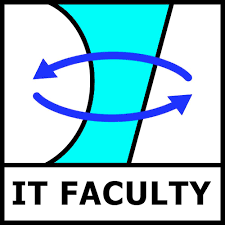


**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA CÔNG NGHỆ THÔNG TIN**

----------



***BÁO CÁO ĐỒ ÁN*  
CƠ SỞ NGÀNH MẠNG**

***ĐỀ TÀI***

1. **Tìm hiểu Hook windowns và xây dựng chương trình Keylog**
2. **Tìm hiều Ipv4 và xây dựng chương trình my\_scanip**

**GVHD**: **ThS.** **NGUYỄN THẾ XUÂN LY**

**SVTH**: **Lê Trung Nam**

**MSSV**: **102170036**

***Đà Nẵng****, 2020*

MỞ ĐẦU

Đồ án cơ sở ngành mạng là một trong những đồ án quan trọng của ngành Công Nghệ Thông Tin. Thông qua đồ án này em là người làm đồ án này sẽ đạt được những kiến thức cơ bản, nền tảng và có cơ hội được nghiên cứu chuyên sâu hơn về bộ môn *Lập trình mạng và Nguyên lí hệ điều hành.*

Trong đồ án này, về phần *Nguyên lí hệ điều hành*, đề tài nghiên cứ về Hook windowns và từ đó xây dựng chương trình keylog, còn phần *Lập trình mạng*, đề tài nghiên cứu, tìm hiểu về Ipv4 để rồi viết chương trình my\_scanip. Với sự đồng hành và hỗ trợ nhiệt tình của các thầy cô bộ môn ngành Mạng cùng sự nỗ lực cố gắng của bản thân, bản báo cáo *Đồ án cơ sở ngành mạng* đã hoàn thành đúng với thời hạn được giao.

Đặc biệt em xin trân trọng cảm ơn thầy Nguyễn Thế Xuân Ly đã hướng dẫn và chỉ bảo tận tình trong quá trình em nghiên cứu đồ án này.

***Sinh viên thực hiện***

Lê Trung Nam

MỤC LỤC

[MỞ ĐẦU 2](#_Toc57142226)

[MỤC LỤC 3](#_Toc57142227)

[PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH 4](#_Toc57142228)

[**CHƯƠNG 1. CỞ SỞ LÍ THUYẾT** 4](#_Toc57142229)

[**1.** **Hook là gì?** 4](#_Toc57142230)

[**2.** **Phân loại Hook** 4](#_Toc57142231)

[**2.1. Phân loại theo phạm vi hoạt động** 4](#_Toc57142232)

[**2.2. Phân loại theo thông điệp xử lý** 4](#_Toc57142233)

[**3.** **Cách sử dụng** 4](#_Toc57142234)

[**3.1** **Cài đặt hook** 4](#_Toc57142235)

[**3.2.** **Giải phóng Hook** 4](#_Toc57142236)

[**4.** **Win 32 API** 4](#_Toc57142237)

[**4.1.** **Giới thiệu** 4](#_Toc57142238)

[**4.2.** **Các hàm thường dùng** 4](#_Toc57142239)

[**CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG** 5](#_Toc57142240)

[**1.** **Yêu cầu** 5](#_Toc57142241)

[**1.1.** **Yêu cầu chức năng** 5](#_Toc57142242)

[**1.2.** **Yêu cầu về giao diện** 5](#_Toc57142243)

[**2.** **Phân tích và thiết kế hệ thống** 5](#_Toc57142244)

[**CHƯƠNG 3. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ** 5](#_Toc57142245)

[PHẦN II. LẬP TRÌNH MẠNG 5](#_Toc57142246)

[**CHƯƠNG 1. CƠ SỞ LÝ THUYẾT** 5](#_Toc57142247)

[**1.** **IPV4 là gì** 5](#_Toc57142248)

[**2.** **Cấu trúc địa chỉ Ipv4** 5](#_Toc57142249)

[**3.** **Các lớp địa chỉ Ipv4** 5](#_Toc57142250)

[**CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG** 5](#_Toc57142251)

