```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <WinSock2.h>
#include <Windows.h>
#include <time.h>
#pragma comment(lib, "WS2_32.lib")

#define PORT 53
#define BUFFER_SIZE 1024
#define LRUSIZE 10
const char* LOCAL_IP = "127.0.0.1";
const char* SERVER_IP = "10.3.9.4";
const char* LOCAL_FILE = "dnsrelay.txt";

typedef struct DNSHEADER {
    unsigned short ID;
    unsigned short Flag;
    unsigned short Qdcount;
    unsigned short Ancount;
    unsigned short Nscount;
    unsigned short Arcount;
}DNSHEADER;         //dns头

typedef struct DNSQUESTION {
    unsigned short Qtype;
    unsigned short Qclass;
}DNSQUESTION;       //dns问题

typedef struct DNSRESOURCE {
    unsigned short Type;
    unsigned short Class;
    unsigned int TTL;
    unsigned short Length;
}DNSRESOURCE;       //dns答案（上面三个结构体用到的不多，写出来方便查看结构）

typedef struct Node {
    char IP[20];
    char DN[80];
    struct Node* next;
}Node;                  //链表储存文件中IP和DomainName

Node* HEAD;                     //全局变量，文件信息的链表头

typedef struct NNode {
    char IP[20];
    char DN[80];
    struct NNode* pre;
    struct NNode* next;
}NNode;

typedef struct LRUList {
    int size;
    NNode* head;
    NNode* tail;
}LRUList;
```

```
57
58  LRUList LIST;       //存储了双向链表，用于执行LRU缓存
59
60  /*
61  将全局变量list初始化,设置双向链表的头，尾以及size
62  */
63  void initList() {
64      LIST.size = 0;
65      LIST.head = (NNode*)malloc(sizeof(NNode));
66      LIST.tail = (NNode*)malloc(sizeof(NNode));
67      LIST.head->pre = NULL;
68      LIST.head->next = LIST.tail;
69      LIST.tail->pre = LIST.head;
70      LIST.tail->next = NULL;
71  }
72
73  /*
74  *将节点p插入到head的后面
75  */
76  void putFirst(NNode* p) {
77      p->next = LIST.head->next;
78      p->pre = LIST.head;
79      LIST.head->next = p;
80      p->next->pre = p;
81  }
82
83  /*
84  *根据LRU算法将不存在的节点放入双向链表中
85  */
86  void setNNode(NNode* p) {
87  /*  原本想要留下这一段的，后来想到在lookup2函数中就可以将查找到的数据直接提前，所以注
      释掉了
88      NNode* q = LIST.head;
89      while (q->next != LIST.tail) {
90          //找到了就放到head后面
91          if (!strcmp(q->next->DN, p->DN)) {
92              putFirst(p);
93              p = q->next;
94              q->next = p->next;
95              p->next->pre = q;
96              free(p);
97              return;
98          }
99          q = q->next;
100     }*/
101     //没找到且缓存没满的时候放到头部，size++
102     if (LIST.size < LRUSIZE) {
103         putFirst(p);
104         LIST.size++;
105     }
106     else {
107         putFirst(p);
108         p = LIST.tail->pre;
109         LIST.tail->pre = p->pre;
110         p->pre->next = LIST.tail;
111         free(p);
```

```
112     }
113 }
114
115 /*
116 *读取LIST，返回查询域名的结果，包含域名则返回1，否则返回0
117 *domainName为需要读取的域名，ip为域名对应的IP地址，查找到的话就将结果填入到ip数组
118 */
119 int LookUp2(char* domainName, char* ip) {
120     NNode* p = LIST.head;
121     int flag = 0;
122     while (p->next!=LIST.tail) {
123         if (strcmp(domainName, p->next->DN) == 0) {
124             p = p->next;
125             p->pre->next = p->next;
126             p->next->pre = p->pre;
127             p->pre = LIST.head;
128             p->next = LIST.head->next;
129             LIST.head->next = p;
130             p->next->pre = p;
131             flag = 1;
132             char* t1 = p->IP;
133             char* t2 = p->IP;
134             int i = 0;
135             /*通过t1，t2两个指针进行读取，当读到'.'时，说明当前数字结束*/
136             while (*t1 != '\0') {
137                 if (*t1 == '.') {
138                     *t1 = '\0';
139                     ip[i] = (char)atoi(t2);
140                     i++;
141                     t2 = t1 + 1;
142                     *t1 = '.';
143                 }
144                 t1++;
145             }
146             /*字符串转数字*/
147             ip[i] = (char)atoi(t2);
148 //          printf("从缓存中找到\n");
149             return flag;
150         }
151         p = p->next;
152     }
153     return flag;
154 }
155
156 /*
157 *读取文件，并返回链表头的节点,同时输出文件中保存的信息
158 *参数file为需要读取的文件的路径，level表示调试等级
159 */
