计 算 机 操 作 系 统

D·W·巴伦

译者：张尤腊　仲萃豪　彭志松　顾毓清

译 者 的 话

电子计算机自从1946年问世以来，发展异常迅速，目前已进入第四代。与此同时，作为计算机系统重要组成部分的操作系统亦得到飞速发展，目前操作系统已基本上普及化，这对于提高计算机的使用效率和方便用户起了极大的作用。可以预言，随着计算机的进一步发展，操作系统的使用将会更加普及，在性能方面将会进入更为高级的阶段。

我国计算机事业的发展亦很迅速，因此要求操作系统相应跟上。为了适应这一形势的发展，我们翻译了这本书，以供从事这方面工作的同志参考。

本书着重介绍操作系统设计的一般原则。作者从操作系统的内部结构出发，把操作系统分成三个基本组成部分——执行程序、数据管理和作业控制，並分别地详细论述了这三部分内容。本书取材比较全面，並在一定程度上反映了当前的最新水平，这对于我们了解国外操作系统的概貌会有所帮助。

目前操作系统仍在发展，其专用名词日益增多，已有专用名词亦很不统一，译文中对这些专用名词的译法肯定会有不妥之处。为了便于读者对照，我们将其中的某些重要专用名词附于书后，供同志们参阅。

在翻译过程中，我们曾请教过唐稚松、冯康、许孔时、曹东启等同志，这些同志都给予热情的指导与帮助。译稿也有一部分同志阅读过，並提了一些宝贵意见。在此，我们对这些同志表示感谢。

由于我们水平所限，译文中错误和不妥之处肯定不少，敬请读者批评指正。

译 者

1973年12月

目　　录

序　　言

操作系统正成为计算机系统中整个软设备的极重要的一部分，目前许多计算机制造商把主要投资都花费在操作系统的研制上。在那些不懂操作系统的人们中间，普遍认为编制操作系统是一种高级技术。由于这一原因，虽然有关操作系统的文章正开始在杂志上登载，但是，它们大部分都是为初学的人而写的。在程序设计界，许多基本的专业知识还未成文，仅仅以口头资料的形式存在，并且许多技术多半是几个组织各自独立地研究出来的。

当务之急是要改进现有计算机所配备的软设备之质量。Johnson 博士曾说过：今天的许多软件处于这种情况，它们似乎都是在这种假定下做出来的：只要它能运行，就没有人去仔细研究它是怎样编制出来的。

本书的目的，是通过汇集操作系统中现有的大量专门技术知识，来为改进目前所处状态作出一点贡献。任何一个想总结一门飞跃发展的学科的作者，每逢他在过去的历史、现有实际经验以及将来可能的发展这三者之间均衡地取材时，他会感到左右为难，就操作系统来说，这个问题特别尖锐。其原因是，到目前为止，操作系统的发展仍然具有很大的片面性和实用性，所以Dijkstra*对这种发展特点曾作出过这样恰如其分的刻划："耗费巨大，收效不多"。幸而，我们现在已开始清楚地了解到我们对此应做些什么工作，而且也有了一些一般原则，使得能稍微抽象地讨论操作系统的设计。总有一天会有人为操作系统写出一部杰出的著作，就象是Jean Sammett+最近为程序设计语言写的那部著作一样，并且会有人对现有的各系统作出一部综合性的述评。但是这些都不是本书的目的。本书的前二章介绍操作系统一般课题，并扼要地介绍一些典型系统的内部结构。对于那些具有这方面知识的读者，这两章可以略去不读；与此相反，对那些不希望深入一步了解操作系统的读者，就只要读这两章就够了。本书的其他各章试图介绍构成系统设计基础的某些一般原则，这些原则是通过从各种各样系统中取出的例子来加以说明的，然而那一个系统都没有详细介绍。本书除了介绍操作系统中较重要的部分之处，还论述了系统与其用户之间的关系，这一部分是很容易被忽视的，这里所说的用户不仅仅是指程序员，而且还包括计算机装置的管理人员。

这里假定读者已具有计算机及程序设计的有关知识，包括用汇编语言进行程序设计的

---

* 'The Structure of the THE Multi—Programming System'。Commun.A.C.M., 11,341 (1968.5)。

\+ SAMMETT J.E, Programming Languages : History and Fundamentals (Prentice—Hall, 1969)。

经验。因此本书适合于那些从事软设备工作达一定时间的人，或者已学过高年级计算机学课程的人。为本书取材是一件非常麻烦的事，这是一个复杂的问题，要付出巨大的努力。虽然操作系统设计所需的智力常常被低估，但是实际上它是一件艰难的工作。象许多其他课题一样，对于操作系统这一课题，我们也能引用Chaucer的一句话：

“生命是如此短促，而学会一门技术却又多么冗长”。

D·W·巴伦　　1970年

第一章　操作系统的作用

操作系统是大多数大型计算机系统的主要组成部分。除小型机器以外，不通过某类操作系统而直接使用计算机，那是不可能的，而且一个低效的系统对机器的处理能力也有惊人的影响，常常使得昂贵的快速硬件的效果大为降低。此外，由于操作系统是用户和计算机之间的交接面，因此它在设计方面的任何缺点，在程序员的注意下，就能陆续被发现。

