11.11 进程间传递Socket文件描述符¶

问题¶

你有多个Python解释器进程在同时运行，你想将某个打开的文件描述符从一个解释器传递给另外一个。 比如，假设有个服务器进程相应连接请求，但是实际的相应逻辑是在另一个解释器中执行的。

解决方案¶

为了在多个进程中传递文件描述符，你首先需要将它们连接到一起。在Unix机器上，你可能需要使用Unix域套接字， 而在windows上面你需要使用命名管道。不过你无需真的需要去操作这些底层， 通常使用 `multiprocessing` 模块来创建这样的连接会更容易一些。

一旦一个连接被创建，你可以使用 `multiprocessing.reduction` 中的 `send_handle()` 和 `recv_handle()` 函数在不同的处理器直接传递文件描述符。 下面的例子演示了最基本的用法：

```
import multiprocessing
from multiprocessing.reduction import recv_handle, send_handle
import socket

def worker(in_p, out_p):
    out_p.close()
    while True:
        fd = recv_handle(in_p)
        print('CHILD: GOT FD', fd)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd) as s:
            while True:
                msg = s.recv(1024)
                if not msg:
                    break
                print('CHILD: RECV {!r}'.format(msg))
                s.send(msg)

def server(address, in_p, out_p, worker_pid):
    in_p.close()
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(address)
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)
        send_handle(out_p, client.fileno(), worker_pid)
        client.close()

if __name__ == '__main__':
    c1, c2 = multiprocessing.Pipe()
    worker_p = multiprocessing.Process(target=worker, args=(c1,c2))
    worker_p.start()

    server_p = multiprocessing.Process(target=server,
                args=(('', 15000), c1, c2, worker_p.pid))
    server_p.start()
```

```
    c1.close()
    c2.close()
```

在这个例子中，两个进程被创建并通过一个 `multiprocessing` 管道连接起来。 服务器进程打开一个socket并等待客户端连接请求。 工作进程仅仅使用 `recv_handle()` 在管道上面等待接收一个文件描述符。 当服务器接收到一个连接，它将产生的socket文件描述符通过 `send_handle()` 传递给工作进程。 工作进程接收到socket后向客户端回应数据，然后此次连接关闭。

如果你使用Telnet或类似工具连接到服务器，下面是一个演示例子：

bash % python3 passfd.py SERVER: Got connection from ('127.0.0.1', 55543) CHILD: GOT FD 7 CHILD: RECV b'Hellorn' CHILD: RECV b'Worldrn'
此例最重要的部分是服务器接收到的客户端socket实际上被另外一个不同的进程处理。 服务器仅仅只是将其转手并关闭此连接，然后等待下一个连接。

讨论¶

对于大部分程序员来讲在不同进程之间传递文件描述符好像没什么必要。 但是，有时候它是构建一个可扩展系统的很有用的工具。例如，在一个多核机器上面， 你可以有多个Python解释器实例，将文件描述符传递给其它解释器来实现负载均衡。

`send_handle()` 和 `recv_handle()` 函数只能够用于 `multiprocessing` 连接。 使用它们来代替管道的使用（参考11.7节），只要你使用的是Unix域套接字或Windows管道。 例如，你可以让服务器和工作者各自以单独的程序来启动。下面是服务器的实现例子：

```
# servermp.py
from multiprocessing.connection import Listener
from multiprocessing.reduction import send_handle
import socket

def server(work_address, port):
    # Wait for the worker to connect
    work_serv = Listener(work_address, authkey=b'peekaboo')
    worker = work_serv.accept()
    worker_pid = worker.recv()

    # Nowrun a TCP/IP server and send clients to worker
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(('', port))
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)

        send_handle(worker, client.fileno(), worker_pid)
        client.close()

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
        print('Usage: server.py server_address port', file=sys.stderr)
        raise SystemExit(1)
```

```
    server(sys.argv[1], int(sys.argv[2]))
```

运行这个服务器，只需要执行 *python3 servermp.py /tmp/servconn 15000* ，下面是相应的工作者代码：

```
# workermp.py

from multiprocessing.connection import Client
from multiprocessing.reduction import recv_handle
import os
from socket import socket, AF_INET, SOCK_STREAM

def worker(server_address):
    serv = Client(server_address, authkey=b'peekaboo')
    serv.send(os.getpid())
    while True:
        fd = recv_handle(serv)
        print('WORKER: GOT FD', fd)
        with socket(AF_INET, SOCK_STREAM, fileno=fd) as client:
            while True:
                msg = client.recv(1024)
                if not msg:
                    break
                print('WORKER: RECV {!r}'.format(msg))
                client.send(msg)

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print('Usage: worker.py server_address', file=sys.stderr)
        raise SystemExit(1)

    worker(sys.argv[1])
```

要运行工作者，执行执行命令 *python3 workermp.py /tmp/servconn* . 效果跟使用Pipe()例子是完全一样的。 文件描述符的传递会涉及到UNIX域套接字的创建和套接字的 `sendmsg()` 方法。 不过这种技术并不常见，下面是使用套接字来传递描述符的另外一种实现：

```
# server.py
import socket

import struct

def send_fd(sock, fd):
    '''
    Send a single file descriptor.
    '''
    sock.sendmsg([b'x'],
                 [(socket.SOL_SOCKET, socket.SCM_RIGHTS, struct.pack('i', fd))])
    ack = sock.recv(2)
    assert ack == b'OK'

def server(work_address, port):
    # Wait for the worker to connect
    work_serv = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
    work_serv.bind(work_address)
    work_serv.listen(1)
    worker, addr = work_serv.accept()
```

```
    # Nowrun a TCP/IP server and send clients to worker
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(('',port))
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)
        send_fd(worker, client.fileno())
        client.close()

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
        print('Usage: server.py server_address port', file=sys.stderr)
        raise SystemExit(1)

    server(sys.argv[1], int(sys.argv[2]))
```

下面是使用套接字的工作者实现：

```
# worker.py
import socket
import struct

def recv_fd(sock):
    '''
    Receive a single file descriptor
    '''
    msg, ancdata, flags, addr = sock.recvmsg(1,
                                socket.CMSG_LEN(struct.calcsize('i')))

    cmsg_level, cmsg_type, cmsg_data = ancdata[0]
    assert cmsg_level == socket.SOL_SOCKET and cmsg_type == socket.SCM_RIGHTS
    sock.sendall(b'OK')

    return struct.unpack('i', cmsg_data)[0]

def worker(server_address):
    serv = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
    serv.connect(server_address)
    while True:
        fd = recv_fd(serv)
        print('WORKER: GOT FD', fd)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd) as client:
            while True:
                msg = client.recv(1024)
                if not msg:
                    break
                print('WORKER: RECV {!r}'.format(msg))
                client.send(msg)

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print('Usage: worker.py server_address', file=sys.stderr)
        raise SystemExit(1)
```

```
    worker(sys.argv[1])
```

如果你想在你的程序中传递文件描述符，建议你参阅其他一些更加高级的文档， 比如 `Unix Network Programming by W. Richard Stevens (Prentice Hall, 1990)` . 在Windows上传递文件描述符跟Unix是不一样的，建议你研究下 `multiprocessing.reduction` 中的源代码看看其工作原理。