Making Extension Module

Weongyo Jeong

2006.2.25

Autralia Melbourne
email : weongyo@gmail.org

Abstract

This document has a detailed information about how you can make a extension module for ROVM Server.

Contents

1 Introduction

This document is still under construction.

The concept of making extention module for ROVM Server is affected by python. So, it has a similar way with python.

The extension module is produced with a thought like below.

- **The extension module is also a class.**
  You can define methods and fields in your extension module as defining those in a class.

  This way which reads a information from 'extension module' is similar with a way which reads a information from a class which has the "ENVLANG File Format" type.

2 Preparation and Compile

To make a extension module for ROVM, you need to install 'a extension module library'. This 'extension module library' is included in ROVM Server package. If you installed ROVM Server on your system, it was already installed.

If you want to write a extension module, you need to check whether the following files exists on your system. Let the path of ROVM Server be `/usr/local/rovm`

- `/usr/local/rovm/include/el.h`
  This header file include structures and prototypes which are supported by 'extension module library'. All extension modules must include this header file.
- `/usr/local/rovm/lib/libel.so`
  This shared library includes real codes of 'extension module library'. This is necessarily for linking and compiling your extension module.

The following Makefile.am file is a example to make a simple str extension module

```
AM_CFLAGS = -Wall -I/usr/local/rovm/include

core_LTLIBRARIES = str.la
coredir = $(prefix)/core

str_la_SOURCES = str.c
str_la_LDFLAGS = -module -L/usr/local/rovm/lib -lel
```

3 Define a entry point

To load your extension module, you need to define a entry point of module. So, you should define a entry point like below.

```
int
init_<modulename> (RvClass *cls)
{
  ...
}
```

<modulename> must equal with the name of class. For example, the name of your class is pcre, the filename must be pcre.so (for UNIX) and the entry point function name must be init_pcre.

An argument CLS is a pointer of class structure and is empty. It will be filled by defining methods or fields. All processes like defining methods or fields are executed by calling functions which be defined at 'extension module library'

The return value of this function will be 0 if it's successful, -1 if it occurs errors

3.1 Memory Management

The functions which be defined in 'extension module library' use garbage-collected heaps. So, you don't need to worry the release of memory.

However, if you make a your own memory allocation, you must release your memory. If you don't care about it, it will make some troubles.

4 Class

4.1 Define Fields

To define fields in the class, you should use RvDefineFields () function which is a pre-defined function.

```
int RvDefineFields (RvClass *cls, RvFieldDef *fds);
```

The first argument CLS is a argument of init_<modulename> and the second argument is a struct pointer of RvFieldDef. Currently, the structure of RvFieldDef is defined as follows.

```
typedef struct efielddef RvFieldDef;

struct efielddef
{
  const char *name;
  const char *type;
};
```

In case of type, you should follow the definition of "ENVLANG File Format'.

For example, you can define like below.

```
static RvFieldDef field[] =
  {
    { "value",  "[" },
    { "len",    "I" },
    { NULL, NULL },
  };
```

4.2 Define Methods

If you want to define a method in the class, you should use RvDefineMethods () function.

```
int RvDefineMethods (RvClass *cls, RvMethodDef *mds);
```

The structure of RvMethodDef is like below.

```
typedef struct emethoddef RvMethodDef;

struct emethoddef
{
  const char *name;
  const char *type;

  int (*func) (RvObject *cls, RvValue *arg, RvValue *ret);
};
```

As you can see, your method definition in the extension module always should like below.

```
int
<funcname> (RvObject *self, RvValue *arg, RvValue *ret)
{
  ...
}
```

Also, the method name and type always should have UTF-8 encoding. Specially, in case of `type`, you should follow the definition of "ENVLANG File Format".

The first SELF argument of the method points to a self ObjectRef. The second ARG argument of the method points to the argument group of the method. To handle the argument of the method, it's good to read 6.1 section. The third RET argument of the method points to the return value space of the method.

At last, let see the return value of function. If your module meets some errors during it runs, you can return -1, then ROVM Server stops running your opcodes and send the error messages which be logged in the ticket to the server or client.

Logging a error using `RvLogError ()` function is documented at 7.3.

5 Fields

5.1 Set and Get the value

To setting a value of the fields, the following functions are supported.

```
int RvSetField (RvObject *self, const char *name, const char *type, ...);
int RvGetField (RvObject *self, const char *name, const char *type, ...);
RvValue *RvGetFieldValue (RvObject *self, const char *name, const char *type);
```

The description of each function is as follows.

- `RvSetField`
  It finds a field that the name of a field is NAME and the type is TYPE, then It sets the value of the field with '...'.

About type, see a table of 6.2 section.

- `RvGetField`
  It finds a field that the name of a field is NAME and the type is TYPE, then It gets the value of th field. '...' is a space to save the value of the field. You need to pass the pointer of variable.

- `RvGetFieldValue`
  It is similar with `RvGetField` function, but this function does not get the value of the field. This function gets the pointer of RvValue structure which is a space to save the value of the field. This function used to handle ArrayRef or StringRef.

