NAME

setserial − get/set Linux serial port information

SYNOPSIS

setserial [−abqvVWz] device **[** parameter1 **[** arg **]] ...**

setserial -g [−abGv] device1 ...

DESCRIPTION

setserial is a program designed to set and/or report the configuration information associated with a serial port. This information includes what I/O port and IRQ a particular serial port is using, and whether or not the break key should be interpreted as the Secure Attention Key, and so on.

During the normal bootup process, only COM ports 1-4 are initialized, using the default I/O ports and IRQ values, as listed below. In order to initialize any additional serial ports, or to change the COM 1-4 ports to a nonstandard configuration, the **setserial** program should be used. Typically it is called from an *setserial* script, which is usually run out of */etc/init.d*.

The *device* argument specifies which device to configure or to interrogate. Examples: /dev/ttyS0, /dev/ttyS1, /dev/ttyS2, /dev/ttyS3, etc.

If no parameters are specified, **setserial** will print out the port type (i.e., 8250, 16450, 16550, 16550A, etc.), the hardware I/O port, the hardware IRQ line, its "baud base," and some of its operational flags.

If the **−g** option is given, the arguments to setserial are interpreted as a list of devices for which the characteristics of those devices should be printed.

Without the **−g** option, the first argument to setserial is interpreted as the device to be modified or characteristics to be printed, and any additional arguments are interpreted as parameters which should be assigned to that serial device.

For the most part, superuser privilege is required to set the configuration parameters of a serial port. A few serial port parameters can be set by normal users, however, and these will be noted as exceptions in this manual page.

OPTIONS

Setserial accepts the following options:

−a When reporting the configuration of a serial device, print all available information.

−b When reporting the configuration of a serial device, print a summary of the device's configuration, which might be suitable for printing during the bootup process, during the /etc/rc script.

−G Print out the configuration information of the serial port in a form which can be fed back to setserial as command-line arguments.

−q Be quiet. **Setserial** will print fewer lines of output.

−v Be verbose. **Setserial** will print additional status output.

−V Display version and exit.

−W Do wild interrupt initialization and exit. This option is no longer relevant in Linux kernels after version 2.1.

−z Zero out the serial flags before starting to set flags. This is related to the automatic saving of serial flags using the −G flag.

PARAMETERS

The following parameters can be assigned to a serial port.

All argument values are assumed to be in decimal unless preceded by "0x".

port port_number
The **port** option sets the I/O port, as described above.

irq irq_number
The **irq** option sets the hardware IRQ, as described above.

uart uart_type
This option is used to set the UART type. The permitted types are **none**, 8250, 16450, 16550, 16550A, 16650, 16650V2, 16654, 16750, 16850, 16950, and 16954. Using UART type **none** will disable the port.

Some internal modems are billed as having a "16550A UART with a 1k buffer". This is a lie. They do not have really have a 16550A compatible UART; instead what they have is a 16450 compatible UART with a 1k receive buffer to prevent receiver overruns. This is important, because they do not have a transmit FIFO. Hence, they are not compatible with a 16550A UART, and the autoconfiguration process will correctly identify them as 16450's. If you attempt to override this using the **uart** parameter, you will see dropped characters during file transmissions. These UART's usually have other problems: the **skip_test** parameter also often must be specified.

autoconfig
When this parameter is given, **setserial** will ask the kernel to attempt to automatically configure the serial port. The I/O port must be correctly set; the kernel will attempt to determine the UART type, and if the **auto_irq** parameter is set, Linux will attempt to automatically determine the IRQ. The **autoconfig** parameter should be given after the **port**,**auto_irq**, and **skip_test** parameters have been specified.

auto_irq
During autoconfiguration, try to determine the IRQ. This feature is not guaranteed to always produce the correct result; some hardware configurations will fool the Linux kernel. It is generally safer not to use the **auto_irq** feature, but rather to specify the IRQ to be used explicitly, using the **irq** parameter.

^auto_irq
During autoconfiguration, do *not* try to determine the IRQ.

skip_test
During autoconfiguration, skip the UART test. Some internal modems do not have National Semiconductor compatible UART's, but have cheap imitations instead. Some of these cheesy imitations UART's do not fully support the loopback detection mode, which is used by the kernel to make sure there really is a UART at a particular address before attempting to configure it. So for certain internal modems you will need to specify this parameter so Linux can initialize the UART correctly.

