watchdog(8) - Linux man page

Name

watchdog - a software watchdog daemon

Synopsis

watchdog [**-f**|**--force**] [**-c** *filename*|**--config-file** *filename*] [**-v**|**--verbose**] [**-s**|**--sync**] [**-b**|**--softboot**] [**-q**|**--no-action**]

Description

The Linux kernel can reset the system if serious problems are detected. This can be implemented via special watchdog hardware, or via a slightly less reliable software-only watchdog inside the kernel. Either way, there needs to be a daemon that tells the kernel the system is working fine. If the daemon stops doing that, the system is reset.

watchdog is such a daemon. It opens */dev/watchdog*, and keeps writing to it often enough to keep the kernel from resetting, at least once per minute. Each write delays the reboot time another minute. After a minute of inactivity the watchdog hardware will cause the reset. In the case of the software watchdog the ability to reboot will depend on the state of the machines and interrupts.

The watchdog daemon can be stopped without causing a reboot if the device */dev/watchdog* is closed correctly, unless your kernel is compiled with the *CONFIG_WATCHDOG_NOWAYOUT* option enabled.

Tests

The watchdog daemon does several tests to check the system status:

• Is the process table full?

• Is there enough free memory?

• Are some files accessible?

• Have some files changed within a given interval?

• Is the average work load too high?

• Has a file table overflow occurred?

• Is a process still running? The process is specified by a pid file.

• Do some IP addresses answer to ping?

• Do network interfaces receive traffic?

• Is the temperature too high? (Temperature data not always available.)

• Execute a user defined command to do arbitrary tests.

• Execute one or more test/repair commands found in /etc/watchdog.d. These commands are called with the argument **test** or **repair**.

If any of these checks fail watchdog will cause a shutdown. Should any of these tests except the user defined binary last longer than one minute the machine will be rebooted, too.

Options

Available command line options are the following:

-v, **--verbose**
: Set verbose mode. Only implemented if compiled with *SYSLOG* feature. This mode will log each several infos in *LOG_DAEMON* with priority *LOG_INFO.* This is useful if you want to see exactly what happened until the watchdog rebooted the system. Currently it logs the temperature (if available), the load average, the change date of the files it checks and how often it went to sleep.

-s, **--sync**
: Try to synchronize the filesystem every time the process is awake. Note that the system is rebooted if for any reason the synchronizing lasts longer than a minute.

-b, **--softboot**
: Soft-boot the system if an error occurs during the main loop, e.g. if a given file is not accessible via the **stat**(2) call. Note that this does not apply to the opening of */dev/watchdog* and */proc/loadavg*, which are opened before the main loop starts.

-f, **--force**
: Force the usage of the interval given or the maximal load average given in the config file.

-c *config-file*, **--config-file** *config-file*
: Use *config-file* as the configuration file instead of the default */etc/watchdog.conf*.

-q, **--no-action**
: Do not reboot or halt the machine. This is for testing purposes. All checks are executed and the results are logged as usual, but no action is taken. Also your hardware card or the kernel software watchdog driver is not enabled. Temperature checking is also disabled since this triggers the hardware watchdog on some cards.

Function

After **watchdog** starts, it puts itself into the background and then tries all checks specified in its configuration file in turn. Between each two tests it will write to the kernel device to prevent a reset. After finishing all tests watchdog goes to sleep for some time. The kernel drivers expects a write to the watchdog device every minute. Otherwise the system will be reset. As a default **watchdog** will sleep for only 10 seconds so it triggers the device early enough.

Under high system load **watchdog** might be swapped out of memory and may fail to make it back in in time. Under these circumstances the Linux kernel will reset the machine. To make sure you won't get unnecessary reboots make sure you have the variable *realtime* set to *yes* in the configuration file *watchdog.conf*. This adds real time support to **watchdog**: it will lock itself into memory and there should be no problem even under the highest of loads.

Also you can specify a maximal allowed load average. Once this load average is reached the system is rebooted. You may specify maximal load averages for 1 minute, 5 minutes or 15 minutes. The default values is to disable this test. Be careful not to set this parameter too low. To set a value less then the predefined minimal value of 2, you have to use the **-f** option.

You can also specify a minimal amount of virtual memory you want to have available as free. As soon as more virtual memory is used action is taken by **watchdog**. Note, however, that watchdog does not distinguish between different types of memory usage. It just checks for free virtual memory.

If you have a watchdog card with temperature sensor you can specify the maximal allowed temperature. Once this temperature is reached the system is halted. The default value is 120. There is no unit conversion so make sure you use the same unit as your hardware. **watchdog** will issue warnings once the temperature increases 90%, 95% and 98% of this temperature.

When using file mode **watchdog** will try to **stat**(2) the given files. Errors returned by stat will **not** cause a reboot. For a reboot the stat call has to last at least one minute. This may happen if the file is located on an NFS mounted filesystem. If your system relies on an NFS mounted filesystem you might try this option. However, in such a case the *sync* option may not work if the NFS server is not answering.

watchdog can read the pid from a pid file and see whether the process still exists. If not, action is taken by **watchdog**. So you can for instance restart the server from your *repair-binary*.

watchdog will try periodically to fork itself to see whether the process table is full. This process will leave a zombie process until watchdog wakes up again and catches it; this is harmless, don't worry about it.

