第 5 章 进 程 管 理

在多道程序系统中，同时存在多个程序在运行。它们共享系统的资源，轮流使用CPU，彼此之间相互制约和依赖，表现出复杂的行为特性。进程是为了刻画并发程序的执行过程而引入的概念，进程管理就是对并发程序的运行过程的管理，也就是对 CPU 的管理。

进程管理的功能是跟踪和控制所有进程的活动，为它们分配和调度 CPU，协调进程的运行步调。进程管理的目标是最大限度地发挥 CPU 的处理能力，提高进程的运行效率。

5.1 进　　程

进程是现代操作系统的核心概念，它用来描述程序的执行过程，是实现多任务操作系统的基础。操作系统的其他所有内容都是围绕着进程展开的。因此，正确地理解和认识进程是理解操作系统原理的基础和关键。

5.1.1 程序的顺序执行与并发执行

1. 程序的顺序执行

如果程序的各操作步骤之间是依序执行的，程序与程序之间是串行执行的，这种执行程序的方式就称为顺序执行。顺序执行是单道程序系统中的程序的运行方式。

程序的顺序执行具有如下特点：

（1）顺序性。CPU 严格按照程序规定的顺序执行，仅当一个操作结束后，下一个操作才能开始执行。多个程序要运行时，仅当一个程序全部执行结束后另一个程序才能开始。

（2）封闭性。程序在封闭的环境中运行，即程序运行时独占全部系统资源，只有程序本身才能改变程序的运行环境。因而程序的执行过程不受外界因素的影响，结果只取决于程序自身。

（3）可再现性。程序执行的结果与运行的时间和速度无关，结果总是可再现的，即无论何时重复执行该程序都会得到同样的结果。

总的说来，这种执行程序的方式简单，且便于调试。但由于顺序程序在运行时独占全部系统资源，因而系统资源利用率很低。DOS 程序就是采用顺序方式执行的。

2. 程序的并发执行

单道程序、封闭式运行是早期操作系统的标志，而多道程序并发运行是现代操作系

统的基本特征。由于同时有多个程序在系统中运行，使系统资源得到充分的利用，系统效率大大提高。

程序的并发执行是指若干程序或程序段同时运行。它们的执行在时间上是重叠的。程序的并发执行有以下特点：

（1）间断性。并发程序之间因竞争资源而相互制约，导致程序运行过程的间断。例如，在单 CPU 的系统中，多个程序需要轮流占用 CPU 运行，未获得 CPU 的程序就必须等待。

（2）没有封闭性。当多个程序共享系统资源时，一个程序的运行受其他程序的影响，其运行过程和结果不完全由自身决定。例如，一个程序计划在某一时刻执行一个操作，但很可能在那个时刻到来时它没有获得 CPU，因而也就无法完成该操作。

（3）不可再现性。由于没有了封闭性，并发程序的执行结果与执行的时机以及执行的速度有关，结果往往不可再现。

可以看出，并发执行程序虽然可以提高系统的资源利用率和吞吐量，但程序的行为变得复杂和不确定。这使程序难以调试，若处理不当还会带来许多潜在问题。

3. 并发执行的潜在问题

程序在并发执行时会导致执行结果的不可再现性，这是多道程序系统必须解决的问题。我们用下面的例子来说明并发执行过程对运行结果的影响，从而了解产生问题的原因。

设某停车场使用程序控制电子公告牌来显示空闲车位数。空闲车位数用一个计数器 C 记录。车辆入库时执行程序 A，车辆出库时执行程序 B，它们都要更新同一个计数器 C。程序 A 和程序 B 的片段如图 5-1 所示。

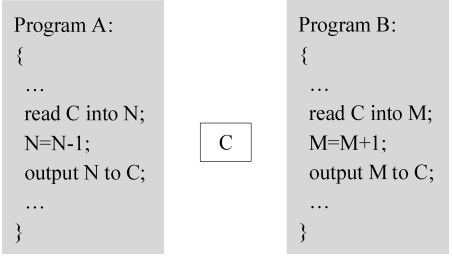

图 5-1　两个程序并发运行，访问计数器 C

更新计数器 C 的操作对应的机器语言有 3 个步骤：读取内存 C 单元的数据到一个寄存器中，修改寄存器的数值，然后再将其写回 C 单元中。

由于车辆出入库的时间是随机的，程序 A 与程序 B 的运行时间也就是不确定的。当出入库同时发生时，将使两个程序在系统中并发运行。它们各运行一次后 C 计数器的值应保持不变。但结果可能不是如此。