^skip_test
During autoconfiguration, do *not* skip the UART test.

baud_base baud_base
This option sets the base baud rate, which is the clock frequency divided by 16. Normally this value is 115200, which is also the fastest baud rate which the UART can support.

spd_hi Use 57.6kb when the application requests 38.4kb. This parameter may be specified by a non-privileged user.

spd_vhi
Use 115kb when the application requests 38.4kb. This parameter may be specified by a non-privileged user.

spd_shi
Use 230kb when the application requests 38.4kb. This parameter may be specified by a non-privileged user.

spd_warp
Use 460kb when the application requests 38.4kb. This parameter may be specified by a non-privileged user.

spd_cust
Use the custom divisor to set the speed when the application requests 38.4kb. In this case, the baud rate is the **baud_base** divided by the **divisor**. This parameter may be specified by a non-privileged user.

spd_normal
Use 38.4kb when the application requests 38.4kb. This parameter may be specified by a non-privileged user.

divisor divisor
This option sets the custom divisor. This divisor will be used when the **spd_cust** option is selected and the serial port is set to 38.4kb by the application. This parameter may be specified by a non-privileged user.

sak Set the break key at the Secure Attention Key.

^sak disable the Secure Attention Key.

fourport
Configure the port as an AST Fourport card.

^fourport
Disable AST Fourport configuration.

close_delay delay
Specify the amount of time, in hundredths of a second, that DTR should remain low on a serial line after the callout device is closed, before the blocked dialin device raises DTR again. The default value of this option is 50, or a half-second delay.

closing_wait delay
Specify the amount of time, in hundredths of a second, that the kernel should wait for data to be transmitted from the serial port while closing the port. If "none" is specified, no delay will occur. If "infinite" is specified the kernel will wait indefinitely for the buffered data to be transmitted. The default setting is 3000 or 30 seconds of delay. This default is generally appropriate for most devices. If too long a delay is selected, then the serial port may hang for a long time if when a serial port which is not connected, and has data pending, is closed. If too short a delay is selected, then there is a risk that some of the transmitted data is output at all. If the device is extremely slow, like a plotter, the closing_wait may need to be larger.

session_lockout
Lock out callout port (/dev/cuaXX) accesses across different sessions. That is, once a process has opened a port, do not allow a process with a different session ID to open that port until the first process has closed it.

^session_lockout
Do not lock out callout port accesses across different sessions.

pgrp_lockout
Lock out callout port (/dev/cuaXX) accesses across different process groups. That is, once a process has opened a port, do not allow a process in a different process group to open that port until the first process has closed it.

^pgrp_lockout
Do not lock out callout port accesses across different process groups.

hup_notify
Notify a process blocked on opening a dialin line when a process has finished using a callout line (either by closing it or by the serial line being hung up) by returning EAGAIN to the open.

The application of this parameter is for getty's which are blocked on a serial port's dialin line. This allows the getty to reset the modem (which may have had its configuration modified by the application using the callout device) before blocking on the open again.

^hup_notify
Do not notify a process blocked on opening a dialin line when the callout device is hung up.

split_termios
Treat the termios settings used by the callout device and the termios settings used by the dialin devices as separate.

^split_termios
Use the same termios structure to store both the dialin and callout ports. This is the default option.

callout_nohup
If this particular serial port is opened as a callout device, do not hangup the tty when carrier detect is dropped.

^callout_nohup
Do not skip hanging up the tty when a serial port is opened as a callout device. Of course, the HUPCL termios flag must be enabled if the hangup is to occur.

low_latency
Minimize the receive latency of the serial device at the cost of greater CPU utilization. (Normally there is an average of 5-10ms latency before characters are handed off to the line discipline to minimize overhead.) This is off by default, but certain real-time applications may find this useful.

^low_latency
Optimize for efficient CPU processing of serial characters at the cost of paying an average of 5-10ms of latency before the characters are processed. This is the default.

CONSIDERATIONS OF CONFIGURING SERIAL PORTS

It is important to note that setserial merely tells the Linux kernel where it should expect to find the I/O port and IRQ lines of a particular serial port. It does *not* configure the hardware, the actual serial board, to use a particular I/O port. In order to do that, you will need to physically program the serial board, usually by setting some jumpers or by switching some DIP switches.

This section will provide some pointers in helping you decide how you would like to configure your serial ports.

The "standard MS-DOS" port associations are given below:

/dev/ttyS0 (COM1), port 0x3f8, irq 4
/dev/ttyS1 (COM2), port 0x2f8, irq 3
/dev/ttyS2 (COM3), port 0x3e8, irq 4
/dev/ttyS3 (COM4), port 0x2e8, irq 3

Due to the limitations in the design of the AT/ISA bus architecture, normally an IRQ line may not be shared between two or more serial ports. If you attempt to do this, one or both serial ports will become unreliable if you try to use both simultaneously. This limitation can be overcome by special multi-port serial port boards, which are designed to share multiple serial ports over a single IRQ line. Multi-port serial cards supported by Linux include the AST FourPort, the Accent Async board, the Usenet Serial II board, the Bocaboard BB-1004, BB-1008, and BB-2016 boards, and the HUB-6 serial board.

