轨道交通实时以太网交换机启动性能的分析与优化

佘 磊[1] 赵曦滨[1] 陈 渝[2] 施鹤远[1] 韦 康[1]
(清华大学软件学院 北京 100084)[1] (清华大学计算机科学与技术系 北京 100084)[2]

摘 要 轨道交通实时以太网交换机作为新一代列车通信网络的重要组成部分,保障了列车状态信息与控制命令的交换和传输。实时以太网交换机基于嵌入式Linux系统构建,系统启动包括BootLoader启动、内核镜像加载、内核启动、用户空间初始化4个部分,其启动耗时直接影响着交换机甚至整个列车通信网络的性能。通过对嵌入式Linux启动过程与JFFS2,UBIFS等文件系统在嵌入式系统上挂载耗时的研究与分析,针对交换机启动过程提出相应的加速策略。采用内核裁剪、调整文件系统类型与优化系统启动参数等优化策略,使交换机的启动性能大幅提升。实验表明,交换机系统的启动时间从原有的26.69s减少至7.15s,启动时间减少了73.2%。

关键词 嵌入式Linux,交换机,启动优化,文件系统,内核裁剪
中图法分类号 TP399,TP316 **文献标识码** A

Analysis and Optimization of Boot-up Performance for Railway Real-time Ethernet Switch

SHE Lei[1] ZHAO Xi-bin[1] CHEN Yu[2] SHI He-yuan[1] WEI Kang[1]
(School of Software, Tsinghua University, Beijing 100084, China)[1]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[2]

Abstract As the important component for the next generation of train communication networks, railway real-time Ethernet switch ensures the data exchange and transmission for status and control information. The boot-up of the switch, based on the embedded Linux system, includes BootLoader starting, kernel image loading, Linux kernel boot and user space initialization. The boot-up time directly affects the performance of switch or even the entire train communication networks. The boot-up process of the embedded Linux system and the mounting cost for various file systems, e. g. JFFS2 and UBIFS, were studied. The strategy for each process in boot-up was proposed to reduce the time cost. By reducing kernel, replacing file system and optimizing boot-up parameters, the performance of system boot-up was significantly improved. Experiments show that the boot-up time decreases from 26.69 seconds to 7.15 seconds, which is reduced by 73.2%.

Keywords Embedded Linux, Switch, Boot-up optimization, File system, Kernel reduce

1 引言

轨道交通基础通信设备是列车信息交换网络的重要载体。随着机车的不断更新发展,列车网络传输和处理的信息类型和数据量不断增多,这对下一代轨道交通核心交换机的设计提出了很高的要求,如数据传输实时性、支持多种网络协议、系统启动迅速。

轨道交通实时以太网交换机采用PowerPC架构的嵌入式Linux系统构建。对于嵌入式系统,常用的操作系统有Linux,VxWorks,QNX,eCos,WinCE,FreeRTOS等[9],其中,Linux凭借其灵活性、健壮性和开源,发展迅速。近年来,开发者不断对Linux内核进行改进,实时性得到了提高。这些改进使得Linux内核能与商用实时操作系统相媲美[8-9],也使得Linux更加适用于嵌入式控制领域。

嵌入式系统不同于服务器级和桌面级系统,对于桌面级系统,系统启动性能对用户使用体验有影响,但不影响系统核心业务。对于服务器级系统,系统启动后长时间运行,不会频繁开关机,因此其启动性能不是主要关心的问题。而对于嵌入式系统,用户对系统的启动时间比较敏感。特别在智能设备越来越多的今天,用户不愿意花费几十秒甚至是几分钟来等待设备就绪,因此减小嵌入式系统的启动时间十分重要。

当前有很多加速Linux系统启动和分析Linux启动过程的开源项目。如开源项目initng(The Next Generation Init System)和upstart,它们通过将Init进程并行化来加速系统启动过程。Init进程是内核启动后的第一个进程,负责用户空间初始化服务。Linux系统原有的Init进程是串行的,不能充分发挥多核CPU的优势。initng在启动时会检查启动项之间的依赖,重新安排启动顺序,不仅减少了等待延迟,也实现了较好的CPU与I/O平衡[19]。upstart通过脚本设置异步事件实现了启动的并行化,目前已被Ubuntu发行版本采