如果两个程序的运行时序如图 5-2（a）所示，即一个程序对 C 进行更新的操作是在另一个程序的更新操作全部完成之后才开始，则 C 被正确地更新了。如果两个程序的运行时序如图 5-2（b）所示穿插地进行，即当一个程序正在更新 C，更新操作还未完成时，CPU 发生了切换，另一个程序被调度运行，并且也对 C 进行更新。在这种情况下会导致

错误的结果。

时间	T0	T1	T2	T3	T4	T5
程序A	C→N	N−1	N→C			
程序B				C→M	M+1	M→C
C的值	100	100	99	99	99	100

(a) 两个程序顺序访问C, 更新正确

时间	T0	T1	T2	T3	T4	T5
程序A	C→N			N−1	N→C	
程序B		C→M	M+1			M→C
C的值	100	100	100	100	99	101

(b) 两个程序交叉访问C, 更新错误

图 5-2 并发程序的执行时序影响执行结果

可以看出，导致 C 更新错误的原因是两个程序交叉地执行了更新 C 的操作。概括地说，当多个程序在访问共享资源时的操作是交叉执行时，则会发生对资源使用上的错误。

5.1.2 进程的概念

进程的概念最早出现在 20 世纪 60 年代中期，此时操作系统进入多道程序设计时代。多道程序并发显著地提高了系统的效率，但同时也使程序的执行过程变得复杂与不确定。为了更好地研究、描述和控制并发程序的执行过程，操作系统引入了进程的概念。进程概念对于理解操作系统的并发性有着极为重要的意义。

1. 进程

进程（process）是一个可并发执行的程序在一个数据集上的一次运行。简单地说，进程就是程序的一次运行过程。

进程与程序的概念既相互关联又相互区别。程序是进程的一个组成部分，是进程的执行文本，而进程是程序的执行过程。两者的关系可以比喻为电影与胶片的关系：胶片是静态的，是电影的放映素材。而电影是动态的，一场电影就是胶片在放映机上的一次“运行”。对进程而言，程序是静态的指令集合，可以永久存在；而进程是动态的过程实体，动态地产生、发展和消失。

此外，进程与程序之间也不是一一对应的关系，表现在以下两点：

（1）一个进程可以顺序执行多个程序，如同一场电影可以连续播放多部胶片一样。

（2）一个程序可以对应多个进程，就像一本胶片可以放映多场电影一样。程序的每次运行就对应了一个不同的进程。更重要的是，一个程序还可以同时对应多个进程。例如系统中只有一个 vi 程序，但它可以被多个用户同时执行，编辑各自的文件。每个用户的编辑过程都是一个不同的进程。

2. 进程的特性

进程与程序的不同主要体现在进程有一些程序所没有的特性。要真正理解进程，首

先应了解它的基本性质。进程具有以下几个基本特性：

（1）动态性。进程由“创建”而产生，由“撤销”而消亡，因“调度”而运行，因“等待”而停顿。进程从创建到消失的全过程称为进程的生命周期。

（2）并发性。在同一时间段内有多个进程在系统中活动。它们宏观上是在并发运行，而微观上是在交替运行。

（3）独立性。进程是可以独立运行的基本单位，是操作系统分配资源和调度管理的基本对象。因此，每个进程都独立地拥有各种必要的资源，独立地占有 CPU 运行。

（4）异步性。每个进程都独立地执行，各自按照不可预知的速度向前推进。进程之间的协调运行由操作系统负责。

3. 进程的基本状态

在多道系统中，进程的个数总是多于 CPU 的个数，因此它们需要轮流地占用 CPU。从宏观上看，所有进程同时都在向前推进；而在微观上，这些进程是在走走停停之间完成整个运行过程的。为了刻画一个进程在各个时期的动态行为特征，通常采用状态模型。

进程有 3 个基本的状态：

（1）就绪态。进程已经分配到了除 CPU 之外的所有资源，这时的进程状态称为就绪状态。处于就绪态的进程，一旦获得 CPU 便可立即执行。系统中通常会有多个进程处于就绪态，它们排成一个就绪队列。

