ROVM Reference Document

Weongyo Jeong

2006.3.20

Autralia Melbourne
email : weongyo@gmail.org

Abstract

This document explains detailed reserved-commands and opcodes of ROVM Server and describes the structure of ROVM.

Contents

1 Introduction

ROVM is short for Remote Object Virtual Machine.

This project is a part of envlang project and take charge of executing opcodes at low level. This project is a most important part of project.

The purposes of ROVM are 1) to read/write/execute remote objects, 2) to guarantee the stability communicating each ROVM Servers and 3) to help developer comfortable debugging environment.

When ROVM Server is executed, Server listens for a request of user at default port (a current default port number of ROVM Server is 4390). When a request accepted, ROVM Server process it and return the result.

1.1 License

Currently, the ROVM is licensed under GPL and some part codes of ROVM are licensed under Apache License.

XXX We decided to use GPL license at now, but we aren't sure that we keep it.

2 ROVM Debugging

In this section, we will see how we can debug ROVM Server easily using GDB.

When we are debugging, you need to compile our program with `-g` option and `-O0` option to see a exact line of sources.

```
$ CFLAGS='-g -O0' ./configure
```

To debug ROVM efficiently, you may need a starting point. We recommand rc_process_request () function of $prefix/src/request.c file.

The former part of rc_process_request () function do below.

- It process options and set up functions.
- It open/bind/listen network socket.
- It make a listen thread and work threads
- It make a Garbage Collector thread which do the garbage collection periodically.

and the later part of rc_process_request () function do below.

- It process user's data and return the result to user.
- It runs user's opcodes.
- It loads a class which be made up of "ENVLANG File Format".
- It executes Garbage Collector.

After running ROVM Server, you can use ROVM Client to grasp the point. You can understand easily.

```
$ gdb src/rovm
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it
under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.
Type "show warranty" for details.
This GDB was configured as "i386-linux"...
Using host libthread_libibrary "/lib/libthread_db.so.1".

(gdb) b rc_process_request
Breakpoint 1 at 0x804ebc1: file request.c, line 523.
(gdb) r
Starting program: /projects/rovm/src/rovm
[Thread debugging using libthread_db enabled]
[New Thread 16384 (LWP 9790)]
[New Thread 32769 (LWP 9791)]
[New Thread 16386 (LWP 9792)]
[New Thread 32771 (LWP 9793)]
[New Thread 49156 (LWP 9794)]
[New Thread 65541 (LWP 9795)]
[New Thread 81926 (LWP 9796)]
[New Thread 98311 (LWP 9797)]
[New Thread 114696 (LWP 9798)]
[Thread 16386 (LWP 9792) exited]
```

2.1 Debugging of Garbage Collector

If you want to debug Garbage Collector (aka GC), the simplest way is to print log information on screen. As modifying GGC_DEBUG_LEVEL macro which be defined at `$prefix/src/ext/ggc-zone.c`, you can print log messages on screen. Also you can modify log level of GC.

The levels of GC are as follows.

```
/* Define GGC_DEBUG_LEVEL to print debugging information.
     0: No debugging output.
     1: GC statistics only.
     2: Page-entry allocations/deallocations as well.
     3: Object allocations as well.
     4: Object marks as well.  */
#define GGC_DEBUG_LEVEL (4)
```

3 Reserved Commands

'Reserved Commands' are for communicating each ROVM Server. Also you can connect ROVM Server and use these commands for debugging. These command are not for running VM. You can see the reference manual of opcodes at 4 section.

Let see each items of description below.

Number The unique number of reserved command. A size of reserved command is 1 byte. So we can make commands up to 256.

Size The size of reserved command. Some opcodes have a fixed size, but others have variable size. If reserved command has variable size, you can see a detailed-information at each opcode document.

Return Value If a client or server sends some commands to another server, server which receives data must response that request. This column represents how many cases can be happened.

Return Value Size It means the size of return value.

Explaination A detailed explaination

Example you can see how reserved command can be used to.

Network Status This column represents how the data flows through the network. '→' means SEND, '←' means RECV.

The byte order of all commands which be explained below is a network byte order (big endian)

3.1 REQ

Request a new Ticket ID. If we need to authenticate, this command also enclose with user ID and password.

