基于 Barrelfish 的实时 libOS 设计与实现*

陈 龙,龙 翔,王 雷
(北京航空航天大学 计算机学院,北京 100191)

摘 要:将 libOS 概念应用到对多核系统 Barrelfish 的实时性改造上,构建一个同时满足多核实时性要求的系统框架,并且通过实验验证了这种思路的可行性。

关键词:众核;实时性;libOS;操作系统
中图分类号:TP316.2 **文献标识码**:A **文章编号**:1674-7720(2013)02-0090-03

Design and implementation of a real-time libOS based on Barrelfish

Chen Long, Long Xiang, Wang Lei
(School of Computer, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: A real-time operating system can be considered as a run time on many-core operating system to satisfy the requirement of real-time and many-core structure at the same time. This design trys to implement a real-time architecture that takes Barrelfish as the bottom many-core operating system and VxWorks as libOS.

Key words: many-core; real-time; libOS; operating system

随着当前系统结构中处理器核心数目的增加,传统的微内核和单一内核结构已经无法满足众核结构扩展性要求,因此,业界提出了如 FOS、Barrelfish 等满足多核要求的操作系统原型。但是这些系统原型都没有实时性方面的考虑。而今,在国防科技、航空航天、工业控制、电信行业、医疗行业、信息家电等多个领域都同时提出对多核、实时性的需求。

为了满足这种需求,本文基于当前现有多核操作系统和实时操作系统,构建一种同时满足多核和实时性要求的系统框架。

1 系统介绍

1.1 Barrelfish 操作系统

Barrelfish 的核心思想在于 cpu driver 的概念,为一部分体系结构相关代码,每个核心上都运行单个 cpu driver,它们之间不共享任何状态,不做任何形式的同步,结构上更像是一个分布式系统。与传统操作系统结构相比,Barrelfish 系统可以保证很好的扩展性[1]。

Barrelfish 采用 dispatcher 和 thread 双层调度模型,不支持抢占,使用轮转调度策略。dispatcher 相当于进程,每个 dispatcher 可以有许多 thread,这种调度模型奠定了整个系统框架的基础。本文以此操作系统作为底层多核结构的支撑。

1.2 VxWorks 以及 libOS

libOS 是 MIT 在 exokernel 操作系统中提出的[1],libOS 将一个操作系统作为应用运行在下层 kernel 上,kernel 上可以运行多个 libOS,应用开发者可以完全自由地使用它们,甚至可以将它们替换成自己的抽象以达到提高性能的目的。底层的硬件访问允许程序员实现一个自定义的抽象,去掉不需要的,这通常可以大大地提高程序的性能。本文所要构建的系统框架采用了 libOS 的概念,这也是本文的主要思想之一。

本文采用 VxWorks 5.5 构建上层 libOS。VxWorks5.5 与其他版本的系统对比具有系统结构简单、易于裁剪和分析等优势,采用基于抢占的优先级调度,辅以同优先级的时间片轮转调度,具有 256 个优先级供任务使用,VxWorks5.5 不区分用户态和内核态,所以没有状态切换的性能损耗,更有利于分析其实时性能。

2 系统构建过程

本文构建的系统框架如图 1 所示。

libOS 的构建过程主要包括对 Barrelfish 的结构分析与改造、对 VxWorks5.5 的结构分析与改造(包括启动过程分析与裁剪、系统内核任务管理以及内存管理代码的

* 基金项目:国家高技术研究发展计划资助项目(863 计划)(2011AA01A204)

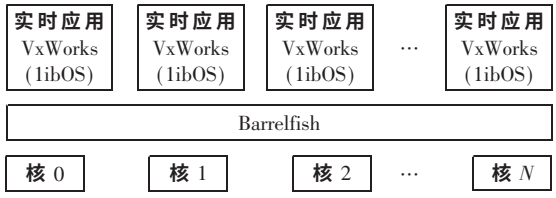

图 1 系统框架

移植、内核底层结构相关汇编代码部分的改造)及添加实时任务等。

2.1 启动分析及改造

Barrelfish 的用户程序为常见的 ELF 格式，加载后的 ELF 以 dispatcher 的方式运行，dispatcher 可以包含许多 thread，在调用用户 main 函数之前，libbarrelfish 完成对 dispatcher、thread 初始化等操作，如图 2 所示。其中 barrelfish_init_disabled () 为 libbarrelfish 的入口，这里设置 dispatcher 的入口为 run_entry，即 dispatcher 被调度执行的入口，thread_init_disabled () 为线程初始化，这个线程最终会执行用户 main() 函数。

