Elanix虚拟操作系统中加载器的设计与实现[1]

高峭，苏杭，王小鸽，陈志成

(清华大学计算机科学与技术系，北京，100084)

Email：gao-z03@mails.tsinghua.edu.cn

摘　要：Elanix 是和欣操作系统(Elastos)在 Linux 上的虚拟操作系统，其目标是使 Elastos 应用程序跨 Linux 系统二进制兼容运行，加载器则是达到这一目标的重要工具。文章在分析 PE 和 ELF 文件格式的基础上，提出了 Elanix 加载器的设计原理和其上的应用程序运行模式。实现了加载过程中的内存空间分配、动态链接库模式转换、自描述构件解析等关键技术。应用表明 Elanix 加载器具有良好的可行性，这为 Elanix 的全面实现奠定了重要基础。

关键词：Elanix，加载器，Elastos，CAR

中图分类号：TP302.1　　文献标识码：A

The Design and the implementation of Elanix Loader

GAO Zheng, SU Hang, WANG Xiaoge, CHEN Zhicheng

(Department of Computer Science and Technology of Tsinghua University, Beijing 100084)

Abstract: Elanix is a Linux-based virtual operating system of Elastos. The goal of Elanix is to provide a compatible environment on Linux, in which the applications of Elastos can be run in the form of binary. Loader is one of the most important tools to achieve this goal in Elanix. This paper analyses the difference between PE and ELF, putting forward the design of theory for Loader and the running framework of Elastos applications in Elanix. The key techniques include implementing the memory allocation during loading process, transforming the model of Dynamic Link Library, and resolving the self-description information from the CAR Components. In practice, it has proved that the design of Loader in Elanix is feasible. Loader provides the vital foundation for the full-scale implementing of Elanix.

Key Words: Elanix, Loader, Elastos, CAR

1. 背景

“和欣”操作系统[1]（英文名称为“Elastos”）是构件化的网络嵌入式操作系统，具有多进程、多线程、抢占式、多优先级任务调度等特性。目前，Elastos已经可以在包括PC、ARM、MIPS等多种体系架构上运行。Elastos提供的功能模块全部基于CAR[2]（Component Assembly RunTime）构件技术，这是Elastos操作系统的精髓。CAR构件技术规定了构件间相互调用的标准，每个CAR构件都包含自描述信息，可以在运行时动态裁剪组装。CAR构件技术贯穿于整个Elastos操作系统技术体系中。

为了使“和欣”应用程序及 CAR 构件在除 Elastos 系统之外的其它操作系统上运行，我

[1]**基金项目**：国家高技术研究发展计划（863 计划）项目（编号 2001AA113400, 2003AA1Z2090）

作者简介：高峭（1973-），男，北京人，硕士，主要研究方向为 WEB 环境下构件运行机制；苏杭（1979-），男，安徽马鞍山人，研究生，主要研究方向虚拟操作系统；王小鸽，清华大学操作系统与中间件技术研究中心，中心副主任，副教授；陈志成，清华大学计算机科学与技术系，博士后，研究方向为操作系统。

们分别在 Windwos 上和 Linux 上研发了“和欣”虚拟操作系统”。其中，文中的 Elanix 便是 Elastos 在 Linux 上的虚拟操作系统。 Elanix 并不是对 PC 硬件的仿真，而是在 Linux 上建立与 Elastos 相似的软件运行环境。通过 Elanix 可以直接加载运行 Elastos SDK 开发的可执行程序和 CAR 构件，从而在 Linux 上实现 Elastos 的跨平台特性。Elanix 即可作为 Elastos 应用程序的运行平台，也可以作为 Elastos 应用程序的开发平台。

Elastos应用程序和CAR构件所采用的二进制文件格式为PE[3]（Portable Executable）格式，与Windows上可执行文件格式相同。然而，Linux系统支持最广泛的可执行文件格式为ELF(Executable and Link Format)[4]，它不支持PE文件格式。因此为了使Elastos应用程序能在Elanix中运行，必须在Elanix中设计实现支持PE格式的文件加载器。Elanix加载器的功能是加载运行Elastos SDK上开发的应用程序和自描述CAR构件。加载过程不仅包括将可执行文件映射到Linux地址空间中，还需要考虑地址空间的分配策略，动态库和CAR构件加载，系统调用实现等方面的问题。文章在分析PE和ELF加载方式、参照WINE[5]的运行机制的基础上，充分考虑Elastos系统和CAR构件的自身特点，设计并实现了Elanix虚拟操作系统加载器，文章对Elanix加载器的设计原理、运行机制、以及实现过程中的关键技术进行了阐述。

2. Linux 与 Elastos 系统中的可执行文件加载方式

2.1 Linux 上 ELF 文件加载

Linux 加载 ELF 可执行文件的的过程如下：首先加载 ELF 文件中的代码段、数据段以及其它相关各段，并加载所需的函数库和动态链接库。加载完毕后，修改 ELF 文件中输入函数地址值，使其为相应加载库中函数的地址。最后将控制交给可执行程序的入口函数， 程序开始运行。一般而言，ELF 文件加载后，其地址空间分布如图 1（a）所示，可以看出，Linux 系统内核占用了 1G（0xc0000000 以上）的地址空间，用户可以使用剩下的 3G 地址空间。

2.2 Elastos 上的可执行文件加载

Elastos 上加载 PE 可执行文件的过程与在 Linux 上加载 ELF 的过程基本一致。Elastos 上加载 PE 文件后地址空间的分布如图 1（b）所示。其中 0x400000（4M）是 PE 文件默认的加载基地址，有些编译器会默认此值并忽略重定位信息，如果碰到这种没有重定位信息的 PE 文件，其将不能加载到除 0x400000 以外的其它地址空间，否则无法进行重定位。PE 格式的 DLL 文件，其默认加载地址是 0x10000000（256M）。一般而言，DLL 中都存在重定位信息，因此可以将其加载到任意的地址空间。Elastos 用户可以使用的空间为 2G，系统使用剩余的 2G。

2.3 二种文件加载的比较

PE 格式和 ELF 格式的区别在于：(1)对文件头的定义和节（section）表的描述上，每个节中的内容也略有差别，但这些差别都是形式上的，其本质都是为了将可执行文件映射到相应的地址空间。(2)Elastos 在加载 CAR 构件过程中与普通的 PE 格式文件还有些不同：在加载 CAR 构件时，需要解析 CAR 构件所包含的自描述信息，并将这些信息保存，以便于其他进程对该构件的访问。(3)ELF 加载起始地址 0x8000000（128M），其下的 128M 空间没有使用，而 PE 默认加载地址是 0x400000（4M）开始的地址，因此 0x400000 到 0x8000000 之间存在 124M 的空间没有被 Linux 使用。(4) Linux 中共享库的加载地址为 0x40000000（1G）开始的地址，Elastos 动态库的加载地址为 0x10000000（256M）开始的地址，两地址间同样存在未使用的地址空间。

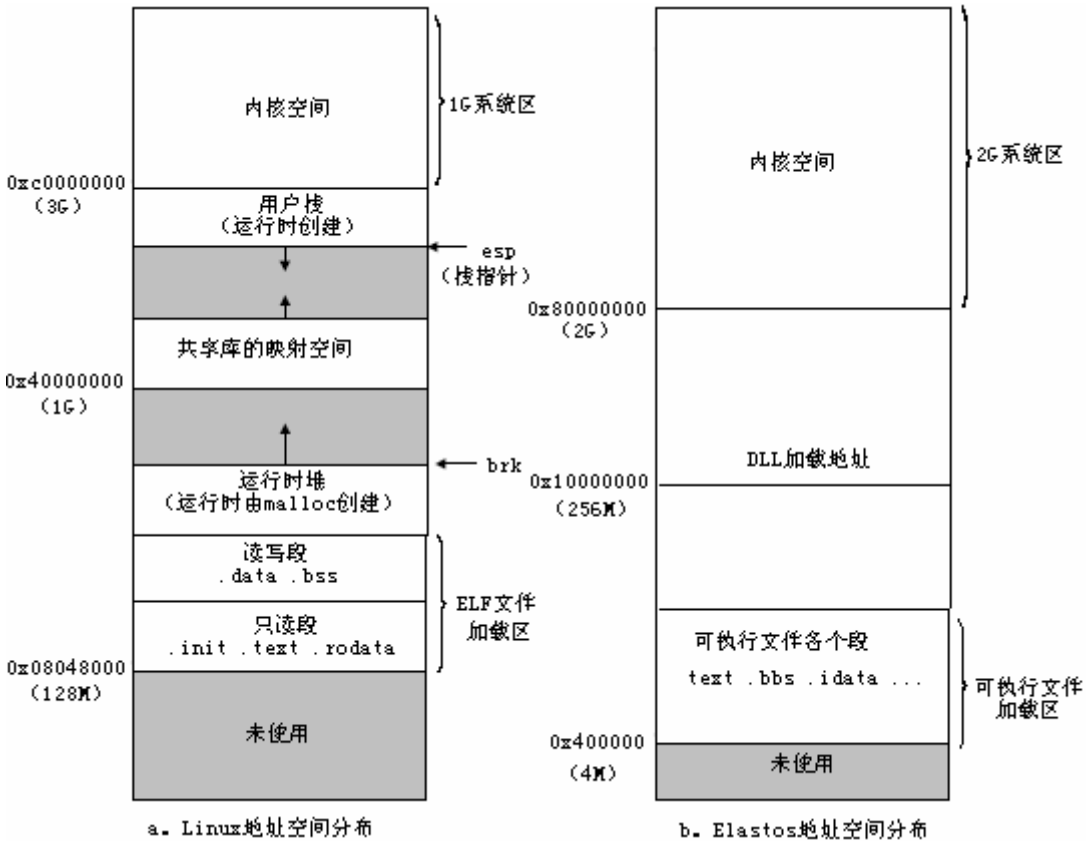


图 1　linux 和 windows 地址空间分布

3. Elanix 加载器设计原理与应用程序运行模式

3.1 Elanix 加载器的设计原理

由于可执行文件格式不同，Elastos应用程序无法在Linux下直接加载运行，但是Elastos应用程序和Linux下的ELF程序之间仍存在一些共同点：两者都是运行在PC机上；使用的机器指令集都是基于IA-32[6]（Intel Architecture 32-bit）；过程调用都是遵循以栈帧[6]（Stack Frame）结构来进行的；最为重要的是，ELF格式和PE格式文件的代码段并没有显著的不同，通过反汇编命令可以看到，两者的汇编指令完全兼容。

指令级的兼容性为执行 Elastos 应用程序提供了运行的基础。基于此，Elanix 加载器的设计原理为：

(1)在 Linux 环境下编写 PE 格式的加载程序（Elanix 加载器），该加载器是一个 ELF 格式的可执行文件，其可在 Linux 环境下加载 Elastos 应用程序、动态连接库以及 CAR 构件；

(2)当加载完成后，进行必要的外部函数地址的设置，然后加载器将控制权交给 Elastos 可执行程序的入口函数，至此加载器的使命结束；

(3)Elastos 应用程序开始运行，直到程序结束。其结构如图 2。

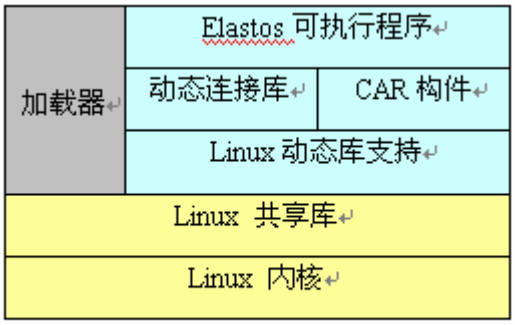


图 2 加载器结构图

3.2 Elanix 的地址空间分布

将 Elastos 应用程序映射到地址空间，需要通过 Linux 系统调用分配所需的地址空间。在 Linux 分配地址空间主要有两种方式：一种是利用 new，malloc 等比较常见的系统函数来从堆中分配地址空间；另一种是通过 mmap 来分配或预留地址空间。Elanix 加载器采用第二种方式分配地址空间，其好处是可以指定分配空间的起始地址，从而可以在指定地址加载 Elastos 应用程序。一般将 exe 文件加载到从 0x400000 开始的地址空间（以防缺少重定位信息），而 DLL 则可加载到任意可用的地址空间。Elanix 加载器加载 Elastos 应用程序和相应的 DLL 后，其地址空间分布如图 3 所示（假设 DLL 加载的起始地址为 0x10000000），其中灰色部分是加载 Elastos 应用程序和 DLL/CAR 文件的可用空间。

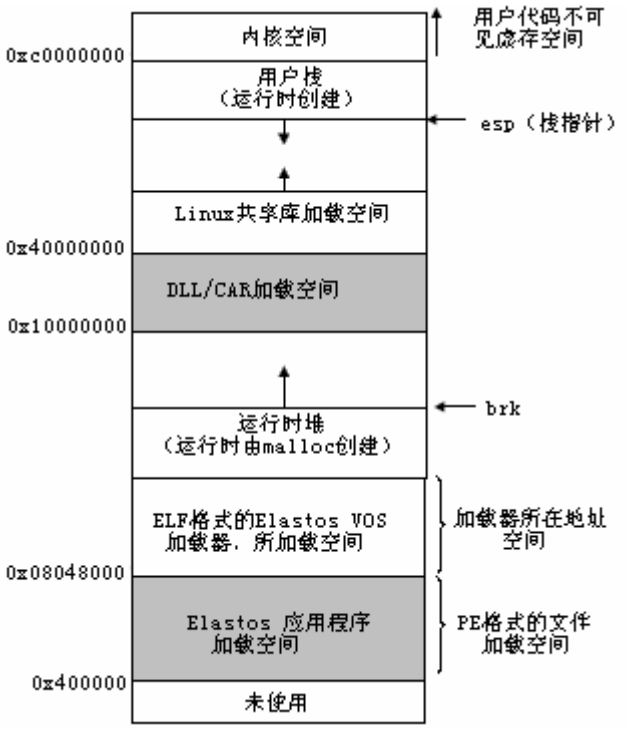
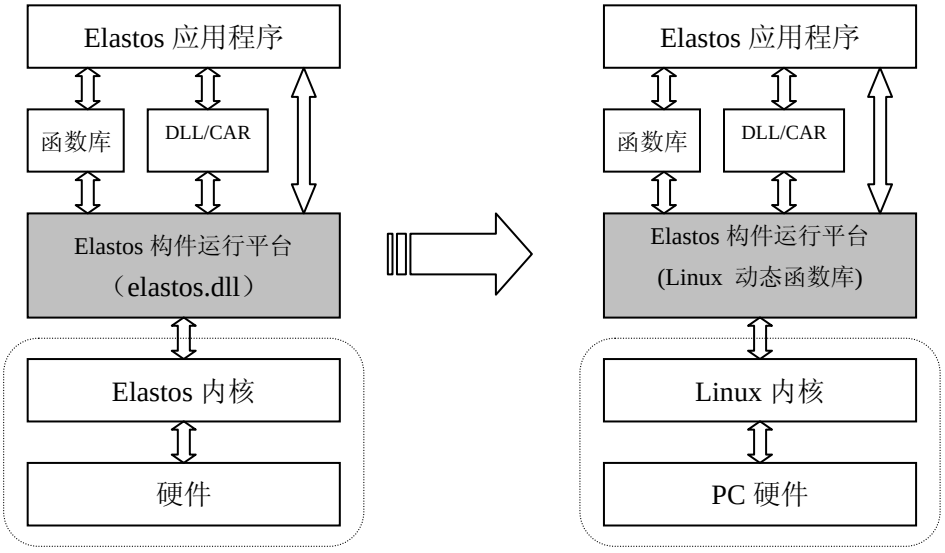


图 3 Elanix 加载器加载 Elastos 应用程序和 DLL/CAR 构件后地址空间的分布

3.3 Elastos 应用程序及 CAR 构件的运行模式

对于 Elastos 应用程序来说，仅仅加载到其所在的地址空间，还无法保障其顺利运行，因为 Elastos 应用程序还需要系统调用支持，这就需要在 Linux 上对 Elastos 的系统调用 API 进行重映射或单独实现。Elastos 系统调用被封装在 Elastos.dll 中。通过分析 Elastos 层次结构（如图 4（a）），可以看出，所有的应用程序以及 DLL/CAR 构件的系统调用都是通过 Elastos.dll 进行的。因此只需在 Elanix 上实现一个与 Elastos.dll 中功能相同的 Linux 函数库，即可提供 Elastos 应用程序所需的系统调用，这样 Elanix 中应用程序的运行结构就变成了图 4（b）所示。图 4 中（a）与（b）灰色的部分隔离了应用程序以及 CAR 构件对底层系统的访问。当 Elanix 加载器将应用程序、DLL/CAR 以及 Linux 上实现的 Elastos 系统调用函数库加载到各自的地址空间后，Linux 中就实现与 Elastos 相同的运行环境，从而保证了 Elastos 应用程序在 Elanix 上的正确执行。



（a） 图 4 Elastos 的应用程序的运行层次 （b）

4. Elanix 加载器实现中的关键技术

4.1 内存分配策略

通过 mmap 可以一次从 Linux 中分配多个页面的空间，但是也要考虑到没有足够的空间供分配的问题，同时还要根据加载的 PE 文件中的段（section）的类型，设置所分配内存的属性，如是否可读写，是否可执行等，以提高安全性和稳定性。因此，有必要维护两张地址分布表，一个记录了已分配的地址空间起止地址，大小和属性等，另一个记录了未分配的地址空间的起始地址，大小等。每次需要对 PE 文件进行空间分配时，就在未分配地址表中查找合适大小的空闲空间，然后进行映射，并将映射结果记录到已分配表中；对于那些不再使用的模块（如 DLL），也可以动态的卸载出地址空间，并将释放的空间加入到未分配表中，以用于他模块的加载。

LoadPEFile 是加载 PE 文件的关键函数，如下框图所示。它可用来加载 PE 格式的.dll 和.exe 文件，参数 stream 是 PE 文件的文件指针，ldBase 是所要加载的地址，该函数实现了将 PE 文件加载到指定的内存区域，并根据情况进行重定位处理。

```
DWORD LoadPEFile(FILE* stream, DWORD ldBase)
{
    //打开文件，判断文件格式，读取 PE 文件中段结点得信息.....
    no = GetSectionNum(stream);//获得 PE 文件中段的个数
    for(i=0;i<no;i++)
    {
        GetSectionInfo(stream, i, &ish);//获得段信息
        start = ldBase + ish.irtualAddress; //计算段加载的起始地址
        FLAG ＝ GetSectionFlag(&ish);//获得段的属性（读，写，执行）
        //用 mmap 进行从段到地址空间的映射。
        mmap((void*)start, pish。SizeOfRawData, FLAG, MAP_ANONYMOUS, -1, 0);
    }
    DoRelocation(stream, ,ldBase);//对加载的 PE 文件进行重定位处理
    ……
}
```

4.2 从 elastos.dll 到 elastos.so 的模式转换

前面已经提出了需要在 Linux 上实现与 elastos.dll 功能相同的库文件，以实现对 Elastos 系统调用的支持。在 Linux 上与 DLL 最相似的就是 so 文件了，so 文件也可以看成是 Linux 下的 DLL。通过 Linux 提供的 dlopen()，dlsym()，等系统函数加载 so 文件，并获得 so 中实现函数的地址，再由加载器修改 Elastos 应用程序所需的外部函数地址值，使其与 so 中获得的函数入口地址一致，这样就实现了从 elastos.dll 到 elastos.so 的转变（如图 5 所示）。

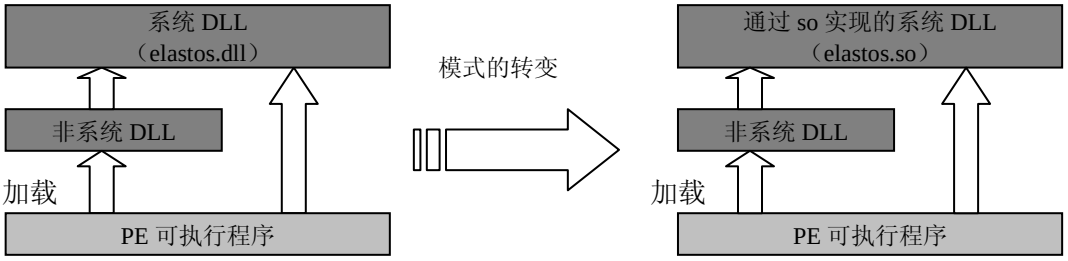


图 5 模式转变

4.3 解析 CAR 构件的自描述信息

CAR 构件包含了自描述信息，这些信息被称为元数据，并被打包在.exe 或.dll 文件的资源段中。构件以接口方式向外提供服务，元数据的作用就是描述接口的相关信息，如接口的函数布局，函数参数属性等。有了这种描述，不同构件之间的调用才成为可能，构件的远程化，进程间通讯，自动生成 Proxy 和 Stub 及自动 Marshalling、Unmarshalling 才能正确地进行。因此，对 CAR 构件元数据的解析是 Elanix 上支持 CAR 构件的关键。可以通过分析 PE 文件的资源段信息，获得 CAR 构件的自描述信息，并根据获得的信息创建构件对象。同时这些元数据需要保存在 Linux 系统的适当位置（Elanix Server 模块负责），以便于 Elanix 加载的不同应用程序之间的通信。

5 应用范例

下面以计算器为例，说明加载的过程。

计算器的文件名为 calculator.exe，其是在 Elastos SDK 环境下开发的应用程序，其可在 Elastos－PC 环境下正常运行。该程序将命令行参数传递给一个能够进行简单计算的 CAR 构件（cal.dll），该构件的需要调用 elastos.dll 提供的字符处理 API，cal.dll 计算完成后，将结果返回给 calculator.exe，calculator.exe 通过 elasotos.dll 提供的系统函数进行打印输出，并将结果在屏幕上显示。模块间调用关系如图 6（a）。

（a）模块关系图　　（b）运行结果

图 6 加载运行示例

在基于 Linux 的“和欣”虚拟操作系统上，加载器加载 calculator.exe 到从 0x400000 开

始的地址空间，将 cal.dll 加载到从 0x10000000 开始的地址空间，并通过 dlopen 加载 elastos.so，以替换 elastos.dll。加载器获得 calculator.exe 所需的输入函数表项，修改成相应的函数地址（elastos.so 的或 cal.dll 中）。完成加载后，将控制交给 calculator.exe 中的入口函数，calculator.exe 运行，直到退出。其运行结果如图 6（b）。

6 结束语

Elastos 操作系统是我国信息产业部重点支持的，自主研发的下一代网络嵌入式操作系统，目前已经在数控设备、医疗仪器、智能通信、数字电视等领域得到应用。考虑到 Elastos 与 Linux、Windows 的共存与兼容，分别研发了 Elanix、Elawin 虚拟操作系统。

文章在分析 PE 和 ELF 不同文件格式的基础上，主要阐述了 Elanix 中加载器的设计原理、Elastos 应用程序在 Elanix 上的执行模式、以及加载过程中的内存分配、动态链接库模式转换、自描述构件解析等关键技术，为 Elanix 虚拟操作系统对 Elastos 应用程序的二进制兼容运行奠定了重要基础，这对从事虚拟机技术相关的科研与工程技术人员具有一定的参考价值。

参考文献：

[1] 《和欣 2.0》资料大全，科泰世纪科技有限公司，2004，11.

[2] CAR 技术简介，，科泰世纪科技有限公司，2004，11.

[3] Microsoft Portable Executable and Common Object File Format Specification, 1999.

[4] Executable and Linking Format Spec v1.2, TIS Committee, 1995.

[5] Uwe Bonnes, Jonathan Buzzard, Zoran Dzelajlija, et al. Wine Developer’s Guide.

[6] Randal E. Bryant, David O’Hallaron. Computer Systems A programmer’s Perspective, Prentice Hall, Inc, 2003.

[7] 胡玉杰，李善平，Windows 程序在 Linux 上的运行，计算机工程，2003 年 7 月 P169-170.

[8] 聂岚，卢正鼎，董俊，魏东林，一种改进的 Linux 内存分配机制，华中科技大学学报（自然科学版），2002 年 7 月，P45-47.

[9] Michael Franz, Dynamic Linking of Software Components, Computer, March, 1997, P74-81.

附：

联系人(作者)： 陈 志 成 （收）

清华大学信息技术研究院：操作系统与中间件技术研究中心

地址： 清华大学信息科学技术大楼4-308室OS中心(100084)

电话： +86 010-62793762-803, 13552869296

传真： +86 010-62793739

Email：Chen-zc@tsinghua.edu.cn