（2）运行态。进程已经获得 CPU，正在运行，这时的进程状态称为运行态。在单 CPU 系统中，任何时刻只能有一个进程处于运行态。

（3）等待态。进程因某种资源不能满足，或希望的某事件尚未发生而暂停执行时，则称它处于等待态。系统中常常会有多个进程处于等待态，它们按等待的事件分类，排成多个等待队列。

4. 进程状态的转换

进程诞生之初处于就绪态，在其后的生存期间内不断地从一个状态转换到另一个状态，最后在运行态结束。图 5-3 所示是一个进程的状态转换图。

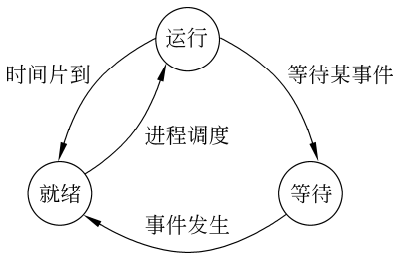

图 5-3　进程的状态转换图

引起状态转换的原因如下：

- 运行态→等待态：正在执行的进程因为等待某事件而无法执行下去。例如，进程申请某种资源，而该资源恰好被其他进程占用，则该进程将交出 CPU，进入等待状态。
- 等待态→就绪态：处于等待状态的进程，当其所申请的资源得到满足时，则系统

将资源分配给它，并将其状态变为就绪态。

- 运行态→就绪态：正在执行的进程的时间片用完了，或者有更高优先级的进程到来，系统会暂停该进程的运行，使其进入就绪态，然后调度其他进程运行。
- 就绪态→运行态：处于就绪态的进程，当被进程调度程序选中后，即进入 CPU 运行。此时该进程的状态变为运行态。

5.1.3 进程控制块

进程由程序、数据和进程控制块 3 个基本部分组成。程序是进程执行的可执行代码，数据是进程所处理的对象，进程控制块用于记录有关进程的各种信息。它们存在于内存，其内容会随着执行过程的进展而不断变化。在某个时刻的进程的执行内容（指代码、数据和堆栈）被称为进程映像（process image）。进程映像可以看作是进程的剧本，决定了进程推进的路线和行为。进程控制块则是进程的档案。系统中每个进程都是唯一的。即使两个进程执行的是同一映像，它们也都有各自的进程控制块，因此是不同的进程。

进程控制块（Process Control Block，PCB）是为管理进程而设置的一个数据结构，用于记录进程的相关信息。当创建一个进程时，系统为它生成 PCB；进程完成后，撤销它的 PCB。因此，PCB 是进程的代表，PCB 存在则进程就存在，PCB 消失则进程也就结束了。在进程的生存期中，系统通过 PCB 来感知进程，了解它的活动情况，通过它对进程实施控制和调度。因此，PCB 是操作系统中最重要的数据结构之一。

PCB 记录了有关进程的所有信息，主要包括以下 4 方面的内容。

（1）进程描述信息。

进程描述信息用于记录一个进程的标识信息和身份特征，如家族关系和归属关系等。通过这些信息可以识别该进程，了解进程的权限，以及确定这个进程与其他进程之间的关系。

系统为每个进程分配了一个唯一的整数作为进程标识号 PID，这是最重要的标识信息。系统通过 PID 来标识各个进程。

（2）进程控制与调度信息。

进程的运行需要由系统进行控制和调度。进程控制块记录了进程的当前状态、调度策略、优先级、时间片等信息。系统依据这些信息实施进程的控制与调度。

（3）资源信息。

进程的运行需要占用一些系统资源，必要的资源包括进程的地址空间、要访问的文件和设备以及要处理的信号等。进程是系统分配资源的基本单位。系统将分配给进程的资源信息记录在进程的 PCB 中。通过这些信息，进程就可以访问分配到的各种资源。

（4）现场信息。

进程现场也称为进程上下文（process context），包括 CPU 的各个寄存器的值。这些值刻画了进程的运行状态和环境。退出 CPU 的进程必须保存好这些现场信息，以便在下次被调度时继续运行。当进程被重新调度运行时，系统用它的 PCB 中的现场信息恢复 CPU 现场。现场一旦恢复，进程就可以从上次运行的断点处继续执行下去了。

