ROVM Tutorial Document

Weongyo Jeong
Translator: Vincent Lee

2006.2.19

Australia Melbourne
email : weongyo@gmail.org
Translator email : handrake@gmail.com

Abstract

This document is for a beginner of ROVM and describes how you can use ROVM in your system and what you can with ROVM.

Contents

1 Introduction **1**

2 Overview **2**

3 How to calculate 2+4 using ROVM **2**
3.1 Executing Client . . . 2
3.2 Getting a ticket . . . 2
3.3 Typing opcode . . . 3
3.4 Sending Opcode . . . 3
3.5 Checking Stack Status . . . 4
3.6 Dropping Ticket . . . 4

4 What's a Ticket, anyway? **4**

5 How to deal with Class **6**
5.1 Getting Object Reference . . . 6
5.2 How to use String Reference . . . 7
5.3 Calling Method . . . 8

1 Introduction

Welcome to the world of ROVM.

ROVM stands for 'Remote Object Virtual Machine', which basically means programming with objects that are executed on a remote (not local) machine.

This document is written under the assumption that a reader already has set up and configured at least ROVM Server and ROVM Client. If you don't have ROVM Server up and running on your local machine, the bottom line is you have the ip address (or domain name, not much difference) and port number of some server that does.

At the time of writing (Feb 19, 2006), this document consists only of few pages and 20 opcodes. However, as we progress, more opcodes will be added, and/so it will be possible for a user to be able to write a new function or opcode using those already implemented.

2 Overview

There exists two ways of communication in ROVM. One is a commucation between ROVM Server ↔ ROVM Client which will be described later in this document, and the other is between ROVM Server ↔ ROVM Server which has not yet been put into the code.

In case of ROVM Server ↔ ROVM Client, user will write a program and send it through ROVM Client to Server, and the server will take it and execute.

If it's between ROVM Server ↔ ROVM Server, a running program (either already installed on system, or ROVM opcode file formatted program sent to the server) in one server will request to the other and receive it dynamically if that requested object (can be Class Object, etc...) is in the other server.

3 How to calculate 2+4 using ROVM

As a first example, we are going to do a simple arithmatic: '2 + 4', since all we can do at this point (Feb 1, 2006) with ROVM Server and ROVM client is `<char number> + <char number>`.

3.1 Executing Client

Now let's execute ROVM Client. This can be done easily like the following.

```
C:\> python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>>
```

3.2 Getting a ticket

In order to use a resource on ROVM server, you have to get what's called a ticket. This 'ticket' can be thought of as 'session' of a web server. Thus, it has all the information that you need to execute your program on ROVM Server and also the status of the program.

If it is considered by 'Garbage Collector' as an unused ticket or you explicitly specify its termination by invoking '**REQEND**', this ticket will vanish from ROVM server.

Ticketing can be done using '**REQ**' command.

```
>>> req e://192.168.58.129:8888
Opening Socket 192.168.58.129:8888
Ticket '3K$\xe9\xb9\xfd\xc7\xb1\x9dqo\x07\xd3E\xae__\x16C\xbf'
Closing Socket.
>>>
```

Always remember that you use REQ command as e://<hostname>:<port>

3.3 Typing opcode

If you were able to get a ticket without any error, now you have a permission or privilege to send commands to a server for execution. Let's start out doing something simple.

First of all, you have to be in 'opcode mode' in order to type opcodes.

```
>>> opcode
Using ticket '3K$\xe9\xb9\xfd\xc7\xb1\x9dqo\x07\xd3E\xae__\x16C\xbf'
... cpush 2
... cpush 4
... iadd
...
>>>
```

'cpush' pushes char typed number into a stack. 'iadd' pops 2 lots from the stack, add them, and push the result back to the stack.

To exit from 'opcode mode', just type <ENTER>.

However, the opcodes you have written are not delivered to the server unless you expicitly say '**SEND**'. Before doing '**SEND**', let's type in '**PRINTOPCODE**' first. You can see that the opcodes you wrote have been translated into binary before actually proceeding to ROVM server.

```
>>> printopcode
#0 '\x10\x02'
#1 '\x10\x04'
#2 ''
>>>
```

3.4 Sending Opcode

Now, let's start VM by invoking '**SEND**'.

```
>>> send
Opening Socket 192.168.58.129:8888
Return VOID
Closing Socket.
>>>
```

VOID type is returned as 'return value'. This is because there is no opcode in our code that alters it.

