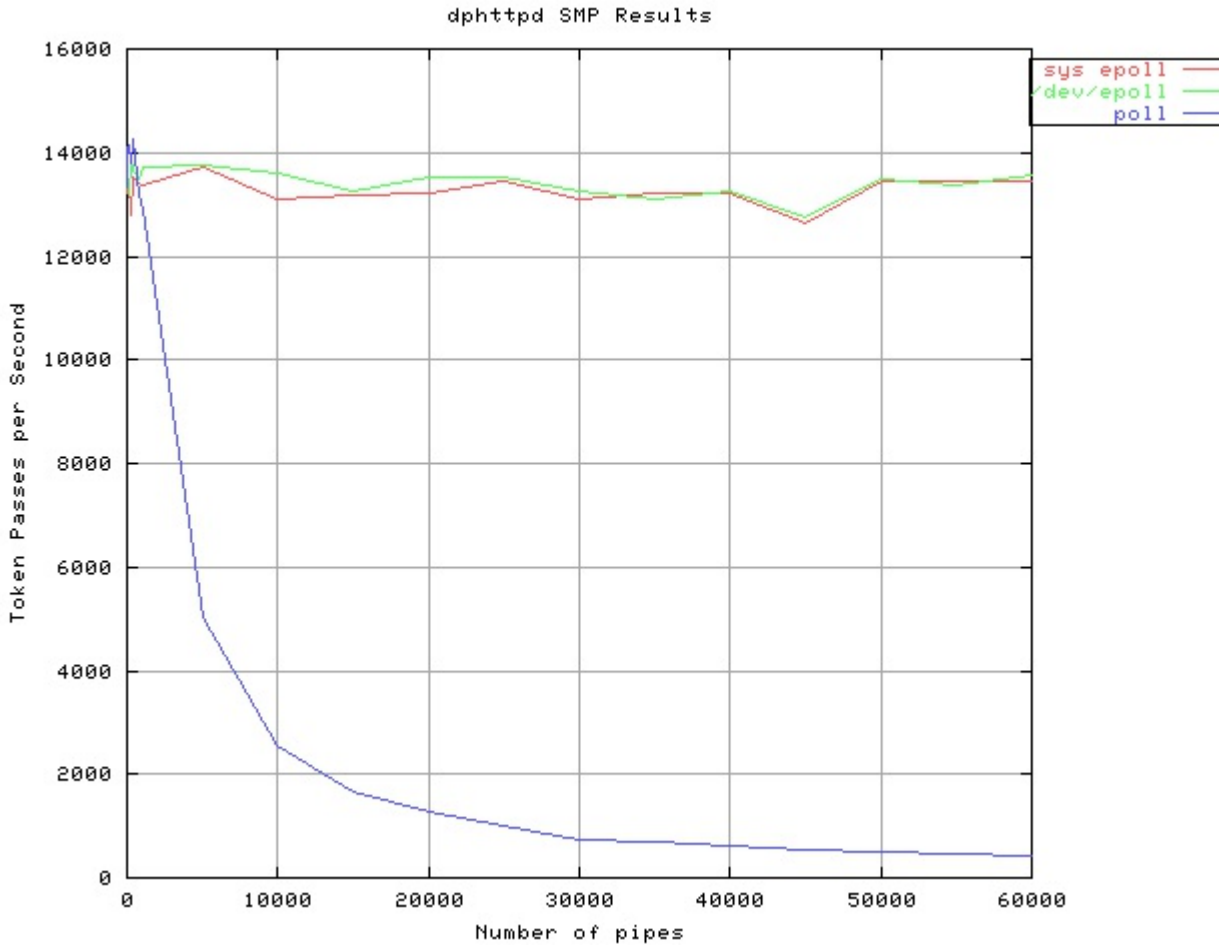
# 寻找自我的博客

## python网络编程-epoll

分类： Python 2012-08-23 20:16 310人阅读 评论(0) 收藏 举报

英文原文： http://scotdoyle.com/python-epoll-howto.html




epoll和poll比较

介绍
从2.6版本开始，python 提供了使用linux epoll 的功能. 这篇文章通过3个例子来大致介绍如何使用它. 欢迎提问和反馈.

