ASN.1ライトウェイト符号化規則用コンパイラ

6V−6

堀内　浩規　　小花　貞夫　　鈴木　健二
国際電信電話株式会社　研究所

1. はじめに

近年OSIの利用技術が進み、業界ネットワークやLANの構築でOSIが導入され、通信処理の高速化への要求が高まってきた。これに対する解決方法の一つとして、応用層プロトコルを定義するASN.1[1]の基本符号化規則(BER)[2]の改良が注目されており、ISO等で現在検討しているライトウェイト符号化規則(LWER)[3]はその一例である。LWERは計算機の内部表現に基づいた符号化方式により大幅な符号化/復号処理時間の短縮が期待でき[4]、今後はLWERを使用する機会が増えることが予想される。このため、LWERを扱うOSI応用層プログラムを容易に開発可能とするLWERコンパイラが重要となる[5]。本稿ではLWERコンパイラの実装概要を報告する。

2. LWERの概要

LWERは、ワードを符号化の単位とした符号化規則である。特に、OctetString等の可変長の値を持つ型は、値の長さ/オフセット(オフセットと値との相対位置を示す)/値の3組により符号化を行い、可変長の値は符号化データの後部にまとめて配置することでプログラミング言語との対応を容易としている。また、転送構文はワード長(16/32/64ビット)とワードを構成するバイトの順序(Big Endian/ Little Endian)の組合せにより、6種が定義されている。

3. LWERコンパイラの実装

3.1 設計方針

LWERコンパイラを、以下の設計方針により、SUN(Unix)およびVAX(VMS)上に実装した。

① ASN.1による抽象構文の定義からC言語の型定義および符号化/復号関数を自動生成する。

② 生成する符号化/復号関数は、自システムと異なるワード長やバイト順序を持つ計算機との通信も可能とするため、6種類全ての転送構文に対応する。

③ 生成するC言語の型定義は、LWERコンパイラの動作する計算機のワード長とバイト順序が同一の転送構文の符号化に対応するものとする。

3.2 応用層プロトコルプログラムの開発におけるLWERコンパイラの位置づけ

図1に示すように、①LWERコンパイラは対象となる応用層プロトコルの抽象構文の入力に対して、C言語の型定義と符号化/復号関数を生成する。②生成されたC言語の型定義と符号化/復号関数を利用して、状態遷移等のユーザプログラムを作成する。③ユーザプログラムとともに、符号化/復号関数、共通関数をコンパイル/リンクし、応用層プロトコルプログラムを得る。ここで、共通関数は、生成した符号化/復号関数で共通的に使用されるバイト順序変換、メモリ管理等の関数である。

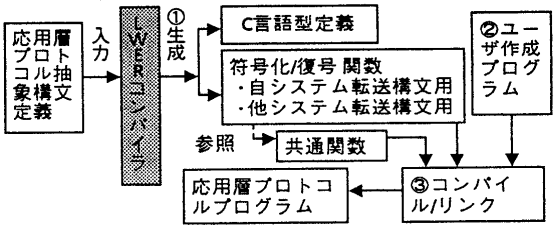

図1 LWERコンパイラの位置づけ

3.3 C言語型定義の生成

抽象構文中のASN.1の型定義からC言語の型定義の生成では、BOOLEAN/INTEGER等の固定長の型はint、可変長の値を持つBitStringやOctetString等は値の長さと値を格納するポインタをメンバに持つ構造体、SEQUENCEは対応する要素を持つ構造体を生成する。入力抽象構文定義と生成されるC言語の型定義例を図2に示す。

```
Record ::= SEQUENCE {
    number   INTEGER,
    name     Name,
    dateOfHire   IA5String,
    children  ChildInf }
ChildInf ::= SEQUENCE OF Name
Name ::= SET{
    first [1]   IA5String,
    last  [2]   IA5String }
```

(a) 入力抽象構文定義

```
struct field { int      length;
    char     *field; };
struct Name { struct field first;
    struct field last; };
struct ChildInf    {   int num;
    struct Name *child;  };
struct Record {    int   number;
    struct Name   name;
    struct field   dateOfHire;
    struct ChildInf    children; };
```

