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

$ man rtc.4

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

NAME

rtc - real-time clock

SYNOPSIS

#include <linux/rtc.h>

int ioctl(fd, RTC_request, param);

DESCRIPTION

This is the interface to drivers for real-time clocks (RTCs).

Most computers have one or more hardware clocks which record the cur?

rent "wall clock" time. These are called "Real Time Clocks" (RTCs).

One of these usually has battery backup power so that it tracks the

time even while the computer is turned off. RTCs often provide alarms

and other interrupts.

All i386 PCs, and ACPI-based systems, have an RTC that is compatible

with the Motorola MC146818 chip on the original PC/AT. Today such an

RTC is usually integrated into the mainboard's chipset (south bridge),

and uses a replaceable coin-sized backup battery.

Non-PC systems, such as embedded systems built around system-on-chip

processors, use other implementations. They usually won't offer the same functionality as the RTC from a PC/AT.

RTC vs system clock

RTCs should not be confused with the system clock, which is a software clock maintained by the kernel and used to implement gettimeofday(2) and time(2), as well as setting timestamps on files, and so on. The system clock reports seconds and microseconds since a start point, de? fined to be the POSIX Epoch: 1970-01-01 00:00:00 +0000 (UTC). (One common implementation counts timer interrupts, once per "jiffy", at a frequency of 100, 250, or 1000 Hz.) That is, it is supposed to report wall clock time, which RTCs also do.

A key difference between an RTC and the system clock is that RTCs run even when the system is in a low power state (including "off"), and the system clock can't. Until it is initialized, the system clock can only report time since system boot ... not since the POSIX Epoch. So at boot time, and after resuming from a system low power state, the system clock will often be set to the current wall clock time using an RTC. Systems without an RTC need to set the system clock using another clock, maybe across the network or by entering that data manually.

RTC functionality

RTCs can be read and written with hwclock(8), or directly with the ioctl(2) requests listed below.

Besides tracking the date and time, many RTCs can also generate inter? rupts

* on every clock update (i.e., once per second);
* at periodic intervals with a frequency that can be set to any power- of-2 multiple in the range 2 Hz to 8192 Hz;
* on reaching a previously specified alarm time.

Each of those interrupt sources can be enabled or disabled separately.

On many systems, the alarm interrupt can be configured as a system wakeup event, which can resume the system from a low power state such as Suspend-to-RAM (STR, called S3 in ACPI systems), Hibernation (called S4 in ACPI systems), or even "off" (called S5 in ACPI systems). On

some systems, the battery backed RTC can't issue interrupts, but an? other one can.

The /dev/rtc (or /dev/rtc0, /dev/rtc1, etc.) device can be opened only once (until it is closed) and it is read-only. On read(2) and se? lect(2) the calling process is blocked until the next interrupt from that RTC is received. Following the interrupt, the process can read a long integer, of which the least significant byte contains a bit mask encoding the types of interrupt that occurred, while the remaining 3 bytes contain the number of interrupts since the last read(2).

ioctl(2) interface

The following ioctl(2) requests are defined on file descriptors con? nected to RTC devices:

RTC_RD_TIME

Returns this RTC's time in the following structure:

```
struct rtc_time {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;     /* unused */
    int tm_yday;     /* unused */
    int tm_isdst;    /* unused */
};
```

The fields in this structure have the same meaning and ranges as for the tm structure described in gmtime(3). A pointer to this structure should be passed as the third ioctl(2) argument.

RTC_SET_TIME

Sets this RTC's time to the time specified by the rtc_time structure pointed to by the third ioctl(2) argument. To set the RTC's time the process must be privileged (i.e., have the CAP_SYS_TIME capability).

RTC_ALM_READ, RTC_ALM_SET

Read and set the alarm time, for RTCs that support alarms. The alarm interrupt must be separately enabled or disabled using the RTC_AIE_ON, RTC_AIE_OFF requests. The third ioctl(2) argument is a pointer to an rtc_time structure. Only the tm_sec, tm_min, and tm_hour fields of this structure are used.

