C Language Programming: Homework #9

作業說明

Prefix 介紹

- 在HW9作業裡面，需要用Linked-List儲存的資料都是Prefix，而Prefix的格式為a.b.c.d/n，前面a.b.c.d的部分代表的是IP，n則是代表這個Prefix的Length，所以說可以用以下的資料結構在Linked-List裡面來儲存Prefix。

```
struct prefix {
        unsigned ip;
        unsigned char len;
        struct prefix *next;
}
```

- 由於IP裡面，a.b.c.d可以轉換為32 bits的二進位表示，所以IP的部分使用一個unsigned (int) 來儲存，而Legnth的長度不會超過32，因此使用一個unsigned char來儲存。

IP轉換二進位

- 前面一張投影片有提到IP可以用 32 bits的二進位來表示，詳細的轉換方式如以下例子。

- 假設有一個IP是　192.168.0.1，那我們就把這四個數字都轉成二進位依序排列，也就是11000000 10101000 00000000 00000001，總共是4組8-bit的數字，因此可以放進一個unsigned int (32 bits)裡面。

  192.168.0.1 => 11000000 10101000 00000000 00000001

Prefix & IP

- 接下來要介紹Prefix & IP之間的關係，作業說明(j)裡面有提到要寫一個search(…)搜尋某個IP 來看結果是successful 還是 fails，在介紹比對方法之前我們要先了解什麼是Prefix。

- 前面有提到Prefix的格式為a.b.c.d/n，而這個Prefix格式可以再轉成帶有 don't care (*) 的ternary bit pattern，轉換方法如以下例子。

  192.168.0.1/16 => 11000000 10101000 ******** ********

  1.2.3.4/28 => 00000001 00000010 00000011 0000****

- Length = 16的意思就是將左邊16個bit維持原樣，而剩下的bit都改成don't care (*) 。

Prefix & IP (續)

- 了解Prefix 的 ternary bit pattern 之後，再來要介紹如何用IP去得到搜尋結果，詳細流程如以下例子。

- 假設現在程式裡面只存了兩個Prefixes (1) 192.168.0.0/16 (2) 192.168.0.1/32

，而根據上一頁的投影片，我們可以將這兩個Prefixes轉為以下形式

Prefix 1 : 11000000 10101000 ******** ********

Prefix 2 : 11000000 10101000 00000000 00000001

- 然後我們現在有三組IP要讓程式去搜尋找到結果

(1) 192.168.0.1 (2) 192.168.1.2 (3) 1.2.3.4

Prefix & IP (續)

- 我們先把三組IP轉為二進位表示

  IP 1 : 11000000 10101000 00000000 00000001

  IP 2 : 11000000 10101000 00000001 00000010

  IP 3 : 00000001 00000010 00000011 00000100

- 首先搜尋第一組IP，搜尋方法就是將這個IP跟所有的Prefixes一個一個做比較，所以第一步是比較Prefix 1和 IP 1，由於Prefix 1的後16個bit都是don‘t care (*)，因此我們只需要比較Prefix 1和IP 1的前16個 bit，兩者的前16個 bit相同，因此我們說IP 1 match Prefix 1。

  Prefix 1 : 11000000 10101000 ******** ********

  IP 1 : 11000000 10101000 00000000 00000001

Prefix & IP (續)

- 第二步是比較Prefix 2和 IP 1，由於Prefix 2的並不存在don‘t care，因此我們需要比較Prefix 2和IP 1的完整 32 bit，兩者32 bit完全相同，因此我們說IP 1 match Prefix 2。

  Prefix 2 :　11000000 10101000 00000000 00000001

  IP 1 :　11000000 10101000 00000000 00000001

- 只要一個IP有match到任何一個prefix，那麼這個搜尋結果就會是successful，反之則是fails，因此IP 1同時match Prefix 1和 Prefix 2，所以IP 1的搜尋結果是successful。

Prefix & IP (續)

