

Contents

DANH MỤC HÌNH VẼ	3
MỞ ĐẦU	4
PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH	5
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT.....	5
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG	5
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ.....	5
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	5
PHẦN II: LẬP TRÌNH MẠNG	6
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT.....	6
1. IP Helper.....	6
• Tổng quát về IP Helper.....	6
• Mục đích của IP Helper	7
• Nơi áp dụng IP Helper	7
• Đối tượng phát triển	7
• Các yêu cầu về môi trường run-time	7
2. Windows Sockets 2	7
• Mục đích.....	7
• Đối tượng phát triển	8
• Các yêu cầu về môi trường run-time	8
3. Sử dụng IP Helper	8
4. Lệnh netstat	10
• Cú pháp	10
• Tham chiếu.....	10
• Chú giải.....	11
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG.....	12
1. Yêu cầu.....	12
2. Phương pháp triển khai	12
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ.....	16
1. Cài đặt chương trình	16

• Hàm lấy thông tin các kết nối TCP đang hoạt động	16
• Hàm lấy thông tin các cổng UDP trên đó máy tính đang lắng nghe	18
• Hàm lấy thông tin thống kê theo giao thức	19
• Chương trình chính	23
2. Kết quả chạy chương trình.....	24
3. Đánh giá kết quả	25
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	25

DANH MỤC HÌNH VẼ

Hình 1: Kết quả chạy chương trình với tùy chọn my_netstat	24
Hình 2: Kết quả chạy chương trình với tùy chọn my_netstat -p udp -a	24
Hình 3: Kết quả chạy chương trình với tùy chọn my_netstat -s	25

DANH SÁCH BẢNG BIỂU

Bảng 1: Danh sách tham chiếu của lệnh netstat	11
Bảng 2: Danh sách tùy chọn của chương trình my_netstat	12

DANH SÁCH TỪ VIẾT TẮT

MỞ ĐẦU

PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH

TIÊU ĐỀ: Tìm hiểu và mô phỏng các giải thuật lập lịch trên hệ thống thời gian thực

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

PHẦN II: LẬP TRÌNH MẠNG

TIÊU ĐỀ: Tìm hiểu về IP Helper và xây dựng chương trình my_netstat

Yêu cầu đề tài:

1. Tìm hiểu về IP Helper và chương trình netstat.
2. Xây dựng được chương trình my_netstat để mô phỏng chương trình netstat.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1. IP Helper

- Tổng quát về IP Helper
 - Trình trợ giúp giao thức Interenet (IP Helper) hỗ trợ quản trị mạng cho máy tính cục bộ bằng cách cho phép các ứng dụng truy xuất thông tin về cấu hình của mạng của máy tính cục bộ và sửa đổi cấu hình đó. IP Helper cũng cung cấp các cơ chế thông báo để đảm bảo rằng một ứng dụng được thông báo khi các khía cạnh nhất định của cấu hình mạng của máy tính cục bộ bị thay đổi.
 - Nhiều hàm của IP Helper truyền các tham số có cấu trúc đại diện cho các loại dữ liệu được liên kết với công nghệ MIB (Management Information Base – là một cấu trúc dữ liệu gồm các đối tượng được quản lý, được dùng cho các thiết bị chạy trên nền tảng TCP/IP). IP Helper sử dụng các cấu trúc này để thể hiện các thông tin mạng khác nhau, chẳng hạn như các mục đệm ARP (Address Resolution Protocol – giao thức phân giải địa chỉ). Mặc dù IP Helper API sử dụng các cấu trúc cấu trúc này nhưng IP Helper khác biệt so với MIB và SNMP (Simple Network Management Protocol – quản lý giao thức mạng đơn giản).
 - IP Helper cung cấp các khả năng áp dụng trong các lĩnh vực sau:
 - Lấy thông tin cấu hình mạng
 - Quản lý bộ điều hợp mạng
 - Quản lý giao diện
 - Quản lý các địa chỉ IP

- Sử dụng giao thức phân giải địa chỉ
 - Lấy thông tin về giao thức Internet và giao thức tin nhắn điều khiển Internet
 - Quản lý định tuyến
 - Nhận thông báo về các sự kiện mạng
 - Lấy thông tin về giao thức điều khiển truyền vận (TCP) và giao thức dữ liệu người dùng (UDP).
- Mục đích của IP Helper
 - IP Helper API cho phép truy xuất và sửa đổi cài đặt cấu hình mạng cho máy tính cục bộ.
 - Nơi áp dụng IP Helper
 - IP Helper API có thể được áp dụng trong bất kì môi trường điện toán nào có cấu hình mạng và cấu hình TCP/IP có thể lập trình được. Một số ứng dụng điển hình bao gồm các giao thức định tuyến IP và các chương trình SNMP (Simple Network Management Protocol – quản lý giao thức mạng đơn giản).
 - Đối tượng phát triển
 - IP Helper API được thiết kế để sử dụng bởi các nhà lập trình viên C/C++.
 - Các yêu cầu về môi trường run-time
 - IP Helper API có thể được sử dụng trên tất cả các nền tảng Windows. Không phải tất cả hệ điều hành đều hỗ trợ tất cả các hàm của IP Helper. Nếu một hàm của IP Helper được sử dụng trên một nền tảng không được hỗ trợ, lỗi ERROR_NOT_SUPPORTED sẽ được trả về.