Number	1 (0x1)
Size	variable size
Format	0x01 `<auth type>`
Return Value	The following cases can be happened. • **TICKET** Ticket ID (Unique HASH value) • **ERROR** ERROR Format
Return Value Size	The following cases can be happened. • If it's **TICKET**, 21 bytes • If it's **ERROR**, it has variable size.
Explaination	Request a new Ticket ID to ROVM Server. Using this ticket, we can send all commands to Server.
Example Network Status	REQ → 0x01 0x00 ← 0x04(TICKET) (Ticket ID sizes which are 20 bytes) ← 0x06(ERROR) ...

Let's see `<auth type>`. This flag points to the mode of the authentication.

If the value of `<auth type>` is 0x00, it points to the **anonymous mode**. When ROVM Server don't permit the anonymous mode, it will return 'ERROR'.

If the value of `<auth type>` is 0x01, it points to the **user authentication mode**. The format is as follows.

```
0x01 <1 byte auth type> <1 byte userid length> <userid string>
     <1 byte password length> <password string>
```

3.2 REQEND

Destroy Ticket ID.

Number	2 (0x2)
Size	1 byte + Ticket ID (20 bytes)
Format	0x02 `<Ticket ID 20 bytes>`
Return Value	`<OK>`
Return Value Size	1 byte
Explaination	Send a signal to ROVM Server which destroy all information and memory allocation related with a Ticket ID. In case of receiving this signal, ROVM Server releases all information of Ticket ID.
Example	REQEND
Network Status	→ 0x02 `<Ticket ID 20 bytes>` ← `<OK>`

3.3 OPCODE

Excutes opcodes.

Number
3 (0x3)

Size
Variable size

Format
0x03 `<TicketID 20bytes>` `<Opcode size : 4 bytes>` `<opcodes>`

Return Value
The following cases can be happened.

- RETURN
- ERROR

Return Value Size

- According to RETURN type, the size is changed.
- ERROR

Explaination
Send opcodes to ROVM Server using a valid Ticket ID. The size of command is 4 bytes and that type is unsigned int.

Example

```
OPCODE    <Opcode>    <Opcode argument>
OPCODE    new         rovm
OPCODE    pushint     1
```

Network Status

→ `<TicketID 20bytes>` 0x00000008 0x51 weongyo

← `<RETURN> <1 byte return type> <According to return type, it's variable size>`

3.4 TICKET

Send a new created Ticket ID. Always the sizes of Ticket ID are 20 bytes and it's SHA-1 binary data.

Number	4 (0x4)
Size	1 byte + Ticket ID (20 bytes)
Format	0x04 `<Ticket ID 20bytes>`
Return Value	Nothing
Return Value Size	Nothing
Explaination	When ROVM Server receives a request from client, ROVM Server make a valid Ticket ID and send it if client has a privilege to create ticket.
Example	**TICKET** `<Valid Ticket ID>`
Network Status	← 0x04 `<Valid Ticket 20 bytes>`

3.5 OK

This command is for a notification that it means a request of the client or server received successfully and be processed.

Number	5 (0x5)
Size	1 byte
Format	0x05
Return Value	Nothing
Return Value Size	Nothing
Explaination	If the return value does not need to be made, ROVM Server send this reserved command to notify the client or server that these requests received successfully. Then the client or server can know whether the delivery status is ok or not.
Example	**OK**
Network Status	
	← 0x05

3.6 ERROR

This command is for sending error messages.

Number	6 (0x6)
Size	1 byte + 2 bytes error msg count + [1 byte error level + 2 bytes str length + string]
Format	0x06 `<2 byte total count> <1byte error type> <2bytes string length> <string>`
Return Value	Nothing
Return Value Size	Nothing
Explaination	When a error message happens during execution, ROVM Server sends 'ERROR' command to the client or server. 'ERROR' command includes the text body.
Exmaple	**ERROR** `<2 bytes error msg count> [<1 byte error type> <2 bytes str length> <string>]`
Network Status	← 0x06 0x0001 0x01 0x0010 "Can't find class"

A detailed-type is like below.

- *2 bytes error message counts*
  unsigned short
- *1 byte error level*
  unsigned char
- *2 bytes string length*
  unsigned short

What is 'error level'?
It means what the level of error is happened during execution and how the error is serious. The levels of error type which supported currently are like below.

