ПРИМЕРЫ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ АЛГОРИТМОВ ВЫЧИСЛЕНИЯ КОНТРОЛЬНОЙ СУММЫ CRC32

Мыцко Е.А.
Томский политехнический университет, 634050, Россия, г. Томск, пр. Ленина, 30
E-mail: evgenvt@tpu.ru, evgenrus70@mail.ru

Введение

Циклические избыточные коды (CRC) в основном используют для определения ошибок при передаче данных по различным протоколам (Ethernet, ZigBee), а также при архивации данных (WinRaR, WinZip). Существуют разные алгоритмы вычисления CRC и способы их реализации. Некоторые программные реализации алгоритмов вычисления CRC (обратный табличный алгоритм), совместимых с WINRAR, PKZIP, ETHERNET можно встретить в свободном доступе [1]. Также в литературе [2] можно встретить другие реализации вычисления CRC, дающие результат, отличный от алгоритмов, совместимых с WINRAR, PKZIP, ETHERNET (прямой табличный алгоритм) [3]. Существуют и другие реализации CRC, не встречающиеся в свободном доступе, которые будут описаны в данной работе.

Программная реализация табличного алгоритма

Особенность данной реализации заключается в побайтовом вычислении контрольной суммы CRC32 с использованием таблицы предвычисленных значений на основе образующего полинома (104C11DB7h) [3]. Входным параметром в данном случае является имя файла, для которого рассчитывается контрольная сумма, а выходным – значение CRC32 в шестнадцатеричной системе счисления. Далее представлен пример реализации вычисления контрольной суммы CRC32, совместимой с WINRAR, PKZIP, ETHERNET на языке C++. Полный текст исходного кода приведён в [1].

```
unsigned long crc32_table [256] =
{
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba,
……
0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
}
int Get_CRC(char* text)
{
    unsigned long ulCRC = 0xffffffff;
    int len;
    unsigned char* buffer;
    len = strlen(text);
    buffer = (unsigned char*)text;
    while(len--)
     ulCRC = (ulCRC >> 8) ^ crc32_table[(ulCRC &
     0xFF) ^ *buffer++];
     return ulCRC ^ 0xffffffff;  }
```

В приведённом исходном коде crc32_table – таблица предвычисленных значений (всего содержит 256 значений), len – размер сообщения в байтах, buffer – буфер, содержащий сообщение, для которого рассчитывается CRC32, ulCRC – контрольная сумма CRC32.

По исходным кодам программной реализации был восстановлен табличный алгоритм вычисления CRC32 (рис.1) и назван обратным табличным алгоритмом, так как данные считываются с младших байт [4].

Программная реализация матричного алгоритма

Реализация матричного алгоритма вычисления CRC32 заключается в применении операции умножения вектора (выдвинутый байт) на матрицу по модулю 2 [4]. Матрица рассчитывается на основе образующего полинома [3] и имеет меньшую размерность, чем при табличном алгоритме. На основе программной реализации табличного алгоритма был получен исходный код матричного алгоритма вычисления CRC32, пример которого представлен далее.

```
unsigned int crc32_table[8] =
     {
     0x77073096,      0xee0e612c,      0x76dc419,
     0xedb8832,      0x1db71064,      0x3b6e20c8,
     0x76dc4190, 0xedb88320
      };
int Get_CRC(char* text)
{
     ……
     while(len--)
   {
      crcb = (ulCRC & 0xFF) ^ buffer[i++];
     ulCRC = (ulCRC >> 8) ^
     (crcb & 0x1) * crc32_table[0]  ^
     ((crcb & 0x2) >> 1) * crc32_table[1]  ^
      ((crcb & 0x4) >> 2) * crc32_table[2]  ^
     ((crcb & 0x8) >> 3) * crc32_table[3]  ^
      ((crcb & 0x10) >> 4) * crc32_table[4]  ^
     ((crcb & 0x20) >> 5) * crc32_table[5]  ^
      ((crcb & 0x40) >> 6) * crc32_table[6]  ^
     ((crcb & 0x80) >> 7) * crc32_table[7]  ;
   }
    return ulCRC ^ 0xffffffff; }
```

В данном примере в функции Get_CRC был заменён основной цикл вычисления контрольной суммы согласно описанию алгоритма [3,5], а также изменена матрица crc32_table. На рис. 2 представлено схематичное описание матричного алгоритма согласно его программной реализации.

Особенность матричного алгоритма заключается в том, что его можно модифицировать, используя сдвиг по 2 или 4 байта. При этом будет

увеличиваться размер матрицы, используемой при вычислениях. Для двухбайтового сдвига матрица расширится до 64-х байт; для четырёхбайтового сдвига – до 128 байт [4].

