**Linux Watchdog Daemon – Configuring**

[Back to PSC's home page](https://www.crawford-space.co.uk/old_psc/index.html)  
[Back to Watchdog](https://www.crawford-space.co.uk/old_psc/watchdog/Linux-Watchdog.html)

From: <https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html>

There are a number of tests and options that can be configured for the watchdog daemon, and this page is still "work in progress" to describe them. Typically the source of configuration data is the corresponding 'man' page, such as <http://linux.die.net/man/5/watchdog.conf>  
  
This page is intended to detail the configuration options normally set in /etc/watchdog.conf and should be considered after reading the [general tests](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-general-tests.html) page that provides an overview of what the daemon can do.

**Table of Variables**

In the following table a "string" is some text, and "string/R" means you can have repeated lines for multiple cases of the configured parameter, for example:  
  
ping = 192.168.1.1  
ping = 192.168.1.100  
  
The type "yes/no" implies a Boolean true/false choice that is configured by "yes" for true, and "no" for false.  
  
An "object" is any testable thing (file age, daemon PID, ping target, etc) that has a state associated with it. This is basically everything except some internal watchdog actions.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Function / Description** |
| [admin](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Administrative_Settings) | string | This is the email user name of the person to be notified when the system is rebooting, the default is "root". Assumes the sendmail program is installed and configured correctly. |
| [allocatable-memory](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Memory_Test) | integer | This is similar to the older min-memory configuration, but actively tests for a given number of allocatable memory pages (typically 4kB/page on x86 hardware). Zero to disable test. |
| [change](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#File_Monitoring) | integer | Time limit (in seconds) for a specified file time-stamp to age. Must come after the corresponding 'file' entry. |
| [file](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#File_Monitoring) | string/R | The path/name of a file to be checked for existence and, (if 'change' given) for age. |
| [heartbeat-file](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Heartbeat_File) | string | Name of the file for diagnostic heartbeat writes a time\_t value (in ASCII) on each write to the watchdog device. |
| [heartbeat-stamps](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Heartbeat_File) | integer | Number of entries in debug heartbeat file. |
| [interface](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Network_Interface) | string/R | Name of interface (such as eth0) in /proc/net/dev to check for incoming (RX) bytes. |
| [interval](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Watchdog_Device__Time) | integer | Time interval (seconds) between polling for system health. Default is 1, but should not be more than [watchdog timeout]-2 seconds. |
| [log-dir](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Administrative_Settings) | string | Path for watchdog log directory where the heartbeat file is usually kept, and where the files for re-directing test/repair scripts are kept. Default is /var/log/watchdog |
| [logtick](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Verbosity) | integer | Number of polling intervals between periodic "verbose" status messages. Default is 1 (i.e. every poll event). |
| [max-load-1](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Load_Averages) | integer | Limit on the 1-minute load-average before a reboot is triggered. Set to zero to ignore this test. |
| [max-load-5](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Load_Averages) | integer | Limit on the 5-minute load-average before a reboot is triggered. Set to zero to ignore this test. |
| [max-load-15](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Load_Averages) | integer | Limit on the 15-minute load-average before a reboot is triggered. Set to zero to ignore this test. |
| [max-temperature](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Temperature_Sensors) | integer | Limit on temperature before shut-down, Celsius. |
| [min-memory](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Memory_Test) | integer | Minimum number of memory pages (typically 4kB/page on x86 hardware). Zero to disable test. |
| [pidfile](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Process_Monitoring_by_PID_File) | string/R | Path/name of a PID file related to a daemon to be monitored. |
| [ping](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Network_ping) | string/R | The IP address of a target for ICMP "ping" test. Must be in numeric IPv4 format such as 192.168.1.1 |
| [ping-count](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Network_ping) | integer | Number of ping attempts per polling interval. Must be >= 1 and default is 3 (hence with 1 second polling interval ping delay must be less than 333ms). |
| [priority](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Administrative_Settings) | integer | The scheduling priority used with a call to the [sched\_setscheduler()](http://linux.die.net/man/2/sched_setscheduler) function to configure the round-robin (SCHED\_RR) priority for real-time use (only applicable if 'realtime' is true). |
| [realtime](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Administrative_Settings) | yes/no | This flag is used to tell the watchdog daemon to lock its memory against paging out, and also to the permit real-time scheduling. It is **strongly recommended** to do this! |
| [repair-binary](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_0_Test__Repair) | string | The path/name of a program (or bash script, etc) that is used to make a repair on failed tests (other than auto-loaded V1 test scripts). |
| [repair-maximum](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Repair__Retry_Time-Out) | integer | Number of repair attempts on one "object" without success before giving up and rebooting. Default is 1, and setting this to zero will allow any number of repair attempts. |
| [repair-timeout](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_0_Test__Repair) | integer | Time limit (seconds) for the repair action. Default is 60 and beyond this a reboot is initiated. |
| [retry-timeout](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Repair__Retry_Time-Out) | integer | Time limit (seconds) from the first failure on a given "object" until it is deemed bad and a repair attempted (if possible, otherwise a reboot is the action). Default is 60 seconds. |
| [sigterm-delay](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Watchdog_Device__Time) | integer | Time between the SIGTERM signal being sent to all processes and the following SIGKILL signal. Default is 5 seconds, range 2-300. |
| [temperature-device](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Temperature_Sensors) | string | (depreciated) This was used in V5.13 and below for the old /dev/temperature style of device. With V5.15 the use of temperature-sensor is used and old style no longer supported. |
| [temperature-poweroff](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Temperature_Sensors) | yes/no | This flag decides if the system should power-off on overheating (default = yes), or perform a system halt and wait for Ctrl-Alt-Del reactivation (the "no" case). |
| [temperature-sensor](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Temperature_Sensors) | string/R | Name of the file-like device that holds temperature as an ASCII string in milli-Celsius, typically generated by the lm-sensors package. |
| [test-binary](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_0_Test__Repair) | string/R | The path/name of a V0 test program (or bash script, etc) used to extend the watchdog's range of health tests. NOTE: The V0 test binary should be considered as 'depreciated' and used for reverse compatibility only, and the the [V1 test/repair script](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_1_TestRepair) mode of operation used when ever possible. |
| [test-directory](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_1_TestRepair) | string | The path name of the directory for auto-loaded V1 test/repair scripts. Default is: test-directory=/etc/watchdog.d This ability can be disabled completely by setting it to no string: test-directory= If the directory is not present it is ignored in any case. |
| [test-timeout](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_0_Test__Repair) | integer | Time limit (seconds) for any test scripts. Default is 60. This can be set to zero to disable the time-out, however, in this case a hung program will never be actioned, though all other tests will continue normally. |
| [verbose](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Verbosity) | yes/no | Provides basic control of the verbosity of the status messages. Previously this was only possible on the -v / --verbose [command line options](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-commandline.html). |
| [watchdog-device](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Watchdog_Device__Time) | string | The name of the device for the watchdog hardware. Default is /dev/watchdog If this is not given (or disabled by setting it to no string) the watchdog can still function, but will not be effective as any internal watchdog faults or kernel panic will be unrecoverable. |
| [watchdog-timeout](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Watchdog_Device__Time) | integer | The timeout to set the watchdog device to. Default is 60 seconds and it is not recommended to change this without good reason. Not all watchdog hardware supports configuration, or configuration to second resolution, etc. |