Name	Level	Description
ERRLOG_EMERG	0	system is unusable
ERRLOG_ALERT	1	action must be taken immediately
ERRLOG_CRIT	2	critical conditions
ERRLOG_ERR	3	error conditions
ERRLOG_WARNING	4	warning conditions
ERRLOG_NOTICE	5	normal but significant condition
ERRLOG_INFO	6	informational
ERRLOG_DEBUG	7	debug-level messages

3.7 RETURN

This command is for delivering the result which is the return value.

Number	7 (0x7)
Size	1 byte + 1 byte type + [Depending on type, it has variable size]
Format	0x07 `<type>` `<content>`
Return Value	Nothing
Return Value Size	Nothing
Explaination	After finishing a work to process opcodes, ROVM Server always send the result. 'RETURN' command is a kind of the result. Currently, the return type is defined like below and will be added more.
Example	**RETURN** `<1 byte type>` `<the size of type>`
Network Status	← 0x07 0x01 0x00

At below, there is a table to show you what the return types are supported by ROVM Server. The byte order of all return types is little endian except ArrayRef and StringRef.

Name	Type Number	Type Size	Comparing with C language type
VOID	0 (0x0)	0 bye	void
ArrayRef	1 (0x1)	variable size (see below)	
BOOLEAN	2 (0x2)	1 byte	unsigned char
CHAR	3 (0x3)	1 byte	char
SHORT	4 (0x4)	2 bytes	short
INT	5 (0x5)	4 bytes	int
FLOAT	6 (0x6)	4 bytes	float
DOUBLE	7 (0x7)	8 bytes	double
ObjectRef	8 (0x8)	0 bytes	void *
StringRef	9 (0x9)	variable size (see below)	

In case of ArrayRef, it has variable size. The detailed structure is like below.

```
RETURN <ArrayRef_TYPE> <4 byte arrlen> [ <1bytetype> <typecontent> [ ... ] ]
```

If we explain each columns to you,

- *4 byte arrlen*
  The total count of ArrayRef elements.
- *1 byte type*
  The element type. ArrayRef can have many types in one array. That is to say that ArrayRef is independent of ROVM Server's types.
- *typecontent*
  The element value. See 'Type Size' of upper table.

In case of StringRef, it also has variable size and a detailed infomation is explained below.

```
RETURN StringRef <4 byte string length> <strings>
```

Let see a example.

```
arrayref ([(int 4)
           (stringref "Hello World!")
           (float 1.0)])
```

If ROVM Server send a result of processing opcodes like upper, The retuen bytes will be like below.

```
0x07                    (RETURN)
0x01                    (ArrayRef Type)
0x00 0x00 0x00 0x03     (ArrayLen : Network endian)
0x05                    (INT)
0x04 0x00 0x00 0x00     (INT value : little endian)
0x09                    (StringRef)
0x00 0x00 0x00 0x0c     (String Length)
Hello World!
0x06
<4 byte : float data>
```

'RETURN' command is under construction. It will be updated when any changes be made.

3.8 GETSTACK

Get current stack status of Ticket ID.

Number	8 (0x8)
Size	1 byte + 20 bytes TICKET ID
Format	0x08 `<Ticket ID>`
Return Value	**ERROR**
Return Value Size	If it's **ERROR**, it has variable size.
Explaination	Receives and prints the current status of Ticket ID. It's return value is **ERROR** and the type of error be set as ERRLOG_-DEBUG This command is for debugging.
Example	**GETSTACK** `<20 bytes TICKET ID>`
Network Status	→ 0x08 `<20 bytes TICKET ID>` ← 0x07 0x01 0x00

4 Opcode Commands

In this section, we mention all opcodes of ROVM Server. Opcode's size is 1 byte and can be existed up to 256 opcodes. Sorting order is a **numerial order**.

The description of each columns are as follows.

Number Unique opcode number

Format Opcode format.

Stack This column represents how the status of stack can be changed after your opcodes are ran. The left means 'previous stack status' as the basis of '→', the right means 'after stack status'

Example This column represents how you can use this opcode if you use ROVM Client.

4.1 NOP

Do nothing.

