**广州大学学生实验报告**

**开课学院及实验室：计算机学院/电子实验楼518 2019年 11月25日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学 院** | **计算机科学与网络工程学院** | **年级/专业/班** | **计科173** | **姓名** | 谢绍波 | **学号** | 1706100109 |
| **实验课程名称** | **计算机网络实验** | | | | | **成绩** |  |
| **实验项目名称** |  | | | | | **指导老师** | 刘淼 |
| **一、实验目的**  Windows或Linux环境下运行，程序应在单机上运行  **二、实验环境**  1、win10  2、vscode  **三、实验内容与步骤**   1. **写一个程序来模拟网桥功能。**   模拟实现网桥的转发功能，以从文件中读取帧模拟网桥从网络中收到一帧，即从两个文件中读入一系列帧，从第一个文件中读入一帧然后从第二个文件中再读入一帧，如此下去。对每一帧，显示网桥是否会转发。  **分析**：用程序模拟网桥功能，可以假定用两个文件分别代表两个网段上的网络帧数据。而两个文件中的数据应具有帧的特征，即有目的地址，源地址和帧内数据。程序交替读入帧的数据，就相当于网桥从网段中得到帧数据，当然如果模拟的数据量比较少，也可以用两个数组代替两个文件存放帧数据，同样达到代表两个网段上的帧数据的效果。  对于网桥来说，能否转发帧在于把接收到的帧与网桥中的转发表相比较。判断目的地址后才决定是否转发。由此可见转发的关键在于构造转发表。这里转发表可通过动态生成。  **程序设计思路及实现：**  交替读取两个文件内的数据，然后查找转发表。如果找不到则提示并将源  地址和接口信息加入转发表中，如果找到则提示转发到哪个接口，如果数据中  源地址和目的地址都来自同一接口则提示不转发。  **程序流程图：**    #include<fstream>  #include<iostream>  using namespace std;  #define INTER\_COUNT 2           //接口数量  #define TABLE\_MAX\_CUONT 255     //转发表最大数量  int tablecount = 0;             //转发表数量  struct trantable{               //转发表结构体      char inter;                 //接口      char dest;  }table[TABLE\_MAX\_CUONT] = {0,0,0,0};  struct dataframe{               //数据帧结构体      char source;      char dest;  };  void update\_table(struct dataframe data,int inter){      //更新转发表      table[tablecount].dest = data.source;      table[tablecount].inter = inter;      tablecount++;       //转发表数量++  }  int search\_table(struct dataframe data){      //查找转发表      int temp1 = 0,temp2 = 0;      for(int i = 0; i<tablecount; i++){          if(table[i].dest == data.dest){              temp1 = table[i].inter;          }          if(table[i].dest == data.source){              temp2 = table[i].inter;          }      }      if(temp1 == temp2 && temp1 && temp2){          return 3;      }      else if(temp1){          return temp1;      }else{          return 0;      }  }  int main(){      int flag;      struct dataframe data;      fstream file1("file1.txt"),file2("file2.txt");      while(!file1.eof()&&!file2.eof()){          file1>>data.source>>data.dest;          flag = search\_table(data);          cout<<"数据帧（Source： "<<data.source<<",Destination： "<<data.dest<<") ";          switch(flag){              case 0:{                  puts("无法转发");                  update\_table(data,1);                  break;              }              case 3:{                  puts("在同一接口，不转发");                  break;              }              default:{                  cout<<"转发至接口： "<<flag<<endl;                  break;              }          }          file2>>data.source>>data.dest;          flag = search\_table(data);          cout<<"数据帧（Source： "<<data.source<<",Destination： "<<data.dest<<") ";          switch(flag){              case 0:{                  puts("无法转发");                  update\_table(data,2);                  break;              }              case 3:{                  puts("在同一接口，不转发");                  break;              }              default:{                  cout<<"转发至接口： "<<flag<<endl;                  break;              }          }      }      file1.close();      file2.close();  }       1. **编写一个计算机程序用来计算一个文件的16位效验和。