

OSdog

This chapter describes the functionality and use of the OSdog facility.

The OSdog is the last line of defense that the system provides against a hung machine, whether caused by a software or hardware failure. Only a subset of total failures result in a hang (where the machine is not scheduling threads sufficiently quickly); some can result in a crash from which the machine will recover with a reboot.

The OSdog helps to maintain availability by providing a limit to the recovery time for such cases. The OSdog does not attempt to catch all system failures. It assumes that your application is capable of determining if there is sufficient I/O throughput, memory, and computing resources for its needs. The availability of thread scheduling enables you to monitor these and take appropriate action when required.

The chapter contains the following sections:

- “Operation” on page 94
- “Design Principles” on page 95
- “OSdog Software” on page 96
- “Diagnosing OSdog Timeouts” on page 101
- “Debugging the Operating System” on page 104
- “Handling Panics” on page 104
- “User Patting Description and Example” on page 106
- “Developing a Custom User Pat Daemon” on page 107
- “Configuring the Default User Patting” on page 108

Operation

Hardware

The OSdog protects the system from hangs that can be caused by either hardware or software failure. Each system motherboard contains a hardware timer that increments until either:

- The timer is reset to zero (the OSdog is *patted*), or
- The timer reaches an expiry value, at which point an OSdog reset is issued

Under normal operating conditions, the software regularly pats the OSdog timer, indicating that the system is operating correctly. If the system hangs, the pats do not occur, and the timer causes an OSdog reset event, which allows the system to reboot and recover normal operation.

The effect of an OSdog reset is dependent upon the system mode of operation (fault-tolerant mode or split mode). In fault-tolerant mode, an OSdog reset event causes the appropriate side of the system—CPUset, CAF, disks and PCI cards—to be power cycled. In split mode, only the appropriate side's CPUset is reset due to the potential of resource sharing across sides.

There are two OSdog devices (one per motherboard), and these are associated with that motherboard's side of the system. Each OSdog must be patted individually (this is transparent to the user) to stop an OSdog reset occurring. The OSdog is only enabled on CPUsets that are performing useful work. In particular, the OSdog is not enabled on a secondary CPUset (in fault-tolerant mode) because that CPUset is performing system monitoring operations and it is not appropriate to enable the OSdog at that point. When the secondary CPUset is brought into sync, the OSdog will be enabled for that side.

Solaris Software

The Solaris operating environment provides a multiple stage OSdog operation that simplifies the diagnosis of hangs. In addition to the hardware OSdog protection, if the software detects a system hang that would cause the hardware OSdog to trigger, it panics the system to cause a reboot and to produce a core dump. This makes it possible to diagnose a large number of system hangs.

The principle of operation behind this feature is similar to the Solaris kernel deadman.

Design Principles

The OSdog feature has been developed with the following in mind:

- The OSdog triggers when the system is hung.
- The OSdog does not trigger when the system is not hung.
- All OSdog events are reported.
- The OSdog only runs on a side of the system whose CPUset is doing useful work.
- OSdog protects as much code as possible.
- A latent fault checker performs periodic checks for possible OSdog faults.
- Wherever possible, diagnostic information is saved for fault analysis.

Configuration

The OSdog does not have to be used to protect a system. You can alter the configuration of each motherboard's OSdog, either to enable or disable it. This allows for situations where the expected system operation exceeds the bounds set for OSdog operation (for instance, debugging).

The configuration information is stored in the motherboard configuration EEPROMs at locations `EE_MBD OSDOG_A` and `EE_MBD OSDOG_B`. Note that both locations should contain the same value. This information is read during system initialization and is used to determine the initial state of the OSdog. After this time, the OpenBoot PROM caches the information in the NVRAM device on the CPUset and makes it available as variables `osdog-a` and `osdog-b`. It also makes it available as properties `osdog-a` and `osdog-b` in the `/u4ft-options` device node.

Note – Both motherboard EEPROMs should contain the same data. An update of the OSdog configuration involves changing the EEPROMs on both motherboards.

The only supported method of updating the OSdog configuration is using `set-conf-osdog` in the OpenBoot PROM. This command must be used immediately after a system reset (and must be followed by a system reset).