(b) 生成されるC言語型定義例

図2 抽象構文と対応するC言語の型定義例

3.4 符号化/復号関数の生成

3.4.1 符号化関数の生成

(1) 自システムと同一の転送構文の場合

LWERコンパイラは、表1に示す符号化関数の基本生成規則に従い、抽象構文の型定義毎に符号化関数を生成する。例えば、INTEGER等の固定長の型は、単純に変数を符号化結果格納領域(バッファ)にコピーし、OctetString等の可変長の型は、値の長さはコピーを行い、ポインタ値の代わりにオフセットを書き込み、値のバッファ後部へのコピーを行う。SEQUENCEは、各要素の型に応じて、定義順にバッファに書き込む。図2(a)の抽象構文に対して生成される符号化関数のうち、型Record、ChildInfに対応する符号化関数を図3に示す。

ASN.1 Compiler for Light Weight Encoding Rules

Hiroki HORIUCHI, Sadao OBANA and Kenji SUZUKI
KDD R & D Laboratories

表1 符号化関数の基本生成規則

ASN.1の型	生成する符号化関数の処理
BOOLEAN, INTEGER,REAL ,ENUMERATE	値の設定された変数を関数memcpy()を用い、バッファにコピー。
BitString, OctetString, CharString, ObjectIdent	①値の長さを示すメンバ(int length)は関数memcpy()でコピーを行う。②ポインタ値(char *field)の代わりにオフセットを書き込む。③値はバッファの後部にコピー。
NULL	変数が在る場合にはコピー。
SEQUENCE SET	要素の型に応じ、定義順に符号化する。オプショナルな要素は、ポインタ値0の時はコピー、非0の場合はオフセット値を書き込み、値を後部に書き込む。ポインタを含まない要素が連続する時は、複数まとめてコピーする。
SEQUENCE OF SET OF	①要素数を示すカウンタ(int number)は単純にコピーを行う。②ポインタの代わりにオフセットの値を書き込む。③バッファ後部にカウンタの数だけ、要素の型に従いコピーする。
CHOICE	①選択要素を識別するインデックス(int index)および共用体をコピー。②選択要素にポインタが有る時は、バッファ内のポインタ値をオフセットに書き換え、バッファ後部に値を書き込む。

```
/** 型Recordの符号化関数 (32Bit/BigEndian) **/
mERecord32No(Inp, str1, str2)
    struct Record  *Inp;    /* 入力変数 */
    unsigned char **str1;  /* 符号化結果の格納位置*/
    unsigned char **str2;  /* 符号化後部の位置*/
{   int  i;   int  leng;    int  size; unsigned char *ptr;
    int        cnt; unsigned char *str3;
    memcpy(*str1,&(Inp->number),4); /*number符号化*/
    (*str1) += 4;
    if(mEName32No(&(Inp->name), str1, str2) == FALSE)
        return FALSE;  /* nameの符号化 */
    if(mFieldTypeEncode32No(&(Inp->dateOfHire), str1, str2, FALSE)
== FALSE)  /* dateOfHireの符号化 **/
        return FALSE;
    if(mEChildInf32No(&(Inp->children), str1, str2) == FALSE)
return FALSE;  /* childrenの符号化 */
                return TRUE;
}
/* 型ChildInfの符号化関数 (32Bit/BigEndian) */
mEChildInf32No(Inp, str1, str2)
struct ChildInf *Inp;    /* 入力変数 */
unsigned char **str1;  /* 符号化結果の格納位置*/
unsigned char **str2;  /* 符号化後部の位置*/
{   int       i;  int        leng;   int        size;
    unsigned char *ptr;   int     cnt; unsigned char *str3;
    OneEncode32No((&(Inp->number)), (*str1));
    if(Inp->number)        /* カウンタの符号化 */
        leng = IsDiffLength((*str1), (*str2), 4);
    else       leng = 0;
    OneEncode32No((&leng), (*str1));
    str3  = *str2;
    *str2 += (sizeof(struct Name) * Inp->number);
    ptr = (unsigned char *)Inp->name;
    for(i=size=0; i < Inp->number; i++,size += sizeof(struct
Name)){
        if(mEName32No(&ptr[size], &str3, str2) == FALSE)
            return FALSE; }   /* Nameの符号化 */
            return TRUE;
}
```

