5.8. Extended Example: DNS Client

This Extended Example is a minimal client for performing a DNS query for IPv4 addresses. Given a domain name (such as `example.com`), the client sets up the DNS question in `setup_dns_request()` (lines 200 – 234). The `packed` attribute of the `dns_record_a_t` (lines 31 – 39) ensures that the compiler does not add any unnecessary padding that would make the question improperly structured. The `build_domain_qname()` function (lines 95 – 134) replaces the dots in the domain name with an integer to denote the length of the next field. Lines 65 – 82 perform the actual query, sending the request to OpenDNS using a UDP socket. The client is only designed to support DNS `A` type records for IPv4. I.e., this client does not support `CNAME`, `NS`, or `MX` records. The `print_dns_response()` function (lines 153 – 198) will stop if any other record types are encountered.

```c
#include <arpa/inet.h>
#include <assert.h>
#include <inttypes.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdint.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

/* Structure of the bytes for a DNS header */
typedef struct {
  uint16_t xid;
  uint16_t flags;
  uint16_t qdcount;
  uint16_t ancount;
  uint16_t nscount;
  uint16_t arcount;
} dns_header_t;

/* Structure of the bytes for a DNS question */
typedef struct
{
  char *name;
  uint16_t dnstype;
  uint16_t dnsclass;
} dns_question_t;

/* Structure of the bytes for an IPv4 answer */
typedef struct {
  uint16_t compression;
  uint16_t type;
  uint16_t class;
  uint32_t ttl;
  uint16_t length;
  struct in_addr addr;
} __attribute__((packed)) dns_record_a_t;

char * build_domain_qname (char *);
void print_byte_block (uint8_t *, size_t);
void print_dns_response (uint8_t *);
uint8_t * setup_dns_request (char *, size_t *);

int
main (int argc, char *argv[])
{
  if (argc != 2)
    {
      fprintf (stderr, "ERROR: Must pass a domain name\n");
      return 1;
    }

  char *hostname = argv[1];

  /* Set up the packet and get the length */
  size_t packetlen = 0;
  uint8_t *packet = setup_dns_request (hostname, &packetlen);

  /* Print the raw bytes formatted as 0000 0000 0000 ... */
  printf ("Lookup %s\n", hostname);
  print_byte_block (packet, packetlen);

  /* Send the packet to OpenDNS. Create an IPv4 UDP socket to
     208.67.222.222 (0xd043dede), the IP address for OpenDNS.
     DNS servers listen on port 53. */
  int socketfd = socket (AF_INET, SOCK_DGRAM, 0);
  struct sockaddr_in address;
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = htonl (0xd043dede);
  address.sin_port = htons (53);

  /* Send the request and get the response */
  sendto (socketfd, packet, packetlen, 0, (struct sockaddr *)&address,
          (socklen_t)sizeof (address));

  socklen_t length = 0;
  uint8_t response[512];
  memset (&response, 0, 512);
  ssize_t bytes = recvfrom (socketfd, response, 512, 0,
                            (struct sockaddr *)&address, &length);

  /* Print the raw bytes formatted as 0000 0000 0000 ... */
  printf ("Received %zd bytes from %s:\n", bytes,
          inet_ntoa (address.sin_addr));
  print_byte_block (response, bytes);

  /* Parse the DNS response into a struct and print the result */
  print_dns_response (response);

  return 0;
}

char *
build_domain_qname (char *hostname)
{
  assert (hostname != NULL);

  char *name = calloc (strlen (hostname) + 2, sizeof (uint8_t));

  /* Leave the first byte blank for the first field length */
  memcpy (name + 1, hostname, strlen (hostname));

  /* Example:
     +---+---+---+---+---+---+---+---+---+---+---+
     | a | b | c | . | d | e | . | c | o | m | \0|
     +---+---+---+---+---+---+---+---+---+---+---+

     becomes:
     +---+---+---+---+---+---+---+---+---+---+---+---+
     | 3 | a | b | c | 2 | d | e | 3 | c | o | m | 0 |
     +---+---+---+---+---+---+---+---+---+---+---+---+
   */

  uint8_t count = 0;
  uint8_t *prev = (uint8_t *)name;
  for (int i = 0; i < strlen (hostname); i++)
    {
      /* Look for the next ., then copy the length back to the
         location of the previous . */
      if (hostname[i] == '.')