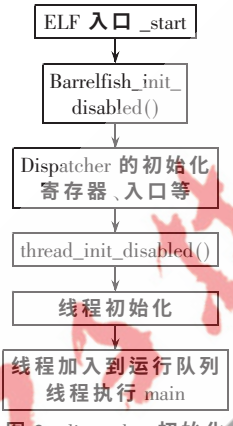

图 2 dispatcher 初始化

dispatcher 调度是 Barrelfish 最基本的部分。改造后的 dispatcher 初始化过程如图 3 所示。VxWorks 启动包括两个方面：BSP 启动和 VxWorks 系统镜像启动。改造 VxWorks 的启动流程，裁剪掉不相关代码，保留任务管理、存储管理等核心代码，将两个启动阶段合并为一个，改造之后的 libOS 启动过程如图 4 所示。

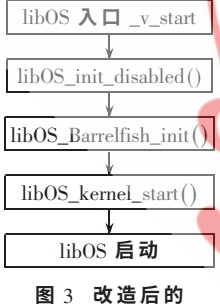

图 3 改造后的 dispatcher 启动

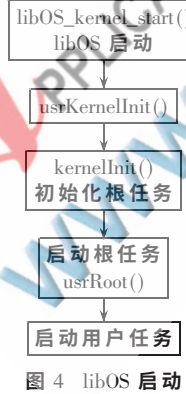

图 4 libOS 启动

2.2 中断处理

libOS 构建过程中需要解决的一个关键问题是 libOS 如何做中断处理，本文中 libOS 作为 dispather 运行在 Barrelfish 的用户空间，而 Barrelfish 的中断处理发生在内核态，上层无法接收中断，这将导致移植后的 libOS 无法保证抢占式调度策略。此外，在接收不到时钟中断的情况下，libOS 也无法做任务挂起、定时等一系列基于时钟的操作。

为了解决这个问题，可以从以下几个方面考虑：(1) dispatcher 响应时钟中断；(2)Barrelfish 判断中断类型；(3) libOS 处理中断流程。

除了由时钟中断处理程序调用 dispatch() 外，其他情况(如一个 dispatcher 退出)也可能引起 dispatch() 被调用。想要只有时钟中断被 libOS 感知，就要对这些情况进行区分，在 Barrelfish 的 kernel/dispatch.c 中增加一个全局变量：

```
enum dispatch_type dispatch_type =DISPATCH_TYPE_DEFAULT;
```

每当开始调度一个 dispatcher 时，根据调度的理由设置 dispatch_type 的值。若发生时钟中断，则在 Barrelfish 的中断处理程序中将其设置为：

```
dispatch_type=DISPATCH_TYPE_TIMER_INTERRUPT;
```

libOS_run 则是重新定义的 libOS 的 disp_run 的 c 函数入口，其函数原型为：

```
void libos_run(dispatcher_handle_t handle, enum dispatch_type type);
```

第二个参数指定了调度 libOS 的理由，当其值为 DISPATCH_TYPE_TIMER_INTERRUPT 时，说明是由于在 libOS 的时间片内发生了时钟中断，可以进行相应的时钟中断处理。整个流程如图 5、图 6 所示。

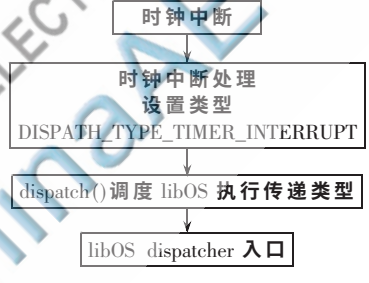

图 5 Barrelfish 时钟中断响应

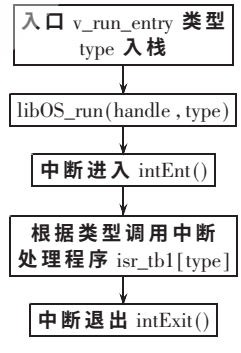

图 6 libOS 时钟中断处理

2.3 任务切换部分的改造

VxWorks 上下文切换分两种情况：同步和异步[4]。VxWorks 的这部分代码由汇编指令写成，其中涉及到一部分特权指令，这些指令无法在用户态执行。此外针对原来中断处理的部分也要进行改造，改造后的流程如图 7 所示。