2. Windows Sockets 2

- Mục đích
 - Windows Sockets 2 (Winsock) cho phép các nhà lập trình viên tạo ra các ứng dụng internet, intranet hoặc có thể kết nối mạng nâng cao để truyền dữ liệu ứng dụng qua mạng, độc lập với giao thức mạng đang sử dụng. Với

Winsock, lập trình viên được cung cấp khả năng kết nối mạng nâng cao của hệ điều hành Microsoft Windows chẳng hạn như multicast (phát đa hướng) và Quality of Service (QoS – chất lượng của dịch vụ).

- Winsock theo mô hình Windows Open System Architecture (kiến trúc hệ thống mở Windows), nó xác định giao diện nhà cung cấp dịch vụ tiêu chuẩn (SPI) giữa giao diện lập trình ứng dụng (API) với các chức năng được xuất của nó và các ngăn xếp giao thức. Winsock sử dụng mô hình socket được phổ biến lần đầu tiên bởi Berkeley Software Distribution (BSD) UNIX. Sau đó, nó đã được điều chỉnh cho Windows trong Windows Sockets 1.1, trong đó các ứng dụng Windows Sockets 2 tương thích ngược. Lập trình Winsock trước đây tập trung vào TCP / IP. Một số thực tiễn lập trình hoạt động với TCP / IP không hoạt động với mọi giao thức. Do đó, API Windows Sockets 2 thêm các chức năng khi cần thiết để xử lý một số giao thức.
- Đối tượng phát triển
 - Windows Sockets 2 được thiết kế để sử dụng bởi các nhà lập trình viên C/C++. Làm quen với mạng Windows là bắt buộc.
- Các yêu cầu về môi trường run-time
 - Windows Sockets 2 có thể sử dụng trên tất cả các nền tảng Windows.

3. Sử dụng IP Helper

- Dưới đây là các bước để tạo viết một ứng dụng sử dụng IP Helper
 - Tạo mới một project C++ và thêm mới một source file C++ vào project.
 - Đảm bảo rằng Build environment đã đề cập đến các thư mục Include, Lib và Src của Platform Software Development Kit (SDK).
 - Đảm bảo rằng Build environment đã liên kết đến file IP Helper Library Iphlpapi.lib và Winsock Library WS2_32.lib.
 - Bắt đầu lập trình ứng dụng IP Helper. Sử dụng IP Helper API bằng cách include các IP Helper header file vào source file.


```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>

#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>

int main() {
    return 0;
}
```

- Một số lưu ý:
 - Tập tiêu đề Iphlpapi.h (header file) là bắt buộc đối với các ứng dụng sử dụng IP Helper. Iphlpapi.h sẽ tự động bao gồm các tập tiêu đề khác với các cấu trúc (structures) và các bảng liệt kê (enumerations) được sử dụng bởi IP Helper.
 - Tập tiêu đề Winsock2.h cho Windows Sockets 2 được yêu cầu bởi hầu hết các ứng dụng sử dụng IP Helper. Dòng `#include <winsock2.h>` phải được đặt trước dòng `#include <iphlpapi.h>`
 - Tập tiêu đề winsock2.h bên trong bao gồm các thành phần chính từ tập windows.h, do đó thường không có dòng `#include <windows.h>` trong chương trình sử dụng IP Helper. Nếu cần thiết có thể dùng dòng lệnh trên nhưng phải bắt đầu bằng `#define WIN32_LEAN_AND_MEAN` macro. Vì các lý do lịch sử, windows.h mặc định đã bao gồm winsock.h cho Windows Sockets 1.1. Các khai báo trong winsock2.h sẽ xung đột với các khai báo trong windows.h. Macro WIN32_LEAN_AND_MEAN sẽ ngăn chặn việc tập winsock.h được bao gồm bởi tập windows.h.

4. Lệnh netstat

Lệnh netstat có chức năng hiển thị các kết nối TCP đang hoạt động, các cổng mà tại đó máy tính đang nghe và đồng thời hiển thị các thông kê Ethernet, bảng định tuyến IP, thông kê IPv4 (cho giao thức IP, ICMP, TCP và UDP) và số liệu thông kê IPv6 (cho IPv6, ICMPv6, TCP qua IPv6 và UDP qua giao thức IPv6). Nếu không có tham số, netstat sẽ hiển thị các kết nối TCP đang hoạt động.

