第3章

操作系统的形成和发展

如果想知道操作系统是干什么的，那么现代庞大复杂的操作系统都是应该学习的，但这远远超出人们的认知能力，而且也没必要。我们需要从简单的系统中把握操作系统的核心概念。在电子计算机发展的最初十年中并不存在操作系统，操作系统是计算机系统发展到一定阶段的产物。从早期的计算机开始，沿着操作系统从无到有的过程，探究操作系统形成和发展的轨迹，从而发现操作系统发展的规律，把握其最本质的内涵，这是本章的目的。

本章将从历史的角度阐述操作系统发展过程中形成的一系列基本概念，从系统和用户两种角度介绍支撑操作系统的关键技术，向读者展示一个生动的、多维度视角下的操作系统。当我们回到最简单、原始的系统结构中去，才会发现操作系统最关键的作用和特征，有利于把握系统的发展规律。

3.1 早期的人机交互

在计算机发展的早期，计算机系统的操作自动化程度很低，往往需要操作员完成一些非常基本的工作。操作员必须对计算机系统非常了解，一般人根本玩不了那些庞大而复杂的计算机系统。所以，那时的计算机操作员都是计算机专家，他们往往既是计算机系统的设计者、程序员，也是操作员。像图灵、冯·诺依曼这样的计算机大师，也都花了大量时间泡在计算机机房中，图 3.1 是冯·诺依曼和他研制的计算机。计算机系统往往是边研制边使用，图 3.2 显示了计算机机房的情景。

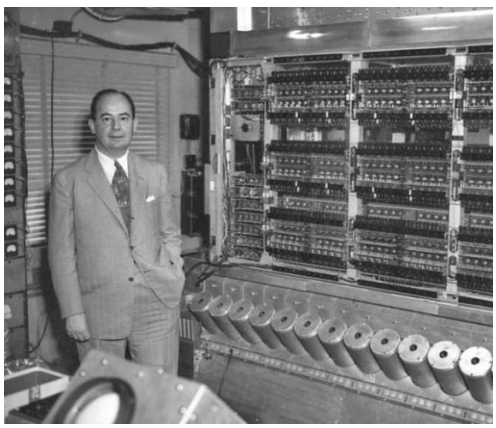

图 3.1　冯·诺依曼和他研制的计算机

图 3.2　早期计算机机房的场景

早期计算机的主要功能都集中在计算上，所有辅助的工作都由操作员来完成。在执行一个用户程序时，操作员可能做的工作如下：①将程序纸带装入纸带输入机，按下按钮启动纸带机；②通过面板开关设置内存地址，装入汇编程序、编译程序或用户程序到内存的某个区域；③从某个地址（程序入口）执行程序；④启动、暂停或结束程序的运行，等等。总之，操作员会根据程序的执行情况操作计算机的控制面板、纸带机、打印机等设备，并及时处理各种意外情况。操作员的工作贯穿程序的整个执行过程，从程序的装入到卸下程序，取走输出结果，以及程序运行过程中各个阶段的控制。

用户提交程序和数据到计算机系统上，以完成一个计算任务，这个任务包含程序、数据及其相应的运行过程，称为**作业**（Job），也就是用户交给计算机系统的一项工作。从系统管理的角度看，作业的具体呈现形式就是程序和数据。就像我们进入商店，商家会习惯地把我们称为顾客而不是人一样，计算机系统也是用“作业”这个词来特指用户提交给系统执行的程序和数据，并包含它是一个任务的意思。“作业”这个词背后隐含系统的一个目标，就是要高效地、尽快地完成好一个作业，就像商家希望和每一位顾客都完成一笔大交易。

用户要完成一个任务可能需要多个步骤，例如，一个用户可能需要在计算机上先编译、连接，然后才能执行程序。所以，一个作业的执行过程往往由多个阶段构成，每个阶段都包含程序的运行，输入或输出数据，这样的阶段称为**作业步**。一个作业步完成后，系统往往需要在操作员的控制下，进入下一个作业步。例如，操作员通过往纸带输入机装入编译程序，决定当前作业步要完成的工作，然后通过某个按钮启动编译程序的执行。同样，当一个作业执行完后，操作员需要取下该作业的所有输出结果，然后准备下一个作业的执行。由此可见，作业内作业步的控制，以及作业之间的切换都是在操作员的控制下完成的，人大量地参与了作业的执行过程，并起到了主导作用，这是早期计算机系统中人机交互的特点。

当操作员装卸纸带或设置控制面板上的开关时，计算机什么事都干不了，处于等待状态。在一个每秒 1 万次的计算机系统中，如果一个用户程序在计算机上的执行时间为 9min，操作员的操作时间为 1min，也就是计算机等待的时间，那么计算机的利用率为 90%。人们，尤其是那些花钱买计算机的人们，对此应该没有什么抱怨。然而，当计算机的速度提高到 10 倍之后，达到每秒 10 万次，上述用户程序的执行时间缩短为 0.9min，而操作时间，即使人们再练习，也不会有大的改变，如表 3.1 所示。此时的计算机利用率为 0.9/(0.9＋

1),小于 50%,这对于耗费巨资建造的计算机系统来说是不可接受的。

表 3.1 CPU 利用率的变化

CPU 速度	总 CPU 时间	总操作时间	CPU 利用率
1 万次/秒	9 分钟	1 分钟	= 90%
10 万次/秒	0.9 分钟	1 分钟	< 50%

由于作业内的各个步骤以及作业之间的切换是由人工操作完成的,CPU 必须等待人的操作,然而与计算机相比,人太慢了。例如,如果编译程序需要读入用户程序进行编译,那么它就要等待操作员首先安装好纸带,然后启动纸带输入机。计算机每天都在进步,而人的操作水平则停滞不前,操作员成为制约系统效率提升的关键因素。

显然,操作员的手工作业控制方式已经很难再继续下去了,计算机系统需要一种更高效的作业控制方式。一台机器,在它的初级阶段,往往需要人类更多的关照,一旦技术上成熟之后,人类自然会退居二线。作业的执行过程摆脱人的控制,实现自动化,这也是计算机系统发展的必然。

3.2 批 处 理

为了提高作业执行过程中作业控制的效率,人们首先想到利用作业的共性改进操作流程。系统管理员将收到的所有要执行的用户作业分类,例如,按照作业所使用的语言,分成 FORTRAN 语言类作业、Cobol 语言类作业、汇编语言类作业。作业被分类之后。系统会先运行某一类作业,如 FORTRAN 类型的作业,从而可以一次性装入 FORTRAN 的编译程序,然后分别装入所有的 FORTRAN 作业进行编译、执行。这样,当某一类语言的程序全部运行完之后,再运行另一类语言的程序。与过去每个作业都要换装相应的编译程序相比,这种集中同类程序的方式省下了大量的编译程序装入时间。上述管理模式的基本思想可以总结为:将相同类型的工作集中起来一起处理,从而简化处理流程,这种管理模式称为**批处理**。

3.2.1 批处理系统

就像现代化工厂中的流水线,批处理思想的核心是抽象出多个任务的共性集中处理,通过简单重复的方式来完成复杂的工作。上述思想可以进一步应用到所有作业的控制过程中。

尽管不同的作业有不同的作业步,每个作业步所完成的任务也不一样,然而,它们之间也存在共性:都需要装入内存才能运行,这是冯·诺依曼计算机的基本特征。这样,可以进一步应用批处理的思想,将作业的装入和执行过程规范化,用一个称为**监控程序**的程序来自动完成,实现一批作业的装入和执行,以代替操作员的工作。监控程序构成了早期操作系统的核心。

在系统启动前,系统管理员会把所有要执行的用户作业装入磁盘或磁带中,然后启动计算机执行监控程序。监控程序运行的基本过程如下:①从磁盘中读入一个用户作业进入内

存并执行；②用户作业运行结束后，判断磁盘上是否还有下一个作业，若有则转①，否则停机。这个装载过程的特点是：通过监控程序完成作业的自动装入，可以连续装载多个作业而无须操作员的干预。该过程由于减去了人工操作时间，因此大大提高了计算机的效率。如果用户程序在执行过程中出错，监控程序会把内存中的程序和数据保存起来（供程序员事后查找错误），然后继续处理下一个作业。图 3.3 展示了监控程序的执行框图。