[**1.** **Phân tích các yêu cầu** 5](#_Toc57142252)

[**2.** **Phân tích và thiết kế hệ thống** 5](#_Toc57142253)

[**CHƯƠNG 3. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ** 5](#_Toc57142254)

[**1.** **Cài đặt chương trình** 5](#_Toc57142255)

[**2.** **Kết quả thực hiện** 5](#_Toc57142256)

# PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH

**TIÊU ĐỀ: Tìm hiểu về Hook windowns và xây dựng chương trình Keylog**

## **CHƯƠNG 1. CỞ SỞ LÍ THUYẾT**

### **Hook là gì?**

Hook là một cơ chế trong lập trình sự kiện, cho phép ứng dụng có thể cài đặt một hàm giám sát vào quá trình lưu chuyển các thông điệp. Hay nói cách khác Hook là một cơ chế cho phép chặn các sự kiện (chuột, bàn phím, thông điệp) trước khi chúng được gửi tới hàng đợi của ứng dụng. Hàm dùng để chặn một loại sự kiện riêng biệt được gọi là Hook procedure/ Hook function, khi nhận được sự kiến Hook procedure có thể thay đổi hoặc có thể loại bỏ sự kiện đó.

Hook có khả năng can thiệp rất sâu vào hệ thống, nó có thể làm hệ thống hoạt động chậm chạp hơn hoặc thập chí có thể làm treo hệ thống nếu như xử lý không tốt. Vì thế chúng ta chỉ nên sử dụng Hook trong những trường hợp cần thiết và phải loại bỏ nó khi đã hoàn thành xong công việc.

### **Phân loại Hook**

**2.1. Phân loại theo phạm vi hoạt động**

* Hook cục bộ (Thread Hook hay loacal Hook): là một kỹ thuật Hook dùng để bẫy sự kiện ngay trong tiến trình cài đặt, hàm giám sát được đặt sau hàng đợi thông điệp ứng dụng.
* Hook toàn cục (Global Hook): hàm giám sát được cài đặt sau hàng đợi thông điệp hệ thống, kiểm soát toàn bộ tiến trình trong hệ thống. Thủ tục hook toàn cục phải được phai báo ở một dll (Dynamic link library) riêng biệt.

