# Учебный проект «Разработка консольного анализатора протоколов»

# Задание на практическое занятие 2.1.

Разработка приложения анализа трафика вычислительной сети, хранящегося в файле libpcap формата.

Исходные данные: pcap-файл с записью трафика. Формат файла – см. документацию по библиотеке libpcap (или просто файл pcap.h).

1. Открыть файл, считать его заголовок, подсчитать количество пакетов, вычислить длины пакетов (минимальная максимальная, средняя).
2. Вывести полученные данные на экран.
3. Считать из файла сетевые пакеты.
4. Подсчитать, сколько пакетов IP\не IP, для пакетов с IP подсчитать, сколько пакетов с транспортным протоколом UDP\TCP\другой. Вывести результаты на экран.
5. Закрыть файл.

Требования к реализации:

Для файлового ввода-вывода и работы со строками использовать стандартную библиотеку языка С.

## Приложение 1. Структура libpcap-файла и пакетов Ethernet.

struct pcap\_file\_header {

bpf\_u\_int32 magic;

u\_short version\_major;

u\_short version\_minor;

bpf\_int32 thiszone; /\* gmt to local correction \*/

bpf\_u\_int32 sigfigs; /\* accuracy of timestamps \*/

bpf\_u\_int32 snaplen; /\* max length saved portion of each pkt \*/

bpf\_u\_int32 linktype; /\* data link type (LINKTYPE\_\*) \*/

};

typedef struct timeval {

long tv\_sec;

long tv\_usec;

};

**tv\_sec**

Time interval, in seconds.

**tv\_usec**

Time interval, in microseconds. This value is used in combination with the **tv\_sec** member to represent time interval values that are not a multiple of seconds.

struct pcap\_pkthdr {

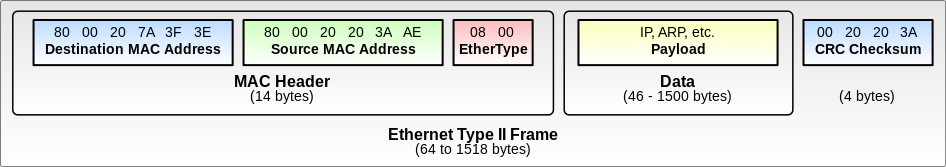
struct timeval ts; /\* time stamp \*/

bpf\_u\_int32 caplen; /\* length of portion present \*/

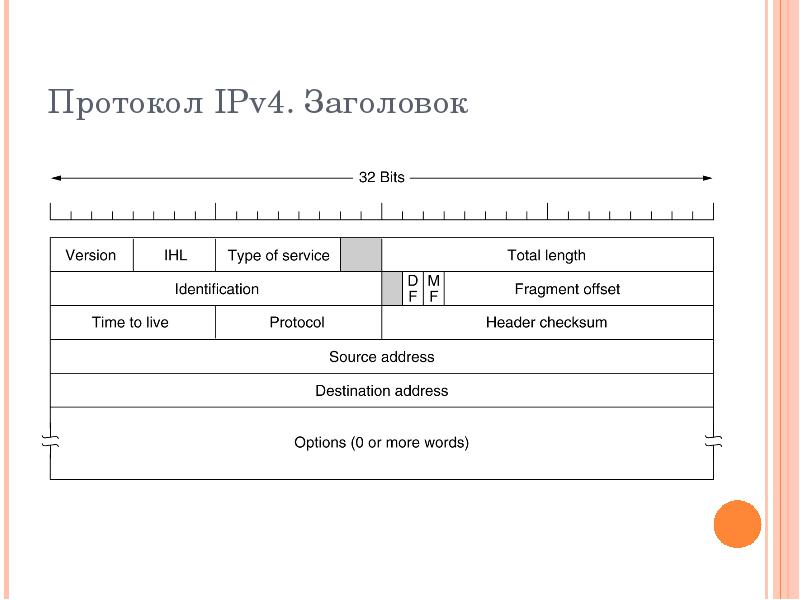
bpf\_u\_int32 len; /\* length this packet (off wire) \*/

};

Пакет Ethernet



Пакет IP



Интерпретация поля Protocol

|  |  |
| --- | --- |
| 1 | ICMP |
| 6 | TCP |
| 17 | UDP |
| И ещё 253 значения | |

#### Приложение 2. Потоковый ввод-вывод в стиле С

Особенностью С является отсутствие в этом языке структурированных файлов. Все файлы рассматриваются как не структурированная последовательность байтов. При таком подходе понятие файла распространяется и на различные устройства.

