked the hardware to see if what it shows is correct (it hasn't).
- setserial messages: They see them displayed on the screen at boot-time (or by giving the setserial command) and erroneously think that the result always shows how their hardware is actually configured.
- /proc/interrupts: When their serial device isn't in use they don't see its interrupt there, and erroneously conclude that their serial port can't be found (or doesn't have an interrupt set).

- /proc/ioports and /proc/tty/driver/serial: People think this shows the actual hardware configuration when it only shows about the same info (possibly erroneous) as setserial.

6.5 IRQ & IO Address Must be Correct

There are really two answers to the question "What is my IO and IRQ?" 1. What the device driver thinks has been set (This is what setserial usually sets and shows.). 2. What is actually set in the hardware. Both 1. and 2. above should be the same. If they're not it spells trouble since the driver has incorrect info on the physical serial port. In some cases the hardware is disabled so it has no IO address or IRQ.

If the driver has the wrong IO address it will try to send data to a non-existing serial port --or even worse, to some other device. If it has the wrong IRQ the driver will not get interrupt service requests from the serial port, resulting in a very slow or no response. See Extremely Slow: Text appears on the screen slowly after long delays. If it has the wrong model of UART there is also apt to be trouble. To determine if both I0-IRQ pairs are identical you must find out how they are set in both the driver and the hardware.

6.6 What is the IO Address and IRQ per the driver ?

Introduction

What the driver thinks is not necessarily how the hardware is actually set. If everything works OK then what the driver thinks is likely correct (set in the hardware) and you don't need to investigate (unless you're curious or want to become a guru). Ways to determine what the driver thinks include: boot-time messages I/O Address & IRQ: Boot-time messages, the /proc directory "files" The /proc directory and setserial, and the "setserial" command.

I/O Address & IRQ: Boot-time messages

In many cases your ports will automatically get low-level configured at boot-time (but not always correctly). To see what is happening, look at the start-up messages on the screen. Don't neglect to check the messages from the BIOS before Linux is loaded (no examples shown here). These BIOS messages may be frozen by pressing the Pause key (while holding down shift). It's often tricky to freeze them and you may need to hit Ctrl-Alt-Del while Linux is booting to start rebooting and try again. What these messages display may change as booting progresses and it's often tricky to freeze it at exactly the right words.

Use Shift-PageUp to scroll back to the messages after they have flashed by. Shift-PageDown will scroll in the opposite direction. The `dmesg` command (or looking at logs in /var/log) will show only the first of these two messages. Here's an example of the start-up messages (as of 2004, almost the same as for 1999). Note that in older versions ttyS1 was displayed in these messages as ttyS01, etc.

At first you see what was detected (but the irq is only a wild guess):

```
Serial driver version 4.27 with no serial options enabled
ttyS0 at 0x03f8 (irq = 4) is a 16550A
ttyS1 at 0x02f8 (irq = 3) is a 16550A
ttyS2 at 0x03e8 (irq = 4) is a 16550A
ttyS4 at port 0xeff0 (irq = 10) is a 16550A
```

Note that ttyS0-ttyS2 were detected by probing the standard addresses while ttyS4 is a PCI port detected by probing the PCI configuration. Later setserial shows you what was saved in a configuration file (which you

may edit), but it's not necessarily correct either:

```
Loading the saved-state of the serial devices...
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
/dev/ttyS2 at 0x03e8 (irq = 5) is a 16550A
```

Note that the configuration file only had ttyS1-2 in it so that ttyS0 and ttyS4 were not affected by it. There is also a slight discrepancy: The first message shows ttyS2 at irq=4 while the second shows it at irq=5. In most cases the second message is the correct one. But if you're having trouble, it may be misleading. Before reading the explanation of all of this complexity in the rest of this section, you might just try using your serial port and see if it works OK. If so it may not be essential to read further.

The second message is from the `setserial` program being run at boot-time from a script in the /etc directory tree. It shows what the device driver thinks is the correct configuration. But this too could be wrong. For example, the irq could actually be set to irq=8 in the hardware (both messages wrong). The irq=5 could be there because the configuration file is incorrect.

With old jumper-set serial ports Linux sometimes gets IRQs wrong because it doesn't by default probe for IRQs. It just assumes the "standard" ones (first message) or accepts what is in a configuration file (second message). Neither of these is necessarily correct. If the serial driver has the wrong IRQ, the serial port is very slow or doesn't seem to work at all.

The first message is a result of Linux probing the ISA serial port addresses but it doesn't probe for IRQs. If a port shows up here it exists but the IRQ may be wrong. Linux doesn't check IRQs because doing so is not foolproof. It just assumes the IRQs are as shown because they are the "standard" values. You may check them manually with `setserial` using the `autoconfig` and `auto_irq` options but this isn't guaranteed to be correct either.

The data shown by the BIOS messages (which you see at first before Linux is booted) are what is initially set in the hardware. If your serial port is Plug-and-Play (PnP) then it's possible that "isapnp" or "setpci" will run and change these settings. Look for messages about this after Linux starts. The last serial port message shown in the example above should agree with the BIOS messages (as possibly modified by isapnp or setpci). If they don't agree then you either need to change the setting in the port hardware or use setserial to tell the driver what is actually set in the hardware.

Also, if you have Plug-and-Play (PnP) serial ports, they can only be found by PnP software unless the IRQ and IO has been set inside the hardware by Plug-and-Play software. Prior to kernel 2.4 this was a common reason why the start-up messages did not show a serial port that physically exists. A PnP BIOS may automatically low-level configure them. PnP configuring will be explained later.

The /proc directory and setserial

Type "setserial -g /dev/ttyS*". There are some other ways to find this info by looking at "files" in the /proc directory. Be warned that there is no guarantee that the same is set in the hardware.

`/proc/ioports` will show the IO addresses that the drivers are using. `/proc/interrupts` shows the IRQs that are used by drivers of currently running processes (that have devices open). It shows how many interrupts have actually be issued. `/proc/tty/driver/serial` shows much of the above, plus the number of bytes that have been received and sent (even if the device is not now open).

Note that for the IO addresses and IRQ assignments, you are only seeing what the driver thinks and not necessarily what is actually set in the hardware. The data on the actual number of interrupts issued and bytes processed is real however. If you see a large number of interrupts and/or bytes then it probably means that the device is (or was) working. But the interrupts might be from another device. If there are no bytes received (rx:0) but bytes were transmitted (tx:3749 for example), then only one direction of flow is working (or being utilized).

Sometimes a showing of just a few interrupts doesn't mean that the interrupt is actually being physically generated by any serial port. Thus if you see almost no interrupts for a port that you're trying to use, that interrupt might not be set in the hardware. To view /proc/interrupts to check on a program that you're currently running (such as "minicom") you need to keep the program running while you view it.

6.7 What is the IO Address & IRQ of my Serial Port Hardware?

Introduction

If it's PCI or ISA PnP then what's set in the hardware has been done by PnP methods. Even if nothing has been set or the port disabled, PnP ports may still be found by using "lspci -v" or "isapnp --dumpregs". Ports disabled by jumpers (or hardware failures) are completely lost. See ISA PnP ports, PCI: What IOs and IRQs have been set?, PCI: Enabling a disabled port

PnP ports don't store their configuration in the hardware when the power is turned off. This is in contrast to Jumpers (non-PnP) which remain the same with the power off. That's why a PnP port is more likely to be found in a disabled state than an old non-PnP one.

PCI: What IOs and IRQs have been set?

For PCI, the BIOS almost always sets the IRQ and may set the IO address as well. To see how it's set use "lspci -vv" (best) or look in /proc/bus/pci (or for kernels <2.2 /proc/pci). The modem's serial port is often called a "Communication controller". Look for this. If lspci shows "I/O ports at ... [disabled]" then the serial port is disabled and the hardware has no IO address so it's lost and can't be used. See PCI: Enabling a disabled port for how to enable it.

If more than one IO address is shown, the first one is more likely to be it. You can't change the IRQ (at least not with "setpci") This is because if one writes an IRQ it it's hardware register no action is taken on it. It's the BIOS that should actually set up the IRQs and then write the correct value to this register for lspci to view. If you must, change the IO address with "setpci" by changing the BASE_ADDRESS_0 or the like.

PCI: Enabling a disabled port

If the port communicates via an IO address then "lspci -vv" should show "Control: I/O+ ..." with + meaning that the IO address is enabled. If it shows "I/O-" (and "I/O ports at ... [disabled]") then you may need to use the setpci command to enable it. For example "setpci -d 151f:000 command=101". 151f is the vendor id, and 000 is the device id both obtained from "lspci -n -v" or from /proc/bus/pci or from "scanpci -v". The "command=101" means that 101 is put into the command register which is the same as the "Control" register displayed by "lspci". The 101h sets two bits: the 1 sets I/O to + and the 100 part keeps SERR# set to +. In this case only the SERR# bit of the Control register was initially observed to be + when the lspci command was run. So we kept it enabled to + by setting bit 8 (where bit 0 is I/O) to 1 by the first 1 in 101. Some serial cards

don't use SERR# so if you see SERR#- then there's no need to enable it so then use: command=1. Then you'll need to set up "setserial" to tell the driver the IO and IRQ.

Bit 8 is actually the 9th bit since we started counting bits from 0. Don't be alarmed that lspci shows a lot of - signs showing that the card doesn't have many features available (or enabled). Serial ports are relatively slow and don't need these features.

Another way to enable it is to let the BIOS do it by telling the BIOS that you don't have a plug-and-play operating system. Then the BIOS should enable it when you start your PC. If you have MS Windows9x on the same PC then doing this might cause problems with Windows (see Plug-and-Play-HOWTO).

ISA PnP ports

For an ISA Plug-and-Play (PnP) port one may try the `pnpdump` program (part of `isapnptools`). If you use the --dumpregs option then it should tell you the actual IO address and IRQ set in the port. It should also find an ISA PnP port that is disabled. The address it "trys" is not the device's IO address, but a special address used for communicating with PnP cards.

Finding a port that is not disabled (ISA, PCI, PnP, non-PnP)

Perhaps the BIOS messages will tell you some info before Linux starts booting. Use the shift-PageUp key to step back thru the boot-time messages and look at the very first ones which are from the BIOS. This is how it was before Linux started. Setserial can't change it but isapnp or setpci can. Starting with kernel 2.4, the serial driver can make such changes for many (but not all) serial ports.

Using "scanport" (Debian only ??) will probe all I/O ports and will indicate what it thinks may be serial port. After this you could try probing with setserial using the "autoconfig" option. You'll need to guess the addresses to probe at (using clues from "scanport"). See What is Setserial.

For a port set with jumpers, the IO ports and IRQs are set per the jumpers. If the port is not Plug-and-Play (PnP) but has been setup by using a DOS program, then it's set at whatever the person who ran that program set it to.

Exploring via MS Windows (a last resort)

For PnP ports, checking on how it's configured under DOS/Windows may (or may not) imply how it's under Linux. MS Windows stores its configuration info in its Registry which is not used by Linux so they are not necessarily configured the same. If you let a PnP BIOS automatically do the configuring when you start Linux (and have told the BIOS that you don't have a PnP operating system when starting Linux) then Linux should use whatever configuration is in the BIOS's non-volatile memory. Windows also makes use of the same non-volatile memory but doesn't necessarily configure it that way.

6.8 Choosing Serial IRQs

If you have Plug-and-Play ports then either a PnP BIOS or a serial driver may configure all your devices for you so then you may not need to choose any IRQs. PnP software determines what it thinks is best and assigns them (but it's not always best). But if you directly use isapnp (ISA bus) or jumpers then you have to choose. If you already know what IRQ you want to use you could skip this section except that you may want to know that IRQ 0 has a special use (see the following paragraph).

IRQ 0 is not an IRQ

While IRQ 0 is actually the timer (in hardware) it has a special meaning for setting a serial port with setserial. It tells the driver that there is no interrupt for the port and the driver then will use polling methods. Such polling puts more load on the CPU but can be tried if there is an interrupt conflict or mis-set interrupt. The advantage of assigning IRQ 0 is that you don't need to know what interrupt is set in the hardware. It should be used only as a temporary expedient until you are able to find a real interrupt to use.

Interrupt sharing, Kernels 2.2+

Sharing of IRQs is where two devices use the same IRQ. As a general rule, this wasn't allowed for the ISA bus. The PCI bus may share IRQs but one can't share the same IRQ between the ISA and the PCI bus. Most multi-port boards may share IRQs. Sharing is not as efficient since every time a shared interrupt is given a check must be made to determine where it came from. Thus if it's feasible, it's nicer to allocate every device its own interrupt.

Prior to kernel 2.2, serial IRQs could not be shared with each other except for most multiport boards. Starting with kernel 2.2 serial IRQs may be sometimes shared between serial ports. In order for sharing to work in 2.2 the kernel must have been compiled with CONFIG_SERIAL_SHARE_IRQ, and the serial port hardware must support sharing (so that if two serial cards put different voltages on the same interrupt wire, only the voltage that means "this is an interrupt" will prevail). Since the PCI bus specs permit sharing, any PCI card should allow sharing.

What IRQs to choose?

The serial hardware often has only a limited number of IRQs. Also you don't want IRQ conflicts. So there may not be much of a choice. Your PC may normally come with `ttyS0` and `ttyS2` at IRQ 4, and `ttyS1` and `ttyS3` at IRQ 3. Looking at `/proc/interrupts` will show which IRQs are being used by programs currently running. You likely don't want to use one of these. Before IRQ 5 was used for sound cards, it was often used for a serial port.

Here is how Greg (original author of Serial-HOWTO) set his up in /etc/rc.d/rc.serial. rc.serial is a file (shell script) which runs at start-up (it may have a different name or location). For versions of "setserial" after 2.15 it's not always done this way anymore but this example does show the choice of IRQs.

```
/sbin/setserial /dev/ttyS0 irq 3        # my serial mouse
/sbin/setserial /dev/ttyS1 irq 4        # my Wyse dumb terminal
/sbin/setserial /dev/ttyS2 irq 5        # my Zoom modem
/sbin/setserial /dev/ttyS3 irq 9        # my USR modem
```

Standard IRQ assignments:

```
 IRQ  0    Timer channel 0 (May mean "no interrupt".  See below.)
 IRQ  1    Keyboard
 IRQ  2    Cascade for controller 2
 IRQ  3    Serial port 2
 IRQ  4    Serial port 1
 IRQ  5    Parallel port 2, Sound card
 IRQ  6    Floppy diskette
 IRQ  7    Parallel port 1
 IRQ  8    Real-time clock
 IRQ  9    Redirected to IRQ2
 IRQ 10    not assigned
```

```
IRQ 11    not assigned
IRQ 12    not assigned
IRQ 13    Math co-processor
IRQ 14    Hard disk controller 1
IRQ 15    Hard disk controller 2
```

There is really no Right Thing to do when choosing interrupts. Try to find one that isn't being used by the motherboard, or any other boards. 2, 3, 4, 5, 7, 10, 11, 12 or 15 are possible choices. Note that IRQ 2 is the same as IRQ 9. You can call it either 2 or 9, the serial driver is very understanding. If you have a very old serial board it may not be able to use IRQs 8 and above.

Make sure you don't use IRQs 1, 6, 8, 13 or 14! These are used by your motherboard. You will make her very unhappy by taking her IRQs. When you are done you might want to double-check `/proc/interrupts` when programs that use interrupts are being run and make sure there are no conflicts.

6.9 Choosing Addresses --Video card conflict with ttyS3

Here's a problem with some old serial cards. The IO address of the IBM 8514 video board (and others like it) is allegedly 0x?2e8 where ? is 2, 4, 8, or 9. This may conflict with the IO address of `ttyS3` at 0x02e8. Your may think that this shouldn't happen since the addresses are different in the high order digit (the leading 0 in 02e8). You're right, but a poorly designed serial port may ignore the high order digit and respond to any address that ends in 2e8. That is bad news if you try to use `ttyS3` (ISA bus) at this IO address.

For the ISA bus you should try to use the default addresses shown below. PCI cards use different addresses so as not to conflict with ISA addresses. The addresses shown below represent the first address of an 8-byte range. For example 3f8 is really the range 3f8-3ff. Each serial device (as well as other types of devices that use IO addresses) needs its own unique address range. There should be no overlaps (conflicts). Here are the default addresses for commonly used serial ports on the ISA bus:

```
ttyS0 address 0x3f8
ttyS1 address 0x2f8
ttyS2 address 0x3e8
ttyS3 address 0x2e8
```

Suppose there is an address conflict (as reported by `setserial -g /dev/ttyS*`) between a real serial port and another port which does not physically exist (and shows UART: unknown). Such a conflict shouldn't cause problems but it sometimes does in older kernels. To avoid this problem don't permit such address conflicts or delete /dev/ttySx if it doesn't physically exist.

6.10 Set IO Address & IRQ in the hardware (mostly for PnP)

After it's set in the hardware don't forget to insure that it also gets set in the driver by using `setserial`. For non-PnP serial ports they are either set in hardware by jumpers or by running a DOS program ("jumperless") to set them (it may disable PnP). The rest of this subsection is only for PnP serial ports. Here's a list of the possible methods of configuring PnP serial ports:

- Using a PnP BIOS CMOS setup menu (usually only for external modems on ttyS0 (Com1) and ttyS1 (Com2))
- Letting a PnP BIOS automatically configure a PnP serial port See Using a PnP BIOS to I0-IRQ Configure
- Doing nothing if the serial driver recognized your card OK

- Using `isapnp` for a PnP serial port (non-PCI)
- Using setpci (pciutils or pcitools) for the PCI bus

The IO address and IRQ must be set (by PnP) in their registers each time the system is powered on since PnP hardware doesn't remember how it was set when the power is shut off. A simple way to do this is to let a PnP BIOS know that you don't have a PnP OS and the BIOS will automatically do this each time you start. This might cause problems in Windows (which is a PnP OS) if you start Windows with the BIOS thinking that Windows is not a PnP OS. See Plug-and-Play-HOWTO.

Plug-and-Play (PnP) was designed to automate this io-irq configuring, but for Linux it initially made life much more complicated. In modern Linux (2.4 kernels --partially in 2.2 kernels), each device driver has to do its own PnP (using supplied software which it may utilize). There is unfortunately no centralized planning for assigning IO addresses and IRQs as there is in MS Windows. But it usually works out OK in Linux anyway.

Using a PnP BIOS to I0-IRQ Configure

While the explanation of how to use setpci or isapnp for io-irq configuring should come with such software, this is not the case if you want to let a PnP BIOS do such configuring. Not all PnP BIOS can do this. The BIOS usually has a CMOS menu for setting up the first two serial ports. This menu may be hard to find. For an "Award" BIOS it was found under "chipset features setup" There is often little to choose from. For ISA serial ports, the first two ports normally get set at the standard IO addresses and IRQs. See More on Serial Port Names

Whether you like it or not, when you start up a PC, a PnP BIOS starts to do PnP (io-irq) configuring of hardware devices. It may do the job partially and turn the rest over to a PnP OS (which Linux is in some sense) or if thinks you don't have a PnP OS it may fully configure all the PnP devices but not configure the device drivers.

If you tell the BIOS that you don't have a PnP OS, then the PnP BIOS should do the configuring of all PnP serial ports --not just the first two. An indirect way to control what the BIOS does (if you have Windows 9x on the same PC) is to "force" a configuration under Windows. See Plug-and-Play-HOWTO and search for "forced". It's easier to use the CMOS BIOS menu which may override what you "forced" under Windows. There could be a BIOS option that can set or disable this "override" capability.

If you add a new PnP device, the BIOS should PnP configure it. It could even change the io-irq of existing devices if required to avoid any conflicts. For this purpose, it keeps a list of non-PnP devices provided that you have told the BIOS how these non-PnP devices are io-irq configured. One way to tell the BIOS this is by running a program called ICU under DOS/Windows.

But how do you find out what the BIOS has done so that you set up the device drivers with this info? The BIOS itself may provide some info, either in its setup menus of via messages on the screen when you turn on your computer. See What is set in my serial port hardware?. Other ways of finding out is to use lspci for the PCI bus or isapnp --dumpregs for the ISA bus. The cryptic results it shows you may not be clear to a novice.

6.11 Giving the IRQ and IO Address to Setserial

Once you've set the IRQ and IO address in the hardware (or arranged for it to be done by PnP) you also need to insure that the "setserial" command is run each time you start Linux. See the subsection Boot-time Configuration

7. Configuring the Serial Driver (high-level) "stty"

7.1 Introduction

This configuring is normally done by your communications program such as wvdial. It may do much of it without even letting you know what it's done. In olden days it was done with the "stty" utility. If you set something manually with stty, the communications program may change the setting to something else so it's usually best to just let the communications program handle it. See What is stty ?

7.2 Hardware flow control (RTS/CTS)

See Flow Control for an explanation of it. You should always use hardware flow control if possible. Your communication program or "`getty`" should have an option for setting it (and hopefully it's enabled by default). It needs to be set both inside your modem (by an init string or default) and in the device driver. Your communication program should set both of these (if you configure it right).

If none of the above will fully enable hardware flow control. Then you must do it yourself. For the modem, make sure that it's either done by the init string or is on by default. If you need to tell the device driver to do it is best done on startup by putting it in a file that runs at boot-time. See the subsection Boot-time Configuration You need to add the following to such a file for each serial port (example is ttyS2) you want to enable hardware flow control on:

```
stty -F /dev/ttyS2 crtscts
or
stty crtscts < /dev/ttyS2
```

If you want to see if flow control is enabled do the following: In minicom (or the like) type AT&V (or ATI4 on 3Com modems) to see how the modem is configured and look for &K3 (or &H1 on 3Com modems) which means hardware flow control. Then without exiting the communications program (such as minicom) see if the device driver knows about it by typing: stty -F /dev/ttyS2 -a. Look for "crtscts" (without a disabling minus sign). Remember that communication programs change these settings so you may want to check them after you have started up your communication program.

7.3 Speed Settings

Besides flow control there is speed. See What Speed Should I Use with My Modem. There's also are parity and bits-per-byte settings. Normally the port is set by the communications program at 8N1 (8-bits per byte, No parity, and 1 stop bit). If you're running PPP then you must use 8N1. So if you get a complaint that it's not 8-bit clean then it's likely not 8N1 like it should be.

7.4 Ignore CD Setting: clocal

Normally a CD (Carrier Detect) signal (on the CD wire for an external modem) is required before a serial port can be opened. But if stty has negated clocal (-clocal), then the port requires CD raised for the port to open and remain open. Actually, a skilled programmer can write the program in such a way as to force the port to open even when CD and clocal say not to. So if stty shows -clocal then there might be a problem with opening the port. But for dial-in, in some cases you may want -clocal so that when the remote modem stops sending a carrier and CD drops, the port will close and terminate all processes running on it.

