Збірка (C) – для логічних вузлів з двома вхідними стрілками;

Ластівчин хвіст (S) – для логічних вузлів з трьома вхідними стрілками;

Метелик (B) – для логічних вузлів з чотирма вхідними стрілками.

Кожен з вузлів дерева катастроф являє собою перетворювач «входи-вихід» за правилами агрегації «вектор-скаляр», які наведені в табл. 2, де поряд з традиційними для теорії нечітких множин операціями min і max використовуються середнє геометричне і середнє арифметичне.

Ілюстрація методу аналізу катастроф виконана на прикладі дерева відмов, що моделює дорожню аварію на Т-подібному перехресті. Переваги методу полягають у наступному:

1. Відсутня необхідність проведення трудомістких експериментів, пов'язаних з отриманням ймовірностей первинних подій, які впливають на ймовірність відмови системи (або іншої небажаної події). Замість ймовірностей використовуються можливості первинних подій, рівні яких оцінюються експертно або на основі вимірюваних параметрів і відповідних функцій належності.

2. Спостереження за динамікою зміни вхідних параметрів дозволяє переходити до on-line моніторингу рівня можливості відмови системи.

3. Застосування моделей теорії катастроф дозволяє спостерігати нелінійні ефекти, пов'язані з різким збільшенням можливості відмови при незначних змінах вхідних параметрів.

Література

1. Rotshtein A. Selection of Human Working Condition Based on Fuzzy Perfection, Journal of Computer and Systems Sciences International, 2018, 57(6):927–937.

2. Rotshtein A. Risk Analysis: Fuzzy Cognitive Map vs Fault Tree, Journal of Computer and Systems Sciences International, 2019, 2: 200–211.

УДК 004.4:004.7: 378

ОСОБЛИВОСТІ ВИКОРИСТАННЯ ЕЛЕМЕНТІВ СИСТЕМНОГО ПРОГРАМУВАННЯ ПРИ РОБОТІ З IP-АДРЕСАМИ ТА МАСКАМИ ПІДМЕРЕЖІ

Ю. С. Антонов

Під час написання різноманітних програм, призначених для роботи у мережі [1-3], або спеціальних програм калькуляторів, призначених для інженерів комп’ютерних мереж [4], виникає необхідність використання IPv4 адреси та маски підмережі. Подібні програми визначають основні мережеві характеристики такі, як: адреса підмережі, адреси першого та останнього хоста, широкомовна адреса, кількість доступних адрес та хостів. Однак, під час написання таких програм, розробники або здобувачі вищої освіти можуть припускатися певних помилок, які можуть впливати на оптимальну роботу програми.

Мета цієї роботи узагальнити деякі практики та продемонструвати способи оптимального представлення IPv4 адрес та коректної роботи з ними.

У зрозумілому для людині форматі IPv4 адресу прийнято записувати як чотири десяткових цілих числа (від 0 до 255 включно), відокремлених одне від одного крапками, наприклад: *10.2.0.1*, *192.168.0.1* [2, 5]. Аналогічний вигляд має і маска підмережі [2, 5]. Для зберігання чисел у діапазоні від 0 до 255 цілком достатньо одного байту, а отже для зберігання IPv4 адреси повністю необхідно чотири байта (32 біта).

Під час написання програми IPv4 адресу можна представляти різними способами, наприклад:

1. Ціле число довжиною 32 біта;
2. Масив (список) чотирьох цілих чисел довжиною 8 біт;
3. Рядок символів;
4. Масив (список) чотирьох рядків символів.

Найбільш оптимальним та правильним варіантом представлення IPv4 адреси є використання цілого числа довжиною 32 біта (варіант №1). У цьому випадку, знаходження адреси підмережі, першої адреси хоста у підмережі, останньої адреси хоста у підмережі та широкомовної адреси зводиться до використання декількох побітових та арифметичних операцій [6], та не потребує роботи з такими елементами структур даних як масиви або списки. Саме такий варіант використовується під час розробки ядра операційної системи або стандартних бібліотек для більшості мов програмування [3]. У Лістингу 1 наведено код програми на мові C#, який дозволяє по заданій адресі та префіксу мережі визначати основні мережеві характеристики. Аналогічний код можна реалізувати й на інших мовах програмування що підтримують побітові операції (C, C++, Java, PHP, Python, тощо).

