The Linux Modem−HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

The Linux Modem−HOWTO

David S.Lawyer mailto:dave@lafn.org

v0.10, May 2000

Help with selecting, connecting, configuring, trouble−shooting, and understanding modems for a PC. See Serial−HOWTO for multiport serial boards.

1.Introduction

2.Modems for a Linux PC

3.Modem Pools, Digital Modems

4.Modem & Serial Port

5.Configuring Overview

6.Configuring the Serial Port

7.Modem Configuration (excluding serial port)

8.Serial Port Devices /dev/ttyS2, etc.

9.Interesting Programs You Should Know About

10.Trying Out Your Modem (Dialing Out)

11.Dial−In

12.Uugetty for Dial−In (from the old Serial−HOWTO)

13.What Speed Should I Use with My Modem?

14.Communications Programs And Utilities

15.What Are UARTs? How Do They Affect Performance?

16.Troubleshooting

17.Flash Upgrades

18.Other Sources of Information

1.Introduction

1.1 DSL, Cable, and ISDN Modems in other HOWTOs

This HOWTO covers conventional modems for PC's, mainly modems on the ISA bus. However each new version sees more info added about modems for the PCI bus.

- DSL modems: see the mini−howto: ADSL
- Cable−Modems−HOWTO (was once a LDP mini−Howto) http://www.cs.unm.edu/~vuksan/linux/Cable−Modem.html
- ISDN Howto (not a LDP Howto) http://sdb.suse.de/sdb/en/html/isdn.html: drivers for ISDN "Modems". Much related info on this is in German. For a tutorial on ISDN see http://public.swbell.net/ISDN/overview.html

See also Appendix D: Other Types of Modems

1.2 Also not covered: PCMCIA Modems, PPP

For modems on the PCMCIA bus see the PCMCIA−HOWTO: PCMCIA serial and modem devices. This HOWTO doesn't cover PPP (used to connect to the Internet via a modem) or communication programs. Except it does show how to use communication programs to test that your modem works OK and can make phone calls. If you want to use a modem to connect to the Internet then you need to set up PPP. There's a lot of documentation for PPP (including a PPP−HOWTO which is being revised). Some of it might be found in /usr/doc/ppp or the like.

1.3 Copyright, Disclaimer, Trademarks, & Credits

Copyright

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) 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 requires MS−Windows and not in the trademark sense.

Credits

The following is only a rough approximation of how this this document (as of 2000) was created: About 1/4 of the material here was lifted directly from Serial−HOWTO v. 1.11 by Greg Hankins. mailto:gregh@cc.gatech.edu (with his permission). About another 1/4 was taken from that Serial−HOWTO and revised. The remaining 1/2 is newly created by the author: David S. Lawyer mailto:dave@lafn.org.

1.4 Contacting the Author

Please don't email me asking which modem to buy or asking if a certain modem will work under Linux. Look at the huge list at Software (Internal) Modems Also, please don't ask me how to configure a modem unless you've looked over this HOWTO and still can't do it.

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 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 come out every month or so since modem situation is rapidly changing (and since I'm still learning). 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://metalab.unc.edu/LDP/mirrors.html If you only want to quickly check the date of the latest version go to http://metalab.unc.edu/LDP/HOWTO/Modem−HOWTO.html and compare it to the version you are currently reading: v0.10, May 2000

1.6 New in this Version

Modem−Sharing mini−howto, digital modems, Newcom modem, more re "no response to AT" and "can't find modem".

1.7 What is a Modem ?

A modem is a device that lets one send digital signals over ordinary telephone lines not designed for digital signals. If telephone lines were all digital then you wouldn't need a modem. It 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 (which includes such a program) 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 their computer. This is called "dial−in".

There are two basic types of modems for a PC: external and internal. The external sets on your desk outside the PC while the internal is not visible since it's inside the PC. The external modem plugs into a connector on the back of the PC known as a "serial port". The internal modem is a card that is inserted inside the computer and has an (invisible) serial port built into it. For a more detailed comparison see External vs. Internal. Thus when you get an internal modem, you also get a dedicated serial port (which can only be used with the modem and not with anything else such as another modem or a printer). In Linux, the serial ports are named ttyS0, ttyS1, etc. (usually corresponding respectively to COM1, COM2, etc. in Dos/Windows).

The serial port is not to be confused with the "Universal Serial Bus" (USB) which uses a special modular connector and may be used with modems in the future. See Modem & Serial Port Basics for more details on modems and serial ports.

Modems often 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.

1.8 Quick Install

External Modem Install

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 (on ISA bus)

(For the PCI bus see PCI Bus Support Underway and PCI Modems.) If the modem says it will only work under MS Windows, you may be out of luck. If you already have 2 serial ports, make the modem the 3rd serial port (ttyS2 = COM3). Find an unused IRQ number to use. In the past IRQ 5 was often used but today

IRQ 5 is also used for sound cards. Then set the jumpers (or the like) on the internal modem to the unused IRQ and IO address such as 3E8 for ttyS2.

"Or the like" (in the previous sentence) may be a bit tricky. If the modem is a Plug and Play (PnP) for the ISA bus, the equivalent probably can be done using the "isapnp" program which comes with "isapnptools". See "man isapnp" or the FAQ for it. See also "Plug−and−Play−HOWTO. With a PnP−BIOS you may be able to tell the CMOS setup menu that you don't have a PnP OS and then the BIOS may set a suitable IRQ and IO address in the modem card. If you want to "force" the BIOS to set a certain IRQ and/or IO then you may be able to do this using Windows9x on the same PC. 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. There may even be a way to disable PnP using software (under Windows) that came with the modem.

Finally you must also find the file where "setserial" is run and add a line something like: "setserial /dev/ttyS2 irq5". Except for setserial v2.15 and later you may (if you distribution lets you) just run "setserial" on the command line and the results are saved to a configuration file. See What is Setserial for more info. See the next subsection All Modems for further instructions on quick installation.

All Modems

Plug the modem into a telephone line. Then start up a communication program such as minicom and go to the configuration menu for the serial port. Assign it a high baud rate a few times higher than the bit rate of your modem. See Speed Table for the "best" speeds to use. Tell it the full name of your serial port such as /dev/ttyS1. Set hardware flow control (RTS/CTS). Now you need to save these settings and exit minicom. Then start minicom again, type AT to see if your modem is there and responds with OK. Then go to the dial directory (or menu) and dial a number.

2.Modems for a Linux PC

2.1 External vs. Internal

A modem for a PC may be either internal or external. The internal one is installed inside of your PC (you must remove screws, etc. to install it) and the external one just plugs into a serial port connector on a PC. Internal modems are less expensive, are less likely to to suffer data loss due to buffer overrun, usually use less electricity, and use up no space on your desk.

External modems are much easier to install and require less configuration. They 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.

Unfortunately most external 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 costs you about $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 k (although as of late 1998 most new internal modems don't either −−but some do). If a new internal modem had a 16650 UART it would put less load on the CPU (but almost none do as of late 1998).

Internal modems present a special problem for Linux, but will work just as well as external modems provided you avoid the high percentage of them that will work only for MS Windows, and also provided that you spend time (sometimes a lot of time) to configure them correctly. Some of the modems which will work only under MS Windows are, unfortunately, not labeled as such. If you buy a new one, make sure that you can return it for a refund if it will not work under Linux.

While most new modems are plug−and−play you have various ways to deal with them:

- Use the "isapnp" program
- Have a PnP BIOS do the configuring
- Patch the kernel to create a PnP Linux (not currently available)

Each of the above has shortcomings. Isapnp documentation is difficult to understand although reading the Plug−and−Play−HOWTO (at present incomplete) will aid in understanding it. If you want the PnP BIOS to do the configuring, all you need to do is to make sure that it knows you don't have a PnP operating system. But it may not do it correctly. To find out what it's done see What is set in my serial port hardware?. Patching the kernel has worked in the past but no patch seems to be currently available. Check out the website for it.

There are many Linux users that say that it's a lot simpler just to get an external modem and plug it in. But since new peripherals are mostly PnP today, you may eventually need to deal with it, so why delay the inevitable? Still, the most expedient (and expensive) solution is an external modem (if you have a free serial port).

2.2 External Modems

PnP External Modems

Many external modems are labeled "Plug and Play" (PnP) but they should all work fine as non−PnP modems. Since you usually plug the modem into a serial port which has its own IRQ number and IO address, the modem needs no PnP features to set these up. However, the serial port itself may need to be configured (IRQ number and IO address) unless the default configuration is OK.

How can an external modem be called PnP since it can't be configured by PnP? Well, it 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 model number. Then you might not need to configure application programs by telling them what port the modem is on (such as /dev/ttyS2 or COM3). But since you don't have such a PnP operating system you will need to configure your application program manually by giving it the /dev id (such as /dev/ttyS2).

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. Modems use straight through cable, with no pins crossed over. Most computer stores should have these. Make sure you get the correct gender. If you are using the DB9 or DB25 serial port at your computer, it will always be male which means that the connector on the cable should be female. 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

- 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)

2.3 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 the ISA slots and others for the PCI 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 (via IO address space on the ISA bus inside the computer). Such configuration commands can be sent by a PnP BIOS, the isapnp program (for the ISA bus only) or by a PnP operating system. The configuring of them is built into Windows 95/98 OSs. Under Linux you have a choice of ways (none of which is always easy) to io−irq configure them:

1. Use "isapnp" which may be run automatically at every boot−time
2. Use a PnP BIOS alone (which runs at every boot−time)
3. Patch Linux to make it a PnP operating system

2.4 Software (Internal) Modems (mostly winmodems)

Software modems turn over much (or even almost all) of the work of the modem to the main processor (CPU) chip of your computer (such as a Pentium chip). Complex proprietary software programs (drivers) do this work on the CPU. A majority of internal modems made after about mid−1998 *don't* work with Linux since they are software modems which only work under Windows and are often called "winmodems". Although a few volunteers were willing to try writing Linux drivers for these modems, specs were not made available so this couldn't be done. Prior to about 2000, no software modem could be used with Linux due to no drivers for them under Linux.