- 再來我們要搜尋第二組IP，第一步一樣是比較Prefix 1和 IP 2，如同前述，兩者的前16 bit相同，所以 IP 2 match Prefix 1。

  Prefix 1 : 11000000 10101000 ******** ********

  IP 2 : 11000000 10101000 00000001 00000010

- 第二步是比較Prefix 2和 IP 2，如同前述，兩者的後2 bit不同，因此我們說IP 2not match Prefix 2。

  Prefix 2 : 11000000 10101000 00000000 00000001

  IP 2 : 11000000 10101000 00000000 00000010

- IP 2 因為 match了Prefix 1，所以IP 2的搜尋結果也是successful。

Prefix & IP (續)

- 最後我們要搜尋第三組IP，第一步一樣是比較Prefix 1和 IP 3，如同前述，兩者的前16 bit不相同，所以 IP 3 not match Prefix 1。

  Prefix 1 : 11000000 10101000 ******** ********

  IP 3 : 00000001 00000010 00000011 00000100

- 第二步是比較Prefix 2和 IP 2，如同前述，兩者的 32 bit不同，因此我們說IP 3 not match Prefix 2。

  Prefix 2 : 11000000 10101000 00000000 00000001

  IP 3 : 00000001 00000010 00000011 00000100

- IP 3 因為 沒有match任何Prefix，所以IP 3的搜尋結果是fails。

Segment

- 介紹完IP如何match Prefix之後，接下來要介紹如何對 Prefixes 做Segment 分為多個groups儲存。

- 在作業說明(d)裡面有提到程式必須讀入一個數字 d，並將 Prefixes 分成 2^d個 groups 儲存，每個 group 都由一條 Linked-List 構成，分組的方法由以下例子來說明。

- 假設總共有四組Prefixes要被程式儲存，分別為 (1) 32.0.0.1/16　　(2) 40.0.0.2/24　　(3) 160.0.0.3/28　　(4) 128.0.0.0/1，程式讀入了數字d = 2，代表我們必須將 Prefixes 分為 4 (2^2) 個groups。

Segment (續)

- 首先我們一樣用前面投影片所紹的方法將 Prefix 轉為 Ternary bit pattern 表示

  Prefix 1 : 00100000 00000000 ******** ********

  Prefix 2 : 00101000 00000000 00000000 ********

  Prefix 3 : 10100000 00000000 00000000 0000****

  Prefix 4 : 1******* ******** ******** ********

- 分組的方式就是看這些Prefixes的前 d 個 bit，根據這 d-bit 的 pattern 來決定這個Prefix會被分到哪個group裡面。在這個例子中，我們使用的d = 2 ，所以說我們必須去看Prefix 1到Prefix 4的前 2 個 bit。

Segment (續)

- 接著我們看Prefix 1，因為Prefix 1的前 2 個 bit 為 00，所以我們將它分進group 0 裡面。

  Prefix 1 : 00100000 00000000 ******** ********

- 再來是Prefix 2，Prefix 2的前 2 個 bit 一樣為 00，所以Prefix 2也是屬於group 0。

  Prefix 2 : 00101000 00000000 00000000 ********

- 接著是Prefix 3，Prefix 3的前 2 個 bit 為 10，所以我們將它分進group 2 (10)裡面。

  Prefix 3 : 10100000 00000000 00000000 0000****

Segment (續)

- 最後是Prefix 4，在這裡我們觀察到Prefix 4 的前 2 個 bit 為 1*，而 1* 並不是一個數字，因此我們不知道該把它放到哪個group裡面。

  Prefix 4 : 1******* ******** ******** ********

- 為了解決上面這種情形所產生的問題，根據作業說明(e) 裡面的敘述，我們可以再用另一個 special group 專門儲存這種類型(那些Prefix Length < d的)的Prefixes。

- 所以說最後我們總共將這些Prefixes們分成了$2^d + 1$個groups，以這個例子來說就是 4 groups + 1 special group = 5 個 groups。

Segment (續)