        {
          *prev = count;
          prev = (uint8_t *)name + i + 1;
          count = 0;
        }
      else
        count++;
    }
  *prev = count;

  return name;
}

void
print_byte_block (uint8_t *bytes, size_t length)
{
  printf ("  ");
  for (int i = 0; i < length; i++)
    {
      printf ("%02x", bytes[i]);
      if (i == length - 1)
        printf ("\n");
      else if ((i + 1) % 16 == 0)
        printf ("\n  ");
      else if ((i % 2) != 0)
        printf (" ");
    }
  printf ("\n");
}

void
print_dns_response (uint8_t *response)
{
  /* First, check the header for an error response code */
  dns_header_t *response_header = (dns_header_t *)response;
  if ((ntohs (response_header->flags) & 0xf) != 0)
    {
      fprintf (stderr, "Failed to get response\n");
      return;
    }

  /* Reconstruct the question */
  uint8_t *start_of_question = response + sizeof (dns_header_t);
  dns_question_t *questions
    = calloc (sizeof (dns_question_t), response_header->ancount);
  for (int i = 0; i < ntohs (response_header->ancount); i++)
    {
      questions[i].name = (char *)start_of_question;
      uint8_t total = 0;
      uint8_t *field_length = (uint8_t *)questions[i].name;
      while (*field_length != 0)
        {
          total += *field_length + 1;
          *field_length = '.';
          field_length = (uint8_t *)questions[i].name + total;
        }
      questions[i].name++;
      /* Skip null byte, qtype, and qclass */
      start_of_question = field_length + 5;
    }

  /* The records start right after the question section. For each record,
     confirm that it is an A record (only type supported). If any are not
     an A-type, then return. */
  dns_record_a_t *records = (dns_record_a_t *)start_of_question;
  for (int i = 0; i < ntohs (response_header->ancount); i++)
    {
      printf ("Record for %s\n", questions[i].name);
      printf ("  TYPE: %" PRId16 "\n", ntohs (records[i].type));
      printf ("  CLASS: %" PRId16 "\n", ntohs (records[i].class));
      printf ("  TTL: %" PRIx32 "\n", ntohl (records[i].ttl));
      printf ("  IPv4: %08" PRIx32 "\n",
              ntohl ((uint32_t)records[i].addr.s_addr));
      printf ("  IPv4: %s\n", inet_ntoa (records[i].addr));
    }
}

uint8_t *
setup_dns_request (char *hostname, size_t *packetlen)
{
  /* Set up the DNS header */
  dns_header_t header;
  memset (&header, 0, sizeof (dns_header_t));
  header.xid= htons (0x1234);    /* Randomly chosen ID */
  header.flags = htons (0x0100); /* Q=0, RD=1 */
  header.qdcount = htons (1);    /* Sending 1 question */

  /* Set up the DNS question */
  dns_question_t question;
  question.dnstype = htons (1);  /* QTYPE 1=A */
  question.dnsclass = htons (1); /* QCLASS 1=IN */
  question.name = build_domain_qname (hostname);

  /* Copy all fields into a single, concatenated packet */
  *packetlen = sizeof (header) + strlen (hostname) + 2
               + sizeof (question.dnstype) + sizeof (question.dnsclass);
  uint8_t *packet = calloc (*packetlen, sizeof (uint8_t));
  uint8_t *p = (uint8_t *)packet;

  /* Copy the header first */
  memcpy (p, &header, sizeof (header));
  p += sizeof (header);

  /* Copy the question name, QTYPE, and QCLASS fields */
  memcpy (p, question.name, strlen (hostname) + 2);
  p += strlen (hostname) + 2;
  memcpy (p, &question.dnstype, sizeof (question.dnstype));
  p += sizeof (question.dnstype);
  memcpy (p, &question.dnsclass, sizeof (question.dnsclass));

  return packet;
}
```

« 5.7. Wireless Connectivity: Wi-Fi, Bluetooth, and Zigbee :: Contents :: 6.1. Concurrency with Multithreading »