In ping mode **watchdog** tries to ping the given IP addresses. These addresses do not have to be a single machine. It is possible to ping to a broadcast address instead to see if at least one machine in a subnet is still living.

Do not use this broadcast ping unless your MIS person a) knows about it and b) has given you explicit permission to use it!

watchdog will send out three ping packages and wait up to <interval> seconds for the reply with <interval> being the time it goes to sleep between two times triggering the watchdog device. Thus a unreachable network will not cause a hard reset but a soft reboot.

You can also test passively for an unreachable network by just monitoring a given interface for traffic. If no traffic arrives the network is considered unreachable causing a soft reboot or action from the repair binary.

watchdog can run an external command for user-defined tests. A return code not equal 0 means an error occured and watchdog should react. If the external command is killed by an uncaught signal this is considered an error by watchdog too. The command may take longer than the time slice defined for the kernel device without a problem. However, error messages are generated into the syslog facility. If you have enabled softboot on error the machine will be rebooted if the binary doesn't exit in half the time **watchdog** sleeps between two tries triggering the kernel device.

If you specify a repair binary it will be started instead of shutting down the system. If this binary is not able to fix the problem **watchdog** will still cause a reboot afterwards.

If the machine is halted an email is sent to notify a human that the machine is going down. Starting with version 4.4 **watchdog** will also notify the human in charge if the machine is rebooted.

Soft Reboot

A soft reboot (i.e. controlled shutdown and reboot) is initiated for every error that is found. Since there might be no more processes available, watchdog does it all by himself. That means:

1.

Kill all processes with SIGTERM.

2.

After a short pause kill all remaining processes with SIGKILL.

3.

Record a shutdown entry in wtmp.

4.

Save the random seed from */dev/urandom*. If the device is non-existant or there is no filename for saving this step is skipped.

5.

Turn off accounting.

6.

Turn off quota and swap.

7.

Unmount all partitions except the root partition.

8.

Remount the root partition read-only.

9.

Shut down all network interfaces.

10.

Finally reboot.

Check Binary

If the return code of the check binary is not zero **watchdog** will assume an error and reboot the system. Be careful with this if you are using the real-time properties of watchdog since **watchdog** will wait for the return of this binary before proceeding. An positive exit code is interpreted as an system error code (see *errno.h* for details). Negative values are special to **watchdog**:

-1

Reboot the system. This is not exactly an error message but a command to **watchdog**. If the return code is -1 **watchdog** will not try to run a shutdown script instead.

-2

Reset the system. This is not exactly an error message but a command to **watchdog**. If the return code is -2 **watchdog will simply refuse to write the** kernel device again.

-3

Maximum load average exceeded.

-4

The temperature inside is too high.

-5

/proc/loadavg contains no (or not enough) data.

-6

The given file was not changed in the given interval.

-7

/proc/meminfo contains invalid data.

-8

Child process was killed by a signal.

-9

Child process did not return in time.

-10

Free for personal use.

Repair Binary

The repair binary is started with one parameter: the error number that caused **watchdog** to initiate the boot process. After trying to repair the system the binary should exit with 0 if the system was successfully repaired and thus there is no need to boot anymore. A return value not equal 0 tells **watchdog** to reboot. The return code of the repair binary should be the error number of the error causing **watchdog** to reboot. Be careful with this if you are using the real-time properties since **watchdog** will wait for the return of this binary before proceeding.

Test Directory

Executables placed in the test directory are discovered by watchdog on startup and are automatically executed. They are bounded time-wise by the test-timeout directive in watchdog.conf.

These executables are called with either "test" as the first argument (if a test is being performed) or "repair" as the first argument (if a repair for a previously-failed "test" operation on is being performed).

The as with test binaries and repair binaries, expected exit codes for a successful test or repair operation is always zero.

If an executable's test operation fails, the same executable is automatically called with the "repair" argument as well as the return code of the previously-failed test operation.

For example, if the following execution returns 42:

/etc/watchdog.d/my-test test

The watchdog daemon will attempt to repair the problem by calling:

/etc/watchdog.d/my-test repair 42

This enables administrators and application developers to make intelligent test/repair commands. If the "repair" operation is not required (or is not likely to succeed), it is important that the author of the command return a non-zero value so the machine will still reboot as expected.

Note that the watchdog daemon may interpret and act upon any of the reserved return codes noted in the Check Binary section prior to calling a given command in "repair" mode.

Bugs

None known so far.

Authors

The original code is an example written by Alan Cox <alan@lxorguk.ukuu.org.uk>, the author of the kernel driver. All additions were written by Michael Meskes <meskes@debian.org>. Johnie Ingram <johnie@netgod.net> had the idea of testing the load average. He also took over the Debian specific work. Dave Cinege <dcinege@psychosis.com> brought up some hardware watchdog issues and helped testing this stuff.

Files

/dev/watchdog

The watchdog device.

/var/run/watchdog.pid

The pid file of the running **watchdog**.

See Also

watchdog.conf(5)

Referenced By

checkquorum(8), **wd_keepalive**(8), **wdmd**(8)