160 Node* OpenFileT(char* file, int level) {
161     initList();
162     int flag = 0;                       //用于标记当属于IP还是域名
163     FILE* f;
164     if ((f = fopen(file, "r")) == NULL) {
165         perror("Can't open the file");
166         exit(1);
167     }
```

```
168     char temp[BUFFER_SIZE];             //储存每一行的字符串
169     Node* head = (Node*)malloc(sizeof(Node));
170     head->next = NULL;
171     Node* p = head;
172     int n = 0;
173     while (!feof(f)) {
174         p->next = (Node*)malloc(sizeof(Node));
175         p = p->next;
176         fgets(temp, BUFFER_SIZE, f);
177         for (int i = 0, fl = 0, iplength = 0; i < BUFFER_SIZE; i++) {
178             /*IP*/
179             if (fl == 0) {
180                 if (temp[i] != ' ') {
181                     p->IP[i] = temp[i];
182                 }
183                 /*为空格时，IP范围结束*/
184                 else {
185                     iplength = i;
186                     fl = 1;
187                     p->IP[i] = '\0';
188                 }
189             }
190             /*域名*/
191             else if (fl == 1) {
192                 if (temp[i] != '\n') {
193                     p->DN[i - iplength - 1] = temp[i];
194                 }
195                 /*为换行时，域名范围结束*/
196                 else {
197                     p->DN[i - iplength - 1] = '\0';
198                     break;
199                 }
200             }
201         }
202         p->next = NULL;
203         n++;
204     }
205     /*调试等级为2时才输出文件信息*/
206     if (level == 2) {
207         printf("\n读取文件信息如下：\n");
208         for (Node* p = head->next; p; p = p->next) {
209             printf("\t%s  ", p->DN);
210             printf("%s\n ", p->IP);
211         }
212         printf("文件读取完毕,共%d条信息\n", n);
213     }
214     return head;
215 }
216
217 /*
218 *读取链表，返回查询域名的结果，包含域名则返回1，否则返回0
219 *domainName为需要读取的域名，ip为域名对应的IP地址，查找到的话就将结果填入到ip数组中，
    head为链表头
220 */
221 int LookUp1(char* domainName, char* ip, Node* head) {
222     Node* p = head;
```

```
223     int flag = 0;
224     while (p->next) {
225         if (strcmp(domainName, p->next->DN) == 0) {
226             flag = 1;
227             char* t1 = p->next->IP;
228             char* t2 = p->next->IP;
229             int i = 0;
230             /*通过t1，t2两个指针进行读取，当读到'.'时，说明当前数字结束*/
231             while (*t1 != '\0') {
232                 if (*t1 == '.') {
233                     *t1 = '\0';
234                     ip[i] = (char)atoi(t2);
235                     i++;
236                     t2 = t1 + 1;
237                     *t1 = '.';
238                 }
239                 t1++;
240             }
241             /*字符串转数字*/
242             ip[i] = (char)atoi(t2);
243             return flag;
244         }
245         p = p->next;
246     }
247     return flag;
248 }
249
250 /*
251 *读取双向链表和链表，返回不同的值
252 */
253 int LookUp(char* domainname, char* ip, Node* head) {
254     if (LookUp2(domainname, ip))
255         return 2;
256     if (LookUp1(domainname, ip, head))
257         return 1;
258     return 0;
259 }
260
261 /*
262 *输出域名时使用
263 */
264 void PrintName(char* buf) {
265     char* p = buf;
266     while (*p != 0) {
267         if (*p >= 33) {
268             printf("%c", *p);
269         }
270         else {
271             printf(".");
272         }
273         p++;
274     }
275 }
276
277 /*
278 *输出以buf为头的响应包中，从from开始表示的CNAME
```

```
279 */
280 void PrintNameTemp(char* buf, char* from) {
281     char* p = from;
282     while (*p != 0) {
283         /*当前位为0xC0时，表示下一位为指针，指向buf+下一位数字的位置*/
284         if (*p == (char)192) {
285             PrintNameTemp(buf, buf + *((unsigned char*)p + 1));
286             return;
287         }
288         else if (*p >= 33)
289             printf("%c", *p);
290         /*小于33的以'.'进行输出*/
291         else {
292             printf(".");
293         }
294         p++;
295     }
296     return;
297 }
298 
299 /*
300 *输出以buf为头的域名
301 */
302 void DomainName(char* buf)