操作系统以适应提高中心处理机和外部设备的利用率的要求而发展起来，现在操作系统的主要功能之一仍然是：通过使工作流程自动化，通过按计算机的时间尺度而不是按操作人员的时间尺度来迅速地对系统资源的管理作出决策，使硬件的利用达到最优。然而，随着现代计算机复杂性的增加，操作系统亦给用户提供帮助，例如，按逻辑文件结构来组织磁盘存储器；此外，现代计算机有这样一种趋势：使某些功能，特别是与输入输出有关的功能，一部分由硬件完成，一部分由软件完成〔注〕①，这样，操作系统的功能就进一步扩大了。

在早期的计算中，计算机是“手动操作”的。就是说，操作员（那时往往就是程序员）通过挂上磁带和把卡片装入卡片输入机等办法建立一道作业，然后通过控制台上的操作开关来启动程序。如果程序请求操作员干预，操作员就采取适当的措施，之后再启动程序。最后，当作业完成时，操作员卸下磁带，取下卡片输入机中的卡片和打印机上的结果，然后启动建立下一道作业。这种工作方法仅在建立时间与作业的运行时间比较可以忽略不计时才行得通。早期的计算机非常之慢，所以通常可以这样做，然而随着计算机速度提高，建立时间对运行时间的比率增加到无法接受的百分比，系统就发展成从一道作业到另一道作业的自动过渡。

计算机处理速度的提高也使得中心处理机与输入输出外部设备的速度差距明显地表现出来。改进这种不匹配的努力导致了两种发展，这两种发展对操作系统都有极大的意义。第一个发展是输入输出通道的引进。这是一块硬件，它能控制一台或几台外部设备，一旦它被启动以后，就独立于中心处理机而运行，这样使得输入输出与计算重叠起来。机器提供了一些指令，程序能够凭借这些指令开始通道传送，之后再询问通道传送是否完成。第二个发展是“脱机输入输出”过程的引进。代替计算机直接使用慢速外部设备，从卡片输入机来的输入信息被传送到磁带，程序通过从磁带读出卡片映像获得输入信息。类似地，程序把它的输出信息当作卡片映像或行式映像写到磁带上，之后再被传送到适合的打印机或卡片穿孔机上去输出。这种脱机传送一开始是用专用的硬件实现，然而不久便发现，用一台小型计算机当作“卫星机”更加经济〔注〕②。从原则上说，这两种发展使得有可能获

〔注〕①这样的软件有时取名为固件。

②最著名的组合之一是将小型商用机 IBM1401 用作大型科学用的机器 IBM 7090/94的卫星机。

得高效率的输入输出。主机只和磁带打交道，并且由于是按块从磁带读写信息，所以通道和缓冲区的配合使用使得输入输出几乎完全重叠。但是付出的代价是，设计输入输出程序变得非常困难，程序员要处理缓冲、封锁和解除封锁部件记录，要同步自主通道。这就导致了引进子程序包，通常称它为IOCS（即输入输出控制系统）。这些子程序包承担全部输入输出缓冲和同步，这种方法几乎成了输入输出的标准做法〔注〕。虽然IOCS不是我们所理解的那种操作系统，但它引出了重要的原则：即普通程序员可以把组织输入输出的职责作为一种功能委托给系统。这一原则是所有大型操作系统所固有的。

一旦向磁带脱机输入输出这一原则建立后，作业定序系统的发展道路便敞开了。全部要求就是，把若干道作业（一批）记录在一盘磁带上，在一道程序结束时不停机，把控制转给操作系统（或监督程序），此系统立即启动该批中的下一道作业。（这是经过简化的粗糙叙述，但是就眼前来说足够了。）第一个这样的系统也许是共享操作系统(SOS)[1]，它是公认的FORTRAN监督系统FMS[2]的前躯。由于这样的系统能自动地一道接一道地执行一盘磁带上的各作业，所以通常称它为成批系统。

FMS和它的模仿系统盛行的时间不太长，操作系统发展过程中第二个值得注意的事情——分时就出现了。它的起因是硬件的发展，即中断的概念。我们已经解释了经由自主通道来同步输入输出操作的困难。中断的思想是，通道在完成分配给它的工作时，或者在需要程序干预的某种故障状态发生时，给计算机发信号，这样来排除这一困难。当中断发生时，处理机在保存好工作寄存器的内容后，离开它当前正在执行的程序，进入通道程序，去处理输入输出通道。处理完输入输出工作以后，处理机继续运行被中断的程序。这样，对于一道程序来说，有可能使各外部设备以全速运行，而不会在外部传送过程中经常不断地受到阻碍。

