实验内容

1、 设计路由表所采用的数据结构。要求能够根据目的 IPv4 地址来确定分组处理行为。
2、 IPv4 分组的接受和发送
3、 IPv4 分组的转发。对于需要转发的分组进行处理，获得下一跳的 IP 地址，然后调用发送接口函数做进一步处理

问题

之前用字典树的方法来进行最长前缀匹配，可是总是运行到如下 if 一句报错。

```
if (subRoot->next[keyPart[0]] == NULL)
                subRoot->next[keyPart[0]] = new Node();
```

值得注意的是，把 if 注释掉之后，直接运行下面这句是可以的。

```
subRoot->next[keyPart[0]] = new Node();
```

用写文件的方法 debug，确定 keyPart[0]是否数组越界，判断 subRoot 是不是 NULL，可是发现直到报错也没有越界，subRoot 也不是 NULL，百思不得其解

解决方法

最后选择放弃字典树，采用最笨的循环判断前缀的方法。

代码

```
/*
* THIS FILE IS FOR IP FORWARD TEST
*/
#include "sysInclude.h"
#include <vector>
using namespace std;

// system support
extern void fwd_LocalRcv(char *pBuffer, int length);

extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop);

extern void fwd_DiscardPkt(char *pBuffer, int type);

extern unsigned int getIpv4Address( );

// implemented by students
struct item{
   int dst;
   int nexthop;
};
```

```
vector<item> table; // Router Table
void stud_Route_Init()
{
    table.clear();
    return;
}

void stud_route_add(stud_route_msg *proute)
{
    item tmp;
    tmp.dst = ntohl(proute->dest) & (0xffffffff << (32 - ntohl(proute->masklen)));
    tmp.nexthop = ntohl(proute->nexthop);
    table.push_back(tmp);
    return;
}

int matchPrefix(int prefix, int destIP){
    char prefix_bitmap[33];
    char destIP_bitmap[33];
    unsigned int tmp = 0x80000000;
    int prefix_length = 0;
    while (prefix != 0){
        prefix_bitmap[prefix_length] = (prefix & tmp != 0);
        if (prefix_bitmap[prefix_length])
            prefix -= tmp;
        prefix_length++;
        tmp = tmp >> 1;
    }
    prefix_bitmap[prefix_length] = -1;
    for (int i = 0; i < 32; ++i){
        // 走到这里表示前缀都匹配完了，可以直接返回长度了
        if (prefix_bitmap[i] == -1)
            return i;
        // destIP 和对应位与一下，如果是 0 的话表示 prefix 不是前缀，那么直接返回-1
        if (destIP & (prefix_bitmap[i] << (31 - i)) == 0)
            return -2;
    }
    // 走到这里表示 prefix 是 32 位，第 33 位才是-1，那么表示前缀是 32
    return 32;
}


int stud_fwd_deal(char *pBuffer, int length)
{
```

```
    // same as the previous lab
    int version = pBuffer[0] >> 4;
    int ihl = pBuffer[0] & 0xf;
    int ttl = (int)pBuffer[8];
    int checkSum = ntohs(*(short unsigned int*)(pBuffer + 10));
    int dstip = ntohl(*(unsigned int*)(pBuffer + 16));
    if (dstip == getIpv4Address()){
        fwd_LocalRcv(pBuffer, length);
        return 0;
    }
    if (ttl <= 0){
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
        return 1;
    }
    vector<item>::iterator it;
    int maxlen = -1;
    item maxlenItem;
    for (it = table.begin(); it != table.end(); it++){
        int tmp = matchPrefix(it->dst, dstip);
        // 不匹配会返回-2，肯定不会进入这个 if
        if (tmp > maxlen){
            maxlen = tmp;
            maxlenItem = *it;
        }
    }
    // 遍历之后会得到最大匹配长度和最大匹配的表项
    if (maxlen >= 0){
            char* buf = new char[length];
            memcpy(buf, pBuffer, length);
            buf[8]--;
            unsigned short int newchecksum = 0;
            for (int i = 0; i < 2 * ihl; ++i){
                if (i == 5)
                    continue;
                newchecksum += (buf[i*2] << 8) + (buf[i*2+1]);
            }
            newchecksum = htons(0xffff-newchecksum);
            memcpy(buf + 10, &newchecksum, sizeof(unsigned short int));
            fwd_SendtoLower(buf, length, maxlenItem.nexthop);
            return 0;
    }
    fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
    return 1;
}
```