阻塞式socket通讯
第一个例子是一个简单的python3.0版本的服务器代码，监听8080端口的http请求，打印结果到命令行，回应http response给客户端.

行 9: 建立服务器的socket
行 10: 允许11行的bind()操作，即使其他程序也在监听同样的端口. 不然的话，这个程序只能在其他程序停止使用这个端口之后的1到2分钟后才能执行.
行 11: 绑定socket到这台机器上所有IPv4地址上的8080端口.
行 12: 告诉服务器开始响应从客户端过来的连接请求.
行 14: 程序会一直停在这里，直到建立了一个连接. 这个时候，服务器socket会建立一个新的socket，用来和客户端通讯. 这个新的socket是accept()的返回值，address对象标示了客户端的IP地址和端口.
行 15-17: 接收数据，直到一个完整的http请求被接收完毕. 这是一个简单的http服务器实现.
行 18: 为了方便验证，打印客户端过来的请求到命令行.
行 19: 发送回应.
行 20-22: 关闭连接，以及服务器的监听socket.

例1

```
import socket

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)

connectiontoclient, address = serversocket.accept()
request = b''
while EOL1 not in request and EOL2 not in request:
            request += connectiontoclient.recv(1024)
print(request.decode())
connectiontoclient.send(response)
connectiontoclient.close()

serversocket.close()
```

例2：

我们在15行加上了一个循环，用来循环处理客户端请求，直到我们中断这个过程(在命令行下面输入键盘中断，比如Ctrl-C)．这个例子更明显地表示出来了，服务器socket并没有用来做数据处理，而是接受服务器过来的连接，然后建立一个新的socket，用来和客户端通讯.
最后的23-24行确保服务器的监听socket最后总是close掉，即使出现了异常.

```
import socket

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)

try:
            while True:
                        connectiontoclient, address = serversocket.accept()
                        request = b''
                        while EOL1 not in request and EOL2 not in request:
                            request += connectiontoclient.recv(1024)
                        print('-'*40 + '\n' + request.decode()[:-2])
                        connectiontoclient.send(response)
                        connectiontoclient.close()
finally:
            serversocket.close()
```

异步socket和linux epoll的优势
第2个例子里面的socket采用的是阻塞方式，因为python解释器在出现事件之前都处在停止状态．16行的accept()一直阻塞，直到新的连接进来．19行的recv()也是一直阻塞，直到从客户端收到数据(或者直到没有数据可以接收).

21行的send()也一直阻塞，直到所有需要发送给客户端的数据都交给了linux内核的发送队列.

当一个程序采用阻塞socket的时候，它经常采用一个线程(甚至一个进程)一个socket通讯的模式. 主线程保留服务器监听socket，接受进来的连接，一次接受一个连接，然后把生成的socket交给一个分离的线程去做交互. 因为一个线程只和一个客户端通讯，在任何位置的阻塞都不会造成问题. 阻塞本身不会影响其他线程的工作.
多线程阻塞socket模式代码清晰，但是有几个缺陷，可能很难确保线程间资源共享工作正常，可能在只有一个CPU的机器上效率低下.
C10K(单机1万连接问题!) 探讨了其他处理并行socket通讯的模式. 一种是采用异步socket. socket不会阻塞，直到特定事件发生. 程序在异步socket上面进行一个特定操作，并且立即得到一个结果，不管执行成功或者失败. 然后让程序决定下一步怎么做. 因为异步socket是非阻塞的，我们可以不采用多线程. 所有的事情都可以在一个线程里面完成. 虽然这种模式有它需要面对的问题，它对于特定程序来说还是不错的选择. 也可以和多线程合起来使用: 单线程的异步socket可以当作服务器上面处理网络的一个模块，而线程可以用来访问阻塞式的资源，比如数据库.

