OS/omicron V4 におけるインタプリタ型言語とネイティブコードのリンクの実現

1F−8

山本 康弘、早川 栄一、並木 美太郎、高橋 延匡
東京農工大学　工学部

1. はじめに

プログラムをさまざまな言語を使用して作成することで記述性、信頼性を向上させることが可能である。例えば、ハードウェア操作や内部データ管理部を自由度の高い C で記述し、インタフェース部を Java で記述して OS を構築することも考えられる。

しかし、実際に異なる言語で記述されたモジュール間でリンクを行う場合には多くの制約があり、これを行うことは困難である。

そこで筆者らは、OS/omicron V4（以下 V4）[1] 上にインタプリタ型言語とネイティブコード間のリンクを、特別な手続きなしで行える環境を設計し、Java* とネイティブコード間のリンク機構を実現した。

2. 従来のインタプリタ型言語の問題点

(1) 他言語とのリンクが困難

従来のインタプリタ型言語は、他言語で記述されたモジュールとのリンクを考えてなく、プログラムの再利用が行えないので、この場合対象言語で新規作成する必要がある。

また、Java はネイティブコードとリンクするインタフェースを用意しているが、メソッドの呼出しには、特別な手続きをプログラム中に記述する必要がある。

(2) インタプリタ起動手続きが必要である

プログラムの実行にはインタプリタの起動を必要とする。このため、ネイティブコードからインタプリタ型言語で記述されたプログラムを呼び出すときも、インタプリタ起動のための手続きを明示する必要がある。

3. 設計方針

(1) 特別な手続きなしにリンクを可能にする

開発において、モジュールの再利用を考えたときには、インタプリタ型言語も他の言語とのリンクを行える必要がある。

しかし、Java のようにリンクのための特別な手続きを必要とした場合には、特定のモジュールとしかリンクできなくなってしまう。V4 ではダイナミックリンクにより、実行時にユーザプログラムを構築できるが、リンク先を特定してはこの特徴を活かせない。

このため、特別な手続きなしに他言語とのリンクを可能にする。

(2) インタプリタの起動手続きをなくす

プログラム中でインタプリタの起動を必要とした場合には、(1) と同様にリンク先を特定したモジュールになってしまう。このため、ユーザが明示的にインタプリタの起動を示す必要をなくす。

4. 実行環境の設計

4.1 全体構成

V4 上のインタプリタ型言語の実行環境の全体構成を図 1 に示す。

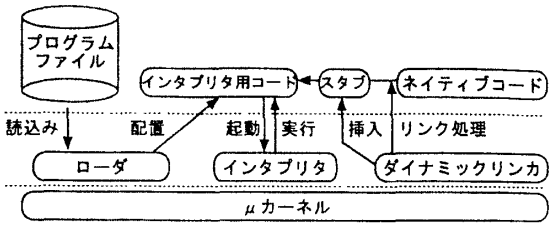

図 1 全体構成

実行環境は次の三つの部分に分けられる

(1) ダイナミックリンカ
(2) インタプリタ
(3) ローダ

4.2 ダイナミックリンカ

異なる言語モジュール間でダイナミックリンクが発生した場合には、そのままリンクすることはできない。そこで、リンク時にリンク先とリンク元モジュールの属性を参照し、図 2 のように自動的にスタブを挿入することで、特別な記述なしにリンクを可能にする。

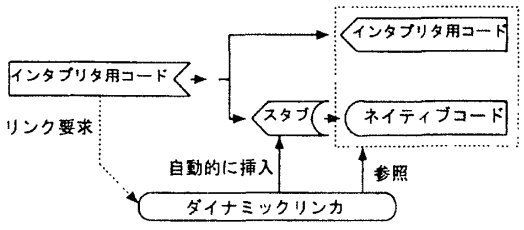

図 2 リンク処理

The Implementation of Linking a Interpreted Program Code with Native Program Code on OS/omicron V4.
Yasuhiro YAMAMOTO, Eiichi HAYAKAWA, Mitarou NAMIKI and Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology

また、各言語ごとにデータの持ち方や引数の評価順序などが異なってくるため、変数や関数情報が格納されているリンクテーブルには、各言語ごとに特有のデータ構造を格納し、型変換やスタック操作などが行えるようにする。

4.3 インタプリタ

起動手続きをユーザが明示する必要をなくすため、インタプリタは対象言語で記述されたプログラムが出現したときに自動的に起動させる。また、関数コールなどは、ネイティブコードで使用しているスタックを使用し、相互に呼び出すことを可能にする。外部モジュールからアクセスできるデータは、リンクテーブルに登録し、テーブルを通してアクセスする方法に統一する。

4.4 ローダ

ネイティブコードとのリンクを行うために、プログラムをコード部、データ部、リンクテーブルに分割し、属性を付けて、メモリ中に格納する。ここでは、プログラムファイルの解析や、必要ならばリンクテーブルの生成を行う。

5. Java 実行環境の実現

設計に沿って、V4 上で Java の実行環境を実現した。Java はビッグエンディアンのデータ構造と、Unicode を採用しているが、現在はローダが V4 の実行環境であるリトルエンディアンと JIS X 0208 に変換する。インタプリタは、テストプログラムが実行できるだけの命令（全体の約 3 分の 1）を実装した。

これらの実現規模を表 1 に示す。また、性能評価としてインタプリタの起動とダイナミックリンク処理の実行時間を表 2 に示す。

表 1 実現規模

内容	記述行数
ローダ部分	3289
インタプリタ部	3156
ダイナミックリンカ部	213

表 2 性能評価

内容	所要時間（μs）
ダイナミックリンク処理	6437
インタプリタの起動	428

ダイナミックリンクにおける Java の処理が占める割合は 5%以下で、ほとんどが通常のリンク処理である。インタプリタの起動は実行不可属性のフォールトを起動キーにしている。これにより、インストラクションポインタが直接コードを指した場合にも、インタプリタを起動できるといった柔軟性を持たせているため、このオーバヘッドは十分許容範囲内であると考える。

6. 考察

リンクのテストは Java と C のプログラム間で、相互に関数を呼び出すことで行った。このとき、C と Java のネーミングルールの違いから次のような問題が発生した。

Java は引数が異なる場合には、メソッドに同一ネームを許しているが、C では関数に同一ネームは許されていないので、V4 上の C 言語処理系である CAT386 [2] はリンクテーブルに引数情報を出力しない。このため、Java でパブリック指定された同一ネームのメソッドが存在したとき、ダイナミックリンカはリンク先を特定できず、最初に遭遇したものとリンクしてしまう。

このように、ある言語で通用しているルールが他の言語とのリンクでは通用しないといったことが存在する。このため、リンクテーブルの生成方法や呼出し規則として、共通のルールを設定する必要がある。

7. おわりに

本稿では、OS/omicron V4 上のインタプリタ型言語とネイティブコードのリンクの実現について述べた。成果としては、インタプリタ型言語である Java と C とのリンクを可能にし、V4 における現在のリンクテーブルの問題を明らかにしたことが挙げられる。今後の課題として、リンクテーブル生成時のルールの作成と、それに沿った言語処理系の変更が考えられる。

謝辞

本研究は、文部省科学研究費補助金（基盤研究（B）（2）課題番号 08458064）により行われた。

参考文献

[1] Hayakawa, et.al : "Basic Disign of SHOSHI Operating system that Supports Handwriting Interface.", 情報処理学会論文誌, Vol.35, No12

[2] 中村浩之, 他 : "80386 用 OS/omicron 開発のための言語 C 処理系の実行環境の設計", 情報処理学会第 44 回全国大会, 2F-10, 1992

[3] Tim Lindholm, Frank Yellin : "The Java™ Virtual Machine Specification"

*Java は米国 Sun Microsystems の商標である