一开始，这种技术是用在单道程序内。然而不久就懂得，任何时候在机器内都存放几道程序，每当一道程序被暂停以等待外部传送时，另一道程序将能使用处理机，这样就使部件得到更好的利用。这称为多道程序。如果受外部设备限制的程序（即程序的执行时间主要由花在输入输出的时间来决定）和受处理机限制的作业一起分时，则多道程序特别有效。显然，需要有一个复杂的操作系统，以便与同时执行中的各程序保持密切的联系。虽然引进这种思想的第一台计算机是Ferranti Orion[3]，可是多道程序的概念在Ferranti（后来叫ICL）Atlas[4]中才被完全采纳。这个方法显然有可能最大限度地利用机器的各部件，此外它还有其它两方面有意义的应用：

（i）在经典的FMS型系统中，卫星机是受外部限制的，而主机大抵是受处理机限制的。这样有一个好处，没有卫星机也行，改之由主机分时给两个活动，这是因为花在处理外部中断的时间仅占整个处理机能力的一小部分。由于没有卫星机，因此就不存在一批作业从卫星机到主机的物理传送问题，因而虽然这类系统往往称为成批系统，最好叫做连续流或作业堆积系统。

（ii）没有什么理由可以认为分时仅限于那些等待输入输出设备的程序。如果有若干

〔注〕至少在美国是这样。当时市场上还没有大型的英国机器，所以没有提出这种要求。这就造成一批程序员不熟悉这种技术的原因，在英国的某些软件中，这种影响仍然是明显的。

道程序同时在机器中，它们按照某种计时基准分享处理机，每道程序依次运行某一时间量，那么它们好象是同时在运行。若每道程序都经由一台联机电传打字机或类似的终端与一个用户相联，则这种做法是相当吸引人的，因为给每个用户这样的印象：只他自己独占该机。这样的系统称为多路系统，显然，系统要确保各程序不相互干涉，并且所有的远程用户获得满意的快速回答，因此系统本身将是相当复杂的。

在这领域中的术语是很混乱的。一般上，“分时”和多路同义，不与远程终端相联的各程序分享处理机叫做多道程序。两者的区别在于实时回答要求。一般说来，在多道程序情况中，一道程序一旦启动后，将一直运行下去，直到它算完为止，或它被暂停以等待外部传送为止（或者，在某些系统中，直到它被一道优先数更高的程序抢占为止）。相反，在多路系统中，经过一定时间以后，为了继续回答其他用户，控制权一定从一道程序转到另一道程序。

我们以后将会看到，分时和多道程序引出资源管理的概念。最极端的例子是有一台计算机，它有M个处理机、N个内存单元和P个磁盘存储单元，Q个程序以这样一种方式共享这些资源：每个用户或程序把该机看成是一台与所有其他用户无关的机器，这台机器既要在允许的短时间内回答该用户打入的命令，又要保证最大限度地利用该机所有部件。

现代操作系统通常提供了IOCS功能，它掌管外部设备、组织缓冲、封锁和解除封锁、处理中断。这一切都是在内部完成的，大大减少普通程序员的麻烦。但是，这样做的必然结果是：用户自己就不能控制外部设备了。为了绝对安全起见，应禁止普通程序员使用外部设备的控制命令。因此，机器硬件应能在两种状态下操作，一种是特权状态，在这种状态下，所有的设备都能用；一种是普通状态，在这种状态下，禁止使用某些设备。

如果委托程序员来控制所有的外部设备，那么他甚至无法引用指定的物理设备。他应改而引用“逻辑”设备，而由操作系统去记录各程序员的逻辑设备和真正的物理设备之间当前的对应关系。因此，在多道程序情况下，几个程序都能引用“卡片输入机0”，而没有任何混乱的危险。这种思想还可以进一步发挥，在另一个极端情况下，我们可以有这样的系统，在这个系统中，程序员编写完全与设备无关的、只引用数据集的程序，在运行时，程序员再给操作系统规定这些数据集与实际设备或文件之间的对应关系。

操作系统必须提供的另外一个功能是程序的控制和管理机构。由于各程序是自动地依次（可能重叠）运行，操作员无法进行干涉来处理意外情况，或者让处于死循环的程序停下来，因此操作系统应能对每道程序实施时间限制，可能还规定打印机的输出信息量。有些完善的系统，还允许程序员提供“事件表”或者“作业说明”，来指出在各种事件发生时应采取的措施。

从用户角度看，操作系统的功能分成两大类：作业控制和数据管理。现在我们就更详细的来研究这些功能。（注意，我们今后论及的是相当大型和完善的系统。）我们把一道作业定义为提交给机器的基本的独立工作单位，即是说，它既不和任何别的工作单位有联系，其完成也不依赖于任何其它单位。一道作业经常（事实上通常）是由诸如编译、联结编辑、运行等等这样一些作业段所组成。操作系统必须允许用户把任意个操作当作若干个作业段联结在一起，并且把信息从一个作业段传递到另一作业段。对用户而言，应能使一些作业段的执行由前面的一些作业段的运行成功与否来决定，甚至根据计算结果规定可能的执行顺序。为了更有普遍性，对用户来说，还应能指明若干作业段能同时执行，并且提

