操作系统 指令 文件 直接寻址 (26)

计算机科学 2000Vol. 27№ 12

在操作系统实现指令对文件的直接寻址

Implement Instructions Access Files Directly in Operating Systems

105-107,94 TP316

刘福岩[1,2] 尤晋元[1]

([1]上海交通大学计算机系 上海 200030)([2]华北工学院计算机系)

Abstract In this paper, the concept and implementation of Instructions Directly Access File are discussed, its advantage and profit are analysed. The author belives Instructions Directly Access File should be adopted in operating system

Keywords Operating system, Memory management, File

1. 引言

在计算机系统中，进程的数据存储在内部存储器和外部存储设备上，但操作系统并不允许进程直接访问这两种资源，允许进程直接访问的是另外两个逻辑上的数据存储模型：逻辑空间和文件[1]。逻辑空间是物理存储器的抽象，通过为每个进程构造各自独立的逻辑空间，操作系统不仅实现了各个进程之间数据的相互隔离，而且很方便地实现了存储保护、存储扩充、存储共享等功能。可以这样说，现代操作系统中的存储管理，就是围绕如何构造和维护进程的逻辑空间而展开的。

操作系统允许进程直接访问的另一个逻辑上的数据存储模型是文件。文件是外部存储设备的抽象。利用文件的概念，进程可以通过文件名访问外部存储设备上的数据，而不是利用外部存储设备的物理地址(驱动器、柱面、磁道、扇区等)。这不仅方便了进程访问外部存储设备，也提高了外部存储设备的使用效率，特别是统一了进程访问各种设备的界面，使进程可以使用一组功能调用访问各种不同的外部存储设备，以至于大多数的计算机用户在使用各种外部存储设备时已经很少想到磁道、扇区等概念，而把各种外部存储设备看成是存储文件的容器了。

在操作系统中，逻辑空间和文件是两个不同的概念：逻辑空间属于某一个进程，而文件可以被多个进程所访问；逻辑空间和进程一样是有生命期的，是暂时的，而文件是无生命期的，是永久的；进程可以直接访问逻辑空间中的数据，而对于文件中的数据，进程必须通过读写操作在逻辑空间和文件之间拷贝数据，或者通过存储器映射文件在逻辑空间和文件之间建立起对应关系，以逻辑空间为媒介间接地访问文件中的数据。

长期以来，在操作系统中为进程构造进程逻辑空间和文件两种数据存储模型，一直被认为是理所当然的事情，无论是传统的操作系统，如 VMS、UNIX，还是现代的操作系统，如 MACH[2]、CHOROUS[3]、AMOEBA[3]、PLAN9、Spring OS、Windows NT[4]、MINIX[5]、LINIX，允许进程直接访问的数据存储模型，都是逻辑空间和文件。但是，通过下面的分析，我们认为：操作系统应该为进程仅仅提供一个数据存储模型，该数据存储模型同时具有逻辑空间和文件的优点，即可以象逻辑空间一样被进程直接访问，又能够象文件一样长期永久性地存储。事实上，这是完全能够实现的，实现方法是：取消逻辑空间，实现指令对文件的直接寻址。

2. 指令对文件直接寻址的实现方法

在计算机系统中，指令是对逻辑空间直接寻址的，例如在保护模式下，80X80 和奔腾的汇编语言 MOV DS:[2000H],AX 表示把 AX 寄存器中的内容送入逻辑空间 DS 段中偏移量为 2000H 的地址单元中去，这个 2000H 单元可能是物理内存的某一地址单元，也可能是外部存储设备某一物理位置。在后一种情况下，该指令的执行将引起存储器访问失效异常，由异常处理程序把数据由外部存储设备调入内存，然后修改段表和页表，重新启动指令执行，逻辑空间至物理空间和外部存储设备的对应关系由操作系统维护，如果把映射关系定义为逻辑空间至文件的映射，即可实现指令对文件的直接寻址，具体而言可采用如下实施方案：

(1)CPU 采用段式或段页式存储管理，这显然是不难做到的，现在的许多微处理机芯片例如 80X86 和奔腾等都支持段式和段页式寻址，因此操作系统为进程提供的存储空间是两维的：段和段内偏移量。

(2)在逻辑上，进程的一段对应外存上某一文件中的一段区域。

(3)进程在需要时打开文件，由操作系统负责创建

段表和页表并维护段和文件的对应关系，把段号作为打开文件的标识数返回给进程，这样进程就可以通过该段号访问该段中的数据也就是对应文件中的数据。

(4)当进程所属线程执行指令对某一段也就是某一文件寻址时，由于该数据在外存引起存储器访问失效异常，由异常处理程序负责把数据由外存的文件中读入内存并修改段表和页表后重新启动指令执行。

