容错实时系统的内存管理优化方案及实现

黎忠文[1,2], 郑建仙[1], 罗仁泽[2]

(1. 厦门大学 信息科学与技术学院, 福建 厦门 361005; 2. 电子科技大学 中山学院, 广东 中山 528402)

摘 要: 探讨如何把抢占门限应用于具有优先级提升和内存受限的实时系统, 并进一步研究了互不抢占组内, linux平台上基于增量检测点的内存备份/恢复的方法及原型系统的实现。原型系统采用了可扩展的链接方式作为检测点文件格式, 使得数据更加紧凑有序; 在内存的备份操作上, 提出了"影子脏位"算法来识别两检测点之间有无被改变的部分。原型实验验证了所提出方法的有效性。

关键词: 抢占门限; 内存管理; EtoC; 增量检测点

中图分类号: TP311 文献标识码: A 文章编号: 1671-654X(2007)03-0063-03

引言

实时性、容错性和内存受限性是嵌入式操作系统应具有的主要特点。设置检查点和回卷恢复 CRR (checkpointing and rollback recovery)是实时系统中常用的一种容错技术。目前检查点回卷技术已经在 Unix、Windows系统得到相当研究[1,2]。近几年来随着 Linux系统趋向完善, 对 Linux平台上检测点机制的研究也越来越多, 其代表有 UCLTP[3]。对于内存受限性, 业界人士已经开始探讨新的任务堆栈优化方法[4,5], 其中 EtoC[4]模型 (Execute to Completion, EtoC)是一种解决思路。在这种新模型中, 每个任务一旦开始执行, 则一直运行, 直到完成, 显然共享同一堆栈的各任务应互不抢占。考虑到 EtoC的调度具有减少上下文切换、降低内存空间的需求等显著优点, 本文在探讨优先级可提升的系统中, 基于抢占门限的任务堆栈优化新方法以及非抢占给内实施增量检测点的内存备份与恢复技术等的技术问题。本文设任务集 Ψ 是 n 个任务集合, $\Psi=\{\tau_1, \tau_2, \ldots, \tau_n\}$, 每个任务 τ_i 可用五元组 $\langle C_i, D_i, T_i, p_i, \delta_i\rangle$表示。其中 T_i 为任务 τ_i 两个实例间最小间隔时间, τ_i 可以是周期和非周期任务; C_i、D_i、p_i 和 δ_i 分别为任务 τ_i 的执行时间、死限、优先级和抢占门限, 且满足 $C_i \leqslant D_i \leqslant T_i$, $p_i \leqslant \delta_i$, $i=1, 2, \ldots, n$。对于每一个任务, τ_i 采用基于检测点的容错策略, 即当 τ_i 出错后由回卷到最近的检测点重新执行, 这时赋予 τ_i 更高的优先级 $\bar{p}_i$, $\bar{p}_i > p_i$。此外为了便于说明了, 本文把这时的 τ_i 记为 $\bar{\tau}_i$, 称为替代任务。

1 基于互不抢占组的内存划分

对于 $\forall \tau_i, \tau_k \in \Psi$, 若存在 $(p_i \leqslant \delta_k) \wedge (p_k \leqslant \delta_i)$且整个任务集可调度, 或者 τ_i, τ_k 的执行有严格的先后顺序 (比如 τ_k 的执行时间必须在 τ_i 结束运行以后), 则称 τ_i, τ_k 为互不抢占任务, 可分配到同一互不抢占任务组内, 共享同一堆栈。

1)互不抢占组的构造。以(主任务, 替代任务)为单位进行互不抢占任务组的构造。对于 $(\tau_i, \bar{\tau}_i)$, 令其优先级 $\bar{\bar{p}}_i = \bar{p}_i$, 抢占门限 $\bar{\bar{\delta}}_i = \delta_i$。设 Q 为各任务单元按抢占门限的值依非递减序排成的队列。取 Q 的第一个元素 $(\tau_i, \bar{\tau}_i)$设为第一个互不抢占分组 G_1, 然后依次遍历 Q 的元素。如果 Q 中存在一个元素 $(\tau_k, \bar{\tau}_k)$满足: $\forall (\tau_i, \bar{\tau}_i) \in G_1$, 都有 $\bar{\bar{p}}_k \leqslant \bar{\bar{\delta}}_i$, 则 $(\tau_i, \bar{\tau}_i)$与 $(\tau_k, \bar{\tau}_k)$互不抢占。这是因为 Q 按任务的抢占门限非递减次序排列, 即满足 $\bar{\bar{\delta}}_i \leqslant \bar{\bar{\delta}}_k$, 又因 $\bar{\bar{p}}_i \leqslant \bar{\bar{\delta}}_i$, 所以必有 $\bar{\bar{p}}_i \leqslant \bar{\bar{\delta}}_k$ 成立, 故对 $(\tau_i, \bar{\tau}_i)$与 $(\tau_i, \bar{\tau}_k)$有 $\bar{\bar{p}}_k \leqslant \bar{\bar{\delta}}_i$ 和 $\bar{\bar{p}}_i \leqslant \bar{\bar{\delta}}_k$ 同时成立, 也即 $(\tau_i, \bar{\tau}_i)$与 $(\tau_i, \bar{\tau}_k)$为互不抢占任务元素, 于是把 $(\tau_k, \bar{\tau}_k)$加入 G_1。一旦遍历完 Q 中的所有任务元素, 则产生第一个互不抢占任务组 G_1。重复上面的过程, 直到 Q 为空, 这时多个互不抢占任务组构造完毕。