- Cú pháp

```
netstat [-a] [-e] [-n] [-o] [-p <Protocol>] [-r] [-s] [<Interval>]
```

- Tham chiếu

Tham biến	Mô tả
-a	Hiển thị tất cả các kết nối TCP đang hoạt động, cũng như các cổng TCP và UDP mà trên đó máy tính đang nghe.
-e	Hiển thị số liệu thông kê Ethernet, chẳng hạn như số byte và các gói được gửi và nhận. Tham số này có thể được kết hợp với -s.
-n	Hiển thị các kết nối TCP đang hoạt động, tuy nhiên, các địa chỉ và số cổng được biểu thị dưới dạng số chứ không thể xác định tên cụ thể.
-o	Hiển thị các kết nối TCP đang hoạt động và bao gồm cả ID tiến trình (Process ID - PID) cho mỗi kết nối. Bạn có thể tìm ứng dụng bằng cách tra cứu PID trên tab Processes trong Windows Task Manager. Tham số này có thể được kết hợp với -a, -n và -p.
-p	Hiển thị kết nối cho giao thức được chỉ định bởi <i>Protocol</i> . Trong trường hợp này, <i>Protocol</i> có thể là tcp, udp, tcpv6 hoặc udpv6. Nếu tham số này được sử dụng với -s để hiển thị số liệu thông kê theo giao thức, <i>Protocol</i> có thể là tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6 hoặc ipv6.
-s	Hiển thị số liệu thông kê theo giao thức. Theo mặc định, các số liệu thông kê được hiển thị cho các giao thức TCP, UDP, ICMP và IP. Nếu giao thức IPv6 được cài đặt, các thông kê sẽ được hiển thị cho giao thức TCP thông qua IPv6, UDP qua IPv6, ICMPv6 và IPv6. Tham số -p có thể được sử dụng để chỉ định một tập các giao thức.
-r	Hiển thị nội dung của bảng định tuyến IP (IP routing table). Thông tin này tương đương với lệnh ip route. Sau mỗi giây, thông tin được chọn sẽ được hiển thị lại. Nhấn CTRL

	+ C để dừng quá trình hiển thị lại. Nếu tham số này bị bỏ qua, netstat chỉ in thông tin đã chọn một lần.
/?	Hiển thị trợ giúp tại command prompt.

Bảng 1: Danh sách tham chiếu của lệnh netstat

- Chú giải
 - Các tham số được sử dụng với lệnh này phải được bắt đầu bằng dấu gạch nối (-) thay vì dấu gạch chéo (/).
 - Lệnh netstat cung cấp số liệu thống kê cho những đối tượng sau đây:
 - Tên của giao thức (TCP hoặc UDP).
 - Địa chỉ cục bộ. Địa chỉ IP của máy tính cục bộ và số cổng đang được sử dụng. Tên của máy tính cục bộ tương ứng với địa chỉ IP và tên của cổng được hiển thị trừ khi tham số -n được chỉ định. Nếu cổng chưa được thiết lập, số cổng được hiển thị dưới dạng dấu hoa thị (*).
 - Địa chỉ từ xa. Địa chỉ IP và số cổng của máy tính từ xa được kết nối. Các tên máy tính từ xa tương ứng với địa chỉ IP và cổng được hiển thị trừ khi tham số -n được chỉ định. Nếu cổng chưa được thiết lập, số cổng được hiển thị dưới dạng dấu hoa thị (*).
 - Trạng thái. Cho biết trạng thái của các kết nối TCP. Các trạng thái có thể như sau: CLOSE_WAIT CLOSED ESTABLISHED FIN_WAIT_1 FIN_WAIT_2 LAST_ACK listEN SYN_RECEIVED SYN_SEND timeD_WAIT để biết thêm thông tin về trạng thái của kết nối TCP, tham khảo Rfc 793.
 - Lệnh này chỉ có thể sử dụng nếu giao thức Internet Protocol (TCP/IP) được cài đặt như một thành phần trong thuộc tính của bộ điều hợp mạng trong hệ thống kết nối Mạng (Network Connections).

CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

1. Yêu cầu

Xây dựng chương trình my_netstat với các tùy chọn sau:

Tùy chọn	Mô tả
my_netstat	Hiển thị tất cả các kết nối TCP đang hoạt động
my_netstat -p tcp -a	
my_netstat -a	Hiển thị tất cả các kết nối TCP đang hoạt động cũng như các cổng TCP và UDP trên đó máy tính đang lắng nghe
my_netstat -p udp -a	Hiển thị tất cả các cổng UDP mà trên đó máy tính đang lắng nghe
my_netstat -s	Hiển thị số liệu thống kê theo giao thức. Theo mặc định, các số liệu thống kê được hiển thị cho các giao thức TCP, UDP, ICMP và IP. Nếu giao thức IPv6 được cài đặt, các thống kê sẽ được hiển thị cho giao thức TCP thông qua IPv6, UDP qua IPv6, ICMPv6 và IPv6.

Bảng 2: Danh sách tùy chọn của chương trình my_netstat

2. Phương pháp triển khai

- Hiển thị tất cả các kết nối TCP đang hoạt động
 - Sử dụng hàm GetTcpTable() để lấy thông tin bảng kết nối TCP
 - Cú pháp:

```

DWORD GetTcpTable(PMIB_TCPTABLE pTcpTable, PDWORD pdwSize, BOOL bOrder);
typedef struct _MIB_TCPTABLE
{
    DWORD dwNumEntries; //Specifies how many entries are in the table
    field
    MIB_TCPROW table[ANY_SIZE]; //Is a pointer to an array of MIB_TCPROW
    structures
    //that contain TCP connection information
} MIB_TCPTABLE, *PMIB_TCPTABLE;

typedef struct _MIB_TCPROW
{
    DWORD dwState; //Specifies the state of the TCP connection
    DWORD dwLocalAddr; //Specifies a local IPv4 address for the
    connection
    DWORD dwLocalPort; //Specifies a local port for the connection
    DWORD dwRemoteAddr; //Specifies the remote IPv4 address for the
    connection
    DWORD dwRemotePort; //Specifies the remote port for the connection
} MIB_TCPROW, *PMIB_TCPROW;

```

- Hiện thị các cổng UDP trên đó máy tính đang lắng nghe
 - Sử dụng hàm GetUdpTable() để lấy thông tin tất cả các cổng UDP mà trên đó máy tính đang lắng nghe.