本文受核高基重大专项(2016ZX01038101)资助。

佘 磊(1986-),男,硕士生,主要研究方向为信息安全、操作系统,E-mail:shel14@mails.tsinghua.edu.cn;赵曦滨(1973-),男,副教授,主要研究方向为异构网络可靠性分析、信息系统安全、人工智能等;陈 渝(1973-),男,副教授,主要研究方向为操作系统、普适计算、嵌入式系统等;施鹤远(1993-),男,博士生,主要研究方向为嵌入式系统、无线传感网络;韦 康(1993-),男,硕士生,主要研究方向为操作系统。

纳为 Init 的替代程序[20]。Linux 系统提供了启动性能分析工具，例如 Bootchart。Bootchart 通过 proc 文件系统收集 Init 进程的运行信息，跟踪 Init 进程启动，统计启动过程中 CPU 和 I/O 的使用情况，可以帮助用户分析系统启动的耗时瓶颈，生成启动引导图[18]。但嵌入式 Linux 系统由于高度定制化，很多分析工具无法直接使用或不适用，这给嵌入式系统的启动优化带来挑战。

目前嵌入式系统启动加速的研究偏向于对局部功能模块的优化，如文件系统[12,14]，并没有系统性地解决启动加速问题。本文从实际项目出发，系统性地分析了影响嵌入式系统启动性能的因素；针对每个耗时因素给出优化策略，并进行了对比实验。

2　嵌入式 Linux 系统的启动过程分析

嵌入式 Linux 系统的启动大致划分以下 4 个阶段：BootLoader 启动、内核镜像加载、内核启动、用户空间初始化。开发者根据平台架构和开发需选择相应的 BootLoader，本交换机选用 U-Boot 作为 BootLoader[10]。BootLoader 启动阶段是从 U-Boot 启动加电到其调用 bootcmd 之前。内核镜像加载阶段是从 bootcmd 执行，到 head.S 中内核的第一条指令执行之前。内核启动阶段是从内核第一条指令执行开始，到调用系统初始化进程(/bin/init)之前。用户空间初始化阶段是从调用系统初始化进程开始，到自定义初始化脚本(/etc/rc)执行结束。为了优化交换机启动性能，首先统计系统启动的各个过程及其耗时，以便针对其耗时较大的部分进行相应优化。

为详细记录系统启动中的关键启动过程及其耗时，需要对配置做如下修改。首先在编译 U-Boot 和 Linux 内核时打开 debug 选项，这样在系统启动过程中可以尽可能多地输出日志信息。为了显示日志信息的时间戳，需打开内核 printk 配置的时间戳功能。U-Boot 启动、内核镜像加载等部分的时间戳只能使用 ts 命令在其日志输出时添加时间戳。获取的关键启动信息如表 1 所列。

表 1　系统启动耗时

启动阶段	时间戳/s	标志性启动事件	耗时/s
U-Boot 启动	0.6415	PCI 设备初始化	0.1160
	0.6445	标准输入、输出、出错初始化	0.0029
	1.7139	网络设备初始化	1.0691
	1.9278	IDE 总线初始化	0.2141
	3.1280	IDE Device 侦测	1.2000
	6.1298	U—Boot 等待进入下载模式延迟	3.0018
加载内核镜像	6.1549	加载内核镜像完成	0.0209
	8.3327	镜像文件校验完毕	2.1778
	12.7165	内核镜像文件解压	4.3837
内核启动	12.7504	输出第一条日志信息	0.0338
	14.3496	模块初始完毕，准备加载文件系统	1.5992
	23.7307	JFFS2 文件系统挂载完毕	9.3810
	23.8477	内核资源释放	0.1170
	25.8487	挂载 proc，sys 文件系统	2.0010
用户态程序启动	26.1356	启动脚本运行	0.2869
	26.7376	用户空间应用启动	0.6020

基于表 1 的系统启动过程耗时来统计启动过程中主要的耗时阶段及其占比，结果如图 1 所示。可以看出，启动耗时主要集中在 JFFS2 挂载、内核镜像解压、U-Boot 等待延迟等部分。下面分析测试中主要耗时阶段的具体过程，从通用场景与实际交换机需求两方面，提出启动性能优化的方法和策略。