供互锁信息以控制这样的并行运算。用户应能动态地产生新的作业段，这也是很重要的。

在数据管理方面，我们已谈到操作系统在控制外部设备方面的作用。此外，还必须提供一种手段，借助于这种手段把信息按块存储起来和此后收回，而无须详细规定如何使用后备存储器。这一方面是各作业段之间的信息传递，例如来自宏加工程序的输出信息作为输入信息被传给编译程序；另一方面是文件中的信息长期保存，这时系统必须采取必要的措施来保护被存储信息的安全。一个比较好的系统还容许用户编写这样的程序，它根据数据文件运算，而与文件的物理形态无关。这意味着要有共同的内部格式和由操作系统支配的交接程序。

从前面所述中读者对完美的现代操作系统所具有的各种功能已有所了解，也许有一种感觉，筹备这些功能的工作量相当庞大。操作系统还有一个功能尚未谈及，它是起了硬件和软件之间接口的作用。为了进行诸如输入输出、后备存储器管理等这些操作，要提供哪些硬件设备，各计算机之间很不一致，因而为了适应这些设备的操作的需要，操作系统中有一部分专门用作这种管理的。

第二章　操作系统设计简介

我们在这一章中介绍一些简单的操作系统。各个系统的介绍是不太详细的，这样我们就能够有一些篇幅来引进许多思想和概念，它们构成往后将要涉及的更复杂系统的基础。

一个简单的作业堆积系统

我们首先介绍一个最简单的作业堆积系统，熟悉IBM1130监督程序[5]的读者，将能认出这里所介绍的系统非常类似于该IBM系统。我们假设有图1所示的那样一台小型计算机，为的是能够把一系列作业装入卡片输入机，并且使它们顺次执行。假定操作系统如图2所示固定留驻在内存中，而各种编译程序存储在磁盘固定位置上。

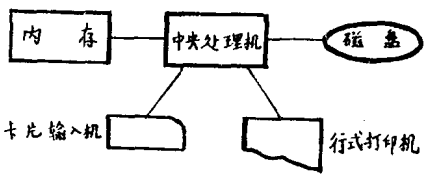

图1　机器构型

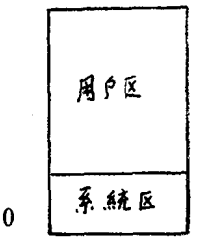

图2　用于操作系统的存储器划分

要求用户除了提供他的作业以外，还提供三张控制卡片。这些卡片的第一列中都穿有字符$，可以根据这个字符来识别它们。第一张卡片是作业卡，它的格式是：

```
$ JOB  代码  编译程序
```

其中“代码”是用户的记账码，“编译程序”是FORTRAN、ASSEMBLER 等等。第二张控制卡片用来把程序和它的数据分开，其格式为：

```
$  ENTRY
```

第三张控制卡片用来表示作业结束，其格式为：

```
$  EOJ
```

当操作系统首先被启动时，它从卡片输入机读卡片，直至遇到$JOB卡片为止（为了跳过先前的程序遗留下来未读的数据卡片，这样做是必要的）。读入$JOB卡片后，把它

在打印机上印出，同时验明该程序要用哪种编译程序。系统有一个表，它列出每种编译程序在磁盘上的位置，系统根据此表把所需之编译程序调到用户区。接着启动该编译程序，编译程序就读程序卡片，编译结果（多半是可再定位的二进制形式）送到磁盘上，同时在打印机上印出。编译完成后，编译程序把控制转给操作系统，然后操作系统寻找另一张控制卡片。在正常情况下找到的这张卡片应是$ENTRY，如果没有编译错误，则操作系统就把装配程序放置在用户区，并且开始装配磁盘上的再定位二进制程序；如果有编译错误，则随即寻找下一张$JOB卡片，并且重复以上过程。$EOJ 的用处在于防止有毛病的程序读过头它的数据末端。运行时，输入输出程序必须检查$EOJ卡片，若企图读此卡片的话，则因出错而强行停止。显然计算机必须永不停机，因此必须修改STOP指令及任何通常导致停机的故障条件，以便把控制转给操作系统。同样，也应使任何PAUSE指令不起作用。

