Trabajo Práctico Número 5:
Múltiples Bibliotecas Dinámicas

Santiago Pérez Heisig, Alejandro Ezequiel Giorgi, Ignacio Gobet
Roberto Iván Bravo

Departamento de Ingeniería e Investigaciones Tecnológicas,
Universidad Nacional de La Matanza, Buenos Aires, Argentina

Resumen. El SO Sodium permite la utilización de Bibliotecas Dinámicas. Estos archivos contienen distintas funciones, que pueden ser utilizadas y compartidas entre varios procesos. El objetivo del presente trabajo es lograr la administración de múltiples bibliotecas dinámicas en Sodium. Para lo cual se realiza la presente investigación.

1 Introducción

El objetivo de esta investigación consiste en obtener la información necesaria para comenzar el desarrollo de la adaptación del Sistema Operativo Sodium para permitir la existencia de varias bibliotecas dinámicas en memoria y ser utilizadas por diferentes procesos de usuario en un momento dado. Para lograr esto es necesario primero ahondar en los siguientes temas: Utilización de la tabla GDT para arquitecturas Intel x86; programación assembler, en especial llamadas a funciones y pasaje de parámetros entre lenguaje C y assembler. Uitlización de la Libc de Linux para apertura y cierre de bibliotecas dinámicas en sus funciones: dlopen, dlsym, dlclose.

2 Alcance

Nos limitaremos al estudio de los temas necesarios a saber para el desarrollo de las funciones y estructuras para la administración de múltiples bibliotecas dinámicas. Cómo así también una investigación teórica sobre el manejo de bibliotecas dinámicas en los diferentes sistemas operativos conocidos.

3 Tabla Global de Descriptores (GDT)

La tabla GDT es una estructura utilizada por la familia de procesadores Intel x86 con la finalidad de definir las características de las áreas de memoria utilizada por programas en ejecución incluyendo varios datos útiles como por ejemplo: la dirección base, el tamaño del segmento de memoria, los privilegios de acceso (lectura, escritura, ejecución, etc).
Cada descriptor en esta tabla posee una longitud de 8 bytes. Cada uno puede ser utilizado indistintamente como descriptor para Task State Segment (TSS), Local Descriptor Table (LDT) o *callgate*
Para que un programa pueda referenciar un segmento, éste debe utilizar el índice de la GDT que describe a dicho segmento, este índice se denomina *selector*. Para ser utilizado normalmente se lo debe cargar en algún registro de segmento. [5]

Para acceder a un byte en un segmento se necesita tanto el selector del segmento como el offset. El selector provee el acceso al descriptor del segmento en la GDT, del cual el procesador obtiene la dirección base del segmento en el espacio lineal de direcciones (de la memoria total). Luego el offset provee la ubicación de dicho byte relativo a la dirección base. Este mecanismo puede ser utilizado para acceder a cualquier código válido, dato, segmento de stack, teniendo el cuenta el nivel de protección que el procesador está ejecutando en ese momento. [6]

4 Interactuando entre C y Assembler

Para escribir funciones en Assembler que puedan ser llamadas desde otras en C se deben saber tres convenciones:

* Pasaje de parámetros

Los parámetros en C hacia una función (tag) en assembler, el compilador los pasa mediante el stack de atrás hacia adelante. Supongamos una función Suma definida en Assembler.

Desde C:

```
extern int Sum();

int a1, a2, x;

x = Sum(a1, a2);
```

El Stack:

```
ESP->EIP
     a1  Valor de a1
     &a2 Dirección de a2
```

De esta manera quedan disponibles desde Assembler en stack dichos valores.

4 Libc: Bibliotecas Dinámicas – Ejemplo práctico

Ejemplo de utilización de carga dinámica de biblitotecas en Linux:

```
#include <dlfcn.h>
#include <stdio.h>

main()
{
  void *libc;
  void (*printf_call)();

  if(libc=dlopen("/lib/libc.so.5",RTLD_LAZY))
  {
    printf_call=dlsym(libc,"printf");
    (*printf_call)("hello, world\n");
  }

}
```

En este ejemplo vemos como se realiza la apertura de la biblioteca (libc.so) con dlopen. Luego con dlsym se encuentra el offset de la función requrida (printf) y mediante la notación de funciones se hace la llamada pasando los parámetros necesarios. [9] y [10]

6 Distintos tipos de Bibliotecas Dinámicas

6.1 SO Windows

Hay cuatro diferentes escenarios en el desarrollo de dll's en Windows, dependiendo de si usan MFC (Microsoft Foundation Classes) o no, y de la forma en que hacen el enlazado con dicha API.

DLL con enlace estático a MFC
Es un DLL que usa MFC internamente, y las funciones exportadas en el DLL pueden ser usadas por ejecutables MFC y por ejecutables no MFC. Las funciones son usualmente exportadas desde los DLL's usando la interfaz estándar de C. Un DLL estáticamente enlazado con MFC no puede ser también enlazado dinámicamente con las DLL's dinámicas de MFC, por lo que está restringido a una aplicación, como cualquier otro DLL. Tiene las siguientes características:

1. El cliente ejecutable puede estar escrito en cualquier lenguaje que soporte DLL's. No tiene que ser una aplicación MFC.
2. El DLL puede "enlazar a las mismas bibliotecas de enlace dinámico usadas por las aplicaciones,

DLL con enlace dinámico a MFC
Es un DLL dinámicamente enlazado con MFC, es un DLL que utiliza MFC internamente, y las funciones exportadas en el DLL pueden ser llamadas por ambos ejecutables MFC y no MFC. Este DLL es construido, usando la versión MFC de bibliotecas de enlace dinámico. Las funciones son usualmente exportadas de un DLL regular usando la interfaz estándar de C. Un DLL dinámicamente enlazado a MFC no puede ser también enlazado estáticamente al propio MFC. Tiene las siguientes características:

1. El cliente puede escribirse en cualquier lenguaje que soporte DLL's, no tiene que ser una aplicación MFC.
2. A diferencia del DLL regular con enlace estático, este tipo de DLL es dinámicamente enlazado al MFC DLL.

DLL con extención MFC
Es un DLL que típicamente implementa clases reusables, derivadas de alguna clase de biblioteca MFC. Son creadas usando la versión de enlace dinámico de MFC. Sólo ejecutables MFC pueden usar esta versión de DLL's. Con este tipo, se pueden derivar nuevas clases de MFC customizadas, y luego ofrecer esta modificación de MFC a quienes usen tu DLL. Se usa generalmente para pasar objetos derivados de MFC entre la aplicación y el DLL. Las funciones objeto asociadas con el objeto pasado existen en el módulo donde el objeto fue creado. Justamente porque estas funciones son propiamente exportadas al usar la versión DLL compartida de MFC, se pueden pasar libremente punteros a objetos MFC entre una aplicación y el DLL de extensión que ésta carga.

DLL Win32 sin uso de MFC
Es un DLL que no usa MFC internamente, y las funciones exportadas en el DLL pueden ser llamadas por ambos ejecutables MFC o no MFC. Las funciones son usualmente exportadas desde un DLL no MFC usando la interfaz estándar de C.

6.2 SO Linux

En Linux, las bibliotecas estáticas (.A) son beneficiosas en programas pequeños con mínima funcionalidad necesitada. Para programas que requieren múltiples bibliotecas, las bibliotecas dinámicas (.SO) pueden reducir notablemente la carga de memoria (ambos de memoria principal y de disco) del programa. Esto se debe a que varios programas pueden usar una biblioteca compartida simultáneamente, por lo que una

sola copia de la biblioteca es necesitada en memoria. Mientras que con una biblioteca estática, cada programa debe tener su propia copia de dicha biblioteca. Los escenarios dentro del marco dinámico son dos, dependiendo de si lo dinámico es el enlace o la carga de la biblioteca:

Enlace dinámico
Se hace el enlace dinámico entre la biblioteca y la aplicación, y Linux carga la biblioteca durante la ejecución (a menos que ya esté en memoria). Generalmente usado en entornos de mucha funcionalidad compartida, en donde se pretende evitar la redundancia de bibliotecas en memoria, permitiendo el uso común de varios programas sobre una única instancia de la biblioteca a utilizar.

Carga dinámica
El programa llama selectivamente a las funciones de la biblioteca en un proceso llamado "dynamic loading" (carga dinámica). Con carga dinámica, un programa puede cargar una biblioteca específica (al menos que ya esté cargada), y luego llamar a una función particular de dicha biblioteca. Es una técnica muy usada al implementar plugins. Es también una práctica usual al implementar programas donde varias bibliotecas diferentes pueden satisfacer el mismo requisito funcional, por lo que el usuario podrá elegir cual o cuales bibliotecas actuarán.

7 Bibliotecas Dinámicas en Sodium

Las bibliotecas dinámicas son archivos que contienen código objeto creado de forma que pueda ejecutarse independientemente de su ubicación. Esto permite que sean cargadas en tiempo de ejecución por cualquier programa. Como consecuencia, se ahorra espacio en memoria ya que los distintos programas no necesitan (como con las bibliotecas estáticas) tener copiado el contenido de la biblioteca a utilizar.

Su implementación en Sodium se encuentra detallada por la investigación del Anexo I: "Informe Bibliotecas Dinámicas 2011" [4].

8 Versionado de Bibliotecas Dinámicas en Linux

El versionado de bibliotecas compartidas debería ser especificado si se esperan cambios en la interfaz de la aplicación (nombres de funciones, cantidad y tipos de parámetros, nuevas funciones, eliminación de funciones, etc).

La versión de una biblioteca puede ser especificada cuando ésta se compila y más aún sí es de esperarse que la interfaz de la biblioteca cambie en el futuro. Esto es importante para mantener una coherencia con los programas de aplicación que puedan utilizarlas. Esto evita el conocido problema "DLL hell" que ocurre en sistemas Microsoft por conflictos de bibliotecas, cuando una actualización de sistema se hace sobre bibliotecas compartidas y causa la rotura de una aplicación existente que requería la versión anterior y no es compatible con la nueva.

En Linux, múltiples versiones de bibliotecas pueden mantenerse en el sistema por razones de versionado y así evitar conflictos con los programas. El número de la versión es incluído en la tabla de símbolos de tal manera que el link-editor pueda saber con qué versión de la biblioteca enlazar. [1]

Referencias

1. Librerías estáticas, dinámicas y cargables en Linux (en inglés)
   http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html
2. Tipos de DLL's (en inglés)
   http://msdn.microsoft.com/en-us/library/9se914de.aspx

3. Bibliotecas de Enlace Dinámico
   http://es.wikipedia.org/wiki/Biblioteca_de_enlace_din%C3%A1mico

4. Investigación anterior de alumnos de la Cátedra:
   Anexo I: "Informe Bibliotecas Dinámicas 2011"
5. Tabla Global de Descriptores (en inglés)
   http://en.wikipedia.org/wiki/Global_Descriptor_Table

6. IA-32 Intel(R) Architecture Software Developer's Manual, Volumen 3A: System Programming Guide. Parte 1
7. Mezclando C y Assembler (en Inglés)
   http://courses.engr.illinois.edu/ece390/books/labmanual/c-prog-mixing.html

8. Como mezclar C con Assembler
   http://programacionbizarra.blogspot.com.ar/2006/06/cmo-mezclar-c-con-cdigo-de-ensamblador.html

9. Carga Dinámica (en inglés)
   http://www.ibiblio.org/oswg/oswg-nightly/oswg/en_GB.ISO_8859-1/books/linux-c-programming/GCC-HOWTO/x796.html

10. Página Man de dlopen (en inglés)
    http://linux.die.net/man/3/dlopen