Then finally in late 1999 two software modems appeared that could work under Linux. Lucent Technologies unofficially released a Linux binary−only code to support its PCI software modems but bugs were reported in early versions. PC−TEL introduced a new software modem for Linux. Will other companies follow these leads and thus create "linmodems"? For a list of modems which work/don't_work under Linux see modem list. A project to get winmodems to work under Linux is at http://linmodems.org. They also have a mailing list. There is some effort underway at reverse−engineering with at least one report of a winmodem that has been made to work under Linux (but not yet with full functionality). So by the time you read this there may be more linmodems.

If code is made available to operate a "winmodem" under Linux, then one may call it a "linmodem". Is it still a "winmodem"? Perhaps it is since it also works under MS Windows. The term "Winmodem" is a trademark for a certain type of "winmodem".

Here is some more precise terminology regarding software modems. HSP (Host Signal Processor) means that the host processor (your CPU chip) creates the code needed to produce the electrical signal on the phone line. The modem itself just creates whatever electrical waveshape the CPU tells it to. In contrast to this, a "controllerless" modem can create the waveshapes on its own (but can't control the modem). It contains no facilities to deal with bytes being sent and received. It can't compress strings of bytes; it can't check for errors; it can't put them into packets. In other words it can't control the modem but instead has the CPU do all this work using software. The Rockwell HCF (Host Controlled Family) does this. If the software that does all this could be ported to Linux and then there wouldn't be this problem. Besides the above, a modem which doesn't simulate a serial port will not work under Linux.

How do you determine if an internal modem will work under Linux? First see if the name or description of it indicates it's a software modem: HSP, HCF, HSF, controllerless, host−controlled, host−based, and soft−... modem. If it's a software modem it will only work for the rare cases (so far) where a Linux driver is available. 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 check out the modem list on the Web mentioned 4 paragraphs above. If the above doesn't work (or isn't feasible), you can look at the package it came in (or a manual) find the section on the package that says something like "Minimum System Requirements" or just "System Requirements". It may be in fine print. Read it closely. If Windows or a Pentium CPU is listed as one of the requirements then it will likely not work under Linux.

Otherwise, it may work under Linux 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. I once saw a web−page that specifically stated that one model worked under Linux while implying that another model didn't.

Besides the problems of getting a driver, what are the pros and cons of software modems? Since the software modem uses the CPU to do much of its work, the software modem requires less on−board electronics and thus costs less. At the same time, the CPU is heavily loaded by the modem which may result in slower operation. This is especially true if 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 cost savings worth it? In some cases yes, especially if you seldom use the modem or are not running any other CPU intensive tasks when the modem is in use. Thus there are cases where use of a software modem is economically justified. 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 modem can do the job much more efficiently than a general purpose CPU. So if you use the modem a lot it's probably better to avoid software modems (and then you can use a less powerful CPU :−).

2.5 PCI Modems

A PCI modem card is one which inserts into a PCI−bus slot on the motherboard of a PC. Unfortunately, it seems that most PCI modems will not work under Linux but efforts are underway to support some of them. See PCI Bus Support Underway

2.6 Which Internal Modems might not work with Linux

- Software (Internal) Modems only work in rare cases where a Linux driver is available.
- PCI Modems most don't work yet under Linux
- MWave and DSP Modems might work, but only if you first start Windows/Dos each time you power on your PC
- Modems with RPI (Rockwell) drivers work but with reduced performance

MWave and DSP Modems

Such modems use DSP's (Digital Signal Processors) which are programmed by algorithms which must be downloaded from the hard disk to the DSP's memory just before using the modem. Unfortunately, the downloading is often done by Dos/Windows programs so one can't do it from Linux. Ordinary modems that work with Linux often have a DSP too (and may mention this on the packaging), but the program that runs it is stored inside the modem. This is not a "DSP modem" in the sense of this section and should work OK under Linux. An example of a DSP modem is IBM's Aptiva MWAVE.

If a DSP modem modem simulates a serial port, then it is usable with Linux which communicates with modems via the serial port. If you also have Dos/Windows on the same PC you may be able to use the modem: 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, go into 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://maalox.pharmacy.ohio−state.edu/~ejolson/linux/newcom.html.

Rockwell (RPI) Drivers

Modems that require Rockwell RPI (Rockwell Protocol Interface) drivers can still be used with Linux even though the driver software works only under Windows. This is because the 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 modem this command is +H0. Not having data compression available may not be much of a handicap since most long files which you download from the Internet are already compressed and attempts at further compression may only slow things down a bit.

3.Modem Pools, Digital Modems

A modem pool is a number of modems on the same card (such as a multiport modem card) or many modems in an external chassis (something like an external modem). The modems may be analog modems similar to modems used for home/office PCs (can't send at 56k even if they are "56k modems"). They also could be "digital modems" which can send at 56k. By 56k I actually mean all speeds above 33.6k as the 56k modem can't quite do a true 56k speed. 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 will require that you install special drivers for them.

3.1 Analog Modem Pools, Multiport Modem Cards

These are just several (or more) analog modems (the common home/office modem) provided either on a 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 the burden on 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 multiport modem cards. 8 modems/card is common. The cards listed claim to work with Linux and the websites should point you to a driver for them.

Multiport Modem Cards:

- MultiModemISI by Multi−Tech Systems. 56k or 33.6k, PCI or ISA, 4 or 8 ports. ISDN/56k hybrids. http://www.multitech.com/products/
- RAStel by Moreton Bay Products. 56k PCI or ISA, 4 or 8 ports. Also 2 modems + 2 vacant serial ports. http://www.moretonbay.com.au/MBWEB/product/rastel/rastel.htm
- RocketModem by Comtrol. ISA 33.6k, 4 or 8 port. http://www.comtrol.com/SALES/SPECS/Rmodem.htm
- AccelePort (RAS Family) by Digi. http:/www.dgii.com/digi.cfm?p=940564.pi.prd.00000046

3.2 Digital Modems

"digital modems" are much different than the analog modems that most people use in their PCs. They require a digital connection to the telephone line and don't use serial ports for the interface to the computer. Instead, they interface directly to the PC bus via a special card (which may also contain the "digital modems"). They are able to send at 56k, something no analog modem can do. They are often a component of "remote access servers" or "digital modem pools"

The cables from the phone company that carry digital signals have been designed for high bandwidth so that the same cable carries multiple telephone calls. It's done by "time−division multiplexing". So the first task to be done is to separate the phone calls and send each phone call to its own "digital modem". There is also the task in the reverse direction of combining all of the calls onto a single line. These tasks are done by what is sometimes called a "... concentrator".

The digital modem takes the digital signal from the telephone company and after processing it, puts it on the PC's bus (likely sending it to a buffer in memory). Likewise, it handles sending digital signals in the opposite direction 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 "serial modems" call themselves "digital signal processors", "remote access servers", etc. and may not even mention the word "modem". This is technically correct terminology.

Such a 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.

Before reading this

4.Modem & Serial PortBasics

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 serial port 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 section of a chip) known as a

UART. For more details on UARTs see the section What Are UARTs? How Do They Affect Performance?. 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 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 don't work exactly this way but instead use software (running on the CPU) to do much of the modem's work. Unfortunately, such software is often only available for the MS Windows OS (it hasn't been ported to Linux). Thus you can't use most of these modems with Linux See Software (Internal) Modems.

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 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. For the PCI bus it doesn't work exactly this way since the PCI bus has its own system of interrupts. But since the PCI−aware BIOS sets up chips to map these PCI interrupts to IRQs, it seemingly behaves just as described above except that sharing of 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 to see if it matches the device's. If the address matches, then the I/O device reads the data on the data bus.

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 what 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 for 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 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, 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 (1 in old serial ports) 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 is no way (such as flow control) to stop the flow rapidly enough to prevent this. For an internal modem the 16−byte FIFO buffer is on the same card and a good modem will not write to it if it's full. Thus a good internal modem 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 an internal modem over an external.

Interrupts are also issued when the serial port has just sent out all 16 of its bytes from its small transmit 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 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 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 is since the modem contains a dedicated serial port within it.

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 stop the flow of bytes in a wire. It also includes provisions to restart the flow without any loss of bytes. Flow control is needed for modems to allow a jump in instantaneous flow rates.

Example of Flow Control

For example, consider the case where you connect a 36.6k external modem via a short cable to your serial port. The modem sends and receives bytes over the phone line at 36.6k bits per second (bps). 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 eventually overrun (run out of free storage space) unless the 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 the buffer, the level 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.

The above is a simple example of flow control for flow 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 involve 3 buffers: 1. in the modem 2. in the UART chip (called FIFOs) 3. in main memory managed by the serial driver. Flow control protects certain buffers from overflowing. The small UART FIFO buffers are not protected in this way but rely instead on a fast response to the interrupts they issue. FIFO stand for "First In, First Out" which is the way it handles bytes. All the 3 buffers use the FIFO rule but only one of them also uses it as a name. This is the essence of flow control but there are still some more details.

You don't often need flow control in the direction from the modem to a PC. For complex example of a case where it's needed see "Complex Flow Control Example" in the Serial−HOWTO. But if you don't have a high enough speed set between the modem and the computer (serial port speed) then you do need to slow down the flow from the modem to the PC. To do this you must stop the incoming flow of bytes over the telephone line. Your modem must tell the other modem to stop sending. See 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. Modern modems almost always use hardware flow control between the modem and the serial port.