Linux 2.6有一些方式来管理异步socket，python API能够用的有3种: select，poll和epoll. epoll和poll比select性能更好，因为python程序不需要为了特定的事件去查询单独的socket，而是依赖操作系统来告诉你什么socket产生了什么事件. epoll比poll性能更好，因为它不需要每次python程序查询的时候，操作系统都去检查所有的socket，在事件产生的时候，linux跟踪他们，然后在python程序调用的时候，返回具体的列表. 所以epoll在大量(上千)并行连接下，是一种更有效率，伸缩性更强的机制.
采用epoll的异步socket编程示例
采用epoll的程序一般这样操作:

建立一个epoll对象
告诉epoll对象，对于一些socket监控一些事件.
问epoll，从上次查询以来什么socket产生了什么事件.
针对这些socket做特定操作.
告诉epoll，修改监控socket和/或监控事件.
重复第3步到第5步，直到结束.
销毁epoll对象.
采用异步socket的时候第3步重复了第2步的事情. 这里的程序更复杂，因为一个线程需要和多个客户端交互.

行 1: select模块带有epoll功能
行 13: 因为socket默认是阻塞的，我们需要设置成非阻塞(异步)模式.
行 15: 建立一个epoll对象.
行 16: 注册服务器socket，监听读取事件. 服务器socket接收一个连接的时候，产生一个读取事件.
行 19: connections表映射文件描述符(file descriptors，整型)到对应的网络连接对象上面.
行 21: epoll对象查询一下是否有感兴趣的事件发生，参数1说明我们最多等待1秒的时间. 如果有对应事件发生，立刻会返回一个事件列表.
行 22: 返回的events是一个(fileno，event code)tuple列表. fileno是文件描述符，是一个整型数.
行 23: 如果是服务器socket的事件，那么需要针对新的连接建立一个socket.
行 25: 设置socket为非阻塞模式.
行 26: 注册socket的read(EPOLLIN)事件.
行 31: 如果读取事件发生，从客户端读取新数据.
行 33: 一旦完整的http请求接收到，取消注册读取事件，注册写入事件(EPOLLOUT)，写入事件在能够发送数据回客户端的时候产生.
行 34: 打印完整的http请求，展示即使通讯是交错的，数据本身是作为一个完整的信息组合和处理的.
行 35: 如果写入事件发生在一个客户端socket上面，我们就可以发送新数据到客户端了.
行s 36-38: 一次发送一部分返回数据，直到所有数据都交给操作系统的发送队列.
行 39: 一旦所有的返回数据都发送完，取消监听读取和写入事件.
行 40: 如果连接被明确关闭掉，这一步是可选的. 这个例子采用这个方法是为了让客户端首先断开，告诉客户端没有数据需要发送和接收了，然后让客户端断开连接.
行 41: HUP(hang-up)事件表示客户端断开了连接(比如 closed)，所以服务器这端也会断开. 不需要注册HUP事件，因为它们都会标示到注册在epoll的socket.
行 42: 取消注册.
行 43: 断开连接.
行s 18-45: 在这里的异常捕捉的作用是，我们的例子总是采用键盘中断来停止程序执行.
行s 46-48: 虽然开启的socket不需要手动关闭，程序退出的时候会自动关闭，明确写出来这样的代码，是更好的编码风格.

例3：

```
import socket, select

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
```

```
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)
serversocket.setblocking(0)

epoll = select.epoll()
epoll.register(serversocket.fileno(), select.EPOLLIN)

try:
        connections = {}; requests = {}; responses = {}
        while True:
                events = epoll.poll(1)
                for fileno, event in events:
                        if fileno == serversocket.fileno():
                                connection, address = serversocket.accept(
                                connection.setblocking(0)
                                epoll.register(connection.fileno(), select
                                connections[connection.fileno()] = connect
                                requests[connection.fileno()] = b''
                                responses[connection.fileno()] = response
                        elif event & select.EPOLLIN:
                                requests[fileno] += connections[fileno].re
                                if EOL1 in requests[fileno] or EOL2 in rec
                                        epoll.modify(fileno, select.EP
                                        print('-'*40 + '\n' + requests
                        elif event & select.EPOLLOUT:
                                byteswritten = connections[fileno].send(re
                                responses[fileno] = responses[fileno][byte
                                if len(responses[fileno]) == 0:
                                        epoll.modify(fileno, 0)
                                        connections[fileno].shutdown(s
                        elif event & select.EPOLLHUP:
                                epoll.unregister(fileno)
                                connections[fileno].close()
                                del connections[fileno]
finally:
        epoll.unregister(serversocket.fileno())
        epoll.close()
        serversocket.close()
```