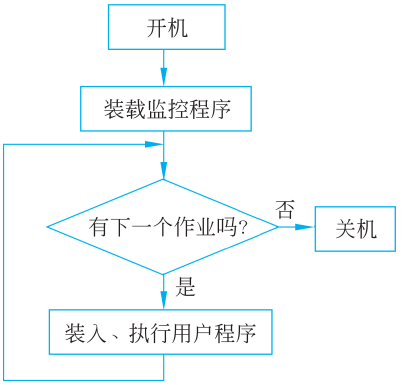

图 3.3　监控程序

监控程序是一个软件，它能够代替人自动地实现作业的装入，控制作业的运行，不能不说是计算机系统管理和计算机软件在技术上的一次跨越。1956 年，通用汽车公司的 Robert L. Patrick 研究了 Gantt 的工业管理学[10]，做了有关吞吐量的实验，提出了连续地、不间断地处理用户作业的设想。随后，通用汽车和北美航空两家公司合作在 IBM 704 计算机上设计了计算机发展史上的第一个操作系统 GM-NAA I/O[11]，实现了上述监控程序的功能，使系统吞吐量增加了 10 倍。这种以批处理技术为核心的操作系统，通常称为**批处理系统**。第一个操作系统的重要意义在于：由软件代替了人工操作，接管了计算机系统中的作业控制。

采用了批处理技术之后，由于大大减少了人的干预，计算机不用等待人的操作，CPU 利用率得到极大的提升。然而，过去由操作员处理的一些程序执行过程中遇到的问题如何解决呢？例如，过去遇上 FORTRAN 编写的程序，操作员会先装入 FORTRAN 编译器，产生目标代码，再装入连接程序，形成可执行程序，然后运行程序。有了批处理系统后，这些工作也可以交给操作系统来做。

操作系统向用户提供一组作业控制命令，允许用户向操作系统发送命令，由操作系统完成某些操作而代替其亲自为之。具体的形式是：用户在提交给系统的作业（程序和数据）中会插入一些作业控制命令，告诉系统如何控制其程序的运行，这些命令采用特殊的标记，不会与程序和数据混淆。在采用卡片记录用户作业的年代，控制命令被存储在具有特殊标记、称为控制卡的卡片上。例如，一个作业的内容如图 3.4 所示，其中有作业控制命令，也有程序和数据。每个作业控制命令以'$'作为开始标记，然后是命令的名字及其参数，其含义分别为：① $JOB 命令，标志一个作业的开始；② $COBOL 命令，请求监控程序装载并执行 Cobol 编译器；③ $LOAD，请求装载编译好的程序；④ $RUN，运行程序；⑤ $END，标识作业结束。

$JOB,…	$COBOL	源程序,…	$LOAD	$RUN	数据,…	$END

图 3.4　一个作业的典型结构

这些控制卡片插入由程序和数据组成的作业流中提交给监控程序，使得监控程序明白用户的意图，能够像操作员一样控制作业的运行。在 IBM 早期的批处理系统中，程序员和监控程序之间以这些控制命令为核心，形成了一种特定的脚本语言——**作业控制语言**（Job Control Language，JCL）。在此之前，操作员是通过按钮、开关来与计算机交互的，采用命令方式是一个大的进步，也导致后来出现了人机交互的命令行方式。

在作业运行时,如果控制命令的执行有问题,程序员是没有机会改的。事实上,程序运行时程序员可能根本就不在现场。例如,如果在编译用户程序的过程中出了错,系统会根据程序员给出的更详细的作业控制命令采取相应的措施,如打印错误信息等。批处理系统虽然运行起来系统效率高,但由于没有交互性,使其应用仅限于某些大容量、高强度的计算和数据处理。像调试程序这样交互性很强的工作要在批处理系统上来完成简直就是一场噩梦。

3.2.2 脱机输入/输出系统

随着 CPU 速度的不断提高,批处理系统用来代替人的操作,大大减少了 CPU 的等待时间。然而,这并没有消除 CPU 的全部等待时间。输入/输出设备在和 CPU 的数据通信过程中,同样远远跟不上 CPU 的速度,拉低了 CPU 利用率,成为 CPU 的另一个“猪队友”。例如,编译程序要从卡片输入机读入包含 1000 张卡片的源程序,需要 100s 的时间。在这组卡片的读入过程中,卡片的送入和数据读出占据了绝大部分时间,相比之下,CPU 将数据存入内存的时间几乎可以忽略不计,也就是说,CPU 在整个输入过程中基本上处于等待状态。数据输入完后,编译 1000 张卡片上的源程序也仅需要几秒钟的时间。同操作员一样,设备速度的提升也远远跟不上 CPU 的发展,CPU 等待输入/输出设备的时间也变得越来越不能接受。

既然设备的速度和 CPU 不在一个量级上,让 CPU 和设备直接通信就很难避免 CPU 的等待。就像处理 CPU 和操作员的关系一样,批处理的思想在这里再一次被应用,形成了脱机输入/输出技术。首先,CPU 要输出数据到设备时,如打印机,并不是直接送到打印机的端口,而是将数据写入一个高速、大容量的外部存储器,如磁带,作为输出缓冲区的磁带往往称为输出井;其次,当存储器上积累了大量的要在打印机上输出的数据之后,由系统管理员将磁带从主机上取下,装入另一个小的计算机上,这台小计算机连接着系统的各种设备,往往称为卫星机;最后,卫星机从磁盘上读出数据,直接送给打印机。同样,如果 CPU 要输入数据时,也要通过卫星机将各作业所需数据集中从输入设备送入磁带,然后由系统管理员把磁带从卫星机上卸下,装入主机,将数据读入与主机相连的磁带中,作为输入缓冲区的磁带往往称为输入井。在主机上运行的作业需要输入数据时,由系统到输入井中取作业的数据。输入/输出设备与主机不直接相连,真正的输入/输出操作是在脱离主机的情况下完成的,所以这种输入/输出方式称为**脱机输入/输出**。图 3.5 显示了主机和设备直接相连的情况,图 3.6 显示了采用脱机输入/输出方式后主机、卫星机和设备之间的关系。

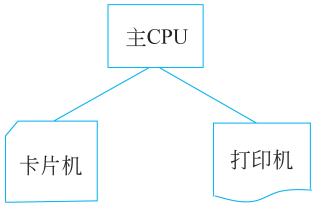

图 3.5 联机输入/输出

系统采用脱机输入/输出方式,实际的输入/输出任务是在卫星机的直接控制下完成的,由于卫星机本身速度慢、价格低,即使卫星机等待设备,对整个系统的资源利用率影响不大。主机不再控制设备,可以把时间主要用于计算上,所以大大提高了主 CPU 的利用率。脱机输入/输出将所有的输入/输出数据集中存放在磁带上之后再一起输入/输出,而不是随着程序的执行随时输入/输出,这是批处理思想的进一步应用。要注意的是,尽管 CPU 不直接控制打印机、卡片机这样的低速输入/输出设备,但是 CPU 在将输出数据送到磁带,或从磁

带读入输入数据时，仍然需要对磁带进行输入/输出操作，CPU 仍然需要等待磁带 I/O 完成，只不过由于磁带的速度相比于其他的设备较快，等待的时间会很短。

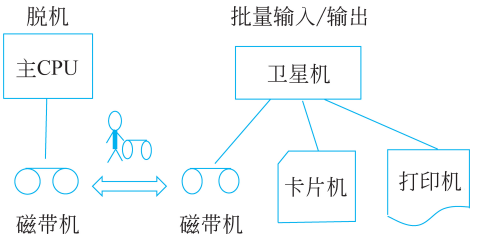

图 3.6　脱机输入/输出

卫星机控制输入/输出，也会面对输入/输出设备速度慢的问题。不过，由于卫星机本身功能弱、速度慢、价格低，这个问题已经可以忽略了。

批处理技术应用在作业管理和输入/输出操作，解决了高速的 CPU 和低速的操作员、设备之间的矛盾，大大提高了 CPU 的利用率，这是计算机系统发展到一定阶段的结果。从应用方面讲，一定存在大量的作业、大量的输入/输出数据需要处理，才会有对批处理技术的需求；从技术方面讲，软件的能力得到进一步的发挥，用软件来管理软件的运行过程，实现了系统管理的自动化，这是操作系统软件最原始的功能。需要指出的是，批处理技术在提高系统自动化的同时，也限制人机交互，操作员没法参与到批处理的过程中，导致大量的应用不能在批处理系统中运行，例如，文本编辑、游戏、程序调试。

