第5卷第4期
1994年4月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 5, No. 4
Apr. 1994

对MINIX操作系统的若干改造和扩充*

黄祥喜

(中山大学计算机科学系,广州 510275)

摘要 本文论述了我们对美国知名学者A.S. TANENBAUM 1986年编制的分时多任务多用户操作系统MINIX所做的若干重要改造和扩充,包括对3.5寸高密盘的支持,AT机扩展内存的利用,进程对换和假脱机系统的实现.

关键词 操作系统,计算机教学,MINIX.

MINIX[1]操作系统是由美国知名学者A.S. TANENBAUM 1986年编制的一个教学用分时多任务多用户操作系统.从用户层看,它很象UNIX(与UNIX V7兼容);然而在内部构造上,它采用了层次化体系结构,各模块之间采用消息机制通信,结构清楚,易于理解,易于修改和扩充.因此,MINIX为学生系统地掌握操作系统课程提供了一个优良的模型环境.

从1989年开始,我们对MINIX系统做了细致的分析和研究,并从内部结构、用户界面、辅助工具等方面对它做了改造和扩充.1992年4月,我们研制的MINIX扩充版,MINIX+1.0操作系统实验环境通过了国家教委组织的专家鉴定.专家们认为,本系统具有国内领先水平,并在功能和结构上优于MINIX初版.本文重点介绍我们在改造和扩充MINIX过程中的一些关键思路.关于MINIX+1.0操作系统实验环境的实现原理将在另文论述.

1 MINIX对3.5寸高密盘的支持

1.1 问题的分析

在MINIX中,与软盘驱动器打交道的唯一程序是内核(Kernel)的floppy.c,它支持的磁盘每磁道至多只能有15个扇区,而3.5英寸高密软盘每磁道有18个扇区(IBM-PC DOS格式),因而受floppy.c限制至多只能使用15个扇区/磁道.这样使用起来虽然也十分正常,但是,它没有充分利用,每磁道要浪费3个扇区.

彻底解决这个问题的方法是专门编制一个3.5英寸盘的驱动程序(?),或修改原有的软盘驱动程序.我们选择后一方案,这应该是一个最佳的选择.因为3.5英寸盘驱动器所用的磁盘控制器(FDC)同5.25英寸盘是一样的,因而FDC的命令相同,端口等约定也一致,没有必要专门编制一个3.5寸盘驱动程序.

---

* 本文1992年1月24日收到,1992年5月30日定稿
本课题受广东省科学基金资助.作者黄祥喜,32岁,副教授,主要研究领域为自然语言理解,操作系统,知识工程.
本文通讯联系人:黄祥喜,广州510275,中山大学计算机科学系

1.2 实现原理

我们对 MINIX 原版的软盘驱动程序 floppy.c 做了以下扩充:增加了几个注释,在几个参数表中加入了有关 3.5 英寸盘的一组参数,将几个常量进行了调整,且将 floppy.c 的 do_rdwt 源程序中循环外的三个语句移入循环内.经过测试,修改后的 floppy.c 可支持 3.5 英寸盘.这不是什么奇迹,是 MINIX 高度模块化和与设备无关性优点的体现.

需要注意的是,当用修改了的 floppy.c 重编译生成一新的 MINIX 系统后,还要在"/dev"目录下建立一设备节点,具体方法如下:

mknod /dev/3ID b 2 1 〈CR〉

这里在"/dev"目录下建立一名为 3ID,主设备号为 2,次设备号为 1 的设备(即 2 号软盘驱动器).然后将 3.5 英寸高密盘插入 2 号驱动器中,打入

mkfs /dev/3ID 1440 〈CR〉

即可建立一 1.44M 的文件系统盘.

