

---

# timer 文件描述符指南

发行版本 3.13.0rc2

Guido van Rossum and the Python development team

九月 19, 2024

Python Software Foundation  
Email: docs@python.org

## Contents

1 例子	1
------	---

---

发布版本  
1.13

本指南讨论了 Python 对 linux timer 文件描述符的支持。

## 1 例子

下面的例子演示了如何使用 timer 文件描述符每秒钟执行两次某个函数：

```
# 真正实用的脚本应当使用非阻塞型计时器，
# 这里我们使用阻塞型计时器是出于简单化考虑。
import os, time

# 创建计时器文件描述符
fd = os.timerfd_create(time.CLOCK_REALTIME)

# 在 1 秒种时启动计时器，间隔时间为半秒
os.timerfd_settime(fd, initial=1, interval=0.5)

try:
    # 处理计时器事件四次。
    for _ in range(4):
        # read() 将会阻塞直到计时器过期
        _ = os.read(fd, 8)
        print("Timer expired")
finally:
    # 记住要关闭计时器文件描述符！
    os.close(fd)
```

为避免 float 类型导致的精度损失，timer 文件描述符允许使用这些函数的 `_ns` 变种形式以整数纳秒值指定初始到期时间和间隔。

这个例子演示了如何使用 `epoll()` 配合 timer 文件描述符来执行等待直到文件描述符准备好读取：

```

import os, time, select, socket, sys

# 创建一个轮询对象
ep = select.epoll()

# 在本例中，使用回环地址向服务器发送 "stop" 命令。
#
# $ 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)

# 以非阻塞模式创建定时器文件描述符。
num = 3
fds = []
for _ in range(num):
    fd = os.timerfd_create(time.CLOCK_REALTIME, flags=os.TFD_NONBLOCK)
    fds.append(fd)
    # 注册定时器文件描述符用于读取事件
    ep.register(fd, select.EPOLLIN)

# 以纳秒精度的 os.timerfd_settime_ns() 启动定时器。
# 定时器 1 间隔为 0.25 秒；定时器 2 间隔为 0.5 秒；依此类推
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:
        # 等待定时器 3 秒到期。
        # epoll.poll() 返回一个 (fd, event) 对的列表。
        # fd 是一个文件描述符。
        # sock 和 conn=[socket.accept() 的返回值] 是套接字对象，而不是文件描述符。
        # 因此使用 sock.fileno() 和 conn.fileno() 来获取文件描述符。
        events = ep.poll(timeout)

        # 如果同时有多个定时器文件描述符准备读取，
        # epoll.poll() 将返回一个 (fd, event) 对的列表。
        #
        # 在本例的设置中，
        # 第 1 个定时器在 0.25 秒后间隔 0.25 秒启动。 (0.25, 0.5, 0.75, 1.0, ...)
        #
        # 第 2 个定时器在 0.5 秒后间隔 0.5 秒启动。 (0.5, 1.0, 1.5, 2.0, ...)
        # 第 3 个定时器在 0.75 秒后间隔 0.75 秒启动。 (0.75, 1.5, 2.25, 3.0, ...)
        #
        # 在 0.25 秒时，只有第 1 个定时器启动。
        # 在 0.5 秒时，第 1 个定时器和第 2 个定时器同时启动。
        # 在 0.75 秒时，第 1 个定时器和第 3 个定时器同时启动。
        # 在 1.5 秒时，第 1 个定时器、第 2 个定时器和第 3 个定时器同时启动。

```

(续下页)

```

#
# 如果一个定时器文件描述符自上次 os.read() 调用后
# 多次发出信号, os.read() 将以主机的类字节顺序
# 返回发出信号的次数。
print(f"Signaled events={events}")
for fd, event in events:
    if event & select.EPOLLIN:
        if fd == sock.fileno():
            # 检查是否有连接请求。
            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():
            # 检查是否有数据要读取。
            print(f"Reading data {fd}")
            data = conn.recv(1024)
            if data:
                # 安全起见你应当捕获 UnicodeDecodeError 异常。
                cmd = data.decode()
                if cmd.startswith("stop"):
                    print(f"Stopping server")
                    is_active = False
                else:
                    print(f"Unknown command: {cmd}")
            else:
                # 已无数据, 关闭连接
                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()

```

这个例子演示了如何使用 select() 配合 timer 文件描述符来执行等待直接文件描述符准备好读取:

```

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))
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

```