Software flow control uses the main receive and transmit 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, 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 long files from my PC to an ISP and a huge amount of retries and errors resulted (but eventually Kermit could send a long file after many retries). Receiving in the other direction (from my ISP to me) worked fine. The problem turned out to be a hardware defect in my modem that had resulted in disabling flow control. My modem's buffer was overflowing (overrunning) on long outgoing files since no "stop" signal was ever sent to the computer to halt sending to the modem. There was no problem in the direction from the modem to my computer since the capacity (say 115.2k) was always higher than the flow over the telephone line. The fix was to enable flow control by putting into the init string an enable−flow−control command for the modem (It should have been enabled by default but something was wrong).

Modem−to−Modem Flow Control

This is the flow control of the data sent over the telephone lines between two modems. Practically speaking, it only exists when you have error correction 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

Although much has been explained about this including flow control, a pair of 16−byte FIFO buffers (in the hardware), and a pair of larger buffers inside a modem there is still another pair of buffers. These are large buffers (perhaps 8k) in main memory also known as serial port buffers. When an application program sends bytes to the serial port (and modem) they first get stashed in the the transmit serial port buffer in main memory. The pair consists of both this transmit buffer and a receive buffer for the opposite direction of byte−flow.

The serial device driver takes out say 16 bytes from this transmit buffer, one byte at a time and puts them into the 16−byte transmit buffer in the serial hardware for transmission. Once in that transmit buffer, there is no way to stop them from being transmitted. They are then transmitted to the modem which also has a fair sized (say 1k) buffer. When the device driver (on orders from flow control) 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.

When it 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. It may switch to running other processes while it's waiting at a flow control stop. The above was a little oversimplified since there is another alternative of having the application program itself do something else while it is waiting to "write".

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 command (not guaranteed to work on all modems) are listed in this HOWTO in the section Modem Configuration

4.12 Serial Software: Device 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. This module will normally get loaded automatically if it's needed. The kernel 2.2 + will do this. 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. If it's still built into the kernel (you might have selected this when you compiled the kernel) don't let the serial module load. If you do and wind up with two serial drivers, it's reported that you can't use the serial ports and get an "I/O error" if an attempt is made to open them.

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 What is Setserial for more info on `setserial`.)

One may modify the driver by editing the kernel source code. Much of the serial driver is found in the file serial.c. For details regarding writing of programs for the serial port see Serial−Programming−HOWTO (currently being revised by Vern Hoxie).

5.Configuring Overview

If you want to use a modem only for MS Windows/Dos, then you can just install almost any modem and it will work OK. With a Linux PC it's not usually this easy unless you use an external modem. All external modems should work OK (even if they are labeled "Plug and Play") But most new internal modems are Plug−and−Play (PnP) and have PnP serial ports. If it's an ISA modem may need to use the Linux "isapnp" program to configure these PnP serial ports. See the Plug−and−Play−HOWTO for more information.

Since each modem has an associated serial port there are two parts to configuring a modem:

- Configuring the modem itself: Done by the communication program
- Configuring the modem's serial port: Done only *partly* by the communication program

Most of the above configuring (but not necessarily most of the effort) is done by the communication program that you use with the modem such as `minicom`, `seyon`, `wvdial` (for PPP). If you use the modem for dial−in, then the `getty` program which you use to present outsiders with a login−prompt, will help configure. Thus to configure the modem (and much of the serial port) you need to configure the communication program (such as the PPP dialer or `getty`).

Unfortunately the above configuring doesn't do the low−level 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 but today it's done by "Plug−and−Play" software.

But there's a serious problem: Linux (as of late 1999) is not a Plug−and−Play operating system but it does have Plug−and−Play tools which you may use to set up the configuration although they are not always very user friendly. This may create a difficult problem for you. The next section will go into this in much more detail.

6.Configuring the Serial Port

6.1 PCI Bus Support Underway

The kernel 2.2 serial driver contains no special support for the PCI bus. But kernels 2.3 and 2.4 will eventually support some PCI serial cards (and modem cards). Many PCI cards need special support in the driver. The driver will read the id number digitally stored on the card to determine how (or if) to support the card. If you have a PCI card which you are convinced is not a winmodem but it will not work, you can help in attempting to create a driver for it. To do this you'll need to contact the maintainer of the serial driver, Theodore (Ted) Y. Ts'o. But first check out the modem list site http://www.o2.net/~gromitkc/winmodem.html for the latest info on PCI modems and related topic.

You will need to email Ted Ts'o a copy of the output of "lspci −vv" with full information about the model and manufacturer of the PCI modem (or serial port). Then he will try to point you to a test driver which might

work for it. You will then need to get it, compile it and possibly recompile your kernel. Then you will test the driver to see if it works OK for you and report the results to Ted Ts'o. If you are willing to do all the above (and this is the latest version of this HOWTO) then email the needed info to him at: mailto:tytso@mit.edu.

PCI modems are not well standardized. Some use main memory for communication with the PC. It you see 8−digit hexadecimal addresses it's not likely to work with Linux. Some require special enabling of the IRQ. The output of "lspci" can help determine if one can be supported. If you see a 4−digit IO port and no long memory address, the modem 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.

6.2 Configuring Overview

In many cases, configuring will happen automatically and you have nothing to do. But sometimes you need to configure (or just want to check out the configuration). If so, first you need to know about the two parts to configuring the serial port under Linux:

The first part (low−level configuring) is assigning it an IO address, IRQ, and name (such as ttyS2). This IO−IRQ pair must be set in both the hardware and told to the serial driver. We might just call this "io−irq" configuring for short. The `setserial` is used to tell the driver. PnP methods, jumpers, etc, are used to set the hardware. Details will be supplied later. If you need to configure but don't understand certain details it's easy to get into trouble.

The second part (high−level configuring) is assigning it a speed (such as 38.4k bits/sec), selecting flow control, etc. This is often done by communication programs such as PPP, minicom, or by getty (which you may run on the port so that others may log into your computer). However you will need to tell these programs what speed you want, etc. by using a menu or a configuration file. This high−level configuring may also be done with the `stty` program. `stty` is also useful to view the current status if you're having problems. See also the Serial−HOWTO section: "Stty". When Linux starts, some effort is made to detect and configure (low−level) a few serial ports. Exactly what happens depends on your BIOS, hardware, Linux distribution, etc. If the serial ports work OK, there may be no need for you to do any configuring. Application programs often do the high−level configuring but you may need to supply them with the required information. With Plug−and−Play serial ports (often built into an internal modem), the situation has become more complex. Here are cases when you need to do low−level configuring (set IRQ and IO addresses):

- Plan to use more than 2 serial ports
- Installing a new serial port (such as an internal modem)
- Having problems with serial port(s)

For kernel 2.2+ you may be able to use more that 2 serial ports without low−level configuring by sharing interrupts. This only works if the serial hardware supports it and may be no easier than low−level configuring. See Interrupt sharing and Kernels 2.2+

The low−level configuring (setting the IRQ and IO address) seems to cause people more trouble (than high−level), although for many it's fully automatic and there is no configuring to be done. Thus most all of this section is on that topic. Until the serial driver knows the correct IRQ and IO address the port will not work at all. It may not even be found by Linux. Even if it can be found, it may work extremely slow if the IRQ is wrong. See Extremely Slow: Text appears on the screen slowly after long delays.

In the Wintel world, the IO address and IRQ 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 review, the low−level configuring consists of putting two values (an IRQ number and IO address) into two places:

1. the device driver (often by running "`setserial`" at boot−time)
2. memory registers of the serial port hardware itself

You may watch the start−up (= boot−time) messages. They are usually correct. But if you're having problems, there's a good chance that some of these messages don't show the true configuration of the hardware (and they are not supposed to). See I/O Address & IRQ: Boot−time messages.

6.3 Common mistakes made re low−level configuring

Here are some common mistakes people make:

- setserial command: They run it (without the "autoconfig" option) and think it has checked out the hardware (it hasn't).
- setserial messages: They see them displayed on the screen at boot−time, and erroneously think that the result 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: People think this shows the hardware configuration when it only shows about the same data (possibly erroneous) as setserial.

6.4 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. Use Shift−PageUp to go back to all the messages after they have flash by. Shift−PageDown will scroll in the opposite direction. The `dmesg` command may be used at any time to view some of the messages but it often misses important ones. Here's an example of the start−up messages (as of mid 1999). Note that ttyS00 is the same as /dev/ttyS0.

```
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
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
ttyS02 at 0x03e8 (irq = 4) is a 16550A

Later you see what was saved, but it's not necessarily correct either:

Loading the saved−state of the serial devices...
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