如果所有的源程序都用高级语言编写，则读过头$EOJ卡片的可能性可以通过适当地设计输入输出程序来消除。倘若需要用汇编语言编程序，那末为了绝对安全，就必须由操作系统对所有的输入输出指令进行考查。这就引出由操作系统对程序进行管理的思想，而且也引出程序把一些工作委托给系统的思想，这些工作要是由程序自己来做那是太危险的。从硬件的观点来说，这需要两种操作方式：一种是普通方式，按这种方式操作时某些指令被禁止用；一种是特权方式或系统方式，按这种方式操作时所有的指令都允许用。当汇编语言程序在运行时还有一个危险是，某个有毛病的程序可能冲掉操作系统的某些部分或全部。这种危险通过把系统安置在程序的下面（见图 2 ）而减小到最低程度，这是由于只有负地址才会导致冲掉。但是为了充分安全，还是需要硬件保护。一种简单方案如下。假定操作系统占用存储单元 0 到N—1，那末，在按普通方式操作时，在存储器的存取进行之前，由硬件把所有的存储地址直接加上N。这种硬件再定位使得按普通方式操作的程序不可能冲掉系统。按特权方式操作时，地址不由硬件再定位，因此操作系统可以访问存储器中任何单元。把普通状态改成特权状态的唯一指令是“进入操作系统”指令，所以系统是百分之百的安全。普通和特权状态的概念是还要进一步讨论的中心问题。一旦建立了这两种不同的状态，普通用户程序就非把许多任务委托给系统不可，因此通常给“进入操作系统”指令附上一个参数，因而变成“由于原因n进入操作系统”。

在小型机器中，操作系统所占用的内存空间可能过多地限制了能够运行的程序的大小。只要注意到下面一点，这个问题就能得到缓和：当一个用户程序（或结果程序）正在运行时，内存中只要有操作系统的相应部分以处理输入输出就行，待需要时才把操作系统的其余部分从磁盘调入。于是，只要程序一超出限制，操作系统的引导程序就把该程序的一部分转储到磁盘上，调入它自己的一部分以代替之。这意味着，用于存储器保护的再定位地址不再是固定的，需要把这个地址保存在基址寄存器中，此寄存器的内容仅在特权方式操作时才能更改。

输入输出的重叠

象上面介绍的那样一种系统，排除了作业之间的操作耽误，这样就确保工作源源不断地流入机器。可是处理机的利用率未必始终是高的，因为机器颇多时间是受输入输出限制的（即处理机闲着等待输入输出设备的操作）。目前大部分机器的硬件允许外部传送与计

算重叠工作，这朝着减轻这一问题前进了一步。例如，假定我们的机器有一个自主地运行的外部通道，我们能够询问这个通道，以断定传送是否已完成。那末我们能够这样组织一个程序，它自主地把信息读入若干内存缓冲区，只当所有缓冲区都用空时，该程序才须等待。这种办法稍加修改也可用于输出（参阅Knuth[6]关于缓冲技术的详细说明）。编译程序可以用这些技术，但是除非它编译得特别慢，否则在编译过程中没有足够的计算时间与全部输入时间重叠。用户程序也可以用这些技术，但是有两个缺点：

（a）需要编制复杂的程序；

（b）像FORTRAN这样一些语言决不允许程序员控制输入输出通道。

我们曾讲过多道程序的用法，它可以作为克服这一困难的一种手段。我们回顾一下，这种技术要求有若干道程序同时放在存储器中，因而当其中一道被暂停，以等待输入输出时，就能让另一道接着运行。这类系统要求硬件具有两个特性：一个是中断，外部设备在中断当前程序之前用中断信号通知传送已完成；一个是存储保护系统，用它来保护同时在内存中的各程序不相互干涉。很明显，操作系统中必须提供相当数量的软件来处理这种共享，另外应注意到，这是在完成一种功能，它完全不同于迄今为止所讨论的操作系统功能。为了强调这种区别，以后我们把这块新的软件叫做执行程序〔注〕。

存　储　保　护

执行程序是与其它许多程序一起共享存储器的。（注意，为此操作系统除执行程序外的其余部分都被算作是“其它程序”。）执行程序占用存储器固定位置，其它各程序被调入到存储器余下部分，从头到尾排列，如图3所示。存储器的这种分配方法有时称为分块。为防止相互干涉，硬件提供一对寄存器：基址寄存器B和限长寄存器L。当启动某一程序时，执行程序把B置成该程序头部的绝对地址，把L置成分给该程序的那段存储块的长度（实际上，基址和限长多半是2的若干次方——例如64或128——的倍数）。这两个寄存器只有按特权方式操作的程序才能修改，对于按普通方式操作的程序，它的任何存储地址都受到检查，看它是否小于L，若是，则在该地址被用来存取存储器以前把B的内容加上。于是程序认为存储器地址是从0到L－1，因此，再定位就程序而言被说成是彻底的，因为它不影响程序编写方式或它的编译结果。

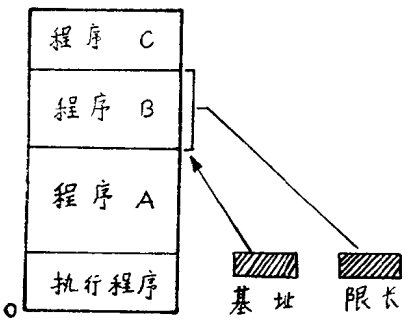

图3　用于多道程序的存储器划分（程序B在执行）

〔注〕ICL1900系列机把管理程序称为执行程序，它这种执行程序除了完成这里定义的执行程序的功能以外，还完成其它方面的功能，例如它花很大部分来控制外部设备，这在别的机器中是用硬件通道实现的，它却用软件实现。