(5)当内存资源比较紧张时，操作系统可以暂时把数据反写回文件中，释放占用的存储资源供其它进程使用。

(6)最后进程关闭文件，把内存中的数据反写回文件中，释放占用的存储资源同时释放相应的段表和页表。

通过以上方案即可实现当进程指令对某一段寻址时，就好象指令对文件直接寻址一样，从而对进程而言没有了逻辑空间的概念，进程不是在逻辑空间中运行，而是在文件上运行。

指令对文件直接寻址，原理上和存储映射文件是很相似的。现在许多操作系统，如 BSD 和 System V UNIX、SunOS 和 Solaris、Windows 95、98 和 NT、Mach 和 Chorous 等都支持存储映射文件。在这些操作系统中，通过把文件的一段区域映射到进程的逻辑空间中，该进程就可以通过对该逻辑空间的访问来直接读写文件中的数据。利用这种方式存取文件比传统的通过读写操作在逻辑空间和文件之间拷贝数据的方法要方便得多，甚至在 Mach 和 Chorous 等操作系统中，传统的读写文件操作就是通过存储映射文件来实现的[2,3]，因此指令对文件直接寻址在技术上是完全可行的。

虽然指令对文件直接寻址和存储映射文件实现原理很相似，但二者却是两个完全不同的概念。

在支持存储映射文件的操作系统中，仍然存在逻辑空间的概念，进程仍然是在逻辑空间中运行的，逻辑空间是进程唯一可以直接访问的数据存储模型。存储映射文件技术，通过在逻辑空间和文件之间建立起对应关系，为进程提供了一种快速、高效、方便的访问文件的方法，没有这项技术，进程仍然可以通过传统的读写文件的方法访问文件中的数据。在支持存储映射文件技术的系统中，逻辑空间和文件这两个概念都是存在的。

而指令对文件直接寻址，强调的是彻底抛弃逻辑空间的概念，强调进程在文件上运行而不是在逻辑空间中运行。当进程执行打开文件的系统功能调用时，该调用返回的文件标识数就是该打开文件对应的段号，把该段号送入段寄存器即可象访问内存一样访问文件中的数据。当进程关闭文件时，不仅要把该段中的数据写入对应的文件中，同时还要释放该段对应的段表和页表。关闭文件后由于段表和页表已不存在，进程也就不能再访问文件中的数据了。

存储器映射文件技术和指令对文件直接寻址，二者之间不同的关键在于逻辑空间是否存在的问题。笔者认为：取消逻辑空间的概念，实现指令对文件的直接寻址，使进程直接在文件之上运行，具有许多传统操作系统所没有的优点，对操作系统的设计具有十分重要的意义。

3. 指令对文件直接寻址的优点

1)精简了操作系统的代码　在传统的操作系统中，需要同时实现内存管理和文件管理。

内存管理需要实现：(1)逻辑空间到物理空间的影射；(2)内存存储空间的管理；(3)对换区管理；(4)进程图像在内外存之间的传输。

文件管理需要实现：(5)进程操纵文件的接口；(6)缓冲区管理；(7)文件数据在缓冲区和外存之间的读写；(8)外存存储空间的管理。

而在支持指令对文件直接寻址的操作系统，进程的指令直接读写文件中的数据，因此不再需要实现(5)进程操纵文件的接口；外存全部用于存储文件，不需要划分出对换区或对换文件，因此也不需要实现(3)对换区管理和(4)进程图像在内外存之间的传输。同时，由于内存存储空间全部用于文件缓冲区，进程页表直接映射到文件缓冲区，因此也不需要实现(2)内存存储空间的管理。这样在结构上仅仅需要实现：(1)逻辑空间到物理空间的影射，也就逻辑空间是到缓冲区的映射、(6)缓存区管理、(7)文件数据在缓冲区和外存之间的读写和(8)外存存储空间的管理。显然指令对文件直接寻址不仅完全可以实现传统的操作系统存储管理和文件管理的功能，而且避免了功能上的重复，精简操作系统的代码。

2)提高了程序运行的速度，也提高了存储资源利用率　在传统的操作系统中，内存资源和外存资源分别被分成了两部分。对内存而言，一部分内存用于存储进程图像，另一部分内存用于读写文件的缓冲区；对外存而言，一部分用于对换区或对换文件实施对进程图像的后援存储，另一部分用于文件存储区存储文件中的数据。

在支持指令对文件直接寻址的操作系统中，内存资源和外存资源是分别作为一个整体来使用的，而不象在传统操作系统中那样被分别分成了两部分。进程页表把进程逻辑地址直接映射到文件缓冲区上，全部内存用于访问文件的缓冲区，因此将提高页表映射的命中率，避免了内外存之间频繁的数据交换，加快了进程运行的速度。