5.1.4 Linux 系统中的进程

在 Linux 系统中，进程也称为任务（task），两者的概念是一致的。

1．Linux 进程的状态

Linux 系统的进程有 5 种基本状态，即可执行态（运行态与就绪态）、可中断睡眠态、不可中断睡眠态、暂停态和僵死态。状态转换图如图 5-4 所示。

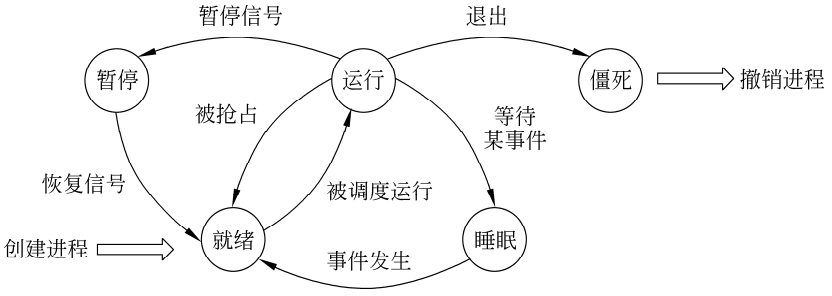

图 5-4　Linux 系统的进程状态转换图

Linux 进程的基本状态定义如下：

（1）可执行态（runnable）：可执行态包含上述状态图中的运行和就绪两种状态。处于可执行态的进程均已具备运行条件。它们或正在运行，或准备运行。

（2）睡眠态（sleeping）：即等待态。此时进程正在等待某个事件或某个资源。睡眠态又细分为可中断的（interruptable）和不可中断的（uninterruptable）两种。它们的区别在于，在睡眠过程中，处于不可中断状态的进程会忽略信号，而处于可中断状态的进程如果收到信号会被唤醒而进入可执行态，待处理完信号后再次进入睡眠态。

（3）暂停态（stopped 或 traced）：处于暂停态的进程是由运行态转换而来的，等待某种特殊处理。当进程收到一个暂停信号时则进入暂停态，等待恢复运行的信号。

（4）僵死态（zombie）：进程运行结束或因某些原因被终止时，它将释放除 PCB 外的所有资源。这种占有 PCB 但已经无法运行的进程就处于僵死态。

2．Linux 进程的状态转换过程

Linux 进程的状态转换过程如下。

新创建的进程处于可执行的就绪态，等待调度执行。

处于可执行态的进程在就绪态和运行态之间轮回。就绪态的进程一旦被调度程序选中，就进入运行状态。当进程的时间片耗尽或有更高优先级的进程就绪时，调度程序将选择新的进程来抢占 CPU 运行。被抢占的进程将交出 CPU，转入就绪态等待下一次的调度。处于此轮回的进程在运行态与就绪态之间不断地高速切换，可谓瞬息万变。因此，对观察者（系统与用户）来说，将此轮回概括为一个相对稳定的可执行态才有意义。

运行态、睡眠态和就绪态形成一个回路。处于运行态的进程，有时需要等待某种资源或某个事件的发生，这时已无法占有 CPU 继续运行，于是它退出 CPU，转入睡眠态。当该进程等待的事件发生后，进程被唤醒，进入就绪态。

运行态、暂停态和就绪态也构成一个回路。当处于运行态的进程接收到暂停执行信

号时，它就放弃 CPU，进入暂停态。当暂停的进程获得恢复执行信号时，就转入就绪态。

处于运行态的进程执行结束后进入僵死态。待父进程（即创建此进程的进程）对其进行相应处理后撤销它的 PCB。此时，这个进程就完成了它的使命，从僵死走向彻底消失。

3．Linux 的进程描述符

Linux 系统用 task_struct 结构来记录进程的信息，称为进程描述符，也就是通常所说的 PCB。系统中每创建一个新的进程，就给它建立一个 task_struct 结构，并填入进程的控制信息。task_struct 中的字段很多，主要包括以下内容：

