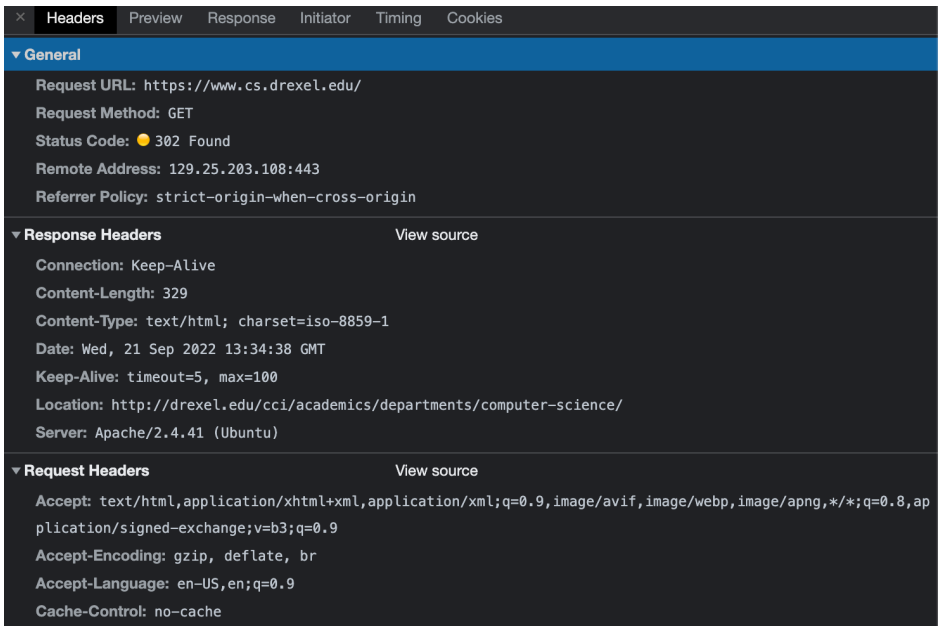
CS472
Computer Networks
Dr. Brian Mitchell

Homework 1
C/C++ Refresher

Introduction

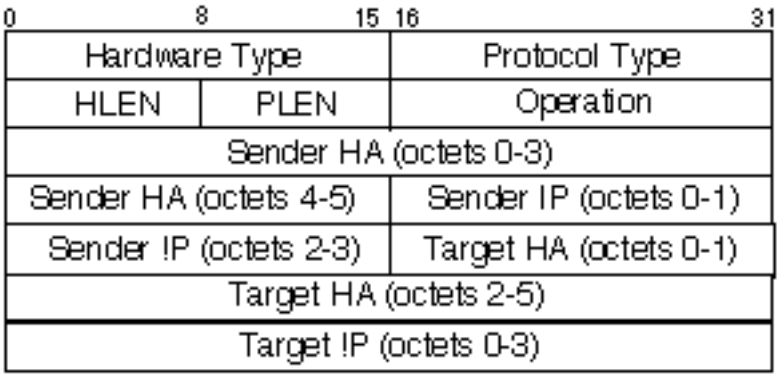
The objective of this assignment is to ensure you have a workable development environment setup and have the opportunity to refresh of your memory around how to do systems programming in C/C++. As I will be mentioning in our first lecture, **this is a hands-on course**. This first assignment will expose you to some of the things we will be doing throughout the class.

We will be looking at network protocols a good bit in CS472, and how they get layered on top of each other (e.g., encapsulated within each other). For example, each layer in a protocol stack will generally have a header, and a payload/data area, where the next layer gets encapsulated within the payload/data area of the previous. The lower layers of the stack, generally have very specific layouts, down to the bit level. For example, ethernet, IP and TCP/UDP have well defined headers with things aligned into bits, bytes and words. Higher level protocols such as FTP, HTTP use other more flexible headers. The example below shows the HTTP headers that I captured going to https://www.cs.drexel.edu – notice they are key-value pairs, terminated by newlines ("\n").



We will get to that later, for now the goal is to develop comfort with parsing and working with headers that have a well-defined binary format. For this homework we will be digging into the ARP protocol. We will cover ARP in week 1, but for this assignment knowing ARP is not

important as the objective is to just dust off the systems programming skills. The following picture shows the layout of the ARP protocol header. If you count the number of bytes in the ARP protocol its 28 bytes, organized into seven 32 bit words.



ARP is described in RFC 826 - https://www.rfc-editor.org/rfc/rfc826.html. Ill spare you the time of reading the RFC for this assignment. For now, some of the fields have specific constant values to indicate that this is an ARP header. These are:

- Hardware Type = 1 (e.g., 0x0001) – this indicates that ARP is flowing over ethernet, which is largely the defacto standard, you can google to find out other values.
- Protocol Type = 0x0800 – this indicates that the protocol is ARP with IPv4
- HLEN = 6 - Length of the hardware address – which for now is 48 bits, or 6 bytes. This is the mac address we will be covering in week one.
- PLEN = 4 – Lenth of the protocol address – IPv4 addresses are 32 bits or 4 bytes
- Operation = {1,2}. The value of 1 indicates that this is an ARP-REQUEST, the value of 2 indicates that this is an ARP-RESPONSE. ARP makes requests, and waits for responses so that's why its 2 flavors
- Sender HA = the MAC Address of the Sender. During a request it's the MAC address of your computer – its 48 bits, or 6 bytes
- Sender IP = This is your IP address, represented as a 32 bit number – its 4 bytes
- Target HA = the MAC address of the destination. On a request, you might not know the targets MAC address, in this case the broadcast mac address is 0xFFFFFFFFFFFF. If you know the MAC address it goes here, a valid MAC address will always supplied in a reply.
- Target IP = The IP address of the target machine. On a request it can also be a broadcast, which would be 0xFFFF. If you know the IP of the target you should put it here. A valid target IP address would always be included if the ARP Operation is a response.

