Организация "исполняемого файла" на базе .so-библиотеки.

Темиргалеев Е. Э.
(24.10.2012 22:37:44 +0400, испр. 30.03.14)

В отличие от *DevLinker* (для Windows), *DevElfLinker* пока не умеет делать выполняемые файлы. Имитировать выполняемый файл можно при помощи динамической библиотеки (.so), в которой слинкованы необходимые, должные быть в выполняемом файле, модули, и вспомогательного файла "пускача" — минимальной Си-программы. Метод был предложен одним из разработчиков проекта OpenBUGS [4], ниже приведено технически уточнённое описание, составленное при участии Ширяева А. В.

1. Метод

Заключается в статической линковке динамической библиотеки с бинарным пускачем-пустышкой на Си. При загрузке пускача ОС подтягивает библиотеку, и собственно Блэкбокс-программа выполняется во время инициализации библиотеки. Таким образом пускач и библиотека, собранная командой *DevElfLinker.LinkDll* или *DevElfLinker.LinkDynDll*, заменяют выполняемый файл, который собирали бы команды *LinkExe* или *Link* (см. *DevLinker*) соответственно, если бы они были в *DevElfLinker*.

Исходник пускача starter.c (имя значения не имеет):

```
"ert0devCmds.RewriteThis('', 'starter.c', 'utf8')"
int main (int argc, char *argv[])
{
return 0;
}
```

Для его сборки динамическая библиотека, имя которой привязывается к бинарному пускачу на этапе (статической) линковки, должна быть в наличии. Содержимое библиотеки роли не играет, поскольку пускач из неё ничего не использует (*).

```
"ert0devCommanders.SelectAndDo('DevCompiler.CompileSelection')"
MODULE Libstub;
END Libstub.
```

```
DevElfLinker.LinkDll example.so := Libstub
```

Достаточно одноразовой сборки пускача, после чего он может использоваться для любой библиотеки с тем же именем.

```
"omcCmdline.Exec('lin')"
  gcc -m32 -o example starter.c example.so -Wl,--no-as-needed,'example.so' -Wl,-rpath,'$ORIGIN' -Wl,-z,'origin' ;
   echo готово ; read
```

Ключи:

— -m32 гарантирует сборку 32-битного приложения и должен быть обязательно указан на 64-битных системах. Дополнительно (для сборки/выполнения 32-битных программ) требуется установка пакета gcc-multilib.

— (*) -Wl,--no-as-needed,'example.so' гарантирует, что потребность библиотеки example.so будет указана линкером в атрибутах выполняемого файла независимо от её использования. В некоторых системах по-умолчанию действует противоположный ключ --as-needed и получившийся пускач своей роли по "активации" библиотеки не выполняет. [1]

— -Wl,-rpath,'$ORIGIN' дополняет выполняемый файл атрибутом-списком путей поиска

требующихся библиотек; $ORIGIN обозначает папку, содержащую выполняемый файл. Данный ключ позволяет без дополнительных настроек размещать рядом две части целого: пускач и .so-библиотеку. [2]
— -Wl,-z,'origin' отмечает возможное наличие $ORIGIN. [1]

Проверить наличие озвученных атрибутов выполняемого файла можно утилитой readelf:
!"omcCmdline.Exec('lin')" readelf -d example ; echo готово ; read
➔выдача ТЕЭ 26.03.2014 11:37:40 +0400➔

Пример 1. "Hello, World!".
Собираем новую библиотеку example.so:
!"ert0devCommanders.SelectAndDo('DevCompiler.CompileSelection')"
```
MODULE Example1;
  IMPORT LinLibc;
  VAR res: INTEGER;
BEGIN
  res := LinLibc.printf("Hello, World!" + 0AX)
END Example1.
```

!DevElfLinker.LinkDll example.so := Example1

Запускаем программу, используя уже готовый пускач.
!"omcCmdline.Exec('lin')" ./example ; echo выполнено ; read

2. Статическая сборка
Нерасширяемый набор программных модулей, компонуемый в выполняемый файл (*DevElfLinker.LinkDll* / *DevLinker.LinkExe*).

Пример 2. "Здравствуй, Мир!".
Для выдачи используется абстрактный (платформенно-независимый) интерфейс *Log*, реализуемый модулем *HostSimpleLog*.
!"ert0devCommanders.SelectAndDo('DevCompiler.CompileSelection')"
```
MODULE Example2;
  IMPORT Log, HostSimpleLog;
BEGIN
  HostSimpleLog.Open;
  Log.String("Здравствуй, Мир!"); Log.Ln
END Example2.
```

!DevElfLinker.LinkDll example.so := Kernel+ Math Strings Log HostSimpleLog Example2
!"omcCmdline.Exec('lin')" ./example ; echo выполнено ; read

3. Сборка с динамическим загрузчиком
Компоновка в выполняемый файл динамического загрузчика модулей, который осуществляет догрузку модулей "на лету" по мере необходимости (*DevElfLinker.LinkDynDll* / *DevLinker.Link*).

Отличие от статической сборки в инициализации — вместо последовательной активации секций инициализации слинкованных модулей в порядке компоновки осуществляется активация тела всего одного модуля — *главного* . Он считается ответственным за инициализацию остальных модулей.

Пускач BlackBox.exe эталонной системы Блэкбокс [3] состоит всего из четырёх модулей *Kernel*, *Files*, *HostFiles* и *StdLoader*. Главным выступает *Kernel*, активирующий инициализацию слинкованных модулей. *StdLoader* во время своей инициализации конфигурирует *Kernel* на использование реализованного в нём механизма динамической загрузки модулей из кодовых файлов (для чего требуются *Files* и его реализация) и (используя его) пытается загрузить модуль *Init*. Дальнейшее развитие событий зависит от *Init* и прочих используемых/задействоваемых им компонентов.

Пример сборки штатного выполняемого файла для Linux:
!DevElfLinker.LinkDynDll example.so := Kernel$+ Files LinErrno HostFiles StdLoader

Пример 3. Динамический "Здравствуй, Мир!".
Приведённый *Init* содержит примеры инициализации необходимых элементов каркаса Блэкбокс:
- модуль *StdInterpreter*, реализующий сервис выполнения команд *Dialog.Call*, — импортируется (хотя и не используется), что обеспечивает его загрузку до *Init*.
- загрузка соответствующего модуля реализации журнала *Log* выполняется посредством выполнения команды *HostSimpleLog.Open*.

"ert0devCommanders.SelectAndDo('DevCompiler.CompileSelection')"

```
MODULE Init;

   IMPORT StdInterpreter, Dialog, Log;

   PROCEDURE Do;
   BEGIN
      Log.String("Здравствуй, Мир!"); Log.Ln
   END Do;

   PROCEDURE Init;
      VAR   res: INTEGER;
   BEGIN
      Dialog.Call("HostSimpleLog.Open", "", res); ASSERT(res = 0, 100)
   END Init;

BEGIN
   Init;
   Do
END Init.
```

Пуск в серверной конфигурации с первичной папкой (здесь — пакет oberoncore-bb-standard-20140330.deb), содержащей необходимые компоненты (*Log*, *Dialog*, *StdInterpreter, Math*, *Strings*).

"omcCmdline.Exec('lin')"

```
   env BB_PRIMARY_DIR=/opt/oberoncore/bb-standard BB_SECONDARY_DIR=$PWD ./example ;
   echo выполнено ; read
```

Список источников

1. http://sourceware.org/binutils/docs/ld/Options.html#Options.
2. http://kernel.org/doc/man-pages/online/pages/man8/ld-linux.so.8.html.
3. http://oberoncore.ru/projects/blackbox.
4. http://oberoncore.ru/projects/openbugs.

Приложение. Исходник CBugs.c

(OpenBugs 3.0.3/Developer/CBugs.c, переформатирован [4])

```
/*    GNU General Public Licence

   This small C program loads the brugs.so ELF shared library and calls the CLI function.
   Save it as a .c file and then compile it on Linux using

            gcc -o bugs CBugs.c -ldl

   This code should work on a Windows machine if brugs.so is repaced by brugs.dll

   A ascii version of this file is in Developer/CBugs.c
*/

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

```
#include <string.h>

int main (int argc, char **argv)
{
   void * handle;
   void (*cli)(void);

   handle = dlopen("./brugs.so", RTLD_LAZY);
   if (!handle)
      return 1;

   * (void **) (&cli) = dlsym(handle, "CLI");
   (*cli)();
   dlclose(handle);

   return 0;
}

/*
   As an alternative to using dlopen to load the brugs library Nathan Coulter has developed the
   following short C program. Compile it on Linux using

               gcc -obrugs -I. -L. brugs.h brugs_cli.c brugs.so

   #include <brugs.h>
   int main ()
   {
      CLI();
   }
*/

/*
   This is equivalent to the component Pascal program

   MODULE BugsClassic;
      IMPORT BugsBRugs;
   BEGIN
      BugsBRugs.CLI
   END BugsClassic.

   (*    DevLinker.Link  dos classicbugs.exe := BugsClassic$ 1 Bugslogo.ico     *)
'/
```