由于在外存上不再划分出对换区或对换文件，外存资源全部用来存储文件中的数据，从而提高了外存资源的利用率；同时由于内存不再划分出用于存储进程图像的存储区，也提高了内存资源的利用率。

3）方便了用户编程　在传统的操作系统中，用户开发程序时，既要设计用户数据在进程逻辑空间中和文件中的数据结构，还要设计用户数据在进程逻辑空间和文件之间的传输。采用存储器映射文件技术，把文件直接映射到进程的逻辑空间中，虽然可以避免数据的传输问题，用户仍然要掌握逻辑空间的概念，特别是需要安排各个文件在逻辑空间中的布局。由于同一文件可以映射到逻辑空间中不同的位置，使得在这些操作系统中很难把逻辑空间中指针的概念和文件偏移量对应起来，在文件上实现指针的概念，这是因为在传统操作系统中，从逻辑上讲指针是一逻辑空间中的地址，和文件偏移量没有一一对应的关系。

采用指令对文件直接寻址，取消逻辑空间的概念，那么用户在开发程序时，仅仅需要设计进程数据在文件中的数据结构，这样不仅避免了设计进程数据在内存中的数据结构，也避免了设计进程数据在内存和文件之间的传输。同时，进程没有了逻辑空间的概念，进程数据直接存储于文件之中，而且是永久性地存储。不需要用户考虑永久性保留数据的问题，即使进程消亡甚至系统关机，进程的数据也仍然是存在的，从而也方便了对进程的跟踪、调试和出错诊断。特别是采用指令对文件直接寻址，要求操作系统采用段式或段页式存储管理，打开文件时创建相应的段表和页表，把段号作为访问文件的标识数返回给调用进程，通过段号直接访问文件中的数据，因此可以在文件上支持指针的概念。在逻辑上指针已不再是逻辑空间中的地址，而是文件中的偏移量了。

4）便于进程的动态迁移，实现分布环境下的负载平衡　在传统操作系统中实现进程的动态迁移，首先要在源结点上挂起要迁移的进程，再把进程图像和其他状态信息从源结点传输到目的结点，然后在目的结点上启动进程继续执行[4]。

如果在操作系统中采用指令对文件直接寻址，那么进程消亡后进程图像被反写回文件中，只要在目的结点上创建一打开相同文件的进程，并恢复进程在源结点上时的状态信息，即可方便地实现进程迁移。在采用指令对文件直接寻址的操作系统中，不需要把进程图像迁移至目的结点。当进程在目的结点继续执行时，进程所属线程访问哪一页，操作系统就从文件中调入哪一页。如果进程迁移后仅仅访问部分内存图像，采用指令对文件直接寻址可以大大减小进程迁移所需要的时间，也减小了网络传输的开销。

5）适合大规模多处理机系统　据文[2]，DRAM的容量每三年增加四倍，硬盘容量的年增长率仅为30%。计算机系统中内存的容量正在不断扩大，相比较而言，外存容量的增加则要慢得多。根据现在制造大规模多处理机系统的技术水平，系统中处理机的数目已达几百，有些系统甚至达几千。考虑到这些系统中外部存储设备的数目远小于处理机数，如此众多的处理机所具有的存储容量之和，不仅有可能接近甚至有可能超过外存的容量。如何在这些系统中实施存储扩充是一个难于解决的问题。

在计算机系统中，外存的主要作用是为用户提供存储信息的永久性载体，主要用途是存储文件，操作系统不可能把大部分外存用于对换区。但对换区开辟太少，存储扩充也就失去了意义。因此对于面向大规模多处理机的操作系统，如何通过内外存交换为进程提供存储扩充，就成了值得研究的问题。在操作系统中取消存储扩充，是解决这个问题的一个最简单的办法。有些操作系统，例如 Amoeba 就是这样实现的[1]。在 Amoeba 操作系统中，彻底取消了存储扩充，进程图像常驻内存。但是从进程的角度来分析，虽然在多处理机系统中总的内存容量很大，单个处理机所能访问的内存并不一定很多，往往还是远小于外存的容量。对于处理机之间通过消息传递而不是共享存储器通讯的大规模多处理机系统，情况尤其如此。因此，对于运行于一个或几个处理机上的进程而言，如果需要处理大量的信息，还是需要系统为其提供存储扩充。

指令对文件直接寻址可以很好地解决这个问题。如果在操作系统中采用指令对文件直接寻址，取消逻辑空间的概念，进程在文件上直接运行，那么，在外存将取消了对换区，外存存储空间全部用于存储文件，同时内存全部用于进程访问文件的缓冲区，进程页表直接映射到文件缓冲区上，划分对换区的问题已不存在，存储扩充问题也就迎刃而解。对进程而言，系统为其提供了存储扩充，每个进程所能直接访问的数据量扩充至整个外部存储器的容量；对操作系统而言，取消了外部存储设备上的对换区，全部外存资源用于存储文件，也解决了传统操作系统中由于开辟对换区引起的外存资源的浪费，提高了外存资源的利用率。

