计算机科学 1996 Vol. 23 №. 6

基于微内核的操作系统中信号的处理

李新明 李 艺
(国防科工委指挥技术学院 北京 101407)

摘 要 One of the most important developments in operating system research is microkernel technology. In this paper, we discuss some issues related to signal processing in UNIX-style operating system based on microkernel, and introduce a mechanism to implement signal processing system. The mechanism has been used in COSIX V2.0, the new generation microkernel-based operating system developed by us.

关键词 Microkernel, Emulator, Servers, Signal, Signal information struction.

随着计算机技术的发展，传统的大内核结构的操作系统面临功能越来越多，内核日益庞大，复杂，可扩充性越来越差的问题。微内核技术以其小巧，可移植性好，可扩充性好等优点已经成为操作系统发展道路上的一个主流方向。在微内核结构的操作系统中，由微内核提供与特定的操作系统无关的基本功能。传统操作系统如 UNIX 的功能以服务器的形式在微内核外完成。本文介绍在这种结构下处理 UNIX 信号的一种机制，该机制已在新一代国产操作系统 COSIX V2.0 中实现。

一、COSIX V2.0 概述

新一代国产操作系统 COSIX V2.0 是基于微内核的，采用服务器和多态结构。其 API 界面遵循 POSIX.1 国际标准，并与 UNIX SVR4.0 二进制兼容。

在 COSIX V2.0 中，任务的地址空间被划分为核心态，系统态和用户态。应用程序在用户态下运行。COSIX V2.0 主要由下述几个部分组成：

①微内核。在核心态下运行，向核外提供一些基本的抽象，包括：任务、线程、存储对象、端口和消息。其中任务是资源分配单位，线程则是 CPU 调度的单位，存储对象用以描述任务的存储空间，端口和消息则用于实现任务间通信。

②基本服务器。在系统态下运行。为专用服务器等提供支持。

③专用服务器。在系统态下运行。包括进程管理服务器(简称 POS)，文件系统服务器(简称 FSS)，设备管理服务器(简称 DMS)，审计服务器(简称 AUS)等。传统操作系统的功能如信号，文件系统，安全等主要在这些服务器中实现。

④动态映射库(简称 DML)。在系统态下运行。设计 DML 的动机是为了能与特定的操作系统的 API 界面兼容。DML 被映射到每个用户进程的系统态空间。

COSIX V2.0 的结构如图 1 所示：

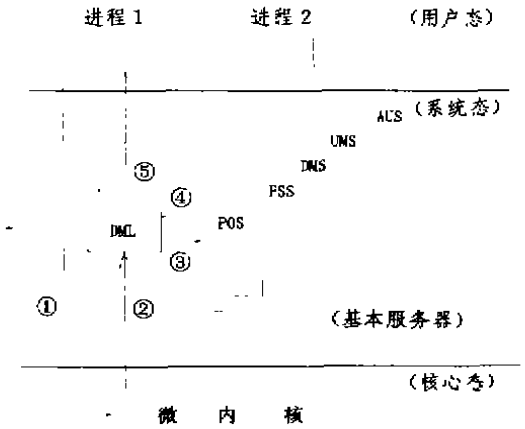

图 1 COSIX V2.0 系统结构

通常，在这种结构下，应用程序通过系统调用获取系统服务的过程如下：①陷入微内核。②在微内核中，保存一些状态，然后重定向到 DML 中。③如果有必要，则向服务器发送消息，执行系统调用的功能。④服务器执行完后，发送回答消息给 DML。⑤DML 将结果返回给应用程序。

顺便说明一点，典型的微内核结构的操作系统如 MACH 或 OSF/1 为两态结构，采用仿真库来实现。这种方法，在系统的安全性、资源的管理上都存在一些很严重的缺陷。比如，在 DML 与服务器共享

数据不会给系统带来致命的问题，但服务器则难以把仿真库作为一个“真正可信”的部分来对待。

二、UNIX 系统的信号机制

在 UNIX 系统中，信号是 UNIX 系统中的一种进程间通信机制，是对一个事件的异步通知。当一个长程序运行时，如果发现了错误，用户想终止它的执行；或内核检查出程序正在执行非法指令或出现其它异常情况；或想暂停一个程序的执行；或想使暂停的程序继续执行；或两个进程间要传送软中断进行某些同步控制，等等情况，均可以通过信号来实现。信号常常用于处理非正常情况。依照 POSIX 标准，并参照 UNIX SVR4.0，COSIX V2.0 共提供 32 种信号。