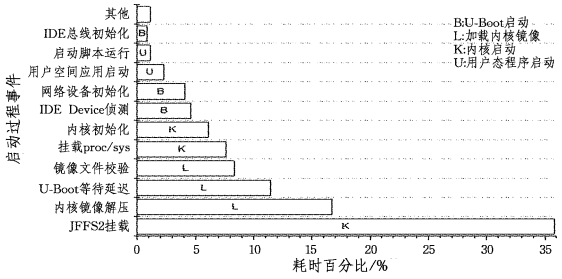

图 1　主要耗时事件

3　系统启动性能优化策略

嵌入式系统从构建开始就要从应用实际出发，考虑整体的性能和功能。硬件方面要选择适合性能的开发板，确定相关外设；软件方面可使用轻量级的库替换标准 C/C++库、定制化 BootLoader[6] 和内核、合适的文件系统、小型化的命令工具集(如 Busybox)等方法[22]。本节从嵌入式文件和存储方案的选择入手，重点给出了 U-Boot、内核镜像加载、内核启动、用户态程序 4 个方面[12]减少耗时的策略，并针对实际场景给出了优化方法。

3.1　嵌入式文件系统的选择和存储方案的优化

对于嵌入式文件系统，主要从文件系统挂载耗时和读写性能进行分析，针对特定的应用场景进行择优[11]。交换机启动过程中耗时占比最大的部分为文件系统挂载，文件系统挂载对启动性能影响巨大，因此本文仅关注文件系统在挂载方面的性能表现。

在嵌入式环境中，存储设备以 Flash 芯片为主。Flash 芯片支持 3 种操作：对页读或写，对整个块的擦除。在对 Flash 中的页进行写之前，必须对页所在的块整体进行擦除(块擦除操作实际上是将其物理上置为 1)。Flash 设备的块擦除次数是有限制的，即 Flash 芯片的寿命是有限的。文件系统在块中记录此块的擦除次数，并进行读写平衡，以达到延长 Flash 寿命的目的[13]。

如图 2 所示，在 Flash 设备上构建文件系统有两条路线：1)直接在内存技术设备(Memory Technology Device，MTD)上构建文件系统；2)使用 FTL(Flash Translation Layer)技术将 MTD 模拟块设备后再构建通用的磁盘文件系统。MTD 层构建在 Flash 设备的硬件驱动之上，与通用块设备层(Generic Block Layer)处于同一层。在 MTD 上构建适合 Flash 设备的文件系统要尽量满足 3 方面的要求：数据压缩、坏块管理、垃圾回收机制。数据压缩提高存储空间的使用效率，减少了 I/O 负载，但增加了 CPU 的负担，数据压缩算法需要在 I/O 负载和 CPU 负载直接做出平衡。Flash 芯片从出厂开始就存在坏块，在使用过程中坏块会不断增加，文件系统必须规避对坏块的操作。垃圾回收即脏块回收，同时还需考虑耗损均衡问题。大部分的 Flash 文件系统都支持日志功能，支持从异常错误恢复的功能[15]。FTL 建立在 MTD 之上，传统的文件系统建立在 FTL 之上[15]。FTL 技术从软件层将 Flash 设备模拟为块设备。FTL 必须提供包括负载平衡、逻辑地址与物理地址的转换、与块设备相同的访问接口等重要特性。FTL 技术最大的挑战在于性能问题，此方面的研究还在发展中[15]。

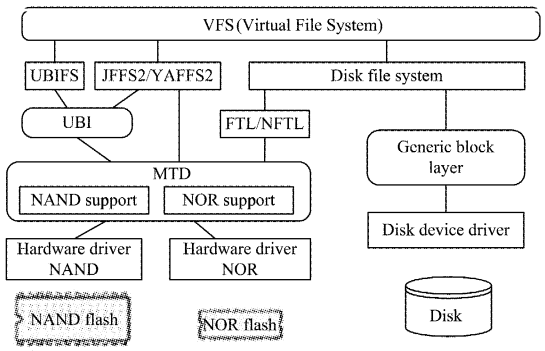

图 2 Linux 存储子系统结构示意图

直接构建在 MTD 层上的文件系统包括 JFFS2，UBIFS，YAFFS2 和 F2FS 等。

JFFS2 文件系统是目前最常用的 FFS(Flash File System)，从 Linux 2.4.10(2001 年)就加入到了内核中。JFFS2 挂载时需要扫描全部区块，以便在内存中建立管理数据视图，其挂载耗时和分区的大小呈线性关系。为了改善这个问题，Ferenc Havasi 提出了在块中增加 summary 信息，以减少挂载时扫描整个分区时间的特性[17]。