[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Watchdog Device & Time**

While it is possible for the watchdog daemon to function as a stand-alone system monitor making use of the numerous checks described here, it reality it is not very effective without the actual "watchdog device". Normally this device consists of a hardware timer that, upon time-out, will reset the computer in the same manner as the hardware reset switch, and a matching device driver module that provides a uniform interface to all of the supported hardware.  
  
One option to identify the watchdog hardware, if your motherboard maker has not listed it, is to install the [lm-sensors package](http://www.lm-sensors.org/) for temperature, voltage, etc, monitoring. On a typical Ubuntu machine you can install this with:  
  
    apt-get install lm-sensors  
  
Once installed, run the 'sensors-detect' script to find out what hardware you have, as often there is a watchdog timer built in to the chip. By default, the watchdog modules are black-listed because some of them start automatically (hence the machine would spontaneously reboot if the watchdog daemon was not running correctly). This list, at least for Ubuntu 12.04, is given in /etc/modprobe.d/blacklist-watchdog.conf  Some professional style of board support IPMI and the driver for that also needs to be specially loaded, see, for example, this [Ubuntu IPMI example](https://github.com/miniwark/miniwark-howtos/wiki/Hardware-Watchdog-Timer-setup-on-Ubuntu-12.04).  
  
If all else fails, and you have no hardware support, you can load the 'softdog' module to emulate some of the capabilities in software. However, this will provide **greatly reduced protection** as there is nothing to recover from a kernel panic, or a bad peripheral driver that blocks a software reboot.  
  
Typically you edit the /etc/modules file and add the appropriate driver's name. When installed (reboot after editing /etc/modules, or via 'modprobe' call) the watchdog driver normally presents the file-like device /dev/watchdog, the details of which can be found in the [API documentation](http://www.mjmwired.net/kernel/Documentation/watchdog/watchdog-api.txt).  
  
The watchdog daemon has 4 settings related to the watchdog device, these are:  
  
watchdog-device = /dev/watchdog  
watchdog-timeout = 60  
interval = 1  
sigterm-delay = 5  
  
The first two define the device API point and the time-out to be configured. However, you need to be aware that not all hardware has 1s resolution, and not all hardware (or drivers) are capable of configuration to arbitrary values. In general, do not change the default 60 second timer value unless you have a very good reason.  
  
The 3rd in the list is the polling interval, which by default is 1 second. While not a property of the watchdog device, it is clearly related in that it must be less than the time-out of the hardware, and realistically it must be at least 2 seconds less than this because:

* The timer hardware is not necessarily synchronised to the polling period, and;
* The health checks could take a small but significant fraction of a second to run, and the "interval=" value is simply the sleep time between loops.

