MyWebUniversity
python
Rocky Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'lirc.4'

$ man lirc.4

LIRC(4)   Linux Programmer's Manual   LIRC(4)

NAME

lirc - lirc devices

DESCRIPTION

The /dev/lirc* character devices provide a low-level bidirectional in?
terface to infra-red (IR) remotes. Most of these devices can receive,
and some can send. When receiving or sending data, the driver works in
two different modes depending on the underlying hardware.

Some hardware (typically TV-cards) decodes the IR signal internally and
provides decoded button presses as scancode values. Drivers for this
kind of hardware work in LIRC_MODE_SCANCODE mode. Such hardware usu?
ally does not support sending IR signals. Furthermore, such hardware
can only decode a limited set of IR protocols, usually only the proto?
col of the specific remote which is bundled with, for example, a TV-
card.

Other hardware provides a stream of pulse/space durations. Such driv?
ers work in LIRC_MODE_MODE2 mode. Sometimes, this kind of hardware
also supports sending IR data. Such hardware can be used with (almost)

any kind of remote. This type of hardware can also be used in LIRC_MODE_SCANCODE mode, in which case the kernel IR decoders will de? code the IR. These decoders can be written in extended BPF (see bpf(2)) and attached to the lirc device.

The LIRC_GET_FEATURES ioctl (see below) allows probing for whether re? ceiving and sending is supported, and in which modes, amongst other features.

Reading input with the LIRC_MODE_MODE2 mode

In the LIRC_MODE_MODE2 mode, the data returned by read(2) provides 32-bit values representing a space or a pulse duration. The time of the duration (microseconds) is encoded in the lower 24 bits. The upper 8 bits indicate the type of package:

LIRC_MODE2_SPACE

Value reflects a space duration (microseconds).

LIRC_MODE2_PULSE

Value reflects a pulse duration (microseconds).

LIRC_MODE2_FREQUENCY

Value reflects a frequency (Hz); see the LIRC_SET_MEASURE_CAR? RIER_MODE ioctl.

LIRC_MODE2_TIMEOUT

Value reflects a space duration (microseconds). The package re? flects a timeout; see the LIRC_SET_REC_TIMEOUT_REPORTS ioctl.

Reading input with the LIRC_MODE_SCANCODE mode

In the LIRC_MODE_SCANCODE mode, the data returned by read(2) reflects decoded button presses, in the struct lirc_scancode. The scancode is stored in the scancode field, and the IR protocol is stored in rc_proto. This field has one the values of the enum rc_proto.

Writing output with the LIRC_MODE_PULSE mode

The data written to the character device using write(2) is a pulse/space sequence of integer values. Pulses and spaces are only marked implicitly by their position. The data must start and end with a pulse, thus it must always include an odd number of samples. The write(2) function blocks until the data has been transmitted by the hardware.   If more data is provided than the hardware can send, the write(2) call fails with the error EINVAL.

Writing output with the LIRC_MODE_SCANCODE mode

The data written to the character devices must be a single struct lirc_scancode.   The scancode and rc_proto fields must filled in, all other fields must be 0. The kernel IR encoders will convert the scan? code to pulses and spaces. The protocol or scancode is invalid, or the lirc device cannot transmit.

IOCTL COMMANDS

The LIRC device's ioctl definition is bound by the ioctl function defi? nition of struct file_operations, leaving us with an unsigned int for the ioctl command and an unsigned long for the argument. For the pur? poses of ioctl portability across 32-bit and 64-bit architectures, these values are capped to their 32-bit sizes.

```
#include <linux/lirc.h>    /* But see BUGS */
int ioctl(int fd, int cmd, ...);
```

The following ioctls can be used to probe or change specific lirc hard? ware settings.   Many require a third argument, usually an int. re? ferred to below as val.

Always Supported Commands

/dev/lirc* devices always support the following commands:

LIRC_GET_FEATURES (void)

Returns a bit mask of combined features bits; see FEATURES.

If a device returns an error code for LIRC_GET_FEATURES, it is safe to assume it is not a lirc device.