3 实验

3.1 测试平台

本文的测试平台为 qemu 模拟器，qemu 对 x86 平台有很好的支持，通过模拟 4 个 x86 核心来启动 Barrelfish。

3.2 测试流程

测试主要过程为：将用户程序 usr_test() 添加到 usrRoot() 根任务中启动，使用 taskSpawn() 创建 3 个不同优先级的任务，任务见表 1。

这个流程能够更好地体现系统基于抢占的优先级调度策略。task3 优先级最低，被 task2 抢占，task2 被

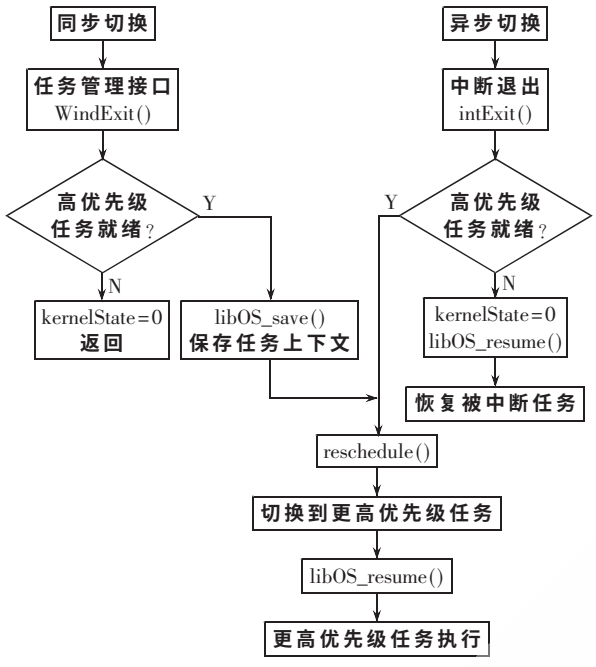

图 7 任务切换流程

表 1 实时任务

	名称	优先级	入口
任务 1	task1	101	taskone
任务 2	task2	102	tasktwo
任务 3	task3	103	taskthree

task1 抢占，最后 task3 改变自己的优先级为最高又抢占 task1。

3.3 测试结果

将生成的 libOS 镜像添加到 grub 的 menu.lst 中，通过 module 参数指定其目标核的编号，启动 Barrelfish 后会自动将其加载到目标核上启动。

本文通过将 libOS 概念应用到对多核系统 Barrelfish 的实时性改造上，构建一个同时满足多核实时要求的系统框架，并且通过实验验证了这种思路的可行性。本工作保证 libOS 在其时间片内的实时调度，而底层 Barrelfish 没有实时性方面的考虑，可以从分析 libOS 的中断延迟、中断响应时间、任务切换时间等方面着手，与移植之前的相应参数做比较，提出对整个系统框架实时性改进的意见。

参考文献

[1] BAUMANN A, BARHAM P, HARRIS T. The multikernel: a new OS architecture for scalable multicore systems. Systems Group[C]. ETH Zurich, 2009.

[2] ENGLER D R. The exokernel operating system architecture [R]. MIT USA, 1998.

[3] BAUMANN A, PETER S, SINGHANIA A. Your computer is already a distributed system, why isn't your OS[C]. 12th Workshop on Hot Topics in Operating Systems, 2009.

[4] 官赐田. 一个高性能操作系统 vxworks 的移植及其实现[D]. 上海: 上海交通大学, 2006.

[5] BAUMANN A, PETER S. Embracing diversity in the Barrelfish manycore operating system[C]. Systems Group, ETH Zurich, 2008.

(收稿日期: 2012-09-18)

作者简介:

陈龙, 男, 1988 年生, 硕士在读, 主要研究方向: 操作系统。

龙翔, 男, 1963 年生, 教授, 主要研究方向: 操作系统。

王雷, 男, 1969 年生, 副教授, 主要研究方向: 操作系统。