2 AT 机扩展内存的利用及 RAM 盘的改造

2.1 问题的分析

由于 MINIX 是工作在 286 实模式下的,这就限制了它直接利用扩展内存.幸好,在 AT 机及其兼容机的 ROM 中包含的 15H 中断调用,提供了主存与扩展内存之间数据传输的手段,使利用扩展内存存贮数据成为可能.这促使我们把扩展内存作为块设备来利用.在 MINIX 所管理的块设备中,RAM 磁盘是其中的一个.它是用预先分配好的部分主存来储存数据.可以设想把扩展内存划为 RAM 磁盘,以腾出更多的内存来运行程序.MINIX 的设备无关性使我们能够较方便地实现这一设想,而无需对整个系统作大范围的调整.在 MINIX 中,RAM 磁盘作为一个块设备,是由 RAM 磁盘驱动程序管理的.RAM 磁盘驱动程序实际上是把紧密相关的四个驱动程序合成一个.它管理了四个次设备:

0:/dev/ram　　1:/dev/mem　　2:/dev/kmem　　3:/dev/null

其中/dev/ram 是一台真正的 RAM 磁盘.

现在我们分析一下 RAM 磁盘的工作原理:在系统自举的时候,系统在内存中划出一部分作为 RAM 磁盘.根据分配给它的存储数量,RAM 磁盘将被分割成几个块.每块大小与真磁盘的块相同.当 RAM 盘驱动程序接到读或写的消息时,它计算被请求的块在 RAM 磁盘存储器中的位置,并调用汇编子过程 phy_copy,在 RAM 盘空间和用户空间之间进行数据复制.具体地,RAM 磁盘驱动程序是在 memory.c 中实现的.这里需要指出的是,在 MINIX 中,RAM 磁盘的大小及位置均不装在 RAM 磁盘驱动程序中,而是在系统自举成功后由文件系统(FS)中的 load_ram 程序向 RAM 磁盘驱动程序发送消息,告诉它 RAM 磁盘的上下限.通过以上分析可以知道,为实现我们的目的,要做两件事:一是设法告诉 RAM 磁盘驱动程序,RAM 磁盘的新位置(即 RAM 盘位于扩展内存区);一是改动 phys_copy 子程序,使它能在 RAM 与扩展内存之间传递数据.

2.2 实现原理

首先是将 RAM 磁盘安装在扩展内存中.为此,我们对 FS 中的 load_ram 程序作如

下修改：

1. 从 Kernel 中读取根文件系统所在的设备号.

2. 调用汇编子过程 get_eramsize 取得系统扩展内存大小至 eramsize，并计算其块数 eram_count.

3. 读 RAM 盘超级块，从中取出 RAM 盘的块数 root_count.

4. 比较 root_count 及 eram_count. 若 eram_count ＞＝ root_count，则置变量 ram_clicks＝0，否则置 ram_clicks＝root_count * (BLOCK_SIZE/CLICK_SIZE).

5. 使用系统调用 BRK2，通知存储管理进程(MM)，当前系统的 RAM 盘的位置及大小.

6. 通知 Kernel 中的 MEM 任务，RAM 盘的位置和大小.

7. 逐一读入根文件系统至 RAM 盘.

在第 4 步中，我们比较 root_count 和 eram_count，若 eram_count＞＝root_count，则说明系统的扩展内存足以容纳根文件系统. 此时，把 ram_click 置为 0，position 置为 0x10000 (AT 机中的扩展内存始于绝对地址 10000H:0). 这并不是 RAM 盘的实际大小. 当调用 BRK2 时，从 MM 传来的消息中检测到 ram_click＝0，说明 RAM 盘将安装在扩展内存中，于是 MM 不会在主存内分配空间给 RAM 盘. 在汇编子过程中，使用了 AT 机的 15H 号中断调来检测当前系统的扩展内存大小.

接下来，改造 klib88. s 中的 phy_copy 子过程，使之能够在主存与扩展内存之间传送数据.

为了能在实模式(realmode)下实现一个内存块从系统地址空间的任意位置复制到系统地址的另一处，可以使用 AT 机的 BIOS 提供的 15H 号中断调用的 87H 号服务(关于 15H 号中断调用的详细情况，请参看文[2]).

在 phy_copy 子过程中，首先检查源地址和目的地址是否在扩展内存. 若是，则使用 15H 号中断调用来传送数据；否则，用一般的串指令传送.