The choice of poll interval has several trade-off conditions to consider:

* Long poll intervals have a small power-use advantage (even though the health checks are not too demanding in CPU time).
* Long poll intervals can be helpful for network tests as they provide more time for 'ping' or data transfer to be detected.
* Short poll intervals can be helpful to detect problems earlier.
* Short poll intervals reduce the chance of monitored process' PIDs being reused should any fail.

As an approximate guide, the poll interval should not be longer than about 1/3 to 1/2 of the hardware time, or about 1/3 of the retry time (whichever is shorter). However, in most situations poll intervals below 5 seconds offer little benefit in terms of rapid recovery as reboot times are usually much longer.  
  
NOTE: With watchdog now running the test binaries asynchronously to the main polling loop, so you can have those run less frequently simply by adding a [sleep](http://linux.die.net/man/1/sleep) call to the script or program (of course, providing it is less than the test-timeout value).  
  
The 4th value controls the timing of the moderately orderly shutdown process. It is the delay between the SIGTERM signal being sent to 'politely request' all processes should terminate, and the following non-ignorable SIGKILL signal. The default is 5 seconds, but this can be increased up to 300 (5 minutes) to allow any slow exiting processes (e.g. virtual machine, database, etc) a chance to clean up before they are stopped for sure.  
  
NOTE: If a hard reset is requested, or if the machine is seriously broken and the watchdog hardware kicks in, then it will result in a brutal stop to all process. It is therefore preferable that applications are designed to recover their databases automatically from any sort of termination. Unfortunately that is not always the case. Thus for a well designed and robust system, additional work may be needed to allow a regular 'snapshot' of consistent database(s) to be made so a clean resumption for each application is possible.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Verbosity**

By default the watchdog daemon only prints out start and stop messages to syslog, and also if something has gone wrong. For normal use this is sufficient as then a simple search for 'watchdog' will bring out normal system start/stop events along with any error events, for example:  
  
    grep -h 'watchdog' /var/log/syslog.1 /var/log/syslog  
  
However, for setting up the watchdog or debugging changes to the code or test/repair scripts it can be useful to get more information about the configuration and the tests being performed. There are two options to configure these messages:  
  
verbose = no  
logtick = 1  
  
The "verbose=" option is configured in the file as a simple yes/no choice, however, it actually has 3 levels and the higher value can be achieved by using the [command line option](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-commandline.html) -v / --verbose twice, which is more common for testing. The option to configure verbosity in the file should be considered only for unusual cases as you can generate a lot of syslog traffic, and that can make it harder to see real problems.  
  
The "logtick=" option allows you to reduce the volume of periodic messages in syslog by only reporting 1 out of N times, though the default is to report all.  
  
Irrespective of the verbosity settings, all errors are logged. However, with a serious system failure they may not be committed to disk for subsequent analysis. You should also consider syslog forwarding to a central computer for log file storage an analysis.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Administrative Settings**

The watchdog has a number of system-specific settings that occasionally the administrator may which to change. This sub-section covers them. The fist of these is the user name for any messages to be emailed upon a system shutdown. If the sendmail program is installed (and configured, of course!) then an email will be sent to the administrator using this email user name:  
  
admin = root  
  
If this is set to a null string (e.g. "admin=" in the file in place of "admin=root") then no email will be attempted.  
  
The watchdog daemon uses a "log directory" for holding files that are used to store the redirected stdout and stderr of any [test or repair programs](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#TestRepair_Scripts). This can be changed with the parameter:  
  
log-dir = /var/log/watchdog  
  
Finally, and very importantly, the watchdog daemon normally tries to lock itself in to non-paged memory and set its priority to real-time so it continues to function as far as possible even if the machine load is exceptionally high. The following parameters can be used to change this, but you are ***strongly advised*** to leave these at their default settings:  
  
priority = 1  
realtime = yes  
  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Temperature Sensors**

NOTE: The older versions (V5.13 and below) assumed a single device that provided a binary character for the temperature in arbitrary units (some Celsius, some Fahrenheit, etc), typically as:  
  
temperature-device = /dev/temperature  
  
This is no longer supported and the keyword in the configuration file was changed to temperature-sensor to avoid compatibility issues when going back from V5.15+ to V5.13 or similar. To use this, read on...  
  