执 行 程 序 的 功 能

执行程序必须完成如下功能：

（ i ）给各程序分配存储器；

（ii）把程序员的外部名变换成物理名；

（iii）把处理机分配给在存储器中的各程序——这是调度问题。

我们现在逐个考察这些功能。

存 储 分 配

每当各道程序进入内存时，它们被安排到存储器的相衔接的各块中。每道程序都必须指出它的存储要求，大于可用空间的程序被拒绝进入。执行程序保存着每道程序的头部绝对地址和该程序的长度，作为每道程序所属信息的一部分。在某道程序完成其任务并被从存储器删去时，就产生一个问题，下一道被调入的程序可能不是恰好和被删去的程序腾出的空位吻合。一个简单的解决办法是，一当某道程序被删去时，把存储器中该程序上面的所有程序往下挪，因而各程序再一次连成一片。新程序总是被安置在所有现有程序的上面（见图 4 ）。这样移动就需要更改执行程序的地址表，并且还要注意，若被移动的程序之一是当前在运行的程序，则只需在再启动它之前适当地调整基址寄存器。（这是非常粗糙的方法，更完善的解决途径在以后各章中介绍。）

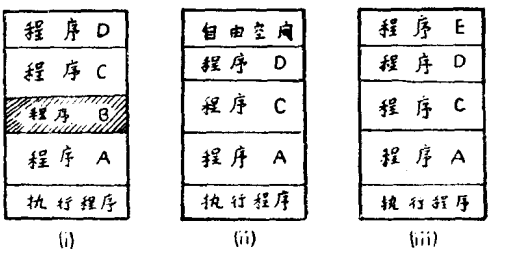

图 4　存储器重新调整

（ i ）准备删去程序B；

（ii）程序C和D下移；

（iii）程序E进入到顶部。

外 部 名 的 变 换

这个问题是简单明了的。执行程序保存一张与程序有关的表，即程序中的外部设备名与真名之间的对照表。所有的外部操作都是在向执行程序提出请求时开始的，执行程序在启动外部通道之前就代之以恰当的物理名。

调 度

这是执行程序中比较难于处理的一部分。程序的有关信息被保存在程序表中。此表包含每个程序的物理地址、长度和一个指示位，指示位有两态：自由和暂停。该表还包括程序名，它的符号个数遵守惯常的规定（通常是一个单元所能容纳的个数）。程序表的登记项

按优先数排列，优先数的指定办法我们在这里无需涉及。

可以用两种办法使执行程序的调度工作开始进行：

（a）用接收外部设备发来的中断信号的办法；

（b）从程序直接进入执行程序的办法。这通常是形式为“暂停我，直到指定的某台外部完成其传送为止”的一种请求。

调度程序的基本原则是，执行处于自由状态的所有程序中优先数最高的那道程序。于是，在执行程序的其它方面的工作做完后，调度程序从优先数高的那一端开始扫描程序表，直到它找到注明为“自由”的某道程序为止，然后启动此程序。这包括适当地置基址、限长寄存器的内容，和把控制计数器再置为该程序上次被暂停时它所到达的地址。这地址可以保存在程序表中，或者按习惯存储在该程序的内存区中，例如，ICL执行程序为此用第8号单元。这程序一旦被启动就一直运行下去，直到它请求执行程序或某种中断发生为止。如果该程序向执行程序请求暂停，它在程序表中的登记项被标明“暂停”。注意，不必把导致暂停的外部设备名记录起来，这是由于每道程序一次只能因为一种原因被暂停。该程序的地址计数器的内容被保存起来，然后进入调度程序以便选择下一次要运行的程序。

如果是某种中断发生，首先要确定中断性质和导致中断的外部设备。我们不谈这个过程的细节，它跟硬件密切相关，现在我们暂时假定，仅有通知传送已完成这一类中断。需要建立程序和外部设备的对应关系，习惯上这是用一个转换表来实现，即对每台外部设备我们都记上当前使用它的程序名。因此，在识别了发中断的设备后，我们就能容易找到有关程序。如果这程序被标明为自由，则不做什么事情；如果它被标明为暂停，则改为自由，然后进入调度程序以便选择要运行的程序。因此如果刚获得自由的程序比较当前运行的程序具有更高的优先数，则该程序优先执行。

很明显，外部设备用得最多的程序应给它最高的优先数。

Spooling 系统

刚才介绍的多道程序，其成功运行有赖于两个条件：适当的不同类型的题目搭配和足够的外部设备。需要不同类型的题目搭配，以使得有足够的计算量来和输入输出重叠；必须有足够的外部设备，以便相当多程序能同时运算。大多数程序需要一台打印机，如果总共只有一台打印机，那末多道程序系统中的程序个数就不可能很多。ICL执行程序试图这样来克服这个困难：允许程序动态地请求和释放外部，使得程序在它有信息要打印之前无需请求行式打印机。可是这不是非常圆满的解决办法，首先这意味着几道程序的输出信息可能混杂出现在一台打印机上，这给操作员带来一个头痛的事情，要把全部输出结果进行分类。可是，多道程序使得Spooling〔注〕已成为非常受欢迎的操作方式。该词的含义是并行的外部联机操作，该词本身是IBM公司首创的，尽管在该名字出现以前这种技术早已存在。Spooling是在使用卫星机这条旧途径的基础上经过修改而得到的最新方案。在熟知的7090/1401系统中，1401把卡片映像传送到磁带，并且把磁带的内容送到打印机，而7090只