One way to keep CD raised is to send "AT&C" to the modem so that CD from the modem will always be on. CD always-on is fine for dial-out but for dial-in, the CD signal is sometimes (but rarely) used to detected an incoming call.

clocal may be asserted by default in recent serial drivers. Minicom raises clocal automatically when it starts up so there is no problem with it opening the port. But it restores the clocal setting to it's original when you exit minicom. But version 6.0.192 of Kermit hung when I set -clocal and tried to "set line ...".

7.5 What is stty ?

`stty` is something like setserial but it sets the speed (baud rate), hardware flow control, and other parameters of a serial port. Typing "stty -F /dev/ttyS2 -a" should show you how ttyS2 is configured. Most of the stty settings are for things that you never need to use with modems. Many of the setting are only needed for Text-Terminals (and some are only needed for antique terminals of the 1970s). Your communication package should automatically set up the several settings needed for modems. For this reason you normally don't need to use stty so it's not covered much in this Modem-HOWTO. But stty is sometimes useful for trouble-shooting. More is said about stty in the Serial-HOWTO or Text-Terminal-HOWTO..

8. Modem Configuration (excluding serial port)

8.1 Finding Your Modem

Before spending a lot of time deciding how to configure your modem, you first need to make sure it can be found and that AT-commands and the like can be sent to it. So I suggest you first give it a very simple configuration using the communication program you will be using on the port and see if it works. If this works you may then want to improve on the configuration, If not then see My Modem is Physically There but Can't be Found. A winmodem may be hard to find and will not work under Linux.

8.2 AT Commands

While the serial port on which a modem resides requires configuring, so does the modem itself. The modem is configured by sending AT commands (or the like) to it on the same serial line that is used to send data.

Most modems use an AT command set. These are cryptic and short ASCII commands where all command strings must be prefaced by the letters AT. AT means: ATtention, expect a command to follow. For example: ATZ&K3<return> This is an AT-command string with two commands here: Z and &K3. Z is short for Z0 and a few modems require that you use Z0 instead of just Z. It's like this for other commands ending in 0. The command string is terminated by a return character (use the <enter> key if you are manually typing it). A string that's too long (40 or more characters) may not work on older modems. You may use either uppercase or lowercase letters.

Unfortunately there are many different variations of the AT command set so that what works for one modem may or may not work for another modem. Thus there is no guarantee that the AT commands given in this section will work on your modem.

Such command strings are either automatically sent to the modem by communication programs or are manually typed in by you. Most communication programs provide a screen where you may change (edit) and save the init string that the communication program will use. The modem itself has a stored configuration (profile) which is like a long init string. It represents the configuration of the modem when it's first tuned on.

You may change it to suit your taste. In most cases there are a few different such configurations (profiles) and there are ways to designate one of them to be active.

If you have a manual for your modem (either on paper or on floppy disk) you might find AT-commands there. 3Com modems (and others ??) have AT-Command help files built into the modem so if you type say "AT$" to the modem it will display some "online help".

You can also find info on AT commands on the Internet. You should first try a site for your modem manufacturer. If this doesn't work out then you can search the Internet using terms that are from AT commands such as &C1, &D3, etc. This will tend to find sites that actually list AT-Commands instead of sites that just talk about them in general. You might also try a few of the sites listed in the subsection Web Sites. Be warned that the AT-commands for a different brand of modem may be somewhat different.

8.3 Init Strings: Saving and Recalling

The examples given in this subsection are from the Hayes AT modem command set. All command strings must be prefaced by the two letters AT. For example: AT&C1&D3^M (^M is the return character). When a modem is powered on, it automatically configures itself with one of the configurations it has stored in its non-volatile memory. If this configuration is satisfactory there is nothing further to do.

If it's not satisfactory, then one may either alter the stored configuration or configure the modem each time you use it by sending it a string of commands known as an "init string" (= initialization string). Normally, a communication program does this. What it sends will depend on how you configured the communications program. Your communication program should allow you to edit the init string and change it to whatever you want. Sometimes the communications program will let you select the model of your modem and then it will use an init string that it thinks is best for that modem.

The configuration of the modem when it's first powered on may be expressed by an init string. You might think of this as the default "string" (called a profile). If your communications program sends the modem another string (the init string), then this string will modify the default configuration. For example, if the init string only contains two commands, then only those two items will be changed. However, some commands will recall a stored profile from inside the modem so a single such command in the init string can thereby change everything in the configuration.

Modern modems have a few different stored profiles to choose from that are stored in the modem's non-volatile memory (it's still there when you turn it off). In my modem there are two factory profiles (0 and 1, neither of which you can change) and two user defined profiles (0 and 1) that the user may set and store. Your modem may have more. To view some of these profiles send the command &V. At power-up one of the user-defined profiles is loaded. For example, if you type the command &Y0 (just Y0 for a 3Com modem) then in the future profile 0 will be used at power-on.

There are also commands to load (activate) any of the stored profiles. Such a load command may be put in an init string. Of course if it loads the same profile that was automatically loaded at power-up, nothing is changed (unless the active profile has been modified since power-up). Since your profile could have thus been modified it's a good idea to use some kind of an init string even if it does nothing more than load a stored profile.

Examples of loading saved profiles:
Z0 loads user-defined profile 0 and resets (hangs up, etc.)
&F1 loads factory profile 1

Once you have sent commands to the modem to configure it the way you want (such as loading a factory profile and modifying it a little) you may save this as a user-defined profile:
&W0 saves the current configuration to user-profile 0.

Many people don't bother saving a good configuration in their modem, but instead, send the modem a longer init string each time the modem is used. Another method is to restore the factory default by &F1 at the start of the init string and then modify it a little by adding a few other commands to the end of the init string. Since there is no way to modify the factory default this prevents anyone from changing the configuration by modifying (and saving) the user-defined profile.

You may choose an init string supplied by someone else that they think is right for your modem. Some communication programs have a library of init strings to select from. The most difficult method (and one which will teach you the most about modems) is to study the modem manual and write one yourself. You could save this configuration inside the modem so that you don't need an init string. A third alternative is to start with an init string that someone else wrote, but modify it to suit your purposes.

If you look at init strings used by communication programs you may see symbols which are not valid modem commands. These symbols are commands to the communication program itself and will not be sent to the modem. For example, ~ may mean to pause briefly.

Where is my "init string" so I can modify it ?

This depends on your communication program (often a PPP program). If this is the latest version of Modem-HOWTO send me info for other cases.

- Gnome: run pppsetup
- wvdial: edit /etc/wvdial.conf
- minicom: hit ^Ao (or possibly ALT-o), then select "Modem and Dialing"

8.4 Other AT Modem Commands

For dial-in see Dial-in Modem Configuration. The rest of this section is mostly what was in the old Serial-HOWTO. All strings must start with AT. Here's a few Hayes AT codes that should be in the string (if they are not set by using a factory default or by a saved configuration).

- E1 command echo ON
- Q0 result codes are reported
- V1 result codes are verbose
- S0=0 never answer (uugetty does this with the WAITFOR option)

Here's some more AT commands for special purposes:

- &C1 CD is only on when you're connected
- &S0 DSR is always on
- X3 Dial even if there is no dialtone (Blind dial. Use where dial-tones don't exist).

Note: to get his old USR Courier V.34 modem to reset correctly when DTR drops, Greg Hankins had to set `&D2` and `S13=1` (this sets bit 0 of register S13). This has been confirmed to work on USR Sportster V.34 modems as well.

Note: some old Supra modems treat CD differently than other modems. If you are using a Supra, try setting `&C0` and *not* `&C1`. You must also set `&D2` to handle DTR correctly.

8.5 Blacklisting

If phone number is dialed a few times with no success, some modems may blacklist a phone number. After a certain time you may try again. Some countries require this to reduce needless repeated dialing. To view the blacklist try %B. To delete the blacklist use these AT commands:

- SR Robotics o 3COM: s40=2 or if NG try s40=7
- Lucent: %t21,18,0
- Rockwell: %tcb
- Cirrus Logic: *nc9

8.6 What AT Commands are Now Set in my Modem?

You may try to use minicom for viewing your modem profile. It's best not to have any other process running on the modem port when you do this. If you have set up minicom for your modem, then you may type on the command line: `minicom -o` to start minicom without restoring the saved modem profile. Then type `at&v` (or atI4 on 3Com modems) to display the profile. To exit minicom without disturbing this profile, use the q (quit) command for exiting without resetting.

The above may not work for various reasons. If the modem has been set not to echo result codes it may not even display any profile. If there is another process running on the modem port at the same time, some of what the modem sends to you is likely to be read by the other process so you will see only part of the profile. Is there some way to temporarily stop the other process on the port so it will not interfere? I tried the "stop" signal using the "kill" command but it didn't work. If this is the latest version of this HOWTO, let me know if you find a way to do it.

If you have at least one process running on the modem port and kill them, the modem's profile may be reset so you will not observe what the original profile was. This will happen if you kill getty (or it's replacements: login or bash) and have &D3 set. The killing of getty (or the like) will drop DTR and reset the modem's profile to the power-on state. To keep getty from respawning when killed, comment it out in /etc/inittab and do an "init q".

8.7 Modem States (or Modes)

Since the channel for sending AT commands to the modem is the same channel that is used for the flow of data (files, packets, etc.) then it's important to cleanly separate the AT commands from the data.

When the modem is first turned on it's in the command mode (also called terminal mode, idle state or AT-command mode). Anything sent to it from the PC is assumed to be an AT command and not data. Then if a dial command is sent to it (ATD...), it dials and connects to another modem. It's now in the on-line data mode (connected) and sends and receives data (such as Internet pages). In this mode, any AT command one trys to send it will not work but will be transmitted to the other modem instead. Except for the escape command. This is +++ with a minimum time delay both at the start and end. The time delay allows the modem to determine that it is likely a real escape and not just +++ in a file being transmitted.

So we have two states so far: AT-command and on-line data. But there is a third important state which is sort of a combination of these two. It's the on-line command mode. This is when the modem maintains a connection (without sending/receiving data) but anything sent from the PC is interpreted as an AT command. This is the state reached with a +++ escape signal or by a DTR drop from the PC provided the &D1 has been set. Then one can send AT commands to the modem including commands which will leave this state and go to one of the other two states.

There are other states also: dialing state and handshaking state but they normally lead to the connected (on-line) state. If they don't then the modem should hang up, thereby returning to the initial AT-command (or idle) state.

9. Serial Port Devices /dev/ttyS4, (or /dev/ttys/4) etc.

9.1 Serial Port Names: ttyS4, etc

Once upon a time the names of the serial ports were simple. Except for some multiport serial cards they were named /dev/ttyS0, /dev/ttyS1, etc. Then around the year 2000 came the USB bus with names like /dev/ttyUSB0 and /dev/ttyACM1 (for the ACM modem on the USB bus).

9.2 The PCI Bus

Since DOS provided for 4 serial ports on the old ISA bus: COM1-COM4, ttyS0-ttyS3 most serial ports on the newer PCI bus used higher numbers such as ttyS4 or ttyS14 (prior to kernel 2.6.13). But since most PCs only came with one or two serial ports, ttyS0 and possibly ttyS1 (for the second port) the PCI bus now may use ttyS2 (kernel 2.6.15 on). All this permits one to have both ISA serial ports and PCI serial ports on the same PC with no name conflicts. 0-1 (or 0-3) are reserved for the old ISA bus (or the newer LPC bus) and 2-upward (or 4-upward or 14-upward) are used for PCI. It's not required to be this way but it often is. On-board serial ports on motherboards which have both PCI and ISA slots are likely to still be ISA ports. Even for all-PCI-slot motherboards, the serial ports are often not PCI. Instead, they are either ISA, on an internal ISA bus or on a LPC bus which is intended for slow legacy I/O devices: serial/parallel ports and floppy drives.

9.3 Serial Port Device Names & Numbers

Devices in Linux have major and minor numbers. The serial port ttySx (x=0,1,2, etc.) is major number 4. You can see this (and the minor numbers too) by typing: "ls -l ttyS*" in the /dev directory. To find the device names for various devices, see the "devices" file in the kernel documentation.

There formerly was a "cua" name for each serial port and it behaved just a little differently. For example, ttyS2 would correspond to cua2. It was mainly used for modems. The cua major number was 5 and minor numbers started at 64. You may still have the cua devices in your /dev directory but they are now deprecated. For details see Modem-HOWTO, section: cua Device Obsolete.

For creating the old devices in the device directory see: the Serial-HOWTO: "Creating Devices In the /dev directory".

9.4 More on Serial Port Names

Dos/Windows use the COM name while the messages from the serial driver use ttyS00, ttyS01, etc. Older serial drivers (2001 ?) used just tty00, tty01, etc.

The tables below shows some examples of serial device names. The IO addresses are the default addresses for the old ISA bus (not for the newer PCI and USB buses).

```
dos     common                    IO       USB-BUS  ( ACM => acm modem )
name     name     major minor address ||  common name      common name
COM1   /dev/ttyS0   4,   64;    3F8      ||  /dev/ttyUSB0   |   /dev/ttyACM0
COM2   /dev/ttyS1   4,   65;    2F8      ||  /dev/ttyUSB1   |   /dev/ttyACM1
COM3   /dev/ttyS2   4,   66;    3E8      ||  /dev/ttyUSB2   |   /dev/ttyACM2
COM4   /dev/ttyS3   4,   67;    2E8      ||  /dev/ttyUSB3   |   /dev/ttyACM3
 -     /dev/ttyS4   4,   68;    various
```

9.5 USB (Universal Serial Bus) Serial Ports

For more info see the usb subdirectory in the kernel documentation directory for files: usb-serial, acm, etc.

9.6 Link ttySN to /dev/modem

On some installations, two extra devices will be created, `/dev/modem` for your modem and `/dev/mouse` for a mouse. Both of these are symbolic links to the appropriate device in `/dev`.

Historical note: Formerly (in the 1990s) the use of `/dev/modem` (as a link to the modem's serial port) was discouraged since lock files might not realize that it was really say `/dev/ttyS2`. The newer lock file system doesn't fall into this trap so it's now OK to use such links.

9.7 Devfs (The Improved but Obsolete Device File System)

Kernel 2.4 introduced the now obsolete optional "device file system" (devfs) with a whole new set of names for everything. But in 2003-4, it was claimed that devfs had unsolvable problems and starting with kernel 2.6.12 it was replaced with "udev" (kernels prior to 2.6.12 also could use udev but with some problems). Although udev doesn't provide all the functionality of devfs, it does handle hot plugging. Also, the use of udev isn't required to run Linux so some people don't use it. But many distributions install it by default.

Devfs was a good idea and was claimed to be more efficient than udev. But unfortunately, the author of devfs didn't maintain it for long and it allegedly became not too well maintained. So for better or worse we now have udev instead although the debate of devfs vs. udev still continues. For a detailed description of devfs see: http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html Also see the kernel documentation tree: filesystems/devfs.

The names of devices for the devfs can be used in udev, but usually are not and may not be simple to activate. Here's the devfs names for serial devices: ttyS1 becomes tts/1, ttyUSB1 becomes /usb/tts/1, and ttyACM1 is /usb/acm/1. Note that the number 1 above is just an example. It could be replaced by 0, 2, 3, 4, etc. Some more examples of udev names: ttyS2 becomes tts/2 (Serial port), tty3 becomes vc/3 (Virtual Console), ptyp1 becomes pty/m1 (PTY master), ttyp2 becomes pty/s2 (PTY slave). "tts" looks like a directory which contains devices "files": 0, 1, 2, etc. All of these new names should still be in the /dev directory although optionally one may put them elsewhere.

For devfs device names in the /dev directory are created automatically by the corresponding driver. Thus, if

serial support comes from a module and that module isn't loaded yet, there will not be any serial devices in the /dev directory. This can be confusing: you physically have serial ports but don't see them in the /dev directory. However, if a device name is told to a communication program and the serial module isn't loaded, the kernel is supposed to try to find a driver for it and create a name for it in the /dev directory.

This works OK if it finds a driver. But suppose there is no driver found for it. For example, if you try to use "setserial" to configure a port that the driver failed to detect, it claims there is no such port. How does one create a devfs port in this case?

For multiport devices for example, /dev/ttyF9 becomes /dev/ttf/9, or in a later version /dev/tts/F9. Substitute for F (or f) whatever letter(s) your multiport board uses for this purpose. A multiport driver is supposed to create a devfs name similar to the above and put it into the /dev directory

9.8 cua Device Obsolete

Each ttyS device has a corresponding cua device. But the cua device is deprecated so it's best to use ttyS (unless cua is required). There is a difference between cua and ttyS but a savvy programmer can make a ttyS port behave just like a cua port so there is no real need for the cua anymore. Except that some older programs may need to use the cua.

What's the difference? The main difference between cua and ttyS has to do with what happens in a C-program when an ordinary "open" command tries to open the port. If a cua port has been set to check modem control signals, the port can be opened even if the CD modem control signal says not to. Astute programming (by adding additional lines to the program) can force a ttyS port to behave this way also. But a cua port can be more easily programmed to open for dialing out on a modem even when the modem fails to raise CD (since no one has called into it and there's no carrier). That's why cua was once used for dial-out and ttyS used for dial-in.

Starting with Linux kernel 2.2, a warning message is put in the kernel log when one uses cua. This is an omen that cua is defunct and should be avoided if possible.

10. Interesting Programs You Should Know About

10.1 What is setserial ?

This part is in 3 HOWTOs: Modem, Serial, and Text-Terminal. There are some minor differences, depending on which HOWTO it appears in.

Setserial problems with linmodems, laptops

The setserial program doesn't seem to work if the serial port is for a linmodem such as ttySHCF0. If you have a Laptop (PCMCIA) don't use `setserial` until you read Laptops: PCMCIA.

Introduction

`setserial` is a program used for the user to communicate with the serial device driver. You normally never need to use it, provided that you only use the one or two serial ports that come as standard equipment with a PC. Even in other cases, most extra serial ports should be auto-detected by modern kernels. Except you'll need to use setserial if you have an old ISA serial port set by jumpers on the physical hardware or if your kernel

(such as 2.2 or older) doesn't both detect and set your add-on PCI serial ports.

`setserial` allows you (or a shell script) to talk to the serial software. But there's also another program, tt/stty/, that also deals with the serial port and is used for setting the port speed, etc.

`setserial` deals with the lower-level configuring of the serial port, such as dealing with IRQs (such as 5), port addresses (such as 3f8), and the like. A major problem with it is that it can't set or configure the serial port hardware: It can't set the IRQ or port addresses into the hardware. Furthermore, when it seemingly reports the configuration of the hardware, it's sometimes wrong since it doesn't actually probe the hardware unless you specifically tell it to. Even then, it doesn't do the modern type of bus probing and some hardware may never be found by it. Still, what it shows is right most all the time but if you're having trouble getting a serial port to work, then there's a fair chance it's wrong.

In olden days, when the IRQ and port address was set by jumpers on the serial card, one would use `setserial` to tell the driver how these jumpers were set. Today, when plug-and-play methods detect how the jumperless serial port is set, `setserial` is not really needed anymore unless you're having problems or using old hardware. Furthermore, if the configuration file used by `setserial` is wrong, then there's trouble. In this case, if you use `setserial` to try to find out how the port is configured, it may just repeat the incorrect information in the configuration file.

`setserial` can sometimes be of help to find a serial port. But it's only of use if you know the port address and use the right options. For modern ports, there's usually better ways to look for them by plug-and-play methods.

Thus the name `setserial` is somewhat of a misnomer since it doesn't set the I/O address nor IRQ in the hardware, it just "sets" them in the driver software. And the driver naively believes that what `setserial` tells it, even if it conflicts with what the driver has found by using plug-and-play methods. Too bad that it fails to at least issue a warning message for such a conflict. Since the device driver is considered to be part of the kernel, the word "kernel" is often used in other documentation with no mention made of any "serial driver".

Some distributions (and versions) set things up so that `setserial` is run at boot-time by an initialization shell script (in the /etc directory tree). But the configuration file which this script uses may be either in the /etc tree or the /var tree. In some cases, if you want `setserial` to run at boot-time, you may have to take some action. `setserial`will not work without either serial support built into the kernel or loaded as a module. The module may get loaded automatically if you (or a script) attempt to use `setserial`.

While `setserial` can be made to probe the hardware I0 port addresses to try to determine the UART type and IRQ, this has severe limitations. See Probing. It can't set the IRQ or the port address in the hardware of PnP or PCI serial ports (but the plug-and-play features of the serial driver may do this). It also can't directly read the PnP data stored in configuration registers in the hardware. But since the device driver can read these registers and setserial tells you what the device driver thinks, it might be correct. Or it could be telling you what `setserial` had previously (and perhaps erroneously) told the driver. There's no way to know for sure without doing some other checks.

The serial driver (for Linux Kernel 2.4+) looks for a few "standard" legacy serial ports, for PnP ports on the ISA bus, and for all supported port hardware on the PCI bus. If it finds your ports correctly, then there's no need to use `setserial`. The driver doesn't probe for legacy IRQs and may get these wrong and it may miss old ISA serial ports set with jumpers on the card.

Besides the man page for `setserial`, check out info in `/usr/doc/setserial.../` or `/usr/share/doc/setserial`. This should tell you how setserial is handled for your distribution of Linux.

While `setserial` behaves the same in all distributions, the scripts for running it, how to configure such scripts (including automatic configuration), and the names and locations of the script files, etc., are all distribution-dependent.

Serial module unload

If a serial module gets unloaded, the changes previously made by `setserial` will be forgotten by the driver. But while the driver forgets it, a script provided by the distribution may save it in a file somewhere so that it can the restored if the module is reloaded.

Giving the `setserial` command

Remember, that `setserial` can't set any I/O addresses or IRQs in the hardware. That's done either by plug-and-play software (run by the driver) or by jumpers for legacy serial ports. Even if you give an I/O address or IRQ to the driver via `setserial` it will not set such values and assumes that they have already been set. If you give it wrong values, the serial port will not work right (if at all).

For legacy ports, if you know the I/O address but don't know the IRQ you may command setserial to attempt to determine the IRQ.

You can see a list of possible commands by just typing `setserial` with no arguments. This fails to show you the one-letter options such as -v for verbose which you should normally use when troubleshooting. Note that setserial calls an IO address a "port". If you type:

```
setserial -g /dev/ttyS*
```

You'll see some info about how the device driver is configured for your ports. In many cases you'll see some ports displayed with what appears at first glance to be erroneous IRQs and addresses. But if you also see: `"UART: unknown"` just ignore the entire line since no serial port exists at that address.