#### **2.2. Phân loại theo thông điệp xử lý**

|  |  |
| --- | --- |
| **Loại Hook** | **Tác dụng** |
| **WH\_CALLWNDPROC** | Cho phép giám sát các thông điệp được gởi tới một cửa sổ. Hệ thống gọi thủ tục Hook của WH\_CALLWNDPROC trước khi gởi thông điệp đến cửa sổ đích và gọi WH\_CALLWNDPROCRET sau khi thủ tục ở cửa sổ đích xử lý xong thông điệp. |
| **WH\_CALLWNDPROCRET** | Hook WH\_CALLWNDPROCRET truyền một con trỏ có cấu trúc CWPRETSTRUCT tới thủ tục Hook. Cấu trúc này chứa giá trị trả về của thủ tục cửa sổ đã xử lý thông điệp |
| **WH\_CBT** | Windows gọi hàm Hook\_CBT trước khi tạo lập, kích hoạt, huỷ, thu nhỏ, phóng to, di chuyển, thay đổi kích thước, ... của sửa sổ giao diện, hoặc trước khi huỷ bỏ một sự kiển chuột hay bàn phím khỏi hàng đợi của hệ thống, hoặc trước khi đặt một điều khiển vào một input nào đó; hoặc trong lúc đồng bộ hàng đợi thông điệp hệ thống. Giá trị trả về của thủ tục Hook cho biết hệ thống sẽ chấp nhận hay la huỷ bỏ những hành động đó hay không. Hook WH\_BCT thường được dùng cho chương trình đào tạo trên máy tính. |
| **WH\_DEBUG** | Hệ thống gọi thủ tục WH\_DEBUG trước khi gọi các thủ hook tục khác trong hệ thống. Ta có thể dùng hook này để xác định xem có cho phép các thủ tục hook khác hay không. |
| **WH\_FOREGROUNDIDLE** | Hook này cho phép ta thực thi các tác vụ với mức ưu tiên thấp khi các tiến trình chạy nền của nó được đặt ở trạng thái nghỉ. Hệ thống gọi thủ tục WH\_FORGETGROUNDIDLE khi chương trình chạy nền chuẩn bị chuyển sang chế độ nghỉ ngơi. |
| **WH\_GETMESSAGE** | Hook này cho phép một chương trình giám sát các thông điệp được trả về bởi các hàm GetMessage và PeekMessage. Ta có thể dùng hook WH\_GETMESSAGE để giám sát sự kiện chuột và bàn phím và các sự kiện khác được gửi đến hàng đợi thông điệp. |
| **WH\_JOURNALPLAYBACK** | Cho phép chèn chương trình chèn thông điệp vào hàng đợi của hệ thống. Ta có thể dùng hook này để chạy lại (play back) các sự kiện chuột, bàn phím được ghi lại bởi hook WH\_JOURNALRECORD. Khi hook WH\_JOURNALPLAYBACK được cài đặt, chuột, bàn phím sẽ bị đóng băng. Hook này chỉ dành riêng cho hook hệ thống, ta không thể dùng để cài đặt hook cục bộ. Trả về một giá trị time-out, giá trị này cho hệ thống biết phải chờ bao lâu trước khi xử lý thông điệp hiện tại nhận được từ hook WH\_JOURNALPLAYBACK. Điều này cho phép hook có thể điều khiển thời gian của sự kiện mà nó play back. |
| **WH\_JOURNALRECORD** | Cho phép giám sát và ghi lại các sự kiện vào (input event). Hook này thường được dùng để ghi lại chuỗi các sự kiện chuột và bàn phím để sau đó được phát lại nhờ hook WH\_JOURNALPLAYBACKHOOK. Hook này chỉ dành riêng để hook hệ thống, ta không thể dùng để cài đặt hook cục bộ. |
| **WH\_KEYBOARD\_LL** | Cho phép giám sát sự kiện vào của bàn phím được gửi tới hàng đợi ứng dụng. |
| **WH\_KEYBOARD** | Cho phép giám sát thông điệp từ bàn phím WH\_KEYDOWN, WM\_KEYUP |
| **WH\_MOUSE\_LL** | Cho phép giám sát sự kiện vào của chuột được gửi tới hàng đợi ứng dụng. |
| **WH\_MOUSE** | Cho phép giám sát thông điệp của chuột được trả về bởi hàm GetMessage hoặc PeekMessage. Ta có thể dùng hàm này để theo giõi sự kiện nhập từ chuột được gửi đến hàng đợi thông điệp. |
| **WH\_MSGFILTER** | Xử lý hoặc thay đổi tất cả các thông điệp cho hộp thoại cửa sổ, hộp thoại thông điệp hoặc là menu của chương trình |
| **WH\_SYSMSGFILTER** | Xử lý hoặc thay đổi tất cả các thông điệp cho hộp thoại cửa sổ, hộp thoại thông điệp hoặc là menu của hệ thống. |
| **WH\_SHELL** | Một chương trình shell có thể dùng hook WH\_SHELL để nhận các thông báo quan trọng. Hệ thống gọi một thủ tục WH\_SHELL khi chương trình shell được kích hoạt và khi một cửa sổ mức cao (top-level) được khởi tạo hoặc huỷ bỏ. |

### **Cách sử dụng**

* 1. **Cài đặt hook**

Bằng việc gọi hàm **SetWindowsHookEx** sẽ cài đặt thủ tục Hook vào điểm bắt đầu của chuỗi Hook, việc cài đặt Hook có thể được thực hiện trên mọi tiến trình trên hệ thống.

Nếu chúng ta muốn sử dụng Hook toàn cục thì cần phải đặt trong thư viện liên kết động (dll).