〔注〕亦称脱机输入输出或伪脱机输入输出。

是带到带地运行各程序。在Spooling系统中，不用单独的卫星机，代之由主机承担这两种功能〔注〕。这种思想也许是Ferranti Orion操作系统[3]首先引进，Atlas操作系统[4]最先采用。按照Spooling系统当前流行的形式，通常使用磁盘存储器，而不是象早期的系统那样用磁带。一个特别成功的磁盘Spooling系统是IBM的HASP[7]。

我们现在介绍一个有代表性的Spooling系统，它是建立在已讨论过的多道程序的基础上的。它由这样几部分组成：

（1）执行程序，它组织Spooling系统中除执行程序外的全部其它程序的多道程序工作。

（2）输入模，它从卡片输入机读入作业，把卡片映像记录在磁盘文件中。

（3）输出模，它把行式映像从磁盘文件传送到行式打印机。

（4）作业定序模，它调入编译程序、启动和终止作业等。

重要的是要认识到，（2）、（3）、（4）项中的那些模都是普通程序，它们和任何其它程序一样，都是在执行程序控制下运行的。唯一不同的是，它们的优先数比任何用户程序高，因为要不然用户程序可能接管处理机从而停止Spooling的运行。现在有一个完全新的因素要考虑。在简单的多道程序系统中，共享机器的各程序彼此间完全无关，实际上每道程序都不觉得有别的程序存在。而现在我们则要求各程序之间互相通讯，例如，在一个新作业已完全进入时，输入模必须能够把这件事通知作业定序模。Spooling系统的作业定序模本质上与前面所述相同，所以除了与Spooling功能有关的那部分以外，我们不把大量的注意力集中于它。

在进一步介绍之前，我们必须对后备存储器的组织工作做某些假定。假定磁盘存储器被用来保存命名文件，文件目录由执行程序掌管，所以为了存取一个文件，程序得先向管理程序发出请求，要求把命名文件打开，然后再发出请求，要求进行传送。有了这些假定之后，简单的Spooling系统就可以组织如下：

告诉程序员，他的全部输入信息必须按卡片映像的形式写在INP文件中，它的全部输出信息必须以打印机行式映像的形式写到OUT文件里。这些文件名就为该程序员专用，一当程序ABC发出请求，要求打开文件时，比方说打开文件INP时，执行程序就打开文件ABC：INP。（关于编译程序找源程序的方法有点复杂。最容易的办法是这样安排的：当作业调度程序开始一道作业时，它启动编译程序，编译程序冒充该作业在运行，所以编译程序完全和任何别的程序一样读INP文件。）

输　入　模

每道作业以控制卡片“$JOB名字”打头，以控制卡片“$ENDJOB”结尾。只要输入机中有卡片，输入模就自主地运行。在看到卡片 $JOB ABC时，它就打开一个新文件ABC：INP，并连续地把各卡片读入此文件，直至它遇到$ENDJOB卡片为止。（读卡片

〔注〕事实上仅在主机不太快时才这样做。现在又有回头使用卫星机的趋向，但目前采用的是处理机与处理机连接的方式把它与主机直接连起来。参阅第八章。

是根据传送要求进行的，然后请求执行程序暂停输入模，直至传送完成为止，所以输入模大部分时间被暂停。）当读到$ENDJOB时，文件ABC：INP 被关闭，作业ABC被 加进作业表，以等待运行。（作业定序模在它正在选择下一道要运行的作业时也读这个表，这里就有一个通讯问题，因为我们必须保证，两个程序不在同一时刻存取该公用表。这个问题将在以后处理）。如果在输入机中仍然有卡片，那么输入模开始输入另一道作业。

作 业 定 序 模

我们对作业定序模在当前作业刚完成时的情况作一说明。执行模从排队表找出下一次要运行的作业。它打开文件INP，建立文件ABC：OUT，然后从INP文件读出第一张卡片映像。这应是一张指明了要用哪一种编译程序的控制卡片。作业定序模现在按以 前 那 样 运行，编译、联结编辑、运行程序，直到该作业完成为 止。INP文件被擦去，OUT文件被关闭，并且被加到当前正在等待打印的文件表中去。（这里又有一个通讯问题。）然后检查排队表以便找出另一道作业。

输 出 模

输出模接受执行模传递给它的文件，并且打印出来。这是自主地进行的，因此象输入模一样，这个模大部分时间也是暂停的。显然，通过使用几个完全相同的输出模的办法，就可以用多个输出设备进行输出。