図3 生成された符号化関数例

(2) 自システムと異なる転送構文の場合

表1に示す基本生成規則に加えて、以下の規則を追加する。

①ワード長が異なる転送構文の場合

int型を持つ要素に対しては、自システムのワード長より小さい転送構文の時は、値があふれないことを検査した後必要なバイトのみを、また、ワード長が大きい転送構文の時は、必要なバイトを付加してバッファへ書き込む。OctetString等の値に対しては、アライメントをワード長に合わせる。

②バイト順序が異なる転送構文の場合

int型を持つ要素に対して、バッファ内に書き込む際にバイト順序変換を行う。

3.4.2 復号関数の生成

復号関数の生成は、基本的には3.4.1節で示した符号化関数の生成規則の逆の規則を適用する。つまり、変数からバッファへのコピーをバッファから変数へのコピーに変更する。また、オフセットを使用する場合には、オフセットの値をポインタとして変更してコピーする。但し、OctetString等の可変長の型については、値の長さやポインタのコピーは行うが、値のコピーは行わない。

4. 評価

4.1 符号化/復号処理機構におけるBERとの差異

従来のBERを対象とするコンパイラ[6]の符号化/復号は、符号化長の計算、値の符号化形式への変換を行った後、識別子、符号化長、値をバッファに書きこむことが必要であるが、本コンパイラの符号化/復号は、基本的には、値の設定された変数をバッファに書き込むことで実現できる。

4.2 符号化/復号処理時間の評価

固定長の要素のみのデータ構造の場合(図4(a))、可変長の要素を含むデータ構造の場合(図4(b))について、自システムと同じワード長とバイト順序を持つ転送構文を使用した際の符号化/復号処理時間を、要素数を変化させて測定した。その結果、BERの場

```
ClassA :: = SEQUENCE {
    elm1   INTEGER,
    elm2   INTEGER,
           . . . .
    elmn   INTEGER }
```

(a) 固定長の要素のみの場合

```
Group :: = SEQUENCE OF Person
Person :: = SEQUENCE {
    number      INTEGER,
    name   IA5String }
```

(b) 可変長の要素を含む場合

図4 評価に使用した抽象構文

合[6]と比較して、固定長の要素のみの場合は約20倍以上高速となり、可変長の要素を含む場合は、符号化/復号処理時間はそれぞれ約3.5倍/約6.0倍高速となった。また、実際の応用層プロトコルの例として、OSI管理のCMIPのm-Get操作要求(相対識別名を5個、同期モードおよびスコープ指定有り)の場合は約4倍高速となった。

5. おわりに

本稿では、実装を行ったLWERコンパイラの概要を報告した。本コンパイラは、これまでのBER用コンパイラと符号化/復号処理機構が異なるとともに、高速な符号化/復号が可能なことを示した。今後、本コンパイラは次世代LAN等の高速通信環境に対応するOSI通信ソフトウェアを構築する上で、有効なツールとなると思われる。最後に日頃御指導頂くKDD研究所小野所長、浦野次長に感謝する。

参考文献

[1]: ISO/IEC 8824 "ASN.1", 1989.

[2]: ISO/IEC 8825 "ASN.1 Baasic Encoding Rules", 1989.

[3]: ISO/IEC JTC1/SC21 N6131, "Working Draft for Light Weight Encoding Rules", July 1991.

[4]: 堀内,小花,鈴木, "ASN.1ライトウェイト符号化規則に関する一考察", 情処第43回全大,4T-03,Oct. 1991.

[5]: 堀内,小花,鈴木, "OSI応用層プロトコル用ASN.1ライトウェイト符号化規則のための符号化/復号処理系の実装と評価", 情処研究報告, DPS-55-2, May 1992.

[6]: 長谷川, 野村, 堀内 "ASN.1支援ツールの開発-コンパイラおよびエディタ-", 情処研究報告,DPS-39-4, Sept.1988.