- (1) 32.0.0.1/16 (2) 40.0.0.2/24 (3) 160.0.0.3/28 (4) 128.0.0.0/1 & d = 2

  Prefix 1 : 00100000 00000000 ******** ********

  Prefix 2 : 00101000 00000000 00000000 ********

  Prefix 3 : 10100000 00000000 00000000 0000****

  Prefix 4 : 1******* ******** ******** ********

  ============= after segment =============

  Group 0 : Prefix 1, Prefix 2

  Group 1 : (Null)

  Group 2 : Prefix 3

  Group 3 : (Null)

  Special Group: Prefix 4

檔案輸出格式

- 在作業說明中有寫到，這支程式你們必須輸出四個檔案，分別為 “length_distribution” , “output_1”, “output_2”, “output_3”。

- 首先在 “length_distribution” 中，你們必須計算在原始的 ”routing_table.txt” 中，每種 prefix length 為 i (i = 0 to 32)的 Prefixes 總共有幾個。因此在”length_distribution”檔案中，必須包含33行的資料，每一行包含一個數字。第一行輸出prefix length = 0 的Prefixes數量，第二行輸出prefix length = 1 的Prefixes數量，依此類推。

- 注意每一行只需要輸出相對應的數字就好，不要輸出任何額外訊息，否則我們會斟酌扣分。

檔案輸出格式(續)

- 接下來介紹剩下三個檔案的輸出格式，在這三個檔案中，你們必須使用 ”trace_file.txt”裡面的那些 IP，去搜尋在不同情況下(詳細請看作業說明)所得到的搜尋結果。

- 假設 “trace_file.txt”裡面有 n 個 IP，那麼在這三個檔案中，必須output n 行的訊息，每一行的輸出為 “successful” 或是 “fails”，第一行輸出 “trace_file.txt”中第一個IP的搜尋結果，第二行輸出第二個IP的搜尋結果，依此類推。

- 注意每一行只需要輸出 “successful” 或是 “fails”，全部都是小寫，並且不包括引號(” ”)的部分，不要輸出任何額外訊息，否則一樣會斟酌扣分。

計算clock cycles

- 在作業說明(k)有提到必須計算search, insert, delete所花費的clock cycles，並且畫成曲線圖放入report中。

- 要注意這張曲線圖要表達的是Frequency，也就是說，假設總共有 a 個 IP 要被 Search，那麼你們就必須將這 a 次的搜尋時間都記錄下來並畫成圖；假設有 b 個 Prefixes 要被 Insert，那麼你們就必須將這 b 次的搜尋時間都記錄下來並畫成圖；假設有 c 個 Prefixes 要被 Delete，那麼你們就必須將這 c 次的搜尋時間都記錄下來並畫成圖。

- 下一張投影片會介紹如何去計算程式的clock cycles。

如何計算clock cycles

- 在HW9提供的檔案裡面有一個clock.c，裡面的程式碼如下

```
#include<stdlib.h>
#include<stdio.h>

unsigned long long int begin,end;

inline unsigned long long int rdtsc()//32-bit
{
        unsigned long long int x;
        asm   volatile ("rdtsc" : "=A" (x));
        return x;
}

inline unsigned long long int rdtsc_64bits()//64-bit
{
        unsigned long long int x;
        unsigned a, d;
        __asm__ volatile("rdtsc" : "=a" (a), "=d" (d));
        return ((unsigned long long)a) | (((unsigned long long)d) << 32);
}
```

如何計算clock cycles (續)

```
int main(){
        begin = rdtsc();
        /*--------------------------     your function--------------------------*/
        end = rdtsc();
        printf("Execute cycles %llu \n",(end-begin));
}
```

- 你們可以使用clock.c裡面所提供的function，參考main()裡面的寫法來得到某段程式碼所花費的clock cycles，也就是上述程式碼裡面begin = rdtsc(); 與 end = rdtsc(); 之間的部分。
- 這次的作業教學就到這裡為止，希望大家都能夠耐心地仔細看完，如果對作業還是有任何問題的歡迎寄信來詢問。