What you need to do

Below im going to give you a header file that sets up the basic data structure:

```c
#ifndef ARPHEADERS_H_INCLUDED
#define ARPHEADERS_H_INCLUDED

#include <stdint.h>

#define ETH_P_ARP 0x0806 /* Address Resolution packet */
#define ARP_HTYPE_ETHER 1  /* Ethernet ARP type */
#define ARP_PTYPE_IPv4 0x0800 /* Internet Protocol packet */
#define ETH_ALEN    6   /* Ethernet MAC addresses are 6 octects - 48 bytes */
#define IP4_ALEN    4   /* Ethernet MAC addresses are 4 octets - 32 bytes */
#define ARP_REQ_OP   1   /* From RFC - Request Op - 1 */
#define ARP_RSP_OP   2   /* From RFC - Response Op - 2 */

typedef uint32_t ipaddress_t;
typedef uint8_t  macaddress_t[ETH_ALEN];

/* Ethernet frame header */
typedef struct {
   uint8_t dest_addr[ETH_ALEN]; /* Destination hardware address */
   uint8_t src_addr[ETH_ALEN];  /* Source hardware address */
   uint16_t frame_type;   /* Ethernet frame type */
} ether_hdr;

/* Ethernet ARP packet from RFC 826 */
typedef struct arp_ether_ipv4{
   uint16_t htype;   /* Format of hardware address */
   uint16_t ptype;   /* Format of protocol address */
   uint8_t hlen;    /* Length of hardware address */
   uint8_t plen;    /* Length of protocol address */
   uint16_t op;    /* ARP opcode (command) */
   uint8_t sha[ETH_ALEN];  /* Sender hardware address */
   uint8_t spa[IP4_ALEN];   /* Sender IP address */
   uint8_t tha[ETH_ALEN];  /* Target hardware address */
   uint8_t tpa[IP4_ALEN];   /* Target IP address */
} arp_ether_ipv4;

#endif
```

Directions:

1. Get the sample code that I have provided for you here: https://github.com/ArchitectingSoftware/CS472-Class-Files/tree/main/hw1-shell. These files give you a scaffold for what I want you to do.

2. The readme file describes your objective. I am providing 3 binary arrays that need to be decoded into arp structures, then I want these arp structures converted to a human readable string and printed out.
3. I am also providing each example in 2 formats, one is encoded as a byte array, the other as a word array. We will see why I want you to do each example twice.
4. As the readme states, you will need to hand in your code, and sample output. You can modify my code in any way that you wish, add functions, remove my functions, etc.

In order to help you, the output of your program for the ex1b and ex1w arrays should look something like the below:

```
→  header-decoding ./decoder
BYTE-ARRAY TO ARP

ARP PACKET DETAILS
     htype:     0x0001
     ptype:     0x0800
     hlen:      6
     plen:      4
     op:        1
     spa:       192.168.1.51
     sha:       01:02:03:04:05:06
     tpa:       192.168.1.1
     tha:       aa:bb:cc:dd:ee:ff


WORD-ARRAY TO ARP

ARP PACKET DETAILS
     htype:     0x0001
     ptype:     0x0800
     hlen:      6
     plen:      4
     op:        1
     spa:       192.168.1.51
     sha:       01:02:03:04:05:06
     tpa:       192.168.1.1
     tha:       aa:bb:cc:dd:ee:ff
```

The above represents the correct output, however, I would not be surprised if your output for the first example looks like the below:

```
→  header-decoding ./decoder
BYTE-ARRAY TO ARP

ARP PACKET DETAILS
     htype:     0x0001
     ptype:     0x0800
     hlen:      6
     plen:      4
     op:        1
     spa:       192.168.1.51
     sha:       01:02:03:04:05:06
     tpa:       192.168.1.1
     tha:       aa:bb:cc:dd:ee:ff


WORD-ARRAY TO ARP

ARP PACKET DETAILS
     htype:     0x0100
     ptype:     0x0008
     hlen:      4
     plen:      6
     op:        256
     spa:       168.192.51.1
     sha:       02:01:04:03:06:05
     tpa:       168.192.1.1
     tha:       bb:aa:dd:cc:ff:ee
```

Notice how the byte version is correct, but the word version is not correct (but it seems somewhat close). Some fields are correct, others have some things reversed. For the purposes of grading if you get the output I show above, I will provide full credit. We will talk about network byte order during class and then it will become clear what is going on.

EXTRA CREDIT

In order to receive extra credit please answer the following questions:

1. Why do you think I would expect most of you to get the incorrect output (shown in the second picture)? Why is this happening?
2. Suggest how you would fix the problem I show above so that you get the correct output for both the byte and word formats?
3. Build the code and put a comment in there around the extra credit and the code you added to address this problem. Note that this could be fixed in as little as one line of code, but if you need a few lines, that's OK. If you have to write something more than 7 lines of code, you are probably doing it wrong.