最快速的方法是用一个32位的整数来存放这个和。记住要处理进位（例如，超过16位的那些位），把它们加到效验和中。**   **要求**：1）以命令行形式运行：check\_sum infile  其中check\_sum为程序名，infile为输入数据文件名。  2）输出：数据文件的效验和  附：效验和（checksum)  **参见RFC1071 - Computing the Internet checksum**   * 原理：把要发送的数据看成16比特的二进制整数序列，并计算他们的和。若数据字节长度为奇数，则在数据尾部补一个字节的0以凑成偶数。 * 例子：16位效验和计算，下图表明一个小的字符串的16位效验和的计算。   为了计算效验和，发送计算机把每对字符当成16位整数处理并计算效验和。如果效验和大于16位，那么把进位一起加到最后的效验和中。  f7_6c  **思路**  这里计算校验和的方法是，先把文件中的所有字符都读成一整个字符串。把每个字符都转换成十进制的ASCII码。因为是把两个字符合成一个16比特的整数，所以这里我把高位和高位相加，低位和低位相加，如果字符的个数为奇数，那么最后一个字符后面还要补上一个字节的0以凑成偶数。然后处理低位相加的进位和高位相加的进位，这里要循环处理高位的进位，因为把高位的进位加到低位上可能还会产生进位。处理的结果还是十进制，转换成十六进制即可。  如果要以命令行形式运行：check\_sum infile。前面的check\_sum是程序运行产生的exe可执行文件，后面的infile则是被读取文件的路径。另外，程序里的main函数还要加上参数，int main(int argc, char\* argv[])，其中命令行的infile作为数据传给argv[1]，程序里用一个变量保存argv[1]作为路径。  **程序流程图：**    #include <iostream>  #include <fstream>  #include <string>  #include <sstream>  #include <vector>  using namespace  std;    string decToHex(int high, int low) {      const std::string hex = "0123456789ABCDEF";      stringstream ss;      ss << hex[high >> 4] << hex[high & 0xf];      ss << hex[low >> 4] << hex[low & 0xf];      return ss.str();  }  int main(int argc, char\* argv[]) {      string str, tmp = "";      ifstream file1(argv[1], ios::in);      vector<int> l, r;      while (getline(file1, tmp)) {          str += tmp;      }      cout << str << endl;      int size = str.size();      int len = 0;      cout << "\nconvert to hexadecimal\n";      for (int i = 0; i < size - 1; i += 2) {          l.push\_back((int)str[i]);          r.push\_back((int)str[i + 1]);          cout << "0x" << decToHex(l[len], r[len]) << " ";          len++;      }      if (size % 2 == 1) {          l.push\_back((int)str[size - 1]);          r.push\_back(0);          cout << "0x" << decToHex(l[len], r[len]) << " ";          len++;      }      cout << endl;      int low = 0, t = 0;      for (int i = 0; i < len; i++) {          low += r[i];          if (low > 255) {              t += (low / 256);              low %= 256;          }      }      int high = t, t2 = 0;      for (int i = 0; i < len; i++) {          high += l[i];          if (high > 255) {              t2 += (high / 256);              high %= 256;          }      }      while (t2 != 0) {          low += t2;          t = 0;          t2 = 0;          if (low > 255) {              t = low / 256;              low %= 256;          }          high += t;          if (high > 255) {              t2 = high / 256;              high %= 256;          }      }      cout << "check\_sum\n" << decToHex(high, low) << endl<<endl;      file1.close();      system("pause");  }   四、实验分析与心得 1. 第一个实验，要理解网桥的原理工作原理：其基本功能是转发网络中的二层报文，从某个端口收到的二层报文，解析二层报文的源MAC和目的MAC，根据源MAC学习形成MAC表，根据目的MAC，原封不动的将该报文转发到适当的出端口，从而保证最终目的设备能收到这个报文。对于网桥来说，能否转发帧在于把接收到的帧与网桥中的转发表相比较。判断目的地址后才决定是否转发。  2. 第二个实验，思路就是，将要校验的文件，按字符读取，将字符由  ASCII码转成16进制表示，与下一字符形成双16字符累加，得到累加值。  心得：  经过本次实验，对网桥的工作原理以及对校验和算法有了更进一步的理解，受益匪浅。 | | | | | | | |