If you add -a to the option -g you will see more info although few people need to deal with (or understand) this additional info since the default settings you see usually work fine. In normal cases the hardware is set up the same way as "setserial" reports. But if you are having problems there is a good chance that `setserial` has it wrong. In fact, you can run "setserial" and assign a purely fictitious I/O port address, any IRQ, and whatever uart type you would like to have. Then the next time you type "setserial ..." it will display these bogus values you've supplied to the driver. They will also be officially registered with the kernel as displayed (at the top of the screen) by the "scanport" command (Debian). Of course the serial port driver will not work correctly (if at all) if you attempt to use such a port. Thus, when giving parameters to `setserial`, "anything goes". Well almost. If you assign one port a base address that is already assigned (such as 3e8) it may not accept it. But if you use 3e9 it will accept it. Unfortunately 3e9 is actually assigned since it is within the range starting at base address 3e8. Thus the moral of the story is to make sure your data is correct before assigning resources with setserial.

Configuration file

While assignments made by setserial are lost when the PC is powered off, a configuration file may restore them when the PC is started up again. In newer versions, what you change by setserial might get automatically saved to a configuration file. When `setserial` runs it uses the info from the configuration file.

Where this configuration file resides depends on your distribution. Look at the start-up scripts somewhere in the /etc/ tree (such as /etc/init.d/ or /etc/rc.d/) and read the startup script for "serial" or "setserial" or the like. It should show where the configuration file(s) reside. In Debian there are 4 options for use of this configuration file:

1. Don't use this file at all. At each boot, the serial driver alone detects the ports and setserial doesn't ever run. ("kernel" option)
2. Save what `setserial` reports when the system is first shutdown and put it in the configuration file. After that, don't ever make any changes to the configuration file, even if someone has made changes by running the `setserial` command on the command line and then shuts down the system. ("autosave-once" option)
3. At every shutdown, save whatever `setserial` detects to the configuration file. ("autosave" option)
4. Manually edit the configuration file to set the configuration. Don't ever do any automatic saves to it. ("manual" option)

In olden days (perhaps before 2000), there wasn't any configuration file and the configuration was manually set (hard coded) inside the shell script that ran `setserial`. See Edit a script (prior to version 2.15).

Probing

You probe for a port with `setserial` only when you suspect that it has been enabled (by PnP methods, the BIOS, jumpers, etc.). Otherwise `setserial` probing will never find it since its address doesn't exist. A problem is where the software looks for a port at specified I/O addresses. Prior to probing with "setserial", one may run the "scanport" (Debian) command to check all possible ports in one scan. It makes crude guesses as to what is on some ports but doesn't determine the IRQ. It's a fast first start. It may hang your PC but so far it's worked fine for me. Note that non-Debian distributions don't seem to supply "scanport". Is there another scan program?

With appropriate options, `setserial` can probe (at a given I/O address) for a serial port but you must guess the I/O address. If you ask it to probe for /dev/ttyS2 for example, it will only probe at the address it thinks ttyS2 is at (2F8). If you tell setserial that ttyS2 is at a different address, then it will probe at that address, etc. See Probing

The purpose of such probing is to see if there is a uart there, and if so, what its IRQ is. Use `setserial` mainly as a last resort as there are faster ways to attempt it such as wvdialconf to detect modems, looking at very early boot-time messages, or using `pnpdump --dumpregs`, or lspci -vv. But if you want to detect hardware with `setserial` use for example :

```
setserial /dev/ttyS2 -v autoconfig
```

If the resulting message shows a uart type such as 16550A, then you're OK. If instead it shows "`unknown`" for the uart type, then there is supposedly no serial port at all at that I/O address. Some cheap serial ports don't identify themselves correctly so if you see "`unknown`" you still might have a serial port there.

Besides auto-probing for a uart type, setserial can auto-probe for IRQ's but this doesn't always work right either. In one case it first gave the wrong irq but when the command was repeated it found the correct irq. In versions of setserial >= 2.15, the results of your last probe test could be automatically saved and put into a distribution-specific configuration file such as `/etc/serial.conf` or `/etc/sysconfig/serial` or `/var/lib/setserial/autoserial.conf` for Debian. This will be used next time you start Linux.

It may be that two serial ports both have the same IO address set in the hardware. Of course this is not normally permitted for the ISA bus but it sometimes happens anyway. Probing detects one serial port when

actually there are two. However if they have different IRQs, then the probe for IRQs may show IRQ = 0. For me, it only did this if I first used `setserial` to give the IRQ a fictitious value.

Boot-time Configuration

While `setserial` may run via an initialization script, something akin to `setserial` also runs earlier when the serial module is loaded (or when the kernel starts the built-in serial driver if it was compiled into the kernel). Thus when you watch the start-up messages on the screen it may look like it ran twice, and in fact it has.

If the first message is for a legacy port, the IRQs shown may be wrong since it didn't probe for IRQs. If there is a second report of serial ports, it may the result of a script such as /etc/init.d/setserial. It usually does no probing and thus could be wrong about how the hardware is actually set. It only shows configuration data that got saved in a configuration files. The old method, prior to setserial 2.15, was to manually write such data directly into the script.

When the kernel loads the serial module (or if the "module equivalent" is built into the kernel) then all supported PnP ports are detected. For legacy (non-PnP) ports, only `ttyS{0-3}` are auto-detected and the driver is set to use only IRQs 4 and 3 (regardless of what IRQs are actually set in the hardware). No probing is done for IRQs but it's possible to do this manually. You see this as a boot-time message just as if `setserial` had been run.

To correct possible errors in IRQs (or for other reasons) there may be a script file somewhere that runs `setserial`. Unfortunately, if this file has some IRQs wrong, the kernel will still have incorrect info about the IRQs. This file is usually part of the initialization done at boot-time. Whether it runs or not depends on how you (and/or your distribution) have set things up. It may also depends on the runlevel.

Before modifying a configuration file, you can test out a "proposed" `setserial` command by just typing it on the command line. In some cases the results of this use of `setserial` will automatically get saved somewhere such as /etc/serial.conf (or autoserial.conf or serial) when you shutdown. So if it worked OK (and solved your problem) then there's no need to modify any configuration file. See Configuration method using /etc/serial.conf, etc..

Edit a script (required prior to version 2.15)

This is how it was done prior to `setserial` 2.15 (1999) The objective was to modify (or create) a script file in the /etc tree that runs setserial at boot-time. Most distributions provided such a file (but it may not have initially resided in the /etc tree).

So prior to version 2.15 (1999) it was simpler. All you did was edit a script. There was no /etc/serial.conf file (or the like) to configure setserial. Thus you needed to find the file that runs "setserial" at boot time and edit it. If it didn't exist, you needed to create one (or place the commands in a file that ran early at boot-time). If such a file was currently being used it's likely was somewhere in the /etc directory-tree. But Redhat <6.0 has supplied it in /usr/doc/setserial/ but you need to move it to the /etc tree before using it.

The script `/etc/rc.d/rc.serial` was commonly used in the past. The Debian distribution used `/etc/rc.boot/0setserial`. Another file once used was `/etc/rc.d/rc.local` but it's may not have run early enough. It was reported that other processes may try to open the serial port before rc.local ran resulting in serial communication failure. Later on it most likely was found in /etc/init.d/ but wasn't normally intended to be edited.

If such a file was supplied, it likely contained a number of commented-out examples. By uncommenting some of these and/or modifying them, you could set things up correctly. It was important use a valid path for `setserial`, and a valid device name. You could do a test by executing this file manually (just type its name as the super-user) to see if it works right. Testing like this was a lot faster than doing repeated reboots to get it right.

For versions >= 2.15 (provided your distribution implemented the change, Redhat didn't at first) it may be more tricky to do since the file that runs setserial on startup, /etc/init.d/setserial or the like was not intended to be edited by the user. See Configuration method using /etc/serial.conf, etc..

An example line in such a script was:

```
/sbin/setserial /dev/ttyS3 irq 5 uart 16550A  skip_test
```

or, if you wanted setserial to automatically determine the uart and the IRQ for ttyS3 you would have used something like this:

```
/sbin/setserial  /dev/ttyS3 auto_irq skip_test autoconfig
```

This was done for every serial port you wanted to auto configure, using a device name that really does exist on your machine. In some cases it didn't work right due to the hardware.

Configuration method using /etc/serial.conf, etc.

Prior to setserial version 2.15 (1999), the way to configure setserial was to manually edit the shell-script that ran setserial at boot-time. See Edit a script (before version 2.15). This was simple, but the simple and clear method has been changed to something that is unnecessarily complex. Today the script and configuration file are two different files instead of one. This shell-script is not edited but gets its data from a configuration file such as `/etc/serial.conf` (or `/var/lib/setserial/autoserial.conf`).

Furthermore you may not even need to edit serial.conf (or the like) because using the "setserial" command on the command line may automatically cause serial.conf to be edited appropriately. This was done so that you may not need to edit any file in order to set up (or change) what setserial does each time that Linux is booted.

What often happens is this: When you shut down your PC the script that ran "setserial" at boot-time is run again, but this time it only does what the part for the "stop" case says to do: It uses "setserial" to find out what the current state of "setserial" is, and it puts that info into the serial configuration file such as `serial.conf`. Thus when you run "setserial" to change the serial.conf file, it doesn't get changed immediately but only when and if you shut down normally.

Now you can perhaps guess what problems might occur. Suppose you don't shut down normally (someone turns the power off, etc.) and the changes don't get saved. Suppose you experiment with "setserial" and forget to run it a final time to restore the original state (or make a mistake in restoring the original state). Then your "experimental" settings are saved. And worst of all, unless you know which options were set in the configuration file, you don't know what will happen. One option in Debian (and likely other distributions) is known as "AUTOSAVE-ONCE" which saves changes only for the first time you make them with the setserial command.

If the option "###AUTOSAVE###" is set and you manually edit serial.conf, then your editing is destroyed when you shut down because it gets changed back to the state of setserial at shutdown. There is a way to disable the changing of serial.conf at shutdown and that is to remove "###AUTOSAVE###" or the like from

first line of serial.conf. In the Debian distribution, the removal of "###AUTOSAVE###" from the first line was once automatically done after the first time you shutdown just after installation. To retain this effect the "AUTOSAVE-ONCE" option was created which only does a save when time the system is shut down for the first time (just after you install or update the setserial program).

The file most commonly used to run setserial at boot-time (in conformance with the configuration file) is now /etc/init.d/setserial (Debian) or /etc/init.d/serial (Redhat), or etc., but it should not normally be edited. For 2.15, Redhat 6.0 just had a file /usr/doc/setserial-2.15/rc.serial which you have to move to /etc/init.d/ if you want setserial to run at boot-time.

To disable a port, use `setserial` to set it to "uart none". This will not be saved. The format of /etc/serial.conf appears to be just like that of the parameters placed after "setserial" on the command line with one line for each port. If you don't use autosave, you may edit /etc/serial.conf manually.

In order to force the current settings set by setserial to be saved to the configuration file (serial.conf) without shutting down, do what normally happens when you shutdown: Run the shell-script `/etc/init.d/{set}serial stop`. The "stop" command will save the current configuration but the serial ports still keep working OK.

In some cases you may wind up with both the old and new configuration methods installed but hopefully only one of them runs at boot-time. Debian labeled obsolete files with "...pre-2.15".

IRQs

By default, both ttyS0 and ttyS2 will share IRQ 4, while ttyS1 and ttyS3 share IRQ 3. But while sharing serial interrupts (using them in running programs) is OK for the PCI bus, it's not permitted for the ISA bus unless you: 1. have kernel 2.2 or better, and 2. you've compiled in support for this, and 3. your serial hardware supports it. See

Interrupt sharing and Kernels 2.2+ If you only have two serial ports, ttyS0 and ttyS1, you're still OK since IRQ sharing conflicts don't exist for non-existent devices.

If you add a legacy internal modem (without plug-and-play) and retain ttyS0 and ttyS1, then you should attempt to find an unused IRQ and set it in your serial port (or modem card) and then use setserial to assign it to your device driver. If IRQ 5 is not being used for a sound card, this could be used for a modem.

Laptops: PCMCIA

If you have a Laptop, read PCMCIA-HOWTO for info on the serial configuration. For serial ports on the motherboard, setserial is used just like it is for a desktop. But for PCMCIA cards (such as a modem) it's a different story. The configuring of the PCMCIA system should automatically run setserial so you shouldn't need to run it. If you do run it (by a script file or by /etc/serial.conf) it might be different and cause trouble. The autosave feature for serial.conf shouldn't save anything for PCMCIA cards (but Debian did until 2.15-7). Of course, it's always OK to use setserial to find out how the driver is configured for PCMCIA cards.

10.2 What is isapnp ?

`isapnp` is a program to configure Plug-and-Play (PnP) devices on the ISA bus including internal modems. It comes in a package called "isapnptools" and includes another program, "pnpdump" which finds all your ISA PnP devices and shows you options for configuring them in a format which may be added to the PnP

configuration file: /etc/isapnp.conf. It may also be used with the --dumpregs option to show the current IO address and IRQ of the modem's serial port. The isapnp command may be put into a startup file so that it runs each time you start the computer and thus will configure ISA PnP devices. It is able to do this even if your BIOS doesn't support PnP. See Plug-and-Play-HOWTO.

10.3 What is wvdialconf ?

`wvdialconf` will try to find which serial port (ttyS?) has a modem on it. It also creates a configuration program for the wvdial program. `wvdial` is used for simplified dialing out using the PPP protocol to an ISP. It can also look for modems which are not currently in use. It will automatically devise a "suitable" init string for the modem but sometimes gets it wrong. Since this command has no options, it's simple to use but you must give it the name of a file to put the init string (and other data) into. For example type: wvdialconf my_config_file_name.

11. Trying Out Your Modem (Dialing Out)

11.1 Are You Ready to Dial Out ?

Once you've plugged in your modem and know which serial port it's on you're ready to try using it. The protocol on the telephone line will be PPP (Point-to-Point Protocol), but PPP often gets set up without you needing to know much about it. If you already have an account with an ISP to connect to the Internet, you could try using a program like "wvdial" to connect to the Internet.

As an alternative to taking one big step using PPP to connect to the Internet, you could do a two step process: First just test out your modem without using PPP (using Minicom or Kermit). Then if your modem works OK, use "wvdial" or another ppp dialer to connect to the Internet. A different strategy is to first try a ppp dialer and then if that doesn't work out, fallback to Minicom or Kermit to see if your modem works OK. Knowing how to use either Minicom or Kermit is handy for dialing out to other modems directly without going thru the Internet. If you are going to use Minicom or Kermit you must find a phone number to dial that will accept phone calls from a computer (without using PPP). Perhaps a local library has such a phone number for its on-line catalog.

Then make sure you are ready to phone. Do you know what serial port (such as ttyS2) your modem is on? You should have found this out when you io-irq configured your serial ports. Have you decided what speed you are going to use for this port? See Speed Table for a quick selection or What Speed Should I Use with My Modem for more details. If you have no clue of what speed to set, try setting it a few times faster than the advertised speed of your modem. Also remember that if you see a menu where an option is "hardware flow control" and/or "RTS/CTS" or the like, select it. Is a live telephone cable plugged in to your modem? You may want to connect this cable to a real telephone to make sure that it can produce a dial tone.

Now you need to select a communication (dialing) program to use to dial out. Internet dialing programs (using PPP) include wvdial, pppconfig (Debian), kppp (KDE), and for Gnome: gnome-ppp or "modem lights". Non-internet dialing programs include: minicom, seyon (X Window), and kermit. See section Communications Programs about some communications programs. Three examples are presented next: Dialing Out with wvdial Dialing Out with Minicom and Dialing Out with Kermit

11.2 Dialing Out with wvdial

Wvdial is a program with not only dials out, but starts PPP and logs you in to an ISP where you get to the Internet. Wvdial may be configured during the installation process or by using the program "wvdialconf". See the man pages for both "wvdialconf" and "wvdial". However, before using wvdial you must do two other tasks not covered by the wvdial documentation:

- set up your network on your PC. The old HOWTO, "ISP-Hookup-HOWTO" has some info on how to do this but fails to mention programs such as wvdial which replaces "chatscripts".
- configure your browser

11.3 Dialing Out with Minicom

Minicom comes with most Linux distributions. To configure it you should be the root user. As root, type "minicom -s" to configure. This will take you directly to the configuration (set-up) menus. This allows you to use the configuration immediately. If you just type "minicom" and then configure, you'll need to leave and restart minicom for the configuration to take effect. Within minicom type ^A to see the bottom status line. This shows to type ^A Z for help (you've already typed the ^A so just type z).

Most of the options don't need to be set for just simply dialing out. To configure you have to supply a few basic items: the name of the serial port your modem is on such as /dev/ttyS2 and the speed such as 115200. These are set at the serial port menu. Go to it and set them. Also (if possible) set hardware flow control (RTS/CTS). Then save them. When typing in the speed, you should also see something like "8N1" which you should leave alone. It means: 8-bit bytes, No parity, 1 stop-bit appended to each byte. If you can't find the speed you want, a lower speed will always work for a test. Exit (hit return) when done and save the configuration as default (dfl) using the menu. Unless you've used the -s option when you called minicom, you'll need to exit minicom and start it again so it can now find the serial port and initialize the modem.

Now you are ready to dial. But first at the main screen you get after you first type "minicom" make sure there's a modem there by typing AT and then hit the <enter> key. It should display OK. If it doesn't, try typing ATQ0 V1 EI and see if you get OK. If you still don't get OK, something is wrong and there is no point of trying to dial. Why you might need to type: ATQ0 V1 E1 is because a modem can be get into a state where is can't display OK and this should get it out of that state.

If you got the "OK" go back to help and select the dialing directory. You may edit it and type in a phone number, etc. into the directory and then select "dial" to dial it. Alternatively, you may just dial manually (by selecting "manual" and then type the number at the keyboard). If it doesn't work, carefully note any error messages and try to figure out what went wrong.

11.4 Dialing Out with Kermit

You can find the latest version of `kermit` at http://www.columbia.edu/kermit/. For example, say your modem was on `ttyS4`, and its speed was 115200 bps. You would do the following:

```
linux# kermit
C-Kermit 6.0.192, 6 Sep 96, for Linux
 Copyright (C) 1985, 1996,
  Trustees of Columbia University in the City of New York.
Default file-transfer mode is BINARY
Type ? or HELP for help.
```

```
C-Kermit>set line /dev/ttyS4
C-Kermit>set carrier-watch off
C-Kermit>set speed 115200
/dev/ttyS4, 115200 bps
C-Kermit>c
Connecting to /dev/ttyS4, speed 115200.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
ATE1Q0V1                          ; you type this and then the Enter key
OK                                ; modem should respond with this
```

If your modem responds to `AT` commands, you can assume your modem is working correctly on the Linux side. Now try calling another modem by typing:

```
ATDT7654321
```

where 7654321 is a phone number. Use ATDP instead of ATDT if you have a pulse line. If the call goes through, your modem is working.

To get back to the `kermit` prompt, hold down the Ctrl key, press the backslash key, then let go of the Ctrl key, then press the C key:

```
Ctrl-\-C
(Back at linux)
C-Kermit>quit
linux#
```

This was just a test using the primitive "by-hand" dialing method. The normal method is to let `kermit` do the dialing for you with its built-in modem database and automatic dialing features, for example using a US Robotics (USR) modem:

```
linux# kermit
C-Kermit 6.0.192, 6 Sep 1997, for Linux
 Copyright (C) 1985, 1996,
  Trustees of Columbia University in the City of New York.
Default file-transfer mode is BINARY
Type ? or HELP for help
C-Kermit>set modem type usr        ; Select modem type
C-Kermit>set line /dev/ttyS4       ; Select communication device
C-Kermit>set speed 115200          ; Set the dialing speed
C-Kermit>dial 7654321              ; Dial
 Number: 7654321
 Device=/dev/ttyS4, modem=usr, speed=115200
 Call completed.<BEEP>
Connecting to /dev/ttyS4, speed 115200
The escape character is Ctrl-\ (ASCII 28, FS).
Type the escape character followed by C to get back,
or followed by ? to see other options.

Welcome to ...  (a welcome message, etc.)

login:
```

12. Dial-In

12.1 Dial-In Overview

Dial-in is where you set up your PC so that others may dial in to your PC (at your phone number) and use your PC. Unfortunately some use the term "dial-in" when what they actually mean is just the opposite: dial-out.

Dial-in works like this: Someone with a modem dials your telephone number. Your modem answers the phone ring and connects. Once the caller is connected, the `getty` program is notified and starts the login process for the caller. After the caller has logged in, the caller then may use your PC. It could be almost as if they were sitting at your monitor-console.

The caller may use a script to automatically log in. This script will be of the expect-send type. For example it expects "login:" and then (after it detects "login:") will send the users login name. It next expects the password and then sends the password, etc. Then once the user has been automatically logged in, the /etc/passwd (password file) might specify that a shell (such as bash) will be started for the user. Or it might specify that PPP is to start so that the user may be connected to the Internet. See the PPP-HOWTO for more details. The program that you use at your PC to handle dialin is called `getty` or `mgetty`. See Getty

An advanced getty program such as mgetty can watch to see if PPP is started by the PC on the other end. If so, the login prompt would be skipped, a PPP connection would be made, and login would take place automatically over the PPP connection.

12.2 What Happens when Someone Dials In ?

Here's a more detailed description of dialin. This all assumes that you are using either mgetty or uugetty. Agetty is inferior and doesn't work exactly the same (see About agetty)

For dialin to work, the modem must be listening for a ring and getty must be running and ready to respond to the call. Your modem is normally listening for incoming calls, but what it does when it gets a ring depends on how it's configured. The modem can either automatically answer the phone or not directly answer it. In the latter case the modem sends a "RING" message to getty and then getty tells the modem to answer the ring. In either case, it may be set up to answer on say the 4th ring. This means that if the call is not for the modem, one must walk/run to the phone and pick it up manually before the 4th ring. Then an ordinary conversation can take place on the telephone. If ons gets to the phone too late one will hear the high pitched tones of the modem which has answered the call.

Once the modem answers the call it sends tones to the other modem (and conversely). The two modems negotiate how they will communicate and when this is completed your modem sends a "CONNECT" message (or the like) to `getty`. When `getty` gets this message, it sends a login prompt out the serial port. Once a user name is given to this prompt `getty` may just call on a program named `login` to handle the login procedure from there on. While `getty` usually starts running at boot-time it should wait until a connection is made before sending out a "login" prompt.