В С существуют средства ввода-вывода. Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке С. Библиотека С поддерживает три уровня ввода-вывода:

1. потоковый ввод-вывод;
2. ввод-вывод нижнего уровня;
3. ввод-вывод для консоли и портов (зависит от ОС).

Поток– это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Чтение данных из потока называется извлечением*,* вывод в поток – помещением*,* или включением*.*

Поток определяется как последовательность байтов и не зави­сит от конкретного устройства, с которым производится обмен (оперативная па­мять, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную об­ласть оперативной памяти — буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

выходной

поток

буфер

буфер

входной

поток

программа

Чтение из потока

Помещение в поток

При работе с потоком можно:

* Открывать и закрывать потоки (связывать указатели на поток с конкретными файлами);
* вводить и выводить строку, символ, форматированные данные, порцию данных произвольной длины;
* анализировать ошибки ввода-вывода и достижения конца файла;
* управлять буферизацией потока и размером буфера;
* получать и устанавливать указатель текущей позиции в файле.

Функции библиотеки ввода-вывода находятся в заголовочном файле <stdio.h>.

Прежде чем начать работать с потоком, его надо инициировать, т. е. открыть. При этом поток связывается со структурой предопределенного типа FILE, определение которой находится в библиотечном файле <stdio.h>. В структуре находится указатель на буфер, указатель на текущую позицию файла и т. п. При открытии потока, возвращается указатель на поток, т. е. на объект типа FILE.

#include <stdio.h>;

. . . . . . . .

FILE \*fp;

. . . . . . . . . . ..

fp= fopen( ”t.txt”, ”r”);

где fopen(<имя\_файла>,<режим\_открытия>) - функция для инициализации файла.

Существуют следующие режимы для открытия файла:

|  |  |
| --- | --- |
| Режим | Описание режима открытия файла |
| r | Файл открывается для чтения, если файл не существует , то выдается ошибка при исполнении программы. |
| w | Файл открывается для записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается. |
| a | Файл открывается для добавления, если фай не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла |
| r+ | Файл открывается для чтения и записи, изменить размер файла нельзя, если файл не существует , то выдается ошибка при исполнении программы. |
| w+ | Файл открывается для чтения и записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается. |
| a+ | Файл открывается для чтения и записи, если фай не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла |

Поток можно открыть в текстовом (t) или двоичном (b) режиме. По умолчанию используется текстовый режим. В явном виде режим указывается следующим образом:

* r+bили rb - двоичный (бинарный) режим;
* r+t или rt – текстовый режим.

В файле stdio.h определена константа EOF, которая сообщает об окончании файла (отрицательное целое число).

При открытии потока могут возникать следующие ошибки:

* файл, связанный с потоком не найден (при чтении из файла);
* диск заполнен (при записи);
* диск защищен от записи (при записи) и т. п.

В этих случаях указатель на поток приобретет значение NULL (0). Указатель на поток, отличный от аварийного не равен 0.

Для вывода об ошибке при открытии потока используется стандартная библиотечная функция из файла <stdio.h>

void perror (const char\*s);

if ((fp=fopen(”t.txt”, ”w)==NULL)

{

// выводит строку символов с сообщением об ошибке

perror(\nошибка при открытии файла);

exit(0);

}

После работы с файлом, его надо закрыть

fclose(<указатель\_на\_поток>);

Когда программа начинает выполняться, автоматически открываются несколько потоков, из которых основными являются:

* стандартный поток ввода (stdin);
* стандартный поток вывода (stdout);
* стандартный поток вывода об ошибках (stderr).

По умолчанию stdin ставится в соответствие клавиатура, а потокам stdout и stderr - монитор. Для ввода-вывода с помощью стандартных потоков используются функции:

* getchar()/putchar() – ввод-вывод отдельного символа;
* gets()/puts() – ввод-вывод строки;
* scanf()/printf() – форматированный ввод/вывод.

Аналогично работе со стандартными потоками выполняется ввод-вывод в потоки, связанные с файлами.

Для символьного ввода-вывода используются функции:

* int fgetc(FILE\*fp), где fp – указатель на поток, из которого выполняется считывание. Функция возвращает очередной символ в форме int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.
* int fputc(int c, FILE\*fp), где fp – указатель на поток, в который выполняется запись, c – переменная типа int, в которой содержится записываемый в поток символ. Функция возвращает записанный в поток fp символ в форме int . Если символ не может быть записан, то возвращается значение EOF.