5.2 String

The following functions is for handling a string. Correctly speaking, it is a StringRef.

```
RvObject *RvStrNew (RvObject *self, char *str, size_t len);
RvObject *RvStrSubstr (RvObject *self, char *str, int offset, size_t len);
int RvStrGetInfo (RvObject *str, char **ptr, int *len);
char *RvStrGetPointer (RvObject *str);
int RvStrSize (RvObject *str);
```

The description of each functions is as follows.

- `RvStrNew`
  Create a new StringRef which has a string STR with LEN length.

- `RvStrSubstr`
  Create a new StringRef which has a string STR whose length is LEN and a offset of string is OFFSET.

- `RvStrGetInfo`
  Extract the string and the length from StringRef. Both PTR and LEN will be set. If it is success, return 0 or return -1 if it is fail.

- `RvStrGetPointer`
  Get a pointer of string from string object STR.

- `RvStrSize`
  Get a length of string.

5.3 Array

Currently, the following functions are supported to create a array, insert a element into a array.

```
RvObject *RvArrayNew (RvObject *self, int length);
int RvArrayPush (RvObject *array, const char *type, ...);
int RvArrayPop (RvObject *array, RvValue **poped_value);
int RvArrayLength (RvObject *array);
RvValue *RvArrayLastValue (RvObject *array);
RvValue *RvArrayGetItem (RvObject *array, size_t idx);
int RvArraySetItem (RvObject *array, size_t idx, RvValue *val);
int RvArrayResize (RvObject *array, int total);
int RvArrayInsert (RvObject *array, int where, const char *fmt, ...);
int RvArraySliceAdvanced (RvObject *array, int lo, int hi, RvObject *v);
int RvArrayExtend (RvObject *self, RvObject *b);
int RvArrayReverse (RvObject *array);
int RvArrayPopWithIndex (RvObject *array, int i, RvValue **poped_value);
RvValue *RvArrayGetEntry (RvObject *array);
```

The description of functions are as follows.

- `RvArrayNew`
  Create a new ArrayRef. The first argument is a self object and the second argument is a initial length of the array. This function returns the pointer of ArrayRef which be created.
- `RvArrayPush`
  Append a new item to the end of the array. The first argument is a ArrayRef pointer and the second argument is a type string. See a table of 6.2 section.

  If some errors were occurred, this function return -1 or return 0 if it was success.
- `RvArrayPop`
  Removes the last item of the array. POPED_VALUE will be set by the popped value.
- `RvArrayLength`
  Return the length of the array.
- `RvArrayLastValue`
  Find the last item of the array and return the RvValue structure pointer which is a space to save the value of the field.
- `RvArrayGetItem`
  Return the RvValue structure pointer of the index IDX of the array.
- `RvArraySetItem`
  Set the value of index IDX with VAL in the array.
- `RvArrayResize`
  Resize the length of the array ARRAY to TOTAL.
- `RvArrayInsert`
  Insert the value into the index WHERE of the array. Depending on FMT, you should pass arguments properly.
- `RvArraySliceAdvanced`
  If V is NULL, it means del ARRAY[LO:HI]. If V is not NULL, it means ARRAY[LO:HI] = v.
- `RvArrayExtend`
  Append items from B to the end of the array SELF.
- `RvArrayReverse`
  Reverse the order of the items in the array.

- `RvArrayPopWithIndex`
  Removes the item with the index I from the array and POPED_VALUE will be set with the popped value.

- `RvArrayGetEntry`
  Return the address pointer of VALUE which points the space of the value of the array.

5.4 Handle a RvValue

The `RvValue` structure is a space to save the value of the field. The following functions can be used to handle those values.

```
int RvValueIsStringRef (RvValue *value);
RvObject *RvValueStringRef (RvValue *value);
RvObject *RvValueArrayRef (RvValue *value);
int RvValueRichCompare (RvValue *o1, RvValue *o2);
RvValue *RvValueCopy (RvObject *self, RvValue *value);
RvValue *RvValueEntryItem (RvValue *entry, int idx);
```

The description of each function is as follows.

- `RvValueIsStringRef`
  Return a boolean value whether VALUE is StringRef or not. If it is a StringRef, return TRUE. If not, return FALSE.

- `RvValueStringRef`
  Return a StringRef pointer if VALUE is a space to save the StringRef.

- `RvValueArrayRef`
  Return a ArrayRef pointer if VALUE is a space to save the ArrayRef.

- `RvValueRichCompare`
  This function compares the value of O1 and the value of O2. If they are same, it returns 1. If they are not same, it returns 0. If some errors was occurred during calculation, it returns -1. In case of comparing the array, this function compares each item of the array. In case of comparing the object, this function will compare the address of each object.

- `RvValueCopy`
  Create a clone of VALUE and return it. When you save a garbage-collected heap into a stack which means the stack (ESP) of cpu and then garbage thread runs the garbage collection, your object can be disappeared. To provent this situation, this function exists.