Now for more details on the two methods of answering the call. The first method is where the modem automatically answers the call. In this case the number of times it will ring before answering is controlled by the S0 register of the modem. If S0 is set to 3, the modem will automatically answer on the 3rd ring. If S0 is set to 0 then the modem will only answer the call if getty sends it an "A" (= Answer) AT command to the

modem while the phone is ringing. (Actually an "ATA" is sent since all modem commands are prefixed by "AT".) This is the second method of answering, known as "manual" answering, since the modem itself doesn't do it automatically (but getty does). You might think it best to utilize the ability of the modem hardware to automatically answer the call, but it's actually better if `getty` answers it "manually".

For the "manual" answer case, `getty` opens the port at boot-time and listens. When the phone rings, a "RING" message is sent to the listening `getty`. Then if `getty` wants to answer this ring, it sends the modem an "A" command. Note that getty may be set to answer only after say 4 "RING" messages (the 4th ring) similar to the automatic answer method. The modem then makes a connection and sends a "CONNECT ..." message to `getty` which then sends a login prompt to the caller. It's not all quite this simple as there are some special tricks used to allow dial-out when waiting for a call. See Dialing Out while Waiting for an Incoming Call

The automatic answer case uses the CD (Carrier Detect aka DCD) wire from the modem to the serial port to tell when a connection is made. It works like this: At boot-time `getty` tries to open the serial port but the attempt fails since the modem has negated CD (the modem is idle). Then the `getty` program waits at the open statement in the program until a CD signal is raised. When a CD signal arrives (perhaps hours later) then the port is opened and `getty` sends the login prompt. While `getty` is waiting (sleeping) at the open statement, other processes can run so it doesn't degrade computer performance. What actually wakes `getty` up is an interrupt which is issued when the CD line from the modem changes its state to on.

You may wonder how getty is able to open the serial port in the "manual"-answer case since CD may be negated. Well, there's a way to write a program to force the port to open even if there is no CD signal raised.

12.3 56k Doesn't Work for Dialin

If you expect that people will be able to dial-in to you at 56k, it can't be done unless you have all the following:

1. You have a digital connection to the telephone company such as a trunkside-T1 or ISDN line
2. You use special digital modems (see Digital Modems)
3. You have a "... concentrator", or the like to interface your digital-modems to the digital lines of the telephone company.

A "... concentrator" may be called a "modem concentrator" or a "remote access concentrator" or it could be included in a "remote access server" (RAS) which includes the digital modems, etc. This type of setup is used by ISPs (Internet Service Providers).

12.4 Getty

Introduction to Getty

A `getty` program (including agetty, mgetty, etc.) is what you run for dialin. You don't need it for dialout. In addition to presenting a login prompt, it also may help answer an incoming telephone call. Originally getty was used for logging in to a computer from a dumb terminal. A major use of it today is for logging in to a Linux system at a console. There are several different getty programs a few of which work OK with modems for dialin. The getty program is usually started either at boot-time or when someone dials in to your computer. It must be called from the /etc/inittab file. In this file you may find some examples which you will likely need to edit a bit.

There are four different getty programs to choose from that may be used with modems for dial-in: `mgetty`, `uugetty`, `getty_em`, and `agetty`. A brief overview is given in the following subsections. `agetty` is the weakest of the four and it's mainly for use with directly connected text-terminals. `mgetty` includes support for fax and voice mail but `uugetty` doesn't. But `mgetty` allegedly lacks a few of the features of `uugetty`. `getty_em` is a simplified version of `uugetty`. Thus `mgetty` is likely your best choice unless you are already familiar with `uugetty` (or find it difficult to get `mgetty`). The syntax for these getty programs differs, so be sure to check that you are using the correct syntax in `/etc/inittab` for whichever getty you use.

In order to see what documentation exists about the various gettys on your computer, use the "locate" command. Type: locate "*getty*" (including the quotes may help). Note that many distributions just call the program getty even though it may actually be agetty, uugetty, etc. But if you read the man page (type: man getty), it might disclose which getty it is. This should be the getty program with path `/sbin/getty`.

How getty respawns

After you log in you will notice (by using "top", "ps -ax", or "ptree") that the getty process is no longer running. What happened to it? Why does getty restart again if your shell is killed? Here's why.

After you type in your user name, getty takes it and calls the login program telling it your user name. The getty process is replaced by the login process. The login process asks for your password, checks it and starts whatever process is specified in your password file. This process is often the bash shell. If so, bash starts and replaces the login process. Note that one process replaces another and that the bash shell process originally started as the getty process. The implications of this will be explained below.

Now in the /etc/inittab file, getty is supposed to respawn (restart) if killed. It says so on the line that calls getty. But if the bash shell (or the login process) is killed, getty respawns (restarts). Why? Well, both the login process and bash are replacements for getty and inherit the signal connections establish by their predecessors. In fact if you observe the details you will notice that the replacement process will have the same process ID as the original process. Thus bash is sort of getty in disguise with the same process ID number. If bash is killed it is just like getty was killed (even though getty isn't running anymore). This results in getty respawning.

When one logs out, all the processes on that serial port are killed including the bash shell. This may also happen (if enabled) if a hangup signal is sent to the serial port by a drop of DCD voltage by the modem. Either the logout or drop in DCD will result in getty respawning. One may force getty to respawn by manually killing bash (or login) either by hitting the k key, etc. while in "top" or with the "kill" command. You will likely need to kill it with signal 9 (which can't be ignored).

About mgetty

`mgetty` was written as a replacement for `uugetty` which was in existence long before `mgetty`. Both are for use with modems but mgetty is best (unless you already are committed to `uugetty`). `mgetty` may be also used for directly connected terminals but doesn't have many features for this purpose. In addition to allowing dialup logins, `mgetty` also provides FAX support, auto PPP detection, and caller-id support. It permits dialing out when mgetty is waiting for an incoming phone call. There is a supplemental program called `vgetty` which handles voicemail for some modems. `mgetty` documentation is fair (except for voice mail), and is not supplemented in this HOWTO. To automatically start PPP one must edit /etc/mgetty/login.conf to use "AutoPPP" (has example). You can find the latest information on `mgetty` at http://www.leo.org/~doering/mgetty/ and http://alpha.greenie.net/mgetty/

About uugetty

`getty_ps` contains two programs: `getty` is used for console and terminal devices, and `uugetty` for modems. Greg Hankins (former author of Serial-HOWTO) used `uugetty` so his writings about it are included here. See Uugetty.

About getty_em

This is a simplified version of ``uugetty''. It was written by Vern Hoxie after he became fully confused with complex support files needed for getty_ps and uugetty.

It is part of the collection of serial port utilities and information by Vern Hoxie available via ftp from scicom.alphacdc.com/pub/linux. The name of the collection is ``serial_suite.tgz''.

About agetty

This subsection is long since the author tried using agetty for dialin. `agetty` is seemingly simple since there are no initialization files. But when I tried it, it opened the serial port even when there was no CD signal present. It then sent both a login prompt and the /etc/issue file to the modem in the AT-command state before a connection was made. The modem thinks all this an AT command and if it does contain any "at" strings (by accident) it is likely to adversely modify your modem profile. Echo wars can start where getty and the modem send the same string back and forth over and over. You may see a "respawning too rapidly" error message if this happens. To prevent this you need to disable all echoing and result codes from the modem (E0 and Q1). Also use the -i option with agetty to prevent any /etc/issue file from being sent.

If you start getty on the modem port and a few seconds later find that you have the login process running on that port instead of getty, it means that a bogus user name has been sent to agetty from the modem. To keep this from happening, I had to save my dial-in profile in the modem so that it become effective at power-on. The other saved profile is for dial-out. Then any dial-out programs which use the modem must use a Z, Z0, or Z1 in their init string to initialize the modem for dial-out (by loading the saved dial-out profile). If the 1-profile is for dial-in you use Z1 to load it, etc. If you want to listen for dial-in later on, then the modem needs to be reset to the dial-in profile. Not all dial-out programs can do this reset upon exit from them.

Thus while agetty may work OK if you set up a dial-in profile correctly in the modem hardware, it's probably best suited for virtual consoles or terminals rather than modems. If agetty is running for dialin, there's no easy way to dial out. When someone first dials in to agetty, they should hit the return key to get the login prompt. `agetty` in the Debian distribution is just named `getty`.

About mingetty, and fbgetty

`mingetty` is a small getty that will work only for monitors (the usual console) so you can't use it with modems for dialin. `fbgetty` is as above but supports framebuffers.

12.5 Why "Manual" Answer is Best

The difference between the two ways of answering is exhibited when the computer happens to be down but the modem is still working. For the manual case, the "RING" message is sent to getty but since the computer is down, getty isn't there and the phone never gets answered. There are no telephone charges when there is no answer. For the automatic answer case, the modem (which is still on) answers the phone but no login message

is ever sent since the computer is down. The phone bill runs up as the waiting continues. If the phone call is toll-free, it doesn't make much difference, although it may be frustrating waiting for a login prompt that never arrives. `mgetty` uses manual answer. `Uugetty` can do this too by using a configuration script.

12.6 Dialing Out while Waiting for an Incoming Call

Here's what could go wrong with a simple-minded manual-answer situation. Suppose another process dials out while getty is listening for a "RING" message from its modem on the serial wire. Then incoming bytes for the dial-out process flow from the modem to the serial port. For example, your modem may send a "CONNECT" message to your serial port when the dial-out process connects. If getty reads this there's trouble since reads are destructive reads. Once getty reads it, then the dial-out process that is expecting "CONNECT" (or something else) can't read it. Thus the dial-out process is likely to fail.

There's a way to avoid this and here's how mgetty does it. When mgetty is listing for an incoming call, it doesn't read anything from the port until it thinks that the characters are for mgetty. Mgetty monitors the port and if characters arrive, it doesn't read them right away. Instead, it first checks to see if another process is using the port. If so, mgetty backs off and closes the port (but the port remains open for the other process). Thus, if another process dials out, mgetty doesn't interfere with it. When the other process finally closes the port, then mgetty resumes "listening". It's a special type of "listening" that refrains from reading until mgetty believes that what it will read is for mgetty (hopefully a "RING" message).

When mgetty checks to see if another process is using the port, it actually checks for valid lockfiles on the port. If the other process failed to use lockfiles, too bad for it. For more details see the mgetty documentation: "How mgetty works". For programmers only: "listening" is actually using the system calls "poll" or "select" to monitor the port. They are likely also used to monitor the port when a non-mgetty process is using the port.

Then there's the problem of modem configuration when using mgetty. Mgetty first sets this configuration when it starts up and uses a user-specified chat script to do it. So the modem is now configured not to auto-answer but to send the RING string to mgetty when the phone rings. Now suppose that while mgetty is waiting for an incoming call, another program makes and outgoing call and reconfigures the modem to something bad for mgetty. To prevent this, when mgetty detects that some other program using the port has exited, mgetty just exits itself. This results in mgetty starting up anew (respawns per the /etc/inittab file) and then mgetty reconfigures the modem so that it's all set up to listen once more for incoming calls.

With auto-answer (not normally used by mgetty), getty is waiting for CD to be raised so that it can open the port. One may dial out, but once a connection is made, the modem's CD is raised. If getty were to then read the port it would eat the characters intended to be read by the dial-out connection. While agetty will have this problem, it's claimed that uugetty will check lockfiles before reading (similar to mgetty).

12.7 Ending a Dial-in Call

There are two major ways to end a dial-in call. The caller may either logout or just hang up. For the hangup case see Caller hangs up

Caller logs out

When the call is over, the normal way to end the connection is for the remote user to log out. This should result in the dial-in PC hanging up the phone line as will be explained shortly. Note that this behavior is not what normally happens when one logs out from a PC (when not using a modem). In this case, the user logs

out and immediately gets a login prompt as an invitation to log in again. But a remote user that types "logout" gets hung up on, and must redial if s/he wants to log in again. If there was no hang-up when the user logged out, the connection would be maintained, and not give anyone else the opportunity to login.

Logging out by the remote user of your dial-in PC will kill the shell that the remote user was using on your dial-in PC. Now, since there is nothing running on this port anymore, the port closes and sends a hangup signal to the modem by negating DTR. This will only happen if stty -a shows hupcl (default). hupcl = Hang UP on CLose => drop DTR (the "hang up" signal) when the last program running on this port is closed). But normally, when the shell is killed it's like getty was killed and getty will respawn (since it's set this way in /etc/inittab). This will almost immediately open the port again and raise DTR. So DTR is said to wink (drop only for only a small fraction of a second and then reassert itself). With modern fast computers, this wink would be too short to be recognized by the modem so the serial driver is supposed to make this wink longer (provided stty has hupcl set, and provided ...). But there was a complaint in 2003 that it's not long enough.

The dial-in PC modem getting the hangup (negated DTR signal) will then hang up the phone line (provided the modem has been configured to do this --see below). The modem should then be ready to answer any new incoming calls. For mgetty, if there is a chat-script to initialize the modem and possibly reset the modem will happen also. If the modem didn't hang up due to too short of a DTR wink, then a newly spawned getty might be able to hang up. mgetty itself creates a long DTR wink when it starts up to hang up the modem. It's claimed that making a respawned getty clean up the mess (the modem still online) left by the previous call, is not the right way to handle this.

When setting up mgetty, one may use a chat-script with the code sequence +++ to the modem to put it into AT command mode if DTR didn't work. The +++ must have both an initial and final minimal time delay. Once in AT command mode, a hangup command (H0) may be sent to the modem as well as other AT commands. One may have things set up to use both this method and the DTR method and so that if one method should fail, the other one will hopefully work. If the PC fails to successfully signal the modem when a logout happens (or fails to use the +++ escape when restarting getty), then the modem is apt to remain in on-line mode and no more incoming calls can be received. It's claimed that this might be a security risk.

When DTR drops (is negated)

When DTR (the "hang-up" signal when negated) is dropped (negated), what the modem does depends on the value of the &D option in the modem's profile. If it's &D0 nothing at all happens (the modem ignores the negation of DTR). Here's what happens when the computer drops DTR:

`&D2`: The modem will hang up and go into AT command mode (off-line) to wait for the next call. Except that it will not be able to automatically answer the phone until DTR is raised again. But since mgetty automatically respawns (if so set in /etc/inittab) then mgetty will immediately restart after a logout and this will raise DTR. So what happens when someone logs out is that DTR only is negated for a fraction of a second (winks) before it gets raised again. During this wink, the DTR must be negated for at least the time specified by register S25, otherwise the modem will not hang up.

`&D3`: or S13 = 1. In this case the modem does a hard reset when DTR drops: It hangs up and restores the saved profile as specified by &Y. It should now be in the same state it was in when first powered on. But mgetty may have a chat script which will send the modem an init string and thus change the profile again. Since these two changes in profile happen at about the same time, could this be a problem (known as "race conditions").

The S25 limit may have no effect so even a very short DTR "wink" is detected. Another brand of modem says the S25 limit is still valid. Thus `&D3` is a stronger "reset" than `&D2` which doesn't restore the saved profile and

could require a longer wink to work.

Under favorable conditions, either `&D3` or `&D2` should work OK. It's reported that for a few modems, only `&D2` works OK. Could this be related to a possible race condition mentioned above if `&D3` is used?

Caller hangs up

Instead of logging out the normal way, a caller may just hang up (by closing the "terminal" program s/he's using, etc). This results in a lost connection and of course a loss of carrier. Other problems could also cause a loss of carrier. The modem hangs up and waits for the next call. Except that there is no mgetty running yet to start the login process.

Here's how getty gets started again: The loss of carrier should negate the CD signal sent by the modem to the serial port (provided `&C1` has been set). When the PC's serial port gets the dropped CD signal it should kill the shell, provided clocal is negated (-clocal) and then getty should respawn. mgetty raises clocal when it starts. Does it later drop clocal?

This paragraph is about other things that happen but do nothing. Only the curious need read it. When the shell is killed, a DTR wink is sent to the modem but since the modem is not on-line anymore and has already hung up due to lost carrier, the modem ignores the drop of DTR. The loss of carrier also negates the DSR signal sent by the modem to the serial port (provided &S1 or &S2 is set) but this signal is ignored (by Linux). The "NO CARRIER" result code should be generated by the modem but where does it go to ?

12.8 Dial-in Modem Configuration

The getty programs have a provision for sending an init string to the modem to configure it. But you may need to edit it. Another method is to save a suitable init string inside the modem (see Init Strings: Saving and Recalling for how to save it in the modem).

The configuration for dial-in depends both on the getty you use and perhaps on your modem. If you can't find suggested configurations in other documentation here are some hints using Hayes AT commands:

- `&C1` Make the CD line to the serial port track the actual state of the carrier (CD raised only when there's carrier). Getty_em requires &C0 (CD always raised)
- `&D3` Do a hard reset of the modem when someone logs out (or hangs up). For some modems it's reported that &D2 is required since they can't tolerate a hard reset ??
- `E0` Don't echo AT commands back to the serial port. This is a must for agetty. Some suggest E1 (echo AT commands) for mgetty. For dial-out you want E1 so you can see what was sent.
- `&K3` Use hardware flow control
- `Q0` Echo results words (such as CONNECT). Most gettys use them. But it's reported an AT&T version of uugetty and agetty require Q2 (no result words for dial-in).
- `S0=?` mgetty suggests S0=0 (manual answer) but you give the number of rings on the mgetty command line. If you set S0=3 the modem will auto-answer on the 3rd ring, etc. Agetty uses auto-answer. So does uugetty (usually).
- `V1` Display results (such as CONNECT) in words (and not in code)
- `X4` Check for dialtone and busy signal

12.9 Callback

Callback is where someone first dials in to your modem. Then, you get a little info from the caller and then call it right back. Why would you want to do this? One reason is to save on telephone bills if you can call the caller cheaper than the caller can call you. Another is to make sure that the caller really is who it claims to be. If a caller calls you and claims to be calling from its usual phone number, then one way to verify this is to actually place a new call to that number.

There's a program for Linux called "callback" that works with mgetty. It's at ftp://ftp.rug.nl/contrib/frank/software/linux/callback/ Step-by-step instructions on how someone installed it (and PPP) is at http://www.stokely.com/unix.serial.port.resources/callback.html

12.10 Distinctive Ring

"Distinctive ring" is where you want the modem to answer phone calls only for certain types of rings like long, short, long, short, etc. To do this, you first need a modem that supports distinctive ring. The Netcomm Roadster modem can be set with an AT command to do the following, for example: It will send to mgetty: DROF=14 DRON=4 RING DROF=4 DRON=2 RING ... meaning that there is a 1.4 seconds of initial silence (DROF=14) followed by .4 seconds of ringing (DRON=4) etc. RING is also reported after each ring. Note that the modem can't be set to answer a certain type of ring, it only informs the program listening on the serial port (such as mgetty) what the ring sequence is. Then if the program likes the sequence, it sends an AT command to the modem to answer the call. Unfortunately, mgetty doesn't recognize such ring sequences but there's a workaround that may work.

Mgetty only waits for a "RING" and will ignore the DRON and DROF (Distinctive Ring OFf) words. For the Netcomm Roadster modem, you can set the delay between sending DRON= and RING. For example you could set a delay of 2.0 seconds. However, if within this 2.0 second period another actual ring occurs, the modem cancels the delayed RING message and never sends it. So you might be able to set this delay so the calls you don't want the modem to answer never send a RING message to mgetty. But for the calls you want mgetty to answer, you get the interval between rings to be long enough so that "RING" is sent to mgetty and mgetty answers the call. This workaround is not always feasible, especially if the telephone company doesn't give you much choice of distinctive rings. Will the above work for other modems that support distinctive ring?

For the above modem, the AT command: AT+VDR=1,24 sets the above delay for 2.4 seconds. You can put this in an "init-chat" parameter in mgetty.config.

12.11 Voice Mail

Voice mail is like an answering machine run by a computer. To do this you must have a modem that supports "voice" and supporting software. Instead of storing the messages on tape, they are stored in digital format on a hard-drive. When a person phones you, they hear a "greeting" message and can then leave a message for you. More advanced systems would have caller-selectable mail boxes and caller-selectable messages to listen to. Free software is available in Linux for simple answering, but doesn't seem to be available yet for the more advanced stuff.

I know of two different voicemail packages for Linux. One is a very minimal package (see Voicemail Software). The other, more advanced, but currently poorly documented, is `vgetty` which supports all ELSA modems and the ITU v.253 standard. It's an optional addition to the well documented and widely distributed

`mgetty` program. In the Debian distribution, you must get the mgetty-voice package in addition to the mgetty package and mgetty-doc (documentation) package.

12.12 Simple Manual Dial-In

This is really doing it manually! It doesn't even permit the caller to login but the caller may "chat" with you, etc. It's a way to answer a call without bothering to edit any configuration files for dial-in or enabling getty. To do it you run a terminal program such as minicom. Make sure it's connected to your modem by typing "AT <enter>" and expect "OK". Then wait for the call. Then you really answer the call manually by typing "ATA" when the phone is ringing. This doesn't run getty and the caller can't login. But if the caller is calling in with a terminal program they may type a message to your screen (and conversely). You both may send files back and forth by using the commands built into the terminal programs (such as minicom). Another way to answer such a call would be to type say "ATS0=3" just before the call comes in to enable the modem to auto-answer on the third ring.

This is one way to crudely transfer files with someone on a MS Windows PC who uses HyperTerminal or Terminal (for Windows 3.x or DOS). These two MS programs are something like minicom. Using this simple manual method (for Linux-to-Linux or MS-to-Linux) requires two people to be present, one on each end of the phone line connection running a terminal communications program. Be warned that if both people type at the same time it's chaos. It's a "last resort" way to transfer files between any two people that have PCs (either Linux or MS Windows). It could also be used for testing your modem or as a preliminary test before setting up dial-in.

12.13 Complex GUI Dial-In, VNC

At the opposite extreme to the simple (but labor intensive) manual dial-in described above, is one that results in GUI graphical interface to the Linux PC. This generally requires that a network running TCP/IP protocol exist between the two computers. One way to get such a "network" is to dial-out to a PC set for dial-in and then run PPP on the phone line. PPP will use TCP/IP protocol encapsulated inside the PPP packets. Both sides must run PPP and mgetty can be configured to start PPP as soon as the caller does. The caller may use a PPP-dialer program just like they were dialing an ISP. Programs such as wvdial, eznet, or chat scripts should do it.

Instead of this tiny network over a phone connection a much larger network (the entire world) is reached via an ISP. For their lowest-rate service many of them use proxy servers that will not give you access to the ports you need to use. Even if they don't use proxy servers, the IP address they give you is only temporary for the session, so you'll need to email this IP to whomever wants to reach you. If you get a more expensive ISP service, then you can avoid these problems.