3　假脱机(SPOOLing)系统的实现

3.1　问题的分析

MINIX 是建立在微机上的，我们不能要求微机有通道结构，因而我们要实现的 SPOOLing 系统和基于通道结构的 SPOOLing 系统有所不同.

具有通道结构的 SPOOLing 系统只是确定某些特定数据或信号(如传送数据地址，打开通道，关闭通道等)，让通道去实现硬件的数据传送. 在我们的 SPOOLing 系统中，可用一个输入或输出进程来完成通道的工作. 当然，这要以占用一定的 CPU 时间为代价.

另外，我们的 SPOOLing 将建立在 MINIX 操作系统之上，而不是作为 MINIX 操作系统的一部分. 这样，输入、输出井将是某些文件，而且这些文件存储在高速块设备中. 在实现方式上，我们准备采用一种与 UNIX 略为不同的方式. 我们希望 SPOOLing 调度进程在没有假脱机任务而被挂起时，由用户提交的输入输出请求来唤醒它，而不是象 UNIX SPOOLing 系统那样定时查询工作目录. 这就需要进程之间能够通信.

3.2　MINIX SPOOLing 系统的实现

图 1 给出了 MINIX SPOOLing 系统的总体结构. 整个系统主要由三部分构成:主控进程 daemon、设备接口进程 spoolin 和 spoolout 以及用户界面 spool.

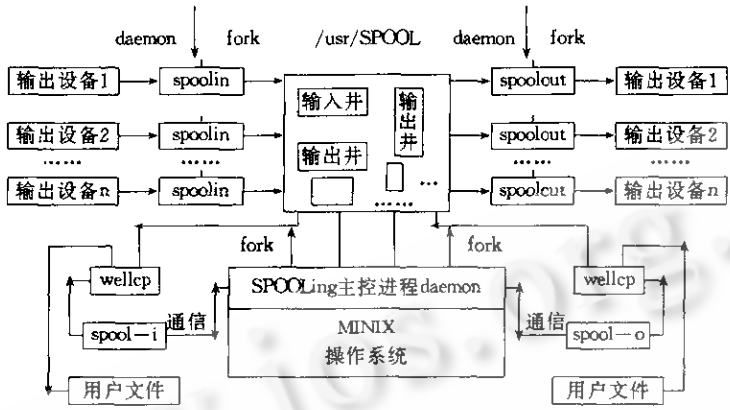

图1 MINIX SPOOLing系统的结构

daemon 进程在加电开启机器后即建立(这一点不难,只要在"/etc/rc"中放入相应的启动命令即可). daemon 进程是整个 SPOOLing 系统的主控进程,它负责管理输入、输出井的调度,接收并应答用户的请求,启动相应的输入(spoolin)、输出(spoolout)进程进行数据传送. spoolin 接收慢速输入设备的数据到由 daemon 指定的输入井中,spoolout 则是将 daemon 指定的输出井输出到慢速输出设备上. 用户界面 spool 进程由用户调用,它负责发出请求给 daemon 进程,从假脱机输入设备读入数据到用户文件,将用户文件输出到假脱机设备上,可查询假脱机设备的状态或用户假脱机任务状态,也可将自己已提交的任务取消. 由于用户没理由在提交任务后去等待用户文件与输入、输出井之间的复制,因而我们还提供了 wellcp 进程,它由 spool 进程建立,将继续完成 spool 发出请求得到应答后的其它工作.

下面给出 MINIX SPOOLing 系统的实现细节.

象 UNIX 那样,MINIX SPOOLing 也将输入、输出井安排在"/usr/SPOOL"中,另外,用"/etc/SPDF"文件来登记假脱机设备的设备名及其性质(如设备类型、设备是否互斥、设备缓冲区大小等). 在 daemon 中,安排了一个设备登记表和三个任务队列(分别为等待、就绪和执行状态队列)的数据结构. 对于整个 SPOOLing 系统,还建立了一消息结构,用于各进程间的通讯.

