**TASK#1**

**Pre-requisites**

Install WinPcap (Packet Capture Library for Windows) and visual stdio code

Before you can work with pcap files in Visual Studio, you need to have the WinPcap library installed on your system.

Create a New Visual Studio Project:

Start by creating a new project in Visual Studio.

Add Source Files:

new source file for your code or add it

Configure Project Properties:

configured to work with WinPcap gcc compiler and some other extension to work in C++ environment.

**This code is essentially copying all the packets from one pcap file to another, allowing for packet processing**

1. **Firstly, we include the necessary headers:**

#include <pcap.h>

#include <iostream>

#include <vector>

#include <string>

* **pcap.h** is the header file for the pcap library, which is used to capture and process network packets.
* **iostream** is a standard header file for input/output streams in C++.
* **vector** and **string** are part of the C++ Standard Library, used here to work with vectors (dynamic arrays) and strings respectively.

1. **Next, we have a function stub for processing packets:**

void processPacket(const u\_char\* packet, const pcap\_pkthdr& packet\_header) {

This function **processPacket** is intended to handle the actual processing of each packet but for now, it's empty. The function takes a packet and its header as arguments.

Now, let's move on to the **main** function:

int main() {

std::string inputPcapFile = "sip-rtp-g729a.pcap";

std::string outputPcapFile = "modified.pcap";

char errbuf[PCAP\_ERRBUF\_SIZE];

* This is the start of our **main** function, which is the entry point of any C++ program.
* We define two strings for the input and output filenames of the pcap files.
* **errbuf** is a buffer to store any error messages.

pcap\_t\* handle = pcap\_open\_offline(inputPcapFile.c\_str(), errbuf);

if (!handle) {

std::cerr << "Error opening pcap file: " << errbuf << std::endl;

return 1;

}

* We try to open the input pcap file in "offline" mode, meaning we're reading from a file, not capturing live data.
* If the file cannot be opened, an error message is printed, and the program exits with a return code of 1.

pcap\_dumper\_t\* outputPcapDump = pcap\_dump\_open(handle, outputPcapFile.c\_str());

if (!outputPcapDump) {

std::cerr << "Error opening output pcap file: " << pcap\_geterr(handle) << std::endl;

pcap\_close(handle);

return 1;

}

We then open a dump file to write packets to. If this fails, we print an error message, close the input file handle, and exit.

pcap\_pkthdr packet\_header;

const u\_char\* packet\_data;

These variables will hold the packet header and data for each packet we read.

while ((packet\_data = pcap\_next(handle, &packet\_header))) {

// Process the packet

processPacket(packet\_data, packet\_header);

// Write the packet to the output pcap file

pcap\_dump(reinterpret\_cast<u\_char\*>(outputPcapDump), &packet\_header, packet\_data);

}

* We enter a loop that continues as long as there are packets to read from the file.
* For each packet, we first process it (with an empty function for now).
* Then we write the packet to the output file using **pcap\_dump**.

Finally, we have the cleanup code:

pcap\_dump\_close(outputPcapDump);

pcap\_close(handle);

std::cout << "Packets processed and written to " << outputPcapFile << std::endl;

return 0;

}

* We close the output dump file and the input pcap handle.
* A message is printed to the console to indicate the process is complete.
* The program exits with a return code of 0, which usually indicates success.

**TASK 2**

1. **Sets Up the Database:**
   * First, i open a database file named "sip\_data.db".
   * If it doesn't exist, it will create it. If there's a problem opening the database, it will print an error and stop.
2. **Creates a Table:**
   * Next, it checks if there's a table named "sip\_data" in the database. If not, it creates one. This table has three text columns: "to\_field", "from\_field", and "call\_id".
   * If there's an issue creating the table, it will print an error message and stop.
3. **Inserts Data:**
   * The program then adds a new row into the "sip\_data" table with specific values for the "to\_field", "from\_field", and "call\_id".
   * If it runs into any problems while inserting the data, it will print an error and stop.
4. **Retrieves and Shows Data:**
   * After inserting the data, the program retrieves all entries from the "sip\_data" table and prints out each row to the screen.
   * It will display the "to\_field", "from\_field", and "call\_id" for each entry.
5. **Cleans Up:**
   * In the end, it closes the database to make sure everything is wrapped up neatly.
6. **Main Function:**
   * The **main** function. It t set up the database and table, insert data, display the data, and then closes the database.
   * If anything goes wrong during setup or inserting data, it will close the database and the program will end with an error code.
   * If all operations are successful, the program ends with a success code.

**Journey of this task:**

* I began the task on a Windows operating system, but I encountered issues with importing Pcap. Despite numerous attempts and even trying on multiple machines, the same error persisted. I repeatedly installed and uninstalled the VS Code compiler, GCC, and other libraries, but to no avail.
* I experimented with both Npcap and WinPcap; however, the issue remained unresolved even manually I add library files. Nevertheless, I persisted and decided to try a different approach by using a Linux-based machine.
* After making some minor modifications, I executed the same code that had been written for Windows on the Linux machine, and it worked successfully. It turned out to be a rewarding experience, teaching me that the task was inherently suited for Linux.