One way to get a GUI interface from the remote PC is to run the GPLed program: Virtual Network Computer (VNC) from AT&T. It has a server part which you run on your Linux PC for dial-in and a viewer (client) part used for dial-out. Neither of these actually does any dialing or login but assumes that you have a network already set up. The VNC server has an X-server built in and may use Linux's twm window manager. See the article on VNC in Linux Magazine: http://www.linux-mag.com/2000-11/desktop_03.html. The AT&T site for VNC is: http://www.uk.research.att.com/vnc/.

With VNC one can also connect to remote Windows PCs, get the Windows GUI on a Linux PC, and run Windows programs on the remote Windows PC. Of course the Windows PC must be running VNC (as a server). Obviously, a GUI connection over a modem will be slower than a text-only connection especially if you run KDE or GNOME or want 16-bit color.

12.14 Interoperability with MS Windows

Once you have dial-in set up, others may call in to you using minicom (or the like) from Unix-like systems. From MS Windows one may call you using "HyperTerminal (or just "Terminal" in Windows 3.1 or DOS).

If in Windows one wants to use dial-up with a network protocol over the phone line it's called "Dial-up Networking". But it probably will not be able to communicate with Linux. For setting up such dial-in in Windows one clicks on "server" while dial-out is the "client. Such dial-in is often called "remote control" meaning that the caller can use your PC, run programs on it, and thus control it remotely.

While it's easy to call in to a text-based Linux system from MS Windows, it's not so easy the other way around (partly because Windows is not text-based and would need to put the caller into DOS where files wouldn't be protected like they are in Linux.

However Windows "Dial-up Networking" can establish a dial-in provided the caller uses certain network protocols over the phone line: MS's or Novel's (two protocols not liked by Linux). So if someone with Windows enables their Dial-up networking server in Windows 98, you can't just dial in directly to it from Linux. This type of dial-in doesn't permit the caller to run most of the programs on the host like Linux does. It's called "remote access" and one may transfer files, use the hosts printer, access databases, etc. Is there some way to interface to Dial-up Networking from Linux??

It is possible for two people to crudely chat and send files using Minicom on the Linux end and HyperTerminal on the Windows end. It's all done manually by two live persons, one on each end of the phone connection. See Simple Manual Dial-In.

At the opposite extreme, one would like to run a dial-in so that the person calling would get a GUI interface. For that a network protocol is normally used. It's possible using PC Anywhere for Windows or VNC for both Linux and Windows. But PC Anywhere doesn't seem to talk to Linux ?? Other Window programs for "remote control" include Laplink, Co-Session, and Microcom. Do any such programs support Linux besides VNC ??

13. Uugetty for Dial-In (from the old Serial-HOWTO)

Be aware that you could use mgetty as a (better?) alternative to `uugetty`. `mgetty` is newer and more popular than uugetty. See Getty for a brief comparison of these 2 gettys.

13.1 Installing getty_ps

Since uugetty is part of getty_ps you'll first have to install getty_ps. If you don't have it, get the latest version from `metalab.unc.edu:/pub/Linux/system/serial`. In particular, if you want to use high speeds (57600 and 115200 bps), you must get version 2.0.7j or later. You must also have libc 5.x or greater.

By default, `getty_ps` will be configured to be Linux FSSTND (File System Standard) compliant, which means that the binaries will be in `/sbin`, and the config files will be named `/etc/conf.{uu}getty.ttyS`*N*. This is not apparent from the documentation! It will also expect lock files to go in `/var/lock`. Make sure you have the `/var/lock` directory.

If you don't want FSSTND compliance, binaries will go in `/etc`, config files will go in `/etc/default/{uu}getty.ttyS`*N*, and lock files will go in `/usr/spool/uucp`. I recommend doing things this way if you are using UUCP, because UUCP will have problems if you move the lock files to where it isn't

looking for them.

getty_ps can also use syslogd to log messages. See the man pages for syslogd(1) and syslog.conf(5) for setting up syslogd, if you don't have it running already. Messages are logged with priority LOG_AUTH, errors use LOG_ERR, and debugging uses LOG_DEBUG. If you don't want to use syslogd you can edit tune.h in the getty_ps source files to use a log file for messages instead, namely /var/adm/getty.log by default.

Decide on if you want FSSTND compliance and syslog capability. You can also choose a combination of the two. Edit the Makefile, tune.h and config.h to reflect your decisions. Then compile and install according to the instructions included with the package.

13.2 Setting up uugetty

With uugetty you may dial out with your modem while uugetty is watching the port for logins. uugetty does important lock file checking. Update /etc/gettydefs to include an entry for your modem. For help with the meaning of the entries that you put into /etc/gettydefs, see the "serial_suite" collected by Vern Hoxie. How to get it is in section See About getty_em. When you are done editing /etc/gettydefs, you can verify that the syntax is correct by doing:

```
linux# getty -c /etc/gettydefs
```

Modern Modems

If you have a 9600 bps or faster modem with data compression, you can lock your serial port to one speed. For example:

```
# 115200 fixed speed
F115200# B115200 CS8 # B115200 SANE -ISTRIP HUPCL #@S @L @B login: #F115200
```

If you have your modem set up to do RTS/CTS hardware flow control, you can add CRTSCTS to the entries:

```
# 115200 fixed speed with hardware flow control
F115200# B115200 CS8 CRTSCTS # B115200 SANE -ISTRIP HUPCL CRTSCTS #@S @L @B login: #
```

Old slow modems

If you have a slow modem (under 9600 bps) Then, instead of one line for a single speed, your need several lines to try a number of speeds. Note that these lines are linked to each other by the last "word" in the line such as #4800. Blank lines are needed between each entry. Are the higher modem-to-serial_port speeds in this example really needed for a slow modem ?? The uugetty documentation shows them so I'm not yet deleting them.

```
# Modem entries
115200# B115200 CS8 # B115200 SANE -ISTRIP HUPCL #@S @L @B login: #57600

57600# B57600 CS8 # B57600 SANE -ISTRIP HUPCL #@S @L @B login: #38400

38400# B38400 CS8 # B38400 SANE -ISTRIP HUPCL #@S @L @B login: #19200

19200# B19200 CS8 # B19200 SANE -ISTRIP HUPCL #@S @L @B login: #9600
```

```
9600# B9600 CS8 # B9600 SANE -ISTRIP HUPCL #@S @L @B login: #4800

4800# B4800 CS8 # B4800 SANE -ISTRIP HUPCL #@S @L @B login: #2400

2400# B2400 CS8 # B2400 SANE -ISTRIP HUPCL #@S @L @B login: #1200

1200# B1200 CS8 # B1200 SANE -ISTRIP HUPCL #@S @L @B login: #115200
```

Login Banner

If you want, you can make `uugetty` print interesting things in the login banner. In Greg's examples, he has the system name, the serial line, and the current bps rate. You can add other things:

```
@B    The current (evaluated at the time the @B is seen) bps rate.
@D    The current date, in MM/DD/YY.
@L    The serial line to which uugetty is attached.
@S    The system name.
@T    The current time, in HH:MM:SS (24-hour).
@U    The number of currently signed-on users.  This is  a
      count of the number of entries in the /etc/utmp file
      that have a non-null ut_name field.
@V    The value of VERSION, as given in the defaults file.
To display a single '@' character, use either '\@' or '@@'.
```

13.3 Customizing uugetty

There are lots of parameters you can tweak for each port you have. These are implemented in separate config files for each port. The file `/etc/conf.uugetty` will be used by *all* instances of `uugetty`, and `/etc/conf.uugetty.ttyS`*N* will only be used by that one port. Sample default config files can be found with the `getty_ps` source files, which come with most Linux distributions. Due to space concerns, they are not listed here. Note that if you are using older versions of `uugetty` (older than 2.0.7e), or aren't using FSSTND, then the default file will be `/etc/default/uugetty.ttyS`*N*. Greg's `/etc/conf.uugetty.ttyS3` looked like this:

```
# sample uugetty configuration file for a Hayes compatible modem to allow
# incoming modem connections
#
# line to initialize
INITLINE=ttyS3
# timeout to disconnect if idle...
TIMEOUT=60
# modem initialization string...
# format: <expect> <send> ... (chat sequence)
INIT="" AT\r OK\r\n
WAITFOR=RING
CONNECT="" ATA\r CONNECT\s\A
# this line sets the time to delay before sending the login banner
DELAY=1
#DEBUG=010
```

Add the following line to your `/etc/inittab`, so that `uugetty` is run on your serial port, substituting in the correct information for your environment - run-levels (2345 or 345, etc.) config file location, port, speed, and default terminal type:

```
S3:2345:respawn:/sbin/uugetty -d /etc/default/uugetty.ttyS3 ttyS3 F115200 vt100
```

Restart `init`:

```
linux# init q
```

For the speed parameter in your `/etc/inittab`, you want to use the highest bps rate that your modem supports.

Now Linux will be watching your serial port for connections. Dial in from another machine and login to you Linux system.

`uugetty` has a lot more options, see the man page for `uugetty)` (often just called `getty`) for a full description. Among other things there is a scheduling feature, and a ringback feature.

14. What Speed Should I Use with My Modem?

By "speed" we really mean the "data flow rate" but almost everybody incorrectly calls it speed. For all modern modems you have no choice of the speed that the modem uses on the telephone line since it will automatically choose the highest possible speed that is feasible under the circumstances. If one modem is slower than the other, then the faster modem will operate at the slower modem's speed. On a noisy line, the speed will drop still lower.

While the above speeds are selected automatically by the modems you do have a choice as to what speed will be used between your modem and your computer (PC-to-modem speed). This is sometimes called "DTE speed" where "DTE" stands for Data Terminal Equipment (Your computer is a DTE.) You need to set this speed high enough so this part of the signal path will not be a bottleneck. The setting for the DTE speed is the maximum speed of this link. Most of the time it will likely actually operate at lower speeds.

For an external modem, DTE speed is the speed (in bits/sec) of the flow over the cable between you modem and PC. For an internal modem, it's the same idea since the modem also emulates a serial port. It may seem ridiculous having a speed limit on communication between a computer and a modem card that is directly connected inside the computer to a much higher speed bus. But it's usually that way since the modem card probably includes a dedicated serial port which does have speed limits (and settable speeds). However, some software modems have no such speed limits.

14.1 Speed and Data Compression

What speed do you choose? If it were not for "data compression" one might try to choose a DTE speed exactly the same as the modem speed. Data compression takes the bytes sent to the modem from your computer and encodes them into a fewer number of bytes. For example, if the flow (speed) from the PC to the modem was 20,000 bytes/sec (bps) and the compression ratio was 2 to 1, then only 10,000 bytes/sec would flow over the telephone line. Thus for a 2:1 compression ratio you would need to set the DTE speed to double the maximum modem speed on the phone line. If the compression ratio were 3 to 1 you would need to set it 3 times faster, etc.

14.2 Where do I Set Speed ?

This DTE (PC-to-modem) speed is normally set by a menu in your communications program or by an option given to the getty command if someone is dialing in. You can't set the DCE modem-to-modem speed since this is set automatically by the modem to the highest feasible speed after negotiation with the other modem.

Well, actually you can set the modem-to-modem speed with the S37 register but you shouldn't do it. If the two modems on a connection were to be set this way to different speeds, then they couldn't communicate with each other.

14.3 Can't Set a High Enough Speed

Speeds over 115.2kbps

The top speed of 115.2k has been standard since the mid 1990's. But by the year 2000, most new serial ports supported higher speeds of 230.4k and 460.8k. Some also support 921.6k. Unfortunately Linux seldom uses these speeds due to lack of drivers. Thus such ports behave just like 115.2k ports unless the higher speeds are enabled by special software. To get these speeds you need to compile the kernel with special patches or use modules until support is built into the kernel's serial driver.

Unfortunately serial port manufacturers never got together on a standard way to support high speeds, so the serial driver needs to support a variety of hardware. Once high speed is enabled, a standard way to choose it is to set baud_base to the highest speed with setserial (unless the serial driver does this for you). The software will then use a divisor of 1 to set the highest speed. All this will hopefully be supported by the Linux kernel sometime in 2003.

A driver for the w83627hf chip (used on many motherboards such as the Tyan S2460) is at https://www.muru.com/linux/w83627hf/

A non-standard way that some manufacturers have implemented high speed is to use a very large number for the divisor to get the high speed. This number isn't really a divisor at all since it doesn't divide anything. It's just serves as a code number to tell the hardware what speed to use. In such cases you need to compile the kernel with special patches.

One patch to support this second type of high-speed hardware is called shsmod (Super High Speed Mode). There are both Windows and Linux versions of this patch. See http://www.devdrv.com/shsmod/. There is also a module for the VIA VT82C686 chip http://www.kati.fi/viahss/. Using it may result in buffer overflow.

For internal modems, only a minority of them advertise that they support speeds of over 115.2k for their built-in serial ports. Does shsmod support these ??

How speed is set in hardware: the divisor and baud_base

Speed is set by having the serial port's clock change frequency. But this change happens not by actually changing the frequency of the oscillator driving the clock but by "dividing" the clock's frequency. For example, to divide by two, just ignore every other clock tick. This cuts the speed in half. Dividing by 3 makes the clock run at 1/3 frequency, etc. So to slow the clock down (meaning set speed), we just send the clock a divisor. It's sent by the serial driver to a register in the port. Thus speed is set by a divisor.

If the clock runs at a top speed of 115,000 bps (common), then here are the divisors for various speeds (assuming a maximum speed of 115,200): 1 (115.2k), 2 (57.6k), 3 (38.4k), 6 (19.2k), 12 (9.6k), 24 (4.8k), 48 (2.4k), 96 (1.2k), etc. The serial driver sets the speed in the hardware by sending the hardware only a "divisor" (a positive integer). This "divisor" divides the "maximum speed" of the hardware resulting in a slower speed (except a divisor of 1 obviously tells the hardware to run at maximum speed).

There are exceptions to the above since for certain serial port hardware, speeds above 115.2k are set by using a very high divisor. Keep that exception in mind as you read the rest of this section. Normally, if you specify a speed of 115.2k (in your communication program or by stty) then the serial driver sets the port hardware to divisor 1 which sets the highest speed.

Besides using a very high divisor to set high speed, the conventional way to do it is as follows: If you happen to have hardware with a maximum speed of say 230.4k (and the 230.4k speed has been enabled in the hardware), then specifying 115.2k will result in divisor 1. For some hardware this will actually give you 230.4k. This is double the speed that you set. In fact, for any speed you set, the actual speed will be double. If you had hardware that could run at 460.8k then the actual speed would be quadruple what you set. All the above assumes that you don't use "setserial" to modify things.

Setting the divisor, speed accounting

To correct this accounting (but not always fix the problem) you may use "setserial" to change the baud_base to the actual maximal speed of your port such as 230.4k. Then if you set the speed (by your application or by stty) to 230.4k, a divisor of 1 will be used and you'll get the same speed as you set.

If you have very old software which will not allow you to tell it such a high speed (but your hardware has it enabled) then you might want to look into using the "spd_cust" parameter. This allows you to tell the application that the speed is 38,400 but the actual speed for this case is determined by the value of "divisor" which has also been set in setserial. I think it best to try to avoid using this kludge.

There are some brands of UARTs that uses a very high divisor to set high speeds. There isn't any satisfactory way to use "setserial" (say set "divisor 32770") to get such a speed since then setserial would then think that the speed is very low and disable the FIFO in the UART.

Crystal frequency is higher than baud_base

Note that the baud_base setting is usually much lower than the frequency of the crystal oscillator since the crystal frequency of say 1.8432 MHz is divided by 16 in the hardware to get the actual top speed of 115.2k. The reason the crystal frequency needs to be higher is so that this high crystal speed can generate clock ticks to take a number of samples of each bit to determine if it's a 1 or a 0.

Actually, the 1.8432 MHz "crystal frequency" may be obtained from a 18.432 MHz crystal oscillator by dividing by 10 before being fed to the UART. Other schemes are also possible as long as the UART performs properly.

14.4 Speed Table

It's best to have at least a 16650 UART for a 56k modem but few modems or serial ports provide it. Second best is a 16550 that has been tweaked to give 230,400 bps (230.4 kbps). Most people still use a 16550 that is only 115.2 kbps but it's claimed to only slow down thruput by a few percent (on average). This is because a typical compression ratio is 2 to 1 and for downloading compressed files (packages) it's 1 to 1. There's no degradation for these cases. Here are some suggested speeds to set your serial line if your modem speed is:

- 56k (V.92): use 115.2 kbps or 230.4 kbps (best)
- 56k (V.90): use 115.2 kbps or 230.4 kbps (best)
- 33.6k (V.34bis): use 115.2 kbps
- 28.8k (V.34): use 115.2 kbps

- 14.4k (V.32bis): use 57600 bps
- 9.6k (V.32): use 38400 bps
- slower than a 9600 bps (V.32) modem: Set the speed to the same speed as the modem (unless you have data compression).

All the above speeds may use V.42bis data compression and V.42 error correction. If data compression is not used then the speed may be set lower so long as it's above the modem speed.

15. Communications Programs And Utilities

While PPP is used for Internet access you also need a dialer program (or script) that will dial a phone number and then start PPP once a connection is made. When the other side answers the phone, then three things happen: a modem connection is established (CONNECT), PPP is started at both ends, and you get logged in automatically. The exact sequence of the last 2 events may vary. Dialer programs for ppp include wvdial, chap scripts, kppp, RP3 (front end to wvdial and ifup), gnome-ppp, and "modem lights" (Gnome). Linuxconf configures some dialers.

There are also older dialer programs which can dial out (via a modem) but don't connect to the Internet. Instead, you get connected to a computer somewhere that puts a text image on your screen. This was much used in the past to connect to Bulletin Boards. See PCs and BBSs Today, it might be used to connect to a remote computer that you may login to (including a PC at home). Programs for this are: `minicom` (the most popular), `Seyon` (X-Windows only) and `Kermit`. Some people have likely also used these programs for dialing out with ppp for the Internet but it's not what they were originally designed for.

15.1 Minicom vs. Kermit

Minicom is only a communications program while Kermit is both a communications program and a file transfer protocol. But one may use the Kermit protocol from within Minicom (provided one has Kermit installed on one's PC). Minicom is menu based while Kermit is command line based (interactive at the special Kermit prompt). While the Kermit program is free software, the documentation is not all free. There is no detailed manual supplied and it is suggested that you purchase a book as the manual. However Kermit has interactive online help which tells all but lacks tutorial explanations for the beginner. Commands may be put in a script file so you don't have to type them over again each time. Kermit (as a communications program) is more powerful than Minicom.

Although all Minicom documentation is free, it's not as extensive as Kermit's. In my opinion it's easier to set up Minicom, there is less to learn, and you can still use kermit from within Minicom. But if you want to write a script for automatically doing file transfers, etc. Kermit is better.

g-kermit is a gpled kermit which has no dialout capabilities.

15.2 List of Communication Software

Here is a list of some communication software you can choose from, If they didn't come with your distribution they should be available via FTP. I would like comparative comments on the dialout programs. Are the least popular ones obsolete?

Least Popular Dialout

- `ecu` - a communications program
- `pcomm` - `procomm`-like communications program with zmodem
- `xc` - xcomm communication package

Most Popular Dialout

- PPP dialers for getting on the internet: `wvdial`, `eznet`, `chat`, `pon` (uses chat),
- `minicom` - `telix`-like communications program. Can work with scripts, zmodem, kermit
- C-Kermit - portable, scriptable, serial and TCP/IP communications including file transfer, character-set translation, and zmodem support
- `seyon` - X based communication program

Fax

By using a fax program, you may use most modems to send faxes. In this case you dial out directly and not via ppp and an ISP. You also pay any long-distance telephone charges. email is more efficient.

- `efax` is a small fax program
- `hylafax` is a large fax program based on the client-server model.
- `mgetty+fax` handles fax stuff and login for dial-ins
- A fax protocol tutorial http://www.iec.org/online/tutorials/vfoip/topic08.html

Voicemail Software

- vgetty is an extension to mgetty that handles voicemail for some modems. It should come with recent releases of mgetty.
- VOCPis a "complete voice messaging" system for Linux.

Dial-in (uses getty)

- `mgetty+fax` is for modems and is well documented (except for voicemail as of early 1999). It also handles fax stuff and provides an alternative to `uugetty`. It's incorporating voicemail (using vgetty) features. See About mgetty
- `uugetty` is also for modems. It comes as a part of the `ps_getty` package. See About getty_ps

Network Connection

- `ser2net`
- `sredird`

Other

- `callback` is where you dial out to a remote modem and then that modem hangs up and calls you back (to save on phone bills).
- `xringd` listens for rings and detects inter-ring times etc.
- `SLiRP` and `term` provide a PPP-like service that you can run in user space on a remote computer with a shell account. See term and SLiRP for more details

- `ZyXEL` is a control program for ZyXEL U-1496 modems. It handles dialin, dialout, dial back security, FAXing, and voice mailbox functions.
- SLIP and PPP software can be found at `ftp://metalab.unc.edu/pub/Linux/system/network/serial/`.
- Other things can be found on `ftp://metalab.unc.edu/pub/Linux/system/serial` and `ftp://metalab.unc.edu/pub/Linux/apps/serialcomm` or one of the many mirrors. These are the directories where serial programs are kept.

15.3 SLiRP and term

`SLiRP` and `term` are programs which are of use if you only have a dial-up shell account on a Unix-like machine and want to get the equivalent of a PPP account (or the like) without being authorized to have it (possibly because you don't want to pay extra for it, etc.). `SLiRP` is more popular than `term` which is almost obsolete.

To use `SLiRP` you install it in your shell account on the remote computer. Then you dial up the account and run SLiRP on the remote and PPP on your local PC. You now have a PPP connection over which you may run a web browser on your local PC such as Netscape, etc. There may be some problems as SLiRP is not as good as a real PPP account. Some accounts may provide SLiRP since it saves on IP addresses (You have no IP address while using SLiRP).

`term` is something like SLiRP only you need to run `term` on both the local and remote computer. There is no PPP on the phone line since `term` uses its own protocol. To use `term` from your PC you need to use a term-aware version of ftp to do ftp, etc. Thus it's easier to use SLiRP since the ordinary version of ftp works fine with SLiRP. There is an unmaintained Term HOWTO.

15.4 MS Windows

If you want someone who uses MS Windows to dial in to your Linux PC then if they use:

- Windows 3.x: use `Terminal`
- Windows 95/98/2000: use `HyperTerminal`

Third party dial-out programs include HyperTerminal Private Edition.

16. Two Modems (Modem Doubling)

16.1 Introduction

By using two modems at the same time, the flow of data can be doubled. It takes two modems and two phone lines. There are two methods of doing this. One is "modem bonding" where software at both ends of the modem-to-modem connection enables the paired modems to work like a single channel.