```
/dev/ttyS2 at 0x03e8 (irq = 5) is a 16550A
```

Note that there is a slight disagreement: The first message shows ttyS2 at irq=4 while the second shows it at irq=5. Your may only have the first message. In most cases the last message is the correct one. But if your 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. 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 someone incorrectly put this into a configuration file (or the like). The fact that Linux sometimes gets IRQs wrong is because it doesn't probe for IRQs. It just assumes the "standard" ones (first message) or accepts what you told it when you configured it (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 serial ports 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. Your may check them manually with `setserial` using the `autoconfig` and `auto_irq` options but this isn't guaranteed to be correct.

The data shown by the BIOS messages (which you see at first) is what is set in the hardware. If your serial port is Plug−and−Play PnP then it's possible that the `isapnp` 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). 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, Linux will not find them unless the IRQ and IO has been set inside the hardware by Plug−and−Play software. This is a common reason why the start−up messages do not show a serial port that physically exists. The PC hardware (a PnP BIOS) may automatically low−level configure this. PnP configuring will be explained later.

6.5 What is the current IO address and IRQ of my Serial Port ?

The previous section indicated how to attempt to do this by looking at the start−up messages. If they give you sufficient info then you may not need to read this section. If they don't then there are some other ways to look into this.

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. They both should be the same. If they're not it spells trouble since the driver has incorrect info on the physical serial port. If the driver has the wrong IO address it will try to send data to a non−existing serial port −−or even worse, to an actual device that is not a serial port. 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.

What does the device driver think?

This is easy to find out. Just look at the start−up messages or type "setserial −g /dev/ttyS*". If everything works OK then what it tells you is likely also set in the hardware. There are some other ways to find this info by looking at "files" in the /proc directory. An important reason for understanding these other ways is to warn you that they only show what the device driver thinks. Some people view certain "files" in the /proc directory and erroneously think that what they see is set in the hardware but "it ain't necessarily so".

/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). Note that in both cases above you are only seeing what the driver thinks and not necessarily what is actually set in the hardware. /proc/interrupts also shows how many interrupts have been issued (often thousands) for each device. You can get a clue from this because if you see a large number of interrupts that have been issued it means that there is a piece of hardware somewhere that is using that interrupt. Sometimes a showing of just a few interrupts doesn't mean that that 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 and it implies that the driver is using the wrong interrupt. 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. To do this, try to jump to a shell without exiting the program.

What is set in my serial port hardware ?

How do you find out what IO address and IRQ are actually set in the device hardware? 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 pciutils can.

One crude method is try probing with setserial using the "autoconfig" option. You'll need to guess the addresses to probe at. See What is Setserial. For a PCI serial port, use the "lspci" command (for kernels <2.2 look at /proc/pci). If your serial port is is Plug−and−Play see the next two subsections.

For a port set with jumpers, its how the jumpers were set. 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.

What is set in my PnP serial port hardware ?

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. If you have an ISA PnP port, it can reach a state where it doesn't have any IO address or IRQ and is in effect disabled. It should still be possible to find the port using the `pnpdump` program.

For Plug−and−Play (PnP) on the ISA bus 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. The address it "trys" is not the device's IO address, but a special

For PnP ports checking on how it's configured under DOS/Windows may not be of much help. Windows stores its configuration info in its Registry which is not used by Linux. It may supply the BIOS's non−volatile memory with some info but it may not be kept in sync with the current Window configuration in the Registry ?? 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 running Linux) then Linux should use whatever configuration is in the BIOS's non−volatile memory.

6.6 Choosing Serial IRQs

If you have a true Plug−and−Play set up where either the OS or a PnP BIOS configures all your devices, then you don't choose your IRQs. PnP determines what it thinks is best and assigns them. But if you use the tools in Linux for Plug−and−Play (isapnp and pcitools) 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. This is quite inefficient but can be tried if there is an interrupt conflict or mis−set interrupt. The advantage of assigning this 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 and Kernels 2.2+

The general rule is that every device should use a unique IRQ and not share them. But there are situations where sharing is permitted such as with most multi−port boards. Even when it is permitted, it may not be 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 nice to allocate every device its own interrupt.

Prior to kernel 2.2, serial IRQs could be shared with each other only for most multiport boards. Starting with kernel 2.2 serial IRQs may be sometimes shared between all 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). Thus even if you have 2.2, it may be best to avoid sharing.

What IRQs to choose?

The serial hardware often has only a limited number of IRQs it can be set at. 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 of 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 coprocessor
IRQ 14    Hard disk controller 1
IRQ 15    Hard disk controller 2
```

There is really no Right Thing to do when choosing interrupts. Just make sure it 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, double−check `/proc/interrupts` when programs that use interrupts are being run and make sure there are no conflicts.

6.7 Choosing Addresses −−Video card conflict with ttyS3

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 (but shouldn't if the serial port is well designed) with the IO address of `ttyS3` at 0x02e8 if the serial port ignores the leading 0 hex digit (many do). That is bad news if you try to use `ttyS3` at this IO address.

In most cases you should use the default addresses if feasible. Addresses shown 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 the serial ports:

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

6.8 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 you have both a PnP serial port and a PnP Linux operating system (see Plug−and−Play−HOWTO).
- Using `isapnp` for a PnP serial port non−PCI)
- Using pciutils (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 was designed to automate this io−irq configuring, but for Linux at present, it has made life more complicated. The standard kernels for Linux don't support plug−and−play very well. If you use a patch to the Linux kernel to covert it to a plug−and−play operating system, then all of the above should be handled automatically by the OS. But when you want to use this to automate configuring devices other that the serial port, you may find that you'll still have to configure the drivers manually since many Linux drivers are not written to support a Linux PnP OS. If you use `isapnptools` or the BIOS for configuring plug−and−play this will only put the two values into the registers of the serial port section of the modem card and you will likely still need to set up setserial. None of this is easy or very well documented as of early 1999. See Plug−and−Play−HOWTO and the isapnptools FAQ.

Using a PnP BIOS to I0−IRQ Configure

While the explanation of how to use a PnP OS 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 and for an "Award" BIOS it was found under "chipset features setup" There is often little to choose from. Unless otherwise indicated in a menu, these first two ports normally get set at the standard IO addresses and IRQs. See Serial Port Device Names & Numbers

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 you probably don't have) or if thinks you don't have a PnP OS it may fully configure all the PnP devices but not configure the device drivers. This is what you want but it's not always easy to figure out exactly what the PnP BIOS has done.

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 change its PnP configuration to accommodate 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?

6.9 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

6.10 Other Configuring

Configuring 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 if you're in luck it might be 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 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 crtscts < /dev/ttyS2
```

If you want to see if flow control is enabled do the following: In minicom (or the like) type AT&V to see how the modem is configured and look for &K3 which means hardware flow control. Then see if the device driver knows about it by typing: stty −a < /dev/ttyS2 Look for "crtscts" (without a disabling minus sign).

7.Modem Configuration (excluding serial port)

7.1 Finding Your Modem

Before spending a lot of time configuring your modem, you 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 it it works. If so, then it's been found. 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.

7.2 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 are prefaced by the letters AT. For example: ATZ&K3 There are two commands here Z and &K3. 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. Another point is that to get the modem to act on the AT command string, a return character must be sent at the end of the string.

Such command strings are either automatically sent to the modem by communication programs or are sent directly by you. Most communication programs provide a screen where you can type commands directly to your modem. This is good for setting up the modem as you can have it remember how it was set even after its powered off.

If you have a manual for your modem you can likely look up the AT command set in it. Otherwise, you may try to find it on the Internet. One may use a search engine and include some actual commands in the search terms to avoid finding sites that just talk about such commands but fail to list them. You might also try a few of the sites listed in the subsection Web Sites

7.3 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). 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 a communication program does this. What it sends will depend on how you configured the communications program or what script you wrote for it if you use Kermit. You can usually edit the init string your communication program uses 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 uses when it's first powered on could 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. Which one of these user−defined profiles is used at power−up depends on another item stored in the profile. If the command &Y0 is given then in the future profile 0 will be used at power−on. If it's a 1 instead of a 0 then profile 1 will be used at power−on.

There are also commands to recall (use it now) any of the 4 stored profiles. One may put such a command in an init string. Of course if it recalls the same profile as was automatically loaded at power−up, nothing is changed unless the active profile has been modified since power−up. Since it could have been modified It's a good idea to use some kind of an init string even if it does nothing more than recalling a stored profile.

Recalling a saved profile (use 1 instead of 0 for profile 1):
Z0 recalls user−defined profile 0 and resets (hangs up, etc.)
&F0 recalls factory profile 0

Once you have sent commands to the modem to configure it the way you want (including recalling 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 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. By doing it this way no one can cause problems by modifying the user−defined profile which is loaded at power−on.

You may pick 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 someone else wrote, but modify it to suit your purposes.

Now 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 (such as meaning to pause briefly) and will not be sent to the modem.

7.4 Other Modem Commands

Future editions of Modem−HOWTO may contain more AT commands but 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       verbose ON
S0=0     never answer (uugetty does this with the WAITFOR option)
```

Here's some more codes concerning modem control lines DCD and DSR:

```
&C1 DCD is on after connect only
&S0 DSR is always on
```

These affect what your modem does when calls start and end. What DTR does may also be set up but it's more complicated.

If your modem does not support a stored profile, you can set these through the `INIT` string in a config file (or the like). Some older modems come with DIP switches that affect register settings. Be sure these are set correctly, too.

Greg Hankins has a collection of modem setups for different types of modems. If you would like to send him your working configuration, please do so: mailto:gregh@cc.gatech.edu You can get these setups at `ftp://ftp.cc.gatech.edu/pub/people/gregh/modem-configs`.

Note: to get his 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 Supra modems treat DCD 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.Serial Port Devices /dev/ttyS2, etc.

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

8.1 Serial Port Device Names & Numbers

Devices in Linux have major and minor numbers. Each serial port may have 2 possible names in the /dev directory: ttyS and cua. Their drivers behave slightly differently. The cua device is deprecated and will not be used in the future. See The cua Device.

Dos/Windows use the COM name while the `setserial` program uses tty00, tty01, etc. Don't confuse these with dev/tty0, dev/tty1, etc. which are used for the console (your PC monitor) but are not serial ports. The table below is for the "standard" case (but yours could be different).

dos		major	minor		major	minor	IO address
COM1	/dev/ttyS0	4,	64;	/dev/cua0	5,	64	3F8
COM2	/dev/ttyS1	4,	65;	/dev/cua1	5,	65	2F8
COM3	/dev/ttyS2	4,	66;	/dev/cua2	5,	66	3E8
COM4	/dev/ttyS3	4,	67;	/dev/cua3	5,	67	2E8

Note that all distributions should come with ttyS devices (and many distributions have the obsolete cua device). You can verify this by typing (don't feel bad if you don't find any obsolete cua devices):

```
linux% ls −l /dev/cua*
linux% ls −l /dev/ttyS*
```

8.2 Link ttySN to /dev/modem ?

On some installations, two extra devices will be created, `/dev/modem` for your modem and `/dev/mouse` for your mouse. Both of these are symbolic links to the appropriate device in `/dev` which you specified during the installation (unless you have a bus mouse, then `/dev/mouse` will point to the bus mouse device).