- Cú pháp

```
DWORD GetUdpTable(  
    PMIB_UDPTABLE pUdpTable,  
    PDWORD pdwSize,  
    BOOL bOrder  
);  
typedef struct _MIB_UDPTABLE  
{  
    DWORD dwNumEntries; //Specifies how many entries are in the table  
    field  
    MIB_UDPROW table[ANY_SIZE]; //Is a pointer to an array of MIB_UDPROW  
                                //structures that contain UDP listener  
    information  
} MIB_UDPTABLE, *PMIB_UDPTABLE;  
typedef struct _MIB_UDPROW  
{  
    DWORD dwLocalAddr; //Specifies the local IPv4 address  
    DWORD dwLocalPort; //Specifies the local port  
} MIB_UDPROW, *PMIB_UDPROW;
```

- Hiện thị số liệu thống kê theo giao thức
 - Sử dụng các hàm GetIpStatisticsEx(), GetIcmpStatisticsEx(), GetTcpStatisticsEx(), GetUdpStatisticsEx() để lấy số liệu thống kê theo giao thức.
 - Cú pháp

▪ Hàm GetIpStatisticsEx()

```

ULONG GetIpStatisticsEx(
    PMIB_IPSTATS Statistics,
    ULONG          Family //The protocol family for which to retrieve
statistics
    //AF_INET for Internet Protocol version 4 (IPv4)
    //AF_INET6 for Internet Protocol version 6 (IPv6)
);
typedef struct _MIB_IPSTATS_LH {
    union {
        DWORD          dwForwarding;
        MIB_IPSTATS_FORWARDING Forwarding;
    };
    DWORD dwDefaultTTL; //The default initial time-to-live (TTL) for
datagrams originating on a particular computer
    DWORD dwInReceives; //The number of datagrams received
    DWORD dwInHdrErrors; //The number of datagrams received that have
header errors
    DWORD dwInAddrErrors; //The number of datagrams received that have
address errors
    DWORD dwForwDatagrams; //The number of datagrams forwarded
    DWORD dwInUnknownProtos; //The number of datagrams received that have
an unknown protocol
    DWORD dwInDiscards; //The number of received datagrams discarded
    DWORD dwInDelivers; //The number of received datagrams delivered
    DWORD dwOutRequests; //The number of outgoing datagrams that IP is
requested to transmit. This number does not include forwarded datagrams
    DWORD dwRoutingDiscards; //The number of outgoing datagrams discarded
    DWORD dwOutDiscards; //The number of transmitted datagrams discarded
    DWORD dwOutNoRoutes; //The number of datagrams for which this
computer did not have a route to the destination IP address. These
datagrams were discarded.
    DWORD dwReasmTimeout; //The amount of time allowed for all pieces of
a fragmented datagram to arrive. If all pieces do not arrive within this
time, the datagram is discarded
    DWORD dwReasmReqds; //The number of datagrams that require re-
assembly
    DWORD dwReasmOks; //The number of datagrams that were successfully
reassembled
    DWORD dwReasmFails; //The number of datagrams that cannot be
reassembled
    DWORD dwFragOks; //The number of datagrams that were fragmented
successfully
    DWORD dwFragFails; //The number of datagrams that have not been
fragmented because the IP header specifies no fragmentation. These
datagrams are discarded
    DWORD dwFragCreates; //The number of fragments created
    DWORD dwNumIf; //The number of interfaces
    DWORD dwNumAddr; //The number of IP addresses associated with this
computer
    DWORD dwNumRoutes; //The number of routes in the IP routing table
} MIB_IPSTATS_LH, *PMIB_IPSTATS_LH;

```

- Hàm GetIcmpStatisticsEx()

```

ULONG GetIcmpStatisticsEx(
    PMIB_ICMP_EX Statistics,
    ULONG          Family
);
typedef struct _MIB_ICMP_EX_XPSP1 {
    MIBICMPSTATS_EX icmpInStats;
    MIBICMPSTATS_EX icmpOutStats;
} MIB_ICMP_EX_XPSP1, *PMIB_ICMP_EX_XPSP1;
typedef struct _MIBICMPSTATS_EX_XPSP1 {
    DWORD dwMsgs; // Specifies the number of messages received or sent.
    DWORD dwErrors; // The number of errors received or sent.
    DWORD rgdwTypeCount[256]; // The type count
} MIBICMPSTATS_EX_XPSP1, *PMIBICMPSTATS_EX_XPSP1;

