实验报告

孔静 2014K8009929022

November 20, 2017

Contents

1 实验题目

- 高效 IP 路由查找实验

2 实验内容

- 实现最基本的前缀树查找
- 实现多 bit 前缀树及优化（叶推 + 压缩指针 + 压缩向量）
- 基于上次课程中给出的 forwarding-table.txt
  - 以最基本的前缀为基准，检查所实现的多 bit 前缀树是否正确
  - 对比两种不同方法的性能

3 实验流程

3.1 文件列表

```
lab9
├── main.c
├── normal_search.c
├── normal_search.h
├── advanced_search.c
├── advanced_search.h
└── forwarding-table.txt
```

3.2 基础搜索

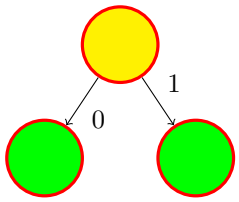
- 数据结构

```
typedef struct node{
  struct node *left, *right; //左右子叶
  u32 prefix; //前缀
  u16 port; //端口
  u8 length; //前缀长度
  u8 type; //是否有效
}node_t;
```

- 函数功能

```
void normal(); //生成树，基于forwarding-table.txt测试，释放树

u16 normal_search(node_t *root_node, u32 ip); //在已有树基础上查找并返回port
void scanf_file(node_t *root_node); //读取文件，并生成树
node_t *init_node_t(u32 prefix, u8 length, u16 port); //malloc并初始化节点
void insert_node_t(node_t *root_node, u32 ip, u8 length, u16 port); //将读取的条目插入
    树中
void free_tree(node_t *root_node); //整棵树的空间释放

void scanf_ip(char *src, u32 *ip); //读取前缀
void scanf_prefix_length(char *src, u8 *length); //读取前缀长度
void scanf_port(char *src, u16 *port); //读取端口
void scanf_row(char *src, u32 *ip, u8 *length, u16 *port); //读取一行条目
```

- 存储图示
  - 每个前缀按位从根节点，0 左 1 右，向下插入
  - 查找同理，从根节点向下进行匹配



3.3 高级搜索

- 数据结构

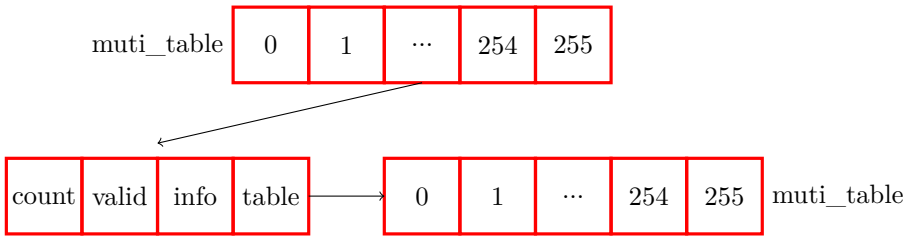
```
typedef struct info{
  u32 prefix; //前缀
  u16 port; //端口
  u8 length; //前缀长度
}info_t;
typedef struct muti_table{
  u8 count; //多少个比特数节点指向该节点
  u8 valid; //是否有效
  info_t *info; //前缀信息
  struct muti_table **table; //下一级多比特表
}muti_table_t;
```

- 函数功能

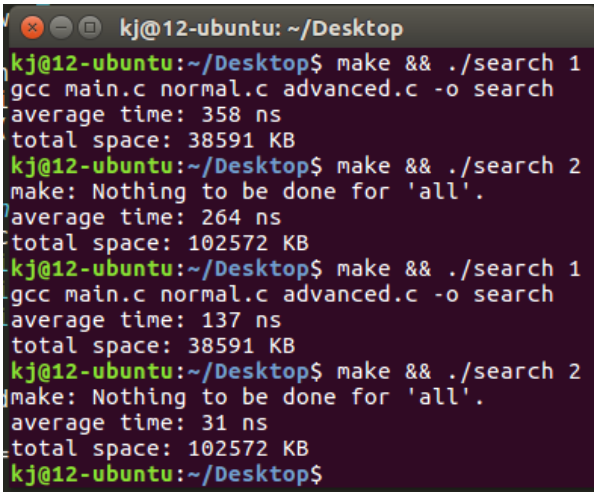
```
void advanced(); //生成树，基于forwarding-table.txt测试，释放树

u16 advanced_search(muti_table_t **muti_table, u32 ip, u8 length); //在已有树基础上查找
    并返回port
```

```
4  void insert_muti_table(muti_table_t **muti_table, u32 ip, u8 length, info_t *info, int
       shift); //将条目插入多比特表
5
6  void scanf_file_advanced(hash_table_t **hash_table, muti_table_t **muti_table); //读取
       条目，并生成多比特表，(hash_table已被弃用，本为存储info信息的哈希表)
7
8  info_t *init_info(u32 ip, u8 length, u16 port); //malloc并初始化节点
9
10 muti_table_t *init_muti_table(info_t *info, int max_length); //malloc并初始化多比特表的
       条目
11 muti_table_t **create_muti_table(); //malloc并生虫化多比特表
12 void free_muti_table(muti_table_t **muti_table); //释放整个多级多比特表
```

- 存储图示
  - 以 8 比特为一级，共 4 级
  - 每一级包括 2^8 个条目
  - 每个条目都有 table 到下一级
  - 按字节，查找插入
  - count 是作为当某个条目同时覆盖多个条目时的计数
  - valid 是作为条目的 info 是否有效，是否需要继续匹配的记号



4 实验结果



5 结果分析

- 直接将 forwarding-table.txt 用于测试，会因为最长前缀匹配的原因，导致有 3578 条报错，端口信息不对。如果加入前缀长度，则均正确。自我感觉应该测试 ok 了。

- 手动代码插入 clock() 函数计时，以及计算 malloc 的总空间数结果如上图。
- 其中./search 1 表示普通搜索，2 表示高级搜索。
- 前两条结果是去除了用 forwarding-table.txt 一边读取一边测试的读取时间，后两条消息则没有。
- 总的来看，普通搜索所用空间小，但是速度慢。
- 高级搜索以空间为代价，提升了速度。