There has been some discussion on the merits of `/dev/mouse` and `/dev/modem`. The use of these links is discouraged. In particular, if you are planning on using your modem for dialin you may run into problems because the lock files may not work correctly if you use `/dev/modem`. However, if you change or remove this link, some applications might need reconfiguration.

8.3 The cua Device

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 DCD 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 assert DCD (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 will be put in the kernel log when one uses cua. This is an omen that cua is on the way out.

9.Interesting Programs You Should Know About

9.1 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.

Introduction

Don't ever use `setserial` with Laptops (PCMCIA). `setserial` is a program which allows you to tell the device driver software the I/O address of the serial port, which interrupt (IRQ) is set in the port's hardware, what type of UART you have, etc. It can also show how the driver is currently set. In addition, it can be made to probe the hardware and try to determine the UART type and IRQ, but this has severe limitations. See Probing. Note that it can't set the IRQ, etc in the hardware of PnP serial ports.

If you only have one or two built−in serial ports, they will usually get set up correctly without using setserial. Otherwise (or if there are problems with the serial port) you will likely need to deal with setserial. Besides the man page for `setserial`, check out info in `/usr/doc/setserial.../` or `/usr/share/doc/setserial`. It should tell you how setserial is handled in your distribution of Linux.

`Setserial` is often run automatically at boot−time by a start−up shell−script for the purpose of assigning IRQs, etc. to the driver. Setserial will only work if the serial module is loaded (or if the equivalent was compiled into your kernel). If you should (for some reason) unload the serial module later on, the changes previously made by `setserial` will be forgotten by the kernel. So `setserial` must be run again to reestablish them. In addition to running via a start−up script, something akin to `setserial` also runs when

the serial module is loaded (or the like). Thus when you watch the start−up messages on the screen it may look like it ran twice, and in fact it has.

Setserial can set the time that the port will keep operating after it's closed (in order to output any characters still in its buffer in main RAM). This is needed at slow baud rates of 1200 or lower. It's also needed at higher speeds if there are a lot of "flow control" waits. See "closing_wait" in the man pg.

Setserial does not set either IRQ's nor I/O addresses in the serial port hardware itself. That is done either by jumpers or by plug−and−play. You must tell setserial the identical values that have been set in the hardware. Do not just invent some values that you think would be nice to use and then tell them to setserial. However, 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 that device driver is configured for your ports. Add a −a to the option −g to 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 without complaint. Of course the serial port driver will not work correctly (if at all) with such bogus values.

While assignments made by setserial are lost when the PC is powered off, a configuration file may restore them (or a previous configuration) when the PC is started up again. In newer versions, what you change by setserial gets automatically saved to a configuration file. In older versions, the configuration file only changes if you edit it manually so the configuration remains the same from boot to boot. See Configuration Scripts/Files

Probing

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 this 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`. To try to detect the physical hardware use the −v (verbose) and `autoconfig` command to `setserial`. 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 versions of setserial >= 2.15, the results of your last probe test may be saved and put into the configuration file `/etc/serial.conf` which will be used next time you start Linux. At boot−time when the serial module loads (or the like), a probe for UARTs is made automatically and reported on the screen. But the IRQs shown may be wrong. The second report of the same is the result of a script which usually does no probing and thus provides no reliable information as to how the hardware is actually set. It only shows configuration date someone wrote into the script or data that got saved in /etc/serial.conf.

It may be that two serial ports both have the same IO address set in the hardware. Of course this is not permitted 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 ficticious value.

Boot−time Configuration

When the kernel loads the serial module (or if the "module equivalent" is built into the kernel) then 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). You see this as a boot−time message just like as if `setserial` had been run. If you use 3 or more ports, this may result in IRQ conflicts.

To fix such conflicts by telling setserial the true IRQs (or for other reasons) there may be a file somewhere that runs `setserial` again. This happens early at boot−time before any process uses the serial port. In fact, your distribution may have set things up so that the setserial program runs automatically from a start−up script at boot−time. More info about how to handle this situation for your particular distribution might be found in file named "setserial..." or the like located in directory /usr/doc/ or /usr/share/doc/.

Configuration Scripts/Files

Your objective is to modify (or create) a script file in the /etc tree that runs setserial at boot−time. Most distributions provide such a file (but it may not initially reside in the /etc tree). In addition, setserial 2.15 and higher often have an /etc/serial.conf file that is used by the above script so that you don't need to directly edit the script that runs setserial. In addition just using setserial on the command line (2.15+) may ultimately alter this configuration file.

So prior to version 2.15 all you do is edit a script. After 2.15 you may need to either do one of three things: 1. edit a script. 2. edit `/etc/serial.conf` or 3. run "setserial" on the command line which will result in `/etc/serial.conf` automatically being edited. Which one of these you need to do depends on both your particular distribution, and how you have set it up.

Edit a script (after version 2.15: perhaps not)

Prior to setserial 2.15 (1999) there was no /etc/serial.conf file to configure setserial. Thus you need to find the file that runs "setserial" at boot time and edit it. If it doesn't exist, you need to create one (or place the commands in a file that runs early at boot−time). If such a file is currently being used it's likely 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. You might use "locate" to try to find such a file. For example, you could type: locate

"*serial*".

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 not a good idea to use this since it may not be run early enough. It's been reported that other processes may try to open the serial port before rc.local runs resulting in serial communication failure.

If such a file is supplied, it should contain a number of commented−out examples. By uncommenting some of these and/or modifying them, you should be able to set things up correctly. Make sure that you are using 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 is a lot faster than doing repeated reboots to get it right. Of course you can also test a single `setserial` command by just typing it on the command line.

If you want setserial to automatically determine the uart and the IRQ for ttyS3 you would add something like:

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

Do this for every serial port you want to auto configure. Be sure to give a device name that really does exist on your machine. In some cases this will not work right due to the hardware so if you know what the uart and irq actually are, may want to assign them explicitly with "setserial". For example:

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

For versions >= 2.15 (provided your distribution implemented the change, Redhat didn't) 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 New configuration method using /etc/serial.conf.

New configuration method using /etc/serial.conf

Prior to setserial version 2.15, the way to configure setserial was to manually edit the shell−script that ran setserial at boot−time. See Edit a script (after version 2.15: perhaps not). Starting with version 2.15 (1999) of `setserial` this shell−script is not edited but instead gets its data from a configuration file: `/etc/serial.conf`. Furthermore you may not even need to edit serial.conf because using the "setserial" command on the command line may automatically cause serial.conf to be edited appropriately.

This was intended to make it so that you don't need to edit any file in order to set up (or change) setserial so it will do the right thing each time that Linux is booted. But there are serious pitfalls because it's not really "setserial" that edits serial.conf. Confusion is compounded because different distributions handle this differently. In addition, you may modify it so it works differently.

What often happens is this: When you shut down your PC the script that runs "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 puts that info into the `serial.conf` file. 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.

If 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 at least one distribution, the removal of "###AUTOSAVE###" from the first line is automatically done after the first time you shutdown just after installation. The serial.conf file will hopefully contain some comments to help you out.

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". 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.

BUG: As of July 1999 there is a bug/problem since with ###AUTOSAVE### only the setserial parameters displayed by "setserial −Gg /dev/ttyS*" get saved but the other parameters don't get saved. Use the −a flag to "setserial" to see all parameters. This will only affect a small minority of users since the defaults for the parameters not saved are usually OK for most situations. It's been reported as a bug and may be fixed by now.

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 share IRQ 4, while ttyS0 and ttyS3 share IRQ 3. But sharing serial interrupts is not permitted unless you: 1. have kernel 2.2 or better, and 2. you've complied 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 an internal modem and retain ttyS0 and ttyS1, then you should attempt to find an unused IRQ and set it both on 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 may be one you can use for a modem. To set the IRQ in hardware you may need to use isapnp, a PnP BIOS, or patch Linux to make it PnP. To help you determine which spare IRQ's you might have, type "man setserial" and search for say: "IRQ 11".

9.2 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.

9.3 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. But you don't need to install PPP in order to use `wvdialconf`. It will only find modems which are not in use. It will also automatically devise a "suitable" init strings 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.

9.4 What is stty ?

`stty` is like setserial but it sets the baud rate and other parameters of a serial port. Typing "stty −a < /dev/ttyS2" should show you how ttyS2 is configured. Most of the settings are for things that you never need to use with modems (such as some used only for old terminals of the 1970s). Your communication package should automatically set up all the setting correctly for modems. But stty is sometimes useful for trouble−shooting.

Two items set by stty are: 1. Hardware flow control by "crtscts" and 2. Ignore the DCD signal from the modem: "clocal". If the modem is not sending a DCD signal and clocal is disabled (stty shows −clocal) then a program may not be able to open the serial port. If the port can't open, the program may just hang, waiting (often in vain) for a DCD signal from the modem.

Minicom sets clocal automatically when it starts up so there is no problem. But version 6.0.192 of Kermit hung when I set −clocal and tried to "set line ..." If −clocal is set and there is no DCD signal then even the "stty" command will hang and there is seemingly no way to set clocal (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 DCD signal) and then exit minicom with no reset so that the DCD signal remains

on. Then you may use stty again.

10.Trying Out Your Modem (Dialing Out) .

10.1 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. Before you try to get the Internet on it or have people call in to you, first try something simpler like dialing out to some number to see if your modem is working OK. Find a phone number that is connected to a modem. It you don't know what number to call, ask at computer stores for such phone numbers of bulletin boards, etc. or see if a local library has a phone number for their 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 for more details. If you have no idea what speed to set, just set 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 the 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. Dialing programs include: minicom, seyon (X−windows), and kermit. See section Communications Programs about some communications programs. Two examples are presented next: Dialing Out with Minicom and Dialing Out with Kermit

10.2 Dialing Out with Minicom

Minicom comes with most Linux distributions. To configure it you should be the root user. Type "minicom −s" to configure. This will take you directly to the configuration (set−up) menus. Alternatively you could just run "minicom" and then 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). From the help menu go to the Configuration menu.

