文章编号:1671-4598(2018)06-0126-04 DOI:10.16526/j.cnki.11—4762/tp.2018.06.032 中图分类号:TP311 文献标识码:A

基于模块动态加载机制的航天器软件重构方案研究

郭宗芝，刘 彬，邹玉龙，田小莉，牛 崇
（中科院 微小卫星创新研究院，上海 201210）

摘要：分析了当今航天领域常规星载嵌入式软件重构方式，指出了其安全性低、灵活性差的缺点，提出了一种基于模块动态加载机制的软件重构方案；该方案具有安全、高效、灵活的特点，还能在不中断系统当前任务的情况下完成软件更新工作；借助龙芯 CPU 硬件平台，结合具体实验，验证了文件系统建立、模块加载、模块执行、模块卸载等过程，全面验证了该方案的合理性及可操作性；实验结果表明，采用该方案可以有效降低在轨航天器软件维护工作的难度和成本，同时提高了应用软件开发的团队协作性，对其他嵌入软件开发设计也有一定的参考价值。

关键词：软件重构；动态加载；文件系统；VxWorks

Research on Spacecraft Software Refactoring Scheme Based on Dynamic Loading Mechanism of Module

Guo Zongzhi, Liu Bin, Zou Yulong, Tian Xiaoli, Niu Chong
(Innovation Academy for Microsatellites of CAS, Shanghai 201210, China)

Abstract: Analyzed the current routine embedded software refactoring mode in aerospace, points out its shortcomings of low security and low flexibility, this paper proposes a software reconstruction scheme based on dynamic loading mechanism module. The scheme has the characteristics of safe, efficient, flexible, and can also update the software without interrupting system under the condition of the current task. Using the godson CPU hardware platform, combining with specific experiment, verified the file system establishing, module loading, unloading process execution, fully proves the rationality and feasibility of the scheme. The experimental results show that this scheme can effectively reduce difficulty and cost of the in-orbit spacecraft software maintenance, and also improve the team collaborative in application software development, also have certain reference value to other embedded software development design.

Keywords: software refactoring; dynamic loading; file system; Vxworks

0 引言

当今航天领域发展迅速，在轨、在研航天器型号数量急剧增加，航天器性能、功能不断增强，星载软件的复杂度越来越高，用户需求更新频率越来越快。随之带来软件开发工作也变得更加复杂，软件版本更迭频率也越来越快，在轨航天器的日常维护管理工作日益艰巨。

在轨软件重构作为星上软件更改的唯一方式变得尤为重要，一方面可以修正已知的软件设计缺陷和错误，最大程度的减少损失；另一方面还可通过注入新程序来适应用户新需求的变化，延长了在轨航天器的服役时间[1]。受航天器运行的特殊环境限制，在轨航天器的软件重构对安全性、可靠性有了更高的要求，而对于星上某些特殊设备，还需要充分考虑软件更新后的快速恢复能力，以期最大程度的减少系统中断时间。综上所述，研究可靠的、安全的软件在轨重构方法有很大的实用价值。

本文以在航天领域应用广泛、高可靠性的 VxWorks 操作系统为研究对象，分析现有星载软件的重构方式的弊端，提出了基于模块加载机制的软件重构方案，并借助龙芯硬件平台对该方案的合理性加以例证。

收稿日期:2017-11-15； **修回日期**:2017-12-20。

作者简介:郭宗芝(1990-),男,山东聊城人,硕士研究生,助理工程师,主要从事星载嵌入式软件开发方向的研究。

1 嵌入式软件重构技术研究

VxWorks 操作系统是美国风河公司（Wind River Systems Inc）设计开发的嵌入式实时操作系统（RTOS），以其良好的可靠性和卓越的实时性被广泛应用于航空、航天、军事及通讯领域。

VxWorks 操作系统基于优先级的抢占式任务调度模式，采用资源共享和优先级继承机制，以微内核为核心，扩展网络系统、I/O 系统、文件系统以及其他功能库，用户可以根据需求进行功能组件的增添和裁剪，其中动态加载器组件（Loader Components）是 VxWorks 操作系统实现模块动态加载的核心和基础。

VxWorks 操作系统提供了两种将目标模块加载到内核中的方式，两种方式均能够在不中断系统当前进程的情况下，对指定的部分软件实施维护和更新，来完成对目标模块的动态加载和卸载。一种是在目标机（Target）与主机（Host）建立关联（一般为 Target Server/Agent 方式）的前提下，在主机端端向目标机动态加载目标模块，这种方式一般用来前期软件功能调试，可以有效加快软件开发进度；还有一种为目标机自身借助文件系统、FTP 等方式，动态加载某一特定模块到内核

中，这种方式可以有效增加软件目标机使用和维护的灵活性。本文研究的便是第二种模块加载方式[2]。

1.1 VxWorks 操作系统模块加载机制

对于嵌入式操作系统，软件设计人员通常在程序设计的最后阶段，将操作系统和应用软件链接编译成一个镜像，并将其固化在 ROM 中。当后续测试、使用过程中，发现程序设计错误或者需求更改时，需要修改代码，重新编译，然后更新 ROM 中存储的目标文件，重新加载程序才能完成软件更新。在此过程中，势必会对中断系统的运行状态。

基于上述开发方式生成的目标文件都相对较大，受星地通讯速率限制，在轨更新程序时费时费力，安全性也不好。除此之外，软件重构时还需要停止当前系统进程，完成固化后需重新启用新程序，对系统恢复正常运转也带来不小的麻烦，上述开发方式的软件层次架构如图 1 所示。

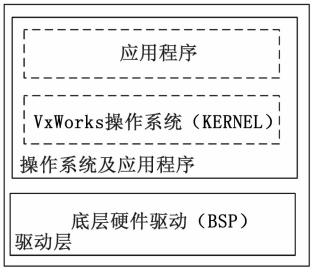

图 1　常规嵌入式软件架构图

而如果在程序设计之初时，按照软件功能划分模块，建立多个子模块过程，编译形成多个目标文件。借助文件系统随时停止、加载、运行某个或某几个特定模块，而不会影响系统当前运行的其他功能，这便是动态加载的核心原理，且单个模块编译生成的目标文件都比较小，维护起来也比较容易[3-6]。

如图 2 所示，基于模块加载机制的软件架构由驱动层、操作系统层、应用层组成，其中操作系统层和驱动层是相对固定，设计人员在对操作系统各个组件完成剪裁、验证后，便相对固定下来，后期改动也较小。为了实现模块的动态加载功能，还必须在操作系统中增加相应的模块加载、管理相关的程序，借助文件系统，根据实际需要，将特定功能模块加载至内核中，更新全局符号表，查找函数符号，运行相应进程。

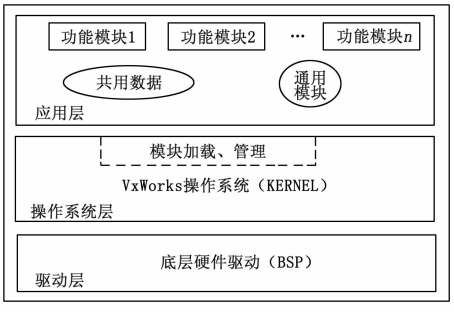

图 2　基于模块加载机制的嵌入式软件架构图

应用层部分是用户自定义部分，是各个项目的特定部分，需要根据需求变化，随时更新代码。而对于航天器的软件，在轨重构也主要是更换这部分程序。

1.2 dosFS 文件系统

VxWorks 操作系统的模块加载组件仅支持 ELF（Executable and Linking Format）文件格式目标文件，这是一种在可执行文件和共享库中都广泛使用的格式，ELF 文件中将执行代码分为代码段（text）、数据段（data）和初值为 0 的数据段（bss）3 个部分，分散的存储在目标文件中，目标文件还包括符号表和重定位信息段信息[7-8]。

对于 VxWorks 操作系统，编译生成的目标文件都是以“.o”或者“.out”（这两种都是 ELF 对象文件）的文件形式存储在 ROM 中，文件中的有些信息，必须通过文件系统才能加载到内核中。在使用文件系统加载符号文件时，首先分析目标文件的文件头表（ELF Header），根据这些信息得到 text、data 和 bss 相关信息，然后在内存映像中建立该模块的信息，然后根据符号信息，对模块符号进行重定位并更新系统符号表，从而完成整个加载过程。

VxWorks 操作系统提供了多种文件系统，包括 VRFS、HRFS、dosFs、rawFS、cdromFs、ROMFS、TSFS 等[9-10]。在 VxWorks 操作系统中的文件系统的架构图如图 3 所示，涉及硬件层、驱动层、应用层等。其中图中方框部分即为文件系统的驱动部分，由其完成对底层硬件资源的分配，并提供相应的接口来供上层 I/O 系统来调用，即 VxWorks 的文件系统和设备驱动程序通过标准的 I/O 操作接口来进行连接。

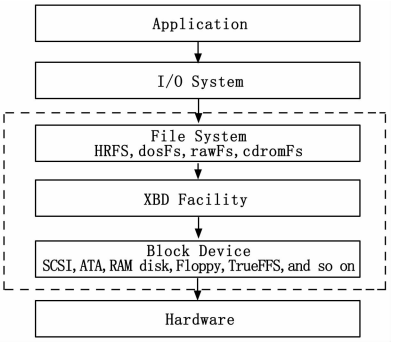

图 3　文件系统架构图

由于本次例程中涉及的文件规模都相对较小，这里选用了相对简单的 dosFs 文件系统来进行实验验证。VxWorks 操作系统的 dosFs 文件系统是一种与 MS_DOS 文件系统兼容的文件系统[11]，能够满足实时应用的多种要求，其与老版本的 dosFs 文件系统相比，有如下新特点：

1）支持层次化的文件和目录结构，能够在一个磁盘上建立一定数目的文件并进行有效的管理；

2）可以将每一个文件指定为连续存储或非连续存储；

3）支持 Microsoft 风格的长文件格式（VFAT）；

4）支持 FAT12，FAT16 和 FAT32 文件分配表格式；

一个简单 dosFs 文件系统可通过建立 RAM Disk 来实现，即分配 CPU 内存的一段存储空间建立软磁盘卷，并将其初始化成 dosFs 文件系统。需要注意的是在分配 RAM Disk 的存储空间时，既可以由操作系统动态分配空间，也可以由用户指定空间位置。而文件系统一旦建立成功，便可以使用通用 I/O 函数（open、close、write、read 等）来访问文件系统，用来新建、打开、关闭以及读写文件了。

2　动态加载机制实验验证

基于上述理论，我们在龙芯 CPU 硬件平台上加以验证，

龙芯 1E 是龙芯中科技术有限公司研制的基于 MIPS 架构的高性能抗辐照宇航级 CPU 芯片，提供了通用的处理器部件和对外接口，包含中断控制器、定时器、RS232 串口控制器、浮点处理器、PCI 和存储器接口（存储器接口支持 SDRAM 和 FLASH ROM）等，龙芯 1E 芯片的外部时钟频率不低于 66 MHz，功耗 3 W[12]。操作系统为 VxWork6.8，开发环境为 WindRiver Workbench3.2。具体配置如表 1 所示。

表 1 验证平台资源配置表

项目	参数	备注
CPU	LS1E	Loongson1E，主频 50～100 MHz
内存	SDRAM	128 MB 支持 ECC 校验
ROM	FLASH	NOR FLASH 512 KB＊2
操作系统	VxWorks6.8	Workbench3.2，MIPS 版本

为了充分验证上述方案，需依次验证模块的固化存储、加载、执行、删除等过程。在验证时还设计了多个模块文件同时加载以及目标文件压缩、解压的过程，验证该方案的可拓展性。

1）首先分别建立操作系统工程和子模块工程，在此次验证实验中，建立了一个 VxWorks Image Project（VIP）和两个 Downloadable Kernel Module Project（DKM），建立两个子模块工程的目的在于验证存在多个模块条件下，函数模块相互调用情况。其中对于两个 DKM 工程，其中一个 DKM 工程仅含有一个函数：

```
void Hello1(void)
{
printf("--- this is Hello1 DKM example ---\r\n");
}
```

另外一个 DKM 工程包含两个函数：

```
void Hello2(void)
{
printf("--- this is Hello2 DKM example ---\r\n");
}
void Hello3(void)
{
printf("--- this is Hello3 DKM example ---\r\n");
}
```

而对于操作系统工程，除了必要组件之外还需要在其中完成模块管理的相关操作，包括建立文件系统，新建模块文件，加载模块，执行模块，删除模块等操作，其中涉及的用户自定义函数有“usrRamDiskInit”、“deleteModule”、“runModule”、“runModule1”等，其中涉及的函数作用和调用的系统函数如表 2 所示。

2）在 Workbench 开发环境中，逐个编译上述工程，分别生成“vxWorks”、“Example1.o”、“Example2.o”等目标文件，其中目标文件“Example2.o”利用 VxWorks 操作系统自带的压缩算法（deflate）生成压缩文件“Example2.z”。将上述目标代码依次固化到 NOR FLASH 中，通过压缩算法可以有效减小目标代码占用。制作压缩文件时可直接在 Workbench 中通过 shell 命令来完成，可采用以下示例：deflate ＜UserApp.out＞ UserApp.bz。

系统上电后，引导程序首先将操作系统文件从 FLASH 加载到 SDRAM 中，之后便开始可以使用用户保留空间建立 RAM Disk，并在其中搭建 dosFs 文件系统。

文件系统建立成功后，依次在其中新建“Example1.o”、“Example2.o”两个文件，文件内容分别从 FLASH 存储的“Example1.o”、“Example2.z”拷贝得到。在处理压缩文件“Example2.z”时，必须先解压存储的目标代码后再进行数据拷贝，解压文件可以使用函数“inflate”来完成，这样包含模块的两个文件已经存在文件系统中了。

3）最后利用 VxWorks 操作系统提供的符号加载函数将其加载到内核之中，操作系统提供了“symLib”、“loadLib”两个函数库可供用户调用。一旦加载完毕，用户就可以调用当前文件中包含的所有函数了。

上述几个过程涉及的目标代码存储及转移的信息流如图 4 所示。

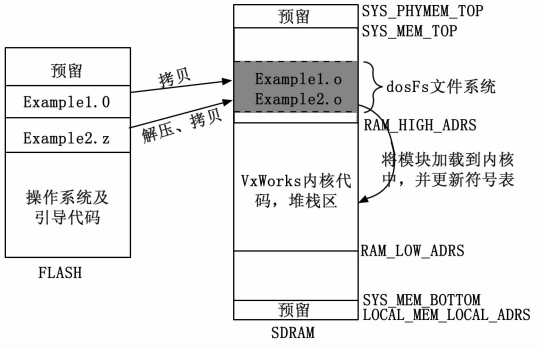

图 4 目标代码存储及转移信息流图

4）在上述软件架构下，当需要进行软件重构时，只需要停止该进程，将特定的目标文件替换，然后重新加载该符号至内核，重新启用该进程，即可完成软件重构整个过程。以上述例程为例，当需要替换“Example1.o”模块时，首先需要在把内核中该模块停止，然后替换该文件，加载该模块到内核中，重新该进程即可。

由于单个目标文件都相对比较小，并且重构时也不需要中断系统的其他进程，可以将对整个系统的影响做到最小，以上过程涉及到的自定义函数如表 2 所示。

表 2 涉及的自定义函数列表

函数	调用的系统函数	备注
usrRamDiskInit	xbdRamDiskDevCreate，dosFsVolFormat，open，write，close	建立 dosFS 文件系统，并新建文件
deleteModule	moduleFindByName，unldByModuleId	查找相应模块并将其从内核中删除
runModule	loadModule，symFindByName，open	加载”Example1.o”中的所有函数
runModule1	loadModule，symFindByName，open	加载”Example2.o”中的所有函数
Hello1	/	包含”Example1.”中用于打印信息
Hello2	/	包在”Example2.o”中用于打印信息
Hello3	/	

3　结果分析

为了直观显示出模块加载的全过程，可以利用 VxWorks 操作系统的 shell 工具，通过命令的方式依次调用各个函数模块，执行上述过程。

1）系统上电后，操作系统首先被加载起来，正常运行后，可以从输出信息中得到操作系统及硬件的启动信息，如图 5 所示，从中可以看到，例程使用的操作系统版本为 VxWorks 6.8，硬件平台为 Loongson1E。

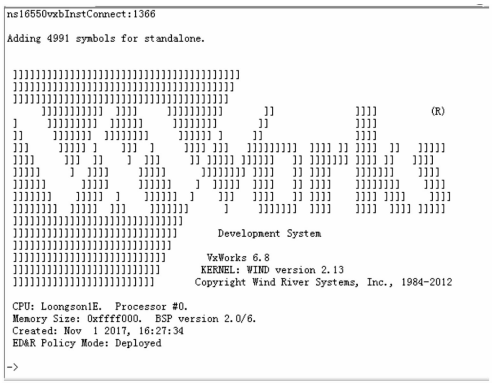

图 5　VxWorks 操作系统启动信息

2）系统初始加载后，此时文件系统还未建立，因此通过“devs”命令查询当前系统设备信息，如图 6 所示，可以看出设备列表中还没有磁盘卷信息；

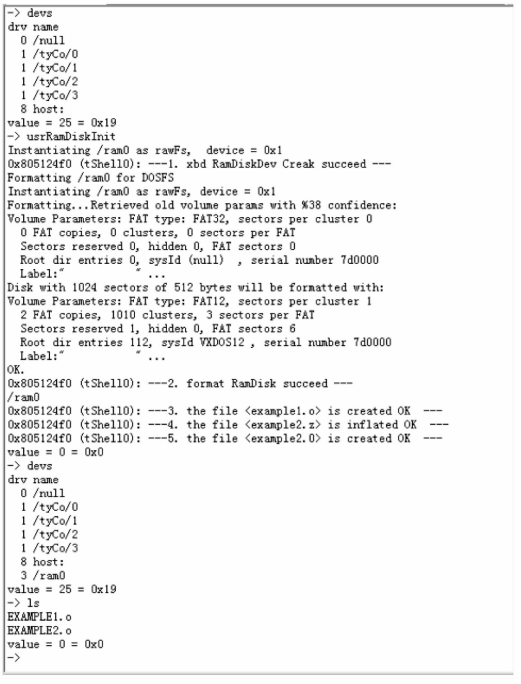

图 6　dosFs 文件系统输出信息

通过命令行调用函数“usrRamDiskInit”，函数执行后，操作系统建立了 512 KB 大小的磁盘卷并将其格式化成“dosFs”格式，与此同时文件“Example1.o”、“Example2.o”也被建立起来。

通过命令行输入命令“devs”后，看以看出系统设备目录下新增了“/ram0”设备。输入命令“ls”可以查看到当前目录下的文件列表，可以看到当前目录下已经存在文件“Example1.o”和“Example2.o”了，这说明文件系统以及文件都被成功建立。

3）虽然文件系统已经建立起来，但依然还未将文件“Example1.o”、“Example2.o”中的模块加载到内核中，此时通过命令行调用函数“Hello1”、“Hello2”、“Hello3”时，系统输出错误信息。

通过命令行调用函数“runModule”和“runModule1”后，两个文件包含的所有函数模块均被加载至内核中，此时可以通过命令行正常调用相关函数，运行结果如图 7 所示，其中在“runModule”函数中在加载成功后同时也调用了函数“Hello1”。

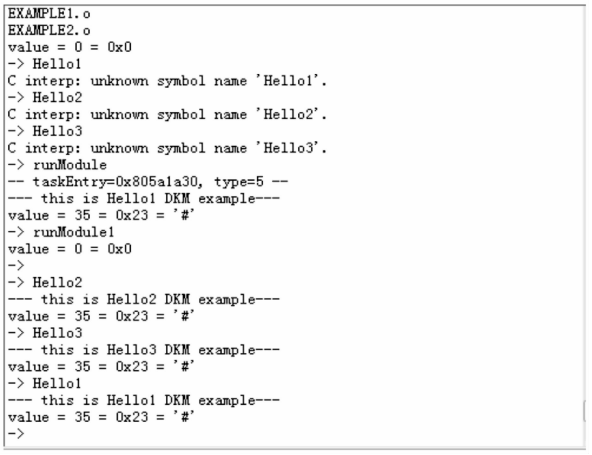

图 7　子函数调用执行信息

5）当模块需要更改时，应先停止相应的线程，并将该模块从内核中删除。通过命令行输入命令“moduleShow”可以查看当前目录下的模块信息，可以看出当前内核中存在“example1.o”、“example2.o”模块。通过命令调用“deleteModule”函数将模块“example1.o”从内核中删除，此时再调用函数“Hello1”，操作系统输出错误调用信息，当通过调用函数“runModule”又重新将模块加载到内存中，可以继续调用

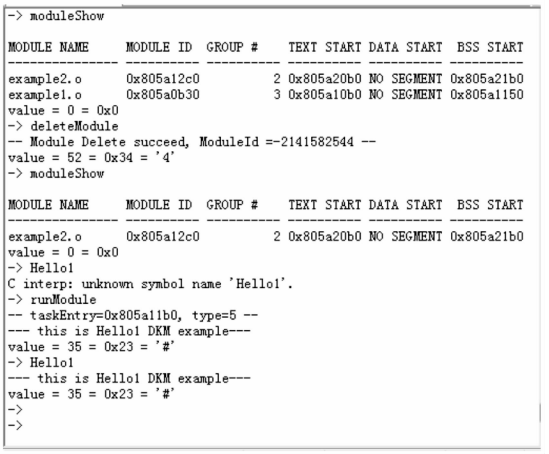

图 8　模块删除与重加载信息

（下转第 148 页）