

# Rakuのサーバーを使った実行

165727F 氏名 福田光希 指導教員：河野 真治

## Abstract

We use VR technologies in computer music game which competes with the musical performance of instruments. HMD (Head Mount Display) and LeapMotion (Hands Tracking Device) are used with UnrealEngine4 (UE4). In this game, bare hands work well in HMD reality without annoying controller in hands. We have solved interferences between hands and HMD visual using transparent display of hands. This game is implemented on UE4 BluePrints which is a visual programming language with wires and boxes. The effectiveness of combination of VR and LeapMotion in music game is confirmed.

## 1 スクリプト言語の高速実行

現在多くのスクリプト言語はインタプリタ型言語であり、実行時にインタプリタの立ち上げ、モジュールを読み込み、スクリプトの解釈、スクリプトの実行といったような処理を担っている。これらの処理の中にはOS上で事前に行うことで、より起動時間、及び処理時間の短縮が予想される。

現在開発の進んでいる言語にスクリプト言語 Raku がある。Raku は任意の VM が選択できるようになっており、主に利用されている VM に C で書かれた MoarVM が存在する。MoarVM は JIT コンパイルなどをサポートしているが、全体的な起動時間及び処理速度が Perl5 や Python , Ruby などの他のスクリプト言語と比較し非常に低速である。

本研究では、スクリプト言語 Raku の起動時間及び、処理速度の改善を図り、研究をするにあたり得られた、OS 上で script 言語を実行する場合の利点と欠点について述べ、今後の展望について記載する。

また、その手法として同一ホスト内で終了せずに実行を続けるサーバープロセスを立ち上げ、このサーバープロセス上で立ち上げておいたコンパイラに実行するファイル名を転送し、サーバー上でコンパイルを行う手法を提案する。著者らは、この提案手法に沿って『Abyss サーバー』を実装している。

## 2 Raku

Raku は元は Perl5 の後継言語の Perl6 として開発されていたが、現在は名称が変更され Raku となっている。Raku の現在の主流な実装は Rakudo である。Rakudo は MoarVM、と NQP と呼ばれる Raku のサブセット、NQP と Raku 自身で記述された Raku という構成である。MoarVM は NQP と Byte Code を解釈する。

NQP とは Not Quite Perl の略で Raku のサブセットである。その為基本的な文法などは Raku に準拠しているが、変数を束縛で宣言するなどの違いが見られる。

この NQP で記述された Raku の事を Rakudo と呼ぶ。Rakudo は MoarVM の他に JVM, Javascript を動作環境と

して選択可能である。

Raku の起動は、MoarVM を起動、nqp をロード、Rakudo をロードもしくはコンパイルし、その後 JIT しながら実行する。

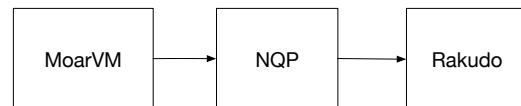


図 1: Raku の構成

## 3 MoarVM

MoarVM は Raku に特化した VM である。C 言語で実装されている。JIT コンパイルなどが現在導入されているが、起動時間などが低速である問題がある。MoarVM 独自の ByteCode があり、NQP からこれを出力する機能などが存在している。

## 4 NQP

NQP とは Raku のサブセットである。その為基本的な文法などは Raku に準拠しているが、変数を束縛で宣言するなどの違いが見られる。

NQP は最終的には NQP 自身でブートストラップする言語であるが、ビルドの最初にはすでに書かれた MoarVM-ByteCode を必要とする。この MoarVMByteCode の状態を Stage0 と言う。Raku の一部は NQP を拡張したもので書かれている為、Rakudo を動作させる為には MoarVM などの VM、VM に対応させる様にビルドした NQP がそれぞれ必要となる。NQP は与えられた Stage0 を使い Stage1 をビルドし、その Stage1 を利用し Stage2 をビルドする事で生成できる。