3.5 Checking Stack Status

Then, let's check how things are in stack. You can conveniently do so by using '**STACK**' command.

```
>>> stack
Opening Socket 192.168.58.129:8888
[DEBUG] [Mon Jan 30 10:42:36 2006] proc_rc.c(170): #0 INT 0x6
Closing Socket.
>>>
```

3.6 Dropping Ticket

Now if you are done using the ticket and have no plan of using it later, it's a good idea that you drop it. This can be done by 'REQEND' command. If the command is successfully carried out, we say that the ticket is completely terminated in ROVM Server, and server cannot take any further commands for execution through that ticket.

```
>>> reqend
Opening Socket 192.168.58.129:8888
<- OK
Closing Socket.
>>>
```

4 What's a Ticket, anyway?

In this section, I'll explain in detail what a Ticket is, basically going through what it does, how it does it and possibly why it does it. Before I have mentioned that Ticket can be thought of as 'session' of a web server.There are two big ways to define Ticket.

- From User point of view
- From Developer point of view inside ROVM server

If you look from user point of view, you can only see an unique ID of the ticket. Every operation, reading, executing, writing, is done using this unique ID, which I called as 'Ticket ID'.

For example, when you go see a movie, you have to get a ticket (is there some other way I don't know about?) The ticket you bought gives you a privilege to watch the movie you want. If you do not have the right ticket, you are NOT allowed to see it. However, it only privileges you, it's not like you can see the entire movie there.

As you can see that entire movie in a theater, you can find all information associated with 'Ticket ID' in 'Ticket structure'. Everything you do with that 'Ticket ID' will be stored in the 'Ticket Structure' and you will be using it until you send 'REQEND' command to server to drop it.

To summarize, 'Ticket ID' and 'Ticket Structure' work together and 'Ticket ID' is needed to find where associated 'Ticket Structure' is located.

Let's see how 'Ticket Structure' operates under the surface. At first, execute ROVM Server (from now on, I'll assume ROVM Server is open in 192.168.58.129 on 8888 port.) and ROVM Client, and do the 'Ticketing'

```
C:\projects\rovmclient>python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>> req e://192.168.58.129:8888
Opening Socket 192.168.58.129:8888
Ticket '\xfc1\x0e\x0cA\xd2\xbaAy(d\xcbL\xf9\xa8X4\xce\x8e\x9a'
Closing Socket.
>>>
```

Above you can see you got the Ticket '\xfc1\x0e\x0cA\xd2\xbaAy(d\xcbL\xf9\xa8X4\xce\x8e\x9a'. It looks a little odd, but that's because it's in binary format. Now, let's execute some opcodes. You can execute each opcode one at a time. (Not possible if it's in loop)

```
>>> opcode
Using ticket '\xfc1\x0e\x0cA\xd2\xbaAy(d\xcbL\xf9\xa8X4\xce\x8e\x9a'
... cpush 65
...
>>> send
Opening Socket 192.168.58.129:8888
Return VOID
Closing Socket.
>>> stack
Opening Socket 192.168.58.129:8888
[DEBUG] [Tue Jan 31 04:04:27 2006] proc_rc.c(167): #0 CHAR 0x41
Closing Socket.
>>>
```

After executing 'CPUSH 65', you can very well see it on stack as expected.

```
>>> opcode
Using ticket '\xfc1\x0e\x0cA\xd2\xbaAy(d\xcbL\xf9\xa8X4\xce\x8e\x9a'
... cpush 4
...
>>> send
Opening Socket 192.168.58.129:8888
Return VOID
Closing Socket.
>>> stack
Opening Socket 192.168.58.129:8888
[DEBUG] [Tue Jan 31 04:04:58 2006] proc_rc.c(167): #1 CHAR 0x4
[DEBUG] [Tue Jan 31 04:04:58 2006] proc_rc.c(167): #0 CHAR 0x41
Closing Socket.
>>>
```

