基于Linux内核的进程检查点系统设计与实现

门朝光 焦 亮 李 香 徐振朋
(哈尔滨工程大学计算机学院高可信计算技术研究中心 哈尔滨 150001)

摘 要 作为一种流行的软件容错机制,检查点与恢复技术的实现模式有两种:用户级和系统级。首先阐述了两者的区别,然后根据Linux可加载内核模块机制提出了一种基于Linux内核的进程检查点与恢复实现方法。利用Linux内核线程实现了检查点与恢复内核模块,并基于此内核模块在用户层构造了一检查点函数库,为用户提供了相应接口。用户通过组合使用这些接口可以高效地实现具体检查点与恢复算法。

关键词 检查点与恢复,用户级,系统级,内核模块,内核线程
中图法分类号 TP302.8 **文献标识码** A

Design and Implementation of Process Checkpointing System Based on Linux Kernel

MEN Chao-guang JIAO Liang LI Xiang XU Zhen-peng
(R&D Center of High dependability Computing Technology, Harbin Engineering University, Harbin 150001, China)

Abstract As a popular software fault-tolerant mechanism, checkpoint and recovery technique can be implemented by two modes: user-level and system-level. First, the differences between the two modes were discussed. Then according to the Linux LKM (Loadable Kernel Module) mechanism, a method was proposed to design a process checkpoint and recovery system based on the Linux kernel. Checkpoint and recovery kernel module was implemented using the Linux kernel thread. Based on this kernel module, a checkpoint library was constructed in the user-level to provide corresponding interfaces for users. By using some selected interfaces, the particular checkpoint and recovery algorithm can be implemented effectively.

Keywords Checkpoint and recovery, User level, System level, Kernel module, Kernel thread

为了保证大型计算程序高可靠地运行,提高计算机资源的利用率,容错技术应运而生。检查点机制是一种典型的软件容错技术,一方面它具有备份进程状态、生成检查点文件的功能;另一方面当操作系统出现故障或因外部差错而导致某些进程终止时,通过加载检查点文件,可以将进程卷回到最近的检查点处以实现进程状态的恢复[1-5]。所以,一个基本的检查点系统应该包括进程状态保存和进程状态恢复功能。

检查点系统有两种设计模式:用户级和系统级[6,7]。基于用户级模式实现的系统通常具有较好的可移植性,但是缺乏透明性、通用性,用户需要根据不同的平台有针对性地修改应用程序源代码,然后重新编译。如果检查点函数以库函数的形式提供,还需要重新链接。修改源代码容易出错而且很多商业代码只拥有二进制可执行文件,对它们重新修改、编译是不可能的。此外,根据Unix/Linux进程概念可知,用户进程只能通过系统调用的方式来间接访问操作系统内核维护的数据,但是无法访问某些与进程状态有关的内核数据结构。因此用户级的检查点不可能完全地记录和恢复一个进程的状态,这样当系统出现故障或进程出错需要卷回恢复时就可能会导致进程无法恢复。具有代表性的用户级检查点系统有Conder[8]和Libckpt[9]。

基于系统级模式实现的检查点系统具有全透明性,检查点机制独立于应用程序[10,11]。用户不必修改程序源代码,应用程序不必和检查点函数库一起编译,这就减少了程序出错的概率。此外,在操作系统内核空间中,与进程状态相关的任何数据结构都是可访问的。这些数据结构包括寄存器、内存区、文件描述符、信号状态等等。因此可以实现对用户进程状态的完全保存。具有代表性的系统级检查点系统有BLCR[12]和CRAK[13]。

在分析参考了以上几种系统的实现后,本文根据Linux可加载内核模块(LKM)机制提出了一种基于Linux内核的进程检查点技术实现方法。在内核层利用Linux线程实现一个可动态加载的检查点与恢复模块,此模块作为内核线程来运行,实现基本的检查点与恢复功能。在用户层基于该内核模块构造了一个检查点函数库,提供给用户直观便利和丰富的检查点接口,便于用户通过组合使用这些接口来实现具体的检查点与恢复算法。

到稿日期:2008-11-20 本文受国家自然科学基金(60873138)资助。
门朝光(1963-),男,博士,教授,博士生导师,CCF高级会员,主要研究方向为分布式系统与并行系统、可信性计算、移动计算技术,E-mail:menchaoguang@hrbeu.edu.cn;焦 亮(1982-),男,硕士研究生,研究方向为并行系统容错技术;李 香(1975-),女,博士,主要研究方向为可信计算;徐振朋(1983-),男,博士研究生,研究方向为可信计算。