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. You may want to exit minicom and start it again so it can now find the serial port and initialize the modem, or you could go to help and tell minicom to 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 something is wrong and there is no point of trying to dial.

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.

10.3 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 `ttyS3`, 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/ttyS3
C−Kermit>set carrier−watch off
C−Kermit>set speed 115200
/dev/ttyS3, 115200 bps
C−Kermit>c
Connecting to /dev/ttyS3, 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/ttyS3       ; Select communication device
C−Kermit>set speed 115200          ; Set the dialing speed
C−Kermit>dial 7654321              ; Dial
 Number: 7654321
 Device=/dev/ttyS3, modem=usr, speed=115200
 Call completed.<BEEP>
Connecting to /dev/ttyS3, 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 ...

login:
```

11.Dial−In

11.1 Overview

Dial−in is where you set up your PC so that others may dial in to your phone number and use your PC. The "point of view" is your PC. When you dial out from your PC you are also dialing in to another computer (but not dialing in to your own computer)

Dial−in works like this. Someone with a modem dials your telephone number. Your modem answers the call and connects. Once the caller is connected, your PC (via the `getty` program) starts the login process for the caller. The original method was to send a login prompt to the caller's screen (manual login). But a more modern method (if you use `mgetty`) is to start PPP (pppd) and let PPP automatically login the caller (no need to manually type in a name or password). See the PPP−HOWTO (new revision expected soon) and docs for `mgetty` for more details.

After the caller has logged in, the caller uses your PC. Using your PC may mean that the caller has a shell account and can use your PC just as if they logged in at the console (or from a text−terminal). It could also mean that they get connected to the Internet thru your PC (via PPP). The program that you use at your PC to handle dialin is called `getty` or `mgetty`. See About mgetty.

If you expect that people will be able to dial−in to you at 56k, it can't be done unless:

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" which includes the digital modems, etc. This type of setup is used by ISPs (Internet Service Providers).

11.2 Getty

`getty` is the program you run for dialin. You don't need it for dialout. In addition to presenting a login prompt, it also answers the telephone. Originally getty was used for logging in to a computer from a dumb terminal. It's currently used for logging in to a Linux console). There are a few different getty programs with slightly different names. Only certain ones work with modems for dialin. This `getty` program is usually started at boot−time. It must be called from the /etc/inittab file. You may find an example in this file of a call to getty which you will likely need to edit a bit. If you use a different getty program than the one shown in such an example, then you will need to edit it quite a bit since the options will have a different format.

There are four different getty programs to choose from that may be used with modems for dialin: `mgetty`, `uugetty`, `getty_em`, and `agetty`. A few details are given in the following subsections. `agetty` is the simplest (and weakest) of the four and some consider it mainly for use with directly connected text−terminals. `mgetty` has support for fax and voice mail but `Uugetty` doesn't. `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.

About mgetty

`mgetty` was written as a replacement for `uugetty` which was in existence long before `mgetty`. Both are for use with modems. Although `mgetty` may be also used for directly connected terminals the documentation for this is hard to pinpoint and mgetty will not (as of mid 1999) support software flow control (used on many terminals) without recompiling. This defect is listed as a bug. In addition to allowing dialup logins, `mgetty` also provides FAX support and auto PPP detection. There is a supplemental program called `vgetty` which handles voicemail for some modems. `mgetty` documentation is good (except for voice mail), and does not need supplementing . Please refer to it for installation instructions. 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. The other gettys are well covered by the documentation that comes with them.

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''. When logging into ``scicom'' as "anonymous", you must use your full e−mail address as the password. For example: greg.hankins@cc.gatech.edu

About agetty and mingetty

`agetty` is a simple, completely functional implementation of `getty` which is best suited for virtual consoles or terminals rather than modems. But it works fine with modems under favorable conditions (except you cannot dial out when agetty is running and waiting for a call). `agetty` in the Debian distribution is just named `getty`.

`mingetty` is a small getty that will work only for consoles (monitors) so you can't use it with modems for dialin.

11.3 What Happens when Someone Dials In ?

The caller runs some sort of communication program that dials your telephone number and your telephone rings. There are two different ways that your PC can answer the phone. One way is for the modem to automatically answer the call. The other way is for getty to sense the ringing and send a command to the modem to answer the call. Once the call is answered, your modem sends tones to the other modem (and conversely). The two modems negotiate how they will communicate and when this is done your modem sends a "CONNECTed" message (or the like) to `getty`. When `getty` gets this message, it sends a login prompt out the serial port. Sometimes `getty` just calls on a program named `login` to handle the logging in. `getty` usually starts running at boot−time but it must wait until a connection is made before sending out a "login" prompt.

Now for more details on the two methods of answering the call. By setting the S0 register of the modem to 3, the modem will automatically answer on the 3rd ring. If it's set to 0 then the modem will only answer the call if getty sends it an "A" (= Answer) command while the phone is ringing. Actually an "ATA" is sent since all modem commands are prefixed by "AT". You might think it best to utilize the ability of the modem to automatically answer the call, but it's actually better if `getty` answers it. If the modem doesn't automatically answer, it's called manual answer (even though `getty` automatically handles it).

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 "ATA" command. The modem then makes a connection and sends a "CONNECT ..." message to `getty` which then sends a login prompt to the caller.

The automatic answer case uses the CD (Carrier Detect) wire from the modem to the serial port to detect when a connection is made. It works like this. At boot−time `getty` tries to open the serial port but the attempt fails since there is normally no CD signal from the modem. Then the `getty` program waits at the

open statement in the program until a CD signal appears. 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 since Linux is a multiprocessing operating system. What actually wakes `getty` up is an interrupt which is issued when the CD line from the modem changes state to on.

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

11.4 Why Manual Answer is Best

The difference between the two ways of answering will show itself 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 phone is answered 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 using a configuration script.

11.5 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.icce.rug.nl/pub/unix/. Step−by−step instructions on how someone installed it (and PPP) is at http://www.stokely.com/unix.serial.port.resources/callback.html

11.6 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 disk. 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`. It's an optional addition to the well documented and widely distributed `mgetty` program. It supports ZyXEL−like voice modem commands. In the Debian distribution, you must get the mgetty−voice package in addition to the mgetty package and mgetty−doc package. Obsolete documentation has been removed from mgetty but replacement

documentation is lacking (except if you use the −h (help) option when running certain programs, etc.). But one sees postings about using it on the mgetty newsgroup. See About mgetty. It seems that `vgetty` is currently not very stable but it's successfully being used and development of it continues. If this is the latest version of this HOWTO can someone who is familiar with vgetty please let me know its current status.

12.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 What is getty? for a brief comparison of these 2 gettys.

12.1 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.

12.2 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: #F11520
```

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 the these lines are linked to each other by the last "word" in the line such as #38400. Blank lines are needed between each entry.

```
# 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: #2400

2400# B2400 CS8 # B2400 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 '@@'.
```

12.3 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.

13.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 possible under the circumstances. But you do have a choice as to what speed will be used between your modem and your computer. 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 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 that way since the modem card probably includes a dedicated serial port which does have speed limits (and settable speeds).

13.1 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 need to set the speed double the maximum modem speed on the phone line. If the compression ratio is 3 to 1 you need to set it 3 times faster.

13.2 Where do I Set Speed ?

This DTE 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.

13.3 Can't Set a High Enough Speed

You need to find out the highest speed supported by your hardware. As of late 1998 most hardware only supported speeds up to 115.2k bps. A few 56k internal modems support 230.4k bps. Recent Linux kernels support high speeds (over 115.2k) but you might have difficulty using it because of one or both of the following reasons:

1. The application program (or stty) will not accept the high speed.
2. Setserial has a default speed of 115,200 (but this default is easy to change)

How speed is set in hardware: the divisor and baud_base

Here's a list of commonly used divisors and their corresponding 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).

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 obviously sets the highest speed. If you happen to have hardware with a maximum speed of say 230.4k, then specifying 115.2k will result in divisor 1 and 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.

Work−arounds for setting speed

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. PROBLEM: stty and many communication programs (as of mid 1999) still have 115.2k as their maximum speed setting and will not let you set 230.4k, etc. So in these cases one solution is not to change anything with `setserial` but mentally keep in mind that the actual speed is always double what you set.

There's another work−around which is not much better. To use it you set the baud_base (with setserial) to the maximal speed of your hardware. This corrects the accounting so that if you set say 115.2k you actually get 115.2k. Now you still have to figure out how to set the highest speed if your communication program (or the like) will not let you do it. Fortunately, setserial has a way to do this: use the "spd_cust" parameter with "divisor 1". Then when you set the speed to 38400 in a communication program, the divisor will be set to 1 in the port and it will operate at maximum speed. For example:

setserial /dev/ttyS2 spd_cust baud_base 230400 divisor 1
Don't try using "divisor" for any other purpose other than the special use illustrated above (with spd_cust).

If there are two or more high speeds that you want to use that your communication program can't set, then it's not quite as easy as above. But the same principles apply. You could just keep the default baud_base and understand that when you set a speed you are really only setting a divisor. So your actual speed will always be your maximum speed divided by whatever divisor is set by the serial driver. See How speed is set in hardware: the divisor and baud_base

Crystal frequency is not baud_base

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

13.4 Speed Table

It's best to have at least a 16650 UART for a 56k modem but few modems support it. Second best is a 16550 that has been tweaked to give 230,400 bps. Here are some suggested speeds to set your serial line if your modem speed is:

- 56k (V.90) use 115200 bps or 230400 bps (a few % faster ?)
- 28.8k (V.34), 33.6k (V.34) use 115200 bps
- 14400 bps (V.32bis), with V.42bis data compression, use 57600 bps
- 9600 bps (V.32), with V.42bis data compression, use 38400 bps
- slower than a 9600 bps (V.32) modem, set your speed to the highest speed your modem supports.

