faulthandler Documentation

Release 3.0

Victor Stinner

Sep 15, 2017

Contents

This module contains functions to dump Python tracebacks explicitly, on a fault, after a timeout, or on a user signal. Call `faulthandler.enable()` to install fault handlers for the `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS`, and `SIGILL` signals. You can also enable them at startup by setting the `PYTHONFAULTHANDLER` environment variable.

The fault handler is compatible with system fault handlers like Apport or the Windows fault handler. The module uses an alternative stack for signal handlers if the `sigaltstack()` function is available. This allows it to dump the traceback even on a stack overflow.

The fault handler is called on catastrophic cases and therefore can only use signal-safe functions (e.g. it cannot allocate memory on the heap). Because of this limitation traceback dumping is minimal compared to normal Python tracebacks:

- Only ASCII is supported. The `backslashreplace` error handler is used on encoding.
- Each string is limited to 500 characters.
- Only the filename, the function name and the line number are displayed. (no source code)
- It is limited to 100 frames and 100 threads.
- The order is reversed: the most recent call is shown first.

By default, the Python traceback is written to `sys.stderr`. To see tracebacks, applications must be run in the terminal. A log file can alternatively be passed to `faulthandler.enable()`.

The module is implemented in C, so tracebacks can be dumped on a crash or when Python is deadlocked.

This module is the backport for CPython 2.7. It is part of Python standard library since CPython 3.3: faulthandler module. For PyPy, faulthandler is builtin since PyPy 5.5: use `pypy -X faulthandler`.

faulthandler works on CPython 2.6-3.5.

- faulthandler website (this page)
- faulthandler project at github: source code, bug tracker
- faulthandler at Python Cheeshop (PyPI)
- Article: New faulthandler module in Python 3.3 helps debugging

CHAPTER 1

Example

Example of a segmentation fault on Linux:

```
$ python
>>> import faulthandler
>>> faulthandler.enable()
>>> import ctypes
>>> ctypes.string_at(0)
Fatal Python error: Segmentation fault

Current thread 0x00007fea4a98c700 (most recent call first):
  File "/usr/lib64/python2.7/ctypes/__init__.py", line 504 in string_at
  File "<stdin>", line 1 in <module>
Segmentation fault (core dumped)
```

CHAPTER 2

Nosetests and py.test

To use faulthandler in nose tests, you can use the nose-faulthandler plugin.

To use it in py.test, you can use the pytest-faulthandler plugin.

CHAPTER 3

Installation

faulthandler supports Python 2.7. It may also support Python 2.5, 2.6, 3.1 and 3.2, but these versions are no more officially supported.

Install faulthandler on Windows using pip

Procedure to install faulthandler on Windows:

- Install pip: download `get-pip.py` and type:

```
\Python27\python.exe get-pip.py
```

- If you already have pip, ensure that you have at least pip 1.4 (to support wheel packages). If you need to upgrade:

```
\Python27\python.exe -m pip install -U pip
```

- Install faulthandler:

```
\Python27\python.exe -m pip install faulthandler
```

Linux packages

Linux distribution	Package name
Debian	python-faulthandler
OpenSuSE	python-faulthandler
PLD Linux	python-faulthandler
Ubuntu	python-faulthandler

Some links:

- Debian python-faulthandler package

- Ubuntu faulthandler source package

pythonxy (Windows)

faulthandler is part of pythonxy distribution: free scientific and engineering development software for Windows.

Install from source code

Download the latest tarball from the Python Cheeseshop (PyPI).

To install faulthandler module, type the following command:

```
python setup.py install
```

Then you can test your setup using the following command:

```
python tests.py
```

You need a C compiler (eg. gcc) and Python headers to build the faulthandler module. Eg. on Fedora, you have to install python-devel package (sudo yum install python-devel).

CHAPTER 4

faulthandler module API

`faulthandler.version` is the module version as a tuple: `(major, minor)`. `faulthandler.__version__` is the module version as a string (e.g. `"2.0"`).