模 之 间 的 通 讯

我们已经碰到一种通讯，即共享表的存取。还有另一种通讯问题。我们迄今为止一直假定，三个模全部时间都在运行，实际上并不完全是这样。例如，若没有输出 文 件 要 打印，输出模就自行暂停，并且在下一道作业完成时，执行模必须 通 知 输 出模有信息要打印。类似地，如果没有在等待执行的作业，执行模就暂停，当有作业完全进入时，由输入模再启动它。实现这一点的方法之一是，使暂停模周期性的重复扫描有关表格，直至供给它信息为止。可是这浪费了处理机的处理能力，最好避免之。

为了对付这种情况，我们必须扩充多道程序系统。迄今为止，一道作业或者是自由或者是暂停（等待外部传送）。我们引进一种新的状态，称为挂起。所谓挂起程序就是，它被暂停以等待从别的程序发来信号。于是，输出模在它没有事情做时，将向管 理 程 序 提出“挂起，到被启动为止”的请求。一当作业完成，执行模提出“启动输出模”的请求，如果输出模已在执行，则就不接受这个请求〔注〕。类似的技术也可用来在另一道作业到达时再唤醒执行模，如果它因为没有工作已经自行暂停的话。

现在我们回到同时存取共享信息表的问题上来。当然，对于单个处理机的情况，我们

〔注〕严格地说，在输出模已在执行时并不是完全不接受启动指令，参阅下面。

不可能有真正的同时存取，我们应该防止的是，正当程序A在更改某个表时，此时为了优先照顾程序B而把处理机让给它，程序B立即读该表，结果读出不正确的信息。

让我们具体地考虑Spooling系统中输入模和加工模之间的交接面。假定这两个模之间的通讯桥梁是一个保存在存储器中的计数器，每当输入模完成一道作业的输入时，它把该计数器加1；当加工模从排队站启动一道作业时，该计数器被减1。如果该计数器变成零，加工模自行暂停，在输入模交付另一道作业时被再启动；如果该计数器达到某个规定的最大值，输入模自行暂停（因为排队站满），在加工模从排队站中取走一道作业时被再启动。目前在大多数机器中，对计数器的检查要两条指令：一是把它从存储器取到累加器，一是进行真正的检查。因此有可能发生下面的情况。有时该计数器为零。加工模从存储器取出该计数器中的数，就在这一瞬间发生某种中断，它致使执行程序挂起加工模，启动输入模。输入模发现刚才读进的卡片是“作业的结尾”，所以把内存中的该计数器加1，给加工模发出再启动信号，并且启动读下一道作业。发给加工模的再启动信号没被接受，因为它在执行表中被注明为“自由”。（它当前不是真正在运行，然而这又完全不同于被暂停，暂停意味着，即使有处理机可供使用也不能运行。）不久加工模被再启动，它检查先前已取出的数，发现它为零，就自行暂停，而实际上排队站中已有一道作业正在等待运行。

这是每当各程序通过共享存储器信息来互锁时可能出现的情况的一个特例。最坏时会发生所谓死等的情况，此时每道程序都被暂停，在等待别的程序发来再启动信号。这种情况可以用略为修改执行程序系统的办法来对付。我们给执行程序所管理的每道程序都建立一个标志，称为“唤醒”标志。如果再启动信号是发给非暂停的程序，就设置唤醒标志。当某道程序自行暂停时，如果唤醒标志已设置的话，则立即再启动（唤醒标志随后被再设置）。在上面给出的例子中，在加工模打算暂停时，其唤醒标志被设置，结果它立即被再启动，计数器被再检查，发现它为非零，因此就顺利地继续进行下去。

这是通讯问题的一个特例，该课题在第三章中将广泛地论及。

刚才介绍的Spooling系统是不彻底的，即是说，用户知道它存在，因为他必须编写涉及到INP、OUT文件的程序。在由操作系统（执行程序）应目的程序之请求完成输入输出的系统中，有可能通过以下处理使得Spooling系统是彻底的：系统存取适当的文件以代替指定的外部。如果要求一个程序在不作任何更改的情况下（除控制卡片可能有更改以外），有或者没有Spooling系统都应能运行，则还要有一个更苛刻的限制。这时必须拦截所有的外部指令，即使Spooling系统在用文件模拟传送也必须拦截。这些文件被当作是虚拟输入输出设备。虚拟设备的概念往后大有用处。

外部指令的拦截是下面更一般化的思想的一种具体形式，这种思想是，由一道程序监督或者管理其它程序的运算，在被管程序试图做某些事情时进行干预。在简单的作业定序系统中我们已注意到，一切非法指令、暂停等，都必须被改换成进入操作系统。在我们的更一般的思想中，这些都是事件，它们导致控制转到管理程序。因此就出现了控制级别，用户程序是在委托程序的控制下运行，而委托程序本身又从属于执行程序。于是，程序不再是要么是处于特权状态要么是处于普通状态这样一种简单的两态系统，我们可以有若干个特权等级。这是一个重要特性，因为它允许我们在每一级应用最小特权原则，即一道程序只占有它真正需要的那些特权。