3.3　多　任　务

随着电子技术的发展，CPU 的运算速度越来越快，尤其是集成电路技术的出现，使 CPU 运算速度按摩尔定律保持了几十年的高速发展。然而，在作业的运行过程中，难免要进行输入/输出或人机交互，会极大地拉低 CPU 的利用率。虽然批处理技术大大减少了 CPU 的等待时间，但很多交互型的作业或实时任务并不适合采用批处理技术。批处理技术解决了作业交替过程中的 CPU 等待问题；脱机输入/输出技术解决了作业执行过程中等待设备输入/输出的问题；然而，交互型作业执行过程中 CPU 大量时间等待人的问题尚未解决。在 CPU 性能快速提高的同时，如何提高 CPU 的利用率，仍然是计算机系统发展的关键问题。

本节分别从原理、实现和应用三个层面上阐述多任务技术，介绍早期的操作系统如何通过多任务技术提高 CPU 利用率。

3.3.1　并发与并行

批处理技术将人和外设从主机系统中分离出来，提高了 CPU 的利用率，但它并没有杜绝 CPU 等待人和外设的问题。即使采用脱机输入/输出方式，CPU 仍然需要读写磁盘或磁带，由于 CPU 的速度和它们不在一个量级上，所以 CPU 还是要等待，而且这对于高速 CPU 来说，仍然是不可接受的。另外，许多计算机应用不能没有人机交互，如游戏、文字编辑等，在这样的系统中，CPU 也无法避免等待。总之，在作业的执行过程中，CPU 等待设备或人是不可避免的。换一种思路，能不能在 CPU 空闲时，给 CPU 安排其他的工作，以提高 CPU

的利用率呢？这正是现在要讨论的问题。

日常生活中，人们经常会遇到不可避免的等待。例如，开车到路口遇上红灯，此时司机可以保持注视信号灯，也可以去观望一下路上的行人、街上的风景，听听音乐来打磨时光。再如，煮面条时，需要等待水烧开，人们此时可以去看书或玩游戏。可见，当我们处于等待状态时，一般都会找另外一件事来做，保持大脑的利用率，很少有人会让大脑保持静息状态。当然，CPU 也可以这么干，当它在执行一个任务 A 的过程中，等待输入/输出操作时，完全可以去执行另一个任务 B 的计算。这里的**任务**(Task)就是计算机系统中完成的一项工作，通常情况下，它和作业的概念没有本质的区别，也是指程序的一次运行过程。在计算机的发展过程中，不同时期、不同领域的研发人员往往使用了不同的词汇来描述同一个概念。

在一个系统中，同时有两个或两个以上的任务在运行，这种任务的运行方式称为**多任务(Multitasking)**。关于两个任务的“同时运行”，人们并没有对它进行严格而详细的描述，所以它包含多种运行方式。例如，日常生活中，我们打开洗衣机开始洗衣服，然后淘好米，启动电饭煲煮饭，洗衣机和电饭煲开始以互不干扰的方式工作。另外，在朋友聚会时也经常是一边吃饭一边聊天，吃饭和聊天也被认为是同时进行的。但这里的“同时”和上例中的“同时”有所不同。吃饭和聊天都用到嘴，二者之间是会相互干扰的。说两个任务是同时执行的，仅是一种宏观上的描述，微观上它们可能是交替的，也可能是并行的。在上面两个例子中，从微观上讲，前者是并行运行的，后者是交替运行的，但从宏观上都可被看作同时进行的任务。两个任务**同时**执行，可以描述为：一个任务还没有结束，另一个任务就开始了，或者说，两个任务从开始到结束的总的运行周期在时间轴上有重叠。

如图 3.7 所示，任务 A 在[t1，t2]时间段上在 CPU 上执行，在[t2，t3]时间段内任务 A 在打印机上输出，此时任务 A 不能在 CPU 上运行，CPU 可以去执行任务 B。在 t3 时刻任务 A 完成了打印，CPU 可以在[t3，t4]时间段上继续执行。执行一段时间[t4，t5]后，任务 A 又要在下一个时间段[t4，t5]执行打印输出，CPU 又可以去执行任务 B 了。按照前面描述的关于“同时”的内涵，任务 A 和任务 B 是同时运行的。实际上，在任何一个时刻，只能有一个任务在 CPU 上运行，微观上看，两个任务在 CPU 上交替执行；宏观上看，两个任务在同时运行，这种任务的运行方式称为**并发**。采用并发运行方式，可以保证 CPU 能够最大程度地被利用。

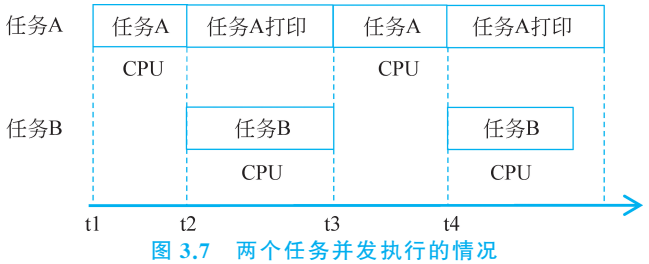

图 3.7　两个任务并发执行的情况

如果一个系统中包含多个 CPU 或多核，可能会采用另一种方式运行两个任务，如图 3.8 所示。[t1，t3]时间段上任务 A 在 CPU1 上运行，[t2，t4]时间段上任务 B 在 CPU2 上运行，其中，t1＜t2＜t3＜t4。由于[t1，t3]和[t2，t4]存在重叠，按照前面关于“同时运行”的描述，这两个任务也属于同时运行，但这种运行方式和并发不一样。不论是在微观上还是在宏观

上看，CPU 都不是交替地执行了两个任务，而是两个同时执行的任务，也就是说，在任何一个时刻，这两个任务都在同时运行，这种同时运行两个任务的方式称为**并行**。

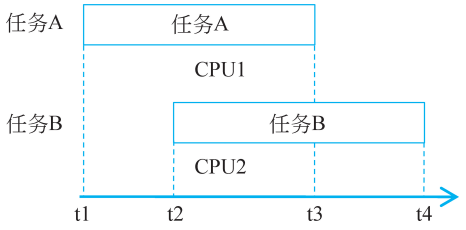

图 3.8　两个任务并行执行的情况

并发和并行是多任务的两种执行方式，除了以上阐述的二者的不同之外，它们也有共性。不妨假设每个任务都是由若干个具有严格顺序的操作组成，在每个任务内部，所有的操作都有严格的顺序，程序中的操作就是这样的。但是在两个任务之间，无论是并发还是并行，都没有规定不同任务中的两个操作之间的顺序，二者是等价的。从系统的角度看，可以用两台 CPU 并行地执行两个任务，也可以用一台 CPU 交替地执行两个任务，最后的执行结果都是符合要求的，这就是并发和并行的共性。并发和并行在逻辑上没有什么区别，其区别体现在具体的实现上。

早期的计算机系统以单 CPU 为主，现在的计算机系统则以多 CPU、多核为主，所以系统中多任务的运行方式既包含并发，也包含并行，这也构成了目前各种虚拟化技术的基础。程序员可以充分利用操作系统提供的多任务运行环境，能够大大提高程序运行的性能，改善用户体验。

当我们在讨论两个 CPU 并行执行时，并没有对它们的执行速度有任何限定，哪怕是其中一个 CPU 暂停一段时间，也不应从逻辑上影响系统的运行结果。所以用一个 CPU 的并发运行方式可以在逻辑上模拟两个 CPU 的并行运行的执行结果，反之亦然。所以，本书讨论的多任务技术，大部分情况下都可以忽略并发与并行的差异，所以如果不特别说明，都假定程序的执行环境是单 CPU 单核的情况，也就是说，多任务都是以并发方式运行的。

3.3.2　多任务的实现

在一个计算机系统中同时运行多个任务，每个任务都会用到 CPU、内存和设备，如何协调多个任务共享系统中的这些资源，正是操作系统的主要功能。本节简单概述实现这些功能的关键思路，加深对操作系统内涵的理解，后面各章将会详细阐述实现的具体机制。