To enable the OSdog, type:

```
ok setenv auto-boot? false
ok reset-all
ok h# 4f set-conf-osdog
ok reset-all
ok setenv auto-boot? true
```

To disable it, type:

```
ok setenv auto-boot? false
ok reset-all
ok h# 0 set-conf-osdog
ok reset-all
ok setenv auto-boot? true
```

OSdog Software

u4ftdog - OSdog Driver

The `u4ftdog` driver is a multi-threaded, multi-instance non-STREAMS driver that is responsible for controlling the operating system watchdog. It exports an `ioctl(2)` interface that enables applications to register that they can report system availability. Once registered, the application is responsible for periodically reporting (via the `ioctl(2)` interface) that the system is available. Failure to do this is classed as a system failure and the `u4ftdog` driver then initiates a system reboot using the `cmn_err (9F) CE_PANIC` function.

The `u4ftdog` driver also enables control of the Netra ft 1800 OSdog hardware, which provides a hard failure recovery mechanism. Failure to regularly pat the OSdog hardware causes a system power cycle/reset to be initiated.

Hardware Interface

The Netra ft 1800 hardware OSdog functionality is provided by a number of registers on each motherboard. These registers implement a timer that counts upwards. In normal system operation, this counter is reset periodically by the

`u4ftdog` driver software writing to the `pat` register. If the register is not patted within a specified time limit, the counter reaches a trigger value, which causes the motherboard hardware to initiate an OSdog timeout.

The actions associated with an OSdog timeout are dependent upon the system's operating mode: in a combined system, the OSdog timeout causes a system power cycle of the side whose OSdog timed out; in split mode, the OSdog timeout causes a CPUset reset on the OSdog's side. The differences in actions are due to the possibility of resource sharing in a split system where one side may be using resources on the other side. If the side is power cycled, the other side's resources are reset.

The OSdog hardware has two possible timeout values:

- Normal (10.74 seconds), which is generally selected when the Solaris operating environment is running
- Extended (85.90 seconds), which is generally selected when system firmware is running

Driver Summary

The `u4ftdog` driver comprises three instances—a control instance and two hardware instances.

- The control instance represents the upper layer of the driver. It handles the user patting `ioctl` interface and performs the patting of the hardware OSdogs.
- The hardware instances represent the OSdog hardware on each motherboard and enable the hardware to be brought online or taken offline in response to CPUset integration, out-of-sync events, and hot swapping of the motherboards. The hardware instances have private interfaces, which are not intended for use by end users.

The driver performs continuous latent fault checking on the OSdog hardware once the OSdog has been enabled. The driver checks monitor bits in the hardware registers and ensures that they are pulsing correctly.

Configuration

The `u4ftdog` driver provides a hardware configuration file, `u4ftdog.conf`, which contains various properties that are used during driver initialization. You should change only the `channel_x_limit` properties; none of the other properties is user-configurable. The properties recognized by the `u4ftdog` driver are:

- `name`
A string property that identifies the driver. It is the same for all instances.

- `parent`
A string that identifies the parent of a particular instance. This property encodes the absolute path of the parent node. For the control instance, this property is denoted as *pseudo* because it is not responsible for driving any hardware.
- `node`
A single character string property that defines the instance of the current node. `A` and `B` represent the OSdog hardware on the respective sides, and `C` represents the control instance.
- `reg`
A 5-tuple integer property that describes the address space that will be mapped by the hardware instances.

The following configuration file properties are common to all three `u4ftdog` instances:

- `channel_x_limit`
A number of integer properties, one per channel (replace `x` with the channel number), that describe initial limits (in seconds) for each pat channel. This is the interval between initialization of the kernel and triggering of the OSdog. These properties are used to provide OSdog protection during the initial stages of the boot. They should be set to a value that represents the time that you want the system to wait before concluding that a particular channel patten has failed to start. If this limit is reached, the software OSdog triggers. You can update this property to reflect individual requirements.
- `no-config`
An integer property that tells the framework that the hardware represented is non self-identifying and will provide a configuration file
- `non-prom`
An integer property that tells the framework that the OpenBoot PROM does not recognize the device; that is, it is not a hardware node
- `u4ft-aware`
An integer property that tells the framework that the instances will provide their own routines to process the `online/offline` state change requests
- `trace`
A string property that encodes the trace flags that cause certain trace messages to be output to the message log

ioctl

The `u4ftdog` driver provides `ioctl`s that enable applications to configure and perform user OSdog patting, and enable system management functions to configure the OSdog.