Number	0 (0x0)
Format	NOP
Stack	(...) → (...)
Example	

```
... nop
```

4.2 BPUSH

Push a boolean. If 1 means TRUE, 0 means FALSE.

Number	13 (0xd)
Format	dpush `<1 byte unsigned char>`
Stack	(...) → (..., `<boolean>`)
Example	
	... bpush 1
	... bpush 0

4.3 FPUSH

Push a float. It's same with C type 'float'.

Number	14 (0xe)
Format	fpush `<4 byte float>`
Stack	(...) → (..., `<float>`)
Example	... fpush 1.0

4.4 DPUSH

Push a double. It's same with C type 'double'.

Number	15 (0xf)
Format	dpush `<8 byte double>`
Stack	(...) → (..., `<double>`)
Example	
	... double 1.0

4.5 CPUSH

Push a char. It's same with C type 'char'

Number	16 (0x10)
Format	cpush `<1 byte number>`
Stack	(...) → (..., `<char number>`)
Example	... cpush -128 ... cpush 127

4.6 HPUSH

Push a short. It's same with C type 'short'

Number	17 (0x11)
Format	hpush `<2 byte short>`
Stack	(...) → (..., `<short>`)
Example	... short 234

4.7 IPUSH

Push a int. It's same with C type 'int'

Number	18 (0x12)
Format	ipush `<4 byte signed number>`
Stack	(...) → (..., `<int number>`)
Example	
	... ipush -2147483648
	... ipush 2147483647

4.8 SPUSH

Push a string. A string which be pushed changes to StringRef automatically. The size of opcode has variable size.

Number	19 (0x13)
Format	spush `<4 byte unsigned string length>` `<string>`
Stack	(...) → (..., *<StringRef>*)
Example	... spush “Hello! ROVM Server”

4.9 ILOAD

Load int from local variable. The index is an unsigned byte and the value of the local variable at index is pushed onto the stack.

Number	21 (0x15)
Format	iload `<1 byte unsigned local idx>`
Stack	(...) → (..., `<value>`)
Example	

... iload 0

... iload 255

4.10 POP

Pop the top stack value.

Number	87 (0x57)
Format	pop
Stack	(..., `<value>`) → (...)
Example	... pop

4.11 DUP

Duplicate the top stack value.

Number	89 (0x59)
Format	dup
Stack	(..., `<value>`) → (..., `<value>`, `<value>`)
Example	... dup

4.12 SWAP

Swap the top two stack values.

Number	95 (0x5F)
Format	swap
Stack	(..., `<value1>`, `<value2>`) → (..., `<value2>`, `<value1>`)
Example	... swap

4.13 IADD

Add int. Both *value1* and *value2* must int type. The values are poped from the stack. The int *result* is *value1* + *value2*.

Number	96 (0x60)
Format	iadd
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... ipush 1
... ipush 2
... iadd
```

4.14 FADD

Add float. Both *value1* and *value2* must be of type float. The values are poped from the stack. The float *result* is *value1* + *value2*. The *result* is pushed onto the stack.

Number	98 (0x62)
Format	fadd
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... fpush 1.0
... fpush 2.0
... fadd
```

4.15 DADD

Add double. Both *value1* and *value2* must be of type double. The values are poped from the stack. The double *result* is *value1* + *value2*.

Number	99 (0x63)
Format	dadd
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... dpush 1123.0

... dpush 2321.0

... dadd
```

4.16 ISUB

Subtract int.

Number	100 (0x64)
Format	isub
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	... ipush 1 ... ipush 2 ... isub

4.17 FSUB

Subtract float.

Number	102 (0x66)
Format	fsub
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	... fpush 1.0 ... fpush 2.0 ... fsub

4.18 DSUB

Subtract double.

Number	103 (0x67)
Format	dsub
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... dpush 5.0
... dpush 2.0
... dsub
```

4.19 IMUL

Multiply int.

Number	104 (0x68)
Format	imul
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... ipush 1
... ipush 2
... imul
```

4.20 FMUL

Multiply float.

Number	106 (0x6a)
Format	fmul
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... fpush 1.0
... fpush 2.0
... fmul
```

4.21 DMUL

Multiply double.

Number	107 (0x6b)
Format	dmul
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	... dpush 1123.0 ... dpush 212.0 ... dmul

4.22 IDIV

Divide int.