2.1 信号的产生

当与某个信号相关的事件发生时，就产生了该信号。信号产生的时机主要有以下几种：

①异常　当程序发生异常时，将生成相应的信号。如当执行非法指令时产生 SIGILL 信号；当出现算术异常时产生 SIGFPE 信号；当出现访问地址违章时产生 SIGSEGV 信号；当调用一个错误的系统调用时产生 SIGSYS 信号；当写一个没有读进程的管道时产生 SIGPIPE 信号；当电源掉电时产生 SIGPWR 信号等。

②终端活动　在系统运行过程中，对终端的某些操作也会导致信号的产生，其中大多数与作业控制相关。如切断一个终端将对以该终端为控制终端的所有进程产生 SIGHUP 信号；当用户在终端上按下中断键或退出键后，分别产生 SIGINT 或 SIGQUIT 信号；当后台进程想从终端输入或输出时产生 SIGTTIN 或 SIGTTOU 信号；当用户按暂停键时产生 SIGTSTP 信号；当再按恢复键时可对被暂停的进程产生 SIGCONT 信号，等等。

③计时器超时　用户可以用 alarm 系统调用对本进程设置超时警告闹钟，当计时器到达所设置的超时时限时，将产生 SIGALRM 信号。

④进程终止　当一个进程终止时，产生 SIGCHLD 信号，通知父进程。由此实现 exit 和 wait 系统对僵死子进程的处理。

⑤用户发送　用户可以调用 kill 或 sigsend 等系统调用向指定的进程发送信号，如发送 SIGKILL 信号给出现错误的正运行程序；进程之间通过发送信号进行同步控制等。

2.2 传统的大内核处理信号的时机

信号是以异步方式处理的，即一个信号产生后，系统将该信号设置到目标进程的 proc 结构中，但该目标进程并不能立即处理它，或者说不能立即检测到有信号，这之间有一段时间，而用户不能确定这段时间的长短，在 UNIX 系统中，通常在以下三种情况下检测信号：

①系统调用返回前。当用户进程调用任何一个系统调用后，在系统调用处理完成返回用户态之前，都要检查当前进程是否有信号要处理。

②外部设备中断或异常返回前。当系统发生外部设备中断或异常后，系统在中断或异常服务程序处理完成后，在返回用户程序之前的系统公共出口处，要检查当前进程是否有要处理的信号。

③时钟中断发生时。在 UNIX 系统中，在每一个最小计时单位里都将发生时钟中断，处理和时钟相关事件，在时钟中断返回前，要检查当前进程是否有信号要处理。

这几个时机的选择可以保证：任何进程都能“及时”地检测到发送给它的信号而进行处理。

2.3 信号的处理动作

信号被检测到后，将根据信号所设置的动作进行相应的处理，与信号相关的动作分为三种：

①默认动作　UNIX 系统对每一个信号都规定了进程创建时相应的默认动作，默认动作有以下几种：

· 将进程终止，这类信号如 SIGKILL，SIGPIPE，SIGINT 等。

· 将进程现场进行核心转贮，生成 core 文件后，终止进程。这类信号如 SIGQUIT，SIGILL，SIGFPE，SIGSEGV，SIGSYS 等。

· 不进行任何处理，忽略该信号。这类信号如 SIGCLD，SIGPWR 等。

· 将进程暂停，这类信号如 SIGSTOP，SIGTTIN，SIGTTOU 等。

· 如果当前进程处理暂停状态，则恢复进程执行，否则忽略该信号，这种信号只有 SIGCONT 一个。

②忽略信号　可以将一个信号的动作设置为忽略或信号的默认动作是忽略信号，则该信号的发送对进程没有任何影响。

③捕获信号　用户可以自己定义一个函数，并设置为某一个信号的信号捕获函数，当信号被交付时，则执行该函数，该函数执行完成时，返回被中断的地方继续执行。

除 SIGKILL 和 SIGSTOP 外，所有的信号都可以被屏蔽，进程检测到被屏蔽的信号时不做任何处理，只有当该信号被解除屏蔽之后，才按信号的动作进行相应的处理。

