CNT5505 Programming Assignment No. 4: Internet Packet Analyzer

(This is an individual assignment. It must be implemented in C++ or C)

.

PURPOSE

- Experience with packet analyzing and Internet packet formats.

DESCRIPTION

In this assignment, you will implement an Internet packet analyzer that processes the dump files produced by a home-made packet sniffer, ethernetdump.c (given in the class, but you do not need to understand it to do the assignment). The analyzer reads packets from a dump file, parses packet headers, accumulates packet statistics, and outputs the statistics and packet information. The dump file is a binary file with the following format:

```
frame_size Ethernet_frame .... frame_size Ethernet_frame
```

The `frame_size` is a 4-byte integer in the network order format (you need to use the ntohl() routine to convert this number into the host format). Each Ethernet frame contains Ethernet header and Ethernet payload (which may contain the entire packet in an upper layer protocol such as IP and ARP). Notice that the dump file is a binary file that does not have any delimiter between data items.

The packet analyzer examines each packet in the dump file and identifies Ethernet, IP, ARP, TCP, UDP, and ICMP packets (other packets are classified as unknown packets). Depending on the command line options, different levels of statistics and the packet information will be printed. If the analyzer is executed without any flags, it will print a summary of the packets in the dump file. The following is an example.

```
<linprog3:947> ./a.out dumpfile.5000
Ethernet frames:        5000
Ethernet broadcast:     346
  ARP packets:          335
  IP packets:           4523
    UDP packets:        27
    TCP packets:        4486
    ICMP packets:       10
    other IP packets:   0
  other packets:        142
<linprog3:948>
```

The analyzer should support the flags, "-c <numofpackets>", "-v", and "-V".

-c <numofpackets>

The number of frames to be processed can be modified with a "-c" flag. For example "./a.out -c 200 dumpfile.5000" will process the first 200 frames in the file and print the summary.

-v

This option turns on the basic verbose mode, which will print one line for every packet. Your program must be able to handle all cases handled by the sample executable. The following example shows the cases you must handle.

```
unknown packet (01:80:c2:00:00:00, 00:04:9a:36:38:9c, 0:26)
```

```
192.168.16.1 -> (broadcast) (ARP) who is 192.168.16.190
192.168.10.151 -> 192.168.10.154 (ARP) who is 192.168.10.154
192.168.1.25 -> 192.168.1.1 (ARP) 192.168.1.25's hardware address is 0:10:18:b:58:80
192.168.16.103 -> 192.168.23.10 (ICMP), Echo Reply (type=0)
128.186.120.2 -> 192.168.16.103 (TCP) sourceport = 22 destport = 42170
192.168.16.103 -> 128.186.120.179 (UDP) sourceport = 32790 destport = 53
```

-V

This option turns on the extended verbose mode. With this option, the program will print packet headers in detail.

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 0
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 1:80:c2:0:0:0
ETHER:  Source      = 0:6:28:f2:55:1
ETHER:  Ethertype   = 0026 (unknown)
ETHER:

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 400
ETHER:  Packet size = 60 bytes
ETHER:  Destination = ff:ff:ff:ff:ff:ff
ETHER:  Source      = 0:e0:b6:1:bd:33
ETHER:  Ethertype   = 0806 (ARP)
ETHER:
ARP:  ----- ARP Frame -----
ARP:
ARP:  Hardware type = 1 (Ethernet)
ARP:  Protocol type = 0800 (IP)
ARP:  Length of hardware address = 6 bytes
ARP:  Length of protocol address = 4 bytes
ARP:  Opcode 1 (ARP Request)
ARP:  Sender's hardware address = 0:e0:b6:1:bd:33
ARP:  Sender's protocol address = 192.168.16.1
ARP:  Target hardware address = ?
ARP:  Target protocol address = 192.168.16.174
ARP:

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 398
ETHER:  Packet size = 60 bytes
ETHER:  Destination = 0:b:db:a7:da:98
ETHER:  Source      = 0:e0:b6:1:bd:33
ETHER:  Ethertype   = 0800 (IP)
ETHER:
IP:  ----- IP Header -----
IP:
IP:  Version = 4
IP:  Header length = 20 bytes
```

```
IP:  Type of service = 0
IP:  Total length = 28 bytes
IP:  Identification = 0
IP:  Flags
IP:    .1.. .... = do not fragment
IP:    ..0. .... = last fragment
IP:  Fragment offset = 0 bytes
IP:  Protocol = 1 (ICMP)
IP:  Header checksum = 91ec
IP:  Source address = 192.168.23.10
IP:  Destination address = 192.168.16.154
IP:  No options
IP:
ICMP:  ----- ICMP Header -----
ICMP:
ICMP: Type = 8 (Echo Request)
ICMP: Code = 0
ICMP: Checksum = aa72
ICMP: Identifier = 19852
ICMP: Sequence number = 1
ICMP:

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 404
ETHER:  Packet size = 92 bytes
ETHER:  Destination = ff:ff:ff:ff:ff:ff
ETHER:  Source      = 0:b:db:a7:da:7d
ETHER:  Ethertype   = 0800 (IP)
ETHER:
IP:  ----- IP Header -----
IP:
IP:  Version = 4
IP:  Header length = 20 bytes
IP:  Type of service = 0
IP:  Total length = 78 bytes
IP:  Identification = 54795
IP:  Flags
IP:    .0.. .... = allow fragment
IP:    ..0. .... = last fragment
IP:  Fragment offset = 0 bytes
IP:  Protocol = 17 (UDP)
IP:  Header checksum = c1aa
IP:  Source address = 192.168.16.153
IP:  Destination address = 192.168.16.255
IP:  No options
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 137
UDP:  Destination port = 137
UDP:  Message length = 58
```

```
UDP:  Checksum = 53df
UDP:

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 468
ETHER:  Packet size = 66 bytes
ETHER:  Destination = 0:13:72:1e:46:7c
ETHER:  Source      = 0:e0:b6:1:bd:33
ETHER:  Ethertype   = 0800 (IP)
ETHER:
IP:  ----- IP Header -----
IP:
IP:  Version = 4
IP:  Header length = 20 bytes
IP:  Type of service = 10
IP:  Total length = 52 bytes
IP:  Identification = 61271
IP:  Flags
IP:    .1.. .... = do not fragment
IP:    ..0. .... = last fragment
IP:  Fragment offset = 0 bytes
IP:  Protocol = 6 (TCP)
IP:  Header checksum = 8190
IP:  Source address = 128.186.120.2
IP:  Destination address = 192.168.16.103
IP:  No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 22
TCP:  Destination port = 42170
TCP:  Sequence number = 4124809487
TCP:  Acknowledgement number = 888630597
TCP:  Data offset = 32 bytes
TCP:  Flags
TCP:       ..0. .... = No urgent pointer
TCP:       ...1 .... = Acknowledgement
TCP:       .... 0... = No push
TCP:       .... .0.. = No reset
TCP:       .... ..0. = No Syn
TCP:       .... ...0 = No Fin
TCP:  Window = 8672
TCP:  Checksum = 1d2e
TCP:  Urgent pointer = 0
TCP:  Options ignored
TCP:
```

Your progrm should print exactly the same as the sample executable. To ensure that your program produces the same output as the sample program, the following output format should be followed.

- An Ethernet address should be printed as 6 hex numbers (for the six bytes) separated by ':'.
- Ethernet *type* (Ethertype) should be printed as a 4-digit hex number (fill with 0's if necessary).

The program should identify IP packets (Ethertype=0x0800) and ARP packets (Ethertype = 0x0806).

- IP *identification* should be printed as a decimal number. IP *header checksum* should be printed as a hex number. The program should identify TCP packets (protocol = 6) and UDP packets (protocol = 17). If the header length is 20 bytes, there is no IP option in the header (IP: No options). Otherwise the program should ignore the options (IP: Options ignored).
- UDP *checksum* should be printed as a hex number and UDP *message length* should be printed as a decimal number.
- TCP *sequence* number and *acknowledgement* number should be printed as unsigned decimal numbers. TCP `window` should be printed as a decimal number and TCP `checksum` should be printed as a hex number. If TCP header length is 20 bytes, there is no TCP option in the header (TCP: No options). Otherwise the program should ignore the options (TCP: Options ignored).
- ICMP checksum should be printed as a hex number, identifier and sequence number should be printed as decimal numbers.

DEADLINE AND MATERIALS TO BE HANDED IN

Deadline: December 6.

You should clean and tar your assignment directory before you submit the project. In the directory, you should have all the needed files to create the executable of the program. You program should work on linprog. You should also have a README file describing how to compile your program and the known bugs in your program.

GRADING POLICY

You program will be graded by comparing (with the diff command) the output from your program with that from the sample executable for different input dumpfiles. **Any difference** will result in the deduction of 20% of the points before further deductions are considered based on functionality.

1. Correct execution with no flags (40)
2. Correct implementation of the "-v" flag (25)
3. Correct implementation of the "-V" flag (25)
4. Correct implementation of the "-c" flag (10)
5. Extra 5 points for being the first to report a bug in the sample executable.

Misc.

To do this assignment, you will know in detail different packet formats including some values and the meanings of the values in header fields. The needed information is much more detailed than what was taught in the class: CNT5505 should have taught enough for you to find the necessary information; and **finding necessary information on your own is a part of this project**.

Your can find the packet for Ethernet, ARP, IP, TCP, UDP, and ICMP from the book and from Internet resources.

TCP/IP protocol standards are published in a series of documents known as *Request for Comments* (RFCs), which are available from the Internet Engineering Task Force (IETF) at:

http://www.ietf.org/rfc.html

The protocol standard for IP is in RFC 791; TCP is specified in RFC 793; UDP is specified in RFC 768. IP and TCP protocols are updated in RFC 1122.