5.18 将文件描述符包装成文件对象¶

问题¶

你有一个对应于操作系统上一个已打开的I/O通道(比如文件、管道、套接字等)的整型文件描述符， 你想将它包装成一个更高层的Python文件对象。

解决方案¶

一个文件描述符和一个打开的普通文件是不一样的。 文件描述符仅仅是一个由操作系统指定的整数，用来指代某个系统的I/O通道。 如果你碰巧有这么一个文件描述符，你可以通过使用 `open()` 函数来将其包装为一个Python的文件对象。 你仅仅只需要使用这个整数值的文件描述符作为第一个参数来代替文件名即可。例如：

```
# Open a low-level file descriptor
import os
fd = os.open('somefile.txt', os.O_WRONLY | os.O_CREAT)

# Turn into a proper file
f = open(fd, 'wt')
f.write('hello world\n')
f.close()
```

当高层的文件对象被关闭或者破坏的时候，底层的文件描述符也会被关闭。 如果这个并不是你想要的结果，你可以给 `open()` 函数传递一个可选的 `colsefd=False` 。比如：

```
# Create a file object, but don't close underlying fd when done
f = open(fd, 'wt', closefd=False)
...
```

讨论¶

在Unix系统中，这种包装文件描述符的技术可以很方便的将一个类文件接口作用于一个以不同方式打开的I/O通道上，如管道、套接字等。举例来讲，下面是一个操作管道的例子：

```
from socket import socket, AF_INET, SOCK_STREAM

def echo_client(client_sock, addr):
    print('Got connection from', addr)

    # Make text-mode file wrappers for socket reading/writing
    client_in = open(client_sock.fileno(), 'rt', encoding='latin-1',
                     closefd=False)

    client_out = open(client_sock.fileno(), 'wt', encoding='latin-1',
                      closefd=False)

    # Echo lines back to the client using file I/O
    for line in client_in:
        client_out.write(line)
        client_out.flush()

    client_sock.close()
```

```
def echo_server(address):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(1)
    while True:
        client, addr = sock.accept()
        echo_client(client, addr)
```

需要重点强调的一点是，上面的例子仅仅是为了演示内置的 `open()` 函数的一个特性，并且也只适用于基于Unix的系统。 如果你想将一个类文件接口作用在一个套接字并希望你的代码可以跨平台，请使用套接字对象的 `makefile()` 方法。但是如果不考虑可移植性的话，那上面的解决方案会比使用 `makefile()` 性能更好一点。

你也可以使用这种技术来构造一个别名，允许以不同于第一次打开文件的方式使用它。 例如，下面演示如何创建一个文件对象，它允许你输出二进制数据到标准输出(通常以文本模式打开)：

```
import sys
# Create a binary-mode file for stdout
bstdout = open(sys.stdout.fileno(), 'wb', closefd=False)
bstdout.write(b'Hello World\n')
bstdout.flush()
```

尽管可以将一个已存在的文件描述符包装成一个正常的文件对象， 但是要注意的是并不是所有的文件模式都被支持，并且某些类型的文件描述符可能会有副作用 (特别是涉及到错误处理、文件结尾条件等等的时候)。 在不同的操作系统上这种行为也是不一样，特别的，上面的例子都不能在非Unix系统上运行。 我说了这么多，意思就是让你充分测试自己的实现代码，确保它能按照期望工作。