1 系统设计

1.1 基本设计思路

此系统的核心部分是位于 Linux 内核层的可动态加载和移除的内核模块。利用此模块，我们不必对 Linux 内核源代码进行修改，克服了修改内核源代码所带来的调试、编译以及维护困难。此模块加载到内核以后，便与内核一起以相同的优先级运行，并与内核协作完成进程检查点设置和卷回恢复功能。此内核模块可设置的检查点包括进程基本状态信息检查点、内存检查点和文件检查点。

上述的 Linux 内核模块实现了系统的基本功能部分，仅仅利用此模块就可以对用户进程进行检查点设置和恢复操作。为了使系统具有灵活性和良好的可扩充性，我们在用户层构造检查点函数库。此函数库是基于内核模块实现的，即通过对内核模块的函数调用来构造库函数，目的是丰富和扩充内核模块功能，给用户提供直观便利和丰富的检查点接口，使用户能够通过组合使用这些接口来实现具体的检查点与恢复算法，进行算法性能评估，从而选择合适的策略来降低检查点开销、优化检查点功能。如果现有的接口不能满足用户需求，我们还可以及时对库函数进行扩充来满足用户的需要。

通过 Linux 内核中的 proc 文件系统、内核模块与用户进程进行信息交互。proc 文件系统具有灵活、方便的访问接口：以普通的文件读写方式即可访问到任何一个权限许可范围内的进程上下文信息。在 proc 文件系统中，每个进程都拥有一个目录，该目录以进程标识符 PID 命名，目录下有操作系统内核为本进程维护的许多状态信息，如进程内存使用状态、内存内容、虚存的逻辑段信息、打开的文件、命令行参数、进程环境信息等。proc 文件系统中不同的文件包含有不同的内核信息，比较重要的有：记录 cpu 信息的 cpuinfo 文件；记录内核锁信息的 locks 文件；记录内存信息的 meminfo 文件；全面统计状态信息的 stat 文件；记录系统识别的分区表信息的 partitions 文件，等等。内核模块通过对 proc 文件系统中与用户进程相关的文件进行操作可以记录用户进程的状态信息。

1.2 系统原型图及系统工作原理

根据以上的设计思路，我们构造出了基于 Linux 内核级的进程检查点系统结构原型，如图 1 所示。

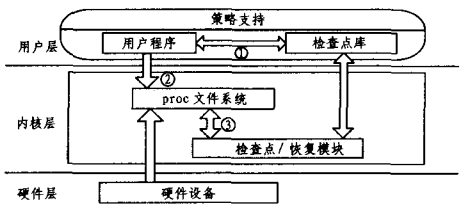

图 1 检查点系统结构图

对一个局部节点上的用户进程设置检查点时，进程标识符 PID 和进程所在节点的处理器 ID 通过 proc 文件系统接口传递给内核模块。内核线程对用户进程采取检查点时，它根据 PID 找到相应的进程，重构进程上下文：进程打开了什么文件，进程占有或释放了多少内存区域等等。由于内核线程最初就对进程相关的地址空间段写保护，因此当应用程序试图向这些页面读写数据时，这些操作就被检测到并被记录下来作为检查点文件内容的一部分。通过 proc 文件系统，此模块还提供一个简单的接口供用户使用，用户可动态地对进程设置检查点或者重启进程。

根据图 1 所示，经过步骤②③，可以实现基本的检查点功能。用户程序运行时，产生一个进程标识符 PID，相应地在 proc 文件系统中产生一个以该进程号命名的文件目录，该目录下包含进程的各种信息，如进程的当前工作目录、进程定义的环境变量、进程打开的文件描述符、进程的地址空间映射文件等，检查点与恢复内核模块可以通过读取、汇总这些信息来保存进程状态，生成检查点文件。

经过步骤①②③，可实现在用户具体策略支持下的程序检查点与恢复功能。这里所说的策略通常指用户新提出的检查点与恢复算法，或是对现有算法的改进，应用此检查点系统平台来实现并测试算法性能。用户可以组合使用检查点库中的函数来实现算法，执行的时候将用户程序加载进来与算法结合在一起运行，然后与步骤②③一样，内核模块对用户进程设置检查点。

无论是经过步骤②③还是经过步骤①②③来对用户进程设置检查点，对于用户来说，这些操作都是透明的，既不需要修改用户程序，也不需要对用户程序和检查点库一起进行编译链接。