4. 关于文件的进一步讨论

现代的操作系统大多是基于微内核的。在这些操作系统中，文件由运行于核外的文件服务器进程来管理。进程通过和文件服务器进行进程通讯访问文件中的数据。从进程的角度来分析，文件不过是被某一进程所维护并可被其他进程通过进程通讯所访问的一种数

（下转第 94 页）

从上面的识别结果中，我们可以看出这四种离散化算法都是有效的。前三种算法效果大致相当，改进算法2效果最差，根据对离散化问题的分析，我们再把识别结果和断点结果结合起来看，改进算法2得到的断点数目和剩余的属性个数较多，把学习实例所属的属性空间划分得太细，以致得到的规则的适应度较小，用得到的规则去测试时，导致测试数据中拒识或误识的比例较大（这说明以 S^+ 中行的1的个数为主来度量断点的重要性不合适）。而基于属性重要性的算法虽然得到的断点较多，但是剩余属性较少，从而减小了学习实例的空间维数。贪心算法和改进算法1在剩余属性个数和断点个数两方面相对折中，也是有效的离散化算法。

结论 通过对离散化问题的系统研究，我们认为，为了能够最大限度地提高识别正确率，减小误识率和拒识率，在进行数据离散化处理时，应该在保持信息系统不可分辨关系不改变的前提下，尽量减小信息系统的复杂程度，使离散化后的信息系统的属性个数尽量少，每一属性上的断点尽量少，由于各信息系统蕴涵的领域知识不同，不可能提出一个离散化算法针对所有的信息系统都是最优的。前面提出的几种离散化算法是从不同的出发点出发的，有其各自的特点，在进行知识获取时，可根据具体情况选择合适的离散化算法。

参考文献

1 Nguyen H S, Skowron A. Quantization of real values attributes, rough set and boolean reasoning approaches. In: Proc. of the Second Joint Annual Conference on Information Science. Wrightsville Beach, NC, 1995. 34～37

2 Nguyen S H, Nguyen H S. Some Efficient Algorithms for Rough Set Methods. In: Proc. of the Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems. Granada, Spain, 1996. 1451～1456

3 曾黄麟. 粗集理论及其应用—关于数据推理的新方法. 修订版，重庆：重庆大学出版社，1998. 83～87

4 常犁云，王国胤，吴渝. 一种 Rough Set 理论的属性约简及规则提取方法. 软件学报，1999，10(11)：1206～1221

5 Knowledge Systems Group. Rosetta Technical Reference Manual, 1999

（上接第107页）

据结构而已。如何管理、保存和维护这一数据结构对进程而言是透明的。对进程而言，无论是文件服务器所维护的文件还是其他进程所维护的数据结构，它们都是进程可以通过进程通讯访问的被其他进程所维护的数据结构，进程感觉不到它们之间有任何差别。如果按照这种观点来看待文件，那么指令对文件直接寻址，就是指令对通过进程通讯访问的被其他进程所维护的数据结构的直接寻址。

在采用指令对文件直接寻址的操作系统中，就象在传统的操作系统中一样，仍然为进程构造逻辑空间，而且是两维的段页式逻辑空间，但逻辑空间中的数据如何存储和维护，则不再由操作系统来负责，而交给了运行于核外的其他进程。进程之间通过进程通讯交换数据，计算机系统中的内存全部用作进程访问其他进程所维护数据结构的缓冲区，通过把进程页表映射到缓冲区上为进程构造可直接访问其他进程数据的逻辑空间。

因此，指令对文件直接寻址实现了操作系统中数据存储和数据处理的分离，数据处理由一个进程实现，数据存储由另一个进程实现。如果数据存储由文件服务器管理，就是上面讨论的指令对文件的直接寻址。

结论 本文提出了指令对文件的直接寻址的概念和实现原理，并分析了和传统的操作系统相比所具有的优点，笔者认为：把逻辑空间和文件统一起来，取消逻辑空间的概念，实现指令对文件的直接寻址，使进程直接在文件之上运行，具有许多传统操作系统所没有的优点，值得在操作系统的设计中采用和推广，对操作系统的设计具有十分重要的意义。

参考文献

1 Tanenbaum A S. Distributed Operating System. Prentice Hall Inc, 1995

2 樊建平，李国杰. 并行操作系统的现状和发展趋势. 计算机研究与发展，1995(1)

3 陈华瑛. Mach 3.0 核心分析. 计算机研究与发展，1994(9)

4 陆桑璐，谢立. 进程的动态迁移技术. 计算机研究与发展，1997(9)

5 江南计算所，吉增瑞. 操作系统的并行处理. 计算机世界，1996