Received: 03/08/2016
Completed 03-22-2016

CS 3423.01

Assignment 6 : Reading Network Traffic from a Tcpdump File
Due 12:00 Midnight, Monday March 28, 2016

Write two programs, `logprt.c` and `prtlog.c`, in C using only low-level I/O to read the data, which will read all of the packets in a tcpdump file, one at a time, printing out the type of packet for each packet read. While only low-level I/O can be used to read the data high-level I/O can be used to write output and error messages including using `perror`.

An example invocation:

```
logprt network.log
prtlog network.log
```

There will be example data files (tcpdump log files) in our course directory:

`/usr/local/courses/cs3423/assign6/data/network*.log`

Some of these files only have a few packets of specific types while `network.log` is a large file with a lot of different packets.

Tcpdump Binary File Structure

Each tcpdump binary file contains an initial header of 24 bytes which contains the basic information about the file. The remainder of the file is simply a sequence of packets each with a packet header which gives the length of the packet. Since you know the size of the tcpdump header and the size of each packet header (which gives the size of the following packet) it is a simple matter to read through the entire file one packet at at time.

The tcpdump header includes the magic constant for the file type which is given below. You may assume that the magic file type is `PCAP_MAGIC` for the files that you are given to run. (you can use `hexedit` or `xxd` to look at the tcpdump file).

```
/* magic constants for various pcap file types */
#define PCAP_MAGIC                  0xa1b2c3d4
#define PCAP_SWAPPED_MAGIC          0xd4c3b2a1
#define PCAP_MODIFIED_MAGIC         0xa1b2cd34
#define PCAP_SWAPPED_MODIFIED_MAGIC 0x34cdb2a1
```

The structure of the data at the beginning of the file is given by the `pcap_file_header` structure:

```
/* pcap data stored once at the beginning of the file */
struct pcap_file_header {
  u_int32_t magic;          /* magic file header */
  u_int16_t version_major;  /* major number */
  u_int16_t version_minor;  /* minor number */
  u_int32_t thiszone;       /* gmt to local correction */
  u_int32_t sigfigs;        /* accurary of timestamp */
  u_int32_t snaplen;        /* max length of saved portion of packet */
  u_int32_t linktype;       /* data link type */
};
```

The file then consists of a sequence of packets, each packet has a prefix header which is described by the `pcap_pkthdr` header which you will NOT use (see below):

```
/* data prefixing each packet */
struct pcap_pkthdr {
  struct timeval ts;      /* timestamp */
  u_int32_t caplen;       /* captured length of the packet */
  u_int32_t len;          /* actual length of the packet */
};
```

There is a slight problem since, with 64 bit systems, you may get 64 or 64/32 bit timeval systems but on the other hand libpcap will always write a 32 bit timeval system in tcpdump log files. This means that if you use the system timeval struct, if it is a 64 bit timeval struct, the caplen and len variables will be incorrect if you simply read. The simple solution is to use your own packet header. So in place of `struct timeval` and `struct pcap_pkthdr` you MUST use:

```
struct timev {
  unsigned int tv_sec;
  unsigned int tv_usec;
};

/* data prefixing each packet */
struct my_pkthdr {
  struct timev ts;
  int caplen;
  int len;
};
```

At this point you should write `logprt.c` (without timing info) which would read in the entire file, printing out the tcpdump header and then reading in each packet header followed by reading in the actual packet into a buffer. For each packet, count and print out the packet number as well as the captured length and the actual length of the packet similar to:

```
Packet 3
Captured Packet Length = 60
Actual Packet Length = 60
```

Once this is working you want to print out the time of each packet relative to the first packet. The code is below.

In order to print the time of each packet relative to the first packet we will use the variable `firsttime` to identify the first packet and select the absolute time of the first packet from its packet header (`phdr`). Thus the first packet has time 0 and all other packets have a time relative to the first packet time. In order to print the time of a packet the following code using static variables (properly initialized) works:

```
if ( firsttime ) {
  firsttime = 0;
  b_sec = phdr.ts.tv_sec;
  b_usec = phdr.ts.tv_usec;
}
```

```
c_sec = (unsigned)phdr.ts.tv_sec - b_sec;
c_usec = (unsigned)phdr.ts.tv_usec - b_usec;
while (c_usec < 0) {
  c_usec += 1000000;
  c_sec--;
}
printf("%05u.%06u\n", (unsigned)c_sec, (unsigned)c_usec);
```

The variables `b_sec` and `c_sec` should be unsigned ints while `b_usec` and `c_usec` should be ints.

Once this code is included then you are done with `logprt.c` modulo debugging.

Header Files

For this stage you simply need the declarations given above in a local header file, `logprt.h`. The header files that I include are

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include "logprt.h"
```

Packets

You need to be able to skip over all unknown packets. This is simple since you know the packet length. By now you should be reading through the entire file one packet at a time printing out the packet number, starting with 0, and the relative time just to be certain that you can do that.

Sample output

```
PCAP_MAGIC
Version major number = 2
Version minor number = 4
GMT to local correction = 0
Timestamp accuracy = 0
Snaplen = 65535
Linktype = 1

Packet 0
00000.000000
Captured Packet Length = 60
Actual Packet Length = 60

Packet 1
00000.000036
Captured Packet Length = 60
Actual Packet Length = 60
```

```
Packet 2
00002.098395
Captured Packet Length = 91
Actual Packet Length = 91

Packet 3
00002.098401
Captured Packet Length = 60
Actual Packet Length = 60
```

Then write `prtlog.c` by extending `logprt.c` by printing out the packet type. This can be seen in the ethernet header which is at the beginning of the packet and is given by the ethernet header:

```
struct eth_hdr {
        eth_addr_t      eth_dst;        /* destination address */
        eth_addr_t      eth_src;        /* source address */
        uint16_t        eth_type;       /* payload type */
};
```

and the ethernet types are given by

```
#define ETH_TYPE_PUP     0x0200          /* PUP protocol */
#define ETH_TYPE_IP      0x0800          /* IP protocol */
#define ETH_TYPE_ARP     0x0806          /* address resolution protocol */
#define ETH_TYPE_REVARP 0x8035           /* reverse addr resolution protocol */
#define ETH_TYPE_8021Q   0x8100          /* IEEE 802.1Q VLAN tagging */
#define ETH_TYPE_IPV6    0x86DD          /* IPv6 protocol */
#define ETH_TYPE_MPLS    0x8847          /* MPLS */
#define ETH_TYPE_MPLS_MCAST     0x8848   /* MPLS Multicast */
#define ETH_TYPE_PPPOEDISC      0x8863   /* PPP Over Ethernet Discovery Stage */
#define ETH_TYPE_PPPOE   0x8864          /* PPP Over Ethernet Session Stage */
#define ETH_TYPE_LOOPBACK       0x9000   /* used to test interfaces */
```

In order to check the ethernet type, because the network is big endian and the x86 architecture is little endian and the ethernet types are short int (16 bits) you need to switch the bytes from the packet before checking. This is done via the network-to-host-short function, `ntohs()` in the following manner:

```
if ( ntohs(ethhdr->eth_type) == ETH_TYPE_IP )
```

Unless the packet is either an IP or an ARP packet (ETH_TYPE_IP or ETH_TYPE_ARP) you will simply skip over the packet after printing the type (i.e. read the next packet).

If the packet is an ARP packet type then immediately following the ethernet header is the arp header which is:

```
/*
 * ARP header
 */
struct arp_hdr {
        uint16_t        ar_hrd; /* format of hardware address */
        uint16_t        ar_pro; /* format of protocol address */
        uint8_t         ar_hln; /* length of hardware address (ETH_ADDR_LEN) */
        uint8_t         ar_pln; /* length of protocol address (IP_ADDR_LEN) */
        uint16_t        ar_op;  /* operation */
};
```

For all arp packets print the operation. These are

```
/*
 * ARP operation
 */
#define ARP_OP_REQUEST          1       /* request to resolve ha given pa */
#define ARP_OP_REPLY            2       /* response giving hardware address */
#define ARP_OP_REVREQUEST       3       /* request to resolve pa given ha */
#define ARP_OP_REVREPLY         4       /* response giving protocol address */
```

On the other hand if the packet is an IP packet then the ethernet header is followed by an ip header without options on a little endian system:

```
/*
 * IP header, without options
 */
struct ip_hdr {
        uint8_t         ip_hl:4,        /* header length (incl any options) */
                        ip_v:4;         /* version */
        uint8_t         ip_tos;         /* type of service */
        uint16_t        ip_len;         /* total length (incl header) */
        uint16_t        ip_id;          /* identification */
        uint16_t        ip_off;         /* fragment offset and flags */
        uint8_t         ip_ttl;         /* time to live */
        uint8_t         ip_p;           /* protocol */
        uint16_t        ip_sum;         /* checksum */
        ip_addr_t       ip_src;         /* source address */
        ip_addr_t       ip_dst;         /* destination address */
};
```

The protocols that we will look for are:

```
#define IP_PROTO_ICMP           1               /* ICMP */
#define IP_PROTO_IGMP           2               /* IGMP */
#define IP_PROTO_TCP            6               /* TCP */
#define IP_PROTO_UDP            17              /* UDP */
```

Any unrecognized protocol should be printed as "UNRECOGNIZED".

The final output should look like:

```
PCAP_MAGIC
Version major number = 2
Version minor number = 4
GMT to local correction = 0
Timestamp accuracy = 0
Snaplen = 65535
Linktype = 1

Packet 0
00000.000000
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   ARP
      arp operation = Arp Request
Packet 1
00000.000036
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   ARP
      arp operation = Arp Reply

Packet 2
00002.098395
Captured Packet Length = 91
Actual Packet Length = 91
Ethernet Header
   IP
      TCP

Packet 3
00002.098401
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   IP
      TCP

Packet 4
00002.129106
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   IP
      TCP
```

```
Packet 5
00002.838747
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   IP
      ICMP



Packet 6
00002.838819
Captured Packet Length = 60
Actual Packet Length = 60
Ethernet Header
   IP
      ICMP
        .
        .
        .
Packet 14
00006.397454
Captured Packet Length = 82
Actual Packet Length = 82
Ethernet Header
   IP
      UDP
```

Obviously we are not printing out all of the data in the packet. There are some issues since the packet is big endian while the intel architecture is little endian which slightly complicates printing out multibyte values. If you want to write portable network packages you will want to look at the libraries, libpcap, libdnet and libnet.

Header Files

Now, instead of putting all of the data structures for the packets in a local header we will use the header files from `libpcap/tcpdump` and `libdnet` packages. These are tarred up in the file

`/usr/local/courses/cs3423/assign6/headers.tar.gz`

Just untar these in your assign6 directory where your source code is and then include them.

Thus your includes should be

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <unistd.h>
#include "pcap.h"
#include "dnet.h"
#include "prtlog.h"
```

The local header file `prtlog.h` should include the prototypes of the functions used in `prtlog.c` and the PCAP magic constants defined at the beginning of the assignment.

Submission

Clean up your assign6 directory so that it only includes `logprt.c`, `prtlog.c` and the header files. The assign6 directory should then be turned in to project6 using the command:

```
submit assign6 project6 3423
```

After the "cutoff" date of this project, the project may be submitted to project6-late for one additional week. There will be a 10% penalty associated with a late submission. The state of all projects can be seen with the command

```
projects cs3423
```