Для построчного ввода-вывода используются следующие функции:

* char\* fgets(char\* s,int n,FILE\* f), где char\*s – адрес, по которому размещаются считанные байты, int n – количество считанных байтов, FILE\* f – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи n-1 байтов или при получении управляющего символа ‘\n’. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается ‘\0’. При успешном завершении работы функция возвращает указатель на прочитанную строку, при неуспешном – 0.

* int puts(char\* s, FILE\* f), где char\*s – адрес, из которого берутся записываемые в файл байты, FILE\* f – указатель на файл, в который производится запись.

Символ конца строки (‘\0’) в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Для блокового ввода-вывода используются функции:

* int fread(void\*ptr,int size, int n, FILE\*f), где void\*ptr – указатель на область памяти, в которой размещаются считанные из файла данные, int size – размер одного считываемого элемента, int n – количество считываемых элементов, FILE\*f – указатель на файл, из которого производится считывание.

В случае успешного считывания функция возвращает количество считанных элементов, иначе – EOF.

* int fwrite(void\*ptr,int size, int n, FILE\*f), где void\*ptr – указатель на область памяти, в которой размещаются считанные из файла данные, int size – размер одного записываемого элемента, int n – количество записываемых элементов, FILE\*f – указатель на файл, в который производится запись.

В случае успешной записи функция возвращает количество записанных элементов, иначе – EOF.

В некоторых случаях информацию удобно записывать в файл без преобразования, т. е. в символьном виде пригодном для непосредственного отображения на экран. Для этого можно использовать функции форматированного ввода-вывода:

* int fprintf(FILE \*f, const char\*fmt,. . .) , где FILE\*f – указатель на файл, в который производится запись, const char\*fmt – форматная строка, . . . – список переменных, которые записываются в файл.

Функция возвращает число записанных символов.

* int fscanf(FILE \*f, const char\*fmt, par1,par2, . . .) , где FILE\*f – указатель на файл, из которого производится чтение, const char\*fmt – форматная строка, par1,par2,. . . – список переменных, в которые заносится информация из файла.

Функция возвращает число переменных, которым присвоено значение.

Средства прямого доступа дают возможность перемещать указатель текущей позиции в потоке на нужный байт. Для этого используется функция

* int fseek(FILE \*f, long off, int org), где FILE \*f - – указатель на файл, long off – позиция смещения, int org – начало отсчета.

Смещение задается выражение или переменной и может быть отрицательным, т. е. возможно перемещение как в прямом, так и в обратном направлениях. Начало отсчета задается одной из определенных в файле <stdio.h> констант:

SEEK\_SET ==0 – начало файла;

SEEK\_CUR==1 – текущая позиция;

SEEK\_END ==2 – конец файла.

Функция возвращает 0, если перемещение в потоке выполнено успешно, иначе возвращает ненулевое значение.

#### 2.2. Обработка элементов файла

Для того чтобы удалить элемент из файла нужно использовать вспомогательный файл. Во вспомогательный файл переписываются все элементы исходного файла за исключением тех, которые требуется удалить. После этого исходный файл удаляется из памяти, а вспомогательному файлу присваивается имя исходного файла.

void del(char \*filename)

{//удаление записи с номером х

FILE \*f;//исходный файл

FILE\*temp;//вспомогательный файл

//открыть исходный файл для чтения

f=fopen(filename,”rb”);

//открыть вспомогательный файл для записи

temp=fopen(”temp”,”wb”)

student a;//буфер для чтения данных из файла

//считываем данные из исходного файла в буфер

for(long i=0; fread(&a,sizeof(student),1,f);i++)

if(i!=x)//если номер записи не равен х

{

//записываем данные из буфера во временный файл

fwrite(&a,sizeof(student)1,temp);

}

else

{

cout<<a<<" - is deleting...";

}

fclose(f);//закрываем исходный файл

fclose(temp); //закрываем временный файл

remove(filename);//удаляем исходный файл

rename(”temp”, filename);//переименовываем временный файл

}

Для корректировки элементов файла используется аналогичный алгоритм. Данные из исходного файла переписываются во вспомогательный файл, но записи, которые нужно изменить записываются в откорректированном виде.

Для добавления элементов в начало или в середину файла также используется вспомогательный файл, в который в нужное место добавляются новые данные.

Для добавления элементов конец файла достаточно открыть его в режиме “a” или “a+” (для добавления) и записать новые данные в конец файла.

f=fopen(filename,”ab”);//открыть файл для добавления

cout<<"\nHow many records would you add to file?";

cin>>n;

for(int i=0;i<n;i++)

{

//прочитать объект

fwrite(&a,sizeof(student),1,f);//записать в файл

}

fclose(f);//закрыть файл