RTC_IRQP_READ, RTC_IRQP_SET

Read and set the frequency for periodic interrupts, for RTCs that support periodic interrupts. The periodic interrupt must be separately enabled or disabled using the RTC_PIE_ON, RTC_PIE_OFF requests. The third ioctl(2) argument is an un? signed long * or an unsigned long, respectively. The value is the frequency in interrupts per second. The set of allowable frequencies is the multiples of two in the range 2 to 8192. Only a privileged process (i.e., one having the CAP_SYS_RESOURCE capability) can set frequencies above the value specified in /proc/sys/dev/rtc/max-user-freq. (This file contains the value 64 by default.)

RTC_AIE_ON, RTC_AIE_OFF

Enable or disable the alarm interrupt, for RTCs that support alarms. The third ioctl(2) argument is ignored.

RTC_UIE_ON, RTC_UIE_OFF

Enable or disable the interrupt on every clock update, for RTCs that support this once-per-second interrupt. The third ioctl(2) argument is ignored.

RTC_PIE_ON, RTC_PIE_OFF

Enable or disable the periodic interrupt, for RTCs that support these periodic interrupts. The third ioctl(2) argument is ig? nored. Only a privileged process (i.e., one having the CAP_SYS_RESOURCE capability) can enable the periodic interrupt if the frequency is currently set above the value specified in /proc/sys/dev/rtc/max-user-freq.

RTC_EPOCH_READ, RTC_EPOCH_SET

Many RTCs encode the year in an 8-bit register which is either interpreted as an 8-bit binary number or as a BCD number. In both cases, the number is interpreted relative to this RTC's Epoch. The RTC's Epoch is initialized to 1900 on most systems but on Alpha and MIPS it might also be initialized to 1952, 1980, or 2000, depending on the value of an RTC register for the year. With some RTCs, these operations can be used to read or to set the RTC's Epoch, respectively. The third ioctl(2) argu? ment is an unsigned long * or an unsigned long, respectively, and the value returned (or assigned) is the Epoch. To set the RTC's Epoch the process must be privileged (i.e., have the CAP_SYS_TIME capability).

RTC_WKALM_RD, RTC_WKALM_SET

Some RTCs support a more powerful alarm interface, using these ioctls to read or write the RTC's alarm time (respectively) with this structure:

```
struct rtc_wkalrm {
    unsigned char enabled;
    unsigned char pending;
    struct rtc_time time;
};
```

The enabled flag is used to enable or disable the alarm inter? rupt, or to read its current status; when using these calls, RTC_AIE_ON and RTC_AIE_OFF are not used. The pending flag is used by RTC_WKALM_RD to report a pending interrupt (so it's mostly useless on Linux, except when talking to the RTC managed by EFI firmware). The time field is as used with RTC_ALM_READ and RTC_ALM_SET except that the tm_mday, tm_mon, and tm_year fields are also valid. A pointer to this structure should be passed as the third ioctl(2) argument.

FILES

/dev/rtc, /dev/rtc0, /dev/rtc1, etc.

RTC special character device files.

/proc/driver/rtc

status of the (first) RTC.

NOTES

When the kernel's system time is synchronized with an external reference using adjtimex(2) it will update a designated RTC periodically every 11 minutes. To do so, the kernel has to briefly turn off periodic interrupts; this might affect programs using that RTC.

An RTC's Epoch has nothing to do with the POSIX Epoch which is used only for the system clock.

If the year according to the RTC's Epoch and the year register is less than 1970 it is assumed to be 100 years later, that is, between 2000 and 2069.

Some RTCs support "wildcard" values in alarm fields, to support scenarios like periodic alarms at fifteen minutes after every hour, or on the first day of each month. Such usage is nonportable; portable user-space code expects only a single alarm interrupt, and will either disable or reinitialize the alarm after receiving it.

Some RTCs support periodic interrupts with periods that are multiples of a second rather than fractions of a second; multiple alarms; programmable output clock signals; nonvolatile memory; and other hardware capabilities that are not currently exposed by this API.

SEE ALSO

date(1), adjtimex(2), gettimeofday(2), settimeofday(2), stime(2), time(2), gmtime(3), time(7), hwclock(8)

Documentation/rtc.txt in the Linux kernel source tree

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/.