Dumping the traceback

`dump_traceback`(*file=sys.stderr*, *all_threads=True*)
Dump the tracebacks of all threads into *file*. If *all_threads* is `False`, dump only the current thread.

Changed in version 2.5: Added support for passing file descriptor to this function.

Fault handler state

`enable`(*file=sys.stderr*, *all_threads=True*)
Enable the fault handler: install handlers for the `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` and `SIGILL` signals to dump the Python traceback. If *all_threads* is `True`, produce tracebacks for every running thread. Otherwise, dump only the current thread.

The *file* must be kept open until the fault handler is disabled: see *issue with file descriptors*.

Changed in version 2.5: Added support for passing file descriptor to this function.

`disable`()
Disable the fault handler: uninstall the signal handlers installed by *`enable()`*.

`is_enabled`()
Check if the fault handler is enabled.

Dumping the tracebacks after a timeout

dump_traceback_later(*timeout*, *repeat=False*, *file=sys.stderr*, *exit=False*)

Dump the tracebacks of all threads, after a timeout of *timeout* seconds, or every *timeout* seconds if *repeat* is `True`. If *exit* is `True`, call `_exit()` with status=1 after dumping the tracebacks. (Note `_exit()` exits the process immediately, which means it doesn't do any cleanup like flushing file buffers.) If the function is called twice, the new call replaces previous parameters and resets the timeout. The timer has a sub-second resolution.

The *file* must be kept open until the traceback is dumped or *cancel_dump_traceback_later()* is called: see *issue with file descriptors*.

This function is implemented using the `SIGALRM` signal and the `alarm()` function. If the signal handler is called during a system call, the system call is interrupted and fails with `EINTR`.

Not available on Windows.

Changed in version 2.5: Added support for passing file descriptor to this function.

cancel_dump_traceback_later()

Cancel the last call to *dump_traceback_later()*.

Dumping the traceback on a user signal

register(*signum*, *file=sys.stderr*, *all_threads=True*, *chain=False*)

Register a user signal: install a handler for the *signum* signal to dump the traceback of all threads, or of the current thread if *all_threads* is `False`, into *file*. Call the previous handler if chain is `True`.

The *file* must be kept open until the signal is unregistered by *unregister()*: see *issue with file descriptors*.

Not available on Windows.

Changed in version 2.5: Added support for passing file descriptor to this function.

unregister(*signum*)

Unregister a user signal: uninstall the handler of the *signum* signal installed by *register()*. Return `True` if the signal was registered, `False` otherwise.

Not available on Windows.

Issue with file descriptors

enable(), *dump_traceback_later()* and *register()* keep the file descriptor of their *file* argument. If the file is closed and its file descriptor is reused by a new file, or if `os.dup2()` is used to replace the file descriptor, the traceback will be written into a different file. Call these functions again each time that the file is replaced.

CHAPTER 5

Differences with stdlib faulthandler

faulthandler is now part of CPython standard library since version 3.3. CPython 3.3 and newer got enhancements which are not available on CPython 2.7 and so this faulthandler backport for CPython 2.7 has limitations:

faulthandler signal handler

The faulthandler signal handler and the `dump_traceback()` function call `PyGILState_GetThisThreadState()` to get the Python thread state of the current thread even if it doesn't hold the GIL. This function uses a *Thread Local Storage* (TLS) variable. Since CPython 3.2, TLS use native functions:

- UNIX/BSD: `pthread_getspecific()`, `pthread_setspecific()`
- Windows: `TlsGetValue()`, `TlsSetValue()`

But CPython 2.7 uses its own implementation of TLS using a single-linked list and a lock. Locks are not signal-safe: using a lock in a signal handler may work or may block forever.

dump_traceback_later()

dump_traceback_later() is implemented with `alarm(seconds)` on CPython 2.7 and so the timer has a resolution of 1 second, whereas CPython 3 uses a "watchdog" thread and a lock with a timeout (`PyThread_acquire_lock_timed()`) with a resolution of 1 microseconds.

