

电子科技大学计算机科学与工程学院

标准实验报告

课程名称：互联网络程序设计

项目名称：非阻塞 DNS 客户端

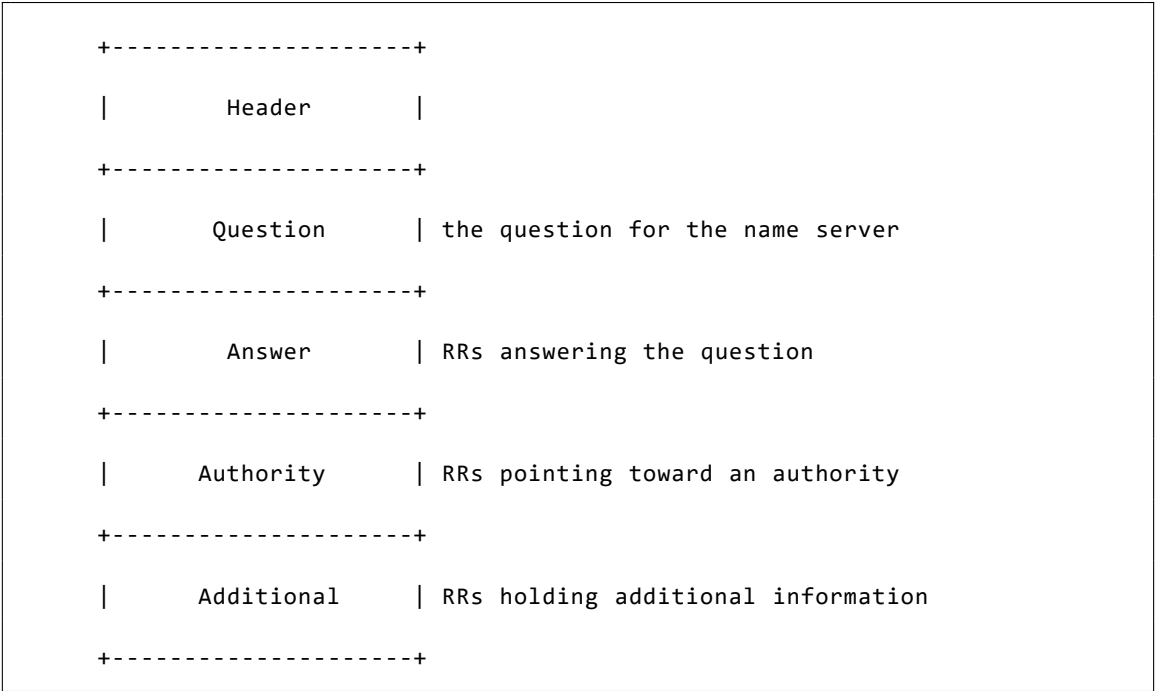
一、实验要求

- 在一个 Reactor 网络库上采用非阻塞方式编写一个 DNS 客户端，获得 ip 地址：
- 1) 查找资料了解 DNS 查询的报文交互过程，确定 RFC 对 DNS 交互报文的定义，根据报文定义设计并实现一个非阻塞 DNS 客户端
 - 2) 基于 Linux 平台，可以借助于 libuv 等现成网络库

二、实验原理

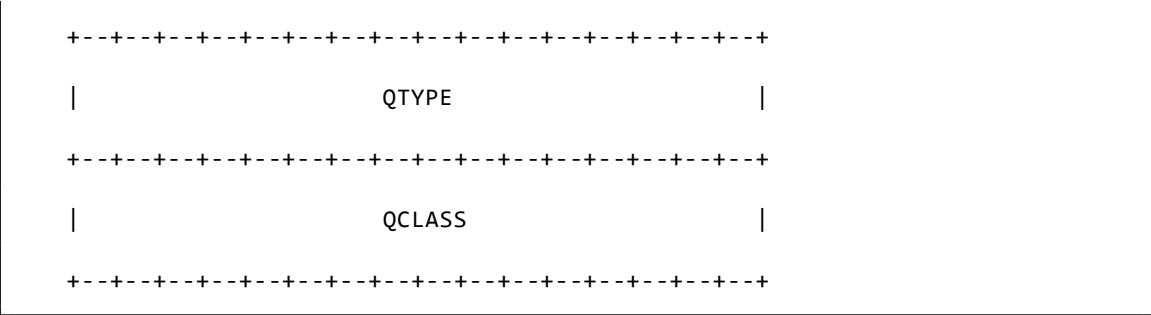
(一) Domain Name System

在 RFC1035 中定义了 DNS 的规范和实现，其中就包含对交互报文的定义。代码 1 指出了报文的基本格式，其由五部分组成。Header 节总是存在的，其指明了剩余的哪些节是存在的、此消息是请求还是响应和一些其他参数。