Để sử dụng thư viện liên kết động thì phải lấy được handle của thư viện đó, để nhận handle chúng ta có thể sử dụng hàm **LoadLibrary** với tham số là tên của thư viện. Khi có được Handle tiếp đến lấy địa chỉ của thủ tục Hook trong DLL bằng cách sử dụng hàm **GetProcAddress.** Sau khi đã có địa chỉ thủ tục Hook cuối cùng ta sẽ dùng hàm **SetWindowsHookEx** để cài đặt thủ tục Hook vào chuỗi Hook.

HHOOK SetWindowsHookExA (

int idHook,

HOOKPROC lpfn,

HINSTANCE hmod,

DWORD dwThreadId

);

Trong đó:

* **idHook**: là loại thủ tục được cài đặt
* **Ipfn**: con trỏ tới thủ tục Hook
* **hMod**: Handle đến file dll chứa thủ tục Hook chỉ bởi con trỏ Ipfn
* **dwThreadId**: đinh danh Thread mà thủ tục Hook được liên kết, nếu bằng 0 thì Hook sẽ là Gloabal.
  1. **Giải phóng Hook**

Để loại bỏ Hook chúng ta sẽ sử dụng hàm **UnhookWindowsHookEx.**

Với Hook Global thì hàm này không thể trả tự do cho hàm DLL

### **Win 32 API**

* 1. **Giới thiệu**

Đây là bộ giao diện lập trình ứng dụng (API) cốt lõi của Microsoft có sẵn trong hệ điều hành Windows. Công cụ hỗ trợ phát triển có sẵn trong cái phiên bản của Microsoft Windows SDK, cung cấp tài liệu cà các công cụ cần thiết để xây dựng phần mềm dựa trên giao diện API Windows và các Windows được liên kết.

API Windows (Win32) tập trung chủ yếu vào ngôn ngữ lập trình C ở chỗ các chức năng và cấu trúc dữ liệu được hiển thị của nó được mô tả bằng ngôn ngữ đó. Tuy nhiên, các API có thể sử dụng bởi bất kỳ ngôn ngữ lập **trình biên dịch.**

* 1. **Các hàm thường dùng**
* **SetWindownsHookExA**

HHOOK SetWindowsHookExA (

int idHook,

HOOKPROC lpfn,

HINSTANCE hmod,

DWORD dwThreadId

);

Trong đó:

* **idHook**: là loại thủ tục được cài đặt
* **Ipfn**: con trỏ tới thủ tục Hook
* **Ipfn**: con trỏ tới thủ tục Hook
* **hMod**: Handle đến file dll chứa thủ tục Hook chỉ bởi con trỏ Ipfn
* **dwThreadId**: đinh danh Thread mà thủ tục Hook được liên kết, nếu bằng 0 thì Hook sẽ là Gloabal.
* **UnhookWindownHookEx**

BOOL UnhookWinowsHookEx (

HHOOK hhk

);

Trong đó:

* **hhk**: Xử lý hook được gỡ, tham số này là một handle của hook lấy được khi gọi hàm SetWindowsHookExA ở trước.
* **CallNextHookEx**

LRSULT CallNextHookEx (

HOOK hhk,

Int nCode,

WPARAM wParam,

LPARAM lParam

);

Trong đó:

* **hhk:** bỏ qua tham số này điền null
* **nCode:** Mã Hook được truyền vào hàm thủ tục Hook hiện tại, thủ tục Hook tiếp theo sử dụng tham số này để biết cách xử lý.
* **wParam:** Giá trị wParam được truyền vào thủ tục Hook hiện tại. Ý nghĩa của tham số này phụ thuộc vào loại Hook liên quan với các dây chuyền Hook hiện tại.
* **lParam:** Giá trị lParam được truyền vào thủ tục Hook hiện tại. Ý nghĩa của tham số này này phụ thuộc vào loại Hook liên quan với các dây chuyền Hook hiện tại.
* **LowLevelKeyboardProc**

LRESULT CALLBACK LowLevelKeyboardProc (

\_In\_ int nCode,

\_In\_ WPARAM wParam,

\_In\_ LPARAM lParam

);

Trong đó:

