The Linux Watchdog driver API

Last reviewed: 10/05/2007

This document describes the state of the Linux 2.4.18 kernel.

Introduction

A Watchdog Timer (WDT) is a hardware circuit that can reset the computer system in case of a software fault. You probably knew that already.

Usually a userspace daemon will notify the kernel watchdog driver via the /dev/watchdog special device file that userspace is still alive, at regular intervals. When such a notification occurs, the driver will usually tell the hardware watchdog that everything is in order, and that the watchdog should wait for yet another little while to reset the system. If userspace fails (RAM error, kernel bug, whatever), the notifications cease to occur, and the hardware watchdog will reset the system (causing a reboot) after the timeout occurs.

The Linux watchdog API is a rather ad-hoc construction and different drivers implement different, and sometimes incompatible, parts of it. This file is an attempt to document the existing usage and allow future driver writers to use it as a reference.

The simplest API

All drivers support the basic mode of operation, where the watchdog activates as soon as /dev/watchdog is opened and will reboot unless the watchdog is pinged within a certain time, this time is called the timeout or margin. The simplest way to ping the watchdog is to write some data to the device. So a very simple watchdog daemon would look like this source file: see samples/watchdog/watchdog-simple.c

A more advanced driver could for example check that a HTTP server is still responding before doing the write call to ping the watchdog.

When the device is closed, the watchdog is disabled, unless the "Magic Close" feature is supported (see below). This is not always such a good idea, since if there is a bug in the watchdog daemon and it crashes the system will not reboot. Because of this, some of the drivers support the configuration option "Disable watchdog shutdown on close", CONFIG_WATCHDOG_NOWAYOUT. If it is set to Y when compiling the kernel, there is no way of disabling the watchdog once it has been started. So, if the watchdog daemon crashes, the system will reboot after the timeout has passed. Watchdog devices also usually support the nowayout module parameter so that this option can be controlled at runtime.

Magic Close feature

If a driver supports "Magic Close", the driver will not disable the watchdog unless a specific magic character 'V' has been sent to /dev/watchdog just before closing the file. If the userspace daemon closes the file without sending this special character, the driver will assume that the daemon (and userspace in general) died, and will stop pinging the watchdog without disabling it first. This will then cause a reboot if the watchdog is not re-opened in sufficient time.

The ioctl API

All conforming drivers also support an ioctl API.

Pinging the watchdog using an ioctl:

All drivers that have an ioctl interface support at least one ioctl, KEEPALIVE. This ioctl does exactly the same thing as a write to the watchdog device, so the main loop in the above program could be replaced with:

```
while (1) {
        ioctl(fd, WDIOC_KEEPALIVE, 0);
        sleep(10);
}
```

the argument to the ioctl is ignored.

Setting and getting the timeout

For some drivers it is possible to modify the watchdog timeout on the fly with the SETTIMEOUT ioctl, those drivers have the WDIOF_SETTIMEOUT flag set in their option field. The argument is an integer representing the timeout in seconds. The driver

returns the real timeout used in the same variable, and this timeout might differ from the requested one due to limitation of the hardware:

```
int timeout = 45;
ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
printf("The timeout was set to %d seconds\n", timeout);
```

This example might actually print "The timeout was set to 60 seconds" if the device has a granularity of minutes for its timeout.

Starting with the Linux 2.4.18 kernel, it is possible to query the current timeout using the GETTIMEOUT ioctl:

```
ioctl(fd, WDIOC_GETTIMEOUT, &timeout);
printf("The timeout was is %d seconds\n", timeout);
```

Pretimeouts

Some watchdog timers can be set to have a trigger go off before the actual time they will reset the system. This can be done with an NMI, interrupt, or other mechanism. This allows Linux to record useful information (like panic information and kernel coredumps) before it resets:

```
pretimeout = 10;
ioctl(fd, WDIOC_SETPRETIMEOUT, &pretimeout);
```

Note that the pretimeout is the number of seconds before the time when the timeout will go off. It is not the number of seconds until the pretimeout. So, for instance, if you set the timeout to 60 seconds and the pretimeout to 10 seconds, the pretimeout will go off in 50 seconds. Setting a pretimeout to zero disables it.

There is also a get function for getting the pretimeout:

```
ioctl(fd, WDIOC_GETPRETIMEOUT, &timeout);
printf("The pretimeout was is %d seconds\n", timeout);
```

Not all watchdog drivers will support a pretimeout.

Get the number of seconds before reboot

Some watchdog drivers have the ability to report the remaining time before the system will reboot. The WDIOC_GETTIMELEFT is the ioctl that returns the number of seconds before reboot:

```
ioctl(fd, WDIOC_GETTIMELEFT, &timeleft);
printf("The timeout was is %d seconds\n", timeleft);
```

Environmental monitoring

All watchdog drivers are required return more information about the system, some do temperature, fan and power level monitoring, some can tell you the reason for the last reboot of the system. The GETSUPPORT ioctl is available to ask what the device can do:

```
struct watchdog_info ident;
ioctl(fd, WDIOC_GETSUPPORT, &ident);
```

the fields returned in the ident struct are:

identity	a string identifying the watchdog driver
firmware_version	the firmware version of the card if available
options	a flags describing what the device supports

the options field can have the following bits set, and describes what kind of information that the GET_STATUS and GET_BOOT_STATUS ioctls can return.

WDIOF_OVERHEAT	Reset due to CPU overheat

The machine was last rebooted by the watchdog because the thermal limit was exceeded:

WDIOF_FANFAULT	Fan failed

A system fan monitored by the watchdog card has failed

WDIOF_EXTERN1	External relay 1

External monitoring relay/source 1 was triggered. Controllers intended for real world applications include external monitoring pins that will trigger a reset.

WDIOF_EXTERN2	External relay 2

External monitoring relay/source 2 was triggered

WDIOF_POWERUNDER	Power bad/power fault

The machine is showing an undervoltage status

WDIOF_CARDRESET	Card previously reset the CPU

The last reboot was caused by the watchdog card

WDIOF_POWEROVER	Power over voltage

The machine is showing an overvoltage status. Note that if one level is under and one over both bits will be set - this may seem odd but makes sense.

WDIOF_KEEPALIVEPING	Keep alive ping reply

The watchdog saw a keepalive ping since it was last queried.

WDIOF_SETTIMEOUT	Can set/get the timeout

The watchdog can do pretimeouts.

WDIOF_PRETIMEOUT	Pretimeout (in seconds), get/set

For those drivers that return any bits set in the option field, the GETSTATUS and GETBOOTSTATUS ioctls can be used to ask for the current status, and the status at the last reboot, respectively:

```
int flags;
ioctl(fd, WDIOC_GETSTATUS, &flags);

or

ioctl(fd, WDIOC_GETBOOTSTATUS, &flags);
```

Note that not all devices support these two calls, and some only support the GETBOOTSTATUS call.

Some drivers can measure the temperature using the GETTEMP ioctl. The returned value is the temperature in degrees fahrenheit:

```
int temperature;
ioctl(fd, WDIOC_GETTEMP, &temperature);
```

Finally the SETOPTIONS ioctl can be used to control some aspects of the cards operation:

```
int options = 0;
ioctl(fd, WDIOC_SETOPTIONS, &options);
```

The following options are available:

WDIOS_DISABLECARD	Turn off the watchdog timer
WDIOS_ENABLECARD	Turn on the watchdog timer
WDIOS_TEMPPANIC	Kernel panic on temperature trip

[FIXME -- better explanations]