1. 共享 CPU

为了让 CPU 总是有事可做，系统需要为 CPU 准备多个可以执行的任务。当一个任务执行输入/输出操作时，CPU 可以转去执行其他的任务。当有多个任务可以执行时，系统需要为 CPU 选择一个最合适的任务来执行，该项工作称为 **CPU 调度**。在任何系统中，调度都是非常重要的工作。例如，警察在十字路口指挥交通时，主要工作是决定哪辆车先走，哪辆车后走。指挥好了，路口的通过率就高，否则可能会引起交通阻塞。同样，在计算机操作系统中有一个调度程序模块专门负责 CPU 调度的工作，以提高计算机系统的吞吐量。一旦 CPU 处于空闲状态，就去执行调度程序，寻找下一个要执行的任务。

一个任务执行过程中用到 CPU、内存和设备，这构成了该任务的执行环境，当另一个任

务运行时,不应该破坏前一个任务的执行环境,否则前一个任务就不能继续执行了。一般来说,操作系统会采用相应的管理措施(见内存管理和设备管理两章)保证分配给一个任务的内存和设备在其未用完之前不被另一个任务所使用,让它们保持任务暂停时的样子。但是,CPU 是唯一的,必须提供给下一个任务使用。系统采用的方法是:保存当前任务的 CPU 状态到内存中,在该任务下次执行之前再恢复 CPU 的状态。**CPU 状态**由 CPU 中寄存器的内容组成,这些寄存器包括通用寄存器、程序计数器、栈指针寄存器等。另外,有些设备,如中断控制器,像 CPU 一样,也是所有任务都需要使用的。因此,中断控制器的状态,例如,其中中断屏蔽字寄存器内容也应该像 CPU 的寄存器内容一样保存起来。

在 CPU 调度程序选中一个任务后,紧跟着要做的事情就是:保存当前任务的上下文,恢复选中任务的上下文,这个过程称为**上下文切换**。每个暂停执行的任务的上下文都保存在和任务相关的一个特定区域,以备恢复运行时使用。CPU 调度和任务的上下文切换都是操作系统的核心功能。完成这些功能本身也需要占用一定的 CPU 时间,是花费在管理上的代价,属于系统开销,系统总是尽量减少 CPU 调度和上下文切换所花费的时间。

在 2.5.1 节中,讨论中断处理时曾介绍过中断处理程序和被中断程序之间存在上下文切换,和现在所讲的任务之间的上下文切换概念是完全一致的。中断处理程序可以理解为不同于当前任务的另一个任务。

2. 共享内存

和 CPU 不一样的是,计算机内存可以划分成不同的区域供不同的程序使用,在早期的计算机系统中一种简单的内存分配方案如图 3.9 所示。内存被操作系统和各个任务分享,每个任务在内存中的位置由起始地址和结束地址(也可以是任务所占空间大小)来界定。

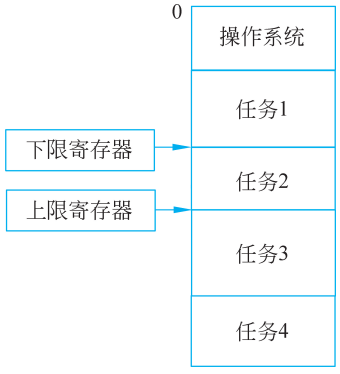

图 3.9 多任务系统中的内存布局

操作系统负责整个内存的管理,包括记录各任务在内存中的位置以及内存中的空闲区域,并为每个任务分配内存空间,任务结束后还要回收内存。多个任务放在同一个内存空间中,任何一个程序都有可能有意或无意地破坏另一个程序,为此,操作系统在将 CPU 交给将要执行的任务之前,一定会为该任务设定访问内存的上下限,防止它破坏其他任务的内存空间,详情参考 1.4.2 节,更高级的管理技术将在第 5 章中介绍。

3. 共享设备

在多任务系统中,各个任务在使用设备时也存在设备共享的问题。例如,当一个任务要使用打印机输出数据时,CPU 转去执行另一个任务,而后一个任务也想使用打印机怎么办?显然,系统不会允许后一个任务再去启动打印机,否则打印出来的数据就混在一起了。为此,操作系统承担起管理系统中所有设备的责任。这主要是通过系统在以下三方面的工作实现的。首先,系统禁止应用程序直接使用设备,只有操作系统才能访问和控制设备,在指令集架构设计时就考虑到了这个问题,输入/输出指令一般都属于特权指令;其次,操作系统负责设备的管理,记录系统中设备的使用情况,例如,哪些设备是空闲的,哪些设备正在为哪些任务服务等;然后,要求所有应用程序在使用设备时必须调用操作系统的程序来实现。通

过以上三方面的工作，可以保证在多任务环境下各任务合理使用设备。

4. 中断机制

当一个任务执行输入/输出操作时，需要让出 CPU，只有等输入/输出操作完成之后，才有继续在 CPU 上执行的资格，那么系统如何知道任务 A 的输入/输出操作是否结束了呢？显然，这只能依靠设备的中断信号！有了中断机制，任务 A 的输入/输出操作完成后，设备向 CPU 发出中断信号，激活属于操作系统的中断处理程序，这实际上是通知操作系统该任务的输入/输出操作已经完成，可以恢复在 CPU 上执行了。这样，该任务才有机会被操作系统再次调度并执行。

如果没有中断机制，当 CPU 从一个任务 A 转去执行另一个任务 B 后，操作系统就没有机会获知任务 A 的执行情况。这样，当任务 A 的输入/输出已经完成，也不可能及时恢复该任务的执行。中断是基于硬件的机制，通过中断，操作系统才能及时获得 CPU 的控制权，并及时了解系统中的各种变化。没有中断机制，实现多任务要困难得多。

5. 总结

由于采用了多任务技术，CPU 的利用率得到大幅提升。事实上，内存、设备的利用率也得到大幅提升，CPU 和设备得以并行工作。多任务技术要求计算机系统硬件和操作系统功能更强大、更复杂，促进了计算机系统本身的发展。

通过对系统的资源划分，例如，将 CPU 分成不同的时间段，将内存分成不同的区域，如图 3.10 所示，各任务之间在共享资源的同时，也不会相互干扰，实现了系统资源的共享和保护。系统中运行的每个任务甚至感知不到其他任务的存在，好像系统资源全部归自己使用。

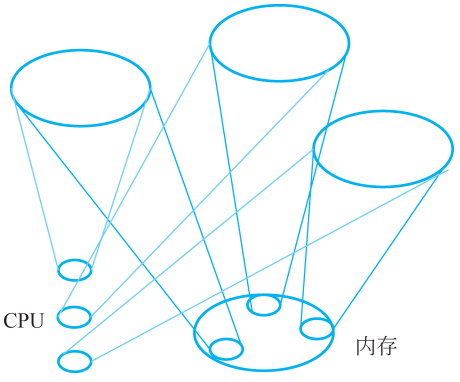

图 3.10　为每个任务提供一个虚拟的运行环境

第一个多任务系统出现于 1961 年之前，是英国 J.Lyons &Co.Ltd.公司的 LEO Ⅲ计算机系统[12]。该系统在内存中同时保存多个任务，任务有不同的优先权，按优先权运行各个任务；系统支持具有优先级、中断屏蔽的中断机制；在硬件方面，该系统已经完全采用了半导体材料，并且使用了微程序控制技术。

3.3.3　分时系统

在单 CPU 系统中，多个不同的任务分享 CPU 的不同时间段，本质上是一种分时技术。采用多任务之后，CPU 不仅可以直接控制慢速的设备，而且保持高效运行。那么对于 CPU 的另一个慢速的合作伙伴——操作员，CPU 能不能做到直接和操作员交互，而且保持高效运行呢？答案是肯定的，这就是分时系统。下面将从原理、机制和策略三个层面阐述分时系统。

1. 多任务和人机交互

在多任务系统中，以 CPU 为中心来安排任务运行的，目标就是尽可能让 CPU 处于工作状态，这样导致的结果就是很多任务可能处于等待 CPU 的状态，毕竟 CPU 不再是专门

为一个任务服务了,CPU 忙不过来,可能会让某些任务或长或短地等待,这对于一般的计算任务并不是什么问题。然而,对于交互性任务,让任务等待 CPU 来处理,意味着让操作员等待,等待时间长了,人就受不了了。例如,你点了一下手机屏幕,等了几秒后结果还出不来,心情会如何呢?