`alarm()` requires to set a signal handler for `SIGALRM` whereas the application may want to use this signal for a different purpose. Raising the `SIGALRM` signal has side effects like interrupting the current syscall which would fail with `EINTR` error whereas Python 2.7 has a bad support of `EINTR` errors. CPython 3.5 now automatically retries syscalls failing with `EINTR`: see PEP 475 – Retry system calls failing with EINTR.

CHAPTER 6

Changelog

Version 3.0 (2017-09-16)

- On Windows, faulthandler.register() now also installs an handler for Windows exceptions.

Version 2.6 (2017-03-22)

- Add support for the `PYTHONFAULTHANDLER` environment variable. Patch written by Ionel Cristian Mărieș.

Version 2.5 (2017-03-22)

- Issue #23433: Fix undefined behaviour in `faulthandler._stack_overflow()`: don't compare pointers, use the `Py_uintptr_t` type instead of `void*`. It fixes `test_faulthandler` on Fedora 22 which now uses GCC 5.
- The `write()` function used to write the traceback is now retried when it is interrupted by a signal.
- Issue #23566: enable(), register(), dump_traceback() and dump_traceback_later() functions now accept file descriptors. Patch by Wei Wu.
- Drop support and Python 2.5, 2.6, 3.1 and 3.2: only support Python 2.7. No Linux distribution use these versions anymore, so it becomes difficult to test these versions.
- Add tox.ini to run tests with tox: it creates a virtual environment, compile and install faulthandler, and run unit tests.
- Add Travis YAML configuration.

Version 2.4 (2014-10-02)

- Add a new documentation written with Sphinx used to built a new website: https://faulthandler.readthedocs.io/
- Python issue #19306: Add extra hints to faulthandler stack dumps that they are upside down.
- Python issue #15463: the faulthandler module truncates strings to 500 characters, instead of 100, to be able to display long file paths.
- faulthandler issue #7: Ignore Windows SDK message "This application has requested the Runtime to terminate it in an unusual way. (...)" in test_fatal_error(). It was not a bug in faulthandler, just an issue with the unit test on some Windows setup.
- Python issue #21497: faulthandler functions now raise a better error if `sys.stderr` is `None`: RuntimeError("sys.stderr is None") instead of AttributeError("'NoneType' object has no attribute 'fileno"').
- Suppress crash reporter in tests. For example, avoid popup on Windows and don't generate a core dump on Linux.

Version 2.3 (2013-12-17)

- faulthandler.register() now keeps the previous signal handler when the function is called twice, so faulthandler.unregister() restores correctly the original signal handler.

Version 2.2 (2013-03-19)

- Rename dump_tracebacks_later() to dump_traceback_later(): use the same API than the faulthandler module of Python 3.3
- Fix handling of errno variable in the handler of user signals
- Fix the handler of user signals: chain the previous signal handler even if getting the current thread state failed

Version 2.1 (2012-02-05)

Major changes:

- Add an optional chain argument to faulthandler.register()

Minor changes:

- Fix faulthandler._sigsegv() for Clang 3.0
- Fix compilation on Visual Studio

Version 2.0 (2011-05-10)

Major changes:

- faulthandler is now part of Python 3.3
- enable() handles also the SIGABRT signal
- Add exit option to dump_traceback_later(): if True, exit the program on timeout after dumping the traceback

Other changes:

- Change default value of the all_threads argument: dump all threads by default because under some rare conditions, it is not possible to get the current thread
- Save/restore errno in signal handlers
- dump_traceback_later() always dump all threads: remove all_threads option
- Add faulthandler.__version__ attribute (module version as a string)
- faulthandler.version is now a tuple
- Rename:
  - dump_traceback_later() to dump_traceback_later()
  - cancel_dump_traceback_later() to cancel_dump_traceback_later()
  - sigsegv() to _sigsegv()
  - sigfpe() to _sigfpe()
  - sigbus() to _sigbus()
  - sigill() to _sigill()
- register() and unregister() are no more available on Windows. They were useless: only SIGSEGV, SIGABRT and SIGILL can be handled by the application, and these signals can only be handled by enable().
- Add _fatal_error(), _read_null(), _sigabrt() and _stack_overflow() test functions
- register() uses sigaction() SA_RESTART flag to try to not interrupt the current system call
- The fault handler calls the previous signal handler, using sigaction() SA_NODEFER flag to call it immediately
- enable() raises an OSError if it was not possible to register a signal handler
- Set module size to 0, instead of -1, to be able to unload the module with Python 3
- Fix a reference leak in dump_traceback_later()
- Fix register() if it called twice with the same signal
- Implement m_traverse for Python 3 to help the garbage collector
- Move code from faulthandler/*.c to faulthandler.c and traceback.c: the code is simpler and it was easier to integrate faulthandler into Python 3.3 using one file (traceback.c already existed in Python)
- register() uses a static list for all signals instead of reallocating memory each time a new signal is registered, because the list is shared with the signal handler which may be called anytime.

Version 1.5 (2011-03-24)

- Conform to the PEP 8:
  - Rename isenabled() to is_enabled()
  - Rename dumpbacktrace() to dump_traceback()
  - Rename dumpbacktrace_later() to dump_traceback_later()
  - Rename cancel_dumpbacktrace_later() to cancel_dump_traceback_later()
- Limit strings to 100 characters

- dump_traceback_later() signal handler doesn't clear its reference to the file, because Py_CLEAR() is not signal safe: you have to call explicitly cancel_dump_traceback_later()

Version 1.4 (2011-02-14)

- Add register() and unregister() functions
- Add optional all_threads argument to enable()
- Limit the backtrace to 100 threads
- Allocate an alternative stack for the fatal signal handler to be able to display a backtrace on a stack overflow (define HAVE_SIGALTSTACK). Not available on Windows.

Version 1.3 (2011-01-31)

- Don't compile dumpbacktrace_later() and cancel_dumpbacktrace_later() on Windows because alarm() is missing

Version 1.2 (2011-01-31)

- Add dumpbacktrace_later() and cancel_dumpbacktrace_later() function
- enable() and dumpbacktrace() get an optional file argument
- Replace dumpbacktrace_threads() function by a new dumpbacktrace() argument: dumpbacktrace(all_threads=True)
- enable() gets the file descriptor of sys.stderr instead of using the file descriptor 2

Version 1.1 (2011-01-03)

- Disable the handler by default, because pkgutil may load the module and so enable the handler which is unexpected
- Add dumpbacktrace() and dumpbacktrace_threads() functions
- sigill() is available on Windows thanks to Martin's patch
- Fix dump_ascii() for signed char type (eg. on FreeBSD)
- Fix tests.py for Python 2.5

Version 1.0 (2010-12-24)

First public release

CHAPTER 7

Similar projects

Python debuggers:

- minidumper is a C extension for writing "minidumps" for post-mortem analysis of crashes in Python or its extensions
- tipper: write the traceback of the current thread into a file on SIGUSR1 signal
- crier: write the traceback of the current thread into a file (eg. `/tmp/dump-<pid>`) if a "request" file is created (eg. `/tmp/crier-<pid>`). Implemented using a thread.
- Python WAD (Wrapped Application Debugger), not update since 2001:

Application fault handlers:

- The GNU libc has a fault handler in debug/segfault.c
- XEmacs has a fault handler displaying the Lisp traceback
- RPy has a fault handler

System-wide fault handlers:

- Ubuntu uses Apport
- Fedora has ABRT
- The Linux kernel logs also segfaults into /var/log/kern.log (and /var/log/syslog). /proc/sys/kernel/core_pattern contols how coredumps are created.
- Windows opens a popup on a fatal error asking if the error should be reported to Microsoft

CHAPTER 8

See also

- Python issue #8863 (may 2010): Display Python backtrace on SIGSEGV, SIGFPE and fatal error
- Python issue #3999 (sept. 2008): Real segmentation fault handler

Index