The following system calls are not supported by the `u4ftdog` driver:

- `read(2)`
- `write(2)`
- `mmap(2)`

During initialization, the control instance creates a number of device nodes (one per pat channel):

```
/dev/u4ftdog:pat $x$ 
```

where x is the pat channel number from 1 to 8.

In addition, each instance creates a device node `/dev/u4ftdog:[ABC]`, which is used by system control functions. These devices should not be used.

User Pat Channel `ioctl`s

The user pat channels enable you to define a custom system hang detection scheme that monitors aspects of the system appropriate to your needs (see `u4ftdogpat(1M)` for an example).

By default, the user pat channels are active only when a process has the channel device special file open and has configured the limit of the channel. This provides a safety net against problems with the monitoring application that cause it to terminate abnormally and thus cause the OSdog to trigger. This behavior can be overridden using the `LINGER_ON_CLOSE` `ioctl` described below.

A typical use of a user pat channel is as follows:

- Open the channel device; set the timeout limit to an appropriate value
- Loop around, monitoring for hangs, and patting the OSdog if the system is not hung; you must pat more frequently than the limit period
- If you want to close the channel, first set the limit to zero (disable the channel) then close the channel device

The following `ioctl`s are applicable to the pat channel devices.

`U4FTIOC OSDOG SET LIMIT`

This enables or disables a user patting channel. The argument is a pointer to `uint32_t`, which represents the desired limit (in seconds) for that pat channel. A pat channel is enabled when a non-zero limit is specified and disabled when a zero limit is specified.

Note – Care should be taken when selecting a limit to ensure that it is appropriate for the expected pat process. In particular, small values (less than 10) are generally inappropriate.

U4FTIOC_OSDOG_PAT

Pat the channel, thus informing the `u4ftdog` driver that the patting process is active and that the driver should reset the pat channel's OSdog timer. The `ioctl` takes no argument (supply a NULL pointer).

The patting must be performed often enough to ensure that the OSdog does not trigger. Take care to ensure that any scheduling delays (if permissible) are accounted for.

U4FTIOC_OSDOG_LINGER_ON_CLOSE

Specify that the driver should not automatically disable the pat channel if the channel is closed. This is provided to enable system shutdown operations to be OSdog protected.

The `ioctl` takes a `uint32_t` parameter whose value is 1 to enable it and 0 to disable it.

Note – If a device is closed and then re-opened, the `ioctl` will be disabled.

Diagnosing OSdog Timeouts

OSdog timeouts are generated in response to hangs detected by either the OSdog hardware or software. These timeouts cause the system to take corrective action by rebooting. The behavior of each type of hang is described below.

Hardware Event

A hardware OSdog timeout is indicative of a number of faults including:

- Hardware connectivity failure between the CPUset and motherboard
- Motherboard failure
- Gross software failure, that is, the system is hung with interrupts locked out

There is independent OSdog hardware on each side of a system. It is responsible for power cycling and resetting its own side if the OSdog is not patted for any reason. This structure means that, in some circumstances, the OSdog can trigger on only one side of a system, which often indicates a hardware or connectivity problem on that side. There is a small time window during CPUset re-integration operations when a hang would cause the OSdog to fire on only one side of the system. This is due to the software conventions used by the CMS. In these circumstances, the system recovers on one side and the CMS initiates recovery on the second side.

Hardware events are reported by the system in the following locations:

- Message on the console during initialization of the OpenBoot PROM

This reports that the reset was due to an OSdog event:

```
Last reset was caused by OSdog expiry
```

- The value of *reset-reason* in the OpenBoot PROM device node

If the reset was due to OSdog, the property has the value `OSDOG`. To examine this device node, type the following:

```
{0} ok cd /
{0} ok .properties
initial-rcp-status 13 46 c6 e2 26 da b7 1b bc e8 12 ad 66 16 b3 15
reset-reason      OSDOG
energystar-v2
...
{0} ok
```

The OpenBoot PROM reports *reset-reason* has the value OSDOG.