```

- Hàm GetTcpStatisticsEx()

```

ULONG GetTcpStatisticsEx(
    PMIB_TCPSTATS Statistics,
    ULONG          Family
);
typedef struct _MIB_TCPSTATS_LH {
    union {
        DWORD          dwRtoAlgorithm;
        TCP_RTO_ALGORITHM RtoAlgorithm;
    };
    DWORD dwRtoMin; // The minimum RTO value in milliseconds
    DWORD dwRtoMax; // The maximum RTO value in milliseconds
    DWORD dwMaxConn; // The maximum number of connections. If this member
is -1, the maximum number of connections is variable.
    DWORD dwActiveOpens; // The number of active opens. In an active open,
the client is initiating a connection with the server.
    DWORD dwPassiveOpens; // The number of passive opens. In a passive
open, the server is listening for a connection request from a client.
    DWORD dwAttemptFails; // The number of failed connection attempts.
    DWORD dwEstabResets; // The number of established connections that
were reset.
    DWORD dwCurrEstab; // The number of currently established connections.
    DWORD dwInSegs; // The number of segments received.
    DWORD dwOutSegs; // The number of segments transmitted. This number
does not include retransmitted segments.
    DWORD dwRetransSegs; // The number of segments retransmitted.
    DWORD dwInErrs; // The number of errors received.
    DWORD dwOutRsts; // The number of segments transmitted with the reset
flag set.
    DWORD dwNumConns; // The number of connections that are currently
present in the system. This total number includes connections in all states
except listening connections.
} MIB_TCPSTATS_LH, *PMIB_TCPSTATS_LH;

```

- Hàm GetUdpStatisticsEx()

```
ULONG GetUdpStatisticsEx(
    PMIB_UDPSTATS Statistics,
    ULONG Family
);
typedef struct _MIB_UDPSTATS {
    DWORD dwInDatagrams; //The number of datagrams received.
    DWORD dwNoPorts; //The number of datagrams received that were
discarded because the port specified was invalid.
    DWORD dwInErrors; //The number of erroneous datagrams received. This
number does not include the value contained by the dwNoPorts member.
    DWORD dwOutDatagrams; //The number of datagrams transmitted.
    DWORD dwNumAddrs; //The number of entries in the UDP listener table.
} MIB_UDPSTATS, *PMIB_UDPSTATS;
```

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

1. Cài đặt chương trình

- Hàm lấy thông tin các kết nối TCP đang hoạt động


```

void retrievingTCPConnectionTable()
{
    PMIB_TCPTABLE pTcpTable;
    DWORD dwSize = 0, dwReturnValue = 0;
    char szLocalAddr[128], szRemoteAddr[128]; struct in_addr IpAddr;
    pTcpTable = (MIB_TCPTABLE *)malloc(sizeof(MIB_TCPTABLE));
    if (pTcpTable == NULL) {
        cout << ("Error allocating memory\n"); return; }
    dwSize = sizeof(MIB_TCPTABLE);
    if ((dwReturnValue = GetTcpTable(pTcpTable, &dwSize, TRUE)) ==
ERROR_INSUFFICIENT_BUFFER) {
        free(pTcpTable);
        pTcpTable = (MIB_TCPTABLE *)malloc(dwSize);
        if (pTcpTable == NULL) {
            printf("Error allocating memory\n"); return; } }
    if ((dwReturnValue = GetTcpTable(pTcpTable, &dwSize, TRUE)) ==
NO_ERROR) {
        wprintf(L"Protocol\tState\t\tLocal Address\t\tRemote
Address\n");
        for (int i = 0; i < (int)pTcpTable->dwNumEntries; i++) {
            IpAddr.S_un.S_addr = (u_long)pTcpTable->
table[i].dwLocalAddr;
            strcpy_s(szLocalAddr, sizeof(szLocalAddr),
inet_ntoa(IpAddr));
            IpAddr.S_un.S_addr = (u_long)pTcpTable->
table[i].dwRemoteAddr;
            strcpy_s(szRemoteAddr, sizeof(szRemoteAddr),
inet_ntoa(IpAddr));
            wprintf(L"TCP\t\t");
            switch (pTcpTable->table[i].dwState) {
                case MIB_TCP_STATE_CLOSED: wprintf(L"CLOSED\t\t");
break;
                case MIB_TCP_STATE_LISTEN: wprintf(L"LISTEN\t\t");
break;
                case MIB_TCP_STATE_SYN_SENT: wprintf(L"SYN-SENT\t");
break;
                case MIB_TCP_STATE_SYN_RCVD: wprintf(L"SYN-
RECEIVED\t"); break;
                case MIB_TCP_STATE_ESTAB: wprintf(L"ESTABLISHED\t");
break;
                case MIB_TCP_STATE_FIN_WAIT1: wprintf(L"FIN-WAIT-1\t");
break;
                case MIB_TCP_STATE_FIN_WAIT2: wprintf(L"FIN-WAIT-
2\t\t"); break;
                default: wprintf(L"UNKNOWN dwState value"); break;
            }
            wprintf(L"%hs : %u\t", szLocalAddr,
ntohs((u_short)pTcpTable->table[i].dwLocalPort));
            wprintf(L"%hs : %u\n", szRemoteAddr,
ntohs((u_short)pTcpTable->table[i].dwRemotePort));
        }
    }
    else {
        printf("\tGetTcpTable failed with %d\n", dwReturnValue);
        free(pTcpTable); return; }
    if (pTcpTable != NULL) {
        free(pTcpTable); pTcpTable = NULL; }
}