错误而没有找出错误的代码

```
/*
* THIS FILE IS FOR IP FORWARD TEST
*/
#include "sysInclude.h"
#include <map>
using namespace std;

// system support
extern void fwd_LocalRcv(char *pBuffer, int length);

extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop);

extern void fwd_DiscardPkt(char *pBuffer, int type);

extern unsigned int getIpv4Address();

// implemented by student
struct Node
{
	bool isleaf;
	unsigned int nexthop;
	Node *next[2];
	Node(){
		isleaf = false;
		nexthop = 0;
		for (int i = 0; i < 2; ++i)
			next[i] = NULL;
	}
};
struct Table
{
	Node *root;

	Table(){
		root = new Node();
	}

	void clear(){
		for (int i = 0; i < 2; ++i)
			freeSubTree(root.next[i]);
		delete root;
```

```
}
void freeSubTree(Node *subRoot){
	if (subRoot == NULL)
		return;
	for (int i = 0; i < 2; ++i)
		freeSubTree(subRoot->next[i]);
	delete subRoot;
}

void add(unsigned int key, unsigned int value){
	char keyPart[32];
	int length = 0;
	unsigned int tmp = 0x80000000;
	while (key != 0){
		keyPart[length] = (key & tmp != 0);
		if (keyPart[length]){
			key -= tmp;
			keyPart[length] = '1';
		}
		else
			keyPart[length] = '0';
		tmp = tmp >> 1;
		++length;
	}
	keyPart[length] = 0;
	 recursionAdd(root, keyPart, length, value);
}

// this function can be private
void recursionAdd(Node *subRoot, char *keyPart, int length, unsigned int value){
	if (length == 0){
		subRoot->isleaf = true;
		subRoot->nexthop = value;
		return;
	}
	// this `if` is wrong! Why??
	if (subRoot->next[keyPart[0] - '0'] == NULL)
		subRoot->next[keyPart[0] - '0'] = new Node();
	 recursionAdd(subRoot->next[keyPart[0] - '0'], keyPart + 1, length - 1, value);
}

bool getValue(unsigned int key, unsigned int &value){
	char keyPart[32];
	int length = 0;
```

```
        unsigned int tmp = 0x80000000;
        while (key != 0){
            keyPart[length] = (key & tmp != 0);
            if (keyPart[length]){
                key -= tmp;
                keyPart[length] = '1';
            }
            else
                keyPart[length] = '0';
            tmp = tmp >> 1;
            ++length;
        }
        keyPart[length] = 0;
        return recursionGetValue(root, keyPart, length, value);
    }

    // 这个函数根据最长公告前缀得到值，如果得到则返回 true
    bool recursionGetValue(Node *subRoot, char *keyPart, int length, unsigned int &value){
        if (subRoot->next[keyPart[0] - '0'] == NULL){
            if (subRoot->isleaf == true){
                value = subRoot->nexthop;
                return true;
            }
            return false;
        }

        if (recursionGetValue(subRoot->next[keyPart[0] - '0'], keyPart + 1, length - 1, value) ==
true)
            return true;

        if (subRoot->isleaf == true){
            value = subRoot->nexthop;
            return true;
        }
        return false;
    }
};
Table table;

void stud_Route_Init()
{
    table.clear();
    return;
}
```

```
void stud_route_add(stud_route_msg *proute)
{
	unsigned int dest = ntohl(proute->dest) & (0xffffffff << (32 - ntohl(proute->masklen)));
	table.add(dest, (unsigned int)ntohl(proute->nexthop));
	return;
}


int stud_fwd_deal(char *pBuffer, int length)
{
	// get the variable
	char *tempIndex = pBuffer;
	unsigned short version = ((unsigned char)tempIndex[0]) >> 4;
	unsigned short IHL = tempIndex[0] & 15;
	// be careful to add 8 to index;
	tempIndex = tempIndex + 8;
	unsigned short ttl = ((unsigned char)tempIndex[0]);
	tempIndex += 2;
	unsigned int headerChecksum = ((unsigned short *)tempIndex)[0];
	tempIndex += 2;
	unsigned int srcip = ntohl(*((unsigned int *)tempIndex));
	tempIndex += 4;
	unsigned int dstip = ntohl(*((unsigned int *)tempIndex));

	if (version != 4){
		fwd_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
		return 1;
	}
	if (IHL < 5){
		fwd_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
		return 1;
	}
	if (ttl == 0){
		fwd_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
		return 1;
	}
	// it's hard to identify the boardcast packet, because boardcast packet
	// need host to all 1, not ip to all 1.
	if (dstip == getIpv4Address()){
		fwd_LocalRcv(pBuffer, length);
		return 0;
	}
```

```
    unsigned int nexthop;
    if (table.getValue(dstip, nexthop)){
        char* buf = new char[length];
        memcpy(buf, pBuffer, length);
        buf[8]--;
        unsigned short int newchecksum = 0;
        for (int i = 0; i < 2 * ihl; ++i){
            if (i == 5)
                continue;
            newchecksum += (buf[i * 2] << 8) + (buf[i * 2 + 1]);
        }
        newchecksum = htons(0xffff - newchecksum);
        memcpy(buf + 10, &newchecksum, sizeof(unsigned short int))
        fwd_SendtoLower(buf, length, nexthop);
        return 0;
    }
    else{
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return 1;
    }
}
```