```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
const char* LOCAL_IP = "127.0.0.1";
const char* SERVER_IP = "192.168.124.1";
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

/*
*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
*参数file为需要读取的文件的路径，level表示调试等级
*/
Node* OpenFileT(char* file, int level) {

    int flag = 0;                       //用于标记当属于IP还是域名
    FILE* f;
    if ((f = fopen(file, "r")) == NULL) {
        perror("Can't open the file");
        exit(1);
    }
    char temp[BUFFER_SIZE];             //储存每一行的字符串
```

```
57      Node* head = (Node*)malloc(sizeof(Node));
58      head->next = NULL;
59      Node* p = head;
60      int n = 0;
61      while (!feof(f)) {
62          fgets(temp, BUFFER_SIZE, f);
63          for (int i = 0, fl = 0, iplength = 0; i < BUFFER_SIZE; i++) {
64              /*IP*/
65              if (fl == 0) {
66                  if (temp[i] != ' ') {
67                      p->IP[i] = temp[i];
68                  }
69                  /*为空格时，IP范围结束*/
70                  else {
71                      iplength = i;
72                      fl = 1;
73                      p->IP[i] = '\0';
74                  }
75              }
76              /*域名*/
77              else if (fl == 1) {
78                  if (temp[i] != '\n') {
79                      p->DN[i - iplength - 1] = temp[i];
80                  }
81                  /*为换行时，域名范围结束*/
82                  else {
83                      p->DN[i - iplength - 1] = '\0';
84                      break;
85                  }
86              }
87          }
88          p->next = (Node*)malloc(sizeof(Node));
89          p = p->next;
90          p->next = NULL;
91          n++;
92      }
93      /*调试等级为2时才输出文件信息*/
94      if (level == 2) {
95          printf("\n读取文件信息如下：\n");
96          for (Node* p = head; p; p = p->next) {
97              printf("\t%s  ", p->DN);
98              printf("%s\n ", p->IP);
99          }
100         printf("文件读取完毕,共%d条信息\n", n);
101     }
102     return head;
103 }
104
105 /*
106 *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
107 *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
108 */
109 int LookUp(char* domainName, char* ip, Node* head) {
110     Node* p = head;
111     int flag = 0;
```

```
112     while (p) {
113         if (strcmp(domainName, p->DN) == 0) {
114             flag = 1;
115             char* t1 = p->IP;
116             char* t2 = p->IP;
117             int i = 0;
118             /*通过t1，t2两个指针进行读取，当读到'.'时，说明当前数字结束*/
119             while (*t1 != '\0') {
120                 if (*t1 == '.') {
121                     *t1 = '\0';
122                     ip[i] = (char)atoi(t2);
123                     i++;
124                     t2 = t1 + 1;
125                 }
126                 t1++;
127             }
128             /*字符串转数字*/
129             ip[i] = (char)atoi(t2);
130             return flag;
131         }
132         p = p->next;
133     }
134     return flag;
135 }
136 
137 /*
138 *输出域名时使用
139 */
140 void PrintName(char* buf) {
141     char* p = buf;
142     while (*p != 0) {
143         if (*p >= 33) {
144             printf("%c", *p);
145         }
146         else {
147             printf(".");
148         }
149         p++;
150     }
151 }
152 
153 /*
154 *输出以buf为头的响应包中，从from开始表示的CNAME
155 */
156 void PrintNameTemp(char* buf, char* from) {
157     char* p = from;
158     while (*p != 0) {
159         /*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*/
160         if (*p == (char)192) {
161             PrintNameTemp(buf, buf + *((unsigned char*)p + 1));
162             return;
163         }
164         else if (*p >= 33)
165             printf("%c", *p);
166         /*小于33的以'.'进行输出*/
167         else {
```

```
168             printf(".");
169         }
170         p++;
171     }
172     return;
173 }
174 
175 /*
176 *输出以buf为头的域名
177 */
178 void DomainName(char* buf)
179 {
180     printf("[DomainName = ");
181     PrintName(buf);
182     printf("]\t");