Before attempting to configure for temperature, make sure you have installed the lm-sensors package and run the sensors-detect script. That should help identify the hardware and offer to add it to your /etc/modules file so it is there on reboot as well. It is also worth looking to see if there is any motherboard-specific configuration to help with the scaling and presentation of the data:  
<http://www.lm-sensors.org/wiki/Configurations>  
Once running, the package presents the results in virtual files under /sys most commonly as something under /sys/class/hwmon but finding the simple path is not easy as they often contain symbolic link loops (bad!). To find the original hardware entries, this command can be used:  
  
find /sys -name 'temp\*input' -print  
  
You should get an answer something like:  
  
/sys/devices/platform/coretemp.0/temp2\_input  
/sys/devices/platform/coretemp.0/temp3\_input  
/sys/devices/platform/coretemp.0/temp4\_input  
/sys/devices/platform/coretemp.0/temp5\_input  
/sys/devices/platform/w83627ehf.2576/temp1\_input  
/sys/devices/platform/w83627ehf.2576/temp2\_input  
/sys/devices/platform/w83627ehf.2576/temp3\_input  
  
In this example from the tracking PC the first 4 are the CPU internal core temperature sensors, and the final 3 are the hardware monitors (the w83627ehf module provides the hardware monitoring, and the matching w83627hf\_wdt module the watchdog timer). With V5.15 of the watchdog you can have multiple temperature devices, for example, in the watchdog.conf file:  
  
temperature-sensor = /sys/class/hwmon/hwmon0/device/temp1\_input  
temperature-sensor = /sys/class/hwmon/hwmon0/device/temp2\_input  
  
And so on for all temperature sensors you wish to use.

**Warning:** There is currently a bug/feature where by the order of loading the temperature sensor modules determines the abstracted names (e.g. the first module loaded becomes /sys/class/hwmon/hwmon0 and the second /sys/class/hwmon/hwmon1 etc.)  
  
If using the abstracted paths (e.g. /sys/class/hwmon/hwmon0) rather then the device paths (e.g. /sys/devices/platform/w83627ehf.2576) then make sure you black-list any modules that are automatically loaded by adding a suitable entry to one of the files in /etc/modprobe.d/ and then add all modules for temperature sensing to /etc/modules as that appears to force deterministic enumeration.

Since the new lm-sensors style of monitoring provides files in milli-Celsius the watchdog now always works in Celsius, and the maximum temperature is set using the configuration option, for example:  
  
max-temperature = 120  
  
The daemon generates warnings as it crosses 90%, 95% and 98% of the threshold, and will also provide a syslog message if it drops back below 90% once more. If the maximum temperature is exceeded then it initiates a power-off shut-down. You can configure this to halt the system instead (where it is theoretically reboot-able using Ctrl+Alt+Del) changing this configuration option from 'yes' to 'no'.  
  
temperature-poweroff = yes  
  
Note: An over temperature condition is one of those consider non-repairable in V5.15, so shut-down will happen no matter what the repair binary might have tried.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Load Averages**

The watchdog can monitor the 3 load average figures that are indicative of the machine's task queue depth, etc. These are averaged by a simple filter with time constants of 1, 5 and 15 minutes and are read from the virtual file /proc/loadavg  
   
Before using this option, it is important to have a good idea of what they mean to the machine:  
  
[https://en.wikipedia.org/wiki/Load\_(computing)](https://en.wikipedia.org/wiki/Load_%28computing%29)

<http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages>

In a simple form, a load average above 1 per CPUs indicates tasks are being held up due to a lack of resources, either CPU time or I/O delays. This is not a problem if it is only happening at peak times of the day and/or if it is only by a modest amount (say 1-2 times the number of CPUs).  
  
When things go really wrong, for example lots of I/O waiting on a downed network file system, or a fork bomb is filling the machine with useless resource-sucking processes (either malicious or just a badly designed/implemented program), then the averages normally go well above 5 times the number of CPUs (e.g. on our 4-core single CPU tracking PCs that would be above 5\*4 = 20).  
  
Unless you are pretty sure what range of averages your system normally encounters, keep to the high side!  
  
For example, we have seen on 8-core box with a failed 10Gig network connection average 120 for several hours and it was almost impossible to SSH in to, in this case 15 per CPU core was an indication of failure. Hence a threshold of something like 10 per core would be reasonably safe. You might also want to configure a slightly lower threshold for the 15-minute average, say around 5-7 per core, to deal with persistent problems.  
  
The thresholds are set in the configuration file using options of the form:  
  
max-load-1 = 40  
max-load-5 = 30  
max-load-15 = 20  
  
Note: The older 5.13 version of the watchdog daemon would compute 5 and 15 minute thresholds from the 1 minute threshold if nothing was configured (using 75% and 50% respectively), however 6.0 only tests those thresholds you explicitly set. For example, if you comment out the entry in the configuration file "max-load-15=20" with V5.13 it is still tested based on 50% of max-load-1, but with V5.15 it is not tested at all.  
  
**Caution!** In some cases you can enter a reboot loop due to pending requests. For example, in a clustering/fail-over situation the act of rebooting on high load might simply transfer the load to another machine, potentially triggering a cascade of reboots. Or if a web server ends up with a lot of clients waiting for it during an outage, and the built-up requests all resume immediately when the machine becomes live again, so the load averages increase and it reboots, and the clients are still waiting...  
  