所以,尽管人的操作比设备还慢,在使用多任务技术时,可以让一般的输入/输出任务等待,但不能让人等待,否则会被关机的。一般来讲,人等待机器的耐心也就几秒,而且随着技术的进步,人的耐心越来越差。玩游戏时,机器的一点点卡顿,都会带来不愉悦的感受。操作员从提交输入信息到系统回送结果的这段时间称为系统的**响应时间**。响应时间描述了系统响应用户需求的及时性,一个友好的系统应该将响应时间控制在用户能够接受的范围之内。如果用户提交的任务所需的处理时间很长,不能在几秒内完成怎么办呢?例如,用户要备份一组很大的文件。此时程序可以通过回送中间处理进度的方式,告诉用户工作正在进行,并没有死机。在桌面系统中经常见到的进度条就是起到了这样的作用。

总之,对于人机交互类型的任务,系统既要提高 CPU 利用率,给 CPU 准备足够多的任务,不让 CPU 等待,同时又不能让操作员明显感到在等待机器。

2. 交互型任务的实现机制

如果 CPU 一门心思地去执行某个任务,直到该任务遇到输入/输出操作,那么 CPU 就有可能长时间执行不到其他任务,这种执行方式对于交互型任务是不能接受的。在包含交互型任务的系统中,操作系统应该做的是定期去关照每一个交互型任务。图 3.11 显示了玩转盘子的杂技,演员同时转动多个盘子,他必须周期性地为每个盘子加速,如果超时,就玩砸了。演员自然有一套周期性地关照每一个盘子的办法。

图 3.11 周期性地为每个盘子加速

那么 CPU 能在一个任务的执行过程中想到要执行其他的任务吗?显然 CPU 自身是做不到这一点的,只能依赖 CPU 之外的机制,这就是中断。一般的中断信号是没有规律的,只有时钟中断是周期性的,起到激活操作系统、调度 CPU 去执行其他任务的作用。每次时钟中断,操作系统的中断处理程序被执行,检验一下是否需要将 CPU 转去执行其他的交互任务,以保证所有的交互任务都能得到及时的处理。

假设系统中仅有 n 个交互型任务,交互型任务的响应时间为 rt,那么 CPU 应该在时间 rt 内为每一个任务服务一次,让用户感觉到其任务正在 CPU 上运行。在一个 rt 的时间周期内,每个任务获得的 CPU 运行时间称为**时间片**。平均来说,时间片的大小为 rt/n(这里忽略了上下文切换时间)。如果规定响应时间为 1s,每个任务获得的时间片为 20ms,那么系统可以支持 50 个交互型任务同时运行。

依靠时钟中断,周期性地执行交互型的任务,对用户请求能够做出及时的响应,这样的多任务操作系统称为**分时系统**。

3. 分时系统考虑的因素

分时系统是直接与用户交互的,所以响应时间是衡量分时系统性能的关键指标。将所有任务轮流执行一次的轮转周期可以表示为

$$T = n \times q \tag{3.1}$$

其中，n 为用户数量，q 为时间片大小。假设 CPU 的速度极快，用户提出的请求都能在一个时间片内完成，也就是在 T 内所有的用户都会得到一次响应，所以 T 也可被看作系统的响应时间。然而，当 T 限定后，n 越大，q 越小，所以当系统中任务的数量过多时，时间片有可能很小，导致 CPU 在一个时间片内不能完成某些用户提交的请求，从而延迟了对用户的响应。在桌面计算机上运行的应用程序太多时，就会出现应用程序反应迟缓的现象；某些网站被短时间内大量访问，也会没有响应，都是系统中任务太多的极端例子。

在一个轮转周期 T 内会发生 n 次任务之间的上下文切换，上下文切换也是耗费时间的，所以时间段 T 并没有全部用于执行用户任务。假设每次上下文切换时间为 cs，式(3.1)就要修改为

$$T = n \times (q + \mathrm{cs}) \tag{3.2}$$

即 $q = T/n - \mathrm{cs}$，注意这里响应时间和时间片不再是一种正比例关系。当用户任务很多时，系统忙于上下文切换，就剩不下多少时间执行用户任务了。可以想象，一个杂技演员可以轻松玩转 5 个盘子，10 个盘子时仅可以勉强玩转，20 个盘子时就不可能玩起来了。在一般的多任务系统中，一个任务遇到输入/输出时才会发生上下文切换，但在分时系统中，上下文切换更加频繁，系统开销显著。

4. 分时技术的意义和应用

和一般的多任务系统相比，分时系统面向人机交互型的任务，任务调度采用了定时轮流执行的方式。

一般的多任务系统是不支持像文字编辑、程序调试、游戏等大量的交互应用的，只有分时系统为这类应用提供了运行的环境。

历史上最早的分时系统是 CTSS(Compatible Time Sharing System)，1961 年在麻省理工学院研制成功。CTSS 名字中“Compatible”表示兼容，指明该系统还具有批处理系统功能。在完成前台的交互型作业的请求之后，CPU 可以去执行后台的批处理作业。该系统由约翰·麦卡锡(John McCarthy，人工智能之父，见图 3.12)提出，具体在费尔南多·考巴托(Fenando Jose Corbato，见图 3.13)领导下完成。另外，考巴托还领导研制了 CTSS 之后另一个著名的操作系统 Multics。麦卡锡因在人工智能领域的贡献而在 1971 年获得计算机界的最高奖项图灵奖。考巴托因为研制分时操作系统 CTSS 和 Multics 获得 1990 年的图灵奖。在研制 CTSS 的过程中，为了保护用户信息，考巴托在系统中要求用户输入密码登录以验证身份，被广泛地认为是最早的计算机安全机制之一。

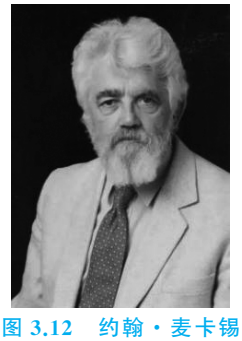

图 3.12　约翰·麦卡锡

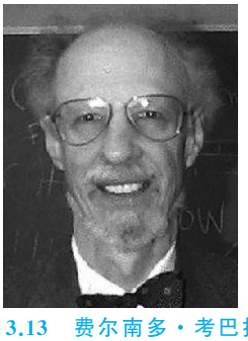

图 3.13　费尔南多·考巴托

3.4 操作系统的概念

操作系统作为一个相对独立的软件系统已经存在 60 多年的时间,随着技术的不断更新,操作系统自身以及它的运行环境都发生了巨大的变化。本节将从时间和空间两个维度探讨操作系统的发展与变迁,阐述操作系统的本质内涵。

3.4.1 操作系统概念的形成

装入程序、监控程序都可被看作操作系统最原始的雏形,它们都是一些为应用程序提供服务的程序,只有这些软件在采用了批处理技术、多任务技术之后,它们才发展成为一个相对独立的软件,形成了操作系统的概念。下面来回顾操作系统概念的形成过程。

1. 批处理技术

在批处理系统中,监控程序实现了作业运行过程的自动控制,其主要工作就是自动地装入各种程序,包括编译程序、连接程序和用户程序,然后执行它们。这说明监控程序不同于一般的程序,像装入程序一样,是一个管理程序的程序。

监控程序需要调用装入程序,比装入程序的功能又提升了一步,是批处理系统的核心,可被看作操作系统的雏形。它代替操作员的人工操作,大大提高了操作效率,这也许是为什么后来被称为操作系统(Operating System)的原因。

历史上第一个操作系统 GM-NAA I/O 就实现了以监控程序为核心的批处理系统的功能。

2. 为用户程序提供服务

程序的运行本来是很简单的事情,不需要什么服务,就像早期人类的农耕生活,自给自足。但是,程序员们很快发现各种程序中往往包含着一些相同的功能,子程序可以简化程序设计、实现程序共享,紧接着程序库出现了。早期的程序库,就像农民自家的余粮,仅给自己人使用。程序员设计新程序时会引用自家程序库中的程序并将之连接进来,形成可执行程序,这和现在使用的私有库没有什么区别。随着软件产业的进一步发展,不同的程序员之间需要共享的代码越来越多,各种公有的程序库出现了。例如,现在的 C 语言、Java 等都有被广泛使用的程序库。这些共有的程序库都是面向特定范围的程序,如 OpenGL 面向绘图程序,C 库面向 C 程序。

