Rocky Enterprise Linux 9.2 Manual Pages on command 'systemd-coredump.8'

$ man systemd-coredump.8

SYSTEMD-COREDUMP(8) systemd-coredump SYSTEMD-COREDUMP(8)

NAME

systemd-coredump, systemd-coredump.socket, systemd-coredump@.service -

Acquire, save and process core dumps

SYNOPSIS

/usr/lib/systemd/systemd-coredump

/usr/lib/systemd/systemd-coredump --backtrace

systemd-coredump@.service

systemd-coredump.socket

DESCRIPTION

systemd-coredump@.service is a system service to process core dumps. It

will log a summary of the event to systemd-journald.service(8),

including information about the process identifier, owner, the signal

that killed the process, and the stack trace if possible. It may also

save the core dump for later processing. See the "Information about the

crashed process" section below.

The behavior of a specific program upon reception of a signal is

governed by a few factors which are described in detail in core(5). In

particular, the core dump will only be processed when the related resource limits are sufficient.

Core dumps can be written to the journal or saved as a file. In both cases, they can be retrieved for further processing, for example in gdb(1). See coredumpctl(1), in particular the list and debug verbs.

By default, systemd-coredump will log the core dump to the journal, including a backtrace if possible, and store the core dump (an image of the memory contents of the process) itself in an external file in /var/lib/systemd/coredump. These core dumps are deleted after a few days by default; see /usr/lib/tmpfiles.d/systemd.conf for details. Note that the removal of core files from the file system and the purging of journal entries are independent, and the core file may be present without the journal entry, and journal entries may point to since-removed core files. Some metadata is attached to core files in the form of extended attributes, so the core files are useful for some purposes even without the full metadata available in the journal entry.

Invocation of systemd-coredump

The systemd-coredump executable does the actual work. It is invoked twice: once as the handler by the kernel, and the second time in the systemd-coredump@.service to actually write the data to the journal and process and save the core file.

When the kernel invokes systemd-coredump to handle a core dump, it runs in privileged mode, and will connect to the socket created by the systemd-coredump.socket unit, which in turn will spawn an unprivileged systemd-coredump@.service instance to process the core dump. Hence systemd-coredump.socket and systemd-coredump@.service are helper units which do the actual processing of core dumps and are subject to normal service management.

It is also possible to invoke systemd-coredump with --backtrace option. In this case, systemd-coredump expects a journal entry in the journal Journal Export Format[1] on standard input. The entry should contain a MESSAGE= field and any additional metadata fields the caller deems reasonable. systemd-coredump will append additional metadata fields in

the same way it does for core dumps received from the kernel. In this mode, no core dump is stored in the journal.

CONFIGURATION

For programs started by systemd, process resource limits can be set by directive LimitCORE=, see systemd.exec(5).

In order to be used by the kernel to handle core dumps, systemd-coredump must be configured in sysctl(8) parameter kernel.core_pattern. The syntax of this parameter is explained in core(5). systemd installs the file /usr/lib/sysctl.d/50-coredump.conf which configures kernel.core_pattern accordingly. This file may be masked or overridden to use a different setting following normal sysctl.d(5) rules. If the sysctl configuration is modified, it must be updated in the kernel before it takes effect, see sysctl(8) and systemd-sysctl(8).

In order to be used in the --backtrace mode, an appropriate backtrace handler must be installed on the sender side. For example, in case of python(1), this means a sys.excepthook must be installed, see systemd-coredump-python[2].

The behavior of systemd-coredump itself is configured through the configuration file /etc/systemd/coredump.conf and corresponding snippets /etc/systemd/coredump.conf.d/*.conf, see coredump.conf(5). A new instance of systemd-coredump is invoked upon receiving every core dump. Therefore, changes in these files will take effect the next time a core dump is received.