303 {
304     printf("[DomainName = ");
305     PrintName(buf);
306     printf("]\t");
307 }
308 
309 /*
310 *输出以buf为头的域名
311 *同时会将数值小于33的字符改成'.'，便于在LookUp()函数中进行比较
312 *也会将域名转换成小写
313 */
314 void ToDomainName(char* buf) {
315     char* p = buf;
316     while (*p != 0) {
317         if (*p >= 33) {
318             if (*p >= 'A' && *p <= 'Z')
319                 *p = *p + 32;
320         }
321         else {
322             *p = '.';
323         }
324         p++;
325     }
326 }
327 
328 /*
329 *根据报文设置缓存，只在dns0和dns1中使用
330 */
331 void setBuf(char* buf) {
332     char* ip = (char*)malloc(4);
333     char* p = buf + 6;                      //此时指向ancount，表示答案数量
334     unsigned short n = *(unsigned short*)p; //强制转换成2B的short类型
```

```
335     n = ntohs(n);                           //注意大小端的调整
336     p = buf + 12;
337     p = p + strlen(p) + 1 + 4;
338     int first = 0;
339     //p指向第一个answer部分
340     for (unsigned short i = 0; i < n; i++) {
341         p = p + 2;
342         unsigned short type = ntohs(*(unsigned short*)p);   //answer类型
343         p = p + 8;
344         unsigned short length = ntohs(*(unsigned short*)p); //answer中的data长度
345         /*IPV4地址*/
346         if (type == 1) {
347             ip = p + 2;
348             if (!first) {
349                 NNode* node = (NNode*)malloc(sizeof(NNode));
350                 strcpy(node->DN, buf + 13);
351                 ToDomainName(node->DN);
352                 sprintf(node->IP, "%u.%u.%u.%u", *(ip) & 0xff, *(ip + 1) & 0xff, *(ip
                      + 2) & 0xff, *(ip + 3) & 0xff);
353                 setNNode(node);
354                 return;
355             }
356         }
357         p = p + 2 + length;
358     }
359 }
360
361 /*
362 *构造响应报文，在主机中查询到时使用，主要是通过更改查询报文实现的
363 *buf为询问报文，ip为查询到的IP地址，level为调试等级，只有调试等级为2时，会输出不安全信
      息
364 */
365 void Respond(char* buf, char* ip, int level) {
366     DNSHEADER* header = (DNSHEADER*)buf;
367     DNSRESOURCE* resouce;
368     /*IP为0.0.0.0响应报文flag中的RCODE为0x3，表示域名不存在*/
369     if (ip[0] == (char)0 && ip[1] == (char)0 && ip[2] == (char)0 && ip[3] == (char)0)
          {
370         /*调试等级为2时才会输出*/
371         if (level == 2)
372             printf("IPaddr is 0.0.0.0, the domainname is unsafe.\n");
373         header->Flag = htons(0x8183);
374     }
375     /*正常情况的响应报文flag为0x8180*/
376     else {
377         header->Flag = htons(0x8180);
378     }
379     /*回答数为1*/
380     header->Ancount = htons(1);
381
382     char* dn = buf + 12;                            //指向报文中的question头
383     char* name = dn + strlen(dn) + 1 + 4;           //指向报文中的answer头
384     unsigned short* nameTemp = (unsigned short*)name;
385     *nameTemp = htons(0xC00C);                      //将answer的前两个字节写成0xC0
386     /*对answer部分进行填写*/
387     resouce = (DNSRESOURCE*)(name + 2);
```

```
388     resouce->Type = htons(1);
389     resouce->Class = htons(1);
390     resouce->TTL = htons(0x0FFF);
391     resouce->Length = htons(4);
392     /*填入IP答案*/
393     char* data = (char*)resouce + 10;
394     *data = *ip;
395     *(data + 1) = *(ip + 1);
396     *(data + 2) = *(ip + 2);
397     *(data + 3) = *(ip + 3);
398 }
399
400 /*
401 *调试等级为1时使用，一般只读响应包，用于输出时间和客户端IP地址，以及域名
402 */
403 void PrintAnswerLess(SOCKADDR_IN client, const char* buf) {
404     time_t NowTime = time(NULL);
405     struct tm* t = localtime(&NowTime);
406     printf("\n%d/%02d/%02d,%02d:%02d:%02d  ", t->tm_year + 1900, t->tm_mon + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
407     printf("Client %d.%d.%d.%d    ", client.sin_addr.S_un.S_un_b.s_b1, client.sin_addr.S_un.S_un_b.s_b2, client.sin_addr.S_un.S_un_b.s_b3, client.sin_addr.S_un.S_un_b.s_b4);
408
409     char* ip = (char*)malloc(4);
410     char* p = buf + 12;
411     DomainName(p + 1);
412 }
413
414 /*
415 *调试等级为2时使用，用于读取查询包和响应包，并根据不同类型进行输出
416 *查询包输出时间、查询类型、以及客户端地址
417 *相应包则输出时间、答案（包括三种类型IPV4地址、CNAME和PTR指针）、服务器IP和客户端IP
418 */
419 void PrintAnswerMore(SOCKADDR_IN client, const char* buf, char* server_ip) {