- 进程标识号（pid）：标识该进程的一个整数。
- 归属关系（uid、gid）：进程的属主和属组的标识号。
- 家族关系（parent、children、sibling）：关联父进程、子进程及兄弟进程的链接指针。
- 链接指针（tasks、run_list）：将进程链入进程链表和可执行队列的指针。
- 状态（state）：进程当前的状态。
- 调度信息（policy、prio、time_slice）：调度使用的调度策略、优先级和时间片等。
- 记时信息（start_time、utime、stime）：进程建立的时间以及执行用户代码与系统代码的累计时间。
- 信号信息（signal、sighand）：进程收到的信号以及使用的信号处理程序。
- 退出码（exit_code）：进程运行结束后的退出代码，供父进程查询用。
- 文件系统信息（fs、files）：包括文件系统及打开文件的信息。
- 地址空间信息（mm）：进程使用的地址空间。
- 硬件现场信息（thread）：进程切换时保存的 CPU 寄存器的内容。
- 运行信息（thread_info）：有关进程运行环境、状况的 CPU 相关信息。

4. 查看进程的信息

查看进程信息的命令是 ps（process status）命令。该命令可查看记录在进程描述符 task_struct 中的几乎所有信息。

ps 命令

【功能】查看进程的信息。

【格式】ps [选项]

【选项】

-e	显示所有进程。
-t *tty*	显示终端 *tty* 上的进程。
-f	以全格式显示。
-o	以用户定义的格式显示。
a	显示所有终端上的所有进程。
u	以面向用户的格式显示。
x	显示所有不控制终端的进程。

-C *cmd*　显示命令名为 *cmd* 的进程。

n　显示 PID 为 *n* 的进程。

【说明】

（1）默认只显示在本终端上运行的进程，除非指定了-e、a、x 等选项。

（2）没有指定显示格式时，采用以下默认格式，分 4 列显示：

PID　TTY　TIME　CMD

各字段的含义如下：

PID　进程标识号。

TTY　进程对应的终端，“?”表示该进程不占用终端。

TIME　进程累计使用的 CPU 时间。

CMD　进程执行的命令名。

（3）指定-f 选项时，以全格式分 8 列显示：

UID　PID　PPID　C　STIME　TTY　TIME　CMD

各字段的含义如下：

UID　进程属主的用户标识号。

PPID　父进程的标识号。

C　进程最近使用的 CPU 时间。

STIME　进程开始时间。

其余同上。

（4）指定 u 选项时，以用户格式分 11 列显示：

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

各字段的含义如下：

USER　同 UID。

%CPU　进程占用 CPU 的时间与进程总运行时间之比。

%MEM　进程占用的内存与总内存之比。

VSZ　进程虚拟内存的大小，以 KB 为单位。

RSS　占用实际内存的大小，以 KB 为单位。

STAT　进程当前状态，用字母表示。含义如下：

R　可执行态；

S　可中断睡眠态；

D　不可中断睡眠态；

T　暂停态；

Z　僵死态。

START　同 STIME。

COMMAND　同 CMD。

其余同上。

例 5-1 ps 命令用法示例：

```
$ ps                    #以默认格式显示本终端上的进程的信息
 PID    TTY          TIME  CMD
 9805   pts/0    00:00:00  bash
 9835   pts/0    00:00:00  ps
$ ps -ef             #以全格式显示当前系统中所有进程的信息
UID    PID  PPID  C  STIME  TTY      TIME  CMD
 root    1     0  0  11:26 ?    00:00:03  /usr/lib/systemd/system --
switched-root --system --deserialize 23
 root    2     0  0  11:26 ?    00:00:00  [kthreadd]
 root    4     2  0  11:26 ?    00:00:00  [kworker/0:0H]
 ...
$ ps aux             #以用户格式显示当前系统中所有进程的信息
 USER    PID %CPU %MEM    VSZ  RSS  TTY    STAT START  TIME  COMMAND
 root      1  0.0  0.1   1948  816  ?      S    11:26  0:03  /usr/lib/
 systemd/system -- switched-root --system --deserialize 23
 root      2  0.0  0.0      0    0  ?      S    11:26  0:00  [kthreadd]
 ...
 cherry 9805  0.0  0.2   6184 1524  pts/0  S    22:31  0:00  bash
 cherry 9876  0.0  0.1   5980  940  pts/0  R    22:36  0:00  ps aux
 ...
$_
```

5.2 进程的运行模式

进程的运行紧密依赖于操作系统的内核。因此，理解进程的运行机制需要首先认识内核，了解内核的运行方式，进而了解进程在核心态与用户态下的不同执行模式。

5.2.1 操作系统的内核

操作系统的内核是硬件的直接操控者，因此不同硬件平台的内核实现也有所不同。对 PC 机来说，x86 是 32 位 PC 机的硬件架构，x86-64 是在 x86 的基础上扩展而成的 64 位架构，简称为 x64。这两种架构也是目前最为流行的硬件平台。因此，本教材中的内核部分将以 x86 和 x64 为基础架构，介绍 32 位和 64 位 Linux 系统的实现技术。