183 }
184 
185 /*
186 *输出以buf为头的域名
187 *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
188 */
189 void ToDomainName(char* buf) {
190     char* p = buf;
191     while (*p != 0) {
192         if (*p >= 33) {
193             if (*p >= 'A' && *p <= 'Z')
194                 *p = *p + 32;
195 //          printf("%c", *p);
196         }
197         else {
198             *p = '.';
199 //          printf(".");
200         }
201         p++;
202     }
203 }
204 
205 /*
206 *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
207 *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
208 */
209 void Respond(char* buf, char* ip, int level) {
210     DNSHEADER* header = (DNSHEADER*)buf;
211     DNSRESOURCE* resouce;
212     /*IP为0.0.0.0响应报文flag中的RCODE为0x3，表示域名不存在*/
213     if (ip[0] == (char)0 && ip[1] == (char)0 && ip[2] == (char)0 && ip[3] == (char)0) {
214         /*调试等级为2时才会输出*/
215         if(level == 2)
216             printf("IPaddr is 0.0.0.0, the domainname is unsafe.\n");
217         header->Flag = htons(0x8183);
218     }
219     /*正常情况的响应报文flag为0x8180*/
220     else {
221         header->Flag = htons(0x8180);
```

```
222     }
223     /*回答数为1*/
224     header->Ancount = htons(1);
225
226     char* dn = buf + 12;                                //指向报文中的question头
227     char* name = dn + strlen(dn) + 1 + 4;               //指向报文中的answer头
228     unsigned short* nameTemp = (unsigned short*)name;
229     *nameTemp = htons(0xC00C);                          //将answer的前两个字节写成0xC0
230     /*对answer部分进行填写*/
231     resouce = (DNSRESOURCE*)(name + 2);
232     resouce->Type = htons(1);
233     resouce->Class = htons(1);
234     resouce->TTL = htons(0x0FFF);
235     resouce->Length = htons(4);
236     /*填入IP答案*/
237     char* data = (char*)resouce + 10;
238     *data = *ip;
239     *(data + 1) = *(ip + 1);
240     *(data + 2) = *(ip + 2);
241     *(data + 3) = *(ip + 3);
242 }
243
244 /*
245 *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
246 */
247 void PrintAnswerLess(SOCKADDR_IN client,const char* buf) {
248     time_t NowTime = time(NULL);
249     struct tm* t = localtime(&NowTime);
250     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);
251     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);
252
253     char* ip = (char*)malloc(4);
254     char* p = buf + 12;
255     DomainName(p + 1);
256 }
257
258 /*
259 *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
260 *查询包输出时间、查询类型、以及客户端地址
261 *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
262 */
263 void PrintAnswerMore(SOCKADDR_IN client, const char* buf, char* server_ip) {
264     printf("*************************************************************************\n");
265
266     time_t NowTime = time(NULL);
267     struct tm* t = localtime(&NowTime);
268     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);
269     char* ip = (char*)malloc(4);
270     char* p = buf + 6;                          //此时指向ancount，表示答案数量
271     unsigned short n = *(unsigned short*)p; //强制转换成2B的short类型
272     n = ntohs(n);                               //注意大小端的调整
273     p = buf + 12;
```

```
274     printf("[ID = 0x%x]\n", ntohs((*((unsigned short*)buf)) & 0xFFFF));
275     DomainName(p + 1);
276
277     /*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*/
278     unsigned short* tt = buf + 2;
279     /*查询包中的信息输出*/
280     if (*tt >> 15 == 0) {
281         char* temp = buf + 12;
282         temp = temp + strlen(temp) + 1;
283         printf("\n[TYPE = %u]\n", ntohs((*(unsigned short*)temp)) & 0xFFFF);
284         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);
285         return;
286     }
287     /*响应包中的信息输出*/
288     printf("\n\n");
289     p = p + strlen(p) + 1 + 4;              //p指向第一个answer部分
290     for (unsigned short i = 0; i < n; i++) {
291         p = p + 2;
292         unsigned short type = ntohs(*(unsigned short*)p);   //answer类型
293         p = p + 8;
294         unsigned short length = ntohs(*(unsigned short*)p); //answer中的data长度