Optional Commands

Some lirc devices support the commands listed below. Unless otherwise stated, these fail with the error ENOTTY if the operation isn't sup? ported, or with the error EINVAL if the operation failed, or invalid arguments were provided. If a driver does not announce support of cer? tain features, invoking the corresponding ioctls will fail with the er? ror ENOTTY.

LIRC_GET_REC_MODE (void)

If the lirc device has no receiver, this operation fails with the error ENOTTY. Otherwise, it returns the receive mode, which will be one of:

LIRC_MODE_MODE2

The driver returns a sequence of pulse/space durations.

LIRC_MODE_SCANCODE

The driver returns struct lirc_scancode values, each of which represents a decoded button press.

LIRC_SET_REC_MODE (int)

Set the receive mode. val is either LIRC_MODE_SCANCODE or LIRC_MODE_MODE2. If the lirc device has no receiver, this oper? ation fails with the error ENOTTY.

LIRC_GET_SEND_MODE (void)

Return the send mode. LIRC_MODE_PULSE or LIRC_MODE_SCANCODE is supported. If the lirc device cannot send, this operation fails with the error ENOTTY.

LIRC_SET_SEND_MODE (int)

Set the send mode. val is either LIRC_MODE_SCANCODE or LIRC_MODE_PULSE. If the lirc device cannot send, this operation fails with the error ENOTTY.

LIRC_SET_SEND_CARRIER (int)

Set the modulation frequency. The argument is the frequency (Hz).

LIRC_SET_SEND_DUTY_CYCLE (int)

Set the carrier duty cycle. val is a number in the range [0,100] which describes the pulse width as a percentage of the total cycle. Currently, no special meaning is defined for 0 or 100, but the values are reserved for future use.

LIRC_GET_MIN_TIMEOUT (void), LIRC_GET_MAX_TIMEOUT (void)

Some devices have internal timers that can be used to detect when there has been no IR activity for a long time. This can help lircd(8) in detecting that an IR signal is finished and can speed up the decoding process. These operations return integer

values with the minimum/maximum timeout that can be set (mi?
croseconds). Some devices have a fixed timeout. For such driv?
ers, LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT will fail
with the error ENOTTY.

LIRC_SET_REC_TIMEOUT (int)

Set the integer value for IR inactivity timeout (microseconds).
To be accepted, the value must be within the limits defined by
LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT. A value of 0 (if
supported by the hardware) disables all hardware timeouts and
data should be reported as soon as possible. If the exact value
cannot be set, then the next possible value greater than the
given value should be set.

LIRC_GET_REC_TIMEOUT (void)

Return the current inactivity timeout (microseconds). Available
since Linux 4.18.

LIRC_SET_REC_TIMEOUT_REPORTS (int)

Enable (val is 1) or disable (val is 0) timeout packages in
LIRC_MODE_MODE2. The behavior of this operation has varied
across kernel versions:

* Since Linux 4.16: each time the lirc device is opened, time?
out reports are by default enabled for the resulting file de?
scriptor. The LIRC_SET_REC_TIMEOUT operation can be used to
disable (and, if desired, to later re-enable) the timeout on
the file descriptor.

* In Linux 4.15 and earlier: timeout reports are disabled by
default, and enabling them (via LIRC_SET_REC_TIMEOUT) on any
file descriptor associated with the lirc device has the ef?
fect of enabling timeouts for all file descriptors referring
to that device (until timeouts are disabled again).

LIRC_SET_REC_CARRIER (int)

Set the upper bound of the receive carrier frequency (Hz). See
LIRC_SET_REC_CARRIER_RANGE.

LIRC_SET_REC_CARRIER_RANGE (int)

Sets the lower bound of the receive carrier frequency (Hz). For this to take affect, first set the lower bound using the LIRC_SET_REC_CARRIER_RANGE ioctl, and then the upper bound using the LIRC_SET_REC_CARRIER ioctl.