* **nCode**: mã thủ tục của Hook, dùng để xác định thông điệp
* **wParam**: định danh của thông điệp ban phím
* **lParam**: con trỏ đến cấu trúc **KBDLLHOOKSTRUCT**:

***typedef struct tagKBDLLHOOKSTRUCT {***

***DWORD*** *vkCode;*

***DWORD*** *scanCode;*

***DWORD*** *flags;*

***DWORD*** *time;*

***ULONG\_PTR*** *dwExtraInfo;*

***}*** *KBDLLHOOKSTRUCT, \*LPKBDLLHOOKSTRUCT, \*PKBDLLHOOKSTRUCT;*

Trong đó:

* **vkCode**: mã bàn phím ảo, có giá trị từ 1 đến 254
* **scanCode**: mã quét phần cứng của bàn phím
* **flags**: để kiểm tra giá trị tổ hơp các phím
* **time**: thời gian chạy thôn điệp này
* **dwExraInfo**: thông tin bổ sung liên quan đến thông điệp
* **GetKeyState**

Lấy trạng thái của phím ảo được chỉ định, trả về 1 là xuống, còn 0 là lên.

SHORT GetKeyState (

Int nVirtKey

);

* **GetKeyboardState**

Sao chép tình trạng của 256 phím ảo đến bộ đệm chỉ định

BOOL GetKeyboardState (

PBYTE lpKeyState

);

* **ToAscii**

Dịch mã phím ảo và trạng thái bàn phím được chỉ định sang ký tự hoặc các ký tự tương ứng

int ToAscii (

UINT uVirtKey,

UINT uScanCode,

const BYTE \*lpKeyState,

LPWORD lpChar,

UINT uFlags

);

Trong đó:

* **uVirtKey**: mã phím ảo cần dịch
* **uScanCode**: mã quét phần cứng của phím cần dịch
* **lpKeyState**: con trỏ đến 1 mảng 256-byte chứa các trạng thái bàn phím hiện tại
* **lpChar**: Bộ đệm nhận ký tự hoặc các ký tự đã dịch.
* **uFlags**: Tham số này phải là 1 nếu một menu đang hoạt động, hoặc 0 nếu không.
* **LoadLibraryA**

Tải module được chỉ định vào không gian địa chỉ của tiến trình gọi. Module được chỉ định có thể làm cho các module khác được tải.

HMODULE LoadLibraryA (

LPCSTR lpLibFileName

);

Trong đó:

* **lpLibFileName**: tên của module, có thể là tên của thư viện (.dll) hoặc module thi hành (.exe). Tên chỉ định là tên của tập tin của module và không liên quan tên chứa trong module. Xác định trong từ khoá thư viện tập tin (.def)
* **GetProcAddress**

Truy xuất địa chỉ của một hàm hoặc biến đã xuất từ ​​thư viện liên kết động (dll) được chỉ định.

FARPROC GetProcAddress (

HMODULE hModule,

LPCSTR lpProcName

);

Trong đó:

* **hModule**: handle của thư viện dll chứa hàm hoặc biến. Handle này là giá trị trả về của các hàm LoadLibrary, LoadLibraryEx, LoadPackageLibrary hoặc GetModuleHandle.
* **lpProcName**: Tên hàm hoặc biến hoặc giá trị thứ tự của hàm. Nếu tham số này là một giá trị thứ tự, nó phải ở từ bậc thấp; từ bậc cao phải bằng 0.

**CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

1. **Yêu cầu**
   1. **Yêu cầu chức năng**

Chương trình sẽ ghi lại toàn bộ thao tác bàn phím của người dùng máy tính và lưu lại trong một tập tin (.txt)

* 1. **Yêu cầu về giao diện**

Vì đây là một chương trình có tính chất gián điệp nên sẽ không có đồ hoạ, để biết được chương trính có chạy hay không người dùng có thể xem trong task manager.

* 1. **Yêu cầu về tương thích**

Do Win 32 API hỗ trợ tốt trên ngôn ngữ C/C++ nên em lựa chọn ngôn ngữ C++ để viết chương trình, công cụ sử dụng là DevC và Visual Studio Code.