Далее приведен пример и описание реализации (рис.3) для четырёхбайтового матричного алгоритма.

```
unsigned int crc32_table[32] =
{
    0xb8bc6765,  0xaa09c88b,  x8f629757,
0xc5b428ef,
    0x5019579f,  0xa032af3e,  0x9b14583d,
0xed59b63b,
    0x1c26a37, 0x384d46e, 0x709a8dc, 0xe1351b8,
    0x1c26a370,  0x384d46e0,  0x709a8dc0,
0xe1351b80,
    0x191b3141,  0x32366282,  0x646cc504,
0xc8d98a08,
    0x4ac21251,  0x958424a2,  0xf0794f05,
0x3b83984b,
    0x77073096,  0xee0e612c,  0x76dc419,
0xedb8832,
    0x1db71064,  0x3b6e20c8,  0x76dc4190,
0xedb88320
};
int Get_CRC(char* text)
{
 ……..
   while(len--)
   {
    crcb = (ulCRC& 0xFFFFFFFF) ^ buffer[i++];

        ulCRC =
        (crcb & 0x1) * crc32_table[0]  ^
        ((crcb & 0x2) >> 1) * crc32_table[1]  ^
            ……
        ((crcb  &  0x80000000)  >>  31)  *
        crc32_table[31] ;
   }
```

Для реализации четырёхбайтового матричного алгоритма необходимо внести соответствующие модификации, такие как увеличение размерности матрицы и обработка сообщения в цикле по 4 байта.

В данном примере для расчёта CRC для сообщения, длина которого не кратна 4-м байтам, применяется дополнительный цикл с побайтовой обработкой.

Таким образом, для программной реализации матричного алгоритма вычисления контрольной суммы CRC32 в функции Get_CRC исходного кода [1] необходимо заменить основной цикл вычисления CRC на цикл, соответствующий описанию матричного алгоритма. Также необходимо изменить матрицу предвычисленных значений и учесть в коде обработку сообщений, не кратных требуемому количеству байт согласно реализуемому матричному алгоритму.

Основным преимуществом матричного алгоритма над табличным является то, что для его реализации требуется значительно меньше памяти, при этом алгоритм является достаточно простым для модификации. Так, для реализации табличного алгоритма требуется 1 Кб памяти на хранение таблицы, в то время, как для матричного алгоритма всего 32 байта, для матричного двухбайтового – 64 байта, для четырёхбайтового – 128 байт.

Заключение

В данной статье приведены примеры программных реализаций алгоритмов вычисления контрольной суммы CRC32. Используя примеры из данной работы и исходный код программы, имеющийся в свободном доступе, можно легко осуществить программную реализацию матричного алгоритма вычисления CRC32. Для рассмотренных программных реализаций восстановлены описания алгоритмов вычисления контрольной суммы. На основе приведённых примеров можно сделать вывод о том, что для реализации матричного алгоритма требуется меньше памяти, чем для табличного, при этом реализацию матричного алгоритма достаточно просто модифицировать, увеличив количество обрабатываемых байт за итерацию в основном цикле исходного кода вычисления CRC.

Литература

1. 32 bit Cyclic Redundancy Check Source Code for C++. // Create Window Website. 2011. URL: http://www.createwindow.com/programming/crc32/ (дата обращения: 01.10.2014).

2. Ross N.W. A Painless Guide to CRC Error Detection Algorithms. // Dr Ross Williams. 1993. URL: http://www.ross.net/crc/download/crc_v3.txt (дата обращения: 01.10.2014).

3. Мыцко Е. А. , Мальчуков А. Н. Исследование программных реализаций алгоритмов вычисления CRC совместных с PKZIP, WINRAR, ETHERNET // Известия Томского политехнического университета. – 2013 – Т. 322 – №. 5. – С. 170-175

4. Мыцко Е. А. , Мальчуков А. Н. Особенности программной реализации вычисления контрольной суммы CRC32 на примере PKZIP, WINZIP, ETHERNET [Электронный ресурс] // Вестник науки Сибири. Серия: Информационные технологии и системы управления. – 2011 – №. 1 – С. 279-282. – URL:http://sjs.tpu.ru/journal/issue/view/2/showToc/sect/4

5. Мыцко Е.А., Мальчуков А.Н. Исследование программных реализаций табличного и матричного алгоритмов вычисления контрольной суммы CRC32 [Электронный ресурс] // Вестник науки Сибири. Серия: Информационные технологии и системы управления. – 2011 – №. 1 – С. 273-278. – URL: http://sjs.tpu.ru/journal/issue/view/2/showToc/sect/4