UBIFS 是开放设计的新一代 Flash 文件系统，从 Linux 2.6.27(2008 年)开始加入内核主线。UBIFS 增加了 UBI 层，UBI 层提供从物理块到逻辑块的映射功能，实现了坏块管理和耗损平衡，简化了 UBIFS 的开发[4]。为了实现加速文件系统挂载，UBIFS 将管理信息存储在 index nodes 中，因此建立内存管理数据视图，不许进行全区扫描，其挂载时间随 Flash 分区大小的增长缓慢变化[16]。

YAFFS2(Yet Another Flash File System version 2)是为了改进 JFFS/JFFS2 不能支持大容量的 NAND Flash 而设计的。目前 YAFFS2 以 patch 的形式进入内核。相比于 JFFS2，它不支持数据压缩，但其速度快，挂载时间短，内存占用少，跨平台支持(Linux，eCos，WinCE，pSOS ThreadX)。

F2FS(Flash-Friendly File System)主要的应用领域为智能手机的 NAND Flash 存储设备。F2FS 是基于日志结构的文件系统，特别针对 NAND 闪存介质做出了很多改进，使之可以应对之前日志型 Flash 文件系统出现的 Wandering Tree 的滚雪球效应和清理开销大等问题。Wandering Tree 是指当文件数据块被更新时，会引起上层数据结构的递归更新。

嵌入式系统在存储设备选择上倾向于采用读写速度更快的 Flash 芯片。NAND Flash 和 NOR Flash 是经常使用的 Flash 存储设备。NOR Flash 提供了较好的随机读特性，这种特性特别适合将其作为程序的存储设备；NAND Flash 在读写之间做出平衡，容量比 NOR Flash 大，适合普通数据的存储[1]。

由于 NOR Flash 的读性能更好，我们将 U-Boot、内核镜像、根文件系统和设备树描述文件部署在 NOR Flash 存储器上，存储分配方案如表 2 所列。支持 NOR Flash 芯片的文件系统有 JFFS2 和 UBIFS，其中 UBIFS 牺牲存储空间以减少启动时间，保证了文件系统的挂载性能[14]。因此交换机采用 UBIFS 文件系统。在实验中，我们验证了 UBIFS 和 JFFS2 文件系统的挂载性能。

表 2 存储分配方案

Flash 类型	分区名称	大小	存储内容
NOR Flash	BootLoader	512KB	U-Boot
	kernel	4.5MB	Linux
	File system	22MB	JFFS2 或 UBIFS
	DTB	512KB	二进制设备描述树
NAND Flash	辅助存储	128MB	可执行程序与配置信息

3.2 U-Boot 启动延迟优化

U-Boot 在系统启动过程中通过读取设备树文件对硬件进行初始化。选择合理正确的设备树文件能够节约硬件初始化时间。在 U-Boot 的编译配置中，只初始化部分硬件也可减少延迟。U-Boot 的工作模式分为两种：启动加载模式和下载模式。只有系统开发人员需要进入下载模式，便于对系统进行修改、配置，而普通用户不需要进入下载模式，因此我们将 boot delay 设置为 0，以消除等待延迟。

在实际交换机应用中，通过配置编译选项(如 CONFIG_OF_IDE_FIXUP)和修改环境变量(如 bootm)，来避免不必要的硬件初始化和等待延迟。

3.3 内核镜像加载性能优化

内核镜像加载阶段包括：U-Boot 从环境变量 bootm 中获取的值为内核文件在 Flash 芯片上存储的起始位置，将内核加载至内存，解压内核，进行校验，打印内核信息，将系统的控制权交给 Linux 内核。

内核镜像文件可以选择压缩，压缩之后的内核能够节省存储空间，也能够减少内核加载的时间，但是需要耗费时间进行解压，可根据系统 CPU 的性能和存储空间的大小做出均衡。在 PowerPC 环境下，U-Boot 使用特有的 uImage 内核镜像格式引导内核启动。uImage 镜像文件使用 mkimage 工具在 zImage 前加 64 字节的文件头生成。头用于说明内核版本、加载位置、生成时间、大小等信息。zImage 是由内核编译生成的镜像 vmlinux 经过 gzip 压缩，再加入一段解压代码构成的。一般情况下，NOR Flash 芯片不存储未压缩的内核镜像文件(zImage 的大小不到 vmlinux 的 0.5%)。对于 NOR Flash 类型的存储设备，硬件使用 ECC 等算法进行校验，不必使 U-Boot 对内核镜像文件进行校验。因此通过在 NOR Flash 上存储压缩的内核镜像，可以省略内核镜像文件校验步骤，使得内核镜像加载性能优化只需考虑减少内核解压耗时。