In such cases you might want to set the 1 and 5 minute thresholds on the high side (say 10-20 times the number of CPU cores, maybe more) and rely on a more conventional threshold of around 5 times the number of cores for the 15 minute average. Ultimately you really should be quite sure of what an acceptable heavy load is, and what an exceptional one is, and be sure that a system reboot is the best way to deal with it. For averages of 5 or more per CPU core then it is probably the best option as the machine will generally be fairly unresponsive.  
  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Network Monitoring**

The daemon can monitor the machine's network connection(s) is several ways, the most obvious being:

* Passively by looking at received data on a [network interface](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Network_Interface) using the "interface =" option.
* Actively by 'pinging' with an [ICMP packet](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Network_ping) to an external target using the "ping =" option.
* Indirectly by [monitoring a file](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#File_Monitoring) on a network file system using the "file =" option.

These methods all tell you if an interface is usable, but do not tell you if a reboot will help. For example, if someone reboots your network switch then all 3 methods will tell you the system has a fault, but rebooting the machine will not help!  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Network Interface**

One of the options for monitoring network activity is to look at the number of received bytes on one or more of the interface devices. These devices are listed by commands such as "ifconfig" that report on the network setting (including the received and sent volume of data), and the raw values can be seen by looking in the special file /proc/net/dev  
  
The test is enabled by the "interface=" option, for example:  
  
interface = eth0  
  
More than one interface can be checked by including more lines similar to the above, but this test is only for **physical interfaces**, so aliased IP addresses (seen as "eth0:1" and similar in ifconfig's output) cannot be checked for correct operation.  
  
The basic check here is that successive intervals see a different value of RX bytes, implying the interface is up and receiving something from the network. Short periods of outage are OK with V5.15 of the daemon if the retry-timeout value is used (default 60 seconds).  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Network "ping"**

The watchdog daemon can also actively test a network using the "ping" packet, more formally known as ICMP type 0 - Echo Reply. This sends out a special data burst and listens for an acknowledgement, implying that the network interface, the network itself, and the target machine are all OK.  
  
NOTE: Before using this option you must get permission from the administrator of the network, and of the target machine, that this action is acceptable.  
  
The test is enabled by the "ping=" keyword, for example:  
  
ping = 192.168.1.1  
ping = 192.168.1.100  
  
The ping target, as shown in this example, is the IPv4 numeric address of the intended machine.  
  
The daemon normally attempts up to 3 pings per poll interval, and if one of those is successful the link is assumed to be alive. The number of attempts per interval is configured by the value:  
  
ping-count = 3  
  
Unlike TCP/IP links, there is no guarantee of an ICMP packet getting through, so it is sensible to attempt more than one test before assuming a link is dead. However, going to a high value of "ping-count=" results in a small window for return before the packet is discarded for not matching the recently-sent value, potentially leading to a failure to detect.  
  
The default settings (1 second polling and 3 ping/interval) puts the upper limit on network round-trip delay as 333ms. It is unlikely you would see such a long delay unless going via a geostationary satellite, which is very unlikely on a LAN. However, you should always check with the "ping" command what the typical delays are before using this option.  
  
NOTES:

1. The daemon (currently) has no DNS look-up ability, nor is it able to handle IPv6 addresses.
2. A machine address of -1 (255.255.255.255) cannot be used, even though it is a legitimate value, because it matches the error return value of the function [inet\_addr()](http://linux.die.net/man/3/inet_addr).
3. It has been reported by some users that older/slower computers sometimes don't respond quickly enough to the ping packet with the default 1s polling interval, so you may need to try 5 or 10 seconds.
4. Caution should be used with the ping option, because if the target machine (or the network switch, etc) should be interrupted than the watchdog will reboot. Therefore if ping is the best option for a given situation, choose a reliable and local target: often the network/router will respond to ping and is the shortest path and least likelihood of being rebooted.
5. If the ["verbose"](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Verbosity) option is enabled, the successful ping response times are logged to syslog.

Make sure you test this option with differing system loads (CPU & network)!  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**File Monitoring**

The daemon can be configured check one or more files. The basic test is if the file exists (which can check the mount status of various partitions and/or network file systems), but it can also be configured to check the file age (for example, to check for activity on log files, incoming data, etc). In addition, the V5.15 version performs this test using a process fork, so it indirectly checks for other serious errors (out of process table space, memory, etc).  
  
The basic test requires an entry of the form:  
  
file = /var/log/syslog  
  
In this example it will check for the existence of that file, however, to check that the file is being updated, the next configuration line could be something like:  
  
change = 1800  
  
This will modify the file check to also verify that the time stamp of the file, in this example, has changed in the last 1800 seconds. You must provide a "change=" line after **every file** you want age-tested.  
  
NOTE: If using this test on a file that is hosted on a network file system you need to ensure reasonable time synchronisation of the two computers, as normally the file's time-stamp is updated based upon the file server's time when it is written/closed.  
  