Number	108 (0x6c)
Format	idiv
Stack	(..., <value1>, <value2>) → (..., <result>)
Example	... ipush 10 ... ipush 2 ... idiv

4.23 FDIV

Divide float.

Number	110 (0x6e)
Format	fdiv
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... fpush 10.0
... fpush 2.0
... fdiv
```

4.24 DDIV

Divide double.

Number	111 (0x6f)
Format	ddiv
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... dpush 10.0
... dpush 2.0
... ddiv
```

4.25 IREM

Remainder int.

Number	112 (0x70)
Format	irem
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... ipush 3
... ipush 2
... irem
```

4.26 FREM

Remainder float.

Number	114 (0x72)
Format	frem
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	

```
... fpush 3.0
... fpush 2.0
... frem
```

4.27 DREM

Remainder double.

Number	115 (0x73)
Format	drem
Stack	(..., `<value1>`, `<value2>`) → (..., `<result>`)
Example	... dpush 33.0 ... dpush 22.0 ... drem

4.28 IRETURN

Return int from method. The current method must have return type boolean, char, short, or int.

Number	172 (0xac)
Format	ireturn
Stack	(..., `<value>`) → *empty*
Example	... ireturn

4.29 FRETURN

Return float from method. The current method must have return type float. The value must be of type float.

Number	174 (0xae)
Format	freturn
Stack	(..., `<float value>`) → *empty*
Example	... freturn

4.30 DRETURN

Return double from method. The current method must have return type double. The value must be of type double.

Number	175 (0xaf)
Format	dreturn
Stack	(..., `<double value>`) → *empty*
Example	... dreturn

4.31 ARETURN

Return reference from method. The ObjectRef must be of type reference.

Number	176 (0xb0)
Format	areturn
Stack	(..., `<Reference>`) → *empty*
Example	... areturn

4.32 RETURN

Return void from method. The current method must have return type void.

Number	177 (0xb1)
Format	return
Stack	(...) → *empty*
Example	... areturn

4.33 CALL

Call a instance method; dispatch based on class.

Number	182 (0xb6)
Format	call `<name> <type>`
Byte Format	`<opcode> <1 byte name length> <name string> <2 byte type length> <type string>`
Stack	(..., *<objectref>*, [*arg1*, [*arg2* ...]]) → (...)
Example	... call __init__ (S)V ... call abc (SII)I

If the method have the return value, the return value is pushed onto the stack. If the return type is void, then skip it.

4.34 NEW

Create new object. Memory for a new instance of that class is allocated from the garbage-collected heap. If ROVM Server can't find that class or can't be loaded, Server sends 'ERROR' messages to the client or server. Below is a detailed-information of 'Byte Format'

- Opcode
  Opcode. 1 byte
- Type (1 byte)
  If `<hostname>` format is IPv4, this value will be 0x00, if not, this value will be 0x01.

  Warning) Currently IPv6 is not supported.
- HostName Length (2 bytes)
  The length of hostname string.
- Hostname String
  The string of hostname.
- Port (2 bytes)
  The port number. It's 2 bytes and 'unsigned short' type. If value is 0x0000, ROVM Server will use default port (4390) number.
- Path Length (2 bytes)
  The length of class path. It's 'unsigned short' type.
- Path String
  The location of class which be loaded.

Number	187 (0xbb)
Format	new `e://<hostname>[:<port>]<path>`
Byte Format	`<opcode> <type> <hostname length> <hostname string> <2 byte port> <2 byte path length> <path string>`
Stack	(...) → (..., *<objectref>*)
Example	… `new e://192.168.58.129:4390/testsuite/insert` … `new e://envlang.kldp.net/testsuite/insert`

4.35 NEWARRAY

Create new array. The count must be of type int. It is popped off the stack. The count represents the number of elements in the array to be created. A new array whose components are of length *count* is allocated from the garbage-collected heap. A reference *ArrayRef* to this new array object is pushed into the operand stack.

In ROVM Server, ArrayRef has no type. This means any values can be inserted onto array.

Number	188 (0xbc)
Format	newarray
Stack	(..., `<count>`) → (..., `<arrayref>`)
Example	... ipush 0 ... newarray