课程设计报告

1. 系统的功能设计

系统基于DNS协议实现了一个DNS中继服务器，并在性能方面采用了非阻塞IO，多线程，连接池等提高效率的方法。

* 1. 接收DNS报文

接收一个DNS的报文，如果是非DNS协议规定的包会不予处理，接受报文时采用非阻塞式IO，以减少CPU在等待数据包和复制数据包时所花的时间。处理DNS报文时，采用多线程，为每一个DNS报文建立一个线程。

* 1. 解析DNS报文

主要解析收到的请求包和回复包。采用BytesIO实现字节流的读写，通过struct来实现报文字段的unpack，拆解后的字节流以Message类存储

* 1. 封装DNS报文

封装主要封装的DNS回复包，当查询的请求在本地数据库中时，要自己构造DNS响应包。报文采用BytesIO来实现读写，用struct的pack来连接字段。

* 1. 转发DNS报文

当遇到无法处理的DNS报文以及遇到在本地数据库查询不到的资源记录时，会转发请求包到别的DNS服务器，其中会遇到消息ID的转换。

当在响应包中遇到无须缓存的资源记录时，会直接把响应包转发给查询方。

* 1. 存储资源记录

本地在运行过程中会缓存一些的资源记录到数据库中，以便下次有相同请求到来时，直接构造回复包，不用再转发给其他DNS服务器

* 1. 超时处理

如果一个请求包在3s之内未收到答复，那么在之后收到该回复的响应包会被舍弃，也不会将包中资源记录存储在数据库中

* 1. 数据库管理

数据库负责管理资源记录，数据库连接采用连接池技术，允许不同线程调用连接池获得数据库连接，数据库支持四种资源记录：A，NS，CNAME，MX

* 1. 输出调试信息

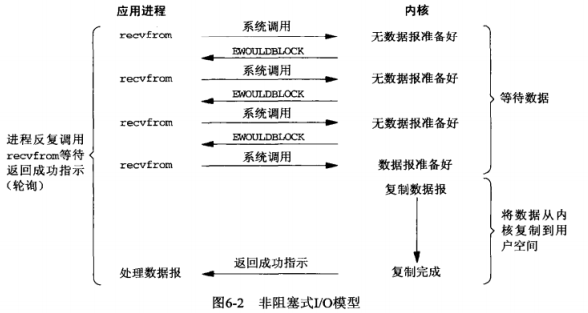
输出服务器在运行过程中的各种调试信息，调试信息按照级别分为两级。第一级调试信息会输出时间坐标，报文ID，客户端IP地址，查询域名等。第二级调试信息会在第一级的基础上输出更冗长的调试记录，如当前线程ID，消息转换元组，数据库增删改查信息等。

二、模块划分

2.1 网络I/O的设计与选择

采用python原生的socket,使用定期轮询的方式处理请求，请求到来时为每个请求开设一个线程处理。

2.1.1 非阻塞式IO设计



如图为非阻塞式IO的模型，本程序的轮询时间为0.5s每次，每隔0.5s会去查询数据报是否准备完毕，如果准备完毕就开始数据包操作。

*with* \_ServerSelector() *as* selector:

selector.register(self, selectors.EVENT\_READ)

*while* not self.\_\_isShutDownRequest:

ready = selector.select(pollInterval)

*if* ready:

self.\_handleRequestNoblock()

2.1.2 多线程设计

每读到一个数据包就为该数据包的处理单开一个线程去处理，这样就可以做到从微观上多个请求的并发处理，而从宏观上的多请求并发处理不需要采用多线程。

t = threading.Thread(*target* = self.processRequestThread,

*args* = (request, clientAddress))

t.daemon = self.daemonThreads

*if* not t.daemon and self.\_blockOnClose:

*if* self.\_threads is None:

self.\_threads = []

self.\_threads.append(t)

t.start()

2.2 数据库的设计与选择

数据库采用sqlite3,工作在并行模式,采用连接池的方式,初始化指定数目的连接供线程处理调用。

2.2.1 数据库连接池

数据库连接池调用DBUtiles中的PooledDB。PooledDB 可以在不同线程之间共享打开的数据库连接。当一个线程关闭了一个非共享连接，则会返还到空闲连接池中等待下次使用。