This is best achieved by using the NTP daemon on both. If you have security issues big enough to prevent even a firewall-filtered access to a selection of 4 or so NTP servers, then you are doing something very important and hence should buy your own GPS/time-server for local use (ideally two for redundancy)!  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Process Monitoring by PID File**

The usual method of managing daemon on a Linux system relies on each daemon writing its process identification number (PID) to a file. This file is used for the 'stop' or 'restart' sort of action when you need to manage a running process. It has the advantage of being a unique identifier (while the process is running) so there is no risk of accidentally killing another process of the same name.  
  
These files are usually kept in the /var/run directory (along with other lock/run status files), and all daemons are supposed to remove its PID file on normal exit to clearly indicate the process has stopped.  
  
The watchdog daemon can be configured to check for the running of other daemons by means of these PID files, for example, the current Ubuntu syslog service can be checked with this entry:  
  
pidfile = /var/run/rsyslogd.pid  
  
When this test is enabled, the watchdog tries to open the PID file and read the numeric value of the PID from it, then it uses the [kill()](http://linux.die.net/man/2/kill) function to attempt to send the null (zero) signal to this process to check it is running.  
  
You could use the watchdog daemon's repair script to act on a process failure by restarting it, however, the usual way of doing this is via the respawn command. For Ubuntu 12.04 that uses upstart to manage system processes, this is covered here: <http://upstart.ubuntu.com/wiki/Stanzas#respawn> but remember also to set up respawn limits to prevent a fault endlessly retrying. The equivalent for systemd (e.g. for Ubuntu 15.10 and later) is documented here: <http://www.freedesktop.org/software/systemd/man/systemd.service.html> (search for the "Restart=" section).  
  
More generally, you need to consider why a process might fail, and if that is best fixed via a reboot. If you have set a respawn limit, then eventually it will stay failed and the watchdog can then reboot to hopefully recover from the underlying fault (out of memory, resource unmounted, etc).  
  
NOTES:

* If a daemon crashes and fails to clean up the PID file there is a slight possibility of its old PID being re-used within the retry-timout period on a machine with a lot of activity. In such cases you may wish to set the retry time to a small value, say 1-2 seconds, so that a process restart is not going to trigger a fault action but a real outage will (for example, the administrator doing this with 'service rsyslog restart', or a HUP signal used to reload and rotate logfiles, etc).
* The reading of the PID files has no protection against the system calls blocking, this should not happen on a local file system but is a risk on a network file system (which is why the [file monitor](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#File_Monitoring) uses a process fork). However, this is a very unlikely situation given that most daemons use /var/run for the files.

[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Memory Test**

There are two options for testing how much free memory is left in the system, and immediately rebooting if it falls below an acceptable amount. The parameters are configured in "memory pages" as these are the smallest allocatable block for the virtual memory system, with 4096 bytes per page for a x86-based machine. The original memory 'test' is to check for the reported free memory (min-memory) as a [passive test](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Memory_Measurement) of resources, but later an option was added to attempt to allocate memory as an [active test](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Active_Testing) of available resources (allocatable-memory).  
  
For example, to configure a passive test for a 40MB threshold:  
  
min-memory = 10000  
  
However, this is not as simple and easy test to use as you might imagine! The reasons for this difficulty are:

* Understanding the memory indicators used.
* How much memory is usable in practice.
* The Linux "Out Of Memory killer" (OOM).

**Memory Measurement**

The first of these is reasonably easy to explain, the watchdog daemon reads the special file /proc/meminfo and parses it for the two entries such as these:  
  
MemFree:      13099484 kB  
Buffers:          888468 kB  
Cached:         14686428 kB  
  
Together they imply this example machine has 28674380kB (24.34GB) of "free memory" which is a total of 7168595 pages of 4kB to the virtual memory manager. A more details description is available here: [http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/proc.txt](http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/proc.txt#n832)  
  
The program 'free' provides an easy way of getting the memory use statistics for the machine, and 'top' also provides a useful summary.  
  
Why this calculation method, and not including free swap space? Well basically if you start using any significant amount of swap space you risk a very slow machine or the OOM killer causing havoc. By using a test that represents available memory before any swap is factored in then any test threshold will result in some RAM always being free even if swap is disabled.  
  
The traditional set-up for a Linux box is to have twice the physical RAM size as swap space, in practice the main reasons for this were either historical (when RAM was so small that swap was *really* needed) or to support hibernation (where the system RAM and state are saved to disk to allow a resume later to the same state). However, there are arguments for less, or more, depending on the machine's work load.  
  
Understanding what the machine has to do is essential for sensible configuring of the watchdog!  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Usable Memory**

So could this example machine, in practice, run a 24GB footprint program? Well it depends, but probably the answer is no!  
  
On the positive side, normally a lot of RAM is "used" providing file system caching, in which case you could run a large program and just suffer less effective disk caching as Linux will relinquish cache in preference to swapping other stuff to disk.  
  
Otherwise, if you are using more than a small fraction of your physical RAM in the form of swap space, then your machine may become horribly slow and the load-averages will climb high as a result. This could reboot the machine if you are [testing load averages](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Load_Averages).  
  
