
timer file descriptor HOWTO

Release 3.13.0b1

Guido van Rossum and the Python development team

June 04, 2024

Python Software Foundation
Email: docs@python.org

Contents

1	Examples	1
----------	-----------------	----------

Release
1.13

This HOWTO discusses Python's support for the linux timer file descriptor.

1 Examples

The following example shows how to use a timer file descriptor to execute a function twice a second:

```
# Practical scripts should use really use a non-blocking timer,  
# we use a blocking timer here for simplicity.  
import os, time  
  
# Create the timer file descriptor  
fd = os.timerfd_create(time.CLOCK_REALTIME)  
  
# Start the timer in 1 second, with an interval of half a second  
os.timerfd_settime(fd, initial=1, interval=0.5)  
  
try:  
    # Process timer events four times.  
    for _ in range(4):  
        # read() will block until the timer expires  
        _ = os.read(fd, 8)  
        print("Timer expired")  
finally:  
    # Remember to close the timer file descriptor!  
    os.close(fd)
```

To avoid the precision loss caused by the `float` type, timer file descriptors allow specifying initial expiration and interval in integer nanoseconds with `_ns` variants of the functions.

This example shows how `epoll()` can be used with timer file descriptors to wait until the file descriptor is ready for reading:

```

import os, time, select, socket, sys

# Create an epoll object
ep = select.epoll()

# In this example, use loopback address to send "stop" command to the server.
#
# $ telnet 127.0.0.1 1234
# Trying 127.0.0.1...
# Connected to 127.0.0.1.
# Escape character is '^]'.
# stop
# Connection closed by foreign host.
#
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(("127.0.0.1", 1234))
sock.setblocking(False)
sock.listen(1)
ep.register(sock, select.EPOLLIN)

# Create timer file descriptors in non-blocking mode.
num = 3
fds = []
for _ in range(num):
    fd = os.timerfd_create(time.CLOCK_REALTIME, flags=os.TFD_NONBLOCK)
    fds.append(fd)
    # Register the timer file descriptor for read events
    ep.register(fd, select.EPOLLIN)

# Start the timer with os.timerfd_settime_ns() in nanoseconds.
# Timer 1 fires every 0.25 seconds; timer 2 every 0.5 seconds; etc
for i, fd in enumerate(fds, start=1):
    one_sec_in_nsec = 10**9
    i = i * one_sec_in_nsec
    os.timerfd_settime_ns(fd, initial=i//4, interval=i//4)

timeout = 3
try:
    conn = None
    is_active = True
    while is_active:
        # Wait for the timer to expire for 3 seconds.
        # epoll.poll() returns a list of (fd, event) pairs.
        # fd is a file descriptor.
        # sock and conn[=returned value of socket.accept()] are socket objects,
        ↪not file descriptors.
        # So use sock.fileno() and conn.fileno() to get the file descriptors.
        events = ep.poll(timeout)

        # If more than one timer file descriptors are ready for reading at once,
        # epoll.poll() returns a list of (fd, event) pairs.
        #
        # In this example settings,
        # 1st timer fires every 0.25 seconds in 0.25 seconds. (0.25, 0.5, 0.75,
        ↪1.0, ...)
        # 2nd timer every 0.5 seconds in 0.5 seconds. (0.5, 1.0, 1.5, 2.0, ...)
        # 3rd timer every 0.75 seconds in 0.75 seconds. (0.75, 1.5, 2.25, 3.0, .
        ↪..)
        #
        # In 0.25 seconds, only 1st timer fires.
        # In 0.5 seconds, 1st timer and 2nd timer fires at once.
        # In 0.75 seconds, 1st timer and 3rd timer fires at once.

```

(continues on next page)

(continued from previous page)

```
# In 1.5 seconds, 1st timer, 2nd timer and 3rd timer fires at once.
#
# If a timer file descriptor is signaled more than once since
# the last os.read() call, os.read() returns the number of signaled
# as host order of class bytes.
print(f"Signaled events={events}")
for fd, event in events:
    if event & select.EPOLLIN:
        if fd == sock.fileno():
            # Check if there is a connection request.
            print(f"Accepting connection {fd}")
            conn, addr = sock.accept()
            conn.setblocking(False)
            print(f"Accepted connection {conn} from {addr}")
            ep.register(conn, select.EPOLLIN)
        elif conn and fd == conn.fileno():
            # Check if there is data to read.
            print(f"Reading data {fd}")
            data = conn.recv(1024)
            if data:
                # You should catch UnicodeDecodeError exception for safety.
                cmd = data.decode()
                if cmd.startswith("stop"):
                    print(f"Stopping server")
                    is_active = False
                else:
                    print(f"Unknown command: {cmd}")
            else:
                # No more data, close connection
                print(f"Closing connection {fd}")
                ep.unregister(conn)
                conn.close()
                conn = None
        elif fd in fds:
            print(f"Reading timer {fd}")
            count = int.from_bytes(os.read(fd, 8), byteorder=sys.byteorder)
            print(f"Timer {fds.index(fd) + 1} expired {count} times")
        else:
            print(f"Unknown file descriptor {fd}")
finally:
    for fd in fds:
        ep.unregister(fd)
        os.close(fd)
    ep.close()
```

This example shows how `select()` can be used with timer file descriptors to wait until the file descriptor is ready for reading:

```
import os, time, select, socket, sys

# In this example, use loopback address to send "stop" command to the server.
#
# $ telnet 127.0.0.1 1234
# Trying 127.0.0.1...
# Connected to 127.0.0.1.
# Escape character is '^]'.
# stop
# Connection closed by foreign host.
#
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(("127.0.0.1", 1234))
```

(continues on next page)

```

sock.setblocking(False)
sock.listen(1)

# Create timer file descriptors in non-blocking mode.
num = 3
fds = [os.timerfd_create(time.CLOCK_REALTIME, flags=os.TFD_NONBLOCK)
        for _ in range(num)]
select_fds = fds + [sock]

# Start the timers with os.timerfd_settime() in seconds.
# Timer 1 fires every 0.25 seconds; timer 2 every 0.5 seconds; etc
for i, fd in enumerate(fds, start=1):
    os.timerfd_settime(fd, initial=i/4, interval=i/4)

timeout = 3
try:
    conn = None
    is_active = True
    while is_active:
        # Wait for the timer to expire for 3 seconds.
        # select.select() returns a list of file descriptors or objects.
        rfd, wfd, xfd = select.select(select_fds, select_fds, select_fds, timeout)
        for fd in rfd:
            if fd == sock:
                # Check if there is a connection request.
                print(f"Accepting connection {fd}")
                conn, addr = sock.accept()
                conn.setblocking(False)
                print(f"Accepted connection {conn} from {addr}")
                select_fds.append(conn)
            elif conn and fd == conn:
                # Check if there is data to read.
                print(f"Reading data {fd}")
                data = conn.recv(1024)
                if data:
                    # You should catch UnicodeDecodeError exception for safety.
                    cmd = data.decode()
                    if cmd.startswith("stop"):
                        print(f"Stopping server")
                        is_active = False
                    else:
                        print(f"Unknown command: {cmd}")
                else:
                    # No more data, close connection
                    print(f"Closing connection {fd}")
                    select_fds.remove(conn)
                    conn.close()
                    conn = None
            elif fd in fds:
                print(f"Reading timer {fd}")
                count = int.from_bytes(os.read(fd, 8), byteorder=sys.byteorder)
                print(f"Timer {fds.index(fd) + 1} expired {count} times")
            else:
                print(f"Unknown file descriptor {fd}")
finally:
    for fd in fds:
        os.close(fd)
    sock.close()
    sock = None

```