开机加电后,MINIX 系统启动.

系统初始化过程 init 执行"/etc/rc"的"/etc/daemon&"项而建立 SPOOLing 系统的主控进程 daemon.

daemon 进程建立后首先检查假脱机目录"/usr/SPOOL"是否存在(如不存在,则不建立 SPOOLing 系统),接着读假脱机设备登记文件"/etc/SPDF",对假脱机设备进行登记,并进行系统初始化,然后处于接收用户请求的状态(receive(ANY)).

① 当用户要求从假脱机输入设备读入一些数据到用户文件中时,用户是通过接口文件"/usr/bin/spool"与 SPOOLing 系统打交道的. 具体过程是这样的:spool 进程在分析了用

户的要求后，生成一子进程“/etc/wellcp”，然后退出. wellcp 子进程继承了 spool 的工作，向 daemon 进程发送消息，请求假脱机输入(wellcp 进程要告知 daemon 假脱机设备名等). 当 daemon 进程接到该假脱机输入请求时，先判断假脱机设备名是否正确，若不正确，通知 wellcp 进程可能是设备名错了等等；若正确，则建立一输入井，返回一任务号，并将它直接放到就绪队列中；接着生成一 spoolin 进程，从假脱机输入设备读入数据到指定输入井中，这时任务转入执行队列；最后再通知 wellcp 进程将输入井复制到用户文件中. 这样，一次假脱机输入任务宣告完成.

② 当用户要求将用户文件输出到假脱机输出设备时，同样也是通过 spool 进程分析用户要求再生成 wellcp 进程，由 wellcp 继续去同 daemon 打交道. 所不同的是，当 daemon 进程在接到该请求并判断设备名的正确性后，将该任务放入等待队列，接着建立一输出井，通知 wellcp 进程输出井名和任务号以便让它进行用户文件到输出井的复制，wellcp 进程在接到 daemon 的消息后进行井的复制，复制完后通知 daemon 进程. dameon 进程接到通知后，会将该任务从等待队列中按某种算法放入相应设备的就绪队列. 在输出设备空闲时，daemon 进程会从就绪队列中取出一任务交给子进程 spoolout 去输出(注：对于非互斥设备，则在接到 wellcp 的消息(指输出井填满的消息)后立刻生成子进程 spoolout 去输出)，这时相应任务会从就绪队列调入执行队列. spoolout 将输出井输出到设备上，一次假脱机输出任务完成.

③ 若用户通过 spool 要求查看设备状态或用户任务状态，则相应的过程只是发一消息给 daemon 进程，再等待 daemon 进程发回用户要求的消息，并在接到消息后将这些信息打印出来，其控制过程较为简单.

④ 若用户要求取消一任务，也是通过消息机制通知 daemon 进程，由 daemon 进程来判断用户权限，并在用户有权时从三种队列中找出对应任务将其删除.

4　MINIX 进程对换的实现

4.1　问题的分析

进程对换是实现内存扩充的一种重要技术. 我们在 MINIX 中实现此功能时，力图保持 MINIX 原有的风格，即貌似 UNIX：在实现机制上与 UNIX 基本一致；而在内部结构上，采取更适合于自身环境的一些机制.

在 MINIX 中，操作系统分成三个部件：kernel，MM，FS. 而用于实现进程模型的进程表也分布于三者之间. 这决定了与进程控制息息相关的进程对换系统不可以象 UNIX 那样集中实现于其中一个部件中. 相应于 UNIX 的对换进程，我们在 MINIX 的内核增加了一个对换任务 swap_task. 该任务定时地被时钟任务唤醒，并根据当前各用户进程的状态，依照一定的对换算法，决定哪些进程应该被换入，哪些进程可能被换出. 并据此建立换入进程队列和换出进程队列. 然后，通知 MM 开始进程对换动作. 至此，对换任务已完成其主要工作. 而进程对换的实质：在内存与对换区之间传送进程映象这一关键动作由 MM 完成. 为了做到这一点，我们在 MM 中增加了一个仅供 swap_task 使用的系统调用 DO_SWAP. 该系统调用根据一定的进程对换算法，完成进程对换的工作：试图将换入进程队列上的队列逐一换

入,必要时,从换出进程队列中选其一换出,以腾出内存空间.直至出现以下两种情况:

① 换出进程队列已空;

② 找到应换入的进程,但无足够内存空间,且换出进程队列空或换出动作失败.