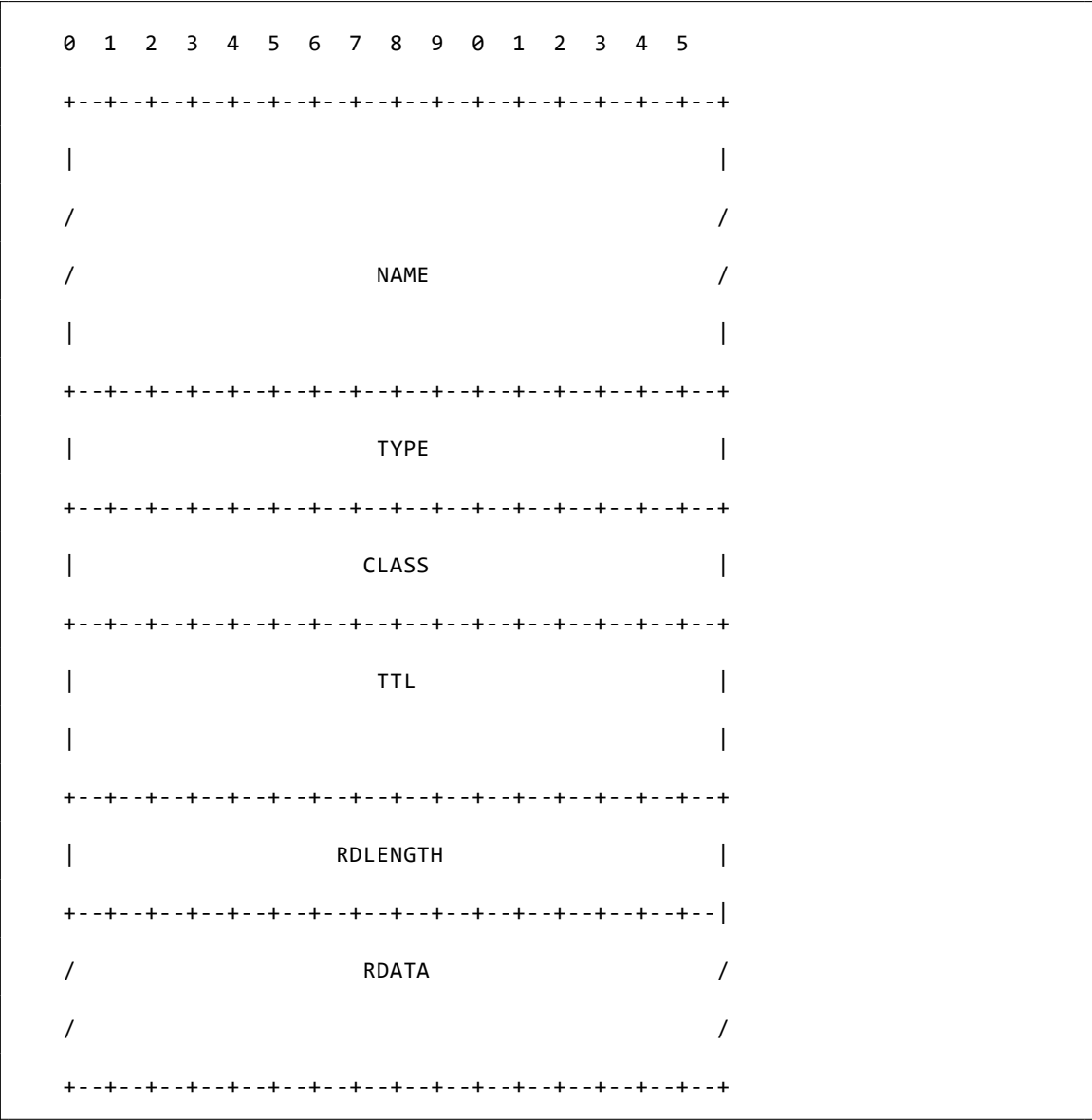
代码 1: DNS message format

代码 2 是 Header 节的定义，由于我们只是实现域名到 ip 地址的转换，所以只需要关注一下几个域：



代码 3: Question section format

代码 4 是 DNS 服务器响应的格式，在本例中只关心 RDATA 域，当请求是 A 时，其为 4 字节数据，存储了查询到的 ip 地址。



代码 4: Resource record format

(二) Unicorn Velociraptor

libuv 库是多平台 C 库，提供对基于事件循环的异步 I/O 的支持。它支持 epoll、kqueue、Windows 的 IOCP 和 Solaris 的事件端口。它主要设计用于 Node.js，但也可用于其他软件项目如 Julia 或 pyuv 等。它最初是 libev 或 Microsoft IOCP 上的抽象，libev 只支持 Unix 系统而不支持 Windows 上的 IOCP，在 node-v0.9.0 的 libuv 版本中去除了对 libev 的依赖。

三、实验步骤

(一) 总体架构设计

由于实现一个非阻塞的客户端，理所应当需要接受用户多个输入。所以在 libuv 事件循环中绑定标准输入并且设置 read 回调函数。实现 DNS 客户端需要以 udp 的方式创建一个 socket，并在事件循环中绑定，并设置 recv 回调函数。总体执行流程如图 1 所示，libuv 接受到用户输入时调用 read 回调函数，其将会根据用户的输入构建 DNS 请求报文发送给 DNS 服务器。DNS 服务器在查询结束后响应，libuv 收到服务器的响应报文后调用 recv 回调函数，从报文中提取出 ip 地址并输出。

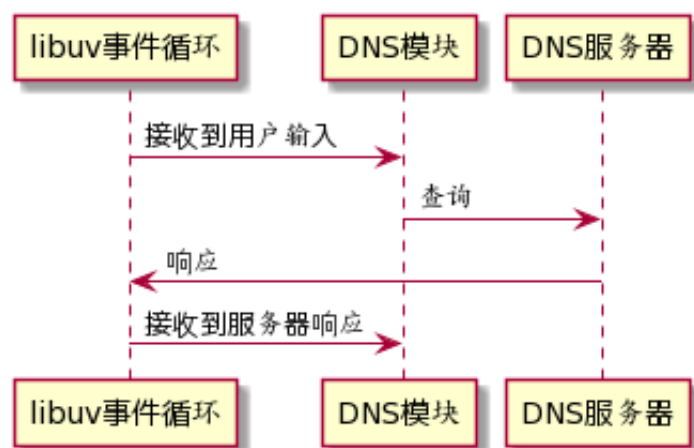


图 1: 程序执行流程

(二) DNS 模块的设计与实现

DNS 模块的核心代码就是根据用户请求的域名构建请求报文，和从服务器的响应报文中提取出 ip 信息。有了实验原理章节的介绍，已经对报文结构有了足够

的理解，接下来就是编写代码实现具体功能。

代码 5 是构造请求报文的主要代码：13-19 行构造报文的 Header 节，赋值 id、将 qcount 域设置为 1，表示本次请求只包含一个域名；21-38 行构造报文的 Question 节，首先根据压缩算法写入请求的域名，然后设置 QTYPE 和 QCLASS 域。

```
1 uv_buf_t make_dns_query_msg(char *host_name, ssize_t len_host_name){