1. **Phân tích và thiết kế hệ thống**

**CHƯƠNG 3. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ**

# PHẦN II. LẬP TRÌNH MẠNG

**Tiêu Đề: Tìm hiểu Ipv4 và xây dựng chương trình my\_scanIP**

## **CHƯƠNG 1. CƠ SỞ LÝ THUYẾT**

### **IPV4 là gì**

Ipv4 (Internet Protocol Version 4) là phiên bản thứ tư trong quá trình phát triển của các giao thức internet. IP –Internet Protocol là một giao thức của chồng giao thức TCP/IP thuộc về lớp Internet, tương ứng với lớp thứ 3 của ( lớp Network) của mô hình OSI. Hiện nay , IP được sử được sử dụng rộng rãi trong mọi hệ thống mạng trên toàn thế giới.

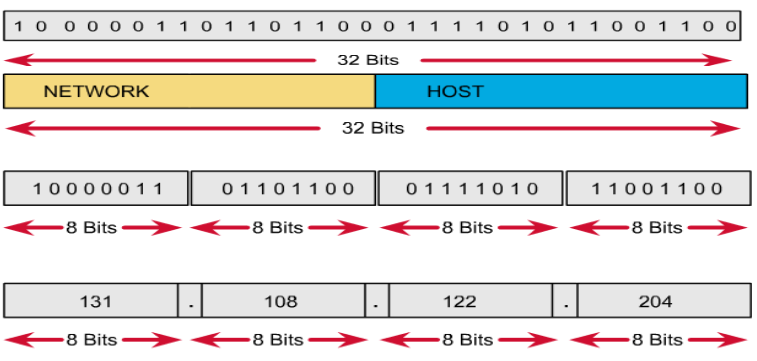
Ipv4 là giao thức hướng dữ liệu, được sử dụng cho hệ thống chuyển mạch gói. Đây là giao thức hoạt động dựa trên nguyên tắc truyền tốt nhất có thể, trong đó nó không quan tâm đến thứ tự gói tin truyền cũng như không đảm báo gói tin đến đích hay việc lặp gói tin ở đích, việc xử lý vấn đề này dành cho lớp trên của chồng giao thức TCP/IP. Tuy nhiên thì Ipv4 có cơ chế đảm bảo tính toàn vẹn dữ liệu thông qua sử dụng những gói kiểm tra (checksum).

### **Cấu trúc địa chỉ Ipv4**

Mỗi địa chỉ IP bao gồm 32 bit nhị phân, được chia thành 4 cụm 8 bits (gọi là các octet). Để tránh sai sót và dễ đọc, viết thì mỗi octet được biểu diễn dưới dạng thập phân và được ngăn cách bởi dấu chấm.

Mỗi địa chỉ IP bao gồm 2 phần, phân thứ nhất dùng để xác định chính xác nhà mạng X nào đó mà hệ thống kết nối đến, phần này được gọi là phần mạng (network portion - N) và phần thứ 2 dùng để xác định chính xác là máy tính Y nào đó trên mạng X được gọi là phần thiết bị đầu cuối người sử dụng (host portion -H)

Vì thế mỗi địa chỉ IP phải là duy nhất để đảm bảo chính xác khi truyền nhận dữ liệu



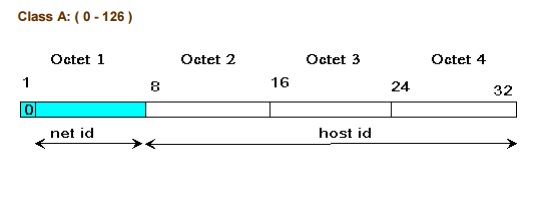
IP và SM (Subnet Mask) có thể được viết vắn tắt là IP/ Tổng số bit 1 trong SM. Từ đó ta có cấu trúc địa chỉ Ipv4 có dạng