420     printf("*************************************************************************\n");
421
422     time_t NowTime = time(NULL);
423     struct tm* t = localtime(&NowTime);
424     printf("%d/%02d/%02d,%02d:%02d:%02d\n", t->tm_year + 1900, t->tm_mon + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
425     char* ip = (char*)malloc(4);
426     char* p = buf + 6;                          //此时指向ancount，表示答案数量
427     unsigned short n = *(unsigned short*)p; //强制转换成2B的short类型
428     n = ntohs(n);                               //注意大小端的调整
429     p = buf + 12;
430     printf("[ID = 0x%x]\n", ntohs((*((unsigned short*)buf)) & 0xFFFF));
431     DomainName(p + 1);
432
433     /*tt指向header中的flag，用于判断第一位的QR，为0表示查询，1表示响应*/
434     unsigned short* tt = buf + 2;
435     /*查询包中的信息输出*/
436     if (*tt >> 15 == 0) {
437         char* temp = buf + 12;
438         temp = temp + strlen(temp) + 1;
439         printf("\n[TYPE = %u]\n", ntohs((*(unsigned short*)temp)) & 0xFFFF);
```

```
440         printf("\nASK FROM CLIENT %d.%d.%d.%d\n", client.sin_addr.S_un.S_un_b.s_b1,
              client.sin_addr.S_un.S_un_b.s_b2, client.sin_addr.S_un.S_un_b.s_b3,
              client.sin_addr.S_un.S_un_b.s_b4);
441         return;
442     }
443     /*响应包中的信息输出*/
444     printf("\n\n");
445     p = p + strlen(p) + 1 + 4;
446     int first = 0;
447     //p指向第一个answer部分
448     for (unsigned short i = 0; i < n; i++) {
449         p = p + 2;
450         unsigned short type = ntohs(*(unsigned short*)p);   //answer类型
451         p = p + 8;
452         unsigned short length = ntohs(*(unsigned short*)p); //answer中的data长度
453         /*IPV4地址*/
454         if (type == 1) {
455             ip = p + 2;
456             printf("[IP = %u.%u.%u.%u]\n", *(ip) & 0xff, *(ip + 1) & 0xff, *(ip + 2) &
                  0xff, *(ip + 3) & 0xff);
457             if (!first) {
458                 NNode* node = (NNode*)malloc(sizeof(NNode));
459                 strcpy(node->DN, buf + 13);
460                 ToDomainName(node->DN);
461                 sprintf(node->IP, "%u.%u.%u.%u", *(ip) & 0xff, *(ip + 1) & 0xff, *(ip
                    + 2) & 0xff, *(ip + 3) & 0xff);
462                 //printf("++++%s\n++++%s", node->DN, node->IP);
463                 setNNode(node);
464                 first = 1;
465             }
466         }
467         /*CNAME*/
468         else if (type == 5) {
469             printf("[CNAME = ");
470             PrintNameTemp(buf, p + 3);
471             printf("]\n");
472         }
473         /*PTR指针*/
474         else if (type == 12) {
475             printf("[PTR = ");
476             PrintName(p + 3);
477             printf("]\n");
478         }
479         /*IPV6地址，我的电脑从来没收到过IPV6指针，可以写但没必要*/
480         else if (type == 28) {
481         }
482         p = p + 2 + length;
483     }
484     /*答案数为0*/
485     if (n == 0) {
486         printf("No answer in this package\n");
487     }
488     /*输出服务器和客户端信息*/
489     printf("\nGET FROM SERVER %s\n", server_ip);
490     printf("SEND TO CLIENT %d.%d.%d.%d\n", client.sin_addr.S_un.S_un_b.s_b1,
          client.sin_addr.S_un.S_un_b.s_b2, client.sin_addr.S_un.S_un_b.s_b3,
```

```
        client.sin_addr.S_un.S_un_b.s_b4);
491
492     printf("************************************************************************\n");
493 }
494
495 /*
496 *无调试等级时使用
497 */
498 void dns0() {
499     char buf[BUFFER_SIZE];      //保存包的信息
500
501     /*准备UDP通信*/
502     WSADATA wsaData;
503     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
504         printf("error\n");
505         exit(1);
506     }
507     /*创建与客户端沟通的套接字*/
508     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
509     if (socketFd == SOCKET_ERROR) {
510         printf("Creat Socket Error\n");
511         exit(1);
512     }
513     /*主机，作为服务器，绑定IP和端口*/
514     SOCKADDR_IN server;
515     server.sin_family = AF_INET;
516     server.sin_port = htons(PORT);
517     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