- A message is placed in the status log during boot, as follows:

```
[380c3a04.123bdd30]      M      0      <u4ftcmd#0>      (  
      OBP reports reset-reason as "OSDOG"
```

Software Event

A software OSdog timeout occurs as a result of an event that causes the system to hang. Specific causes include:

- High level interrupt failure, which causes level 14 soft interrupts to be blocked

This failure actually results in a hardware OSdog event because the OSdog software cannot monitor the system and pat the hardware OSdog. The type of failure occurs only when interrupts on all processors are blocked.

- High level interrupts locked out (levels 11 through 13)

This stops the OSdog from patting the hardware, and the deadman (level 14) interrupts see that patting has stopped and cause the system to panic with the following message:

```
panic[cpu0]/thread=0x3002be80: OSdog patting is too infrequent or  
stopped (OSdog hard hang)  
stopped at:  
Syslimit+0x94:  ta      %icc,%g0 + 125  
kadb[0]:
```

A typical cause of this type of hang is a driver high level interrupt routine that becomes stuck in a non-terminating loop; for example, waiting for a mutex that is never released by the current owner.

It can also occur as a result of a check for a hardware condition that never occurs. That is, the code can look for the completion of a command in a device register, but the register is broken and is returning false values.

- Clock thread stopped (level 10 interrupts not being delivered)

The OSdog monitors the *lbolt* kernel variable, which is incremented once every ten milliseconds. If this is not incremented for 20 seconds (configurable), the OSdog reports a hang:

```
panic[cpu0]/thread=0x3002be80: OSdog detected system hang, pat
channel=0, limit=20
stopped at Syslimit+0x94: ta %icc,%g0 + 125
kadb[0]:
```

Note – This type of hang is always reported as pat channel 0 (zero).

This type of hang is typically caused by scheduler problems, including excessive scheduling load.

- User pats not delivered within the specified limit time

If a user pat channel is enabled and the appropriate pat daemon has not performed a pat ioctl within the specified time limit, the system panics and reports an OSdog hang. This produces a message similar to that in previous case, but the pat channel will be the one that has not been patted.

```
panic[cpu0]/thread=0x3002be80: OSdog detected system hang, pat
channel=1, limit=10
stopped at Syslimit+0x94: ta %icc,%g0 + 125
kadb[0]:
```

Causes of this type of hang are user defined, although it can also result from an excessive boot time, where the pat counter reaches the initial timeout limit before the user patting daemon has started.

Similarly, a timeout can be caused by a shutdown that hangs or takes too long (for example, as a result of NFS unmounting problems).

Software events are reported in a number of ways:

- The most immediate report is the system panic message that appears on the system console. This reports one of the messages (shown above), which indicates the possible causes of a hang.
- The same message is also reported in the CPUset NVRAM log. This is obtained from the CPUset during reboot and stored in the NVRAM log file.

```
[380c490d.51c6f43b] F 0 <u4ftdog#4> ()
OSdog detected system hang, pat channel=1, limit=10
```

After a software OSdog timeout, the OpenBoot PROM does not report that the reset was due to an OSdog event. This is because the OpenBoot PROM reports the state of the hardware and sees the software OSdog timeout as a soft reset.

Debugging the Operating System

The OSdog can cause difficulties when you debug the operating system. In particular, breakpoints set in either `kadb` or OpenBoot PROM bypass normal methods of entry into the debugger and cause the hardware OSdog to trigger. To prevent this from happening and facilitate debugging, `u4ftdog` reduces the level of system protection when a debugger is being used. This means that, in this case, the OSdog is always patted when entering `kadb` or OpenBoot PROM.

The OSdog driver detects that a debugger is loaded by the presence of either `kadb` or the `misc/obpsym` kernel module. Under all other circumstances, the OSdog is patted only in OpenBoot PROM, if you enter this using a console hardware or software break sequence, and when the Solaris operating environment is shutdown to run level 0.

Note – It is essential, therefore, that you do not run with a debugger enabled unless absolutely necessary for debugging purposes.

Handling Panics

Operating system panics need special treatment by the OSdog software to allow time for the file systems to sync and for panic dumps to be made. This is essential to minimize file system check time and to aid fault diagnosis.