The second method is called "modem teaming. Only one end of the connection uses software to make 2 different connections to the internet. Then when a file is to be downloaded, one modem gets the first half of the file. The second modems simultaneously gets the last half of the same file by pretending that it's resuming a download that was interrupted in the middle of the file. Is there any modem teaming support in Linux ??

16.2 Modem Bonding

There are two ways to do this in Linux: EQL and multilink. These are provided as part of the Linux kernel (provided they've been selected when the kernel was compiled). For multilink the kernel must be at least v.2.4. Both ends of the connection must run them. Few (if any) ISPs provide EQL but many provide Multilink.

The way it works is something like multiplexing only it's the other way around. Thus it's called inverse-multiplexing. For the multilink case, suppose you're sending some packets. The first packet goes out on modem1 while the second packet is going out on modem2. Then the third packet follows the first packet on modem1. The forth packet goes on modem2, etc. To keep each modem busy, it may be necessary to send out more packets on one modem than the other. Since EQL is not packet based, it doesn't split up the flow on packet boundaries.

EQL

EQL is "serial line load balancing" which has been available for Linux since at least 1995. An old (1995) howto on it is in the kernel documentation (in the networking subdirectory). Unfortunately, ISPs don't seem to provide EQL.

Multilink

Staring with kernel 2.4 in 2000, experimental support is provided for multilink. It must be selected when compiling the kernel and it only works with PPP.

17. ISDN "Modems"

To use ISDN you must have a special ISDN telephone line supplied by your telephone company at additional cost. An ISDN "modem" is really a Terminal Adapter (TA). Like analog modems, there are internal ones on cards and external ones that connect to serial ports.

17.1 External ISDN "Modems"

Configuring an external ISDN modem on a serial port is about the same as configuring an analog modem. The main difference is in the init string. Unfortunately, the init strings are different for different models of ISDN modems. There is often one AT command for a speed of 64k and another for 128k, etc. So you need to find out what init string to use and tell it to say wvdial, etc.

17.2 Internal ISDN "Modems"

Support for some of these cards can be built into the 2.4 or 2.6 kernels or added as a module. The tty ports are ttyI2 for example. The kernel documentation has an "isdn" subdirectory which describes various drivers which support various isdn cards. A major website for ISDN is http://www.isdn4linux.de

For configuring, one might use the "isdn-config" GUI. A Debian package "isdnutils" is available. There is SuSE ISDN Howto (not a LDP Howto) which is translated from German http://sol.parkland.cc.il.us/sdb/en/html/isdn.html There is an isdn4linux package and a newsgroup: de.alt.comm.isdn4linux. Many of the postings are in German.

18. Troubleshooting

18.1 My Modem is Physically There but Can't be Found

The error message might also be something like "Modem not responding". There are at least 4 possible reasons:

1. Your modem is a winmodem and no working driver has been installed for it. Or the modem is defective or in "online data mode" where it doesn't respond. See winmodem
2. Your modem is disabled since both the BIOS and Linux failed to enable it. It has no IO address.
3. Your modem is enabled and has an IO address but it has no ttyS device number (like ttyS14) assigned to that address so the modem can't be used.
4. You modem does have a ttyS number assigned to it (like ttyS4) but you are using the wrong ttyS number (like ttyS2 instead of ttyS4). See wrong ttySx and/or wvdial and/or minicom (test modem)

Case 1: Winmodem

For a winmodem with no driver (or a defective modem) the serial port that the modem is on can usually be found OK. But when the wvdial program (or whatever) interrogates that port, it gets no response since winmodems need a driver to do anything. So you see a message saying that no modem was found. However, it's likely that the modem card was detected at boot-time and it displays a message implying that a modem was found. So you're told both that the modem was found and that it wasn't found! What it all means is that no working modem has been found, since a modem that doesn't work has been found. Of course it could not be working for reasons other than being a winmodem (or linmodem) with no driver. See Software-based Modems (winmodems, linmodems).

Cases 2-3

Cases 2. and 3. mean that no serial port device (such as /dev/ttyS2) exists for the modem. If you suspect this, see Serial Port Can't be Found.

Case 4: Wrong ttySx number

If you are lucky, the problem is case 4. Then you just need to find which ttyS your modem is on.

wvdial

There's a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a-new-file-name>". It will create the new file as a configuration file but you don't need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf ? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected". See minicom (test modem)

minicom (test modem)

Another way try to find out if there's a modem on a certain port is to start "minicom" on the port (after first setting up minicom for that serial port. You will need to save the setup and then exit minicom and start it again. Then type "AT" and you should see "OK". If you don't, try typing ATQ0 V1 EI. If you still don't get OK (and likely don't even see the AT you typed) then there is likely no modem on the port. This may be due

to either case 1. 2. or 3. above

If what you type is really getting thru to a modem, then the lack of response could be due to the modem being in "online data" mode where it can't accept any AT commands. You may have been using the modem and then abruptly disconnected (such as killing the process with signal 9). In that case your modem did not get reset to "command mode" where it can interact to AT commands. "Minicom" may display "You are already online. Hangup first." (For another meaning of this minicom message see You are already online! Hang up first.) Well, you are sort of online but you are may not be connected to anything over the phone line. Wvdial will report "modem not responding" for the same situation.

To fix this as a last resort you could reboot the computer. Another way to try to fix this is to send +++ to the modem to tell it to escape back to "command mode" from "online data mode". On both sides of the +++ sequence there must be about 1 second of delay (nothing sent during "guard time"). This may not work if another process is using the modem since the +++ sequence could wind up with other characters inserted in between them or after the +++ (during the guard time). Ironically, even if the modem line is idle, typing an unexpected +++ is likely to set off an exchange of control packets (that you never see) that will violate the required guard time so that the +++ doesn't do what you wanted. +++ is usually in the string that is named "hangup string" so if you command minicom (or the like) to hangup it might work. Another way to do this is to just exit minicom and then run minicom again.

Other problems which you might observe in minicom besides no response to AT are:

- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.

18.2 "Modem is busy"

What this means depends on what program sent it. The modem could actually be in use (busy). Another cause reported for the SuSE distribution is that there may be two serial drivers present instead of one. One driver was built into the kernel and the second was a module.

In kppp, this message is sent when an attempt to get/set the serial port "stty" parameters fails. (It's similar to the "Input/output error" one may get when trying to use "stty -F /dev/ttySx"). To get a few of these stty parameters, the true address of the port must be known to the driver. So the driver may have the wrong address. The setserial" command will display what the driver thinks but it's likely wrong in this case. So what the "modem busy" often means is that the serial port (and thus the modem) can't be found.

If you have a pci modem, then use one of these commands: lspci -v, or cat /proc/pci, or dmesg to find the correct address and irq of the modem's serial port. Then check to see if "setserial" shows the same thing. If not, you need to run a script at boot-time which contains a setserial command that will tell the driver the correct address and irq.

18.3 "You are already online! Hang up first." (from minicom)

The modem has its CD signal on. Either you are actually online (a remote modem is sending you a carrier) or your modem has been setup to always send the CD signal. In minicom, type at&v to see if &C or &C0 is set. If so, CD will be on even if you are offline and you'll erroneously get this error message. The fix is to set &C1 in the init string or save it in the modem.

18.4 I can't get near 56k on my 56k modem

There must be very low noise on the line for it to work at even close to 56k. Some phone lines are so bad that the speeds obtainable are much slower than 56k (like 28.8k or even slower). Sometimes extension phones connected to the same line can cause problems. To test this you might connect your modem directly at the point where the telephone line enters the building with the feeds for everything else on that line disconnected (if others can tolerate such a test).

18.5 Uploading (downloading) files is broken/slow

Flow control (both at your PC and/or modem-to-modem) may not be enabled. For the uploading case: If you have set a high DTE speed (like 115.2k) then flow from your modem to your PC may work OK but uploading flow in the other direction will not all get thru due to the telephone line bottleneck. This will result in many errors and the resending of packets. It may thus take far too long to send a file. In some cases, files don't make it thru at all.

For the downloading case: If you're downloading long uncompressed files or web pages (and your modem uses data compression) or if you've set a low DTE speed, then downloading may also be broken due to no flow control.

18.6 For Dial-in I Keep Getting "line NNN of inittab invalid"

Make sure you are using the correct syntax for your version of `init`. The different `init`'s that are out there use different syntax in the `/etc/inittab` file. Make sure you are using the correct syntax for your version of `getty`.

18.7 I Keep Getting: ``Id "S4" respawning too fast: disabled for 5 minutes"

Id "S4" is just an example. In this case look on the line which starts with "S4" in `/etc/inittab` and calls getty. This line causes the problem. Make sure the syntax for this line is correct and that the device (ttyS4) exists and can be found. If the modem has negated CD and getty opens the port, you'll get this error message since negated CD will kill getty. Then getty will respawn only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with getty. Make sure your modem is configured correctly. Look at AT commands `E` and `Q`.

If you use uugetty, verify that your `/etc/gettydefs` syntax is correct by doing the following:

```
linux# getty -c /etc/gettydefs
```

This can also happen when the `uugetty` initialization is failing. See section uugetty Still Doesn't Work.

18.8 Dial-in: When remote user hangs up, getty doesn't respawn

One possible cause is that your modem doesn't reset right when DTR is dropped after someone hangs up. Most Hayes compatible modems do this OK with `&D3`. But for USR Courier, SupraFax, and some other modems, you must set `&D2` (and `S13=1` in some cases). Check your modem manual (if you have one). See Ending a Dial-in Call

18.9 NO DIALTONE

It means exactly what it says. Someone else may be using another telephone on the same line. You also get this error if there is no phone line plugged into the modem, or if the phone line is somehow broken. Try plugging a real telephone into the phone cord used by the modem. Check it for a dialtone.

If for some reason your modem doesn't detect a dialtone, then you can force it to dial anyway by putting X3 in the init string.

18.10 NO CARRIER

This means that the analog sine wave (the carrier) from the other modem isn't present like it should be. If you were already connected, this means that the connection has been lost. There may have been noise on the line or a bad connection. The other modem may have hung up on you for some reason: Perhaps the automatic login process didn't work out OK. Perhaps PPP didn't get started OK. Perhaps a time limit was exceeded.

If you get this error before you get connected, it means that the carrier of the other modem wasn't detected by your modem. This may happen if there is there is no properly working modem on the other end. For example, an answering machine could have picked up your call instead of a modem. NO CARRIER will also happen if the modems fail to negotiate a protocol to use. This can happen if you have an early V.90 modem that first tries to negotiate a high speed X2 or K56flex protocol. These two protocols are obsolete and some ISP servers will drop the connection (hang up) when this happens since they have no understanding of such protocols and don't wait around long enough for the calling modem to fallback to V.90.

If you force your modem to drop the connection by dropping the DTR signal or sending your modem the hangup signal (ATH) you may get this error message. But in this case you (or your software) wanted to drop the connection so there should be no problem. In this case you are only supposed to get NO CARRIER if data was lost. So for most cases of dropping a connection by hangup (or by dropping DTR) you only get an OK message. Your modem dialer program may not even display that to you.

18.11 uugetty Still Doesn't Work

There is a `DEBUG` option that comes with `getty_ps`. Edit your config file `/etc/conf.{uu}getty.ttyS`*N* and add `DEBUG=`*NNN*. Where *NNN* is one of the following combination of numbers according to what you are trying to debug:

```
D_OPT   001             option settings
D_DEF   002             defaults file processing
D_UTMP  004             utmp/wtmp processing
D_INIT  010             line initialization (INIT)
D_GTAB  020             gettytab file processing
```

```
D_RUN   040             other runtime diagnostics
D_RB    100             ringback debugging
D_LOCK  200             uugetty lockfile processing
D_SCH   400             schedule processing
D_ALL   777             everything
```

Setting `DEBUG=010` is a good place to start.

If you are running `syslogd`, debugging info will appear in your log files. If you aren't running `syslogd` info will appear in `/tmp/getty:ttyS`*N* for debugging `getty` and `/tmp/uugetty:ttyS`*N* for `uugetty`, and in `/var/adm/getty.log`. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.

You could also try `mgetty`. Some people have better luck with it.

18.12 (The following subsections are in both the Serial and Modem HOWTOs)

18.13 Serial Port Can't be Found

There are 3 possibilities:

1. Your port is disabled since both the BIOS and Linux failed to enable it. It has no IO address.
2. Your port is enabled and has an IO address but it has no ttyS device number (like ttyS14) assigned to that address so the port can't be used. As a last resort, you may need to use "setserial" to assign a ttyS number to it.
3. Your port does have a ttyS number assigned to it (like ttyS14) but you don't know which physical connector it is (on the back of your PC). See the Serial_HOWTO: "Which Connector on the Back of my PC is ttyS1, etc?" But if you want to find which ttyS port the modem is on see My Modem is Physically There but Can't be Found

First check BIOS messages at boot-time (and possibly the BIOS menu for the serial port). Then for the PCI bus type lspci -v. If this shows something like "LPC Bridge" then your port is likely on the LPC bus which is not well supported by Linux yet (but the BIOS might find it) ?? If it's an ISA bus PnP serial port, try "pnpdump --dumpregs" and/or see Plug-and-Play-HOWTO. If the port happens to be enabled then the following two paragraphs may help find the IO port:

Scanning/probing legacy ports

This is mainly for legacy non-PCI ports and ISA ports that are not Plug-and-Play.

Using "scanport" (Debian only ??) will scan all enabled bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. If you suspect that your port may be at a certain address, you may try manually probing with setserial, but it's a slow tedious task if you have several addresses to probe. See Probing.

18.14 Linux Creates an Interrupt Conflict (your PC has an ISA slot)

If your PC has a BIOS that handles ISA (and likely PCI too) then if you find a IRQ conflict, it might be due to a shortage of free IRQs. The BIOS often maintains a list of reserved IRQs, reserved for legacy ISA cards. If too many are reserved, the BIOS may not be able to find a free IRQ and will erroneously assign an IRQ to the serial port that creates a conflict. So check to see if all the reserved IRQs are really needed and if not, unreserve an IRQ that the serial port can use. For more details, see Plug-and-Play-HOWTO.

18.15 Extremely Slow: Text appears on the screen slowly after long delays

It's likely mis-set/conflicting interrupts. Here are some of the symptoms which will happen the first time you try to use a modem, terminal, or serial printer. In some cases you type something but nothing appears on the screen until many seconds later. Only the last character typed may show up. It may be just an invisible <return> character so all you notice is that the cursor jumps down one line. In other cases where a lot of data should appear on the screen, only a batch of about 16 characters appear. Then there is a long wait of many seconds for the next batch of characters. You might also get "input overrun" error messages (or find them in logs).

For more details on the symptoms and why this happens see the Serial-HOWTO section: "Interrupt Problem Details".

If it involves Plug-and-Play devices, see also Plug-and-Play-HOWTO.

As a quick check to see if it really is an interrupt problem, set the IRQ to 0 with "setserial". This will tell the driver to use polling instead of interrupts. If this seems to fix the "slow" problem then you had an interrupt problem. You should still try to solve the problem since polling uses excessive computer resources.

Checking to find the interrupt conflict may not be easy since Linux supposedly doesn't permit any interrupt conflicts and will send you a /dev/ttyS?: Device or resource busy error message if it thinks you are attempting to create a conflict. But a real conflict can be created if "setserial" has told the kernel incorrect info. The kernel has been lied to and thus doesn't think there is any conflict. Thus using "setserial" will not reveal the conflict (nor will looking at /proc/interrupts which bases its info on "setserial"). You still need to know what "setserial" thinks so that you can pinpoint where it's wrong and change it when you determine what's really set in the hardware.

What you need to do is to check how the hardware is set by checking jumpers or using PnP software to check how the hardware is actually set. For PnP run either "pnpdump --dumpregs" (if ISA bus) or run "lspci" (if PCI bus). Compare this to how Linux (e.g. "setserial") thinks the hardware is set.

18.16 Somewhat Slow: I expected it to be a few times faster

An obvious reason is that the baud rate is set too slow. It's claimed that this once happened by trying to set the baud rate to a speed higher than the hardware can support (such as 230400).

Another reason may be that whatever is on the serial port (such as a modem, terminal, printer) doesn't work as fast as you thought it did. A 56k Modem seldom works at 56k and the Internet often has congestion and bottlenecks that slow things down. If the modem on the other end does not have a digital connection to the phone line (and uses a special "digital modem" not sold in most computer stores), then speeds above 33.6k are not possible.

Another possible reason is that you have an obsolete serial port: UART 8250, 16450 or early 16550 (or the serial driver thinks you do). See "What are UARTS" in the Serial-HOWTO.

Use "setserial -g /dev/ttyS*". If it shows anything less than a 16550A, this may be your problem. If you think that "setserial" has it wrong check it out. See What is Setserial for more info. If you really do have an obsolete serial port, lying about it to setserial will only make things worse.

18.17 The Startup Screen Shows Wrong IRQs for the Serial Ports

For non-PnP ports, Linux does not do any IRQ detection on startup. When the serial module loads it only does serial device detection. Thus, disregard what it says about the IRQ, because it's just assuming the standard IRQs. This is done, because IRQ detection is unreliable, and can be fooled. But if and when setserial runs from a start-up script, it changes the IRQ's and displays the new (and hopefully correct) state on the startup screen. If the wrong IRQ is not corrected by a later display on the screen, then you've got a problem.

So, even though I have my `ttyS2` set at IRQ 5, I still see

```
ttyS02 at 0x03e8 (irq = 4) is a 16550A
```

at first when Linux boots. (Older kernels may show "ttyS02" as "tty02" which is the same as ttyS2). You may need to use `setserial` to tell Linux the IRQ you are using.

18.18 "Cannot open /dev/ttyS?: Device or resource busy

See /dev/tty? Device or resource busy

18.19 "Cannot open /dev/ttyS?: Permission denied"

Check the file permissions on this port with "ls -l /dev/ttyS?"_ If you own the ttyS? then you need read and write permissions: crw with the c (Character device) in col. 1. It you don't own it then it will work for you if it shows rw- in cols. 8 & 9 which means that everyone has read and write permission on it. Use "chmod" to change permissions. There are more complicated (and secure) ways to get access like belonging to a "group" that has group permission. Some programs change the permissions when they run but restore them when the program exists normally. But if someone pulls the plug on your PC it's an abnormal exit and correct permissions may not be restored.

18.20 "Cannot open /dev/ttyS?"

Unless stty is set for clocal, the CD pin may need to be asserted in order to open a serial port. If the physical port is not connected to anything, or if it's connected to something that is not powered on (such an external modem) then there will be no voltage on CD from that device. Thus the "cannot open" message. Either set clocal or connect the serial port connector to something and power it on.

Even if a device is powered on and connected to a port, it may sometimes prevent opening the port. An example of this is where the device has negated CD and the CD pin on your PC is negated (negative voltage).

18.21 "Operation not supported by device" for ttyS?

This means that an operation requested by setserial, stty, etc. couldn't be done because the kernel doesn't support doing it. Formerly this was often due to the "serial" module not being loaded. But with the advent of PnP, it may likely mean that there is no modem (or other serial device) at the address where the driver (and setserial) thinks it is. If there is no modem there, commands (for operations) sent to that address obviously don't get done. See What is set in my serial port hardware?

If the "serial" module wasn't loaded but "lsmod" shows you it's now loaded it might be the case that it's loaded now but wasn't loaded when you got the error message. In many cases the module will automatically loaded when needed (if it can be found). To force loading of the "serial" module it may be listed in the file: /etc/modules.conf or /etc/modules. The actual module should reside in: /lib/modules/.../misc/serial.o.

18.22 "Cannot create lockfile. Sorry"

Sometimes when it can't create a lockfile you get the erroneous message: "... Device or resource busy" instead of the one above. When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls -ld /var/lock" to see if the permissions are OK. Giving rwx permissions for the root owner and the group should work, provided that the users that need to dialout belong to that group. Others should have r-x permission. Even with this scheme, there may be a security risk. Use "chmod" to change permissions and "chgrp" to change groups. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial-HOWTO subsection: "What Are Lock Files".

18.23 "Device /dev/ttyS? is locked."

This means that someone else (or some other process) is supposedly using the serial port. There are various ways to try to find out what process is "using" it. One way is to look at the contents of the lockfile (/var/lock/LCK...). It should be the process id. If the process id is say 100 type "ps 100" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 100". If it refuses to be killed use "kill -9 100" to force it to be killed, but then the lockfile will not be removed and you'll need to delete it manually. Of course if there is no such process as 100 then you may just remove the lockfile but in most cases the lockfile should have been automatically removed if it contained a stale process id (such as 100).

18.24 "/dev/tty? Device or resource busy"

This means that the device you are trying to access (or use) is supposedly busy (in use) or that a resource it needs (such as an IRQ) is supposedly being used by another device and can't be shared. This message is easy to understand if it only means that the device is busy (in use). But it sometimes means that a needed resource is already in use (busy). What makes it even more confusing is that in some cases neither the device nor the resources that it needs are actually "busy".

In olden days, if a PC was shutdown by just turning off the power, a bogus lockfile might remain and then later on one would get this bogus message and not be able to use the serial port. Software today is supposed to automatically remove such bogus lockfiles, but as of 2006 there is still a problem with the "wvdial" dialer program related to lockfiles. If wvdial can't create a lockfile because it doesn't have write permission in the /var/lock/ directory, you will see this erroneous message. See Cannot create lockfile. Sorry

The following example is where interrupts can't be shared (at least one of the interrupts is on the ISA bus). The ``resource busy'' part often means (example for `ttyS2`) ``You can't use `ttyS2` since another device is using ttyS2's interrupt.'' The potential interrupt conflict is inferred from what "setserial" thinks. A more accurate error message would be ``Can't use `ttyS2` since the setserial data (and kernel data) indicates that another device is using `ttyS2`'s interrupt''. If two devices use the same IRQ and you start up only one of the devices, everything is OK because there is no conflict yet. But when you next try to start the second device (without quitting the first device) you get a "... busy" error message. This is because the kernel only keeps track of what IRQs are actually in use and actual conflicts don't happen unless the devices are in use (open). The situation for I/O address (such as 0x3f8) conflict is similar.

This error is sometimes due to having two serial drivers: one a module and the other compiled into the kernel. Both drivers try to grab the same resources and one driver finds them "busy".

There are two possible cases when you see this message:

1. There may be a real resource conflict that is being avoided.
2. Setserial has it wrong and the only reason `ttyS2` can't be used is that setserial erroneously predicts a conflict.

What you need to do is to find the interrupt setserial thinks `ttyS2` is using. Look at /proc/tty/driver/serial. You should also be able to find it with the "setserial" command for `ttyS2`.

Bug in old versions: Prior to 2001 there was a bug which wouldn't let you see it with "setserial". Trying to see it would give the same "... busy" error message.