Hence if you are worried about a memory leak bringing your system to a grinding halt, you need to either:

* Aggressively test load averages (but beware of them peaking due to legitimate activity) to fail on the actual sluggishness of memory paging,
* Set a memory limit that tests for swap use that could be too slow for a usable machine and/or implies a memory leak is happening.
* Better still, look at setting limits on the memory use for any known at-risk processes using the bash command 'ulimit' before starting them, or using [cgroups](https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt) to achieve the same goal of stopping the bad behaviour of a few processes/users from bringing the machine down.

Of course, you might just have some unusual case where a lot of memory is needed, but is cycled slowly and so a lot of swap usage is tolerable, but that is an unusual case. With RAM sizes of 4GB being small for typical PC/laptop use, and disk read/write speeds often being 50-100MB/sec, swapping 4GB could take over a minute of time!  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**The OOM Killer**

Finally, there is the question of when no swap space is used, or attempting to use all of a modest swap size. In this case you have to juggle the limit that is worth rebooting for with the actions of the 'Out Of Memory killer'.  
  
The OOM is used to recover from the occasional program that eats up too much memory and thus risks bringing the machine down, a problem that is complicated by the way Linux over-commits memory allocation and then relies on the OOM killer to deal with situations when it is used up. More information is provided here: <http://lwn.net/Articles/317814/>

* An example of the OOM working as intended is a user leaving a web browser with a large number of tab open for a long time and it eating up all of the system memory. In this case the OOM should recognise it as a good candidate for killing and terminate it, thus saving the rest of the machine.
* Alternatively, an example of the OOM failing badly is a [fork bomb](http://en.wikipedia.org/wiki/Fork_bomb). In this case the machine's memory is rapidly used up by an enormous number of small useless processes. Unfortunately to the OOM, they do not look attractive for killing due to the low memory use per process, and thus it will start killing off more important processes such as syslog, etc.

In the case where little or no swap is used, memory exhaustion is very rapid in some cases (e.g. fork bomb) and it can be difficult to choose a threshold for "min-memory=" that is safe from accidental reboots, but not going to allow the OOM to render the machine unusable due to a memory leak. Thus it might be more sensible to disable the OOM Killer and rely on a modest threshold for the watchdog daemon's memory test.  
  
If you have the option to use swap space, then you probably can leave the OOM Killer at its default state and set a min-memory threshold that guards against unreasonably large swap usage. This, in conjunction with the load averages test, is also a reasonably reliable way of using the watchdog to properly reboot a machine suffering from a fork bomb attack (i.e. without needing the hardware timer to deal with a frozen kernel and risk file system corruption).  
  
Finally another bit of advice - do not use swap ***files*** if you can possibly avoid it. Always use dedicated swap partition(s) on the local storage device(s). This makes the watchdog reboot process quicker and safer as unmounting a bloated swap file can take a long time and is needed to unmount the file system, whereas it is safe to reboot without disabling swap on a partition as it is essentially unstructured space.  
[[top of page]](file:///\\media\pscpc\HTML\Public\watchdog\watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Active Testing**

The active tests uses the [mmap() function](http://linux.die.net/man/2/mmap) to attempt to allocate the configured amount of memory, if successful it is immediately freed. This is used in preference to the [malloc() function](http://linux.die.net/man/3/malloc) due to Linux's policy of over-committing memory. Basically, until you try to use the memory offered by malloc you don't really know if it is available!  
  
However, it is important to realise what active testing implies - if you test for, say, 40MB free then the watchdog will attempt to grab that much on every polling interval before releasing it again. For that short time you might well have almost zero memory free to any other application and the test will pass. In addition, this test will result in other memory being paged to the swap file (if used) to permit its allocation. Thus the active test is a pretty good indication that **some** memory can still be found, but not that any other application could safely use that!  
  
So when using active testing do not try too big a value (unless, of course, you are simply testing the watchdog's behaviour), as the CPU load and implications for [load averages](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Load_Averages) triggered by the allocation and paging of memory can be significant. In addition, you have the exactly the same underlying problem in deriving a meaningful measure of usable memory for the system & applications as in the passive test of reported free memory.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Test/Repair Scripts**

To extend the range of tests that the watchdog daemon can use to probe the machine's health, it is possible to run one or more "binary" (i.e. executable program) by means of a process fork followed by an [execl()/execv()](http://linux.die.net/man/3/execl) call. The return code of this is zero if all is OK, or non-zero to indicate an error.  
  
Similarly it is possible to have a repair binary that is called on most errors to, where possible, correct the error without requiring a reboot. In this case the repair binary returns zero if it believes the error was fixed, or non-zero to signal that watchdog action is needed.  
  
