RubyOS におけるマーシャリングを用いたオブジェクトの保存

吉原 陽香[1]　　佐藤 未来子[1]　　並木 美太郎[1]

1. 背景

オブジェクト指向プログラミング言語である Ruby は，すべてをオブジェクトとして表現し，動的型付け，Mix-in，GC などの機能をもっている．またその可読性や記述性の高さから，「動く擬似コード」とも言われている．

本研究では，Ruby にて OS の実装を行うことで，Ruby で OS を記述することの有用性を追及している[1]．これまでに，OS の搭載されていない Intel x86 アーキテクチャ上に Ruby 処理系を移植した．そしていくつかのデバイスドライバを Ruby で実装し，OS の機能を Ruby で実装できることを示した．本稿では，Ruby オブジェクトを 2 次記憶装置へ保存する方式について述べる．

2. システム設計

実装している OS の全体設計を図 1 に示す．C で書かれた Ruby 処理系を OS のないマシン上で動作させ，その上で計算機資源を Ruby のクラスとして仮想化し，OS の機能を実装するという設計となっている．またハードウェアに依存する処理については C にて記述し，任意のタイミングで GC を動作させメモリの回収をするといった，ハードウェアから独立した処理は Ruby にて実装する設計としている．つまり，ハードウェアに関わらず Ruby で記述した資源管理はそのままで実行できる設計となる．

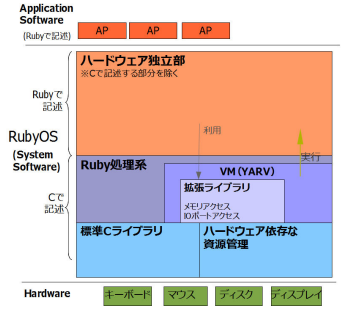

図 1　実装している OS の全体設計

1 東京農工大学
Faculty of Engineering, Tokyo University of Agriculture and Technology.

3. 2 次記憶装置へのオブジェクト保存法

本システムでは，Ruby のオブジェクトをファイル単位ではなく，オブジェクトをマーシャリングして直接 2 次記憶装置に保存する．直接保存することで，メモリと 2 次記憶装置の両方でオブジェクトを単位として扱う．

また，オブジェクトはいつ読み込んでも保存した時点と同じ値をもつものだけを保存する．マシンの電源を切るなどして処理系の状態が変わっても，読み込んだオブジェクトの値が変わらないことを OS としてオブジェクトを保存する条件とする．これに基づいて分類したオブジェクトを表 1 に示す．

表 1 オブジェクトの分類

保存不可	保存時の処理系の状態に依存するもの ・Continuation（継続）・Proc（手続き）， ・Binding（処理系の状態） ・MatchData（正規表現の情報） ・Method・UnboundMethod（メソッド）
保存可能	上記以外すべて ・Integer（数値）・String（文字列）など

3.1 2 次記憶装置上でのオブジェクトの形式

本節では，2 次記憶装置に格納する際のオブジェクトの形式について述べる．Ruby のオブジェクトは処理系内部で構造体もしくは数値として表されており，そのままでは 2 次記憶装置に格納できない．そこで，本方式では Marshal というモジュールを用いてオブジェクトを文字列に変換する．このモジュールはオブジェクトと文字列を相互変換する機能をもっている．本来はオブジェクトをファイルに書き出したり，読み込んだりするための機能だが，本システムでは 2 次記憶装置と直接オブジェクトをやり取りするために用いる．このモジュールを用

いる利点として，オブジェクトの内部形式に関わらず同じ手続きで変換・復元ができることが挙げられる．そのため，本方式では Marshal モジュールを用いることにした．

3.2 読み込んだオブジェクトの参照方法

オブジェクトを利用するためにはその参照を格納した変数を用いる．そのため，保存時にはオブジェクトに対してそのオブジェクトへの参照をもつ変数を対応づける必要がある．これを解決するため，本方式では Ruby の Hash オブジェクト（以下，ハッシュ）を用いる．ハッシュはキーと値を対応づけるオブジェクトであり，その構造は{キー1 => 値 1，キー2 => 値 2，...}のように表現される．