减少内核解压耗时主要是通过内核裁剪使镜像文件大小减小来实现[5]。裁剪内核主要包括：移除不需要的硬件驱动、文件系统、网络协议等模块[23]，将不确定的内核功能编译为内核模块，需要时再加载。需要注意，编译为内核模块的文件要存放在文件系统中，并且需要打开内核动态加载模块的功能。

3.4 Linux 内核启动优化

Linux 内核启动主要完成以下任务：使用从 U-Boot 传入的设备树描述结构建立硬件环境，接管整个系统管理权限，运行内存管理服务，初始化文件系统，建立第一个进程，准备响应用户的输入。对于内核启动性能的优化有两种思路：一种是分析内核启动事件，逐项进行优化；另外一种思路为 XIP(eXecute In Place)和内核休眠技术。

逐项优化即分析内核启动中各个事件的耗时，针对事件提出具体的优化策略。其中最有效的方法为内核裁剪，其不仅移除了不需要的内核模块，减少了内核镜像文件的大小，加速了镜像文件的解压速度，同时加速了内核本身的启动。

利用 XIP 和内核休眠技术。使用 XIP 技术能够跳过加

载内核的阶段,但镜像文件必须为非压缩的;内核休眠技术可以跳过初始化硬件等步骤,直接恢复到运行就绪的内核。使用内核休眠技术需要结合 XIP,但是需要加载更大的内核镜像文件,需要更多的存储空间。在数字电视系统中使用了休眠镜像恢复技术,启动时间减少大约了 50%[3]。

在本项目中,受 Flash 芯片大小的限制,不使用 XIP 和内核休眠技术。针对硬件环境和实际需求,我们优化了内核模块,将内核大小从 4.19MB 裁剪到 1.77MB,其大小减少了 57.5%,同时启动耗时减少了 1.57s。另外,针对 proc/sys 挂载优化也是系统启动优化的一个重要方面。proc/sys 文件系统是虚拟文件系统,具有查看和设置内核参数的功能,sys 文件系统还具有统一管理内核设备的功能。一般情况下,嵌入式系统不需要 proc/sys 文件系统,但在本项目中,内核模块需要通过 proc 文件系统将 FPGA 的控制信息导出给简单网络管理协议(Simple Network Management Protocol,SNMP),以便远程管理。

在本项目中,对文件系统的选择只关注其挂载耗时,实际中嵌入式文件系统的选择与应用的场景相关,需从文件系统的规模、是否可写、是否支持异常恢复、I/O 吞吐率等方面进行权衡。

3.5　用户态程序启动性能优化

应用程序包括系统的各种服务进程和用户程序。服务进程启动信息定义在 rcS 中,可以通过消除不必要的延迟、移除不需要的驱动加载等方式进行优化。用户态的应用程序启动速度与其功能相关。如果应用程序是 ELF(Executable and Linkable Format),则可以利用 Perlink,以省去运行时动态链接和加载的时间。其对类似 OpenOffice.org 等重型软件启动加速效果明显[21]。

交换机中,启动脚本的主要任务为挂载 128M NAND Flash 的 JFFS2 文件系统、加载实时交换机核心功能的内核模块,配置 FPGA。经过优化后的整体运行时间约为 0.9s,在可接受范围内。

4　启动优化实验与结果分析

通过对启动过程的分析,提出了基于 U-Boot 优化、内核裁剪、文件系统替换、启动参数优化的优化策略,通过实际应用上的实验验证了优化策略在启动优化中的效果。下面介绍实验环境配置与实验结果。

4.1　实验环境

本项目中的轨道交通实时以太网交换机采用 P2020 开发板和 FPGA (Vertex7 485T)作为硬件平台,使用 U-Boot、Linux 内核和 busybox 构建软件系统平台。如图 3 所示,在机箱内部有 6 块板卡,包括 1 块电源板、1 块 P2020 开发板、1 块 FPGA 交换板、3 块 PHY 板,其中每块 PHY 板提供 8 个 s3mii 接口。软件环境如表 3 所列,硬件环境如表 4 所列。