14.Communications Programs And Utilities

PPP is by far the most widely used. It's used for Internet access. For dialing out to public libraries, bulletin boards, etc. `minicom` is the most popular followed by `Seyon` (X−Windows only) and `Kermit`.

14.1 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. Since permission is required to include Kermit in a commercial distribution, and since the documentation is not entirely free, some distributions don't include Kermit. In my opinion it's easier to set up Minicom and there is less to learn.

14.2 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: `chat`, `wvdial`
- `minicom` − `telix`−like communications program. Supports 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

- `efax` a small fax program
- `hylafax` a large fax program based on the client−server model.
- `mgetty+fax` handles fax stuff and login for dial−ins

Voicemail Software

- mvm is a Minimal VoiceMail for Linux
- vgetty is an extension to mgetty that handles voicemail for some modems. It should come with recent releases of mgetty.

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

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).
- `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.

14.3 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.What Are UARTs? How Do They Affect Performance?

15.1 Introduction to UARTS

(This section is also in the Serial−HOWTO)

UARTs (**U**niversal **A**synchronous **R**eceiver **T**ransmitter) are serial chips on your PC motherboard (or on an internal modem card). The UART function may also be done on a chip that does other things as well. On older computers like many 486's, the chips were on the disk IO controller card. Still older computer have dedicated serial boards.

The UART's purpose is to convert bytes from the PC's parallel bus to a serial bit−stream. The cable going out of the serial port is serial and has only one wire for each direction of flow. The serial port sends out a stream of bits, one bit at a time. Conversely, the bit stream that enters the serial port via the external cable is converted to parallel bytes that the computer can understand. UARTs deal with data in byte sized pieces, which is conveniently also the size of ASCII characters.

Say you have a terminal hooked up to your PC. When you type a character, the terminal gives that character to its transmitter (also a UART). The transmitter sends that byte out onto the serial line, one bit at a time, at a specific rate. On the PC end, the receiving UART takes all the bits and rebuilds the (parallel) byte and puts it in a buffer.

Along with converting between serial and parallel, the UART does some other things as a byproduct (side effect) of its primary task. The voltage used to represent bits is also converted (changed). Extra bits (called start and stop bits) are added to each byte before it is transmitted. See the Serial−HOWTO section, ``Voltage Waveshapes" for details. Also, while the flow rate (in bytes/sec) on the parallel bus inside the computer is very high, the flow rate out the UART on the serial port side of it is much lower. The UART has a fixed set of rates (speeds) which it can use at its serial port interface.

15.2 Two Types of UARTs

There are two basic types of UARTs: dumb UARTS and FIFO UARTS. Dumb UARTs are the 8250, 16450, early 16550, and early 16650. They are obsolete but if you understand how they work it's easy to understand how the modern ones work with FIFO UARTS (late 16550, 16550A, 16c552, late 16650, 16750, and 16C950).

There is some confusion regarding 16550. Early models had a bug and worked properly only as 16450's (no FIFO). Later models with the bug fixed were named 16550A but many manufacturers did not accept the name change and continued calling it a 16550. Most all 16550's in use today are like 16550A's. Linux will report it as being a 16550A even though your hardware manual (or a label note) says it's a 16550. A similar situation exists for the 16650 (only it's worse since the manufacturer allegedly didn't admit anything was wrong). Linux will report a late 16650 as being a 16650V2. If it reports it as 16650 it is bad news and only is used as if it had a one−byte buffer.

15.3 FIFOs

To understand the differences between dumb and FIFO (First In, First Out queue discipline) first let's examine what happens when a UART has sent or received a byte. The UART itself can't do anything with the data passing thru it, it just receives and sends it. For the original dumb UARTS, the CPU gets an interrupt from the serial device every time a byte has been sent or received. The CPU then moves the received byte out of the UART's buffer and into memory somewhere, or gives the UART another byte to send. The 8250 and 16450 UARTs only have a 1 byte buffer. That means, that every time 1 byte is sent or received, the CPU is interrupted. At low transfer rates, this is OK. But, at high transfer rates, the CPU gets so busy dealing with the UART, that is doesn't have time to adequately tend to other tasks. In some cases, the CPU does not get around to servicing the interrupt in time, and the byte is overwritten, because they are coming in so fast. This is called an "overrun" or "overflow".

That's where the FIFO UARTs are useful. The 16550A (or 16550) FIFO chip comes with 16 byte FIFO buffers. This means that it can receive up to 14 bytes (or send 16 bytes) before it has to interrupt the CPU. Not only can it wait for more bytes, but the CPU then can transfer all 14 (or more) bytes at a time. This is a significant advantage over the other UARTs, which only have 1 byte buffers. The CPU receives less interrupts, and is free to do other things. Data is not lost, and everyone is happy. Note that the interrupt threshold of FIFO buffers (trigger level) may be set at less than 14. 1, 4 and 8 are other possible choices.

While most PC's only have a 16550 with 16−byte buffers, better UARTS have even larger buffers. Note that the interrupt is issued slightly before the buffer get full (at say a "trigger level" of 14 bytes for a 16−byte buffer). This allows room for a few more bytes to be received during the time that the interrupt is being serviced. The trigger level may be set to various permitted values by kernel software. A trigger level of 1 will be almost like a dumb UART (except that it still has room for 15 more bytes after it issues the interrupt).

If you type something while visiting a BBS, the characters you type go out thru the serial port. Your typed characters that you see on the screen are what was echoed back thru the telephone line thru your modem and then thru your serial port to the screen. If you had a 16−byte buffer on the serial port which held back characters until it had 14 of them, you would need to type many characters before you could see what you typed (before they appeared on the screen). This would be very confusing but there is a "timeout" to prevent this. Thus you normally see a character on the screen just as soon as you type it.

The "timeout" works like this for the receive UART buffer: If characters arrive one after another, then an interrupt is issued only when say the 14th character reaches the buffer. But if a character arrives and the next character doesn't arrive soon thereafter, then an interrupt is issued. This happens even though there are not 14 characters in the buffer (there may only be one character in it). Thus when what you type goes thru this buffer, it acts almost like a 1−byte buffer even though it is actually a 16−byte buffer (unless your typing speed is a hundred times faster than normal). There is also "timeout" for the transmit buffer as well.

15.4 UART Model Numbers

Here's a list of UARTs. *TL* is *T*rigger *L*evel

- 8250, 16450, early 16550: Obsolete with 1−byte buffers
- 16550, 16550A, 16c552: 16−byte buffers, TL=1,4,8,14
- 16650: 32−byte buffers. Speed up to 460.8 kbps
- 16750: 64−byte buffer for send, 56−byte for receive. Speed up to 921.6 kbps

- Hayes ESP: 1k−byte buffers.

The obsolete ones are only good for modems no higher than 14.4k (DTE speeds up to 38400 bps). For modern modems you need at least a 16550 (and not an early 16550). For V.90 56k modems, it may be a several percent faster with a 16650 (especially if you are downloading uncompressed files). The main advantage of the 16650 is its larger buffer size as the extra speed isn't needed unless the modem compression ratio is high. Some 56k internal modems may come with a 16650 ??

Non−UART, and intelligent multiport boards use DSP chips to do additional buffering and control, thus relieving the CPU even more. For example, the Cyclades Cyclom, and Stallion EasyIO boards use a Cirrus Logic CD1400 RISC UART, and many boards use 80186 CPUs or even special RISC CPUs, to handle the serial IO.

Most newer PC's (486's, Pentiums, or better) come with 16550A's (usually called just 16550's). If you have something really old the chip may unplug so that you may be able to upgrade by buying a 16550A chip and replacing your existing 16450 UART. If the functionality has been put on another type of chip, you are out of luck. If the UART is socketed, then upgrading is easy (if you can find a replacement). The new and old are pin−to−pin compatible. It may be more feasible to just buy a new serial board on the Internet (few retail stores stock them today).

16.Troubleshooting

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

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don't know what serial port it's connected to then the problem is to find the serial port. See My Serial Port is Physically There but Can't be Found This section is about finding out which serial port has the modem on it.

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 No response to AT

Your problem could be due to a winmodem (or the like) which can't be used with Linux. See Avoid most software modems The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

Another way try to find out if there's a modem on a port is to start "minicom" on the port (after first setting up minicon for the correct 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 (or 0 if it's set for "digit result codes"). The results may be:

- No response. See No response to AT

- 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 stange 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.

No response to AT

The modem should send you "OK" in response to your "AT" which you type to the modem (using minicom or the like). If you don't see "OK" (and in most cases don't even see the "AT" you typed either) the modem is not responding (assuming there is really a modem on the port you are typing to).

One reason that a real modem doesn't respond is that it is in "online data" mode where it can't accept any AT commands. It may be in use by another process. If such a process is running on the port you may see it by typing "ps −t ttyS2" or the like. However the process that's using the serial port (where the modem is) may be running on a terminal such at /dev/tty1 and will not be found using the above command.

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. Thus the message from minicom "You are already online. Hangup 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, putting an unexpected +++ is likely to set off an exchange of packets (or the like) 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.

16.2 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).

16.3 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.

16.4 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`.

16.5 I Keep Getting: ``Id "S3" respawning too fast: disabled for 5 minutes"

Id "S3" is just an example. In this case look on the line which starts with "S3" in `/etc/inittab`. This is causing the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found.

Make sure your modem is configured correctly. Look at registers `E` and `Q`. This can occur when your modem is chatting with `getty`.

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.

16.6 My Modem is Hosed after Someone Hangs Up, or uugetty doesn't respawn

This can happen when your modem doesn't reset when DTR is dropped. Greg Hankins saw his RD and SD LEDs go crazy when this happened. You need to have your modem reset. Most Hayes compatible modems do this with `&D3`, but on his USR Courier, he had to set `&D2` and `S13=1`. Check your modem manual (if you have one).