- `RvValueEntryItem`
  We assume that the value ENTRY is the array of RvValue structure. This function return the index IDX of the array ENTRY.

6 Methods

6.1 Handle Arguments

If you want to use arguments of the method, you should convert the arguments into C types. `RvParseArg ()` function will support it.

```
int RvParseArg (RvObject *self, const char *fmt, ...);
```

Currently, the following character FMTs (format) are supported.

- '['
  This character used to get a ArrayRef. You should pass `RvObject **` as a argument.
- 'b'
  This character used to get a boolean. You should pass `unsigned char *` as a argument.
- 'c'
  This character used to get a char. You should pass `char *` as a argument.
- 'd'
  This character used to get a double. You should pass `double *` as a argument.
- 'f'
  This character used to get a float. You should pass `float *` as a argument.
- 'h'
  This character used to get a short. You should pass `short *` as a argument.
- 'i'
  This character used to get a integer. You should pass `int *` as a argument.
- 's'
  This character used to get the string of the StringRef. You should pass `char **` as a argument.
- 's#'
  This variant on 's' stores into two C variables, the first one is a pointer to a character string, the second one is its length. You should pass `char **` and `int *` as arguments.
- 'S'
  This character used to get a StringRef. You should pass `RvObject **` as a argument.
- 't'
  This character used to get a ObjectRef. You should pass `RvObject **` as a argument.
- 'x'
  If you want to get a RvValue structure pointer of the argument, you can use this character. You should pass `RvValue **` as a argument.

For example, you can use like below.

```
char *pat;
int patlen, lim;

if (RvParseArg (self, args, "s#i", &pat, &patlen, &lim))
  return -1;
```

6.2 Control the return value

the every method function of the extension module has a pointer of the return value as third argument. Setting this value, you can control the return value of the method.

To use it easily, the extension module library provides functions as follows.

```
int RvSetReturn (RvObject *self, RvValue *ret, const char *type, ...);
int RvSetReturnValue (RvValue *ret, RvValue *value);
```

The each description of functions are like below.

- RvSetReturn
  The first argument SELF points to the first argument of method function, RET points to the third argument of method function and TYPE is a type string.

  For example, the type of the method is `(TSI)[`, the character of TYPE is "`[`". There is a table at below for the proper argument.
- RvSetReturnValue
  If you already have a RvValue value and you want to use it as the return value, you can use this function for it.

Type	C Type
V	No need arguments
[	RvObject *
B	unsigned char
C	char
H	short
I	int
F	float
D	double
T	RvObject *
S	RvObject *

7 Etc

7.1 Memory Allocation

In the current extension module library, the following memory allocation function is supported.

```
void *RvMemMalloc (RvObject *self, size_t size);
```

Let see the description.

- RvMemMalloc
  This function use a garbage-collected heap. Thus, you do not need to free a memory. A memory which be allocated is registered on the Delay Stack Slot automatically. This guarantee you that your memory allocation would not be freed until the end of the method call.

  If you want to use a GC memory with `RvSetUserData ()`, it is not a good way.

  We recommend you using normal memory allocation like malloc when you want to allocate some memories and set it in `RvSetUserData ()`.

7.2 User-defined value

When you code some methods of the extension module, you need to pass your user-defined data to the another method. The following functions provide it.

```
int RvSetUserData (RvObject *self, int index, void *data);
void *RvGetUserData (RvObject *self, int index);
```

The argument INDEX means the index number of the array. Internally, to save the user-defined values, we use a array. The maximum index number of the array is 16.

We recommend you using normal memory allocation like malloc when you want to allocate some memories and set it in `RvSetUserData ()`.

7.3 Logging

In current "ENVLANG File Format", there is no a support for the exception.

However, we provide some logging functions which can be used to debug your module.

You set a log message and return -1 which is the method's return value, ROVM Server sends 'ERROR' messages to your.

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

#define ERRLOG_LEVELMASK    7 /* mask off the level value */

#define ERRLOG_WITHERRNO    (ERRLOG_LEVELMASK + 1)
#define ERRLOG_MARK         __FILE__,__LINE__
#endif

int RvLogError (RvObject *, int, const char *, int, const char *, ...);
```

The first argument must use a self object which is the fir argument of the method function. If you pass the another object, you will meet errors.

The second argument means the level of the error. You can select a level of the error in the upper cases.

The third and fourth argument arguments use for `ERRLOG_MARK`. If your log level is `ERRLOG_DEBUG`, it will print the file path and line number in a log message.

The others should follow the format of `vprintf`. Let see a example.

```
RvLogError (self, ERRLOG_ERR, ERRLOG_MARK,
            "cannot add more objects to list");
```

If you want to receive your log messages, you should return the return value of the method with -1.

7.4 Type Functions

```
const char *RvGetTypeName (RvValue *val);
```

`RvGetTypeName` function returns the type name of VAL.

8 What's the best example?

Reading codes of ROVM Library is the best way to understand how to write a extension module. At now, there are various modules in ROVM Library. We recommends you to read a code of MD5 module whose location is at `$prefix/root/test/_md5.c`.