eventfd Documentation

Release 0.2

Aviv Palivoda

March 01, 2016

Contents

Module author: Aviv Palivoda <palaviv@gmail.com>

The python standard library `threading.Event` class provides a simple mechanisms for communication between threads: one thread signals an event and other threads wait for it. In many cases you would like to signal a thread that is currently waiting on a other events to happen using select/poll. The `EventFD` class provides a extension to the `threading.Event` class and can be used to stop the select/poll when signaled.

Note: EventFD support the windows operating system but it is not tested in the CI.

Note: EventFD use the linux eventfd but is not a python binding for eventfd. You might want to try:

- https://pypi.python.org/pypi/linuxfd/
- https://pypi.python.org/pypi/butter/

CHAPTER 1

Event Objects

The `EventFD` class is currently implemented with linux eventfd or `os.pipe()`. the `EventFD` class inherits from the *`eventfd._eventfd.BaseEventFD`* class.

class `eventfd._eventfd.`**`BaseEventFD`**

Class implementing event objects that has a fd that can be selected.

This EventFD class implements the same functions as a regular Event but it has a file descriptor. The file descriptor can be accessed using the fileno function. This event can be passed to select, poll and it will block until the event will be set.

`clear``()`

Reset the internal flag to false.

Subsequently, threads calling wait() will block until set() is called to set the internal flag to true again.

`fileno``()`

Return a file descriptor that can be selected.

You should not use this directly pass the EventFD object instead.

`is_set``()`

Return true if and only if the internal flag is true.

`set``()`

Set the internal flag to true.

All threads waiting for it to become true are awakened. Threads that call wait() once the flag is true will not block at all.

`wait``(`*timeout=None*`)`

Block until the internal flag is true.

If the internal flag is true on entry, return immediately. Otherwise, block until another thread calls set() to set the flag to true, or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof).

This method returns the internal flag on exit, so it will always return True except if a timeout is given and the operation times out.

CHAPTER 2

EXAMPLES

We will implement `socketserver.TCPServer` without polling using `EventFD`:

```
"""This is an example of python HTTP server without polling using the EventFD to shutdown the se

import threading
import select
from socketserver import TCPServer, BaseRequestHandler
import socket
import time

from eventfd import EventFD


class NonPollingHTTPServer(TCPServer):

    def __init__(self, server_address, RequestHandlerClass, bind_and_activate=True):
        self.server_address = server_address
        self.RequestHandlerClass = RequestHandlerClass
        self.__is_shut_down = threading.Event()
        # using an EventFD to signal the server_forever to stop
        self.__shutdown_event = EventFD()
        self.socket = socket.socket(self.address_family,
                                    self.socket_type)
        if bind_and_activate:
            try:
                self.server_bind()
                self.server_activate()
            except:
                self.server_close()
                raise

    # overriding the server_forever class to not poll.
    def serve_forever(self):
        self.__is_shut_down.clear()
        self.__shutdown_event.clear()
        try:
            while True:
                r, w, e = select.select([self.__shutdown_event, self], [], [])

                if self.__shutdown_event in r:
                    break

                if self in r:
                        self._handle_request_noblock()

                    self.service_actions()
            finally:
                self.__is_shut_down.set()

    def shutdown(self):
        self.__shutdown_event.set()
        self.__is_shut_down.wait()


###############################################
# MyTCPHandler class and client function are
# from the python socketserver documentation.
###############################################

class MyTCPHandler(BaseRequestHandler):

    def handle(self):
        # self.request is the TCP socket connected to the client
        self.data = self.request.recv(1024).strip()
        print("{} wrote:".format(self.client_address[0]))
        print(self.data)
        # just send back the same data, but upper-cased
        self.request.sendall(self.data.upper())


def client(ip, port, message):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    try:
        sock.sendall(message)
        response = sock.recv(1024)
        print("Received: {}".format(response))
    finally:
        sock.close()


def test():
    server = NonPollingHTTPServer(("localhost", 0), MyTCPHandler)
    ip, port = server.server_address

    server_thread = threading.Thread(target=server.serve_forever)
    server_thread.daemon = True
    server_thread.start()

    client(ip, port, b"Hello World 1")
    client(ip, port, b"Hello World 2")
    client(ip, port, b"Hello World 3")

    print("requesting server shutdown at {}".format(time.time()))
    server.shutdown()
    print("server was shutdown at {}".format(time.time()))
    server.server_close()


if __name__ == "__main__":
    test()
```

CHAPTER 3

Obtaining the Module

This module can be installed directly from the Python Package Index with pip:

```
pip install eventfd
```

Alternatively, you can download and unpack it manually from the eventfd PyPI page.

CHAPTER 4

Development and Support

eventfd is developed and maintained on Github.

Problems and suggested improvements can be posted to the issue tracker.

4.1 Release History

4.1.1 0.2 (01-03-2016)

- Using linux eventfd where eventfd is avaiable.
- Travis CI using tox.
- Support for windows using socket (only sockets can be selected in windows).

4.1.2 0.1 (27-02-2016)

- EventFD using pipe.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index