295         /*IPV4地址*/
296         if (type == 1) {
297             ip = p + 2;
298             printf("[IP = %u.%u.%u.%u]\n", *(ip) & 0xff, *(ip + 1) & 0xff, *(ip + 2) &
                  0xff, *(ip + 3) & 0xff);
299         }
300         /*CNAME*/
301         else if(type == 5){
302             printf("[CNAME = ");
303             PrintNameTemp(buf, p + 3);
304             printf("]\n");
305         }
306         /*PTR指针*/
307         else if (type == 12) {
308             printf("[PTR = ");
309             PrintName(p + 3);
310             printf("]\n");
311         }
312         /*IPV6地址，我的电脑从来没收到过IPV6指针，可以写但没必要*/
313         else if (type == 28) {
314         }
315         p = p + 2 + length;
316     }
317     /*答案数为0*/
318     if (n == 0) {
319         printf("No answer in this package\n");
320     }
321     /*输出服务器和客户端信息*/
322     printf("\nGET FROM SERVER %s\n", server_ip);
323     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);
324
```

```
325     printf("*************************************************************************\n");
326 }
327
328 /*
329 *无调试等级时使用
330 */
331 void dns0() {
332     char buf[BUFFER_SIZE];      //保存包的信息
333
334     /*准备UDP通信*/
335     WSADATA wsaData;
336     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
337         printf("error\n");
338         exit(1);
339     }
340     /*创建与客户端沟通的套接字*/
341     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
342     if (socketFd == SOCKET_ERROR) {
343         printf("Creat Socket Error\n");
344         exit(1);
345     }
346     /*主机，作为服务器，绑定IP和端口*/
347     SOCKADDR_IN server;
348     server.sin_family = AF_INET;
349     server.sin_port = htons(PORT);
350     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
351
352     /*客户端，不需要设置信息*/
353     SOCKADDR_IN client;
354
355     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
356     if (z != 0) {
357         printf("bind error\n");
358         exit(1);
359     }
360
361     /*创建与dns服务器沟通的套接字*/
362     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
363     if (DnsFd == SOCKET_ERROR) {
364         printf("Creat Socket Error\n");
365         exit(1);
366     }
367     /*设置接收非阻塞，防止包丢失后在循环中卡住*/
368     int timeout = 2000;
369     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
370
371     /*dns服务器，绑定IP和端口*/
372     SOCKADDR_IN Dns;
373     Dns.sin_family = AF_INET;
374     Dns.sin_port = htons(PORT);
375     Dns.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
376
377     char* temp = (char*)malloc(BUFFER_SIZE);    //用来存域名
378     char* ip = (char*)malloc(4);                //存IP
379     unsigned int len = sizeof(client);
380     while (1) {
```

```
381         /*将buf中全填入0*/
382         memset(buf, 0, BUFFER_SIZE);
383 
384         /*从客户端接收信息，接收失败则进行下一步循环*/
385         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
386         if (z < 0) {
387             continue;
388         }
389         /*将域名赋值到temp数组中*/
390         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
391         ToDomainName(temp);
392         /*从链表中查询*/
393         int ifFind = LookUp(temp, ip, HEAD);
394 
395         /*查询到结果则构造响应报文*/
396         if (ifFind == 1) {
397             Respond(buf, ip, 0);
398         }
399         /*未查询到，需要从上层的dns服务器进行查询*/
400         else if (ifFind == 0) {
401             /*将查询包原封不动的发送给dns服务器*/
402             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
403 
404             unsigned short id = *(unsigned short*)buf;
405             unsigned short idtemp;
406             unsigned int i = sizeof(Dns);
407             unsigned int j;
408             /*只在响应包与查询包的id相同时才停止接收，
409             *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
410             *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
411             */
412             do
413             {
414                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
415                 idtemp = *(unsigned short*)buf;
416             } while (idtemp != id);
417         }