2)抢占门限的计算。$\tau_i/\bar{\tau}_i$ 抢占门限的计算算法如 preemption-threshold($\tau_i/\bar{\tau}_i$, $p_i/\bar{p}_i$)所示:

收稿日期: 2007-01-29 修订日期: 2007-03-14

基金项目: 广东省自然科学基金 (06029667)

作者简介: 黎忠文(1970-), 女, 重庆江津人, 博士, 研究方向为嵌入式实时系统高安全和高可靠技术。

Procedure preemption- thresholdi($\tau_i / \overline{\tau}_i$, $p_i / \overline{p}_i$)

10　{ $\delta_i = p_i$ or $\overline{\delta}_i = \overline{p}_i$;

20　ocu(i) = { τ_j, τ_k | $p_j > \delta_i$, $\overline{p}_k > \delta$; j, k∈ (1, …, n), j≠ i, k≠ i};

30　B= min($\overline{p}_k$ | $\overline{\tau}_i$∈ ocu(i)) - 1;

40　while (schedulable(Ψ) = = TURE) and ($\delta_i / \overline{\delta}_i$ < B) do /* schedulable(Ψ)计算 Ψ的可调度性* /

50　{ δ_i+ +; or $\overline{\delta}_i$+ +; }

60　if ($\delta_i / \overline{\delta}_i \geqslant$B)

70　then{ δ_i- -; or $\overline{\delta}_i$- -; }

80　B= δ_i; or B= $\overline{\delta}_i$

90　return(B) }

算法第 30行,从保持替代任务抢占序列的角度,计算主 /替代任务的理论最高抢占门限;第 40和 50行逐步调高任务的可行抢占门限;第 90行返回分配给任务的最高抢占门限。把主 /替代任务按抢占门限非降的偏序排列,依次调用 produce preemption- thresholdi($\tau_i / \overline{\tau}_i$, $p_i / \overline{p}_i$),就可计算出各任务的最高抢占门限。

2　内存的备份与恢复

本文采用链接表映射技术,使用虚拟指针把链表映射到平面线性文件中,实现各个独立模块间的前后无缝衔接。页表项的低第 7位为脏页位,表示该页表项对应的页框在近段时间是否已被写过。这种机制由 i386处理器的 MMU单元硬件实现,但 MMU从来不重置这个标志,重置操作由内核完成。由于容错系统执行检测点操作不能和内核冲刷“脏”物理内存的操作同步,也就不能准确跟踪到某个物理页在两个检测点之间有无改变。为此本文使用脏位影子算法解决这个问题。具体做法是在内核冲刷“脏”物理内存操作中添加一个行为,即在重置脏页位的同时,也设置影子位,以表明影子位对应的物理页在上次检点以来被执行写操作,所以在下次检测点操作时就要将该物理页备份起来,同时也要重置影子位以表明该物理页已经是“干净”的。算法如下:

1)遍历线性地址区链表,备份每个线性区中的相关信息。

2)从进程的 pgd值找到页目录。

3)遍历页目录的每个项,对每一个页目录项执第 4)步操作。

4)判断该页目录项中的 XW标志,如果该页目录项对应只读的物理页,则跳到第 3)步;否则跳到第 5)步。

5)以页目录项的高 20位为地址 addr的高 20位, addr的低 12位全为 0,遍历地址 addr指向的页表,对每一个页表项执行第 6)步操作。

6)判断该页表项中的 XW标志,如果该页表项对应只读的物理页,则跳到第 5)步;否则跳到第 7)步。

7)找到页表项中脏页位的影子位,如果影子位为 1,则备份该页表项对应的物理页,并且重置影子位,再跳到第 6)步;如果影子位为 0,则直接跳到第 6)步。

在此基础上为了实现内存的备份与恢复,我们为容错系统申请一个类型缓冲区,其数据结构类型为 ft_mm_t,具体定义如下:

```
struct ft_mm_t {
int   ft_nr_vma;
unsigned long   start_code, end_code;  /*代码段起止地址
unsigned long   start_data, end_data;  //数据段起止地址
unsigned long   start_brk, brk;  //堆段起止地址
unsigned long   start_stack;
unsigned long   arg_start, arg_end;  //参数段起止地址
unsigned long   env_start, env_end;  /*环境变量起止地址
};
```