2     for(int i=0; i<len_host_name; i++){
3         if(host_name[i] == '\n'){
4             host_name[i] = '\0';
5         }
6     }
7     char *buf = (char *)malloc(MAX_DNS_PACKAGE_SIZE);
8     memset((void *)buf, 0, MAX_DNS_PACKAGE_SIZE);
9
10    DNS_HEADER *dns_hdr = (DNS_HEADER *)buf;
11    int id = get_dns_id(host_name);
12
13    printf("[+] Query for {%d}%s\n", id, host_name);
14    dns_hdr->id = htons(id);
15    dns_hdr->rd = 1;
16    dns_hdr->qcount = htons(1);
17    dns_hdr->ancount = htons(0);
18    dns_hdr->nscount = htons(0);
19    dns_hdr->arcount = htons(0);
20
21    strcpy(buf + sizeof(DNS_HEADER)+1, host_name);
22    char *p = buf+sizeof(DNS_HEADER)+1;
23    u_char i = 0;
24    while (p < buf+sizeof(DNS_HEADER)+strlen(host_name)+1){
25        if(*p == '.'){
```

```

26         p[-i-1] = i;
27         i = 0;
28     }
29     else{
30         i++;
31     }
32     p++;
33 }
34 p[-i-1] = i;
35
36 DNS_QUESTION_SECTION_TAIL *dns_tail = (DNS_QUESTION_SECTION_TAIL *) (p+1);
37 dns_tail->qtype = htons(DNS_QTYPE_A);
38 dns_tail->qclass = htons(DNS_QCLASS_IN);
39
40 return uv_buf_init(buf, sizeof(DNS_HEADER)+sizeof(DNS_QUESTION_SECTION_TAIL
    )\
41         +strlen(host_name)+2);
42 }