在同一台计算机上编程的程序员自然都会对该计算机相关的程序库感兴趣,最典型的莫过于控制计算机设备的输入/输出程序库了。输入/输出程序涉及设备的控制方式、各种参数,以及设备的控制命令,很少有程序员会有喜欢编写这样的程序,所以才有了公共的输入/输出程序库。逐渐地,几乎所有的程序员都离不开这样的程序库了,因为没有程序既不输入也不输出。

于是,系统为用户程序服务的机会来了。操作系统 GM-NAA I/O 就包含输入/输出程序库,这可以从该系统的名字上看出来。操作系统中的输入/输出程序库享有操作系统作为管理者的特权,可以常驻内存,用户程序在运行过程中可以直接调用。就像在高速公路上旅行,可以在服务区的餐厅吃饭,也可以自带干粮,前者相当于享受操作系统提供的服务,后者则是用户程序自带的私有程序库。可见,对于应用程序而言,操作系统从一开始就是一个服

务者。

3. 多任务

操作系统为应用程序提供输入/输出程序库这样的服务，是最简单的功能，接下来操作系统就利用其系统管理者的身份为应用程序提供更广泛的服务。

在多任务系统中，操作系统利用其对系统资源的管理和控制，为每个任务提供了一个运行环境，每个任务都认为自己独占系统的全部资源。多任务系统中对 CPU、内存和设备的管理真正考验了操作系统软件的功能，形成了操作系统特定的核心技术。从 GM-NAA I/O，到 LEO Ⅲ、CTSS 和 Multics，标志着操作系统技术从起步到成熟，操作系统概念的内涵逐渐形成。如果说批处理技术催生了操作系统，那么多任务技术则赋予了操作系统灵魂。

4. 操作系统的描述

操作系统(Operating System)是提高计算机资源利用率，为应用程序提供方便的运行环境，为用户提供友好操作界面的系统资源管理软件。

上述定义中说明了操作系统和计算机资源、用户、应用程序之间的关系，操作系统的核心功能是系统资源管理，所提供的最基本的服务是为应用程序的运行提供平台，为用户提供一个操作环境。上述概念很笼统，只有在读完后续三节，理解了操作系统内核和其他程序的关系之后，才能有一个清晰的认识。

3.4.2　操作系统发展的里程碑

操作系统自诞生之日起就经历了一轮又一轮技术创新的洗礼和市场上的残酷竞争，涌现出了具有里程碑意义的产品。现在来简要阐述那些在技术或市场上曾经或者正在闪亮的明星操作系统，从中或许可以感受操作系统发展的脉络。

1. GM-NAA I/O

1956 年，在 IBM 701 计算机监控程序的基础上，通用汽车和北美航空为 IBM 704 计算机开发了可以称为历史上第一个操作系统的 GM-NAA I/O 系统。GM-NAA I/O 系统实现了批处理功能，并包含访问外设的公共程序库。该系统安装在了当时大约 20 台 IBM 704 计算机上，早期的操作系统都是用户自己开发的。GM-NAA I/O 系统是一个三阶段的、磁带作为二级存储器的批处理系统。首先是输入阶段，将全部作业转换成二进制代码，记录到磁带上；然后启动计算阶段的监控程序，逐个执行每个作业的计算任务，计算结果以二进制代码的形式记录到磁带上；最后生成十进制格式的输出磁带，可以拿到主机房旁边的小型计算机上打印输出了。

2. CTSS

CTSS(Compatible Time-Sharing System)，即兼容分时系统。分时是指多个用户分享使用同一台计算机。多个程序分时共享硬件和软件资源。分时操作系统是一个多用户交互式操作系统。与后期的操作系统相比，CTSS 是一个简单甚至可以说是粗糙的操作系统。尽管如此，它却拥有分时系统必须有的特征：宏观上在同一时间内能完成多个同时进行的交互任务。

3. Multics

Multics 是 1964 年由贝尔实验室、麻省理工学院及美国通用电气公司所共同参与研发的，是一套安装在大型主机上多人多任务的操作系统。Multics 的目的是想要让大型主机可

以达成提供 300 个以上的终端机连线使用,后来因计划进度落后,资金短缺,宣告失败。

Multics 以 CTSS 为基础,运行在美国通用电力公司的大型机 GE-645,目的是连接 1000 部终端机,支持 300 个用户同时在线。

1969 年,因 MULTICS 计划的工作进度过于缓慢,最后终究遭到裁撤的命运,贝尔实验室退出此计划,但这并不是它的全部。

4. UNIX

UNIX 系统是一个分时系统。最早的 UNIX 系统于 1970 年问世。在 20 世纪 60 年代末,Kenneth Thompson 和 Dennis Ritchie 都曾参加过交互方式分时系统 Multics 的研制工作,而开发该系统所使用的工具是 CTSS。这两个系统在操作系统的发展过程中都产生过重大影响。在此基础上,在对当时现有的技术进行精选提炼和发展的过程中,Kenneth Thompson 于 1969 年在小型计算机上开发了 UNIX 系统,后于 1970 年投入运行。

20 世纪 60 年代,美国 AT&T 公司贝尔实验室(AT&T Bell Laboratories)的研究员肯·汤普森(Kenneth Thompson)闲来无事,手痒难耐,想玩一个他自己编的模拟在太阳系航行的电子游戏——*Space Travel* 。他背着老板,找到一台空闲的小型计算机 PDP-7。但这台计算机没有操作系统,而游戏必须使用操作系统的一些功能,于是他着手为 PDP-7 开发操作系统。后来,这个操作系统被命名为 UNICS(Uniplexed Information and Computing Service),后来改名为 UNIX。我们是不是仍然可以从 UNIX 的名字中看到 Multics 系统的影子?

1969 年,美国贝尔实验室的 Kenneth Thompson,以 BCPL 为基础,设计出很简单且很接近硬件的 B 语言(取 BCPL 的首字母),并且用 B 语言编写了初版 UNIX 操作系统。

1971 年,同样酷爱 *Space Travel* 的丹尼斯·里奇为了能早点儿玩上游戏,加入汤普森的开发项目,合作开发 UNIX。他的主要工作是改造 B 语言,使其更成熟。

1972 年,美国贝尔实验室的丹尼斯·里奇在 B 语言的基础上最终设计出了一种新的语言,他取了 BCPL 的第二个字母作为这种语言的名字,这就是 C 语言。

1973 年年初,C 语言的主体完成。汤普森和丹尼斯·里奇迫不及待地开始用它完全重写了 UNIX。此时,编程的乐趣使他们已经完全忘记了那个 *Space Travel* ,一门心思地投入到 UNIX 和 C 语言的开发中。随着 UNIX 的发展,C 语言自身也在不断地完善。直到 2020 年,各种版本的 UNIX 内核和周边工具仍然使用 C 语言作为最主要的开发语言,其中还有不少继承汤普森和里奇之手的代码。

在开发中,他们还考虑把 UNIX 移植到其他类型的计算机上使用。C 语言强大的移植性(Portability)在此显现。机器语言和汇编语言都不具有移植性,为 x86 开发的程序,不可能在 Alpha、SPARC 和 ARM 等机器上运行。而 C 语言程序则可以使用在任意架构的处理器上,只要哪种架构的处理器具有对应的 C 语言编译器和库,然后将 C 源代码编译、连接成目标二进制文件之后即可在哪种架构的处理器运行。

1977 年,丹尼斯·里奇发表了不依赖于具体机器系统的 C 语言编译文本《可移植的 C 语言编译程序》。

5. Alto

1973 年 4 月,第一个可操作的 Alto 计算机在 Xerox PARC 完成。Alto 是第一个把计算机所有元素结合到一起的图形界面操作系统。它使用三键鼠标、位运算显示器、图形窗口、以太网络连接。乔布斯参观了施乐的研发中心,将其图形化的界面移植到苹果计算机系

统中，后来，比尔·盖茨受邀观摩了乔布斯的图形界面，又将其移植到微软的 Windows 界面中。

6. MS-DOS

MS-DOS(Microsoft Disk Operating System)主宰了 20 世纪 80 年代个人计算机操作系统市场，使微软从一个小公司一跃成为全球第一大公司，这得益于计算机巨人 IBM 采用了微软的操作系统。DOS 是一个非常简单的操作系统，仅支持单用户、单任务，主要是为了适应当时个人计算机较低的硬件配置。随着个人计算机硬件的升级和图形界面的发展，20 世纪 90 年代后，DOS 逐渐淡出市场。