The selection of an alternative IRQ line is difficult, since most of them are already used. The following table lists the "standard MS-DOS" assignments of available IRQ lines:

IRQ 3: COM2
IRQ 4: COM1
IRQ 5: LPT2
IRQ 7: LPT1

Most people find that IRQ 5 is a good choice, assuming that there is only one parallel port active in the computer. Another good choice is IRQ 2 (aka IRQ 9); although this IRQ is sometimes used by network cards, and very rarely VGA cards will be configured to use IRQ 2 as a vertical retrace interrupt. If your VGA card is configured this way; try to disable it so you can reclaim that IRQ line for some other card. It's not necessary for Linux and most other Operating systems.

The only other available IRQ lines are 3, 4, and 7, and these are probably used by the other serial and parallel ports. (If your serial card has a 16bit card edge connector, and supports higher interrupt numbers, then IRQ 10, 11, 12, and 15 are also available.)

On AT class machines, IRQ 2 is seen as IRQ 9, and Linux will interpret it in this manner.

IRQ's other than 2 (9), 3, 4, 5, 7, 10, 11, 12, and 15, should *not* be used, since they are assigned to other hardware and cannot, in general, be changed. Here are the "standard" assignments:

IRQ 0	Timer channel 0
IRQ 1	Keyboard
IRQ 2	Cascade for controller 2
IRQ 3	Serial port 2
IRQ 4	Serial port 1
IRQ 5	Parallel port 2 (Reserved in PS/2)
IRQ 6	Floppy diskette
IRQ 7	Parallel port 1
IRQ 8	Real-time clock
IRQ 9	Redirected to IRQ2
IRQ 10	Reserved
IRQ 11	Reserved
IRQ 12	Reserved (Auxiliary device in PS/2)
IRQ 13	Math coprocessor
IRQ 14	Hard disk controller
IRQ 15	Reserved

MULTIPORT CONFIGURATION

Certain multiport serial boards which share multiple ports on a single IRQ use one or more ports to indicate whether or not there are any pending ports which need to be serviced. If your multiport board supports these ports, you should make use of them to avoid potential lockups if the interrupt gets lost.

In order to set these ports specify **set_multiport** as a parameter, and follow it with the multiport parameters. The multiport parameters take the form of specifying the *port* that should be checked, a *mask* which indicate which bits in the register are significant, and finally, a *match* parameter which specifies what the significant bits in that register must match when there is no more pending work to be done.

Up to four such port/mask/match combinations may be specified. The first such combinations should be specified by setting the parameters **port1**, **mask1**, and **match1**. The second such combination should be specified with **port2**, **mask2**, and **match2**, and so on. In order to disable this multiport checking, set **port1** to be zero.

In order to view the current multiport settings, specify the parameter **get_multiport** on the command line.

Here are some multiport settings for some common serial boards:

AST FourPort port1 0x1BF mask1 0xf match1 0xf

Boca BB-1004/8 port1 0x107 mask1 0xff match1 0

Boca BB-2016 port1 0x107 mask1 0xff match1 0
port2 0x147 mask2 0xff match2 0

Hayes ESP Configuration

Setserial may also be used to configure ports on a Hayes ESP serial board.

The following parameters when configuring ESP ports:

rx_trigger
This is the trigger level (in bytes) of the receive FIFO. Larger values may result in fewer interrupts and hence better performance; however, a value too high could result in data loss. Valid values are 1 through 1023.

tx_trigger
This is the trigger level (in bytes) of the transmit FIFO. Larger values may result in fewer interrupts and hence better performance; however, a value too high could result in degraded transmit performance. Valid values are 1 through 1023.

flow_off
This is the level (in bytes) at which the ESP port will "flow off" the remote transmitter (i.e. tell him to stop stop sending more bytes). Valid values are 1 through 1023. This value should be greater than the receive trigger level and the flow on level.

flow_on
This is the level (in bytes) at which the ESP port will "flow on" the remote transmitter (i.e. tell him to resume sending bytes) after having flowed it off. Valid values are 1 through 1023. This value should be less than the flow off level, but greater than the receive trigger level.

rx_timeout
This is the amount of time that the ESP port will wait after receiving the final character before signaling an interrupt. Valid values are 0 through 255. A value too high will increase latency, and a value too low will cause unnecessary interrupts.

CAUTION

CAUTION: Configuring a serial port to use an incorrect I/O port can lock up your machine.

FILES

/etc/serial.conf /etc/init.d/setserial

SEE ALSO

tty(4), **ttys**(4), kernel/chr_drv/serial.c

AUTHOR

The original version of setserial was written by Rick Sladkey (jrs@world.std.com), and was modified by Michael K. Johnson (johnsonm@stolaf.edu).

This version has since been rewritten from scratch by Theodore Ts'o (tytso@mit.edu) on 1/1/93. Any bugs or problems are solely his responsibility.

Debian related problems with this system should be sent to Gordon Russell (gor@debian.org).