epoll有2种模式，边沿触发(edge-triggered)和状态触发(level-triggered)．边沿触发模式下，epoll.poll()在读取/写入事件发生的时候只返回一次，程序必须在后续调用epoll.poll()之前处理完对应事件的所有的数据．当从一个事件中获取的数据被用完了，更多在socket上的处理会产生异常．相反，在状态触发模式下面，重复调用epoll.poll()只会返回重复的事件，直到所有对应的数据都处理完成．一般情况下不产生异常．

比如，一个服务器socket注册了读取事件，边沿触发程序需要调用accept建立新的socket连接直到一个socket.error错误产生，然后状态触发下只需要处理一个单独的accept()，然后继续epoll查询新的事件来判断是否有新的accept需要操作．

例子3采用默认的状态触发模式，例子4展示如何用边沿触发模式．例子4中的25，36和45行引入了循环，直到错误产生(或者所有的数据都处理完了)，32，38 和48行捕捉socket异常．最后16，28，41 和51行添加EPOLLET mask用来设置边沿触发．

例4：

```
import socket, select

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)
serversocket.setblocking(0)

epoll = select.epoll()
epoll.register(serversocket.fileno(), select.EPOLLIN | select.EPOLLET)

try:
            connections = {}; requests = {}; responses = {}
            while True:
                        events = epoll.poll(1)
                        for fileno, event in events:
                                    if fileno == serversocket.fileno():
                                                try:
                                                            while True:
                                                                        connection, address
                                                                        connection.setblock
                                                                        epoll.register(conn
                                                                        connections[connect
                                                                        requests[connection
                                                                        responses[connectio
                                                except socket.error:
                                                            pass
                                    elif event & select.EPOLLIN:
                                                try:
                                                            while True:
                                                                        requests[fileno] +=
                                                except socket.error:
                                                            pass
                                                if EOL1 in requests[fileno] or EOL2 in req
                                                            epoll.modify(fileno, select.EP
                                                            print('-'*40 + '\n' + requests
                                    elif event & select.EPOLLOUT:
                                                try:
                                                            while len(responses[fileno]) >
                                                                        byteswritten = conn
                                                                        responses[fileno] =
                                                except socket.error:
                                                            pass
                                                if len(responses[fileno]) == 0:
                                                            epoll.modify(fileno, select.EP
                                                            connections[fileno].shutdown(s
                                    elif event & select.EPOLLHUP:
                                                epoll.unregister(fileno)
                                                connections[fileno].close()
                                                del connections[fileno]
finally:
            epoll.unregister(serversocket.fileno())
            epoll.close()
            serversocket.close()
```

因为比较类似，状态触发经常用在转换采用select/poll模式的程序上面，边沿触发用在程序员不需要或者不希望操作系统来管理事件状态的场合上面.

除了这两种模式以外，socket经常注册为EPOLLONESHOT event mask，当用到这个选项的时候，事件只有效一次，之后会自动从监控的注册列表中移除.

性能考虑
Listen Backlog Queue Size

在1-4的例子中，12行显示了调用serversocket.listen()方法. 参数是监听等待队列的大小. 它告诉了操作系统，在python代码accept前，缓存多少TCP/IP连接在队列中. 每次python代码调用accept()的时候，一个连接从队列中移除，为新的连接进来空出一个位置. 如果队列满了，新的连接自动放弃，给客户端带来不必要的网络延迟. 一个生产环境下的服务器经常处理几十或者几百的同时连接数，所以参数不应该设置为1. 比如，当采用 ab 来对这些测试程序进行并发100个http1.0客户端请求时，少于50的参数容易造成性能下降.
TCP Options

TCP_CORK 参数可以设置缓存消息直到一起被发送，这个选项，在例子5的34和40行，适合给一个实现 http/1.1 pipelining 的服务器来使用.