418         /*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*/
419         do
420         {
421             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                    sizeof(client));
422         } while (z < 0);
423     }
424 }
425 
426 /*
427 *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
428 */
429 void dns1(char* IP, char* file) {
430     char buf[BUFFER_SIZE];
431 
432     WSADATA wsaData;
433     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
```

```
434         printf("error\n");
435         exit(1);
436     }
437
438     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
439     if (socketFd == SOCKET_ERROR) {
440         printf("Creat Socket Error\n");
441         exit(1);
442     }
443     SOCKADDR_IN server;
444     server.sin_family = AF_INET;
445     server.sin_port = htons(PORT);
446     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
447
448     SOCKADDR_IN client;
449
450     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
451     if (z != 0) {
452         printf("bind error\n");
453         exit(1);
454     }
455
456     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
457     int timeout = 2000;
458     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
459     if (DnsFd == SOCKET_ERROR) {
460         printf("Creat Socket Error\n");
461         exit(1);
462     }
463     SOCKADDR_IN Dns;
464     Dns.sin_family = AF_INET;
465     Dns.sin_port = htons(PORT);
466     Dns.sin_addr.S_un.S_addr = inet_addr(IP);
467
468     char* temp = (char*)malloc(BUFFER_SIZE);
469     char* ip = (char*)malloc(4);
470     unsigned int len = sizeof(client);
471     while (1) {
472         memset(buf, 0, BUFFER_SIZE);
473         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
474         if (z < 0) {
475             continue;
476         }
477         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
478         ToDomainName(temp);
479
480         int ifFind = LookUp(temp, ip, HEAD);
481
482         /*调试等级为1时，还要输出对应的信息*/
483         if (ifFind == 1) {
484             time_t NowTime = time(NULL);
485             struct tm* t = localtime(&NowTime);
486             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);
487             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);
488             DomainName(temp);
489             Respond(buf, ip, 1);
490         }
491         else if (ifFind == 0) {
492             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
493
494             unsigned short id = *(unsigned short*)buf;
495             unsigned short idtemp;
496             unsigned int i = sizeof(Dns);
497             unsigned int j;
498             do
499             {
500                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
501                 idtemp = *(unsigned short*)buf;
502             } while (idtemp != id);
503             /*输出响应包的信息*/
504             PrintAnswerLess(client, buf);
505         }
506
507         do
508         {
509             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                  sizeof(client));
510         } while (z < 0);
511     }
512 }
513
514 /*
515 *同dns1仅解释部分不同代码
516 */
517 void dns2(char* IP) {
518     char buf[BUFFER_SIZE];
519
520     WSADATA wsaData;
521     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
522         printf("error\n");
523         exit(1);
524     }
525
526     SOCKET socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
527     if (socketFd == SOCKET_ERROR) {
528         printf("Creat Socket Error\n");
529         exit(1);
530     }
531     SOCKADDR_IN server;
532     server.sin_family = AF_INET;
533     server.sin_port = htons(PORT);
534     server.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
535
536     SOCKADDR_IN client;
537
538     int z = bind(socketFd, (struct sockaddr*)&server, sizeof(server));
539     if (z != 0) {
540         printf("bind error\n");
541         exit(1);
```

```
542     }
543