7. macOS

macOS 是一套由苹果开发的运行于 Macintosh 系列计算机上的操作系统。1984 年，macOS 是首个在商用领域成功的图形用户界面操作系统。macOS 是苹果公司研制的桌面操作系统系列的最新名字，之前有过很多长相差不多的名字。macOS 目前特指苹果的桌面操作系统，与苹果公司其他智能设备上的操作系统 iOS、tvOS、watchOS 相对应。

8. Windows

微软公司从 1983 年开始研发 Windows 系统，最初的研发目标是在 MS-DOS 的基础上提供一个多任务的图形用户界面。第一个版本 Windows 1.0 于 1985 年问世，它是一个具有图形用户界面的系统软件。直到 1990 年微软推出 Windows 3.0 成为一个重要的里程碑，它以压倒性的商业成功确定了 Windows 系统在个人计算机领域的垄断地位，现今流行的 Windows 窗口界面的基本形式也是从 Windows 3.0 开始基本确定的。

9. Linux

Linux，全称为 GNU/Linux，是一套免费使用和自由传播的类 UNIX 操作系统，是一个基于 POSIX 的多用户、多任务和多 CPU 的操作系统。伴随着互联网的发展，Linux 得到了来自全世界软件爱好者、组织、公司的支持。它除了在服务器方面保持着强劲的发展势头以外，在个人计算机、嵌入式系统上都有着长足的进步。使用者不仅可以直观地获取该操作系统的实现机制，而且可以根据自身的需要来修改完善 Linux，使其最大化地适应用户的需要。

Linux 不仅系统性能稳定，而且是开源软件。其核心防火墙组件性能高效、配置简单，保证了系统的安全。在很多企业网络中，为了追求速度和安全，Linux 被网络运维人员当作服务器使用，甚至当作网络防火墙，这是 Linux 的一大亮点。

Linux 具有开放源码、自由传播、技术社区用户多等特点，开放源码使得用户可以自由裁剪，灵活性高，功能强大，成本低。尤其系统中内嵌网络协议栈，经过适当的配置就可实现路由器的功能。这些特点使得 Linux 成为开发路由交换设备的理想开发平台。

10. 安卓

安卓(Android)是一种基于 Linux 内核的自由及开放源代码的操作系统。主要应用于移动设备，如智能手机和平板电脑，由美国 Google 公司和开放手机联盟领导及开发。Android 操作系统最初由 Andy Rubin 开发，主要支持手机。2005 年 8 月由 Google 收购注资。2007 年 11 月，Google 与 84 家硬件制造商、软件开发商及电信营运商组建开放手机联盟共同研发改良 Android 系统。第一部 Android 智能手机发布于 2008 年 10 月。Android 逐渐扩展到平板电脑及其他领域上，如电视、数码相机、游戏机、智能手表等。

3.4.3 操作系统的地位

计算机系统是由相互联系、互相作用的多个子系统构成的,操作系统只是其中的一部分。只有了解操作系统和其他子系统的关系,从不同的角度认识操作系统,才能对操作系统的概念有全面而深入的认识。

1. 操作系统与硬件平台

可以从三个层次来看计算机硬件的架构:指令集架构、计算机组成和硬件(即计算机组成的实现)。其中,指令集架构在最上层,和程序设计紧密相关,是程序员能够看到并使用的软硬件界面,例如,指令的类型和功能、有哪些寄存器、机器字位数、中断机制、设备的控制方式等。所以指令集架构是操作系统赖以运行的硬件平台。也可以说,在计算机系统的层次架构中,整个硬件系统之上的第一层软件就是操作系统。

自从虚拟机技术出现以后,可以在一个硬件平台上,用软件的方法构造出多个不同架构的虚拟机,其功能完全等价于操作系统赖以运行的硬件平台,就像人们经常看到的,在同一个硬件平台上,同时运行着多个不同的操作系统。所以虚拟机也可被看作操作系统之下的指令集架构。

2. 操作系统和应用程序

操作系统是整个计算机系统的管理者,也是服务提供者,除操作系统之外的所有程序都可被看作应用程序,如数据库系统、编译系统等。所以应用程序接受操作系统的管理,也享受操作系统的服务。操作系统可以给应用程序分配或回收系统资源,这是由操作系统的管理者身份决定的。例如,在分时系统中,操作系统可以把 CPU 分配给应用程序,也可以在时间片到时收回 CPU 的使用权。应用程序通过系统调用直接请求操作系统的服务,也可以通过各种程序库(如 C 函数库,也属于应用程序)间接地请求操作系统的服务。总之,操作系统和应用程序之间的关系就是管理者和被管理者、服务者与请求者之间的关系。

3. 操作系统与运行时系统

运行时系统是伴随高级语言出现的,那时还没有操作系统。运行时也是为应用程序的运行提供服务的,这一点和操作系统一样,区别又在哪里呢?

从某种意义上讲,操作系统取代了运行时系统的一部分服务功能。例如,ALGOL 和 FORTRAN 语言都有自己的运行时系统,其中都有直接控制外部设备的驱动程序,也有子程序调用时参数传递的相关机制。操作系统出现以后,运行时系统中的某些服务功能,如驱动程序,被转移到操作系统中。这样做的理由主要有两个:其一,由操作系统控制设备,运行时系统不必自讨苦吃;其二,设备属于系统资源,必须有一个统一的管理者,不能由各个运行时系统来控制。现在举个例子来说明操作系统和运行时系统的关系。早年人们出门旅行,要带着锅碗瓢盆等这类衣食住行的各类装备,当然这些装备都是人们自己制作并准备好的;后来,商品经济有了一定的发展,有专门的机构提供旅行的所有装备,出门前人们可以购买全套的必要装备,提供旅行装备的如同运行时系统;再后来,商品经济高度发展,无论走到哪里,都有衣食住行的服务提供商,这就是操作系统做的工作。

有些个性化的服务,如程序运行过程中的垃圾回收机制、有关子程序调用的参数传递机制是和各种语言具体实现相关的问题,操作系统没必要身陷其中,仍是各种运行时系统发挥作用的地方。看看现在的 C 和 Java,有各自的运行时系统,在动态内存管理方面就采用了

不同的方法。运行时系统在现代计算机系统中仍然是不可或缺的角色，例如，稻壳等各种沙箱软件，为应用程序提供有特色的运行环境，这些都属于运行时系统，它们在应用程序和操作系统之间表现了其特有的弹性。

用户程序在执行时可以调用操作系统的程序，也可以调用运行时系统的程序，但二者有本质的不同。操作系统为所有的程序（包括运行时系统）提供服务，用户程序执行还是不执行，它都在内存；而运行时系统一般是随着用户程序装入内存的，用户程序结束了，它也不会再待在那里。另外，如果应用程序使用了 C 运行时系统的某些功能，那么受到 C 语言编译系统的支持，具有较好的跨平台性；如果用户程序调用了操作系统的系统调用，那就只能在这种操作系统上运行。图 2.10 说明了用户程序、运行时系统和操作系统之间的层次关系。

3.5　操作系统内核

操作系统和应用程序都是软件，操作系统为应用程序提供运行环境，在应用程序运行过程中承担了服务者、管理者和控制者的角色，这是操作系统最核心的功能。操作系统中实现这些功能的代码就是操作系统内核，简称内核。从计算机启动到关机的整个过程中，内核要么处于应用程序的服务请求，要么处于随时待命的状态，可以说，内核从来不缺席对整个系统的控制，类似 110 警务中心，一直处于运行状态或可运行的状态。很多时候人们会把内核和操作系统等同起来。