至此,该系统调用完成并返回.

图2给出了MINIX对换系统的体系结构.

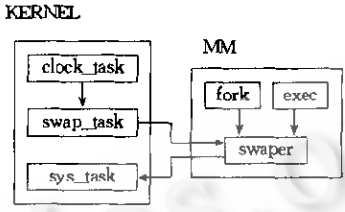

图2 MINIX对换系统的体系结构

4.2 对换设备的管理

在MINIX进程对换系统中,对换设备用来暂存进程在内存的映象.它是一个磁盘中的可配置的块设备.与FS以一个磁盘块为分配单位的块设备管理方式不同,对换系统在对换设备上是以一组连续的磁盘块为单位分配空间的.

因为对换系统管理对换设备的策略不同于FS,故它采用不同于FS的外设管理算法,而是采用与主存管理基本相同的管理策略:类似于主存分配必须是节(click)的整数倍,盘对换区必须是盘块的整数倍.例如,对于4.5K的进程,如果每块1K,就需分配5K的盘对换区.在实现中,我们还采用动态的分配策略,即当进程滚出时才分配盘对换区,而在对换设备的进程映象滚入内存时,释放它的对换区;这与有些操作系统管理对换设备所采用的静态分配策略是不同的.

4.3 进程的换出

在系统运行时,如果MM申请内存而失败,它将换出进程.以下事件可能引起进程的换出:

① 系统调用fork为子进程分配空间.

② 系统调用exec改变自身映象.

③ 为了运行以前被换出,而现在又应被换入的进程.

在发生以上三个事件时,MM的SWAPPER将执行以下进程换出动作:

① 计算要换出的进程大小,并为其分配足够大的盘对换区.

② MM将进程在内存的映象原封不动地拷贝到该进程对换区.

③ 置进程物理地址为0,并通知kernel,该进程已被换出内存.

④ 必要时,释放该进程占用的内存.

在保存进程映象时,MM绕过高速缓存,直接在对换设备和进程地址空间之间传送数据,没有采取文件的形式来保存进程映象.这是为了缩短对换时间,以确保系统的高效率.

4.3.1 系统调用FORK的对换

在系统调用FORK中,当MM找不到足够的内存空间以创建子进程的上下文时,MM

将换出该父进程,但并不释放被(父)进程的内存映象所占据的内存.也就是说以子进程的身份在对换区上复制一个父进程的映象.这与 MM 在内存创建子进程相似,只是此时子进程的映象是在对换设备上而非内存.当换出动作完成后,子进程在对换设备上,MM 将处于对换设备上的子进程置为"换出且就绪"状态.由于子进程处于"换出且就绪"状态,所以 SWAPPER 总会将其换入内存,且核心在内存中总会调度到它,这时子进程完成它的 FORK 系统调用部分然后返回用户态.

4.3.2　系统调用 EXEC 的对换

在系统调用 EXEC 中,当 MM 为新映象分配内存失败时,MM 将在对换设备上创立新映象:首先在对换设备上分配足够的对换区,然后从文件中逐一拷贝代码段,数据段至该进程的对换区,并在对换区上建立标准的进程映象,最后将该进程置为"换出且就绪"状态.至此,进程的新映象已建立在对换设备上.尔后,它将会与其它的被换出的进程一样,被 SWAPPER 换入并投入运行.