Resources used by core dump files are restricted in two ways. Parameters like maximum size of acquired core dumps and files can be set in files /etc/systemd/coredump.conf and snippets mentioned above. In addition the storage time of core dump files is restricted by systemd-tmpfiles, corresponding settings are by default in /usr/lib/tmpfiles.d/systemd.conf. The default is to delete core dumps after a few days; see the above file for details.

Disabling coredump processing

To disable potentially resource-intensive processing by

systemd-coredump, set

Storage=none ProcessSizeMax=0

in coredump.conf(5).

INFORMATION ABOUT THE CRASHED PROCESS

coredumpctl(1) can be used to retrieve saved core dumps independently of their location, to display information, and to process them e.g. by passing to the GNU debugger (gdb).

Data stored in the journal can be also viewed with journalctl(1) as usual (or from any other process, using the sd-journal(3) API). The relevant messages have MESSAGE_ID=fc2e22bc6ee647b6b90729ab34a250b1:

```
$ journalctl MESSAGE_ID=fc2e22bc6ee647b6b90729ab34a250b1 -o verbose
...
MESSAGE_ID=fc2e22bc6ee647b6b90729ab34a250b1
COREDUMP_PID=552351
COREDUMP_UID=1000
COREDUMP_GID=1000
COREDUMP_SIGNAL_NAME=SIGSEGV
COREDUMP_SIGNAL=11
COREDUMP_TIMESTAMP=1614342930000000
COREDUMP_COMM=Web Content
COREDUMP_EXE=/usr/lib64/firefox/firefox
COREDUMP_USER_UNIT=app-gnome-firefox-552136.scope
COREDUMP_CMDLINE=/usr/lib64/firefox/firefox -contentproc -childID 5 -isForBrowser ...
COREDUMP_CGROUP=/user.slice/user-1000.slice/user@1000.service/app.slice/app-....scope
COREDUMP_FILENAME=/var/lib/systemd/coredump/core.Web....552351.....zst
...
```

The following fields are saved (if known) with the journal entry

COREDUMP_UID=, COREDUMP_PID=, COREDUMP_GID=

The process number (PID), owner user number (UID), and group number (GID) of the crashed process.

When the crashed process was part of a container (or in a process or user namespace in general), those are the values as seen outside, in the namespace where systemd-coredump is running.

COREDUMP_TIMESTAMP=

The time of the crash as reported by the kernel (in ?s since the epoch).

COREDUMP_RLIMIT=

The core file size soft resource limit, see getrlimit(2).

COREDUMP_UNIT=, COREDUMP_SLICE=

The system unit and slice names.

When the crashed process was in container, those are the units names outside, in the main system manager.

COREDUMP_CGROUP=

Control group information in the format used in /proc/self/cgroup.

On systems with the unified cgroup hierarchy, this is a single path prefixed with "0::", and multiple paths prefixed with controller numbers on legacy systems.

When the crashed process was in a container, this is the full path, as seen outside of the container.

COREDUMP_OWNER_UID=, COREDUMP_USER_UNIT=

The numerical UID of the user owning the login session or systemd user unit of the crashed process, and the user manager unit. Both fields are only present for user processes.

When the crashed process was in container, those are the values outside, in the main system.

COREDUMP_SIGNAL_NAME=, COREDUMP_SIGNAL=

The terminating signal name (with the "SIG" prefix [3]) and numerical value. (Both are included because signal numbers vary by architecture.)

COREDUMP_CWD=, COREDUMP_ROOT=

The current working directory and root directory of the crashed process.

When the crashed process is in a container, those paths are relative to the root of the container's mount namespace.

COREDUMP_OPEN_FDS=

Information about open file descriptors, in the following format:

fd:/path/to/file

pos: ...

flags: ...

...

fd:/path/to/file

pos: ...

flags: ...

...

The first line contains the file descriptor number fd and the path,

while subsequent lines show the contents of /proc/pid/fdinfo/fd.

COREDUMP_EXE=

The destination of the /proc/pid/exe symlink.

When the crashed process is in a container, that path is relative

to the root of the container's mount namespace.