This time, execute 'CPUSH 4' and check the stack.

```
>>> opcode
Using ticket '\xfc1\x0e\x0cA\xd2\xbaAy(d\xcbL\xf9\xa8X4\xce\x8e\x9a'
... iadd
...
>>> send
Opening Socket 192.168.58.129:8888
Return VOID
Closing Socket.
>>> stack
Opening Socket 192.168.58.129:8888
[DEBUG] [Tue Jan 31 04:05:33 2006] proc_rc.c(170): #0 INT 0x45
Closing Socket.
>>>
```

We now executed 'IADD' command. I didn't tell you this, but ROVM Server and Clinet does NOT keep connection all the time. Among above messages, you can see 'Closing Socket' there which obviously means Connection is closed. After you send 'REQ' command, all-behind-the-scene work is done through 'Ticket ID'.

Like you have just seen, 'Ticket Structure' holds all kind of operations caused by 'Ticket ID'.

```
>>> reqend
Opening Socket 192.168.58.129:8888
<- OK
Closing Socket.
>>>
```

At last, we drop 'Ticket'. By invoking this command, both 'Ticket ID' and 'Ticket Structure' disappear from ROVM Server.

5 How to deal with Class

5.1 Getting Object Reference

In this section, we are going to learn how the proper use of 'NEW' opcode. 'NEW' opcode is one of the most important ones that exist in ROVM; what it does is getting an object of class you want to use.

C++, Java, and any OOP language provide a way to access an object itself, something like 'this' or 'self'. In ROVM, you can do so by using Object Reference.

This Object Reference allows you to access and manipulate fields and methods within the class.

Let me explain how to get Object Reference using 'NEW' opcode. First, start ROVM client to get one ticket. (I'm assuming ROVM server is open in 192.168.58.129 on port 4390.)

```
C:\projects\rovmclient>python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>> req e://192.168.58.129:4390
Opening Socket 192.168.58.129:4390
Ticket 'Q^\x90?3\xed>.\xa1#h\xf4\xf6\xca\xf5\xa7&\xf3\xe2\xb6'
Closing Socket.
```

Insert ‘new’ opcode while in opcode mode. You can see a full document on ‘new’ opcode in “ROVM Client”.

```
>>> opcode
Using ticket 'Q^\x90?3\xed>.\xa1#h\xf4\xf6\xca\xf5\xa7&\xf3\xe2\xb6'
... new e://192.168.58.129:4390/ABCDEF
...
>>> send
Opening Socket 192.168.58.129:4390
Return VOID
Closing Socket.
```

VOID is returned when we type ‘SEND’. This is obvious because ‘new’ opcode does not change return value in any form. Although you cannot really see how that code gets executed on server side, you can check the stack and verify the result. (Object Reference in stack)

```
>>> stack
Opening Socket 192.168.58.129:4390
[DEBUG] [Fri Feb  3 02:45:56 2006] proc_rc.c(174): #0 OBJREF 0x808c8d8
Closing Socket.
>>>
```

Using ‘STACK’ command, you can clearly see that Object Reference is pushed onto the stack as expected.

So you can access fields or call some methods or even another class (I’m talking about inheritance, but not implemented yet.)

```
>>> reqend
Opening Socket 192.168.58.129:4390
<- OK
Closing Socket.
>>>
```

After you are done, drop the ticket and close connection.

5.2 How to use String Reference

In this section, let’s see how to use String Reference.

StringRef is simply a string as name suggests. It’s almost the same as saying, char *msg = “hello, world!” in C.

However, string is an Object in ROVM, that’s why we call it ‘StringRef’ instead of just ‘string’.

If you ever have programmed in some script languages like Java, Python, or Ruby, you probably have seen something like a function declared in string. You do exactly the same thing in ROVM.