容错机制先把进程内存分布数据备份到缓冲区之后,再根据上述的脏位影子算法把进程的内存数据把备份起来。反之在恢复进程时,从检测点文件中读出来恢复到目标进程中。Linux中每个进程描述符中都有一个 thread字段,其类型为 struct thread_struct,它保存着进程切换时刻的硬件文境。对单处理器结构系统而言,在检测点操作过程中,容错机制占用了 CPU资源,其它进程都没有处于运行状态,其退出占用 CPU那个瞬间的硬件文境都保存在 thread中,因为保存其内容的操作比较简单: write(chkpt, (char*) ptask- > thread, sizeof(struct thread_struct)); 在容错系统恢复进程时,它先构造一个完全可运行而且进程状态与最后一个检测点时刻完全一致的进程后,才把它放入就绪进程队列,供内核调度。在新进程完全构造好之前,它没有任何的运行机会,因此恢复进程硬件文境的操作也比较简单: Read(chkpt, (char*) ptask- > thread, sizeof(struct thread_struct))。

3　原型系统

我们在联锁机子系统[6]上实现容错机制。首先进入放置容错系统 ftmod的目录,运行“make install”命令来安装容错系统, ftmod会先在 proc文件系统下创

建子目录／pro／ftmod作为用户界面。为了安全起见,这里只有打开这个开关才能进入用户界面进行操作。输入命令“echo on > switch”并运行它,出现 ftmod中所有的功能通道。运行命令“cat status”可以发现,目前容错系统还没有目标进程,其默认的检测点周期为 10秒,默认的检测点文件路径为／root／chkpoint,我们可以通过 echo命令来修改容错系统的一些参数,以指导检测点操作行为。

现在开始运行联锁机子系统 iis,它是本实验的目标进程。我们又通过命令“ps - A | grep‘ iis’”来寻找目标进程 iis的进程号。容错系统通过进程号来识别目标进程,如图 1中所示, ftmod通过遍历内核中进程列表发现联锁机子系统进程的路径全名为／home／zjx／last／iis,我们输入指令“echo backup> action”来命令容错系统立即运行检测点操作线程。

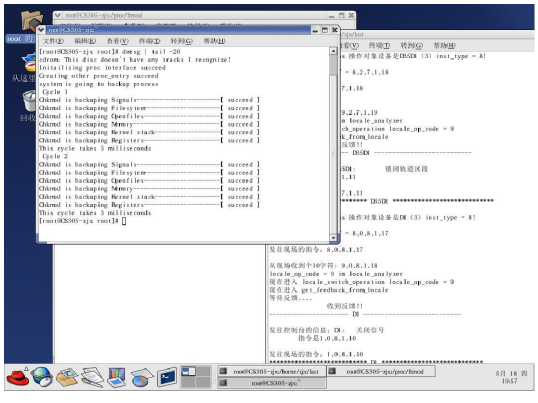

图 1　容错系统对联锁机子系统执行检测点操作图

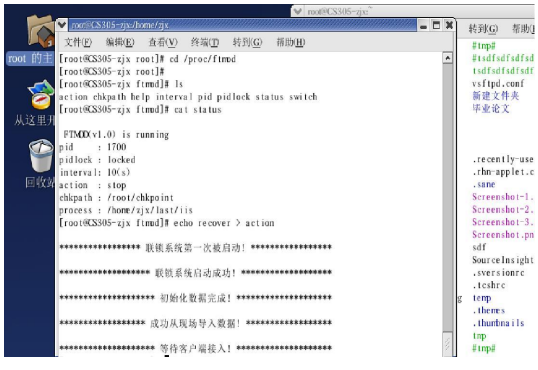

图 2　容错系统重启联锁机子系统图

检测点操作线程一旦被启动,它就会周期性地对目标进程执行检测点操作,直到目标进程运行结束,或者用户通过命令“echo pause”来中止线程,或者通过命令“echo stop”来取消线程。由于容错系统是运行在内核态的,它所打印的全部信息都被送至系统日志缓冲区,而不能直接显示在终端上,只能通过 dmesg命令来读取,我们用命令“dmesg | tail - 20”来读取最后 20行的系统日志,如图 1所示。