544     SOCKET DnsFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
545     int timeout = 2000;
546     setsockopt(DnsFd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int));
547     if (DnsFd == SOCKET_ERROR) {
548         printf("Creat Socket Error\n");
549         exit(1);
550     }
551
552     SOCKADDR_IN Dns;
553     Dns.sin_family = AF_INET;
554     Dns.sin_port = htons(PORT);
555     Dns.sin_addr.S_un.S_addr = inet_addr(IP);
556
557     char* temp = (char*)malloc(BUFFER_SIZE);
558     char* ip = (char*)malloc(4);
559     unsigned int len = sizeof(client);
560     while (1) {
561         memset(buf, 0, BUFFER_SIZE);
562         z = recvfrom(socketFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&client, &len);
563         printf("\n\n");
564         if (z < 0) {
565             continue;
566         }
567         PrintAnswerMore(client, buf, IP);
568         strcpy(temp, buf + sizeof(DNSHEADER) + 1);
569         ToDomainName(temp);
570         int ifFind = LookUp(temp, ip, HEAD);
571         /*输出的信息更多*/
572         if (ifFind == 1) {
573             printf
                  ("*************************************************************************
                  \n");
574             time_t NowTime = time(NULL);
575             struct tm* t = localtime(&NowTime);
576             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);
577             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);
578             printf("[ID = 0x%x]\n", ntohs((*((unsigned short*)buf)) & 0xFFFF));
579
580             DomainName(temp);
581             printf("\n[IP = %u.%u.%u.%u]\n", *ip & 0xff, *(ip + 1) & 0xff, *(ip + 2) &
                   0xff, *(ip + 3) & 0xff);
582             Respond(buf, ip, 2);
583             printf("\nFind From Host %s\n", LOCAL_IP);
584             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);
585             printf
                  ("*************************************************************************
                  \n");
586         }
587         else if (ifFind == 0) {
```

```
588             sendto(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, sizeof(Dns));
589             unsigned short id = *(unsigned short*)buf;
590             unsigned short idtemp;
591             unsigned int i = sizeof(Dns);
592             unsigned int j;
593             do
594             {
595                 j = recvfrom(DnsFd, buf, BUFFER_SIZE, 0, (struct sockaddr*)&Dns, &i);
596                 idtemp = *(unsigned short*)buf;
597             } while (idtemp != id);
598             /*输出响应包信息*/
599             PrintAnswerMore(client, buf, IP);
600         }
601 
602         do
603         {
604             z = sendto(socketFd, buf, sizeof(buf), 0, (struct sockaddr*)&client,
                  sizeof(client));
605         } while (z < 0);
606     }
607 }
608 
609 /*
610 *main()函数会根据命令行参数的不同，进而执行不同的dns函数
611 */
612 int main(int argc, char* argv[], char* envp[]) {
613     printf("\nDNSRELAY, Version 1.4 Build: 2020/08/10 10:39\n");
614     printf("Usage: dnsrelay [-d|-dd] [<dns-server>] [<db-file>]\n\n");
615 
616     if (argc == 1) {
617         printf("调试信息接级别0 无调试信息输出\n");
618         printf("指定名字服务器为 %s:53\n", SERVER_IP);
619         printf("使用默认配置文件 %s\n", LOCAL_FILE);
620         HEAD = OpenFileT(LOCAL_FILE, 0);
621         dns0();
622     }
623     else if (argc == 4 && !strcmp(argv[1], "-d")) {
624         printf("调试信息接级别1 简单调试信息输出\n");
625         printf("指定名字服务器为 %s:53\n", argv[2]);
626         printf("使用指定配置文件 %s\n", argv[3]);
627         HEAD = OpenFileT(argv[3], 1);
628         dns1(argv[2], argv[3]);
629     }
630     else if (argc == 3 && !strcmp(argv[1], "-dd")) {
631         printf("调试信息接级别2 复杂调试信息输出\n");
632         printf("指定名字服务器为 %s:53\n", argv[2]);
633         printf("使用默认配置文件 %s\n", LOCAL_FILE);
634         HEAD = OpenFileT(LOCAL_FILE, 2);
635 //      printf("\n");
636         dns2(argv[2]);
637     }
638     else {
639         printf("参数输入有误，请重新输入\n");
640     }
641     return 0;
642 }
```