1．x86/x64 CPU 的执行模式

CPU 的基本功能就是执行指令。通常，CPU 指令集的指令可以划分为两类：特权指令和非特权指令。特权指令是指对指令本身或指令的操作数具有特殊权限限制的指令。这类指令可以访问系统中所有寄存器、内存单元和 I/O 端口，修改系统的关键设置。例如向控制寄存器加载地址、清理内存、设置时钟、进入中断等都是由特权指令完成的。而非特权指令是那些用于一般性的运算和处理的指令。这些指令只能访问用户程序自己的内存地址空间。

特权指令的权限高，如果使用不当则可能会破坏系统或其他用户的数据，甚至导致系统崩溃。为了安全起见，这类指令只允许操作系统的内核程序使用，而普通的应用程

序只能使用那些没有危险的非特权指令。实现这种限制的方法是在 CPU 中设置了一个代表特权级别的状态字（即 cs 寄存器中的 DPL 字段），修改这个状态字就可以切换 CPU 的运行模式。

x86/x64 的 CPU 支持 4 种不同特权级别，Linux 系统只用到了其中两个，即称为核心态的最高特权级模式 ring0 和称为用户态的最低特权级模式 ring3。在核心态下，CPU 能不受限制地执行所有指令，访问全部的内存地址，从而表现出最高的特权。而在用户态下，CPU 只能执行一般的非特权指令，访问受限的地址空间，因而也就没有特权。

2. 操作系统内核

一个完整的操作系统由一个内核和一些系统服务程序构成。内核（kernel）是操作系统的核心，它负责最基本的资源管理和硬件控制工作，为进程提供运行的环境。内核在系统引导时载入并常驻内存。它运行在核心态，因而能够访问所有的系统资源。

从进程的角度看，内核的功能有两个：一是支持进程的运行，包括为进程分配资源，控制和调度进程的运行；二是为进程提供服务，也就是提供一些内核函数（称为系统调用）供进程调用。由于进程运行在用户态，不能访问系统资源，因此当需要使用某些系统资源时，例如向显示屏输出一些文字，都需要通过调用内核的服务来完成。

3．Linux 系统的内核

图 5-5 是 Linux 系统的体系结构。系统分为 3 层。最底层是硬件层，包括了各种系统硬件和设备。硬件层之上是核心层，它形成了对硬件的第一层包装。对下，它管理和控制硬件；对上，它提供系统服务。用户层由系统的核外程序和用户程序组成，它们都是以用户进程的方式运行在核心之上，为用户提供更高层次的系统包装。

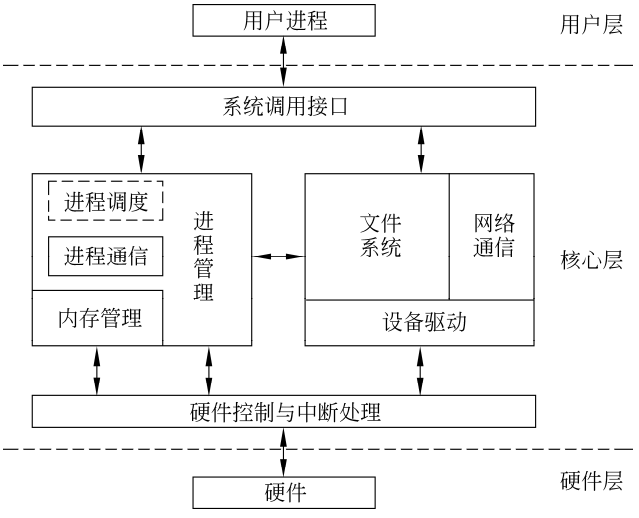

图 5-5　Linux 系统的内核结构

Linux 系统的内核主要由以下成分构成：

（1）系统调用接口。这是进程与内核的接口，进程通过此接口调用内核的功能。

（2）进程管理子系统。负责支持、控制和调度进程的运行。包括以下模块：

- 核心管理模块 kernel，管理 CPU，调度和协调进程的运行。