本方式ではこのハッシュ内のキーが変数の名前，値がその変数に参照されるオブジェクトとなるように，すべてのオブジェクトと変数を対応づける．そしてこのハッシュを一つのオブジェクトとして文字列に変換する．この設計だと 2 次記憶装置とメモリの間ではこのハッシュの操作を行うだけとなり，2 次記憶装置へアクセスするオーバーヘッドが減少し速度向上につながると考えられる．

また，その文字列の大きさも 2 次記憶装置に保存し，読みこむべき大きさがわかるようにする．Marshal モジュールから変換される文字列は先頭が数値以外の文字となるので，文字列を混合せずに文字列の大きさのみを取得できると考えた．

3.3 オブジェクト保存の手順

2 次記憶装置へのオブジェクトの書き込み，読み込みの流れをそれぞれ図 2，図 3 に示す．まず保存する際には，処理系内で作成されるローカル変数名のリストから OS となる Ruby スクリプト内で変数名を一つずつ取り出し，それから参照されるオブジェクトをキーと値として対応づけたハッシュを作成する．これを変換した文字列にその文字列のサイズを結合したものを 2 次記憶装置の先頭セクタから順番に書き込んでいく．そのときのセクタと書き込むデータとの対応を図 4 に示す．また，オブジェクトを読み込む際にはまず先頭セクタに格納された文字列の大きさを読み込む．そしてその分だけ 2 次記憶装置の先頭セクタから読み込んだデータから文字列を取得し，それを復元してハッシュを得る．そのハッシュのキーを変数名とした変数に値であるオブジェクトへの参照をもたせ，オブジェクトを利用可能にする．今回は試作として単純な実装にしているが，今後はマーシャリングされたデータを複数格納できるようにする．

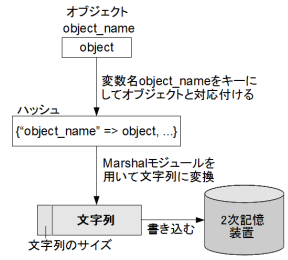

図 2 オブジェクト保存の流れ

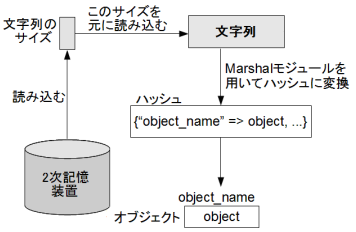

図 3 オブジェクト読み込みの流れ

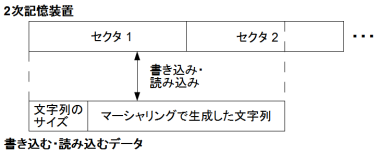

図 4　2 次記憶装置のセクタとデータの対応

4. 実装

2 次記憶装置としてハードディスクを対象としたデバイスドライバを作成し，3.3 節の図 2，図 3 の処理を実装し，それぞれ実行することで正しく動作していることが確認できた．どのようにオブジェクトが変換・保存され，読み込まれるかをポスターと共に展示する予定である．

5. まとめと今後の予定

本稿では，RubyOS におけるオブジェクトの保存方式について述べた．今後の予定としては，Multi-VM に対応することなどが挙げられる．Multi-VM に対応した Ruby 処理系では VM インスタンスを複数生成できるため，各 VM それぞれのオブジェクトを保存するような設計が必要となる．

参 考 文 献

[1] 吉原陽香，笹田耕一，佐藤未来子，並木美太郎："Ruby による OS の構築を目指して"，第 52 回プログラミングシンポジウム予稿集，pp.133-144，Jan，2011.

[2] 川合秀実，「30 日でできる！ OS 自作入門」，毎日コミュニケーションズ，2006.