本程序初始化10个数据库连接

database = DNSDataBase(*mincached*=2, *maxcached*=5, *maxconnections*=10,

*database*='DNSDataBase.db')

其中各个线程之间不允许共享连接，也就是两个线程不能同时获得一个数据库连接

conn = self.linkPoll.connection(*shareable*=False)

2.2.2 数据库表的设计

数据库只有一个资源记录表，所有字段组合作为复合主键

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| NAME | TEXT | 资源记录匹配的节点名称 |
| TYPE | SMALLINT | 两个字节，表示资源类型，比如A表示主机地址 |
| CLASS | SMALLINT | 两个字节，表示资源类，例如IN表示互联网类型 |
| TTL | INT | 四个字节，有符号数，他规定应当再次咨询信息源之前此记录可以被缓存的时间间隔 |
| RDLENGTH | SMALLINT | 两个字节，无符号数，它规定以字节来计的RDATA的长度 |
| RDATA | TEXT | 描述的资源记录，字节串，根据TYPE和CLASS来定 |

2.3 DNS报文的拆封包

采用BytesIO来读写字节流，通过struct来对字段进行pack和unpack，其中format参数以”!”开头，也就是big-endian模式，来保证网络字节序。

2.3.1 类的结构设计

① Message类



② Header类



③ Query类



④ Name类



⑤ ResourceRecord类



⑥ SimpleRecord类



⑦ RecordA类



⑧ RecordCNAME类



⑨ ReocrdNS类



⑩ RecordMX类



2.3.2 UML类图



三、软件流程图

四、测试用例以及运行结果

4.1 各模块测试

4.1.1 并发服务器测试

此部分主要测试服务器能否并发而且非阻塞式的响应请求，线程开启后在处理完任务时是否可以正常关闭结束。

4.1.2 数据库连接池测试

此部分主要测试线程之间能否正常共享使用数据库的连接池。

4.1.3 DNS报文拆包测试

此部分主要测试当收到一个dns报文时，能否将报文内容进行还原

① 域名测试

*#域名拆包*

n = Name()

strio = BytesIO(strio.getvalue())

n.decode(strio)

print(n.name)

测试结果

[Running] python -u "c:\Users\咸鱼\Desktop\DnsServer\test.py"

b'\x03www\x05baidu\x03com\x00\xc0\x0c'

b'www.baidu.com'

4.1.4 DNS报文封包测试

此部分主要测试如何根据请求发送一个dns报文

1. 域名测试

*# 域名封包测试*

def testEncodeName():

addr = b'www.baidu.com'

n = Name(addr)

strio = BytesIO()

nameDict = {}

n.encode(strio, *nameDict*=nameDict)

print(nameDict[addr])

n.encode(strio, *nameDict*=nameDict)

print(strio.getvalue())

测试结果

[Running] python -u "c:\Users\咸鱼\Desktop\DnsServer\test.py"

12

b'\x03www\x05baidu\x03com\x00\xc0\x0c'

②

4.1.5 DNS报文转发测试

从部分主要测试在转发报文中来回id是否匹配，能否正确的实现转发

4.2 综合测试

综合测试主要通过nslookup去测试

4.2.1 四种资源类型测试

4.2.2

五、调试中遇到并解决的问题

1. OSError: [WinError 10051]

此处附上图

StackOverFlow解释如下:

<https://stackoverflow.com/questions/30462729/error-10051-while-trying-to-connect-socket>

2.调用连接池的时候提示连接不能在创建的线程之外使用

check\_same\_thread = false

sqlite3

3.sqlite3不支持smallint等数据类型

自己本着简单的原则，选择了sqlite3这个轻型数据库，但是发现这个数据库在存储时可以识别其他类型数据库的数据格式，比如smallint但是会做一个类型转换

六、小组分工及承担任务比例

七、心得体会

自己本着尝试的心态，想独立做一个东西出来。也算是十分用心，在写代码的途中也遇到了许多问题。但更多的遇到的是对各种技术的探讨与比较以及最后的选择，比如在网络IO这一段,之所以选择非阻塞式IO是因为其比较简单。