2 系统工作流程

2.1 检查点设置过程

内核线程进行如下操作来对一个用户进程采取检查点：

(1) 阻塞用户进程运行；

(2) 跳转到用户进程地址空间；

(3) 保存用户进程映像：保存数据段，包括 3 个区域：初始化数据区、未初始化数据区、堆；保存相关内存段。

(4) 保存进程用户栈：保存栈指针；保存栈空间。

(5) 保存与进程上下文切换有关的项(包括程序计数器 PC、处理机状态字寄存器 PSW 等)；

(6) 保存用户进程打开的文件描述符；

(7) 保存信号状态(包括即将被用户进程接收的信号)和由用户进程定义的信号处理函数；

(8) 启动用户进程运行。

在用户进程映像的保存中，进程正文段(即代码段)通常不需要保存，程序启动时装入程序会对它进行恢复。正文大小一般是不变的，通常在地址空间的低端。创建静态链接的 Linux 进程时，操作系统将整个正文段装入虚地址空间。通常是从地址 0 开始的，Linux 要求正文段必须以“只读”方式装入，因此在运行过程中，正文段不会做任何修改。

此外，如果用户进程运行结束或者通过 proc 文件系统接收到适当的命令，内核线程停止监控用户进程。

2.2 恢复、重启用户进程

内核线程进行以下操作来对用户进程进行后向卷回恢复：

(1) 阻塞用户进程运行；

(2) 恢复进程数据段(进程正文段随着程序的装入自动恢复)；

(3) 恢复非栈的一般的段；

(4) 恢复进程用户栈段；

(5) 恢复与进程上下文切换有关的项；

(6) 根据信号信息表恢复与信号有关的信息；

(7) 恢复与进程相关的文件信息；

(8) 重新启动进程运行。

恢复过程需要根据保存的检查点文件来进行。这个文件可能包含根据检查点算法得到的许多段，每一部分包含着与进程状态相关的信息。例如在增量检查点中，第一段是进程的初始化状态，它是一个全检查点，接下来的段包含从前一个检查点处开始的增量变化。如果只恢复第一段则相当于从头开始启动进程。

3 主要函数设计

3.1 内核模块主要函数设计

(1) 进程控制函数

stop_process(*pid_t pid*)：参数 *pid* 为内核线程通过系统调用 *getpid*()函数得到的当前用户进程标识号，此函数功能是阻塞当前用户进程的运行。

unstop_process(*pid_t pid*)：参数 *pid* 同上面的 *pid* 值，此函数功能是当进程检查点设置完成或进程卷回恢复完成后，重新启动进程。

(2) 进程状态信息保存函数

int *ckpt_store*(*const char* *filename, *pid_t pid*, *int flags*)：参数 *filename* 代表保存检查点映像文件名，*pid* 代表用户进程标识号，*flags* 的值由两个可选项组成：*ckpt_code_file* 和 *ckpt_share_library*。若选择前一项则存储进程二进制文件中的代码段部分，否则不存储，在一般情况下不选择此项；若选择后一项则存储共享库，否则不存储。

(3) 进程状态恢复函数

int *rollback_restore*(*const char* *filename, *pid_t pid*)：参数 *filename* 代表检查点映像文件名，*pid* 代表卷回恢复的用户进程标识号。

(4) 进程消息记录函数

int *message_store*(*int msg_from_pid*, *int msg_to_pid*, *char buf*[*SIZE*])：参数 *msg_from_pid* 代表发送消息的进程，*msg_to_pid* 代表接收消息的进程，字符数组 *buf* 是存储消息内容的内存缓冲区。

(5) 进程消息恢复函数

int *message_restore*(*struct Message* *message, *pid_t pid*)：参数 *message* 是一个指向消息结构体的指针变量，*pid* 是要恢复进程标识符，程序执行时根据 *pid* 从消息结构体数组中找到进程相关的消息进行恢复。

3.2 检查点库主要函数设计

检查点库将检查点与恢复内核模块所提供的系统服务结合起来，并在此基础上进行丰富、扩充而形成用户能够直接使用的丰富的检查点接口，包括不同层次上对用户进程资源的状态检查和状态恢复 API、性能优化和策略支持 API 以及一些辅助性的 API。下面介绍几个主要的 API 函数：

(1) 用户进程状态信息保存接口函数

int *process_ckpt*(*pid_t pid*, char **name*, struct *Option* **option*)：参数 *pid* 是用户进程标识号，*name* 是保存检查点文件的文件名，参数 *option* 是结构体指针变量，通过设置它的成员变量值，可以根据不同的算法进行不同的检查点操作。