To try to resolve this problem reboot or: exit or gracefully kill all likely conflicting processes. If you reboot: 1. Watch the boot-time messages for the serial ports. 2. Hope that the file that runs "setserial" at boot-time doesn't (by itself) create the same conflict again.

If you think you know what IRQ say `ttyS2` is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflict. It's not a good idea to leave it permanently set at 0 since it will put more load on the CPU.

18.25 "Input/output error" from setserial, stty, pppd, etc.

This means that communication with the serial port isn't working right. It could mean that there isn't any serial port at the IO address that setserial thinks your port is at. It could also be an interrupt conflict (or an IO address conflict). It also may mean that the serial port is in use (busy or opened) and thus the attempt to get/set parameters by setserial or stty failed. It will also happen if you make a typo in the serial port name such as typing "ttys" instead of "ttyS".

18.26 "LSR safety check engaged"

LSR is the name of a hardware register. It usually means that there is no serial port at the address where the driver thinks your serial port is located. You need to find your serial port and possibly configure it. See Locating the Serial Port: IO address IRQs and/or What is Setserial

18.27 Overrun errors on serial port

This is an overrun of the hardware FIFO buffer and you can't increase its size. Bug note (reported in 2002): Due to a bug in some kernel 2.4 versions, the port number may be missing and you will only see "ttyS" (no port number). But if devfs notation such as "tts/2" is being used, there is no bug. See "Higher Serial Thruput" in the Serial-HOWTO.

18.28 Modem doesn't pick up incoming calls

This paragraph is for the case where a modem is used for both dial-in and dial-out. If the modem generates a DCD (=CD) signal, some programs (but not mgetty) will think that the modem is busy. This will cause a problem when you are trying to dial out with a modem and the modem's DCD or DTR are not implemented correctly. The modem should assert DCD only when there is an actual connection (ie someone has dialed in), not when `getty` is watching the port. Check to make sure that your modem is configured to only assert DCD when there is a connection (&C1). DTR should be on (asserted) by the communications program whenever something is using, or watching the line, like `getty`, `kermit`, or some other comm program.

18.29 Port gets characters only sporadically

There could be some other program running on the port. Use "top" (provided you've set it to display the port number) or type "ps -alxw". Look at the results to see if the port is being used by another program. Be on the lookout for the gpm mouse program which often runs on a serial port.

18.30 Troubleshooting Tools

These are some of the programs you might want to use in troubleshooting:

- "lsof /dev/ttyS*" will list serial ports which are open.
- "setserial" shows and sets the low-level hardware configuration of a port (what the driver thinks it is). See What is Setserial
- "stty" shows and sets the configuration of a port (except for that handled by "setserial"). See the Serial-HOWTO section: "Stty".
- "modemstat" or "statserial" or "watch head /proc/tty/driver/serial" will show the current state of various modem signal lines (such as DTR, CTS, etc.). The one in /proc also shows byte flow and errors.
- "irqtune" will give serial port interrupts higher priority to improve performance.
- "hdparm" for hard-disk tuning may help some more.
- "lspci" shows the actual IRQs, etc. of hardware on the PCI bus.
- "pnpdump --dumpregs" shows the actual IRQs, etc. of hardware for PnP devices on the ISA bus.
- Some "files" in the /proc tree (such as ioports, interrupts, and tty/driver/serial).

19. Flash Upgrades

Many modems can be upgraded by reprogramming their flash memories with an upgrade program which you get from the Internet. By sending this "program" from the PC via the serial port to the modem, the modem will store this program in its non-volatile memory (it's still there when the power is turned off). The instructions on installing it are usually on how to do in under Windows so you'll need to figure out how to do the equivalent under Linux (unless you want to install the upgrade under Windows). Sending the program to

the modem is often called a download.

If the latest version of this HOWTO still contains this request (see New Versions of this HOWTO) please send me your experiences with installing such upgrades that will be helpful to others.

Here's the general idea of doing an upgrade. First, there may be a command that you need to send your modem to tell it that what follows is a flash ROM upgrade. In one case this was AT** You can do this by starting a communications program (such as minicom) and type. First type AT <enter> to see if your modem is there and answers "OK".

Next, you need to send an file (sometimes two files) directly to the modem. Communication programs (such as minicom) often use zmodem or kermit to send files to the modem (and beyond) but these put the file into packets which append headers and you want the exact file sent to the modem, not a modified one. But the kermit communications program has a "transmit" command that will send the file directly (without using the kermit packets) so this is one way to send a file directly. Minicom didn't have this feature in 1998.

Another way to send the file(s) would be to escape from the communications program to the shell (in minicom this is ^AJ) and then: `cat upgrade_file_name > /dev/ttyS4` (if your serial port is ttyS4). Then go back to the communication program (type fg at the command line prompt in minicom) to see what happened.

Here's an example session for a certain Rockwell modem (C-a is ^A):

```
- Run minicom
- Type AT** : see "Download initiated ..."
- C-a J
- cat FLASH.S37 > /dev/modem
- fg : see "Download flash code ..."
- C-a J
- cat 283P1722.S37 > /dev/modem
- fg : see "Device successfully programmed"
```

20. Other Sources of Information

20.1 Misc

- man pages for: `agetty(8)`, `getty(1m)`, `gettydefs(5)`, `init(1)`, `isapnp(8)`, `login(1)`, `mgetty(8)`, `setserial(8)`
- Your modem manual (if it exists). Some modems come without manuals.
- Serial Suite by Vern Hoxie is a collection of blurbs about the care and feeding of the Linux serial port plus some simple programs.
- The Linux serial mailing list. To subscribe, send email to `majordomo@vger.rutgers.edu`, with ``subscribe linux-serial'' in the message body. If you send ``help'' in the message body, you get a help message. The server also serves many other Linux lists. Send the ``lists'' command for a list of mailing lists.

20.2 Books

I've been unable to find a good up-to-date book on modems.

- The Complete Modem Reference by Gilbert Held, 1997. Contains too much info about obsolete topics. More up-to-date info may be found on the Internet.

- Modems For Dummies by Tina Rathbone, 1996. (Have never seen it.)
- The Modem Technical Guide by Douglas Anderson, 1996.
- Ultimate Modem Handbook by Cass R. Lewart, 1998.
- Black, Uyless D.: Physical Layer Interfaces & Protocols, IEEE Computer Society Press, Los Alamitos, CA, 1996.

20.3 HOWTOs

- Cable-Modem mini-howto
- SuSE ISDN Howto (not a LDP Howto) http://sol.parkland.cc.il.us/sdb/en/html/isdn.html or http://brenner.chemietechnik.uni-dortmund.de/doc/sdb/en/html/isdn.html
- Linux-Modem-Sharing mini-howto. Computers on a network share a single modem for dial-out (like a shared printer).
- Modems-HOWTO: In French (Not used in creating this Modem-HOWTO)
- NET-3-4-HOWTO: all about networking, including SLIP, CSLIP, and PPP
- PPP-HOWTO: help with PPP including modem set-up
- Serial-HOWTO has info on Multiport Serial Cards used for terminals, etc. Covers the serial port in more detail than in this HOWTO.
- Serial-Programming-HOWTO: for some aspects of serial-port programming
- Text-Terminal-HOWTO: (including connecting up with modems)
- UUCP-HOWTO: for information on setting up UUCP

20.4 Usenet newsgroups

- comp.os.linux.answers; FAQs, How-To's, READMEs, etc. about Linux.
- comp.os.linux.hardware; Hardware compatibility with the Linux operating system.
- comp.os.linux.setup; Linux installation and system administration.
- comp.dcom.modems; Modems for all OS's

20.5 Old Modem Database on Internet

Rob Clark had a huge database of modems but it doesn't seem to have been maintained since 2003. The website URLs have changed many times. Right now (mid 2006), it seems that only these two mirrors work:

Modem List mirror1
Modem List mirror2

20.6 Other Web Sites

- Linux Serial Driver home page Includes info about support for PCI modems.
- Hayes AT modem commands Technical Reference for Hayes (tm) Modem Users
- AT Command Set and Register Summary for Analog Modem Modules (Cisco)
- Controlling your Modem with AT Commands
- Modem FAQs:
  Navas 28800-56K Modem FAQ
- Curt's High Speed Modem Page
- Much info on 56k modems 56k Modem = v.Unreliable
- Links to modem manufacturers
- More Links to modem manufacturers

- http://search.fcc.gov/ name="Search for manufacturer by FCC ID">

21. Appendix A: How Analog Modems Work (technical) (unfinished)

21.1 Modulation Details

Intro to Modulation

This part describes the modulation methods used for conventional analog modems. Two other types of modems, cable and ADSL, use the same modulation methods but the situation is more complicated since they divide their frequency spectrum into multiple channels, etc. Cable modems have to share a cable used by many other people. This HOWTO doesn't explain this added complexity of cable and ADSL modems.

Modulation is the conversion of a digital signal represented by binary binary (0 or 1) into an analog signal something like a sine wave. The modulated signal consists pure sine wave "carrier" signal which is modified to convey information. A pure carrier sine wave, unchanging in frequency and voltage, provides no flow of information at all (except that a carrier is present). To make it convey information we modify (or modulate) this carrier. There are 3 basic types of modulation: frequency, amplitude, and phase. They will be explained next.

Frequency Modulation

The simplest modulation method is frequency modulation. Frequency is measured in cycles per second (of a sine wave). It's the count of the number of times the sine wave shape repeats itself in a second. This is the same as the number of times it reaches it peak value during a second. The word "Hertz" (abbreviated Hz) is used to mean "cycles per second".

A simple example of frequency modulation is where one frequency means a binary 0 and another means a 1. For example, for some obsolete 300 baud modems 1070 Hz meant a binary 0 while 1270 Hz meant a binary 1. This was called "frequency shift keying". Instead of just two possible frequencies, more could be used to allow more information to be transmitted. If we had 4 different frequencies (call them A, B, C, and D) then each frequency could stand for a pair of bits. For example, to send 00 one would use frequency A. To send 01, use frequency B; for 10 use C; for 11 use D. In like manner, by using 8 different frequencies we could send 3 bits with each shift in frequency. Each time we double the number of possible frequencies we increase the number of bits it can represent by 1.

Amplitude Modulation

Once one understands frequency modulation example above including the possibilities of representing a few bits by a single shift in frequency, it's easier to understand both amplitude modulation and phase modulation. For amplitude modulation, one just changes the height (voltage) of the sine wave analogous to changing the frequency of the sine wave. For a simple case there could only be 2 allowed amplitude levels, one representing a 0-bit and another representing a 1-bit. As explained for the case of frequency modulation, having more possible amplitudes will result in more information being transmitted per change in amplitude.

Phase Modulation

To change the phase of a sine wave at a certain instant of time, we stop sending this old sine wave and immediately begin sending a new sine wave of the same frequency and amplitude. If we started sending the new sine wave at the same voltage level (and slope) as existed when we stopped sending the old sine wave, there would be no change in phase (and no detectable change at all). But suppose that we started up the new sine wave at a different point on the sine wave curve. Then there would likely be a sudden voltage jump at the point in time where the old sine wave stopped and the new sine wave began. This is a phase shift and it's measured in degrees (deg.) A 0 deg. (or a 360 deg.) phase shift means no change at all while a 180 deg. phase shift just reverses the voltage (and slope) of the sine wave. Put another way, a 180 deg. phase shift just skips over a half-period (180 deg.) at the point of transition. Of course we could just skip over say 90 deg. or 135 deg. etc. As in the example for frequency modulation, the more possible phase shifts, the more bits a single shift in phase can represent.

Combination Modulation

Instead of just selecting either frequency, amplitude, or phase modulation, we may chose to combine modulation methods. Suppose that we have 256 possible frequencies and thus can send a byte (8 bits) for each shift in frequency (since 2 to the 8 power is 256). Suppose also that we have another 256 different amplitudes so that each shift in amplitude represents a byte. Also suppose there are 256 possible phase shifts. Then a certain points in time we may make a shift in all 3 things: frequency, amplitude and phase. This would send out 3 bytes for each such transition.

No modulation method in use today actually does this. It's not practical due to the relatively long time it would take to detect all 3 types of changes. The main problem is that frequent shifts in phase can make it appear that a shift in frequency has happened when it actually didn't.

To avoid this difficulty one may simultaneous change only the phase and amplitude (with no change in frequency). This is called phase-amplitude modulation. It is also called quadrature amplitude modulation (= QAM) since there were only 4 possible phases (quadrature) in early versions of it. This method is used today for the common modem speeds of 14.4k, 28.8k, and 33.6k. The only significant case where this modulation method is not used today is for 56k modems. But even 56k modems exclusively use QAM (phase-amplitude modulation) in the direction from your PC out the telephone line. Sometimes even the other direction will also fall back to QAM when line conditions are not good enough. Thus QAM (phase-amplitude modulation) still remains the most widely used method on ordinary telephone lines.

21.2 56k Modems (V.90, V.92)

The "modulation" method used for speeds above 33.6k is entirely different than the common phase-amplitude modulation used at 33.6k and below. Since ordinary telephone calls are converted to digital signals at the local offices of the telephone company, the fastest speed that you can send digital data by an ordinary telephone call is the same speed that the telephone company uses over its digital portion of its network (for a phone call). What is this speed? Well, it's close to 64kbps. It's sometimes 64k and sometimes less if bits are "stolen" for signalling purposes. If the phone Co. knows that the link is not for voice, bits may not get stolen. The case of 64k will be presented and then it will be explained why the actual speed is lower (56k or less --often significantly less).

Thus 64k is the absolute top speed possible (not counting date compression) for an ordinary telephone call using the digital portion of the circuit that was designed to send digital encodings of the human voice. In order to use 64k, the modems need to either have direct access to the digital portion of the circuit or be able to

determine the exact digital signal that generated a received analog signal (and conversely). This task is far too error prone if both sides of a telephone call have only an analog interface to the telephone company. But if one side has a digital interface, then it's possible (in one direction for V.90 and in both directions for V.92). Thus if your ISP has a digital interface to the phone company, the ISP may send out a certain digital signal over the phone lines toward your PC. The digital signal from the ISP gets converted to analog at the local telephone office near your PC's location (perhaps near your home). Then it's your modem's task to try to figure out exactly what that digital signal was. If it could do this, then transmission at 64k (the speed of the telephone company's digital signal) is possible in this direction.

What method does the telephone company use to digitally encode analog signals? It uses a method of sampling the amplitude of the analog signal at a rate of 8000 samples per second. Each sample amplitude is encoded as a 8-bit byte. (Note: 8 x 8000 = 64k) This is called "Pulse Code Modulation" = PCM. These bytes are then sent digitally on the telephone company's digital circuits where many calls share a single circuit using a time-sharing scheme known as "time division multiplexing". Then finally at a local telephone office near your home, the digital signal is de-multiplexed resulting in the same digital signal as was originally created by PCM. Then this signal is converted back to analog and sent to your home. This analog to digital conversion (and conversely) is done by telephone company hardware called a "codec" (coder/decoder). Each PCM 8-bit byte creates a certain amplitude of the analog signal. Your modem's task is to determine just what that PCM 8-bit byte was, based on the analog amplitude it detects.

This was originally called "modulus conversion". It's now often called "PCM"-something (such as PCM modulation) since its just like encoding/decoding PCM but with the added problem of sampling at the precise time that the codec generated the analog voltage from the digital PCM code.

In order to determine the digital codes the telephone Co. used to create the analog signal, the modem must sample this analog signal amplitude at exactly the same points in time the phone Co. did when it created the analog signal. To do this an 8kHz clock timing signal is generated with help from a residual 4kHz signal on the analog phone line. The creation of amplitudes to go out to your home/office at 8k amplitudes/sec sort of creates a 4kHz signal. Suppose every other amplitude was of opposite polarity. Then there would be a 4kHz sine-like wave created. Each amplitude is in a sense a 8-bit symbol and when to sample amplitudes is known as "symbol timing". The modem's task is to insure that it's 8kHz clock runs at precisely twice the speed of the 4kHz signal (which could drift slightly off 4kHz) and that the modem's clock is synchronized with that used by the telephone company's codec. The actual electronics may use much higher frequency clocks (dividing them down) and take more than a single sample. If you know how this synchronization works, let me know (if this is a recent Modem-HOWTO).

Now the encoding of amplitudes in PCM is not linear. At low amplitudes an increment of 1 in the PCM byte value represents a much smaller increment (delta) in analog signal amplitude than would be the case if the amplitude being sampled were much higher. Thus for low amplitudes it's difficult to distinguish between adjacent byte values. To make it easier to do this (for 56k modems) certain PCM codes representing very low amplitudes are not used. This gives a larger delta between possible amplitudes and makes correct detection of them by your modem easier. Thus half of the amplitude levels are not used (in the downstream direction) by V.90 or V.92. This is tantamount to each symbol (valid amplitude level) representing 7 bits instead of 8. This is where 56k comes from: 7 bits/symbol x 8k symbols/sec = 56k bps. Of course each amplitude symbol is actually generated by 8-bits but only 128 bytes of the possible 256 bytes are actually used by the ISP sender. There is a code table mapping these 128 8-bit bytes to the 128 7-bit bytes. It's not just a simple mapping like ignoring the last bit. Thus to send 7 normal data bytes (8-bits) will take 8 of the above mentioned bytes.

But it's a little more complicated that this. If the line conditions are not nearly perfect or if the direction is upstream (V.92 only), then even fewer possible levels (symbols) are used resulting in speeds under 56k. Also due to US government rules prohibiting high power levels on phone lines, certain high amplitudes levels can't

be used resulting in only about 53.3k at best for "56k" modems in the downstream direction.

Note that the digital part of the telephone network is bi-directional. Two such circuits are used for a phone call, one in each direction. For V.90, the 56k signal is only used in one of these directions: from your ISP to your PC (called the "downstream" direction). For this V.90, the other direction (upstream, from your home/office to the ISP) uses the conventional phase-amplitude modulation scheme with a maximum of 36.6kbps (and not 53.3kbps). For V.92, this upstream direction also uses the PCM method and supports up to 48 kbps. The analog portion of the circuit from your home/office to the nearest telephone Co. office was never intended to be bi-directional since it's only a single twisted pair. But due to sophisticated cancellation methods it's able to convey data simultaneously in both directions as explained in the next subsection. It's claimed that with V.92, it's almost impossible to get maximum thruput in both directions simultaneously due to the difficulties of bi-directional flow on a single circuit.

21.3 Full Duplex on One Circuit

Modern modems are able to both send and receive signals simultaneously. One could call this "bidirectional" or "full duplex". This was once done by using one frequency for sending and another for receiving. Today, the same frequency is used for both sending and receiving. How this works is not easy to comprehend.

Most of the telephone system "main lines" are digital with two channels in use when you make a telephone call. What you say goes over one digital channel and what the other person says goes over the other (reverse) digital channel. Unfortunately, the part of the telephone system which goes to homes (and many offices) is not digital but only a single analog channel. If both modems were directly connected to the digital part of the phone system then bidirectional communication (sending and receiving at the same time) would be no problem because two channels would be available.

But the end portion of the signal path goes over just one circuit. How can there be two-way communication on it simultaneously? It works something like this. Suppose your modem is receiving a signal from the other modem and is not transmitting. Then there's no problem. But if your modem were to start transmitting (with the other received signal still flowing into your modem) it would drown out the received signal. If the transmitted signal was a "solid" voltage wave applied to the end of the line then there is no way any received signal could be present at that point.

But the transmitter has "internal impedance" and the transmitted signal applied to the end of the line is not solid (or strong enough) to completely eliminate the received signal coming from the other end. Thus while the voltage at the end of the line is mostly the stronger transmitted signal a small part of it is the desired received signal. All that is needed is to filter out this stronger transmitted signal and then what remains will be the signal from the other end which we want. To do this, one only needs to get the pure transmitted signal directly from the transmitter (before it's applied to the line) amplify it a determined amount, and then subtract it from the total signal present at the end of the line. Doing this in the receiver circuits leaves a signal which mostly came from the other end of the line.

21.4 Echo Cancellation

An analog signal traveling down a line in one direction may encounter changes in the line that will cause part of the signal to echo back in the opposite direction. Since the same circuit is used for bi-directional flow of data, such echos will result in garbled reception. One way to ameliorate this problem is to send training signals once in a while to determine the echo characteristic of the line. This will enable one to predict the echos that will be generated by any given signal. Then this prediction method is used to predict what echos the transmitted signal will cause. Then this predicted echo signal is subtracted from the received signal. This

cancels out the echoes.

22. Appendix B: Analog Voice Infeasible Over Non-Voice Modem

Sometimes people look for software that will allow them to transmit ordinary analog voice over a modem that doesn't support voice. It doesn't exist since it's just not very feasible to do this. Of course, one can use VOIP to send digital voice over a modem. And when one makes a call and the other side picks up the line, one might be able to hear voice for a few seconds until negotiations for a connection begin.

But once a modem is connected, sending analog voice over it just isn't feasible. For phase-amplitude modulation, carrier frequencies of fixed values are used which doesn't allow the continuously variable frequencies required in analog voice. V.90 and V.92 might be feasible, but if line conditions deteriorate, they fall back to phase-amplitude modulation which won't work. Furthermore, V.90 uses phase-amplitude in one direction. Also, both V.90 and V.92 don't permit all amplitudes to be used, which limits the waveshapes which can be created.

23. Appendix C: "baud" vs. "bps"

23.1 A simple example

``baud" and ``bps" are perhaps one of the most misused terms in the computing and telecommunications field. Many people use these terms interchangeably, when in fact they are not! bps is simply the number of bits transmitted per second. The baud rate is a measure of how many times per second a signal changes (or could change). For a typical serial port a 1-bit is -12 volts and a 0-bit is +12 v (volts). If the bps is 38,400 a sequence of 010101... would also be 38,400 baud since the voltage shifts back and forth from positive to negative to positive, etc. and there are 38,400 shifts per second. For another sequence say 111000111... there will be fewer shifts of voltage since for three 1's in sequence the voltage just stays at -12 volts yet we say that its still 38,400 baud since there is a possibility that the number of changes per second will be that high.

Looked at another way, put an imaginary tic mark separating each bit (even though the voltage may not change). 38,400 baud then means 38,400 tic marks per second. The tic marks at the instants of permitted change and are actually marked by a synchronized clock signal generated in the hardware but not sent over the external cable.

Suppose that a "change" may have more than the two possible outcomes of the previous example (of +- 12 v). Suppose it has 4 possible outcomes, each represented by a unique voltage level. Each level may represent a pair of bits (such as 01). For example, -12v could be 00, -6v 01, +6v 10 and +12v 11. Here the bit rate is double the baud rate. For example, 3000 changes per second will generate 2 bits for each change resulting in 6000 bits per second (bps). In other words 3000 baud results in 6000 bps.