Although they are refereed to as "binaries" in most cases a bash script (or similar) will be used to implement them, possibly with some custom programs. There is a separate section on [writing test/repair scripts](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-test-repair-scripts.html) covering this in much more detail.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Version 0 Test & Repair**

Originally the watchdog daemon had the option to configure a single test binary, and a single repair binary using the keywords:  
  
test-binary = /usr/sbin/watchdog-test.sh  
repair-binary = /usr/sbin/watchdog-repair.sh  
  
These are known as "V0" test & repair actions. With V5.15 of the watchdog it is possible to have multiple V0 test binaries configured this way, but still only one repair binary.  
  
NOTE: The V0 test binary should be considered as 'depreciated' and used for reverse compatibility only, and the the [V1 test/repair script](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Version_1_TestRepair) mode of operation used when ever possible. By doing so the V0 repair binary (see below) only has to support the watchdog built-in tests (ping, file status, etc) and not any test binary.  
  
The test binary is simply called without any arguments, and is expected to return the appropriate value. The repair binary is called with the system error code, and the "object" that caused the error. For example, an "access denied" error for reading file /var/run/somefile.pid would result in this call:  
  
/usr/sbin/watchdog-repair.sh 13 /var/run/somefile.pid  
  
The test action, and the repair action, both have time-out values associated with them. If the binaries takes long than these times they, and their process tree, are killed with SIGKILL and are treated as an error return. These time out values are configured with:  
  
test-timeout = 60  
repair-timeout = 60  
  
In most cases 60 seconds is much longer than needed, and there is a good case for reducing this to, for example, 5 seconds unless the machine is exceptionally busy, or the action could take significant time (e.g. ntpq querying the in-use servers from around the world for synchronisation status).  
  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Version 1 Test/Repair**

Later in the development of the watchdog daemon (around Jan 2011) it had the facility added to automatically load any executable files from a specific directory. This is similar to a number of other Linux services that have locations from which settings or programs are automatically loaded (e.g. /etc/cron.d/).  
  
This default location is /etc/watchdog.d/ but the installation process might not create it (for example, Ubuntu 10.04 and 12.04 ***do not***). The directory is configured by the variable:  
  
test-directory = /etc/watchdog.d  
  
What is special about the "V1" binaries is they are expected to be their own repair program. To illustrate this with an example, if a V1 program is called for a test action the call is like this:  
  
/etc/watchdog.d/test-pid.sh test  
  
If this returned a code of 13 for "access denied", then the same program is called again to repair it, in a manner similar to the V0 repair call, as shown in this example:  
  
/etc/watchdog.d/test-pid.sh repair 13 /etc/watchdog.d/test-pid.sh  
  
In this case it can safely ignore the 3rd argument because it knows it will only ever be called to repair its own actions. As for V0, if a repair is possible then it should do this and return zero, otherwise it should ideally return the original error code (13 in this example).  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Repair & Retry Time-Out**

One of the features added with V5.15 of the watchdog was a more flexible way of dealing with transient errors.  
  
An example of this is monitoring a log file for its age to make sure that something is updating as expected, but during a log-rotation the file might be removed for a short period, leading to the risk of an unwanted reboot. With V5.13 the "softboot" command line option would enable a reboot on file access failure, which is too risky here, otherwise you risk the file being missing for a long time as a real error, but being ignored by the daemon, also a risk.  
  
With V5.15 the solution that was implemented is to have a retry time-out value that is used to test the age of a persistent error, and if it exceeds this time without once going away, then it is treated as an error and a repair or reboot actioned. This time limit is configured as:  
  
retry-timeout = 60  
  
This is the time, in second, from the first error on a given "object" before successive errors trigger an action. If it is set to zero then it acts much like the old "softboot" action and any error is immediately actioned including transient problems (normally too much of a risk).  
  
However, the time-out behaviour depends on at least a 2nd error occurring, even if the poll interval is longer then the retry time-out. Basically if you get a "good" return after an error return it is reset and the time ignored.  
  
Another related feature added with V5.15 is the repair limit. With V5.13 a repair script could return zero even if it failed to successfully repair the problem and no action would be taken even if this was repeated over and over again with no sign of the fault clearing.  
  
Now there is a limit to repair attempts ***without success*** configured by:  
  
repair-maximum = 1  
  
If set to zero then it is ignored (i.e. any number of attempts is permitted, as for old system). Otherwise this is the number of successive repair attempts against one "object" allowed. If the repair is successful at least once (a "good" return from the object's test, which can retry as just described) then the counter is reset.  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

**Heartbeat File**

The heartbeat file is a debug option added by Marcel Jansen (I think) to debug the writes to the watchdog device. It is very unlikely to be used again, but is still included in the code. The configuration of the file name is given by:  
  
heartbeat-file = /var/log/watchdog/heartbeat.log  
heartbeat-stamps = 300  
  
[[top of page]](https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html#Linux_Watchdog_Daemon_-_Configuring)

Last Updated on 26-Aug-2019 by Paul Crawford  
Copyright (c) 2014-19 by Paul S. Crawford. All rights reserved.  
Email psc(at)sat(dot)dundee(dot)ac(dot)uk  
Absolutely no warranty, use this information at your own risk.