518
519     /*客户端，不需要设置信息*/
520     SOCKADDR_IN client;
521
522     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
523     if (z != 0) {
524         printf("bind error\n");
525         exit(1);
526     }
527
528     /*创建与dns服务器沟通的套接字*/
529     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
530     if (DnsFd == SOCKET_ERROR) {
531         printf("Creat Socket Error\n");
532         exit(1);
533     }
534     /*设置接收非阻塞，防止包丢失后在循环中卡住*/
535     int timeout = 2000;
536     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
537
538     /*dns服务器，绑定IP和端口*/
539     SOCKADDR_IN Dns;
540     Dns.sin_family = AF_INET;
541     Dns.sin_port = htons(PORT);
542     Dns.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
543
544     char* temp = (char*)malloc(BUFFER_SIZE);    //用来存域名
545     char* ip = (char*)malloc(4);                //存IP
```

```
546     unsigned int len = sizeof(client);
547     while (1) {
548         /*将buf中全填入0*/
549         memset(buf, 0, BUFFER_SIZE);
550 
551         /*从客户端接收信息，接收失败则进行下一步循环*/
552         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
553         if (z < 0) {
554             continue;
555         }
556         /*将域名赋值到temp数组中*/
557         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
558         ToDomainName(temp);
559 
560 
561         /*确定询问类型*/
562         char* tempp = buf + 12;
563         char* typeptr = tempp + strlen(tempp) + 1;
564         unsigned short type = ntohs((*(unsigned short*)typeptr)) & 0xFFFF;
565 
566         int ifFind = 0;
567         /*询问类型为IPV4并且查询到结果则构造响应报文*/
568         if (type == 1) {
569             ifFind = LookUp(temp, ip, HEAD);
570             if (ifFind) {
571                 int ifFind = LookUp(temp, ip, HEAD);
572                 Respond(buf, ip, 1);
573             }
574         }
575         /*询问类型不是IPV4或者未查询到，需要从上层的dns服务器进行查询*/
576         if(!ifFind){
577             /*将查询包原封不动的发送给dns服务器*/
578             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
579 
580             unsigned short id = *(unsigned short*)buf;
581             unsigned short idtemp;
582             unsigned int i = sizeof(Dns);
583             unsigned int j;
584             /*只在响应包与查询包的id相同时才停止接收，
585             *由于接收的方式采用了非阻塞，当超过设置的超时时间时，会自动返回一个没有答案的响应包
586             *如果dns服务器之前发送的响应包到达比较晚，那么就会与现在所询问的不符合，就会产生所答非所问的情况，所以需要进行一次筛选
587             */
588             do
589             {
590                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
591                 idtemp = *(unsigned short*)buf;
592             } while (idtemp != id);
593             setBuf(buf);
594         }
595         /*将响应包原封不动的送个客户端，发送失败则不断重发至发送成功*/
596         do
597         {
598             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                sizeof(client));
```

```
599         } while (z < 0);
600     }
601 }
602
603 /*
604 *调试等级为1时使用，由于和dns0()函数区别不大，所以只在部分有区别的地方进行解释
605 */
606 void dns1(char* IP, char* file) {
607     char buf[BUFFER_SIZE];
608
609     WSADATA wsaData;
610     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
611         printf("error\n");
612         exit(1);
613     }
614
615     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
616     if (socketFd == SOCKET_ERROR) {
617         printf("Creat Socket Error\n");
618         exit(1);
619     }
620     SOCKADDR_IN server;
621     server.sin_family = AF_INET;
622     server.sin_port = htons(PORT);
623     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
624
625     SOCKADDR_IN client;
626
627     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
628     if (z != 0) {
629         printf("bind error\n");
630         exit(1);
631     }
632
633     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
634     int timeout = 2000;
635     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
636     if (DnsFd == SOCKET_ERROR) {
637         printf("Creat Socket Error\n");
638         exit(1);
639     }