4.4　进程的换入

进程的换入是被动的.一个进程被换出后,它将驻留在它的对换区上,直至对换进程在适当的时候将其换入.对换进程,是唯一的将进程从对换设备上换入内存的进程.在 MINIX 中,对换进程分为两个部件:一是 KERNEL 中的对换任务 SWAP_TASK,一是 MM 中的 SWAPPER.SWAP_TASK 实现对换系统的策略,它定时地被时钟任务唤醒,并建立系统当前环境下的换入队列和换出队列.而对换系统的实现机制则由 MM 中的 SWAPPER 体现出来:它总是试图将换入队列中的进程(它的进程映象保存在对换设备上)从对换设备上换入内存.如果需要主存空间,就将换出队列中某些进程换出内存.如此反复,直至它没事可做(换入队列已空)或不能做任何事情(例如,没有进程可被换出).

在 KERNEL 中,由时钟任务 clock 量度一个进程在内存中或被换出后在对换设备上的驻留时间.当 SWAP_TASK 被唤醒之后,它开始着手建立换入队列及换出队列:搜索进程表中的每一项,选出"换出且就绪"状态的进程,按驻留外存时间递减的顺序加入换入队列中,同时选出"在内存中睡眠"的进程按驻留内存的时间递减的顺序加入换出队列,而把"在内存中就绪"的进程加到换出队列中所有"在内存中睡眠"的进程之后.在这里,并不是把所有"换出"的进程都加入换入队列,是因为处于"换出且睡眠"状态的进程,即使被换入内存,它也将继续它的睡眠状态(在内存中睡眠).与其将它换入内存,还不如让它继续驻留在外存,等待就绪机会;而把"在内存中就绪"的进程放在换出队列中所有"在内存中睡眠"的进程之后,理由是"在内存就绪"的进程很可能很快就调度上,我们应该尽可能地让它享受被调度的权利(被换出后的进程不会被调度).为了给在内存的进程以更多的运行机会,克服抖动并增加系统的吞吐量,我们附加了一个规则:一个要换入的进程必须在对换设备上驻留了两个或两个以上的对换周期(对换周期即对换任务被唤醒的周期),而一个就绪进程在换出前至少在内存中驻留了两个或两个以上的对换周期.

4.5　系统的协调

对换系统的引进,迫使我们对原版 MINIX 的某些控制机制,如消息机制、信号处理机制、进程状态及转换机制等作了较大的调整,以使整个系统能协调地运转.限于篇幅,这里不予详述.

结论：通过本文的工作，我们使 MINIX 在保持原有性能的基础上，增加了以下新的性能：

1. 使 MINIX 中可运行的进程数量由 16 个增加到 50 个.

2. 使 MINIX 可支持 360KB、1.2MB、1.44MB 等各种规格的软盘.

3. 使 MINIX 不仅能管理 640KB 的基本内存，也能管理 AT 机的扩展内存.

4. 使 MINIX 的根文件系统不占用基本内存，从而扩大了 MINIX 可运行的用户程序的规模和数量.

5. 使 MINIX 可支持多设备假脱机输入和输出.

经过改造后的 MINIX 较之 MINIX 原版，结构更加清晰，功能更强，可用性更好，辅助工具更加完整，因此它将在我国的操作系统教学和研究中起到重要的作用.

致谢 参加 MINIX 系统改造和扩充工作的还有：李小林、郑则仲、赖毅强、梅坚、黎伟强、黄东斌、陈德坚，在此特致深切的谢意.

参考文献

1 Tanenbaum A S. 操作系统教程. 北京：世界图书出版公司，1989.

2 Hogan T. PC 软硬件技术资料大全. 北京：清华大学出版社，1990.

3 Bach M T. UNIX 操作系统设计. 北京：北京大学出版社，1989.

SOME REFORMATIONS AND EXTENTIONS FOR MINIX OPERATING SYSTEM

Huang Xiangxi

(*Department of Computer Science, Zhongshan University, Guangzhou* 510275)

Abstract Some important reformations and extentions that the authors made for MINIX operating system, a time—sharing、multiuser、multitask operating system developed by A. S. Tanenbaum in 1986 and used for teaching of operating system course, are described in details in this paper, including support for 3.5″ floppy disk, utilization of extended memory, and implementation of process swapping and spooling mechanisms.

Key words Operating system, computer education, MINIX.