三、微内核结构下信号机制的设计与实现

信号在单一内核中处理时，操作系统内核自己完成了信号的产生、发送和检测工作，并根据需要采取相应的处理动作。但在基于微内核的操作系统中，这些活动不能由微内核单独完成，而必须由微内核、服务器和动态映射库共同完成。

3.1 信号的发送

在 2.1 节中，我们介绍了产生信号的五种活动。在 COSIX V2.0 中，这些信号的发送过程如下：

·对必须转换为信号的程序异常，首先由微内核捕获并做一些处理，然后交给进程管理服务器。它根据异常种类产生相应的信号后发送到目标进程。

·对产生信号的终端活动，由微内核捕获并处理中断后，再交给设备服务器做进一步的处理，然后通知进程管理服务器。进程管理服务器将根据终端活动类型产生信号并发送给目标进程。

·对超时、进程终止及用户通过调用系统调用而发送的信号，由进程管理服务器直接产生并发送给目标进程。

3.2 信号的检测

在大内核操作系统中，需要由用户进程处理的信号总能被目标进程检测到并加以处理，然而在基于微内核的操作系统中，是否也是这样的呢？

①系统调用处理时。由于所有的系统调用都要经过 DML 处理，因此只要应用程序调用系统调用，就能在 DML 中检测到是否有信号要处理。

②设备中断和异常处理时。设备中断和异常处理是在微内核中捕获并由微内核或服务器而不是由 DML 来处理。因此，在目标进程的 DML 中不能检测到是否有要处理的信号，但在进程管理服务器中知道哪个目标进程有信号要处理。

③时钟中断处理时。时钟中断的所有处理都是在微内核内部实现的，所以不可能利用时钟中断来检测当前进程是否有信号要处理。

从上面的分析可以看到：在微内核机制下，如果简单地借鉴 UNIX 系统的信号机制，会发现信号检测的时机不充分，如果一个用户程序永远不调用系统调用，它的信号就永远无法检测到。

3.3 主要实现策略

为在 COSIX V2.0 中实现 UNIX 信号机制，采用了下述策略：

①在信号发送时对信号进行处理。分析信号产生的时机可知：在任何情况下，都是由进程管理服务器来发送信号的，因此可以在信号发送时，对动作为默认或忽略的信号直接处理，而不用再发送给目标进程。

②用伪系统调用处理信号捕获。服务器在发送信号时，如果发现信号被捕获，则必须在目标进程的环境中运行捕获函数，这时，将根据目标进程的状态分别处理。如果目标进程已在 DML 中运行，则可以只将信号设置到由服务器和 DML 共享的数据结构 ptask 中，由目标进程从 DML 中退出前，对信号进行捕获处理。如果进程在运行用户程序，它可能永远也不调用系统调用、而无法进入到 DML 中。因此这种情况下，服务器将修改目标进程的现场，仿佛用户程序刚调用了一个系统调用而重定向到 DML 中，称之为伪系统调用。伪系统调用的功能就是对信号进行处理后，从被中断的地方重新运行。

③用定时线程处理未能处理的信号。通过上述两种策略的处理之后，大多数的信号已被处理，但如果同时发送了多个信号，或信号发送时被屏蔽，但后来同时打开了多个等情况下，都可能有未被处理的信号，为此在服务器中设计一个定时线程，将所有包括未被处理信号的进程链在一起，该线程被定时唤醒，对链中的进程逐个处理其信号。

④用共享的方法处理信号结构。在 UNIX 系统中，每个信号都可能有对应的信号结构，其中包含信号的一些补充信息，如异常地址、异常号、僵死的子进程号等。用户可以在信号捕获函数中访问该结构中的内容，UNIX 系统中将所有的信号结构链在一起，信号处理时从队列中获取对应的信号结构。在微内核结构下，信号结构在服务器中构成，而要从 DML 中传给用户。在 COSIX V2.0 中，我们可以将信号结构"放心地"放在 DML 和服务器共享的存储区中。由于 UNIX 系统提供 32 种信号，并且信号不排队，即如果一个信号在被处理之前已产生多次，也只执行一次信号动作，所以每个信号最多只有一个信号结构。因此，可以在共享区中定义一个 32 项的结构数组，每一项中存放对应信号的信号结构，并有一个标志位表示该信号结构是否有效。