(2) 用户进程状态恢复接口函数

int *process_restart*(*char* **name*, *pid_t pid*)：根据 *name* 所指定的检查点文件模拟一个进程，并恢复其运行。

(3) 增量检查点函数

int *set_ckpt_incremental*(*pid_t pid*)：利用页面保护技术来识别哪些页面应该包括在增量检查点文件中，在设置一个检查点之后，触发 *mprotect*()系统调用，将全部页面的数据空间设置为只读，当对一个保护页面进行写操作时，系统就会捕捉到一个 SEGV 信号，这时将此页面的读写保护设置为读-写，并且此页面被标记为 *dirty*，当设置下一个检查点时，只包含“*dirty*”页面。

(4) 检查点压缩函数

int *ckpt_compress*(*pid_t pid*)：采用标准的压缩算法实现此函数，但是只有在压缩速率大于磁盘写数据速率，并且检查点被明显压缩时才能有效降低系统开销。此函数通常只在并行计算中用到。

(5) 进程内存空间相关操作函数

int *memory_include_bytes*(*caddr_t addr*, *unsigned* int *len*)：在检查点中包含起始地址为 *addr*，长度为 *len* 的内存段。

int *memory_exclude_bytes*(*caddr_t addr*, *unsigned* int *len*)：在检查点中排除起始地址为 *addr*，长度为 *len* 的内存段。

4 系统性能评价

试验环境为：AMD Athlon(tm) Dual Core CPU 4400+，2.31GHz，1G 内存，120G 硬盘。

应用程序采用矩阵幂运算，以在终端输入命令行的形式设置检查点及卷回恢复。

首先我们不使用任何策略，仅使用内核模块提供的功能对应用进程采取检查点及卷回恢复，试验数据如表 1 所列。

表 1 仅用内核模块功能对进程操作

矩阵幂运算幂数	采取检查点耗费时间(ms)	检查点大小(KB)	卷回恢复时间(ms)
1000	527.4	963	95.2
2000	542.1	996	96.3

由表 1 中的数据可以看到系统可以完整的保存、恢复用户进程状态，并且操作是全透明的。

然后我们融入策略支持来对应用进程采取检查点及卷回恢复，采用增量检查点策略，对应用进程进行两次检查点操作(CK1 和 CK2)，两次操作之间的间隔设为 3500ms。试验数据如表 2 所列。

表 2 增量检查点策略支持下的数据

试验数据 / 幂运算幂数	测量内容	采取检查点耗费时间(ms)	检查点大小(KB)	卷回恢复时间(ms)
1000	CK1	526.1	960	93.6
	CK2	47.2	69	94.7
2000	CK1	540.7	992	97.1
	CK2	45.2	63	95.2

由表 2 中的数据可知，对进程第二次采取检查点(CK2)所耗费的时间以及生成的检查点文件的大小与第一次(CK1)相比明显下降，这也正符合增量检查点的算法特点。

由于系统设计遵循机制与策略分离的思想，在内核层实

(下转第 214 页)

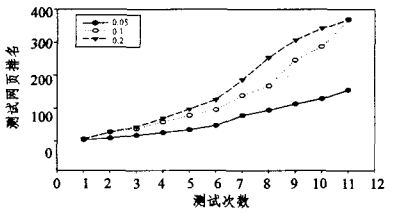

图3 可信排名效果

结束语 本文通过Web挖掘的数据域的可信评测，给出了一个Web挖掘可信域评测系统原型。利用Web域专业分类，评测域的专业度，并利用Web内容可信判定现有的结果和用户的主观评测反馈结果评测域的内容可信度，从而达到评测一个域挖掘的结果的可信度，从而获得Web挖掘的可信数据源，影响最终结果，以求在显示中将可信结果排列在前。在实现方法上，设计了信任域预处理，使之与挖掘引擎紧密结合，共同完成Web上的数据挖掘。本文下一步工作将进一步研究现有的信任评测和实现结果排序，以求更换推断的可信结果。

参考文献

[1] Zhou X, Li Y, et al. Using Information Filtering in Web Data Mining Process // 2007 IEEE/WIC/ACM International Conference on Web Intelligence. 2007:163-169

[2] Michael C, Hsinchun C. A machine learning approach to web page filtering using content and structure analysi. Decision Support Systems, 2008(44):482-494

[3] Ridvan S, Kemal T, Novruz A. A new approach on search for similar documents with multiple categories using fuzzy clustering. Expert Systems with Applications, 2008(34):2545-2554

[4] Manber U, Smit M, Gopal B. WebGlimpse: combining browsing and searching // Proceedings of the USENIX 1997. 1997,1