t/u

Ví dụ: 192.168.1.15/24

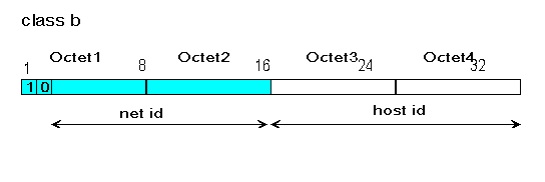
### **Các lớp địa chỉ Ipv4**

* Lớp A: **x.y.z.t/8 với x ; y,z,t**

****

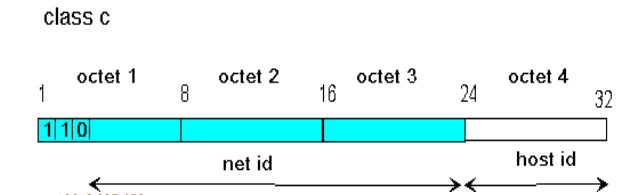
Địa chỉ lớp A sử dụng một octet đầu tiên làm phần mạng, 3 octet còn lại làm phần host. Bit đầu tiên của địa chỉ lớp A luôn là 0. Do 0 và 127 được sử dụng cho mục đích chuyên dụng nên các địa chỉ của mạng lớp A gồm : 1.0.0.0 đến 126.0.0.0. Phần host có 24 bits nên mỗi lớp mạng A có () host

* Lớp B: **x.y.z.t/16 với x ; y,z,t**

****

Địa chỉ lớp B sử dụng 2 octet đầu làm phần mạng, 2 octet sau làm phần host. Hai bits đầu tiên của địa chỉ lớp B luôn là 1 0. Các địa chỉ của lớp mạng B bao gồm 128.0.0.0 đến 191.255.0.0. Phần mạng của lớp B có 16 bit nên tổng số mạng trong lớp B là ( ) và số host trên mỗi mạng lớp B là () host.

* Lớp C: **x.y.z.t/24 với x ; y,z,t**



Địa chỉ lớp mạng C sử dụng 3 octet đầu làm phần mạng, 1 octet sau làm phần host. Ba bits đầu tiên của một địa chỉ lớp C luôn là 1 1 0. Các địa chỉ của lớp mạng C bao gồm 192.0.0.0 đến 223.255.255.0. Phân mạng của lớp C có 24 bit nên tổng số mạng trong lớp C là và số host trên mỗi mạng lớp C là (.

* Lớp C: **x.y.z.t/32 với x ; y,z,t**

Địa chỉ lớp mạng D có u = 32 nên địa chỉ của lớp mạng D ta có thể viết gọn lại thành **x.y.z.t.** Bốn bits đầu tiên của địa chỉ lớp D luôn là 1110

Các địa chỉ của lớp mạng D bao gồm từ 224.0.0.0 đến 239.255.255.0. Lớp này được tạo ra nhằm hỗ trợ cập nhật thông tin giữa các bộ định tuyến (truyền thông 1 – nhiều : multicasting).

* Lớp E**: x.y.z.t/u với**

Gồm các địa chỉ từ giải 240.0.0.0 trở đi, Bốn bits đầu tiền của lớp E luôn là 1111.

## **CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

### **Phân tích các yêu cầu**

* Ứng dụng phải đảm bảo các yêu cầu như sau:

+ Ping đến tất cả các địa chỉ có trong mạng LAN

### **Phân tích và thiết kế hệ thống**

* Sơ đồ hoạt động

Hiên thị danh sách IP có trong mạng LAN

Sử dụng đa luồng để quét từng địa chỉ trong dãy địa chỉ

Tính ra dãy địa chỉ cần quét

Lấy địa chỉ IP máy (address), Subnet mask

Tính địa chỉ network ( lấy địa chỉ address & subnet mask)

* Môi trường triển khai

+ Ngôn ngữ lập trình : Java

+ Công cụ lập trình: Visual Studio Code

+ Hệ điểu hành: Windown 10

## **CHƯƠNG 3. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ**

### **Cài đặt chương trình**

* Lấy địa chỉ IP máy

****

* Tính Subnet mask

****

* Tính địa chỉ network

****

* Dãy địa chỉ cần quét

****

* Ping đến từng địa chỉ

****

### **Kết quả thực hiện**

