コンパイラを用いたパッチデータ作成方法

3N-10

遠城秀和　井村佳弘　境孝之

NTT データ通信(株)

1 はじめに

現在、プログラムの生産性や保守性の向上を目的として高級言語が活用されている。開発時は、高級言語で作成したプログラムの修正はソースコードレベルで行ない、効率の良い開発を進めている。しかし、完成品として利用者に提供したプログラムのバグを修正する場合には、プログラムの走行環境が開発環境と異なり開発に用いた高級言語のコンパイラが無いという問題が生じることがある。

それに比べ差分情報でロードモジュールを直接書換える方法(通常パッチと呼ばれている)は、ロードモジュールファイルをデータとして直接書換えるだけなので走行環境に置かれたプログラムの修正に適している。しかし、パッチデータの作成は高級言語で書かれたプログラムを対象としていないため、アセンブラレベルでのデバッグが必要となり保守効率を低下させている。

本論文では、生産性とともに保守性の向上を目的として高級言語を用いてソースコードレベルで修正情報を記述し、コンパイラを用いてロードモジュールへのパッチデータを作成する方法について述べる。さらに、既存のコンパイラを使ったパッチデータ作成の実験結果について報告する。

2 走行環境での修正時の問題点

ロードモジュールで提供したプログラムを修正する場合、開発環境で修正する場合との根本的違いはソースコードが無いことである。さらに、開発環境で使用したものと同一の言語系(コンパイラ、アセンブラ、リンカ)が無い場合もある。

言語系を用意できる場合でも、使用環境の設定が開発環境と異なる時は修正が一律にできないため手順が複雑化し保守性が悪くなる。

しかも、ソースコードを修正する方法では修正後ロードモジュールの作成を行う手順が必要となり、修正作業全体が複雑になるとともに修正ミスを起こしやすくなる。

3 パッチデータ作成方法

3.1 パッチデータ作成方針

提案する方法では、ソースコードで修正を行うことを前提としているので、実行文単位に修正を行うこととした。また、従来のパッチデータ作成手順では削除修正、追加修正、変更修正のそれぞれで方法が異なっていたが、簡単化のため全ての修正は不要になった部分を無効とし、新たにソースコードを追加することとした。

3.2 パッチデータ作成基本手順

コンパイラを用いてパッチデータを作成する方法としては、(1)修正前と修正後のロードモジュールファイル差分を作成する方法、(2)修正前と修正後のソースコード差分を抽出し差分情報をコンパイラを使ってパッチデータ化する方法の2通りが考えられる。

ロードモジュールファイル差分を作成する方法は、単純に新旧のロードモジュールファイルの差分作成でよく簡単に作成できる。しかし、実行文の追加を行うとロードモジュールの途中からズレが生じそれ以降全てが異ってしまう。そのため大量な差分が生成されパッチデータを少量に押えられず、実用的ではない。

コンパイラを用いてソースコード差分からパッチデータを作成する方法のの基本的手順は以下にようになる。

1. まず高級言語で記述された修正後のソースコードから修正前ソースコードとの差分を抽出する。
2. 次に、抽出された差分情報をコンパイルできる形式に修正し、コンパイラを使ってオブジェクトファイルに変換する。
3. さらに、生成されたオブジェクトファイルを逆アセンブルし、逆アセンブルしたリストからパッチデータを作成する。

この方法では、差分情報だけのパッチデータが作成できるため不要なパッチデータが生成されない利点がある。

そこで、(2)の方法でパッチデータを作成することとした。

4 C 言語上の実現例

UNIX[1]に代表される C 言語のコンパイラは、まず高級言語で書かれたソースコードからアセンブラソースに変換する。その後、コンパイラはアセンブラを起動してアセンブラソースからオブジェクトモジュールを作成し、さらにリンカを起動してロードモジュールを作成している。

コンパイラが生成するオブジェクトファイルを逆アセンブルしパッチデータを作成しようとすると、アドレスに関する以下の問題を解決する必要がある。

Patch Data Generation with a Compiler
Hidekazu ENJO, Yoshihiro IMURA, Takayuki SAKAI
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION

[1]UNIX は ATT のベル研究所が開発した OS です

- スタック上に取られる一時変数のアドレス確定
- 外部参照変数のアドレス確定
- 追加する実行コード部分およびデータ部分の先頭アドレス確定

4.1 スタック上に取られる一時変数のアドレス確定

スタック上に取られる一時変数のアドレスは、コンパイル時に決定され、フレームポインタからの相対アドレスとして参照される。同様に、関数の引き数もフレームポインタからの相対アドレスで参照される。割り付けは、一時変数や引き数とも記述した順序にしたがって行われる。

そこで、修正をする部分を含む関数と一時変数や引き数の記述順序および名前を同一にすれば、追加実行文中で用いる名前を修正される関数と同じ割り付けにできる。具体的には、一時変数や引き数の記述部分だけを抽出したソースコード (図1、図2) を作っておき、後から差分情報を追加する。

```
int     a;

main()
{
  func("abc");
}

func(s)
char    *s;
{
int     b;

  a = 10;
  b = 100;
  printf("%s %d %d\n", s, a, b);
  printf("---\n");
  printf("%s %d %d\n", s, a, b);
}
```

図1 サンプルプログラム

```
int     a;

func(s)
char    *s;
{
int     b;

}
```

図2 一時変数や引数を抽出したソースコード

4.2 外部参照変数のアドレス確定

C言語コンパイラは外部参照をシンボルとして扱い、シンボルに対して属性情報を付加するアセンブルソースを生成している。属性情報として値が確定しているシンボルはアセンブラによりアドレス確定されるが、外部参照シンボルは、値が未定義のため属性情報がリンカの使用形式に変換されるのみで、実際のアドレスとの対応づけはリンカによって行われる。しかし、ロードモジュール中ではすでに外部参照とアドレスは対応づけられているので、外部参照シンボルとアドレスの対応情報が抽出可能である。

そこで、ロードモジュールから外部参照シンボルとアドレスの対応情報を外部参照シンボルの値を定義するアセンブラソースの形式で抽出し、コンパイラが修正情報から生成したアセンブラソースと一緒にアセンブルする。(図3) これにより、アセンブル時にアドレス確定ができるようになった。

```
asm("set main           ,     68");
asm("set environ        ,  65540");
asm("set splimit%       ,  65536");
asm("set a              ,  67404");
asm("set func           ,     88");
asm("set printf         ,    204");
```

図3 シンボルとアドレスの対応情報

4.3 追加する実行コード部分およびデータ部分の先頭アドレス確定

C言語コンパイラは、実行コードやデータを置くアドレスの決定をリンカが処理をするまで先送りにする。したがって、コンパイラが生成するオブジェクトファイルを単純に逆アセンブルしても実行コードやデータを置くアドレスが未確定の状態になってしまう。

しかし、追加する実行コードやデータは前もって用意しておいたエリアの空いている部分に置く必要があり、逆アセンブルする前にアドレスの確定化が必要である。

そこで、与えられたアドレスに実行コードやデータを置く機能がリンカ一般の機能としてあることを利用し、リンカへに実行コードやデータを置くアドレスを指示してオブジェクトファイルを変換しておく。

5 パッチデータ作成実験

図1に示すサンプルプログラムに対する図4の修正情報を使ったパッチデータ作成実験を行った。修正前、修正後の実行結果を図5、図6に示す。

```
int     a;

dummy(s)
char    *s;
{
int     b;

  printf("===\n");
  a = 20;
  b = 400;
  s[0] = 'z';
  printf("===\n");
  asm("jmp func+0x44");
}
```

図4 修正情報の例

```
abc 10 100
---
abc 10 100
```

図5 修正前の実行結果

```
abc 10 100
===
===
zbc 20 400
```

図6 修正後の実行結果

6 まとめ

生産性とともに保守性の向上を目的にソースコード差分を基にしたコンパイラ利用によるロードモジュールへのパッチデータ作成方法について述べた。

コンパイラが生成するオブジェクトファイルは、外部変数や実行コードを置くアドレスなどが未確定であるため、確定化が問題となり3つの対処を行った。その結果、ソースコードレベルでの修正情報からコンパイラを使ってパッチデータを自動的に作成する手順を構築できる見通しを得た。