```

- Hàm lấy thông tin các cổng UDP trên đó máy tính đang lắng nghe

```

void retrievingUDPListenerTable()
{
    PMIB_UDPTABLE pUdpTable;
    DWORD dwSize = 0;
    DWORD dwReturnValue = 0;
    char szLocalAddr[128];
    struct in_addr IpAddr;
    pUdpTable = (MIB_UDPTABLE *)malloc(sizeof(MIB_UDPTABLE));
    if (pUdpTable == NULL) {
        printf("Error allocating memory\n");
        return;
    }
    dwSize = sizeof(MIB_UDPTABLE);
    if ((dwReturnValue = GetUdpTable(pUdpTable, &dwSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
        free(pUdpTable);
        pUdpTable = (MIB_UDPTABLE *)malloc(dwSize);
        if (pUdpTable == NULL) {
            printf("Error allocating memory\n");
            return;
        }
    }
    if ((dwReturnValue = GetUdpTable(pUdpTable, &dwSize, TRUE)) ==
    NO_ERROR)
    {
        wprintf(L"Protocol\tLocalAddress\n");
        for (int i = 0; i < (int)pUdpTable->dwNumEntries; i++) {
            wprintf(L"UDP\t");
            IpAddr.S_un.S_addr = (u_long)pUdpTable->
            table[i].dwLocalAddr;
            strcpy_s(szLocalAddr, sizeof(szLocalAddr),
            inet_ntoa(IpAddr));
            wprintf(L"\t%hs : %u\n", szLocalAddr,
            ntohs((u_short)pUdpTable->table[i].dwLocalPort));
        }
    }
}

```



```
void showIpStatistics()  
{  
    wprintf(L"\nIP Statistics for IPv4: \n");  
    retrievingIpStatistics(AF_INET);  
    wprintf(L"\nIP Statistics for IPv6: \n");  
    retrievingIpStatistics(AF_INET6);  
}
```

○ ICMP

```
void retrievingIcmpStatistics(ULONG family)  
{  
    DWORD dwReturnValue = 0;  
    PMIB_ICMP_EX icmpStatistics;  
    icmpStatistics = (MIB_ICMP_EX *)malloc(sizeof(MIB_ICMP_EX));  
    if (icmpStatistics == NULL) {  
        wprintf(L"Error allocating memory");  
        return;  
    }  
    dwReturnValue = GetIcmpStatisticsEx(icmpStatistics, family);  
    if (dwReturnValue != NO_ERROR) {  
        wprintf(L"GetIcmpStatistics call failed with %d\n",  
dwReturnValue);  
    }  
    else {  
        wprintf(L"\tNumber of incoming ICMP messages: %ld\n",  
icmpStatistics->icmpInStats.dwMsgs);  
        wprintf(L"\tNumber of incoming ICMP errors received: %ld\n",  
icmpStatistics->icmpInStats.dwErrors);  
        wprintf(L"\tNumber of outgoing ICMP messages: %ld\n",  
icmpStatistics->icmpOutStats.dwMsgs);  
        wprintf(L"\tNumber of outgoing ICMP errors received: %ld\n",  
icmpStatistics->icmpOutStats.dwErrors);  
    }  
    if (icmpStatistics) {  
        free(icmpStatistics);  
    }  
}  
  
void showIcmpStatistics()  
{  
    wprintf(L"\nICMP Statistics for IPv4: \n");  
    retrievingIcmpStatistics(AF_INET);  
    wprintf(L"\nICMP Statistics for IPv6: \n");  
    retrievingIcmpStatistics(AF_INET6);  
}
```

○ TCP

```
void retrievingTcpStatistics(ULONG family)
{
    DWORD dwReturnValue = 0;
    PMIB_TCPSTATS tcpStatistics;
    tcpStatistics = (MIB_TCPSTATS *)malloc(sizeof(MIB_IPSTATS));
    if (tcpStatistics == NULL) {
        wprintf(L"Error allocating memory");
        return;
    }
    dwReturnValue = GetTcpStatisticsEx(tcpStatistics, family);
    if (dwReturnValue != NO_ERROR) {
        wprintf(L"GetTcpStatistics call failed with %d\n",
dwReturnValue);
    }
    else {
        wprintf(L"\tActive Opens: %ld\n", tcpStatistics->dwActiveOpens);
        wprintf(L"\tPassive Opens: %ld\n", tcpStatistics->dwPassiveOpens);
        wprintf(L"\tSegments Recv: %ld\n", tcpStatistics->dwInSegs);
        wprintf(L"\tSegments Xmit: %ld\n", tcpStatistics->dwOutSegs);
        wprintf(L"\tTotal # Conns: %ld\n", tcpStatistics->dwNumConns);
    }
    if (tcpStatistics) {
        free(tcpStatistics);
    }
}

void showTcpStatistics()
{
    wprintf(L"\nTCP Statistics for IPv4: \n");
    retrievingTcpStatistics(AF_INET);
    wprintf(L"\nTCP Statistics for IPv6: \n");
    retrievingTcpStatistics(AF_INET6);
}
```