When a panic is initiated, the `u4ftdog` driver switches into panic mode, where it:

- Sets the OSdog hardware to the extended timeout setting
- Relaxes the usual checks for system operation

The latter is achieved by placing a time limit on panic processing. If this time limit is exceeded, the driver invokes a secondary OSdog panic with the message

```
panic[cpu1]/thread=0x64587b80: panic OSdog timeout
```

and then no longer pats the OSdog hardware. This combination of activities provides another opportunity for the panic code to continue, but limits any remaining time to the hardware OSdog timeout period (that is, 85.90 seconds). If the panic has not initiated a system reboot within this time, the hardware OSdog triggers, causing a power cycle and a reboot.

Note – In certain circumstances, particularly when the panic is originated by the interrupt subsystem, it may not be possible to pat the OSdog at all during the panic. In these cases, the OSdog hardware resets the machine in 85.90 seconds unless the panic succeeds within that time. Normally, the panic processing would not succeed.

You can tune the timeout period for panic handling by setting the `u4ftdog_max_panic_time` variable in `/etc/system`, that is:

```
set u4ftdog:max_panic_time=300
```

The variable's value is the number of seconds the system waits before declaring that the panic has failed to complete. The default time for this is 160 seconds.

Panic and the Debugger

A panic provides an opportunity to debug the kernel and determine the cause of the hang. You can perform this either immediately, if the kernel debugger (`kadb`) is loaded, or later using the core file, which is saved as part of the panic.

Note – Check that `savecore` is enabled (see `/etc/rc2.d/S20syssetup`).

In production systems, you may not want the system to enter the kernel debugger. In this case, set the kernel variable `nopanicdebug` to 1 by placing an entry in `/etc/system` as follows:

```
set nopanicdebug = 1
```

Note – The system must be rebooted for this setting to take effect.

User Patting Description and Example

A sample user patting configuration is provided on the system by default. You can tune this configuration to provide cover in a way that is appropriate to your needs. The patting is provided by the `u4ftdogpat` program, which pats the OSdog at a specified time interval. This enables you to monitor thread scheduling for specific user priorities.

The configuration is split into a number of areas:

- **Initial boot** When the `u4ftdog` driver comes online, a default OSdog timeout period is initialized and the pat channel is enabled. The `/etc/init.d/u4ftdog_boot` script is then responsible for starting an initial instance of the patting program, which starts patting the OSdog. This guards against hangs in kernel initialization that stop the boot from getting underway.
- **Normal running** Normal running mode is entered once Solaris has checked disks and mounted the `/usr` partition. In this mode, the OSdog is reconfigured to provide appropriate coverage for the system and `cmsmonitord` is used to monitor the `u4ftdogpat` program and ensure that it is always running. If the daemon dies for any reason, it will be restarted by `cmsmonitord`.

Entry into normal running mode is controlled by the `/etc/init.d/u4ftdog` script. This responds to parameters `start` and `restart` (and also `stop`, but see the following bullet point). The `start` version is executed during the latter stages of `/etc/rcS.d/` scripts and this starts the dynamically-linked version of `u4ftdogpat` with the appropriate configuration parameter, and then kills the statically-linked version.

`restart` is used by `cmsmonitord` if it detects that `u4ftdogpat` has died. In this case, there is no attempt to kill off the statically-linked version.

- **Shutdown** Shutdown may involve delays whilst file systems are being unmounted, for example, syncing file systems. This can take longer than the standard timeout period for user patting. This is compounded by the fact that processes are also killed during shutdown causing patting to stop. To protect against hangs at this stage, the pat channel is reconfigured with a longer timeout period and with the `LINGER_ON_CLOSE` option set to ensure that the channel remains active once the pat process is killed.

The default pat configuration uses pat channel 1. If you want to provide monitoring support for your application, use a different channel and leave the existing monitoring in place.

Developing a Custom User Pat Daemon

The OSDog user patting API has been developed to enable you to specify a criteria for satisfactory system operation and to develop a monitor daemon that checks that this criteria is being met. A suggested algorithm for doing this is as follows:

CODE EXAMPLE 4-1 User Pat Daemon

```
#include <sys/types.h>
#include <fcntl.h>
#include <sys/u4ftdog_io.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int fd;
    int period = 5;      /* Interval between checks on system */
    uint32_t limit = 40; /* Channel timeout value */
    int shutdown = 0;   /* do shutdown? - maybe set by a signal */
    int system_ok = 1;  /* Set by status monitor function */

    /* initialise system monitor application (insert your code) */

    /* Open pat channel */
    fd = open("/dev/u4ftdog:pat2", O_RDWR);

    /* Configure appropriate limit */
    ioctl(fd, U4FTIOC OSDOG_SET_LIMIT, &limit);

    /* Perform loop until shutdown is requested */
    /* A signal handler may be used to do this */
    while (!shutdown) {

        /* check system/application status (insert your code) */

        /* system_ok = sys_status(); */
        if (system_ok) {
            ioctl(fd, U4FTIOC OSDOG_PAT, NULL);
        } else {
            printf("System needs attention\n");

            /* take recovery action (insert your code) */

            /* possibly pat the osdog during recovery period */
            /* (insert your code) */
        }
    }
}
```

CODE EXAMPLE 4-1 User Pat Daemon (Continued)

```
    }
    poll(NULL, 0, (period * 1000));
}
/* Orderly close requested */
/* Disable osdog */
limit = 0;          /* Limit of zero means disable */
ioctl(fd, U4FTIOC OSDOG_SET_LIMIT, &limit);

/* Close pat channel */
close(fd);

/* stop system monitoring (insert your code) */
exit(0);
}
```

Note – Insert your own tested and proven program at the points indicated in the algorithm.

Configuring the Default User Patting

Adjust the configuration for the default user patting by modifying file settings as follows:

- Initial timeout between OSdog coming online and the user pat daemon starting

In the file `/platform/SUNW,Ultra-4FT/kernel/drv/u4ftdog.conf`, change the setting of `channel_x_limit` to a different value. The value specified is the initial timeout limit in seconds.

Note – Replace `x` with the number of the channel which is to be changed.

- Boot timeout

In the file `/etc/init.d/u4ftdog_boot`, change the `-l` parameter for `u4ftdogpat` to the desired timeout limit.

- Normal running timeout

In the file `/etc/init.d/u4ftdog`, change the `-l` parameter for `u4ftdogpat` to the desired timeout limit.

Note – Change only the parameter for the `u4ftdogpat` invocation that is part of the `start` and `restart` option.

- Shutdown timeout

In the file `/etc/init.d/u4ftdog`, change the `-l` parameter for `u4ftdogpat` to the desired timeout limit.

Note – Change only the parameter for the `u4ftdogpat` invocation that is part of the `stop` option.

Adjust other parameters as specified in the `u4ftdogpat` manual page.

Obtaining OSdog Settings Using `cmsfruinfo`

You can obtain the current OSdog settings stored in the motherboard using the `cmsfruinfo` command.

- When the OSdog is disabled:

```
# cmsfruinfo -l A-MBD EE_MBD_OSDOG
EE_MBD_OSDOG=
  EE_MBD_OSDOG_0=0
  EE_MBD_OSDOG_1=0
```

```
# cmsfruinfo -l B-MBD EE_MBD_OSDOG
EE_MBD_OSDOG=
  EE_MBD_OSDOG_0=0
  EE_MBD_OSDOG_1=0
```

- When the OSdog is enabled:

```
# cmsfruinfo -l A-MBD EE_MBD_OSDOG
EE_MBD_OSDOG=
  EE_MBD_OSDOG_0=79
  EE_MBD_OSDOG_1=79
```

```
# cmsfruinfo -1 B-MBD EE_MBD_OSDOG
EE_MBD_OSDOG=
    EE_MBD_OSDOG_0=79
    EE_MBD_OSDOG_1=79
```

System Clock Thread Monitoring

You can adjust the timeout for system clock thread monitoring by setting the variable `u4ftdog_lbolt_timeout` in `/etc/system`.

For example:

```
set u4ftdog:u4ftdog_lbolt_timeout=40
```

The value is the number of seconds to wait before causing the OSdog to trigger. Reboot the system for this change to take effect.

Note – This timeout limit should be smaller than the timeout limits used for user pat monitoring because the user processes are generally dependent upon the clock for process wake up.

Troubleshooting

- The initial OSdog timeout period is too short and you get an OSdog timeout before you can get to a prompt to correct matters.

Boot to single-user mode (using the `-s` option to boot). When a single-user boot is used, the initial timeout period is not set. Once you are at the single-user mode prompt, you can alter the settings appropriately.

- Problems with initial boot

Use `boot -s`