COREDUMP_COMM=, COREDUMP_PROC_STATUS=, COREDUMP_PROC_MAPS=,

COREDUMP_PROC_LIMITS=, COREDUMP_PROC_MOUNTINFO=, COREDUMP_ENVIRON=

Fields that map the per-process entries in the /proc/ filesystem:

/proc/pid/comm (the command name associated with the process),

/proc/pid/exe (the filename of the executed command),

/proc/pid/status (various metadata about the process),

/proc/pid/maps (memory regions visible to the process and their

access permissions), /proc/pid/limits (the soft and hard resource

limits), /proc/pid/mountinfo (mount points in the process's mount

namespace), /proc/pid/environ (the environment block of the crashed

process).

See proc(5) for more information.

COREDUMP_HOSTNAME=

The system hostname.

When the crashed process was in container, this is the container

hostname.

COREDUMP_CONTAINER_CMDLINE=

For processes running in a container, the commandline of the

process spawning the container (the first parent process with a

different mount namespace).

COREDUMP=

When the core is stored in the journal, the core image itself.

COREDUMP_FILENAME=

When the core is stored externally, the path to the core file.

COREDUMP_TRUNCATED=

Set to "1" when the saved coredump was truncated. (A partial core image may still be processed by some tools, though obviously not all information is available.)

COREDUMP_PACKAGE_NAME=, COREDUMP_PACKAGE_VERSION=,

COREDUMP_PACKAGE_JSON=

If the executable contained .package metadata ELF notes, they will be parsed and attached. The package and packageVersion of the 'main' ELF module (ie: the executable) will be appended individually. The JSON-formatted content of all modules will be appended as a single JSON object, each with the module name as the key. For more information about this metadata format and content, see the coredump metadata spec[4].

MESSAGE=

The message generated by systemd-coredump that includes the backtrace if it was successfully generated. When systemd-coredump is invoked with --backtrace, this field is provided by the caller.

Various other fields exist in the journal entry, but pertain to the logging process, i.e. systemd-coredump, not the crashed process. See systemd.journal-fields(7).

The following fields are saved (if known) with the external file listed in COREDUMP_FILENAME= as extended attributes:

user.coredump.pid, user.coredump.uid, user.coredump.gid, user.coredump.signal, user.coredump.timestamp, user.coredump.rlimit, user.coredump.hostname, user.coredump.comm, user.coredump.exe

Those are the same as COREDUMP_PID=, COREDUMP_UID=, COREDUMP_GID=, COREDUMP_SIGNAL=, COREDUMP_TIMESTAMP=, COREDUMP_RLIMIT=, COREDUMP_HOSTNAME=, COREDUMP_COMM=, and COREDUMP_EXE=, described

above.

Those can be viewed using getfattr(1). For the core file described in the journal entry shown above:

```
$ getfattr --absolute-names -d /var/lib/systemd/coredump/core.Web....552351.....zst
# file: /var/lib/systemd/coredump/core.Web....552351.....zst
user.coredump.pid="552351"
user.coredump.uid="1000"
user.coredump.gid="1000"
user.coredump.signal="11"
user.coredump.timestamp="1614342930000000"
user.coredump.comm="Web Content"
user.coredump.exe="/usr/lib64/firefox/firefox"
...
```

SEE ALSO

coredump.conf(5), coredumpctl(1), systemd-journald.service(8), systemd-tmpfiles(8), core(5), sysctl.d(5), systemd-sysctl.service(8).

NOTES

1. Journal Export Format
   https://systemd.io/JOURNAL_EXPORT_FORMATS#journal-export-format
2. systemd-coredump-python
   https://github.com/systemd/systemd-coredump-python
3. kill(1) expects signal names without the prefix; kill(2) uses the prefix; all systemd tools accept signal names both with and without the prefix.
4. the coredump metadata spec
   https://systemd.io/COREDUMP_PACKAGE_METADATA/