在目标进程出现错误时,进程的计算可能会因此丢失,只要保存完整的检测点文件 (默认是／root／chkpoint),我们就可以恢复目标进程到 Linux系统上。在终端上输入命令“echo recover> action”让容错系统重启联锁机子系统,如图 2所示。

4　结束语

本文讨论了在 EtoC任务执行模式下,替代任务优先级高于主任务优先级这种容错方式的安全关键系统的堆栈管理优化方案。并进一步研究互不抢占组所在内存的备份和恢复机制。由于本容错系统只是简单地在每次检测点操作之后将其内容冲刷到磁盘上去,这种处理方式在很多时候是简单而且有效的。但是当两个检测点之间的时间间隔远小于内核线程 kupdate的两次运行时间间隔时,这种处理方式虽然有效但并不高效,因为每次冲刷文件内容都要付出一定的系统开销,所以频繁的冲刷操作必然会导致操作系统的吞吐量降低。我们下一步将致力于这个问题的解决。

参考文献:

[1] 李凯原,杨孝宗. 提高用任务重复的检查点方案的性能[J]. 电子学报, 2000, 28(5): 33- 35.

[2] 刘少锋,汪东升,朱晶. 基于虚拟文件操作的文件检查点设置[J]. 软件学报, 2002, (13): 28- 37.

[3] William R. Dieter, James E. Lumpp. User- level Checkpointing for Linux Threads Programs[EB]. Technical report, http://www.dcs.uky.edu/~chkpt

[4] 杨仕平,熊光泽,陈慧. 面向对象实时多任务系统的优化实现模型[J]. 计算机工程与科学, 2003, 25(5): 56- 59.

[5] Saksena M and Wang Y. Scalable Real- time System Design Using Preemption Thresholds[C]. Proceedings of Real-Time Systems Symposium, Florida, USA, IEEE Computer Press, 27- 30 Nov. 2000: 25- 34.

[6] 郑建仙,黎忠文,陈亮. 防危核技术在铁路微机联锁系统中的应用研究[J]. 铁路计算机应用, 2005, 14(6): 13- 15.

(下转第 82页)

金学院学报, 2003, 24(1): 70- 74.
[2] 刘玉珍, 涂航, 张焕国, 等. 实用智能卡操作系统的设计与实现[J]. 武汉大学学报(自然科学版), 2000, 46(3): 309- 312.
[3] STALLINGS W. Cryptography and Network Security Principles and Practice Second Edition[M]. Prectice Hall Inc, 1999.
[4] 谢冬青. PKI原理与技术[M]. 北京: 清华大学出版社, 2004.
[5] 邓子云. 数字信封技术及其应用研究[J]. 华北水利水电学院学报, 2006, 27(1): 77- 79.

Application of Digital Envelop in Smart Card Symmetry Key Transmission

LIU Xian-bo, WANG Zhao-shun

(*Beijing Information and Engineering School, University of Science and Technology, Beijing* 100083, *China*)

Abstract Going with transference of smart card EMV, using smart card trade is becoming more and more frequent. In the paper the high secret demands of different data interchanges are put forward for application systems, especially security of Symmetry Key input and output. The application of Digital Envelop structure based on RSA gives good a solution to the security problem of Symmetry Key, which can well prevent Key from filching during inputing and outputting them effectively.

Key words smart card; RSA; digital envelop; symmetry key

(上接第 65页)

An Optimal Scheme for Memory Management in Fault-tolerant Real-time System

LI Zhong-wen[1,2], ZHENG Jian-xian[1], LUO Ren-ze[2]

(1. *Computer and Information Engineering College, Xiamen University, Xiamen* 361005, *China*;
2. *Zhongshan Institute, UEST of China, Zhongshan* 528402, *China*)

Abstract After analyzing how to apply preemption- threshold- scheduling into real- time systems among which the recoveries of safety tasks may be executed at higher priority levels and the sizes of memories are limited, we research realization methods about memory backup/recovery on the Linux platform based on increment checkpoints for no-preemption task groups. At last, taking the microcomputer interlocking simulation system as an example, prototype experiment of memory backup/recovery has been done on the Linux platform in the lab. Our prototype experiment takes an extendable checkpoint file layout to make data in a file compact. And with regard of memory backup operation, by using a "showdown of dirty-bit" algorithm to identify the difference between two checkpoints, it implements an increment checkpoint to improve system performance. This experiment has proved the validity of proposed methods.

Key words preemption threshold; memory management; execute to completion; increment checkpoint