本节将从内核和应用程序的关系开始阐述内核的概念，从总体上介绍内核提供的服务、功能和结构，为后续各章详细阐述内核机制和策略提供必要的基础。

3.5.1　内核的概念

操作系统将用户程序装入内存，然后把 CPU 控制权交给用户程序，用户程序执行结束后操作系统还要负责相应的善后工作。用户程序在执行过程中，也会调用操作系统的程序，访问计算机系统资源。所以在早期的计算机系统中，操作系统完全是一个服务者的角色。操作系统为用户程序工作，就像用户程序的一部分，作用如同仆人，操作权限与用户程序相同。事实上，早期的 CPU 就不支持内核模式。然而，没有一成不变的东西，操作系统的地位也会随着它的作用在逐渐发生改变。一般情况下，用户程序执行结束后，就会释放 CPU、内存等资源，退出系统。不同的是，操作系统不会退出，在送走一个用户程序之后，立刻会迎来下一个用户程序，继续为下一个用户程序服务。所以，操作系统成为所有用户的公共服务程序。在多任务系统中，它还需要同时为多个用户程序提供服务，并协调用户程序使用系统的公共资源，防止用户程序之间的相互干扰和破坏。也就是说，操作系统需要对所有的用户程序负责，这时，它变得更像是一个管理机构，其操作权限高于用户程序也就顺理成章了。为了支持操作系统这种身份的变化，需要在指令集架构层面上提供支持，这显示操作系统的发展反过来促进了计算机本身的改进。

权限描述了一个程序能做的事情，具体点就是：一个程序可以对哪个对象执行哪种操作。操作系统具有更高的或超级的权限，意味着操作系统可以做用户程序不能做的事情。例如，操作系统可以决定在什么时间把资源分配给用户程序，以及什么时候回收用户程序占用的资源，但是用户程序不能对操作系统做同样的事情。凭借公共服务，操作系统获得了超

级权限。此时,操作系统已经从仆人的角色升级为整个计算机系统的管理和控制者了。我们把这种从机器加电到关机一直管理和控制计算机系统、具有超级权限的程序称为**操作系统内核**,简称为**内核**(Kernel),有时也特指操作系统。

为了实现内核的超级权限,硬件设计者将 CPU 的运行模式分为内核模式和用户模式。在内核模式下,可以执行 CPU 的全部功能,而在用户模式下,只能执行通常的指令,而不能执行某些关键的指令。例如,输入/输出指令对所有用户程序都要使用的外设进行操作,如果普通用户程序都可以直接操作外设,就会造成它们之间的相互干扰或破坏,所以这类指令只能在 CPU 的内核模式下运行,称为**特权指令**。显然,CPU 内核模式就是专为操作系统内核设置的,内核程序在内核模式下运行,用户程序在用户模式下运行。一般 CPU 的标志寄存器 FLAG 中会有一个 Mode 位来标识当前 CPU 所处的模式。

从原理上讲,CPU 有两种运行模式就可以保证操作系统的主导地位,但很多计算机系统中设计了更多的运行模式。例如,RISC-V 架构的计算机中,CPU 运行模式分为机器模式、监管者模式和用户模式三种。机器模式具有最高级的权限,用于管理计算机硬件,提供虚拟机,而每个虚拟机拥有监管者和用户两种运行模式。其中的监管者模式相当于内核模式,在该模式下运行的程序都属于内核。在不涉及虚拟机的情况下,本书仅针对 CPU 内核模式和用户模式进行讨论。

内核程序看上去和用户程序一样,身上并没有特殊的标签,CPU 是无法区分二者的。那么如何保证内核程序运行时 CPU 处于内核模式,或者说 CPU 处于内核模式时一定是在运行内核程序呢?这要从计算机的启动说起。当计算机加电,CPU 一开始就处于内核模式,引导程序首次装入内存运行的代码就运行在内核模式了,并从此成为 CPU 的“主人”,就像小鸡从蛋壳中出来,把看到的第一个活物当作自己的妈妈。内核程序之所以具有特权,根本原因在于它是 CPU“一睁眼”就能看到并运行的第一个程序,所以内核的特权是天生的。如果某个用户程序也想过把瘾的话,完全可以在自己的机器上这样安排,让计算机一开机就执行用户自己的程序。如果把自己的程序安排在别人的机器上并在内核态下运行,就是安全问题了!这是许多病毒喜欢做的。

当内核要将 CPU 的控制权交给用户程序时,必须先将标志寄存器的模式位改为用户模式之后才能转移,否则后果不堪设想。然而,当用户程序要返回操作系统时却不能这样做,因为在 CPU 的用户模式下是不能通过程序来直接修改标志寄存器内容的,否则,不就谁都可以进入内核模式执行了?用户程序要想转到内核程序中执行,只能通过系统调用指令完成。用户程序通过执行系统调用指令请求操作系统的服务转移到内核中,同时 CPU 也从用户模式切换到内核模式。3.5.2 节将详述该问题。

除了用户程序和内核之间主动地相互转移之外,中断事件也会导致 CPU 模式的切换。在操作系统出现之前,中断事件都是由用户程序自己来处理的。因为这些事件往往事关整个计算机系统,所以在操作系统出现之后,中断事件就交由操作系统内核处理。一旦出现中断事件,CPU 会暂停当前程序的执行,转向内核程序并将 CPU 切换至内核模式,直至处理结束后返回。这形成了中断事件、CPU、操作系统和用户程序之间一种特定的关系。

3.5.2 内核提供的服务(系统调用)

能力越强,责任越重!

内核握有特权，管理整个系统的资源，不允许应用程序随意使用系统的资源。这就要求内核为应用程序提供服务，应用程序通过请求这些服务使用系统资源。当然，内核除了提供访问系统资源的服务之外，也会提供另外一些服务，使程序设计和运行变得更简单和方便。可见，服务者是内核在计算机系统中的一个重要角色。需要注意的是，不同的操作系统内核所提供的服务并不相同，但都会满足程序设计的一般性需求。

1. 服务的分类

一般内核可以为用户程序提供如下几类服务。

(1) 程序运行过程的控制，包括程序装入(load)、执行(execute)、结束(exit)和放弃(abort)。这类服务使得一个程序可以控制自身的运行过程，也可以控制另一个程序的运行过程。

(2) 输入/输出操作，如获取某个设备的状态、从设备输入/输出数据或者对设备的某些控制操作(移动打印纸、落下绘图笔等)。应用程序通过这类服务可以操作自身不能直接访问的设备，也大大简化了程序设计的复杂性。

(3) 文件及目录操作，包括建立、删除文件和目录，读、写文件，打开、关闭文件，获取、设置文件属性等。应用程序一般是不能直接访问像磁盘、磁带这样的外部存储器的，因为它们是公共资源。内核向应用程序提供文件系统的这些服务，使应用程序对外部存储器的使用不仅容易，而且安全可靠。

(4) 分配和释放内存。尽管从程序员的角度可以访问 ISA 架构下的整个虚拟地址空间，但是自从操作系统出现之后，接管了应用程序虚拟地址空间的管理，应用程序只能访问内核授权访问的区域。应用程序各部分所在的虚拟地址范围以及程序对各个区域的访问权限，都必须经内核授权，否则就会发生越界或越权错误。如果应用程序在执行过程中需要更多的内存空间，则必须请求操作系统为其分配虚拟地址空间，用完之后还要释放。应用程序不会使用未经操作系统分配的存储空间。注意，在 C 语言中的应用程序也可以通过 malloc() 来申请内存，但申请到的只是运行时系统管理的内存。运行时系统就像一个批发商，它从内核申请一大块内存，然后根据请求分成小块给应用程序。需要注意的是：无论是虚拟地址空间还是物理地址空间，都是由操作系统管理的，应用程序的访问都受到限制。

(5) 通信服务。在多任务系统中，操作系统要保证各任务之间不会相互干扰。在此前提下可以支持不同任务之间的通信，为应用程序之间的合作带来极大的方便。计算机网络出现之后，内核进一步提供了不同计算机系统上的任务之间的通信服务。

(6) 信息维护。这类服务包括获取、设置日期和时间等系统信息，获取、设置应用程序或用户的相关信息等。

除以上各类服务之外，操作系统内核还为应用程序提供保护、安全、错误检测等各方面的服务。这些服务都是内核根据应用程序的请求而完成的相关工作，系统调用是应用程序直接向内核发出请求的唯一途径。

2. 系统调用的概念

从应用程序设计者的角度，请求内核服务就像调用子程序一样简单就好了。子程序库一旦连接入用户程序中，就成为用户程序的一部分，用户程序可以使用入口地址来调用库中的子程序，就像调用自己的子程序。然而，内核是为整个系统所有的应用程序服务的，它不可能像库程序一样连接入各个应用程序的可执行代码中。为了提高效率，内核的程序必须