23.2 Real examples

The above example is overly simple. Real examples are more complicated but based on the same idea. This explains how a modem running at 2400 baud, can send 14400 bps (or higher). The modem achieves a bps rate greater than baud rate by encoding many bits in each signal change (or transition). Thus, when 2 or more bits are encoded per baud, the bps rate exceeds the baud rate. If your modem-to-modem connection is at 14400 bps, it's going to be sending 6 bits per signal transition (or symbol) at 2400 baud. A speed of 28800 bps is

obtained by 3200 baud at 9 bits/baud. When people misuse the word baud, they may mean the modem speed (such as 33.6k).

Common modem bps rates were formerly 50, 75, 110, 300, 1200, 2400, 9600. These were also the bps rates over the serial_port-to-modem cables. Today the bps modem-to-modem (maximum) rates are 14.4k, 28.8k, 33.6k, and 56k, but the common rates over the serialPort-to-modem cables are not the same but are: 19.2k, 38.4k, 57.6k, 115.2k, 230.4k. The high speed of 230.4k is (as of late 2000) unfortunately not provided by most new (and old) hardware. Using modems with V.42bis compression (max 4:1 compression), rates up to 115.2k bps are possible for 33.6k modems. 203.2k (4 x 53.3k) is possible for 56k modems.

Except for 56k modems, most modems run at 2400, 3000, or 3200 baud. Even the 56k modems use these bauds for transmission and sometimes fall back to them for reception. Because of the bandwidth limitations on voice-grade phone lines, baud rates greater than 2400 are harder to achieve, and only work under conditions of good phone line quality.

How did this confusion between bps and baud start? Well, back when antique low speed modems were high speed modems, the bps rate actually did equal the baud rate. One bit would be encoded per phase change. People would use bps and baud interchangeably, because they were the same number. For example, a 300 bps modem also had a baud rate of 300. This all changed when faster modems came around, and the bit rate exceeded the baud rate. ``baud" is named after Emile Baudot, the inventor of the asynchronous telegraph printer. One way this problem gets resolved is to use the term "symbol rate" instead of "baud" and thus avoid using the term "baud". However when talking about the "speeds" between the modem and the serial port (DTE speed) baud and the symbol rate are the same. And even "speed" is a misnomer since we really mean flow rate.

24. Appendix D: Terminal Server Connection

This section was adapted from Text-Terminal-HOWTO.

A terminal server is something like an intelligent switch that can connect many modems (or terminals) to one or more computers. It's not a mechanical switch so it may change the speeds and protocols of the streams of data that go thru it. A number of companies make terminal servers: Xyplex, Cisco, 3Com, Computone, Livingston, etc. There are many different types and capabilities. Another HOWTO is needed to compare and describe them (including the possibility of creating your own terminal server with a Linux PC). Most are used for modem connections rather than directly connected terminals.

One use for them is to connect many modems (or terminals) to a high speed network which connects to host computers. Of course the terminal server must have the computing power and software to run network protocols so it is in some ways like a computer. The terminal server may interact with the user and ask what computer to connect to, etc. or it may connect without asking. One may sometimes send jobs to a printer thru a terminal server.

A PC today has enough computing power to act like a terminal server except that each serial port should have its own hardware interrupt. PC's only have a few spare interrupts for this purpose and since they are hard-wired you can't create more by software. A solution is to use an advanced multiport serial card which has its own system of interrupts (or on lower cost models, shares one of the PC's interrupts between a number of ports). See Serial-HOWTO for more info. If such a PC runs Linux with getty running on many serial ports it might be thought of as a terminal server. It is in effect a terminal server if it's linked to other PC's over a network and if its job is mainly to pass thru data and handle the serial port interrupts every 14 (or so) bytes. Software called "radius" is sometimes used.

Today real terminal servers serve more than just terminals. They also serve PC's which emulate terminals, and are sometimes connected to a bank of modems connected to phone lines. Some even include built-in modems. If a terminal (or PC emulating one) is connected directly to a modem, the modem at the other end of the line could be connected to a terminal server. In some cases the terminal server by default expects the callers to use PPP packets, something that real text terminals don't generate.

25. Appendix E: Cable and DSL modems

25.1 Introduction

This HOWTO only deals with the common type of analog modem used to connect PC's to ordinary analog telephone lines. There are also higher speed analog modems that use special types of lines: cable and DSL modems. There is also the ISDN "modem" which uses digital signals. While this HOWTO doesn't cover such modems, some links to documents that do may be found at the start of this HOWTO. The next 3 sub-sections: DSL, Cable, and ISDN, briefly discuss such modems. For both DSL and Cable modems, the basic QAM modulation method is similar to ordinary analog modems. See Combination Modulation

25.2 Digital Subscriber Line (DSL)

DSL (often ADSL) uses the existing twisted pair line from your home/office to the local telephone office. This can be used if your telephone line can accept significantly higher speeds than an ordinary modem would use. It replaces the analog-to-digital converter at the local telephone office with one which can accept a much faster flow of data (in a different format of course). The spectrum of the twisted pair line is divided up into various channels. Each channel uses QAM modulation like ordinary modems do. Data is sent over multiple channels. The device which converts the digital signals from your computer to the analog signal used to represent digital data on what was once an ordinary telephone line, is a DSL modem. The DSL modem is often external (takes up space on your desk) and connects to your computer via either an ethernet port or a USB port.

25.3 Cable Modems

The coaxial cables that provide for cable television in homes have additional bandwidth not used for television, mostly at frequencies higher than used for cable TV. This extra bandwidth may be used for connecting computers to ISP's. However, many computers need to share the same cable. The spectrum of the free bandwidth is split up into channels (frequency division multiplexing) and each channel is given time slots to which individual computers are assigned (time division multiplexing). The cable modem converts the digital date from your computer (from a network card: NIC) to the required analog signal, and only broadcasts within it's assigned time slots on it's assigned channel.

26. Appendix F: Connecting 2 Modems Directly Back-to-Back (Leased Lines).

While modems are designed to be connected to telephone lines which have dial tones and voltages on the line, it's claimed that most modems will communicate when just connected to each other back-to-back via a telephone cord. To get this working, type ATD for one modem to dial and ATA for the other modem to answer (or set it for auto answer). Since there will be no dialtone, you may need to set ATX (or whatever) to blind dial (force it to dial without a dialtone). If ATD doesn't work because of no phone number, you could try

ATD0 to send a 0.

If the above doesn't work, you could make a simple power supply to emulate a telephone line. See Connecting two computers using their modems, without a telephone line.

A line without a source of voltage is sometimes called a "dry line" and some modems exist which are designed to work on such lines (or which can be configured to work on such lines). If you connect one of these special modems to a line with voltage on it, it may destroy the modem.

A leased line is one which is usually leased from the telephone company. The end points of the line are under the control of the user who may connect a modem at each end of the line. Leased lines may or may not have a voltage supply on them. See the mini-howto: Leased-Line which covers leased lines where there is neither voltage nor dialtone on the line. One type of leased line used two pairs of wires (one for each direction) using V.29 modulation at 9600 baud. Some brands of leased line modems are incompatible with other brands.

27. Appendix G: Fax pixels (dots)

Here's some info on the bloated bandwidth required for standard fax including the dot density. You can of course send a fax via your modem if you dial the real telephone number of the recipient.

```
A4 paper:    216mm (horizontal) * 297mm (vertical)
normal mode       8dots/mm      * 3.85dots/mm
fine   mode                     * 7.7dots/mm
extra fine mode                 *15.4dots/mm
```

Each dot is either white or black and thus 1 bit. One sheet of A4 paper using fine mode is (216*8) * (297*7.7) = about 4 million dots. With a compression ratio of 8:1 it takes about 50 seconds at 9600bps for transmission.

28. Appendix H: Stty Hanging Problem (prior to 2000)

Here's a problem that existed prior to the year 2000 or thereabouts. It's since been fixed. If -clocal is set and there is no CD signal, then the "stty" command will hang and there is seemingly no way to set clocal to stop the hanging (except by running minicom). But minicom will restore -clocal when it exits. One way to get out of this is to use minicom to send the "AT&C" to the modem (to get the CD signal) and then exit minicom with no reset so that the CD signal always remains on. Then you may use stty again.

29. Appendix G: Antique Modems

29.1 Introduction

By "antique" I mean modems with speeds of under 56k speed (although they don't really run this fast). This appendix compares the antique modems with the modern ones. You should read it if you are interested in modem history or are intending to actually use an antique modem. Also, many modern modems and software still support the old protocols and you might find that such old protocols have been configured by mistake.

29.2 Historical Modem Protocols

- Bell 103 300 bps; frequency shift keying = FSK (1962)
- V.21 300 bps; frequency shift keying (used a different frequency than Bell 103) (1964)

- V.23 1200/75 bps and 600/75 bps asymmetric; 75 bps is the reverse channel; frequency shift keying = FSK (1964)
- Bell 212A 1200 bps; quadrature differential phase shift keying = QDPSK = DPSK (1963?)
- V.22 1200 bps; fallback to 600 bps ; QDPSK = DPSK (1980)
- V.22bis 2400 bps; QAM (1984)
- V.32 9600 bps; QAM (1984 but not widely used until years later)
- V.32bis 14.4k bps; QAM (1991)
- V.34 28.8k bps; QAM (1994) AKA V.fast
- V.34bis 33.6k bps; QAM (1996)
- V.90 56k bps; Modulus Conversion downstream, QAM upstream (1998)
- V.92 56k bps; Modulus conversion in both directions (2000)

"bis" means a second version of the standard.
QAM= Quadrature Amplitude Modulation. The word "Quadrature" is short for "quadrature differential phase shift keying" =QDPSK. "quadrature" mean 4 (quad) implying there are 4 possible phases differences, separated from each other by 90 degrees. Only the V.22 had 4 possible phases. After V.22, more possible phases were added so as to convey more information, but the name "quadrature" was still retained. It just means phase modulation.

The above table excludes historical standards for leased lines: Bell 201B,C (2400 bps) Bell 208A,B (4800 bps; 208B = V.27).

29.3 Historical Overview

Teletypes and dumb terminals

Prior to 1960, 110 bps modems were used for teletype machines (like an electric typewriter only much more noisy). What one typed at a teletype (or had saved on punched paper tape) could be printed on a remote teletype located far away. No computer was involved.

In 1960 AT$amp;T came out with a 300 bps modem (for use on it's phone system). Then in 1962 it started selling Bell 103 modems to the public. Such slow and expensive modems were later mainly used for transmitting data between mainframe computers or for connecting a dumb terminal to a mainframe computer over phone lines. Many dumb terminals didn't even have a screen display, but printed on paper what you typed at the keyboard along with responses from the computer.

PCs and BBSs

With the advent of the personal computer (PCs) in the early 1980s, one could use a modem to dial into a remote mainframe computer. In this case, the PC was used like a dumb terminal. But now files could be transferred and one PC could connect to another via modems.

The 1980s saw the rise of the Bulletin Board System (BBS). A BBS was just a computer with a modem listening for incoming calls. The public could dial up a BBS with a modem and then download free software, participate in discussions on various topics, play on-line games, etc. Dialing in to a BBS was something like going to an Internet site. Except that to go to another BBS site, you would need to dial another number (and possible pay long distance telephone charges). Many BBSs would have a monthly charge but some were run by volunteers and were free. Many companies established BBSs for customers that contained support information, catalogs, etc. In the early 1990s, BBSs were booming. By the mid 1990s some even offered Internet connections. For some history of BBSs see Sysops' Corner: History of BBSing

The Internet

Then came the advance of Internet in the mid 1990s which resulted in the demise of the BBSs near the end of the 1990s. Some BBSs became websites, but when BBSs were dying in droves, websites were quite expensive so most BBSs just disappeared. The Internet contained far more information than any one BBS could maintain so BBSs were no longer competitive.

Modems permitted the public to connect to the Internet. In the 1990s, Modems became fast, cheap and widely used. Then in the late 1990s, faster non-analog "modems" appeared: ISDN, DSL, and cable. The history of these isn't in this HOWTO.

Speeds

Before V.32 (9600 bps), modems typically had speeds of 300 to 2400 bps. Some super fast ones had much higher speeds (such as 19.2k bps) and used non-standard protocols. To utilize these "fast" ones, both modems for a connection needed to support the same proprietary protocol which often meant that they must be the same brand.

Prior to the V.42 standard for error correction and the V.42bis (1990) standard for data compression, the MNP standards were usually used for both error correction and data compression. An X.PC error correction standard was used on some commercial data networks. Compression and error correction were available on some 2400 bps modems.

From 1960 to 1980 most modems only had a speed of 300 bps (which was also 300 baud). This is only 0.3kbps. Modern modems are over 100 times faster. Some old-slow modems are still in use so they are not really "antique" quite yet.

29.4 Proprietary protocols, etc.

These did not conform to the ITU V-series standards. They were used in order to obtain higher speeds before more standardized higher speeds became established. The modem at the other end needs to support the same protocol for this to work. The dates shown below may be only approximate.

- PEP (Packetized Ensemble Protocol 1985): 18k (at best).
- Turbo PEP: 23k 1994?
- Hayes Express 96: 9.6k (Hayes 1987)
- HST: 9.6k (US Robotics 1986)
- HST: 14.4k (US Robotics 1989)
- HST: 16.8k (US Robotics 1992)
- V.32 terbo): 19.2k (AT$amp;T 1993) (V.32ter was rejected as a standard)
- V.FastClass: 28.8k (Rockwell 1993) (V.FastClass is not an ITU standard)
- X2 :57.3k (US Robotics 1997)
- K56: Flex 57.3k (Rockwell 1997)

The PEP used as much bandwidth as feasible by splitting the spectrum into as many as 512 sub-bands. It was supported by Ven-Tel's Pathfinder and Telebit's Trailblazer.

29.5 Autobauding

This term has a few different meanings. In general it means either the automatic adjustment of modem-to-modem speed or of modem-to-serial_port speed.

29.6 Modem-to-modem Speed

Modern modems negotiate the modem-to-modem speed and protocol when they first connect to each other and normally connect to each other at the highest possible speed. If one side can't negotiate, the other side should accept whatever speed and protocol that the fixed side has available. Except that some modern modems may no longer support some of the antique protocols. As a result of negotiations, one modem may need to use a lower speed than its maximum in order to communicate with the other modem. This is sometimes called "autobauding" or "automode". It might also be called "fallback". A more intuitive use of the word "fallback" is where both modems automatically lower their speed due to a noisy line. While autobauding is normally the default, setting a fixed modem-to-modem speed instead of autobauding is done with either an AT command like or by a register like S37.

Early modems didn't have such autobauding or fallback. If you have such a modem, it will likely work OK if the other modem you connect to is a modern one that can adjust it's speed and protocol to yours. But a problem arises if both modems which want to communicate with each other are both antique and don't support automode. In this case they need to be manually set to the same speed and protocol.

Even when this automode existed, there was sometimes a limited choice of speed (like only 1200/300 bps). In olden days (rarely today), a computer dial-in site might have one phone number a certain speed (like say 2400) and another phone number for a different speed (say 1200). It was often not quite this simple as there might be a few different phone number for one speed, or one phone number might support say only a couple of speeds (like 1200/300). Also one phone number might support only a certain protocol such as the Bell 212A modem. Once a site obtained modems that could support a wider variety of speeds and protocols, then there was no need to have different groups of phone lines.

29.7 Modem-to-serial_port Speed

Same speed required

For old modems (mostly under 9600 bps) the modem-to-serial_port speed had to be the same as the modem-to-modem speed. This was because data flowed straight thru the modem without "speed buffering" (storing bytes) inside the modem. This meant that both the modem's serial port and the computer's serial port had to be set to this speed. That is, both ends of the serial cable had to be set for this speed.

One might erroneously reason that if the serial port speed was higher than the modem-to-modem speed, all would work OK since then there would be no bottleneck in the serial line. This works OK in the direction from the modem to the PC since a higher speed line can have a lower thruput speed due to greater time spacing between bytes. But disaster strikes for the flow from the PC to the modem since it would flow into the modem at a speed faster than the modem could transmit the data. Data would be lost since there is no speed buffering.

Equalizing speed

If a modem had only one modem-to-modem speed (or was set by software or physical switches to only operate at one speed), then this wasn't a problem since one would just set the PCs serial port for this speed. And the modem would set it's serial port for this speed. Another way make the speeds equal is for the modem to detect the PC's serial port speed and then set both it's serial speed and it's modem-to-modem speed the same. This will be explained later.

But for modems that used more than one modem-to-modem speed there's a problem. In that case it's not known in advance at what speed the modem will connect to another modem. For example, if a fixed speed modem dialed in to you modem and your modem supported the remote modems fixed speed, then that's the speed it would connect at. If the PC's serial port speed had been previously set to a different speed, then this speed needs to be changed.

But setting the computer's serial port to the modem-to-modem speed was a problem since the modem has no way to directly give commands to the PCs serial port to change it's speed. Only system software can do that. The modem finds out what speed to use based on negotiations with the other modem and thus the change of this serial port speed can't be done in advance. How does the modem communicate its modem-to-modem speed to the system software?

Use "CONNECT" message to set speed

Here's one way to do it. Consider the case of a dial-in modem that others dial into. A getty program will be used to send a login prompt thru the modems to a user's terminal screen. Getty will also be the system software that changes the speed of the serial port. The modem will tell getty what modem-to-modem speed it's using by sending getty a "CONNECT" message giving the modem-to-modem speed (such as "CONNECT 2400").

But there's one problem here. How does one insure that the "CONNECT" message, which the modem sends to getty via the serial port, is sent at the same speed as the PC's serial port which is expecting a "CONNECT" message? Here's how it's done.

When the modem is first sent an init string, the modem detects the speed of the computer's serial port and sets it's modem-to-serial_port speed to this value. Now it can communicate with getty and the "CONNECT" message can get thru.

To sense the speed the modem has examined the "AT" at the beginning of the string. This is sometimes also called autobauding. For modern modems, this same modem-to-serial port speed is always retained, even after the modem connects to another modem and regardless of what the modem-to-modem speed is. But for our old modem this initial serial speed may need to be changed later to that of the modem-to-modem speed once a connection is made.

For example, getty gets a CONNECT 2400 from the modem and switches the PC's serial port speed to 2400. The modem also switches its serial port speed to 2400. Now both ends of the modem serial cable are at 2400 and there is no speed mismatch. Then getty sends a login prompt out over the phone line thru the modems. The 2400 bps is now both the modem-to-modem speed and the modem-to-serial_port speed. Problem solved. Mgetty can do this by configuring it for "autobauding" but it's only of use for very old (and slow) modems.

For dialing out, the same method is used, but now the communication software must handle it instead of getty.

Setting modem-to-modem speeds by the serial speed

Another way to switch modem-to-modem speed was by using a modem feature where the modem would set its modem-to-modem speed to be the same as the modem-to-serial port speed it detected. The Bn AT commands would enable this and determine what protocol to use for each speed. So with this enabled, setting the serial speed by the computer would also set the modem-to-modem speed to be the same. This should result in this modem being inflexible in any speed negotiations between the modems.

Manual bauding

Another (but cruder) way to solve the serial speed problem when dialing in to a site was to get the remote site to change it's modem-to-modem speed to match your serial port speed. It works like this: The person trying to login over a modem connection doesn't see any login prompt because of a speed mismatch. So the person trying to login hits a "break" key to send a break signal over the phone line (via modem) to getty on the remote machine. A break signal will always get through even if there is a speed mismatch in a serial line.

The remote getty gets this break signal and switches the remote's serial port to the next speed as specified in its getty configuration file. This new remote serial port speed causes the remote modem to switch to the same modem-to-modem speed as previously configured by the ATB command. Then the local modem would transmit the login prompt over the local's serial line at this speed. If one doesn't see a login prompt, then they hit the break key again and a new speed is tried. This continues until the remote getty finally gets the speed correct (equal to the serial speed set on the local PC) and a login prompt finally displays. Note that PC keyboards have no "break" key but dumb terminal keyboards did. Mgetty, agetty, and uugetty can do this obsolete break method and it's called "manual bauding".

Unsupported speeds

In Linux, there's a problem if the speed is set to a speed not supported by Linux's serial port (for example 7200 bps). You may dial out and connect at 7200 bps (both modem-to-modem and modem-to-serial_port speed) but you only see garbage since Linux doesn't support 7200 on the serial port. Once you connect there is no simple way to hang up because even the +++ escape sequence can't be sent to the modem over a 7200 baud interface.

Modern modems, speed buffering

To dial out by the antique method using some modern modems set &Q0 N0 and S37=5 (if you want 1200 bps). Some of the S17 settings vary with the make of modem. S37=0 is the default that connects the modern way at the highest speed supported.

Modern modems can use almost any serial port speed. It doesn't depend at all on modem-to-modem speed. To do this, they employ speed buffering and flow control. Speed buffering means that modems have buffers so that there can be a difference between the modem-to-modem speed and the modem-to-serial_port speed. If the flow entering the modem is faster than the flow exiting it, the excess flow is simply stored in a buffer in the modem. Then to prevent the buffer from overflowing, the modem sends a flow control signal to stop the input flow to it. This is true for either direction of flow. See Flow Control for more details.

29.8 Before AT Commands

Hayes introduced the AT command set and other modem manufacturers adopted it as a standard. Before the AT commands, many modems used dip switches to configure the modem. Another command set is the CCITT V.25bis command set. Some modems supported both CCITT and AT commands. The CCITT V.25bis also specifies how Synchronous modem-to-serial_port communication is to take place using either the ASCII or 8-bit EBCDIC character sets.

29.9 Acoustic-Coupling

This is where one connects a modem to a telephone using audio tones that one can hear. The modem contains a microphone and speaker which "talks" directly into the telephone handset without using any wires. It's "wireless" in a sense but uses sound waves instead of radio waves. The modem speaker is placed in contact with the telephone microphone (on the handset) so that the tones from the modem go into the telephone. The modem microphone, picks up tones from the telephone handset speaker. This scheme is called "acoustic coupling".

A major problem is that outside noises can interfere and cause errors. The advantage is convenience: There are no cables to plug in. Most modems that could do this were only 300 baud, but higher speeds were used too. It's said that 9600 bps didn't work very well using this scheme.

29.10 Data Compression and Error Correction

MNP 2, 3, or 4 were used for error correction. MNP 5 was compression. Modern modems generally use V42 (error correction) and V42bis (compression). Many modems support both MNP and V42.

29.11 Historical Bibliography

PC Today, Sept. 2004, v.2 #9, pp 110-111: "The Rise & Fall of Dial-Up"

END OF Modem-HOWTO