LIRC_SET_MEASURE_CARRIER_MODE (int)

Enable (val is 1) or disable (val is 0) the measure mode. If enabled, from the next key press on, the driver will send LIRC_MODE2_FREQUENCY packets. By default, this should be turned off.

LIRC_GET_REC_RESOLUTION (void)

Return the driver resolution (microseconds).

LIRC_SET_TRANSMITTER_MASK (int)

Enable the set of transmitters specified in val, which contains a bit mask where each enabled transmitter is a 1. The first transmitter is encoded by the least significant bit, and so on. When an invalid bit mask is given, for example a bit is set even though the device does not have so many transmitters, this oper? ation returns the number of available transmitters and does nothing otherwise.

LIRC_SET_WIDEBAND_RECEIVER (int)

Some devices are equipped with a special wide band receiver which is intended to be used to learn the output of an existing remote. This ioctl can be used to enable (val equals 1) or dis? able (val equals 0) this functionality. This might be useful for devices that otherwise have narrow band receivers that pre? vent them to be used with certain remotes. Wide band receivers may also be more precise. On the other hand, their disadvantage usually is reduced range of reception.
Note: wide band receiver may be implicitly enabled if you enable carrier reports. In that case, it will be disabled as soon as you disable carrier reports. Trying to disable a wide band re? ceiver while carrier reports are active will do nothing.

the LIRC_GET_FEATURES ioctl returns a bit mask describing features of the driver. The following bits may be returned in the mask:

LIRC_CAN_REC_MODE2

  The driver is capable of receiving using LIRC_MODE_MODE2.

LIRC_CAN_REC_SCANCODE

  The driver is capable of receiving using LIRC_MODE_SCANCODE.

LIRC_CAN_SET_SEND_CARRIER

  The driver supports changing the modulation frequency using LIRC_SET_SEND_CARRIER.

LIRC_CAN_SET_SEND_DUTY_CYCLE

  The driver supports changing the duty cycle using LIRC_SET_SEND_DUTY_CYCLE.

LIRC_CAN_SET_TRANSMITTER_MASK

  The driver supports changing the active transmitter(s) using LIRC_SET_TRANSMITTER_MASK.

LIRC_CAN_SET_REC_CARRIER

  The driver supports setting the receive carrier frequency using LIRC_SET_REC_CARRIER. Any lirc device since the drivers were merged in kernel release 2.6.36 must have LIRC_CAN_SET_REC_CAR?
  RIER_RANGE set if LIRC_CAN_SET_REC_CARRIER feature is set.

LIRC_CAN_SET_REC_CARRIER_RANGE

  The driver supports LIRC_SET_REC_CARRIER_RANGE. The lower bound of the carrier must first be set using the LIRC_SET_REC_CAR?
  RIER_RANGE ioctl, before using the LIRC_SET_REC_CARRIER ioctl to set the upper bound.

LIRC_CAN_GET_REC_RESOLUTION

  The driver supports LIRC_GET_REC_RESOLUTION.

LIRC_CAN_SET_REC_TIMEOUT

  The driver supports LIRC_SET_REC_TIMEOUT.

LIRC_CAN_MEASURE_CARRIER

  The driver supports measuring of the modulation frequency using LIRC_SET_MEASURE_CARRIER_MODE.

LIRC_CAN_USE_WIDEBAND_RECEIVER

The driver supports learning mode using LIRC_SET_WIDEBAND_RE?
CEIVER.

LIRC_CAN_SEND_PULSE

The driver supports sending using LIRC_MODE_PULSE or
LIRC_MODE_SCANCODE

BUGS

Using these devices requires the kernel source header file lirc.h.
This file is not available before kernel release 4.6. Users of older
kernels could use the file bundled in ?http://www.lirc.org?.

SEE ALSO

ir-ctl(1), lircd(8), bpf(2)

https://www.kernel.org/doc/html/latest/media/uapi/rc/lirc-dev.html

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A
description of the project, information about reporting bugs, and the
latest version of this page, can be found at
https://www.kernel.org/doc/man-pages/.