## 5 なぜ Raku は遅いのか

通常 Ruby のようなスクリプト言語ではまず rubyVM が起動し、その後スクリプトを Byte code に変換して実行という手順を踏む。Rakudo はインタプリタの起動時間及び、全体的な処理時間が他のスクリプト言語と比較して非常に低速である。これは Rakudo 自体が Raku で書かれているため、MoarVM を起動し、Rakudo と NQP をコンパイルし、その後スクリプトの Byte code 変換というような手順で進むためである。また Raku は実行時の情報が必要であり、メソッドを実行する際に invoke が走ることも遅い原因である。

## 6 Abyss サーバー

ここでは Abyss サーバーについて説明する。Abyss サーバーは Raku で書かれている。クライアント側から投げられた Raku を実行するためのサーバーである。図 1 は Abyss サーバーを用いたスクリプト言語実行手順である。Abyss サーバーはユーザーが Raku を実行する際、クライアント側から転送されてきたファイルを事前に起動してあるサーバー側が処理し、その実行結果を返す構造となっている。

この手法を用いることで、サーバー上で事前に起動した Rakudo を再利用し、投げられた Raku スクリプトの実行を行うため Rakudo の起動時間を短縮できると推測できる。

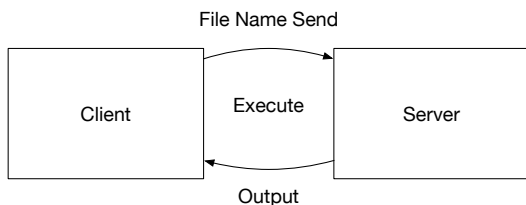


図 2: Abyss サーバーを用いたスクリプト言語実行手順

Code1 は Abyss サーバーのソースコードである。Abyss サーバーは起動すると、まず自身にファイルを転送するためのソケットを生成し、その後ファイルを受け取るための待機ループに入る。

ファイルパスを受け取るとファイルパスをバッファに変換し読み込む、その後読み込んだバッファを文字列にデコードし、ファイルパスの文字列を読み込み、ファイルの中身を式として評価する EVALFILE を用いて、プログラムを実行する。Code1 の 2 行目にある MONKEY - SEE - NO - EVAL は Raku 上で EVALFILE を使用可能にする pragma である。

ソースコード 1: Abyss サーバーの source code

```
use v6.c;
```

```
unit class Abyss::Server:ver<0.0.1>:auth<cpan:
use MONKEY-SEE-NO-EVAL;
use IO::Socket::Unix;
use NativeCall;

sub close(int32) returns int32 is native { ...
sub dup(int32 $old) returns int32 is native {
sub dup2(int32 $new, int32 $old) returns int32

method readeval
{
    my $listen = IO::Socket::Unix.new(
:listen,
:local
:local

my $backup = dup(1);
say DateTime.now;

loop
{
    my $conn = $listen.accept;
    my $sock_msg;
    my $buf = $conn.recv();
    $sock_msg = $buf;
    close(1);
    dup2($conn.native-descriptor(), 1);
    EVALFILE $sock_msg;
    dup2($backup, 1);
    close($backup);
    $conn.close;
}

$listen.close;
}
```

ソースコード 2: クライアント側の source code

```
use IO::Socket::Unix;

my $conn = IO::Socket::INET.new( :host<localhost>
:port(3333) )

$conn.print: 'Absolute file path';

say $conn.lines;

$conn.close;
```

## 7 まとめ

Perl6 スクリプトを投げて実行するサーバーの実装, および「自分でプロセス立ち上げて Perl6 実行する手法」と「既にあるサーバーに投げて Perl6 スクリプトを実行する手法」の差を測るために時間の計測を行った.

## 8 今後の課題

また今回例題として用いたものはスクリプト言語 Raku であったが, その他のスクリプト言語にも応用が利くかどうか検討する必要がある

今回用いた Raku の EVALFILE 自体にクライアント側に出力を返す実装追加することも今後の課題に挙げられる.

## 参考文献

- [1] Andrew Shitov. Perl6 Deep Dive
- [2] 清水隆博, 河野真治. CbC を用いた Perl6 処理系. 琉球大学工学部情報工学科平成 30 年度学位論文 (学士), 2018.
- [3] Perl6 Documentation  
<https://docs.perl6.org> (2019/10/22 アクセス)