[5] Sun A, Lim E P. Performance measurement framework for hierarchical text classification. Journal of the American Society for Information Science and Technology, 2003, 54(11):1014-1028

[6] Yolanda G, Donovan A. Towards content trust of web resources. Journal of web semantics, 2007(11):337-358

[7] Kristin R E. Behind the Web site: An inside look at the production of Web-based textual government information. Government Information Quarterly, 2004(21):337-358

[8] Laender A H, Berthier R N, Altigran S. DEByE-date extraction by example. Data & Knowledge Engineering, 2002, 40(2):121-154

[9] Lin S H, Ho J M. Discovering informative content blocks from Web documents // Proceedings of the Eighth ACM International Conference on Knowledge Discovery and Data Mining. ACM Press, 2002:588-593

[10] Alberto D, Antonio G, Pablo G. User-centred versus system-centred evaluation of a personalization system. Information Processing and Management, 2008(44):1293-1307

[11] http://lucene.apache.org/nutch/index.html

[12] 高克宁，王波，等. WWW网站分类体系包装器WCSW[J]. 东北大学学报：自然科学版，2007，21(1)：44-48

[13] 王伟，张东启. 基于Bayes网络的内容信任[R]. 上海：同济大学嵌入式系统与服务计算教育部重点实验室，2008

(上接第194页)

现基本功能的基础之上，在用户层实现策略支持。这样就使系统具有良好的可扩充性、通用性、灵活性以及可维护性。

此外，通过proc文件系统，本检查点系统提供了一个简单的接口供用户以命令行的形式对进程进行动态操作，方便灵活。采取检查点时使用命令行：flag==checkpoint，需要恢复、重启进程时，使用命令行：flag==recovery，程序将加载检查点文件，回滚到最近的检查点状态重新执行。

结束语 本文简要介绍了用户级进程检查点系统与系统级进程检查点系统各自的优缺点，并在此基础上利用Linux内核模块设计实现了基于Linux内核的进程检查点系统。介绍了此系统的设计思路、工作原理以及工作流程，并且给出了几个主要函数的原型及各自的功能实现，最后对系统的性能进行整体评价，此检查点与恢复系统结构设计具有灵活性、通用性、良好的可扩充性以及可维护性优点。

参考文献

[1] Jose C S, Petrini F, Davis K, et al. Current Practice and a Direction Forward in Checkpoint/Restart Implementation for Fault Tolerance // Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005. IPDPS'05. April 2005:19

[2] Elnozahy M, Alvisi L, Wang Y M. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys, 2002, 34(3):375-408

[3] 汪东升，沈美明，郑纬民，等. 一种基于检查点的卷回恢复与进程迁移系统[J]. 软件学报，1999，10(1)：68-73

[4] 魏晓辉，鞠九滨. 分布式系统中的检查点算法[J]. 计算机学报，1998，21(4)：367-375

[5] Sancho J C, Petrini F, Johnson G, et al. On the Feasibility of Incremental Checkpointing for Scientific Computing // Proceedings of the 18th International Parallel & Distributed Processing Symposium, 2004. IPDPS'04. April 2005:58

[6] Luís M S, Joao G S. System-level versus User-Defined Checkpointing // Proceedings Seventeenth IEEE Symposium on Reliable Distributed Systems, 1998. ISRDS'98. October 1998:68

[7] Meyer N. User and Kernel Level Checkpointing // Proceedings of the Sun Microsystems HPC Consortium Meeting, 2003. April 2003:15

[8] Tannenbaum T, Litzkow M. The Condor distributed processing system. Dr. Dobb's Journal, 1995, 25(2):40-48

[9] Plank J S, Micah B, Gerry K, et al. Libckpt: Transparent checkpointing under unix // Usenix Winter Technical Conference. NewOrleans, Louisiana, USA, 1995

[10] Sankaran S, Jeffrey M S, Barrett B, et al. The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing // Proceedings of the LACSI Symposium, 2005. LACSI'05. October 2003:479

[11] Gioiosa R, Jose C S, Song Jiang, et al. Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers // Proceedings of the 2005 ACM/IEEE SC|05 Conference, 2005. SC'05. 2005:9

[12] Paul H H, Jason C D. Berkeley lab checkpoint/restart (BLCR) for Linux clusters. Journal of Physics, 2006, 46(3):494-499

[13] Zhong Hua, Nieh J. CRAK: Linux Checkpoint / Restart as a Kernel Module. Technical Report CUCS-014-01. Department of Computer Science, Columbia University, New York, November 2001