16.7 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.

16.8 The following subsections are in both the Serial and Modem HOWTOs:

16.9 My Serial Port is Physically There but Can't be Found

If a device (such as a modem) give evidence of working, then the serial port that it's on has been found. If it doesn't work at all, then you need to make sure your serial port can be found.

Check the BIOS menus and BIOS messages. For the PCI bus use lspci. If it's an ISA bus PnP serial port, try "pnpdump −−dumpregs" and/or see Plug−and−Play−HOWTO. Using "scanport" will scan all ISA 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. You may try probing with setserial. See Probing. If nothing seems to get thru the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays.

If two ports have the same IO address then probing it will erroneously indicate only one port. Plug−and−play detection will find both ports so this should only be a problem if at least one port is not plug−and−play. All sorts of errors may be reported/observed for devices on "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). If the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect an IRQ. See Probing.

16.10 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 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 and sometimes drastically decreases your thruput.

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 incorrect information. 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.

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

One 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 the serial driver thinks you have an obsolete serial port (UART 8250,16450 or early 16550). See What Are UARTs?. Use "setserial −g /dev/ttyS*". If it shows anything less than a 16550A, this is likely your problem. Then if "setserial" has it wrong, change it. See What is Setserial for more info. Of course if you really do have an obsolete serial port, lying about it to setserial will only make things worse.

16.12 The Startup Screen Show Wrong IRQs for the Serial 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 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") You have to use `setserial` to tell Linux the IRQ you are using.

16.13 "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 should show 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 ways to get access like belonging to a "group" that has group permission.

16.14 "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.

16.15 "Cannot create lockfile. Sorry"

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: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. 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".

16.16 "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 261 type "ps 261" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 261". If it refuses to be killed use "kill −9 261" 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 161 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 261).

16.17 "/dev/ttyS?: 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. Sometimes it actually is "busy" but in other cases it erroneously appears to be "busy".

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 "... resource busy" error message. This is because the kernel only keeps track of what IRQs are actually in use and conflicts don't happen unless the devices are in use (open).

There are two cases. There may be a real interrupt conflict that is being avoided. But if setserial has it wrong, there may be no reason why `ttyS2` can't be used, except that setserial erroneously predicts a conflict. What you need to do is to find the interrupt setserial thinks `ttyS2` is using. This is easier said than done since you can't use the "setserial" command for `ttyS2` since the IRQ for ttyS2 is supposedly "busy" and you will get the same "... busy" error message. To fix this either 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 `ttyS2` is using then you may look at /proc/interrupts to find what else 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 is 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 conflcit. It's not a good idea to leave it permanently set at 0 since more CPU resources will be used.

This paragraph is mainly for the case when a modem is used for both dial−in and dial−out. If the DCD signal is sent to a port, that port will think it's busy. This problem can arise when you are trying to dial out with a modem when DCD or DTR are not implemented correctly. DCD should only be on (asserted) 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. DTR should be on (asserted) whenever something is using, or watching the line, like `getty`, `kermit`, or some other comm program.

16.18 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" will show the current state of various modem signal lines (such as DTR, CTS, etc.)
- "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 and interrupts).

---

17. 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/ttyS2` (if your serial port is ttyS2). 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"
```

18.Other Sources of Information

18.1 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.

18.2 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.)
- Ultimate Modem Handbook by Cass R. Lewart, 1998.

18.3 HOWTOs

- Cable−Modem mini−howto
- ISDN Howto (not a LDP Howto) http://www.suse.de/Support/sdb_e/isdn.html: drivers for ISDN "Modems". Much related info on this is in German.
- 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 both terminals and banks of modems. Covers the serial port in more detail than in the 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

18.4 Usenet newsgroups

- comp.os.linux.answers FAQs, How−To's, README's, 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

18.5 Web Sites

- Modem List of modems which work/don't_work under Linux http://www.o2.net/~gromitkc/winmodem.html
- Hayes AT modem commands Technical Reference for Hayes (tm) Modem Users
- Rockwell−based modem commands
- Modem FAQs:
  Navas 28800 Modem FAQ
- Curt's High Speed Modem Page

- Much info on 56k modems 56k Modem = v.Unreliable
- Links to modem manufacturers
- Identifying modems by FCC ID

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

19.1 Modulation Details

Intro to Modulation

This part describes the modulation methods used for conventional modems. It doesn't cover the high speed methods (modulus conversion) sometimes used by 56k Modems (v.90). But 56k modems also use the modulation methods described here.

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 (sometimes also called quadrature amplitude modulation = QAM). 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.

19.2 56k Modems (v.90)

The "modulation" method used above 33.6k is entirely different than the common phase−amplitude modulation. 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 the phone call transmission. What is this speed? Well, it's close to 64kbps. It would be 64k but sometimes bits are "stolen" for signalling purposes. But 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 −−usually significantly less).

Thus 64k is the absolute top speed possible 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 modem must know exactly how the telephone company is doing its digital encoding of the analog signals. This task is far too complicated 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 (at least in one direction). 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 (ASCII−like) 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 the local telephone office near your home, the digital signal is de−multiplexed resulting in the same digital signal as was originally created by PCM. This signal is then converted back to analog and sent to your home. Each 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 is (sort of) "amplitude demodulation" but not really. It's not amplitude demodulation because there is no carrier. Actually, it's called "modulus conversion" which is the inverse of PCM. 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. used when it created the analog signal. To do this a timing signal is generated 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".

Now the encoding of amplitudes in PCM is not linear. At low amplitudes an increment of 1 in the PCM byte represents a much smaller increment in analog signal amplitude than would be the case if the amplitude being sampled were higher. Thus for low amplitudes it's difficult to distinguish between adjacent byte values. To make it easier to do this certain PCM codes representing very low amplitudes are not used. This give a larger delta between possible amplitudes and makes correct detection of them by your modem easier. Thus half the amplitude levels are not used by v.90. This is tantamount to each symbol (allowed 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 symbol is actually generated by 8−bits but only 128 bytes of the possible 256 bytes are actually used. There is a code table mapping these 128 8−bit bytes to 128 7−bit bytes.

But it's a little more complicated that this. If the line conditions are not nearly perfect, 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.

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. The 56k signal is only used in one of these directions: from your ISP to your PC. The other direction, from your home/office to the ISP, uses the conventional phase−amplitude modulation scheme with a maximum of 36.6kbps (and not 53.3kbps). Yet due to sophisticated cancellation methods (not explained here) it's able to send simultaneously in both directions.

19.3 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 portions of the signal path go 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 it were to start transmitting (with the other received signal still flowing into the 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.

19.4 Echo Cancellation

A 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.

20.Appendix B: Digital Modem Signal Processing (not done)

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

21.1 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 ... 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 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.

21.2 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. Using modems with V.42bis compression (max 4:1 compression), rates up to 115.2k bps are possible for 33.6k modems (230.4k 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.

22.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.

23.Appendix E: Other Types of Modems

This HOWTO currently only deals with the common type of modem used to connect PC's to ordinary analog telephone lines. There are various other types of modems, including devices called modems that are not really modems.

23.1 Digital−to−Digital "Modems"

The standard definition of a modem is sometimes broadened to include "digital" modems. Today direct digital service is now being provided to many homes and offices so a computer there sends out digital signals directly (well almost) into the telephone lines. But a device is still needed to convert the computer digital signal into the type allowed on telephone circuits and this device is sometimes called a modem. This HOWTO doesn't cover such modems but some links to documents that do may be found at the start of this HOWTO. The next 3 sections: ISDN, DSL and 56k, concern digital−to−digital "modems".

23.2 ISDN "Modems"

The "modem" is really a Terminal Adapter (TA). A Debian package "isdnutils" is available. There is a ISDN Howto in German with an English translation: http://www.suse.de/Support/sdb_e/isdn.html. It's put out by the SuSE distribution of Linux and likely is about drivers available in that distribution. There is an isdn4linux package and a newsgroup: de.alt.comm.isdn4linux. Many of the postings are in German. You might try using a search engine (such as DejaNews) to find "isdn4linux".

23.3 Digital Subscriber Line (DSL)

DSL uses the existing twisted pair line from your home (etc.) to the local telephone office. This can be used if your telephone line can accept higher speeds than an ordinary modem (say 56k) sends over it. It replaces the analog−to−digital converter at the local telephone office with a converter which can accept a much faster flow of data (in a different format of course). The device which converts the digital signals from your computer to the signal used to represent digital data on the local telephone line is also called a modem.

23.4 56k Digital−Modems

For any 56k modem to work as a 56k modem in your home or office the other end must be connected directly to the digital system of the telephone company. Thus ISPs at the other end of the line must obtain special digital modems to provide customers with 56k service. There's more to it than this since banks of many modems are multiplexed onto a high capacity telephone cable that transports a large number of phone calls simultaneously (such as a T1, E1, ISDN PRI, or better line). This requires a concentrator or "remote access server". This has usually been done by stand−alone units (like PC's but they cost much more and have proprietary OSs). Now there are some cards one may insert into a PC's PCI bus to do this.

23.5 Leased Line Modems

These are analog and not digital modems. These special modems are used on lines leased from the telephone company or sometimes on just a long direct wire hookup. Ordinary modems for a telephone line will not normally work on such a line. An ordinary telephone line has about 40−50 volts (know as the "battery) on it when not in use and the conventional modem uses this voltage for transmission. Furthermore, the telephone company has special signals indicating a ring, line busy, etc. Conventional modems expect and respond to these signals. Connecting two such modems by a long cable will not provide the telephone signals on the cable and thus the modems will not work.

A common 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.

24.Appendix F: Fax pixels (dots)

```
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.

END OF Modem−HOWTO