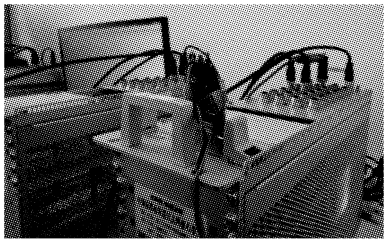

图 3　轨道交通实时以太网交换机实物图

表 3　软件环境

组成部分	软件
Boot Loader	U-Boot 2011.12-rc3
Kernel	Linux 2.6.33.7.2-rt30
文件系统构建	BusyBox 1.21.1
编译环境	Gcc-4.3.2 Glibc-2.8.74

表 4　硬件环境

组成部分	硬件型号
开发板	P2020 RDB
CPU	PowerPC e500v2
内存	512 MB
存储	128MB NOR Flash 128MB NAND Flash

4.2　U-Boot 启动优化

通过消除 U-Boot 等待延迟、取消 PHY 和 IDE 设备的初始化等方法减少了 U-Boot 的启动耗时,使得 U-Boot 启动耗时从原有的 6.13s 减少到 0.86s,如表 5 所列。

表 5　不同配置下 U-Boot 启动总耗时

优化方法	时间/s
原始启动耗时	6.13
关闭等待延迟后	3.13
取消 PHY 初始化后	2.06
取消 IDE 初始化后	0.86

4.3　加载内核镜像优化

为了减少加载内核镜像的耗时,需要对内核镜像的校验和解压耗时进行分析。如表 6 所列,内核的校验、解压耗时和其大小成正比。我们可以从减少内核镜像大小、改进内核解压算法等方面进行优化。

表 6　内核镜像加载耗时对比

事件	加载耗时/s	
	2.9MB 镜像	1.8MB 镜像
镜像文件校验	2.18	1.32
解压	4.38	2.65

4.4　内核启动优化

在对比测试中,分别测试了 2.9MB 内核与 JFFS2 文件系统、1.8MB 内核与 JFFS2 文件系统、1.8MB 内核与 UBIFS 文件系统 3 种组合的系统启动耗时,结果如表 7 所列。对于不同内核大小和文件系统,静默模式[7](quiet mode)耗时均小于默认模式,这是由于减少了日志的输出,同时内核裁剪也减少了内核启动的耗时。

表 7　内核启动模式耗时对比

事件	时间/s		
	2.9MB 内核＋JFFS2	1.8MB 内核＋JFFS2	1.8MB 内核＋UBIFS
正常模式耗时	12.13	10.56	1.48
静默模式耗时	10.20	9.93	0.94
节约时间	1.93	0.63	0.54

如表 8 所列,JFFS2 文件系统挂载时,需要扫描全部区块,以便在内存建立管理数据视图,其耗时与文件系统的大小成正比,JFFS2 with summary[17] 在块中增加了 summary 信息,减少了挂载时扫描全部区块的耗时。UBIFS 将管理信息存储在 Flash 中,挂载耗时与 Flash 的分区的大小无关。不同于 JFFS2 使用表来管理文件索引,UBIFS 文件系统使用的是树,因此 UBIFS 错误恢复的耗时也很小。

表 8 文件系统挂载耗时对比

文件系统类型	耗时/s
JFFS2	9.38
JFFS2 with summary	0.55
UBIFS	0.39
UBIFS recovery	0.42

此外文件系统可以以只读模式或读写模式挂载，实际测量中两者的耗时差距在毫秒级，故表中没有列出。

4.5 实验结果

综合以上所有启动优化策略，我们对优化前与优化后的系统启动过程及其耗时进行了对比。如表 9 所列，通过上述优化策略，系统的启动耗时从 26.69s 减少至 7.15s。

表 9 主要耗时步骤优化实验的结果对比

标志性事件		时间/s	
		优化前	优化后
U-Boot	启动	3.13	0.86
	等待延迟	3.00	0
加载内核镜像	内核镜像校验	2.18	0
	内核镜像解压	4.38	2.65
内核启动	内核初始化	1.62	0.53
	挂载根文件系统	9.38	0.39
	内核启动完成，释放资源	0.11	0.02
	挂载 proc，sys 文件系统	2.00	1.81
用户态程序启动	启动脚本运行	0.29	0.29
	用户空间应用启动	0.60	0.60