```

代码 5: 构造请求报文

代码 6 是接收到响应报文的回调函数，其主要负责解析响应报文，并输出查询结果：11 行首先获得报文的 id 域的数据，用来查找该报文对应请求的域名；14-16 行从报文中读取 RDATA 域的数据，并将查询结果输出。

```

1 void on_dns_read(uv_udp_t *req, ssize_t nread, const uv_buf_t *buf,\
2     const struct sockaddr *addr, unsigned flags){
3     if(nread < 0){
4         fprintf(stderr, "Read error %s\n", uv_err_name(nread));
5         uv_close((uv_handle_t *)req, NULL);
6         free(buf->base);
7         return;

```

```

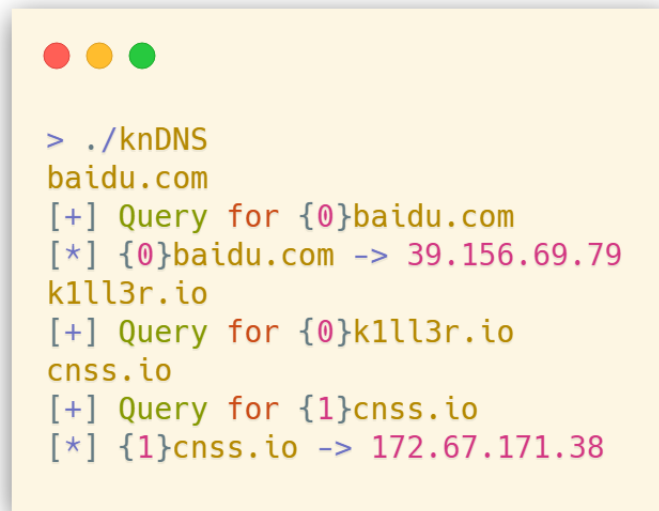
8      }
9      else if(nread > 0){
10         DNS_HEADER *header = (DNS_HEADER *)buf->base;
11         u_int id = ntohs(header->id);
12         char *host_name = get_dns_host_name(id);
13         if (header->ancount){
14             u_char *rdata = (u_char *) (buf->base+nread-4);
15             printf("[*] {%u}%s -> %u.%u.%u.%u\n", id, host_name, rdata[0],
16                    rdata[1],\
17                    rdata[2], rdata[3]);
18         }
19         else{
20             printf("[+] {%u}DNS query failed for %s\n", id, host_name);
21         }
22         // Free the chunk allocated by alloc_stdin_bufffer
23         free(host_name);
24     }
25     free(buf->base);
26     return;
27 }

```

代码 6: 解析响应报文

四、实验数据及结果与分析

图 2 是程序运行时的截图，我们首先查询 `baidu.com`，可以发现程序很好的发送了请求并且解析了服务器的响应。接着查询 `k1ll3r.io`，可能由于缓存没有命中，该次查询没有立即返回。紧接着我们查询 `cnss.io`，程序同样很好的处理了请求并解析了结果。可以发现由于上一次请求没有返回，这一次报文的 `id` 域的数值增加了，很好的处理了非阻塞的情况，也很好的体现了程序非阻塞的特性。

A terminal window with a yellow background and three colored window control buttons (red, yellow, green) at the top left. The text inside the terminal is as follows:

```
> ./knDNS
baidu.com
[+] Query for {0}baidu.com
[*] {0}baidu.com -> 39.156.69.79
k1ll3r.io
[+] Query for {0}k1ll3r.io
cnss.io
[+] Query for {1}cnss.io
[*] {1}cnss.io -> 172.67.171.38
```

图 2: 程序运行截图

五、实验结论、心得体会

借此实验很好地学习了网络程序设计知识，包括很多报告中没有体现的基础知识，例如 TCP/IP 协议的细节、IO 复用编程、多进程多线程并发等等。也深刻地体会到了实现一个能很好处理并发的网络程序的不易。

报告评分：

指导教师签字：