640     SOCKADDR_IN Dns;
641     Dns.sin_family = AF_INET;
642     Dns.sin_port = htons(PORT);
643     Dns.sin_addr.S_un.S_addr = inet_addr(IP);
644
645     char* temp = (char*)malloc(BUFFER_SIZE);
646     char* ip = (char*)malloc(4);
647     unsigned int len = sizeof(client);
648     while (1) {
649         memset(buf, 0, BUFFER_SIZE);
650         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
651         if (z < 0) {
652             continue;
653         }
654         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
```

```
655         ToDomainName(temp);
656
657         /*确定询问类型*/
658         char* tempp = buf + 12;
659         char* typeptr = tempp + strlen(tempp) + 1;
660         unsigned short type = ntohs((*(unsigned short*)typeptr)) & 0xFFFF;
661
662         int ifFind = 0;
663
664         /*调试等级为1时，还要输出对应的信息*/
665         if (type == 1) {
666             ifFind = LookUp(temp, ip, HEAD);
667             if (ifFind) {
668                 time_t NowTime = time(NULL);
669                 struct tm* t = localtime(&NowTime);
670                 printf("\n%d/%02d/%02d,%02d:%02d:%02d  ", t->tm_year + 1900, t->tm_mon
                        + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
671                 printf("Client %d.%d.%d.%d    ", client.sin_addr.S_un.S_un_b.s_b1,
                      client.sin_addr.S_un.S_un_b.s_b2, client.sin_addr.S_un.S_un_b.s_b3,
                      client.sin_addr.S_un.S_un_b.s_b4);
672                 DomainName(temp);
673                 Respond(buf, ip, 1);
674             }
675         }
676         if(!ifFind){
677             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
678
679             unsigned short id = *(unsigned short*)buf;
680             unsigned short idtemp;
681             unsigned int i = sizeof(Dns);
682             unsigned int j;
683             do
684             {
685                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
686                 idtemp = *(unsigned short*)buf;
687             } while (idtemp != id);
688             /*输出响应包的信息*/
689             PrintAnswerLess(client, buf);
690             setBuf(buf);
691         }
692         do
693         {
694             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                  sizeof(client));
695         } while (z < 0);
696     }
697 }
698
699 /*
700 *同dns1仅解释部分不同代码
701 */
702 void dns2(char* IP) {
703     char buf[BUFFER_SIZE];
704
705     WSADATA wsaData;
706     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
```

```
707         printf("error\n");
708         exit(1);
709     }
710
711     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
712     if (socketFd == SOCKET_ERROR) {
713         printf("Creat Socket Error\n");
714         exit(1);
715     }
716     SOCKADDR_IN server;
717     server.sin_family = AF_INET;
718     server.sin_port = htons(PORT);
719     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
720
721     SOCKADDR_IN client;
722
723     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
724     if (z != 0) {
725         printf("bind error\n");
726         exit(1);
727     }
728
729     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
730     int timeout = 2000;
731     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
732     if (DnsFd == SOCKET_ERROR) {
733         printf("Creat Socket Error\n");
734         exit(1);
735     }
736
737     SOCKADDR_IN Dns;
738     Dns.sin_family = AF_INET;
739     Dns.sin_port = htons(PORT);
740     Dns.sin_addr.S_un.S_addr = inet_addr(IP);
741
742     char* temp = (char*)malloc(BUFFER_SIZE);
743     char* ip = (char*)malloc(4);
744     unsigned int len = sizeof(client);
745     while (1) {
746         memset(buf, 0, BUFFER_SIZE);
747         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
748         printf("\n\n");
749         if (z < 0) {
750             continue;
751         }
752         PrintAnswerMore(client, buf, IP);