○ UDP

```
void retrievingUdpStatistics(ULONG family)
{
    DWORD dwReturnValue = 0;
    PMIB_UDPSTATS udpStatistics;
    udpStatistics = (MIB_UDPSTATS *)malloc(sizeof(MIB_UDPSTATS));
    if (udpStatistics == NULL) {
        wprintf(L"Error allocating memory");
        return;
    }
    dwReturnValue = GetUdpStatisticsEx(udpStatistics, family);
    if (dwReturnValue != NO_ERROR) {
        wprintf(L"GetUdpStatistics call failed with %d\n",
dwReturnValue);
    }
    else {
        printf("\tNumber of datagrams received: %ld\n", udpStatistics->dwInDatagrams);
        printf("\tNumber of datagrams discarded because the port
number was bad: %ld\n", udpStatistics->dwNoPorts);
        printf("\tNumber of erroneous datagrams received: %ld\n",
udpStatistics->dwInErrors);
        printf("\tNumber of datagrams transmitted: %ld\n",
udpStatistics->dwOutDatagrams);
    }
    if (udpStatistics) {
        free(udpStatistics);
    }
}

void showUdpStatistics()
{
    wprintf(L"\nUDP Statistics for IPv4: \n");
    retrievingUdpStatistics(AF_INET);
    wprintf(L"\nUDP Statistics for IPv6: \n");
    retrievingUdpStatistics(AF_INET6);
}
```

- Chương trình chính

```
int main()
{
    char commandLine[20];
    while(true) {
        printf("Enter your command (Press q to exit): ");
        scanf("%[^\n]*c", commandLine);

        if (strcmp("q", commandLine) == 0 || strcmp("Q", commandLine)
== 0) {
            break;
        }
        if (strcmp("my_netstat", commandLine) == 0 ||
strcmp("my_netstat -p tcp -a", commandLine) == 0) {
            retrievingTCPConnectionTable();
        }
        else if (strcmp("my_netstat -a", commandLine) == 0) {
            retrievingTCPConnectionTable();
            retrievingUDPListenerTable();
        }
        else if (strcmp("my_netstat -p udp -a", commandLine) == 0) {
            retrievingUDPListenerTable();
        }
        else if (strcmp("my_netstat -s", commandLine) == 0)
        {
            showIpStatistics();
            showIcmpStatistics();
            showTcpStatistics();
            showUdpStatistics();
        }
        else {
            printf("\nPlease enter the valid command line!\n");
        }
    }
    system("pause");
}
```

2. Kết quả chạy chương trình

```

C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat
Protocol      State      Local Address      Remote Address
TCP           LISTEN     0.0.0.0 : 135         0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 445         0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 3306        0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 5040        0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 33060       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49664       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49665       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49666       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49667       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49668       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49669       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49670       0.0.0.0 : 0
TCP           LISTEN     0.0.0.0 : 49676       0.0.0.0 : 0
TCP           LISTEN     127.0.0.1 : 1001       0.0.0.0 : 0
TCP           LISTEN     127.0.0.1 : 5939       0.0.0.0 : 0
TCP           ESTABLISHED 127.0.0.1 : 49671     127.0.0.1 : 49672
TCP           ESTABLISHED 127.0.0.1 : 49672     127.0.0.1 : 49671
TCP           LISTEN     127.0.0.1 : 57633     0.0.0.0 : 0
TCP           LISTEN     192.168.1.6 : 139     0.0.0.0 : 0
TCP           ESTABLISHED 192.168.1.6 : 59699   52.230.7.59 : 443
TCP           ESTABLISHED 192.168.1.6 : 59707   157.240.15.16 : 443
TCP           ESTABLISHED 192.168.1.6 : 59709   74.125.204.188 : 443
TCP           ESTABLISHED 192.168.1.6 : 59730   49.213.114.134 : 443
TCP           CLOSE-WAIT 192.168.1.6 : 60580    23.79.101.35 : 443
TCP           CLOSE-WAIT 192.168.1.6 : 60581    23.79.101.35 : 443
TCP           CLOSE-WAIT 192.168.1.6 : 60582    23.53.215.25 : 80
TCP           CLOSE-WAIT 192.168.1.6 : 60583    23.53.215.25 : 80
TCP           CLOSE-WAIT 192.168.1.6 : 60585    113.171.231.16 : 443