结束语 本文从轨道交通核心交换机的需求出发，分析了 Linux 嵌入式系统的启动流程，重点从 U-Boot 启动、内核镜像加载、内核裁剪、根文件系统挂载等方面对系统启动做出了优化。实验表明，通过本文提出的优化策略，可使系统启动耗时明显减少，满足工程需求，对类似系统启动性能的优化也有借鉴意义。下一步将研究不同的内核镜像压缩算法，提出耗时更短、占用存储空间更少的镜像压缩算法；深入分析 proc 和 sys 虚拟文件的挂载过程，改进其挂载初始化算法。

参 考 文 献

[1] OLIVIER P, BOUKHOBZA J, SENN E. Flashmon V2: monitoring raw NAND flash memory I/O requests on embedded Linux [J]. ACM SIGBED Review, 2013, 11(1): 38-43.

[2] LIM G, HWANG J, PARK K, et al. Enhancing init scheme for improving bootup time in mobile devices[C]// Eighth International Conference on Mobile Computing and Ubiquitous Networking. IEEE, 2015: 149-154.

[3] JO H, KIM H, ROH H G, et al. Improving the startup time of digital TV[J]. IEEE Transactions on Consumer Electronics, 2009, 55(2): 721-727.

[4] KANG D. Enhanced UBI layer for fast boot-up times of mobile consumer devices[J]. IEEE Transactions on Consumer Electronics, 2012, 58(2): 450-454.

[5] ASBERG M, NOLTE T, JOKI M, et al. Fast Linux bootup using non-intrusive methods for predictable industrial embedded systems [C]// Emerging Technologies & Factory Automation. IEEE, 2013: 1-8.

[6] LEE D, WON Y. Bootless Boot: Reducing Device Boot Latency with Byte Addressable NVRAM[C]// IEEE International Conference on High PERFORMANCE Computing and Communications. IEEE, 2013: 2014-2021.

[7] BIRD T R. Methods to improve bootup time in Linux[C]// Proceedings of Linux Symposium. 2004: 79-88.

[8] BARBALACE A, LUCHETTA A, MANDUCHI G, et al. Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application[J]. IEEE Transactions on Nuclear Science, 2008, 55(1): 435-439.

[9] IP B. Performance Analysis of VxWorks and RTLinux [OL]. http://core.ac.uk/display/20958343.

[10] Comparison of boot loaders. , 2016 [OL]. http://en.wikipedia.org/wiki/Comparison_of_boot_loaders.

[11] OLIVIER P, BOUKHOBZA J, SENN E. On benchmarking embedded Linux flash file systems[J]. ACM Sigbed Review, 2012, 9(2): 43-47.

[12] HOCHUNG K, SILCHOI M, SEONAHN K. A Study on the Packaging for Fast Boot-up Time in the Embedded Linux[C]// IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. IEEE, 2007: 89-94.

[13] CAI X, SHAO S. An Optimization Algorithm for UBIFS Wear-Leveling[C]// 2010 2nd International Workshop on Intelligent Systems and Applications (ISA). IEEE, 2010: 1-4.

[14] DEY S, DASGUPTA R. Fast Boot User Experience Using Adaptive Storage Partitioning[C]// Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns. IEEE Computer Society, 2009: 113-118.

[15] WANG W, ZHOU D, XIE T. An embedded storage framework abstracting each raw flash device as an MTD[C]// ACM International Systems and Storage Conference. ACM, 2015: 7.

[16] KANG D. Enhanced UBI layer for fast boot-up times of mobile consumer devices[J]. IEEE Transactions on Consumer Electronics, 2012, 58(2): 450-454.

[17] JFFS2 full summary support[OL]. http://www.infradead.org/pipermail/linux-mtd/2004-November/010887.html.

[18] bootchart: Boot Process Performance Visualization[OL]. http://www.bootchart.org.

[19] initng: The Next Generation Init System, 2017 [OL]. http://initng.sourceforge.net/trac.

[20] Upstart Intro, Cookbook and Best Practices, 2017 [OL]. http://upstart.ubuntu.com/cookbook.

[21] JELINEK J. Prelink [OL]. http://people.redhat.com/jakub/prelink.

[22] YAGHMOUR K, 亚荷毛尔, 马斯特斯, 等. Building embedded Linux system[M]. 东南大学出版社, 2009.

[23] 江梦涛, 潘朋飞, 宋杨, 等. Linux 内核中编译选项、文件以及函数之间依赖关系的解析方法[J]. 计算机科学, 2014, 41(s1): 445-450.