Лістинг 1. Приклад програми.

```
using System;
namespace WorkWithIP
{
   class Program {
      public static void ShowIp(string msg,UInt32 ip){
         string[] a = new string[4];
         byte[] ptr=BitConverter.GetBytes(ip);
         Array.Reverse(ptr);
         Console.WriteLine("{0}: {1}\t {2}",msg,string.Join(".", ptr),
                 Convert.ToString(ip,2));
      }
      public static UInt32 ParseIPv4(string buffer){
         UInt32 ip=0;
         string[] Dlist = buffer.Split('.');
         for(int i = 0; i < 4; ++i)
            ip+= UInt32.Parse(Dlist[i]) << (8 * (3 - i));
         return ip;
      }
      public static UInt32 BuildNetMaskByPreffix(UInt32 Preffix){
         UInt32 NetMask=0,key = (UInt32)1 << 31;
         for (UInt32 i = 1; i <= Preffix; ++i){
            NetMask = key+(NetMask>>1);
         }
         return NetMask;
      }
      public static void Main(string[] args){
         UInt32 ResolvedIP=ParseIPv4(Console.ReadLine());
         UInt32 NetMask = 0,Preffix = UInt32.Parse(Console.ReadLine());
         NetMask = BuildNetMaskByPreffix(Preffix);
         ShowIp("Resolved  IP",ResolvedIP);
         ShowIp("Network Mask",NetMask);
         ShowIp("Network   IP",ResolvedIP & NetMask);
         ShowIp("Broadcast IP",ResolvedIP & NetMask + (~NetMask));
         ShowIp("First HostIP",ResolvedIP & NetMask + 1);
         ShowIp("Last Host IP",ResolvedIP & NetMask + (~NetMask) - 1);
      }   }
}
```

Для реалізації варіанту № 2 необхідно замінити тип *UInt32* на *byte[]* та відповідно поміняти тип функцій і адаптувати код під роботу з масивами.

Варіанти № 2–4 є не ефективними, оскільки під час обробки великої кількості IP адрес будуть вимагати більшої кількості системних ресурсів. Саме ці варіанти часто використовують здобувачі вищої освіти при розв'язанні подібних задач у випадках, коли вони не знайомі з принципами роботи побітових операцій та не мають можливості використовувати стандартні бібліотеки.

Таким чином під час вивчення програмування необхідно обов'язково приділяти увагу побітовим операціям, а наведений у роботі лістинг програми може використовуватись у якості реального навчального прикладу. У разі наявності у здобувачів вищої освіти дисципліни «Комп'ютерні мережі» такі приклади будуть додатково посилювати міждисциплінарні зв'язки та покращувати якість коду.

Література

1. Zhang Yongbin, Liang Ronghua, Ma Huiling. Teaching Innovation in Computer Network Course for Undergraduate Students with Packet Tracer. *International Conference on Future Computer Supported Education, August 22–23, 2012,Fraser Place Central. Seoul*. P. 504–510. https://doi.org/10.1016/j.ieri.2012.06.124
2. Cisco Packet Tracer. URL: https://www.netacad.com/ru/courses/packet-tracer (дата звертання: 26.04.2021)
3. The Linux Kernel Archives. URL: https://www.kernel.org/ (дата звертання: 26.04.2021)
4. IP Calculator. URL: http://jodies.de/ipcalc. (дата звертання: 26.04.2021)
5. Олифер В. Г. Олифер Н. А. Компьютерные сети. Принципы технологии протоколы (4-е изд.). СПб.: Питер, 2010, 916 с.
6. Bitwise and shift operators (C# reference). URL: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators. (дата звертання: 21.04.2021)

УДК 004.2

ПРИКЛАДНА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ В СУЧАСНІЙ УКРАЇНСЬКІЙ НАУЦІ

О. О. Баркалов, Р. М. Бабаков

В основі більшості сучасних комп'ютерних систем лежить принцип мікропрограмного керування, запропонований в роботах американського математика С.К. Кліні і деталізований в роботах академіка В.М. Глушкова. Відповідно до цього принципу будь-яка цифрова система представляється композицією двох основних блоків – операційного пристрою та пристрою керування. Операційний пристрій перетворює дані і формує результат під впливом сигналів, що надходять з пристрою керування. Пристрій керування реалізує задану мікропрограму (алгоритм керування) і виконує функцію координації роботи усіх блоків цифрової системи [1].

Сьогодні відомі різні класи цифрових пристроїв керування, серед яких основними є наступні:

1. Мікропрограмний автомат (МПА) або автомат з «жорсткою» логікою [2, 3]. В ньому заданий алгоритм керування імплементується у вигляді комбінаційної схеми, яка будується за системою булевих рівнянь. Це дозволяє МПА реалізовувати багатоспрямовані мікропрограмні переходи за один такт роботи пристрою та визначає його найбільш швидким і продуктивним серед інших класів пристроїв керування. В той же час логічна схема МПА характеризується порівняно високими витратами апаратури, що впливає на кінцеву вартість пристрою керування і цифрової системи в цілому. До переваг МПА слід віднести високу захищеність від несанкціонованого аналізу та модифікації алгоритму керування, що реалізується автоматом. Це досягається знов-таки за рахунок схемної імплементації алгоритму, оскільки аналіз схеми автомата, реалізованого в базисі сучасних програмувальних схем (FPGA, CPLD, ASIC) є надто ускладненим, а під