Quinta semana de trabajo [28 - 02 de Mayo]

Milton Inostroza Aguilera

5 Mayo de 2008

Resumen

Se modifica borrador de protocolo para mejor adpatación entre pyTOD y TOD:

- parentId en el registro de probe cambia de posición.
- value en el registro de return es agregado.

Se plantea modificar la máquina virtual de Python para agregar un identificador único a cada objeto construido dentro del sistema.

Indice

Lista de Figuras

Lista de Tablas

1 Desarrollo

Se hicieron pequeños cambios al borrador del protocolo de comunicación específicamente al registro de probe y al call de los funciones o métodos en lo referente a sus argumentos.

Se comienza a construir el socket servidor desde Java para que pyTOD se pueda comunicar de forma exitosa con TOD.

2 Protocolo de comunicación - Draft

Se muestran las tablas bases para el registro de los objetos y para el envío de eventos:

2.1 Identificadores bases

La siguiente tabla muestra que cada suceso tiene un identificador en el sistema de capturación de huella.

Suceso	Identificador
Registro	0
Llamada	1
Retorno	2
Asignacion	3

Tabla 1: Identificadores de sucesos

La siguiente tabla muestra que cada objeto tiene un identificador en el sistema de capturación de huella.

Id Objeto	Identificador
Clase	0
método	1
Atributo	2
Funcion	3
Variable local	4
Probe	5
Thread	6

Tabla 2: Identificadores de objetos

La siguiente tabla muestra que cada tipo de datos un identificador en el sistema de capturación de huella.

Type	Identificador
int	0
str	1
float	2
long	3
bool	4
other	5

Tabla 3: Identificadores de tipo de datos

2.2 Registro de objetos

A continuación se muestra el formato que tienen el registro de los diferentes objetos dentro del capturador de huellas:

Se describe el registro del objeto función:

eventId	objectId	Id	name	argsN	$\{\text{argName}_i$	$\text{argId}_i\}$
int	int	int	str	int	str	int

Tabla 4: Registro del objeto función

Se describe el registro del objeto variable local:

eventId	objectId	Id	parentId	name
int	int	int	int	str

Tabla 5: Registro del objeto variable local

Se describe el registro del objeto clase:

eventId	objectId	Id	name	classBases
int	int	int	str	–[1]

Tabla 6: Registro del objeto clase

[1] Aún no se toma decisión para poder registrar las super clases que pueda tener la clase registrada.

Se describe el registro del objeto método:

eventId	objectId	Id	classId	name	argsN	{argName$_i$	argId$_i$}
int	int	int	int	str	int	str	int

Tabla 7: Registro del objeto método

Se describe el registro del objeto atributo:

eventId	objectId	Id	parentId	name
int	int	int	int	str

Tabla 8: Registro del objeto atributo

Se describe el registro del objeto thread:

eventId	objectId	Id	sysId
int	int	int	int

Tabla 9: Registro del objeto thread

Se describe el registro del objeto probe:

eventId	objectId	Id	currentLasti	parentId
int	int	int	int	int

Tabla 10: Registro del objeto probe

2.3 Llamada de objetos

A continuación se muestra el formato que tienen las llamadas de los objetos función y método dentro del capturador de huellas:

Se describe la llamada al objeto función:

eventId	objectId	Id	parentId	argsN	argValue_i
int	int	int	int	int	$-^1$

Tabla 11: Llamada al objeto función

Se describe la llamada al objeto método:

eventId	objectId	Id	parentId	classId	argsN	argValue_i
int	int	int	int	int	int	$-^1$

Tabla 12: Llamada al objeto método

Es importante señalar que todas estas llamadas estan acompañadas de los siguientes datos que se describen a continuación:

probeId	parentTimeStampFrame	depth	currentTimeStamp	threadId
int	double	int	double	int

Tabla 13: Coordenadas

[1] Aún no se toma decisión para poder almacenar los valores de objetos primitivos de Python como son: listas, tuplas, diccionarios, enumeraciones.

2.4 Asignación - Modificación de objetos

A continuación se muestra el formato que tienen las asignaciones — modificaciones de los objetos variable local y atributo dentro del capturador de huellas:

Se describe la asignación - modificacion al objeto variable local:

eventId	objectId	Id	parentId	value
int	int	int	int	–[1]

Tabla 14: Registro del objeto variable local

Se describe la asignación - modificación al objeto atributo:

eventId	objectId	Id	parentId	value
int	int	int	int	–[1]

Tabla 15: Registro del objeto atributo

Es importante señalar que todas estas asignaciones - modificaciones estan acompañadas de los siguientes datos que se describen a continuación:

probeId	parentTimeStampFrame	depth	currentTimeStamp	threadId
int	double	int	double	int

Tabla 16: Coordenadas

2.5 Return

A continuación se muestra el formato que tiene el return dentro del capturador de huellas:

Se describe return:

eventId	value	probeId	hasThrown
int	–[1]	int	bool

Tabla 17: Registro de return

[1] Aún no se toma decisión para poder almacenar los valores de objetos primitivos de Python como son: listas, tuplas, diccionarios, enumeraciones.

3 Perspectiva

Para tener un registro total de los movimientos de todos los objetos dentro del depurador se necesita marcar a cada uno de los objetos dentro de este {instancias de clases} con un identificador único. Esto se realiza de forma exitosa para clases construidas por el programador, pero como en Python todo es un *objeto* no tenemos control de los objetos nativos de este, como lo son entre otros: listas, tuplas, enumeraciones, strings, iteradores. Tampoco tenemos control de los métodos que puedan ser llamados desde esos objetos:

```
a = list(2,3,4)
a.append(70)    #settrace no captura la llamada de append
```

Para poder solucionar de primera instancia lo del identificador único se utilizó la funcion *id()*[1]de la librería estandar de Python, pero lamentablemente al momento de destruir un objeto es probable que esta función asigne el mismo identificador que tenía el objeto que ya no existe a un objeto nuevo.

Consultando en el salón irc de Python-es y Python{inglés} y en las listas de correos respectivas se ha llegado a la conclusión que es necesario hacer un hack a la máquina virtual de Python y añadir la funcionalidad que se necesita. Se plantea realizarlo de la siguiente manera:

- Estudiar la forma de construir extensiones en lenguaje C para Python[2].
- Crear un tipo de dato especial para Python que implemente de forma nativa el identificador único.
- Lograr que pyTOD sea capaz de intervenir ese id y utilizarlo para realizar depuración.
- Si todo va bien y no se observan comportamientos extraños, modificar la clase *object* que es la base de las clases nuevas de python, añadiendo el atribudo del identificador único.

¿Será necesario realizar todo esto para el desarrollo de la memoria?, ¿podemos quedarnos con sólo la depuración de objetos más simples como int, bool, double, float, los definidos por el usuario, etc?

Bibliografía

[1] `http://docs.python.org/lib/built-in-funcs.html`

[2] `http://docs.python.org/ext/ext.html`