总之，微内核与传统的大内核操作系统在体系结构上有很大的差别，如果按照大内核操作系统对

（下转第 93 页）

转换两侧的标记(input,output)是激发的先决条件。图2中,start 转换,在对话开始打开一应用窗口,只有双向关系才能脱离该窗口控制,所有整个对话期间,该窗口保持打开状态。通过转换 Customer Open、Customer New、Customer Search,各自的视图及子对话被打开,Open Customer 视图让用户通过顾客号码打开一个顾客。

Search Customer 视图通过属性查找给出满足一定条件的顾客名单。图中的选择流关系不影响激发条件,如图中的较低位置的 cancel 转换可激发而不管 Search Customer 视图是否标记(open),即若 Search Customer 对话框以前未关闭,那么 cancel 转换关闭顾客名单(Customer List),然后关闭 Search Customer 对话框。

象图中所示,对话网是分层构造的,当图中 Complex place 的子对话被标记时,一个子对话网开始。进一步细化规范,要对转换指明事件和条件,这被看作激发的附加条件,指明转换被激发后执行的动作。图3指出了细化后的 Customer New 转换的规范。

event:Filemenu. Newselect/视图描述静态
condition:Currentselection=Customer
action:Number:=/对话描述动态
GeneratCustomerNo( )

图3

对话网主要用来描述粗粒度对话,而细粒度对话仍是通过文本规范,易于描述和理解。

三、工具支持

为了在实际应用中引入一种方法的具体建模技术、工具的支持是必不可少的。

·用户接口的概念模型的产生:从分析任务的交互过程中确定主要对象,然后确定各自的容器对象和应用对象及其访问关系。对数据对象,从应用说明中得出属性和方法。当然模型的初步建立需要通过指明数据对象间的访问关系,及为组织对话引入新对象以进一步细化。

·对话模型的产生:根据访问关系为每个概念对象、对话转换假定一个视图。通过增加对话框来进行细化。

·从逻辑视图产生物理视图:逻辑视图在对象模型的基础上确定。物理视图由逻辑视图而来,是基于属性类型和值域通过规则(交互对象的选择和布局)来确定。

·可执行的对话的产生:从形式化说明的转换,可产生可执行的对话。

结论 为了精简窗口及导航的数目,设计应通过对象层次的树形表示来细化。比较菜单表格的 UI 设计,建模技术对表示 UI 设计是有益的。我们采用的 OO-GUI 设计,其复杂性有所增加,但对话网能提供一种精确的规范说明。虽然完整的工具支持我们尚在开发阶段,但我们对采用面向对象的分析,并用分析结果指导 GUI 设计抱有极大信心。

(上接第58页)

信号的处理方法,来设计微内核结构下的信号机制,则发现有些信号不能被检测到。如当一个进程不调用系统调用时,发送给它的信号将永远检测不到。本文介绍的微内核结构下的信号机制,它的要点就是在信号发送时进行信号处理,系统调用返回前进行信号处理,增加一个定时线程,定时处理系统中有待处理的信号的进程,而信号的信号结构可在共享区中通过信号结构数组来实现。上述设计思想已在国家的"八五"重点攻关课题"系统软件平台"中的操作系统 COSIX V2.0中实现,并通过了鉴定。当然,该系统在性能优化、可靠性、功能完备等方面还有很多工作要做。

致谢 COSIX V2.0是中软总公司、国防科工委指挥技术学院、中科院软件所、海军计算技术研究所等单位共同研制完成的。作者对以孟庆余教授、周明德教授为首的课题组的同志们所给予的帮助和合作表示感谢。并特别感谢殷树勋高级工程师和韩乃平同志,方案的设计和实现与他们的许多建设性意见和帮助是分不开的。

参考文献

[1]Joseph Boykin, Programming under Mach, U. S. A.: Addison-Wesley Publish Company, 1993

[2]Mike Accetta, Mach: A New Kernel Foundation for UNIX Development, U. S. A.: In Proc. of Summer Usenix, 1986

[3]Golub, Unix as an Application Program, U. S. A.: In Proc. of the Summer Usenix, 1990

[4]周明德等,计算机环境的可移植操作系统界面 POSIX.1,电子工业出版社,1990

[5]杨学良等,UNIX SYSTEM V 内核剖析,电子工业出版社,1990