In ROVM Server, there’s an opcode called ‘SPUSH’. What it does is it converts a string (as in C) to StringRef and load it to the stack. Let’s see how we can something interesting using this command.

```
C:\projects\rovmclient>python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>> req e://192.168.58.129:4390
Opening Socket 192.168.58.129:4390
Ticket 'Vc;d\x8f"\x91\xbc2\xcc\x051[\xe2C\x15E2\x13\x8e'
Closing Socket.
>>>
```

Get a ticket and type in 'SPUSH "some string"'

```
>>> opcode
Using ticket 'Vc;d\x8f"\x91\xbc2\xcc\x051[\xe2C\x15E2\x13\x8e'
... spush "Hello ROVM Server"
...
>>> send
Opening Socket 192.168.58.129:4390
Return VOID
Closing Socket.
>>>
```

Now, check out the stack and you will see StringRef is there. That simply means the string we typed has been converted to StringRef and safely stored on ROVM server.

```
>>> stack
Opening Socket 192.168.58.129:4390
[DEBUG] [Fri Feb 10 01:31:50 2006] proc_rc.c(186): #0 STRINGREF 0x40236008
Closing Socket.
>>>
```

Then let's do something even more fun. Currently, we have StringRef (string reference or object), it's possible to call functions (methods as in C++) and get some returns. At the time of this writing, we have split method in StringRef, so let's cut "Hello ROVM Server" in pieces. Continued from above,

```
>>> opcode
Using ticket 'Vc;d\x8f"\x91\xbc2\xcc\x051[\xe2C\x15E2\x13\x8e'
... call split (T)[
...
>>> send
Opening Socket 192.168.58.129:4390
Return Array [(9, 'Hello'), (9, 'ROVM'), (9, 'Server')]
Closing Socket.
>>>
```

We have 3 arrays returned as a result; it looks like they're splitted with whitespace in between them.

5.3 Calling Method

In this section, we are going to talk about how to call methods using 'CALL' opcode, which is newly added in ROVM Server v0.0.7a (7 means there exist 7 opcodes in current version and a is a minor upgrade to 0.0.7).

This time it's assumed that ROVM Server is open in 192.168.58.129 on port 4390.

```
C:\projects\rovmclient>python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>> req e://192.168.58.129:4390
Opening Socket 192.168.58.129:4390
Ticket '\xf6>t\xe7\xd7|\xd4\x99\x10\x8e\x10.\xab\xf3\xce\xfd5\xa1rT'
Closing Socket.
>>> opcode
Using ticket '\xf6>t\xe7\xd7|\xd4\x99\x10\x8e\x10.\xab\xf3\xce\xfd5\xa1rT'
... new e://192.168.58.129:4390/ABCDEF
... cpush 1
... cpush 2
...
>>> printopcode
#0 (29) '\xbb\x00\x0e\x00192.168.58.129&\x11\x07\x00/ABCDEF'
#1 (02) '\x10\x01'
#2 (02) '\x10\x02'
>>> send
Opening Socket 192.168.58.129:4390
Return VOID
Closing Socket.
>>> stack
Opening Socket 192.168.58.129:4390
[DEBUG] [Sat Feb  4 19:09:35 2006] proc_rc.c(168): #2 CHAR 0x2
[DEBUG] [Sat Feb  4 19:09:35 2006] proc_rc.c(168): #1 CHAR 0x1
[DEBUG] [Sat Feb  4 19:09:35 2006] proc_rc.c(174): #0 OBJREF 0x401e4008
Closing Socket.



>>> opcode
Using ticket '\xf6>t\xe7\xd7|\xd4\x99\x10\x8e\x10.\xab\xf3\xce\xfd5\xa1rT'
... call abc (TII)I
...
>>> printopcode
#0 (13) '\xb6\x03abc\x06\x00(TII)I'
>>> send
Opening Socket 192.168.58.129:4390
Return INT = 3
Closing Socket.
>>> stack
Opening Socket 192.168.58.129:4390
Closing Socket.
>>>
```

To summarize, what we are doing here is, first load ABCDEF class, create ObjectRef, push method arguements onto the stack and execute 'call' opcode. And now we have some return.

All the commands we executed before return VOID as its result, but this time we have "Return INT" for the first time meaning whatever command we executed during the process altered the return value. Simply saying, 'Call' opcode returned it. Just for your reference, abc method is implemeneted as follows. For detailed information on language grammar, please see "ROVM Compiler Manual"

```
.class ABCDEF
    .def __init__ (T)V
        nop
    .defend

    .def abc (TII)I
        iload 1
        iload 2
        iadd
        ireturn
    .defend
.classend
```

abc method is supposed to push second and third arguments, add them, and return the sum.