例5：

```
import socket, select

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)
serversocket.setblocking(0)

epoll = select.epoll()
epoll.register(serversocket.fileno(), select.EPOLLIN)

try:
            connections = {}; requests = {}; responses = {}
            while True:
                        events = epoll.poll(1)
                        for fileno, event in events:
                                    if fileno == serversocket.fileno():
                                                connection, address = serversocket.accept(
                                                connection.setblocking(0)
                                                epoll.register(connection.fileno(), select
                                                connections[connection.fileno()] = connect
                                                requests[connection.fileno()] = b''
                                                responses[connection.fileno()] = response
                                    elif event & select.EPOLLIN:
                                                requests[fileno] += connections[fileno].re
                                                if EOL1 in requests[fileno] or EOL2 in rec
                                                            epoll.modify(fileno, select.EP
                                                            connections[fileno].setsockopt
                                                            print('-'*40 + '\n' + requests
                                    elif event & select.EPOLLOUT:
                                                byteswritten = connections[fileno].send(re
                                                responses[fileno] = responses[fileno][byte
                                                if len(responses[fileno]) == 0:
```

```
                                                connections[fileno].setsockopt
                                                epoll.modify(fileno, 0)
                                                connections[fileno].shutdown(s
                                    elif event & select.EPOLLHUP:
                                                epoll.unregister(fileno)
                                                connections[fileno].close()
                                                del connections[fileno]
finally:
            epoll.unregister(serversocket.fileno())
            epoll.close()
            serversocket.close()
```

另一方面，TCP_NODELAY 可以用来告诉操作系统，任何发给socket.send()的数据必须不经过操作系统的缓存，立刻发送给客户端.

这个选项，在第6个例子的14行，可以给SSH客户端或者其他实时性要求比较高的应用来使用.

例6：

```
import socket, select

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
response  = b'HTTP/1.0 200 OK\r\nDate: Mon, 1 Jan 1996 01:01:01 GMT\r\n'
response += b'Content-Type: text/plain\r\nContent-Length: 13\r\n\r\n'
response += b'Hello, world!'

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(('0.0.0.0', 8080))
serversocket.listen(1)
serversocket.setblocking(0)
serversocket.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)

epoll = select.epoll()
epoll.register(serversocket.fileno(), select.EPOLLIN)

try:
            connections = {}; requests = {}; responses = {}
            while True:
                        events = epoll.poll(1)
                        for fileno, event in events:
                                    if fileno == serversocket.fileno():
                                                connection, address = serversocket.accept(
                                                connection.setblocking(0)
                                                epoll.register(connection.fileno(), select
                                                connections[connection.fileno()] = connect
                                                requests[connection.fileno()] = b''
                                                responses[connection.fileno()] = response
                                    elif event & select.EPOLLIN:
                                                requests[fileno] += connections[fileno].re
                                                if EOL1 in requests[fileno] or EOL2 in req
                                                            epoll.modify(fileno, select.EP
                                                            print('-'*40 + '\n' + requests
                                    elif event & select.EPOLLOUT:
                                                byteswritten = connections[fileno].send(re
                                                responses[fileno] = responses[fileno][byte
                                                if len(responses[fileno]) == 0:
                                                            epoll.modify(fileno, 0)
                                                            connections[fileno].shutdown(s
                                    elif event & select.EPOLLHUP:
                                                epoll.unregister(fileno)
                                                connections[fileno].close()
```

```
                                            del connections[fileno]
finally:
            epoll.unregister(serversocket.fileno())
            epoll.close()
            serversocket.close()
```

源码下载：http://scotdoyle.com/python-epoll-examples.tar.gz