753         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
754         ToDomainName(temp);
755
756         /*确定询问类型*/
757         char* tempp = buf + 12;
758         char* typeptr = tempp + strlen(tempp) + 1;
759         unsigned short type = ntohs((*(unsigned short*)typeptr)) & 0xFFFF;
760
761         int ifFind = 0;
762         /*输出的信息更多*/
```

```
763         if (type == 1) {
764             ifFind = LookUp(temp, ip, HEAD);
765             if (ifFind) {
766                 printf
                    ("**********************************************************************
                    **\n");
767                 time_t NowTime = time(NULL);
768                 struct tm* t = localtime(&NowTime);
769                 printf("%d/%02d/%02d,%02d:%02d:%02d\n", t->tm_year + 1900, t->tm_mon +
                     1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
770                 printf("ASK FROM CLIENT %d.%d.%d.%d\n\n",
                    client.sin_addr.S_un.S_un_b.s_b1, client.sin_addr.S_un.S_un_b.s_b2,
                    client.sin_addr.S_un.S_un_b.s_b3, client.sin_addr.S_un.S_un_b.s_b4);
771                 printf("[ID = 0x%x]\n", ntohs((*((unsigned short*)buf)) & 0xFFFF));
772
773                 DomainName(temp);
774                 printf("\n[IP = %u.%u.%u.%u]\n", *ip & 0xff, *(ip + 1) & 0xff, *(ip +
                    2) & 0xff, *(ip + 3) & 0xff);
775                 Respond(buf, ip, 2);
776                 if (ifFind == 1)
777                     printf("\nFind From Host %s\n", LOCAL_IP);
778                 else if (ifFind == 2)
779                     printf("\nFind From Host Cache\n");
780                 printf("SEND TO CLIENT %d.%d.%d.%d\n",
                    client.sin_addr.S_un.S_un_b.s_b1, client.sin_addr.S_un.S_un_b.s_b2,
                    client.sin_addr.S_un.S_un_b.s_b3, client.sin_addr.S_un.S_un_b.s_b4);
781                 printf
                    ("**********************************************************************
                    **\n");
782             }
783         }
784         if(!ifFind){
785             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
786             unsigned short id = *(unsigned short*)buf;
787             unsigned short idtemp;
788             unsigned int i = sizeof(Dns);
789             unsigned int j;
790             do
791             {
792                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
793                 idtemp = *(unsigned short*)buf;
794             } while (idtemp != id);
795             /*输出响应包信息*/
796             PrintAnswerMore(client, buf, IP);
797         }
798
799         do
800         {
801             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                  sizeof(client));
802         } while (z < 0);
803     }
804 }
805
806 /*
807 *main()函数会根据命令行参数的不同，进而执行不同的dns函数
```

```
808 */
809 int main(int argc, char* argv[], char* envp[]) {
810     printf("\nDNSRELAY, Version 1.5 Build: 2020/09/05 16:23\n");
811     printf("Usage: dnsrelay [-d|-dd] [<dns-server>] [<db-file>]\n\n");
812 
813     if (argc == 1) {
814         printf("调试信息接级别0 无调试信息输出\n");
815         printf("指定名字服务器为 %s:53\n", SERVER_IP);
816         printf("使用默认配置文件 %s\n", LOCAL_FILE);
817         HEAD = OpenFileT(LOCAL_FILE, 0);
818         dns0();
819     }
820     else if (argc == 4 && !strcmp(argv[1], "-d")) {
821         printf("调试信息接级别1 简单调试信息输出\n");
822         printf("指定名字服务器为 %s:53\n", argv[2]);
823         printf("使用指定配置文件 %s\n", argv[3]);
824         HEAD = OpenFileT(argv[3], 1);
825         dns1(argv[2], argv[3]);
826     }
827     else if (argc == 3 && !strcmp(argv[1], "-dd")) {
828         printf("调试信息接级别2 复杂调试信息输出\n");
829         printf("指定名字服务器为 %s:53\n", argv[2]);
830         printf("使用默认配置文件 %s\n", LOCAL_FILE);
831         HEAD = OpenFileT(LOCAL_FILE, 2);
832         dns2(argv[2]);
833     }
834     else {
835         printf("参数输入有误，请重新输入\n");
836     }
837     return 0;
838 }
839 
```