```

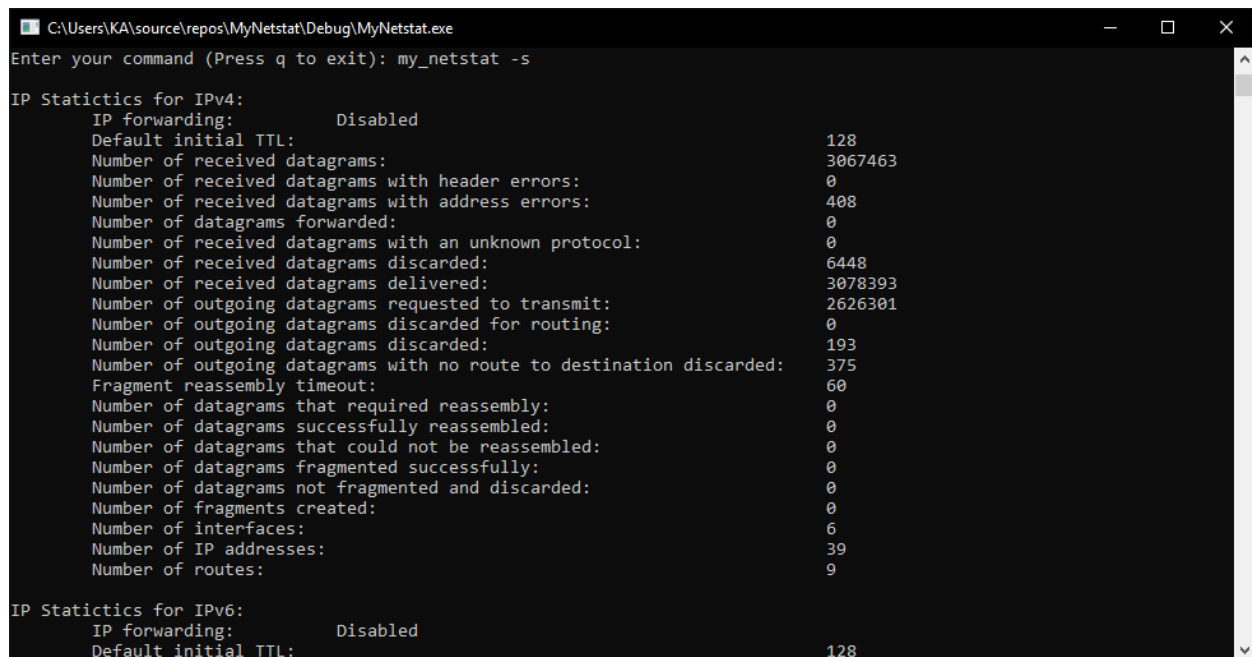
Hình 1: Kết quả chạy chương trình với tùy chọn my_netstat

```

C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat -p udp -a
Protocol      Local Address      Remote Address
UDP           0.0.0.0 : 500       0.0.0.0 : 0
UDP           0.0.0.0 : 4500      0.0.0.0 : 0
UDP           0.0.0.0 : 5050      0.0.0.0 : 0
UDP           0.0.0.0 : 5353      0.0.0.0 : 0
UDP           0.0.0.0 : 5353      0.0.0.0 : 0
UDP           0.0.0.0 : 5355      0.0.0.0 : 0
UDP           0.0.0.0 : 49665     0.0.0.0 : 0
UDP           0.0.0.0 : 49789     0.0.0.0 : 0
UDP           0.0.0.0 : 49790     0.0.0.0 : 0
UDP           0.0.0.0 : 52486     0.0.0.0 : 0
UDP           0.0.0.0 : 52487     0.0.0.0 : 0
UDP           0.0.0.0 : 52508     0.0.0.0 : 0
UDP           0.0.0.0 : 52509     0.0.0.0 : 0
UDP           0.0.0.0 : 53068     0.0.0.0 : 0
UDP           0.0.0.0 : 53069     0.0.0.0 : 0
UDP           0.0.0.0 : 54042     0.0.0.0 : 0
UDP           0.0.0.0 : 54043     0.0.0.0 : 0
UDP           0.0.0.0 : 61232     0.0.0.0 : 0
UDP           0.0.0.0 : 61233     0.0.0.0 : 0
UDP           127.0.0.1 : 1900     0.0.0.0 : 0
UDP           127.0.0.1 : 5353     0.0.0.0 : 0
UDP           127.0.0.1 : 49664     0.0.0.0 : 0
UDP           127.0.0.1 : 50331     0.0.0.0 : 0
UDP           169.254.138.134 : 52499 169.254.138.134 : 54026
UDP           192.168.1.2 : 54048 192.168.1.2 : 64847
UDP           192.168.1.2 : 64847 192.168.1.2 : 64847
UDP           192.168.1.6 : 137    0.0.0.0 : 0

```

Hình 2: Kết quả chạy chương trình với tùy chọn my_netstat -p udp -a



```
C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat -s

IP Statistics for IPv4:
  IP forwarding:           Disabled
  Default initial TTL:      128
  Number of received datagrams: 3067463
  Number of received datagrams with header errors: 0
  Number of received datagrams with address errors: 408
  Number of datagrams forwarded: 0
  Number of received datagrams with an unknown protocol: 0
  Number of received datagrams discarded: 6448
  Number of received datagrams delivered: 3078393
  Number of outgoing datagrams requested to transmit: 2626301
  Number of outgoing datagrams discarded for routing: 0
  Number of outgoing datagrams discarded: 193
  Number of outgoing datagrams with no route to destination discarded: 375
  Fragment reassembly timeout: 60
  Number of datagrams that required reassembly: 0
  Number of datagrams successfully reassembled: 0
  Number of datagrams that could not be reassembled: 0
  Number of datagrams fragmented successfully: 0
  Number of datagrams not fragmented and discarded: 0
  Number of fragments created: 0
  Number of interfaces: 6
  Number of IP addresses: 39
  Number of routes: 9

IP Statistics for IPv6:
  IP forwarding:           Disabled
  Default initial TTL:      128
```

Hình 3: Kết quả chạy chương trình với tùy chọn my_netstat -s

3. Đánh giá kết quả

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN