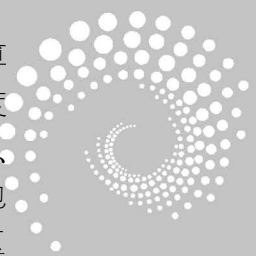
第5章

进程和作业管理

CHAPTER 5



进程是现代操作系统中最重要的概念，它是指计算机中正在运行的程序的实例。它是操作系统进行任务调度和资源管理的基本单位。每个进程都有自己的内存空间、代码、数据和执行状态。本章首先介绍进程的理论知识，包括操作系统引入进程的原因、进程的概念、进程与程序的区别、进程的构成、状态以及进程创建、终止、调度、进程间通信等内容。然后结合 FusionOS 介绍与进程相关的各种命令，包括进程管理命令、进程调度命令、作业的管理和调度以及后台任务的管理方法。最后介绍操作系统对软件包的管理方法。

5.1　程序和进程的概念

5.1.1　引入进程的原因

在介绍进程的概念之前，有必要先介绍一下引入进程的原因。多用户操作系统是一种可以同时为多个用户提供服务的操作系统，在多用户环境下，多个用户可以同时访问计算机系统，执行各自的任务。引入进程的主要动机之一是更好地支持多用户操作系统，使得计算机系统能够同时为多个用户提供服务，并有效地管理和调度多个任务的执行，主要体现在以下几个方面。

并发执行：多用户系统需要支持多个用户同时执行任务。必须通过某种数据结构对这些任务进行管理，进程就是操作系统用于管理多个任务可以在同一时间段内并发执行的数据结构，进程提供了一种方便的方式来管理多个任务。每个用户的任务都可以被看作一个独立的进程，相互之间互不干扰的同时，方便系统进行任务调度、监控和管理。这样的管理方式有助于提高系统的灵活性、并发性和效率。

资源分配和共享：此外，进程可以记录任务在运行过程中使用到的各种系统资源，允许系统同时管理多个任务的资源需求。在多用户系统中，多个用户可能需要同时访问共享资源，如内存、文件、设备等。通过引入进程，系统能够为每个任务分配独立的资源空间，同时支持资源的共享和协同工作。

独立性和稳定性：进程之间是相互独立的，一个用户的任务出现问题或崩溃不会影响其他用户的任务，提高了系统的稳定性。每个用户的任务都在独立的进程中执行，互相隔离，一个用户的错误通常不会波及其他用户。

5.1.2　进程的定义及与程序的区别

下面给出进程的概念。

再比较一下程序和进程的概念。程序(Program)是一组指令的集合，它被设计为在计算机上执行特定任务。程序是静态的，通常存储在磁盘上。进程(Process)是程序的一次执行实例。它是计算机系统中的一个活动单元，具有自己的内存空间、寄存器集合、状态等资源。

为了更好地理解程序与进程的区别，首先以烹饪一顿晚餐为例来说明程序和进程的关系。

程序就好像菜谱，菜谱是一组指令，其中包含一系列步骤，所需的原料和烹饪方法。这相当于程序，是一个静态的指南，描述了如何制作一道菜。而进程就是烹饪过程。当你开始按照菜谱的指示动手烹饪时，就启动了一个进程。在这个过程中，你会逐步执行菜谱中的每个步骤，操作烹饪工具，处理食材，直到最终完成一道菜。这个烹饪过程就相当于一个动态执行的进程。因此程序与进程的关系就类似于菜谱(程序)提供了制作菜品的指导，但它本身并没有生命，不会产生实际的变化。只有当你根据菜谱开始烹饪时，才产生了一个实际的烹饪过程(进程)。多个人可以根据同一份菜谱制作相同的菜品，每个人的烹饪过程都是独立的进程。在这个比喻中，菜谱类比于程序，它是一个静态的指南。而烹饪过程类比于进

程，是动态执行的实例。程序为进程提供了指令和流程，而进程是根据这些指令和流程执行的实际活动。这个例子可以帮助理解程序和进程之间的关系，以及程序如何通过执行变成实际的活动。

因此，程序和进程二者的最本质的区别是：程序是静态的，它只是存储在磁盘上的一组指令，不涉及实际的执行。而进程是动态的，它是程序在执行过程中的实例，具有运行状态、资源占用等。我们知道，在硬盘上的程序是无法被 CPU 直接运行的，因此必须将硬盘上的程序装入内存后才能被 CPU 执行。程序被装入内存后，就要接受操作系统的管理，因此操作系统必须为管理装入内存中的程序建立复杂的数据结构对程序进行管理。同时在程序运行后，会访问计算机系统的很多资源。因此在内存中的程序和数据、管理程序运行的数据结构以及程序运行过程中访问的各种资源就构成了进程。由此可见，进程是一个动态概念。

5.1.3 进程的构成

1. 进程控制块

进程是计算机系统中的执行实体，它由程序、数据和进程控制块三部分构成。程序和数据来自于硬盘上的文件。进程控制块(Process Control Block，PCB)存储进程的相关信息。进程控制块是操作系统中用于管理和维护进程信息的数据结构。PCB 中的信息可以分为多个类别，涵盖了进程的状态、标识、调度信息以及与其他进程的通信等方面。一般而言，以下是 PCB 中的主要信息类别。

1) 进程标识信息

进程标识符(Process ID，PID)：唯一标识系统中的每个进程。

父进程标识符(Parent Process ID，PPID)：标识创建当前进程的父进程。

2) 进程状态信息

进程状态(Process State)：描述进程当前的状态，如运行、就绪、阻塞等。

程序计数器(Program Counter，PC)：存储下一条要执行的指令地址。

寄存器集合：包含进程的寄存器值，包括通用寄存器、程序计数器等。

3) 进程调度和优先级信息

调度状态：描述进程的调度状态，例如，是否可抢占、调度优先级等。

调度器所需的信息：与进程调度相关的信息，如进程的优先级、时间片大小等。

4) 进程控制信息

进程控制状态：包括挂起、终止等标志，表示进程的当前控制状态。

信号处理信息：进程对各种信号的处理方式，如忽略、捕获、默认处理等。

5) 资源管理信息

打开文件表：进程打开的文件列表。

内存管理信息：包括进程的内存分配情况、页面表等。

文件描述符表：记录文件的使用情况，包括文件位置、权限等。

6) 进程通信信息

进程间通信(Inter-Process Communication，IPC)：进程与其他进程通信的相关信息，如消息队列、共享内存等。

7）计时和统计信息

运行时间：记录进程已经运行的时间。

CPU 时间统计：统计进程使用 CPU 的时间。

8）异常和中断处理信息

异常处理表：记录进程在发生异常时应该执行的处理程序。

中断处理表：记录进程在接收到中断时应该执行的处理程序。

这些信息类别构成了 PCB 的主要组成部分，它们共同维护了操作系统对进程的管理和控制。不同的操作系统可能在 PCB 中包含不同的具体信息，但上述类别是常见的。PCB 的目的是保持和管理进程的各种状态和属性，以支持多任务处理。

2. 进程的特征

进程是操作系统中的一个重要概念，具有多个特征，其中一些主要特征如下。

独立性(Independence)：进程是系统中的独立执行单元，它们相互之间是隔离的，各自拥有自己的地址空间、资源和执行环境。进程的运行不受其他进程的影响，保证了系统的稳定性和安全性。

动态性(Dynamicity)：进程是动态创建和销毁的，系统可以根据需要创建新的进程或终止已有的进程。进程的创建和销毁是根据用户的操作或应用程序的需求动态进行的，具有灵活性和可变性。

并发性(Concurrency)：系统中可以同时存在多个进程，并且它们可以并发执行，即在同一时间段内，多个进程可以同时运行。并发性使得系统能够更有效地利用计算资源，提高系统的吞吐量和响应速度。

独立调度(Individual Scheduling)：操作系统对每个进程进行独立调度和管理，根据进程的状态和优先级进行调度决策。不同进程的调度是相互独立的，系统根据需要为每个进程分配 CPU 时间，实现公平和高效的资源分配。

有状态性(Stateful)：进程具有多种状态，包括新建、就绪、运行、阻塞和终止等状态。进程在其生命周期中会经历不同的状态转换，这些状态反映了进程在不同阶段的行为和状态。

共享性(Sharing)：进程之间可以共享资源，如内存、文件、设备等。通过进程间的通信机制，不同进程可以共享数据和信息，实现协作和协同工作。

持久性(Persistence)：进程的执行是持久的，即使在系统重启或断电后，系统也会尽力恢复之前正在执行的进程。进程的持久性确保了系统能够长时间地运行应用程序，保证了应用程序的可靠性和连续性。

这些特征共同构成了进程的基本属性，进程作为操作系统中的核心概念，为系统提供了管理和调度计算任务的基础。理解这些特征有助于深入理解操作系统的工作原理和行为。

5.1.4　进程的状态

在操作系统中，进程可以处于不同的状态，这些状态反映了进程在其生命周期中的不同情况和行为。常见的进程状态包括以下几种。

新建(New)：进程刚刚被创建，但尚未分配到 CPU 时间，处于等待系统分配资源的状态。

就绪(Ready)：进程已经准备好运行，等待系统调度器分配 CPU 时间给它。处于就绪状态的进程已经具备了运行所需的全部资源，只需等待 CPU 时间片的分配。

运行(Running)：进程正在 CPU 上执行指令，处于运行状态。在任意时刻，只有一个进程可以处于运行状态，其他进程可能处于就绪、阻塞或挂起等状态。

阻塞(Blocked)：进程暂时无法继续执行，因为它正在等待某个事件的发生(如 I/O 操作完成、资源可用等)。处于阻塞状态的进程暂时从可运行队列中移除，直到等待的事件发生才能转为就绪状态。

终止(Terminated)：进程执行完成或被终止，不再执行任何指令。进程完成其任务或被操作系统强制终止后，进程将处于终止状态。在此状态下，进程所占用的资源会被释放，PCB(进程控制块)等进程管理数据结构会被销毁。

有些系统还可能存在其他特殊的进程状态，如挂起(Suspended)状态。挂起状态是指进程暂时被挂起，并且不占用系统资源，直到满足某些条件后再恢复执行。

进程**挂起状态(Suspended State)**是指进程暂时被挂起，不再占用 CPU 时间，也不参与系统调度。在挂起状态下，进程的执行被暂停，并且其占用的系统资源(如内存)被保留，但不会消耗 CPU 时间。进程通常会进入挂起状态的原因如下。

等待事件：进程可能因为等待某个事件的发生而被挂起，如等待用户输入、等待 I/O 操作完成等。在事件发生之前，进程无法继续执行，因此被暂时挂起。

内存不足：在内存不足的情况下，操作系统可能会将部分进程挂起，以释放内存资源给其他进程使用。在这种情况下，挂起的进程可能会被放置在磁盘上的交换空间中，以腾出物理内存空间。

被其他进程挂起：某些操作系统允许一个进程主动挂起其他进程，将其置于挂起状态。在这种情况下，挂起的进程可能是因为调度策略、资源争用或其他原因而暂时停止执行。

进程挂起状态可以分为以下两种类型。

内存挂起(Memory Suspended)：进程的执行状态被暂时挂起，但其占用的内存资源仍然保留在内存中。这种挂起状态通常是由于等待事件或其他进程挂起而导致的。

磁盘挂起(Disk Suspended)：进程的执行状态被暂时挂起，并且其占用的内存资源被移动到磁盘的交换空间中。这种挂起状态通常是由于内存不足而导致的，操作系统将进程的内存映像写入磁盘，以释放物理内存空间。

在进程进入挂起状态后，一旦满足了挂起的条件或事件，操作系统可以将进程恢复到就绪状态，允许其继续执行。挂起状态的使用有助于操作系统更有效地管理系统资源，提高系统的整体性能和响应能力。

在不同的操作系统中，进程状态的表示和命名可能有所不同，但通常都会包含上述基本状态。了解和理解进程的状态有助于操作系统更有效地管理进程资源，优化系统的性能和响应能力。

5.1.5 进程的创建与终止

1. 进程创建

进程的创建是指在操作系统中启动一个新的进程。进程创建通常由其他进程(父进程)

或操作系统本身发起。以下是进程创建的一般步骤和过程。

分配进程标识符：操作系统为新进程分配一个唯一的标识符，用于在系统中唯一标识该进程。

分配进程控制块：操作系统为新进程创建一个进程控制块，用于存储和管理进程的信息，包括进程状态、程序计数器、寄存器值、文件描述符、优先级等。

分配资源：操作系统为新进程分配所需的资源，包括内存空间、文件描述符、设备等。这些资源可能是从父进程继承的，也可能是操作系统根据进程的需求分配的。

加载执行程序：新进程通常需要加载执行一个程序或代码，这可以是一个可执行文件、一个脚本或一个已编译的程序。操作系统负责将程序加载到进程的地址空间中，并设置执行环境。

初始化进程状态：新进程被创建后，它的状态通常被设置为就绪状态，表示进程已经准备好运行。然后操作系统将新进程放入就绪队列，等待系统调度器分配 CPU 时间给它。

执行进程：一旦新进程被调度到 CPU 上执行，它开始执行其加载的程序或代码。进程可以执行各种操作，包括计算、I/O 操作、进程间通信等。

进程的创建可以由如下多种方式触发。

系统调用：某些系统调用(如 fork()、exec())可以创建新进程。

用户请求：用户可以启动新的进程，如通过命令行启动一个程序。

系统初始化：操作系统启动时可能会创建一些初始化进程，用于系统初始化和服务启动。

无论是哪种方式，进程创建是操作系统中的一个重要操作，它为系统提供了多任务处理的基础，使得多个程序可以同时运行并协同工作。

2. 进程终止

1) 正常终止

进程正常终止是指进程完成了其任务或工作，并主动退出执行。进程正常终止的过程通常包括以下步骤。

执行完成任务：进程在执行期间完成了其分配的任务或工作。这可能涉及计算、数据处理、文件操作、网络通信等各种操作。

释放资源：进程在终止前需要释放其使用的资源，包括内存、文件描述符、设备等。这确保了系统能够将资源分配给其他进程使用。

发送退出信号：进程在准备退出时通常会发送一个退出信号给其父进程或操作系统。这个信号可以告知父进程或操作系统进程已经完成了其任务，并准备退出执行。

清理工作：在发送退出信号后，进程可能会执行一些清理工作，如关闭打开的文件、释放动态分配的内存、发送其他信号等。

终止进程：进程在完成清理工作后，调用系统调用 exit()或 return 来终止自己的执行。这导致进程的代码段和数据段被操作系统回收，并释放进程控制块等资源。

回收资源：操作系统在收到进程终止信号后，会回收进程的资源，包括释放进程占用的内存、关闭文件描述符、更新进程状态等。

进程正常终止是进程生命周期的一部分，它确保了系统能够及时释放资源，提高系统的

效率和稳定性。在进程终止时，系统可以进行必要的清理和回收工作，以便为新的进程提供资源。

2）异常终止

进程异常终止是指进程在执行过程中遇到了无法处理的异常情况，导致进程被强制终止执行的情况。进程异常终止通常是由于以下一些情况引起的。

内存访问错误（Memory Access Violation）：当进程尝试访问未分配的内存区域、越界访问数组，或者访问受保护的内存区域时，操作系统会发出内存访问错误信号（如段错误，Segmentation Fault），导致进程异常终止。

除零错误（Division by Zero）：当进程尝试对一个数除以零时，会引发除零错误，导致进程异常终止。

非法指令（Illegal Instruction）：当进程尝试执行操作系统不支持的指令或者访问特权指令时，会导致非法指令异常，进程被异常终止。

系统资源耗尽（Resource Exhaustion）：当进程申请的系统资源（如内存、文件描述符、网络连接等）耗尽时，无法继续执行，操作系统会将进程异常终止。

系统调用失败（System Call Failure）：当进程调用系统提供的服务失败，如文件打开失败、网络连接失败等，可能会导致进程异常终止。

硬件故障（Hardware Failure）：当进程所在的计算机硬件发生故障时，可能导致进程异常终止。例如，CPU 故障、内存故障、磁盘故障等。

信号处理失败（Signal Handling Failure）：当进程无法处理收到的信号，或者信号处理器本身发生错误时，可能会导致进程异常终止。

进程异常终止是一种不可预测的情况，会导致进程突然停止执行，可能会丢失数据或造成系统不稳定。为了提高系统的可靠性和稳定性，程序员需要编写健壮的代码，处理可能发生的异常情况，以及监控系统状态并及时处理异常事件。

5.2 进程调度策略与进程间通信机制

5.2.1 基本原理

进程调度是操作系统中的一个重要组成部分，它负责决定哪些进程可以在 CPU 上执行以及何时执行。进程调度的原理通常由调度算法决定，这些算法旨在实现公平性、高效性和响应性。进程调度的原理涉及调度队列、调度算法、上下文切换和调度策略等多个方面。通过合理设计和选择调度算法，可以实现系统的公平性、高效性和响应性，提高系统的性能和用户体验。

调度队列：系统维护了多个调度队列，通常包括就绪队列（Ready Queue）、等待队列（Waiting Queue）等。就绪队列中存放着可以立即执行的进程，而等待队列中存放着等待某些事件发生或资源释放的进程。

调度算法：调度算法是决定哪个进程被选中执行的核心。常见的调度算法包括先来先服务（First-Come，First-Served，FCFS）、最短作业优先（Shortest Job First，SJF）、优先级调

度(Priority Scheduling)、轮转调度(Round Robin Scheduling)等。每种调度算法都有其优缺点,可以根据不同的应用场景选择合适的算法。

上下文切换:当操作系统决定要执行一个新的进程时,会进行上下文切换(Context Switching)。这包括保存当前进程的状态(如寄存器、程序计数器等),并加载下一个进程的状态。上下文切换会引入一定的开销,因此调度算法的效率也与上下文切换的频率有关。

调度策略:操作系统可能会采用不同的调度策略来满足不同的需求,例如,时间片轮转、多级反馈队列调度等。调度策略可以根据系统负载、进程优先级等动态调整,以提高系统的性能和响应性。

调度器(Scheduler):调度器是负责执行调度算法的组件,它会周期性地检查就绪队列中的进程,并根据选定的调度算法选择下一个要执行的进程。调度器通常作为操作系统内核的一部分,并与其他内核组件紧密交互。

1. 优化目标

进程调度策略的设计需要考虑多个因素,以满足不同的系统需求和优化目标。以下是常见的需要考虑的因素。

公平性(Fairness):调度策略应该确保所有进程都能够公平地分享 CPU 时间,避免出现某些进程长时间占用 CPU 而其他进程无法执行的情况。

响应时间(Response Time):调度策略应该尽可能地减少进程的响应时间,使得用户能够快速地得到反馈。尤其是对交互式应用和实时系统而言,响应时间是一个重要的指标。

吞吐量(Throughput):调度策略应该能够最大化系统的吞吐量,即在单位时间内完成的进程数量。高吞吐量意味着系统能够有效地利用资源,提高系统的性能。

等待时间(Waiting Time):调度策略应该尽量减少进程的等待时间,避免进程长时间处于就绪状态而无法执行。减少等待时间可以提高系统的效率和用户体验。

周转时间(Turnaround Time):周转时间是指从进程提交到执行完成所经历的时间。调度策略应该尽可能地减少进程的周转时间,以提高系统的效率和性能。

优先级(Priority):调度策略可以根据进程的优先级来确定执行顺序,确保重要的进程优先执行。优先级通常根据进程的类型、资源需求和用户指定进行调整。

资源利用率(Resource Utilization):调度策略应该尽可能地提高系统资源的利用率,避免资源空闲或浪费。合理的调度策略可以确保系统中的各项资源得到充分利用。

上下文切换开销(Context Switching Overhead):调度策略的设计应该考虑减少上下文切换的开销,以提高系统的效率。频繁的上下文切换会增加系统的负担,降低系统的性能。

综上所述,进程调度策略的设计需要综合考虑公平性、响应时间、吞吐量、等待时间、周转时间、优先级、资源利用率和上下文切换开销等多个因素,以实现系统的高效性、公平性和响应性。

2. 基本算法

进程调度策略是操作系统中用于决定哪些进程应该被选中执行的规则和算法。不同的调度策略旨在实现不同的目标,如提高系统吞吐量、减少平均等待时间、优化资源利用等。

以下是常见的进程调度策略。

先来先服务调度(FCFS):这是最简单的调度策略之一,按照进程到达的顺序来进行调度。当一个进程到达就绪队列时,它被放置在队列的末尾。调度器选择队列中最早到达的进程执行。FCFS 策略简单直观,但可能导致长作业(长时间运行的进程)等待时间过长,影响系统的响应性。

最短作业优先调度(SJF):SJF 策略选择具有最短执行时间的进程先执行。它需要事先知道每个进程的执行时间,因此通常用于批处理系统或可预测性较强的场景。SJF 策略可以最小化平均等待时间,但可能导致长作业长时间等待,而且需要准确预测每个作业的执行时间。

优先级调度:每个进程被分配一个优先级,调度器选择优先级最高的进程执行。优先级可以是静态的(由用户或系统设置)或动态的(根据进程状态和需求动态调整)。优先级调度可以根据系统需求分配资源,但可能导致低优先级进程长时间等待高优先级进程,产生饥饿问题。

轮转调度:轮转调度将 CPU 时间分割成固定长度的时间片,每个进程在一个时间片内执行一定的时间,然后被放回就绪队列等待。如果一个进程在时间片结束时没有完成,则它被放回队列的末尾。轮转调度保证了公平性和响应性,但可能导致上下文切换开销增加。

多级反馈队列调度(Multilevel Feedback Queue Scheduling):这是一种结合了优先级和轮转调度的策略。系统维护多个优先级队列,每个队列有不同的时间片大小。新进程首先进入最高优先级队列,如果没有完成,则降低优先级,并进入下一个队列。这种调度策略可以在不同的场景下实现公平性和响应性。

以上是常见的进程调度策略,每种策略都有其优缺点和适用场景。操作系统通常会根据系统特点和需求选择合适的调度策略,以达到最佳的性能和用户体验。

5.2.2 进程通信

1. 引入的动机

引入进程间通信(Inter-Process Communication,IPC)机制的动机主要包括以下几个方面。

模块化和分布式处理:在复杂的软件系统中,不同的功能通常由不同的模块或进程来实现,引入 IPC 可以让这些模块之间相互通信,实现功能的模块化和分布式处理。

并发和并行处理:当系统需要同时处理多个任务或并发执行多个操作时,引入 IPC 可以实现进程之间的同步和协作,充分利用系统资源,提高系统的并发性和并行性。

资源共享和数据交换:在多进程或多线程的系统中,引入 IPC 可以实现进程之间的数据共享和交换,避免数据冗余和不一致,提高系统的效率和性能。

解耦合和灵活性:使用 IPC 可以将系统中的不同模块解耦合,降低模块之间的依赖性,提高系统的灵活性和可维护性。这样可以使系统更易于扩展和修改,降低系统的维护成本。

分布式系统和网络通信:在分布式系统和网络应用中,不同计算节点之间需要进行数据交换和通信,引入 IPC 可以实现进程间的远程通信和数据传输,支持分布式系统的构建和应用。

实时系统和通信控制:在实时系统中,进程之间的通信需要满足严格的时序要求和实

时性要求，引入 IPC 可以实现进程间的实时数据交换和通信控制，保证系统的可靠性和稳定性。

综上所述，引入进程间通信机制可以实现模块化和分布式处理、并发和并行处理、资源共享和数据交换、解耦合和灵活性、分布式系统和网络通信以及实时系统和通信控制等目标，是构建复杂软件系统和实现高性能分布式应用的重要手段。

2. 基本概念

进程间通信是指两个或多个进程之间进行数据交换和信息传递的机制。实现 IPC 的目的是使不同进程之间能够相互协作、共享数据和完成任务。以下是常见的几种进程间通信机制的原理。

管道(Pipe)：管道是一种单向通信机制，通常用于具有亲缘关系的父子进程间通信。它是由操作系统内核创建的一段内存，其中的数据流向只能是单向的。一个进程通过写入管道，另一个进程通过读取管道来进行通信。管道通常是半双工的，但也可以通过创建两个管道实现全双工通信。

命名管道(Named Pipe)：命名管道是一种特殊的管道，允许无关进程间进行通信。它是一个文件系统路径名，通过文件系统提供的文件访问接口来进行读写操作。命名管道通常用于跨越无关进程之间的通信，可以提供双向通信功能。

信号量(Semaphore)：信号量是一种计数器，用于控制多个进程对共享资源的访问。它允许多个进程在临界区域之间进行同步，并提供了互斥和同步机制。通过对信号量的操作(如增加、减少、等待等)，进程可以协调对共享资源的访问。

消息队列(Message Queue)：消息队列是一种通过操作系统提供的消息缓冲区进行通信的机制。进程可以将消息发送到队列中，另一个进程则可以从队列中接收消息。消息队列通常用于实现异步通信和解耦合。

共享内存(Shared Memory)：共享内存是一种通过在进程间共享内存区域来实现通信的机制。多个进程可以将同一块内存映射到它们的地址空间中，并在其中进行读写操作。共享内存是最快的 IPC 机制之一，但需要额外的同步机制来确保数据一致性和完整性。

套接字(Socket)：套接字是一种用于网络通信的抽象，但也可以在同一台计算机上的进程间进行通信。它提供了一种面向连接的、可靠的双向通信机制，可以实现进程间的数据交换和通信。

这些 IPC 机制各有优缺点，可以根据不同的需求和场景选择合适的机制来实现进程间的通信。通常情况下，多种 IPC 机制可以结合使用，以满足复杂的通信需求。

3. 适用场景

不同的进程间通信机制适用于不同的场景和需求。以下是常见的几种进程间通信机制以及它们适用的场景。

1) 管道(Pipe)

场景：适用于具有父子进程关系的进程间通信，如管道通常用于 Shell 程序中的命令行管道(如 ls|grep)，或者用于父子进程之间的通信。

优点：简单易用，轻量级，适用于一对一的进程通信。

缺点：只能实现单向通信，且只能在有亲缘关系的进程间使用。

2）命名管道（Named Pipe）

场景：适用于无亲缘关系的进程间通信，可以实现双向通信。常用于需要跨越不同进程的通信，如客户端/服务器模型。

优点：支持无亲缘关系的进程间通信，可以实现双向通信。

缺点：有一定的复杂度，需要注意阻塞和非阻塞的处理。

3）信号量（Semaphore）

场景：适用于控制对共享资源的访问，实现多进程之间的同步和互斥。常用于进程池、资源池等场景。

优点：提供了灵活的同步机制，可以实现多进程之间的协作和资源共享。

缺点：需要谨慎设计信号量的计数和使用，避免死锁和资源竞争。

4）消息队列（Message Queue）

场景：适用于需要异步通信和解耦合的场景，常用于不同进程间的消息传递和事件通知。

优点：支持异步通信，可以实现进程之间的解耦合，提高系统的可维护性和扩展性。

缺点：相对于其他通信机制，消息队列通常具有较高的开销，可能会影响系统的性能。

5）共享内存（Shared Memory）

场景：适用于需要高效数据共享和通信的场景，如数据共享、大规模数据交换等。

优点：提供了最快的进程间通信方式，无须数据复制，适用于高性能的数据处理场景。

缺点：需要额外的同步机制来确保数据一致性和完整性，可能会引入复杂性和风险。

6）套接字（Socket）

场景：适用于网络通信和跨网络的进程间通信，如客户端/服务器模型、分布式系统等。

优点：提供了通用的网络通信机制，支持跨网络的进程通信，可以实现分布式系统的构建。

缺点：相对于其他通信机制，套接字通常具有较高的开销和复杂性，可能会增加系统的负担。

根据实际需求和场景的不同，可以选择合适的进程间通信机制来实现进程之间的数据交换和协作。通常情况下，多种通信机制可以结合使用，以满足复杂的通信需求。

5.3 管理进程与调度命令

Linux 是一个多任务系统，经常需要对这些进程进行一些调配和管理。要进行管理，首先就要知道现在的进程情况：有哪些进程、进程的状态如何等。Linux 提供了多种命令来了解进程的状况。

本节主要介绍 Linux 中提供的用于查看进程 PCB 信息的命令，包括查看用户信息的 who 命令，查看状态、优先级、内存管理、资源使用情况等信息的 ps 和 top 命令，以及向进程发送信号，即与进程进行通信的 kill 命令。

5.3.1　进程管理命令

1. who 命令

who 命令主要用于查看当前系统中的用户情况。如果用户想和其他用户建立即时通信，如使用 talk 命令，那么首先要确定的就是该用户确实在线上，否则 talk 进程就无法建立起来。又如，系统管理员希望监视每个登录的用户此时此刻的所作所为，也要使用 who 命令。who 命令应用起来非常简单，可以比较准确地掌握用户的情况，所以使用非常广泛。例如，查看系统中的用户及其状态：

```
$ who
admin       tty1         Jul 28 15:55
admin       pts/0        Aug 5 15:46 (192.168.0.110)
admin       pts/2        Jul 29 19:52 (192.168.0.110)
root        pts/3        Jul 30 12:07 (192.168.0.110)
root        pts/4        Jul 31 10:29 (192.168.0.144)
root        pts/5        Jul 31 14:52 (192.168.0.11)
root        pts/6        Aug 6 10:12 (192.168.0.234)
root        pts/8        Aug 6 11:34 (192.168.0.234)
```

2. ps 命令

ps 命令是最基本又非常强大的进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵尸、哪些进程占用了过多的资源等，大部分进程信息都可以通过执行该命令得到。

ps 命令最常用的还是用来监控后台进程的工作情况，因为后台进程是不与屏幕、键盘这些标准输入/输出设备进行通信的，所以如果需要检测其状况，就可使用 ps 命令。ps 命令的常见选项如表 5-1 所示。

表 5-1　选项说明

选　项	描　述
-e	显示所有进程
-f	全格式
-h	不显示标题
-l	使用长格式
-w	宽行输出
-a	显示终端上的所有进程，包括其他用户的进程
-r	只显示正在运行的进程
-x	显示没有控制终端的进程

例如，显示系统中终端上的所有运行进程，命令如下。

```
$ ps -a
  PID TTY          TIME CMD
12175 pts/6    00:00:00
bash
24526 pts/0    00:00:00
vsftpd
```

```
29478 pts/5    00:00:00 ps
32461 pts/0    1-01:58:33 sh
```

其中，PID 列显示了每个进程在操作系统中的唯一标识符，用于区分不同的进程。TTY 列显示了与进程关联的终端设备或伪终端。TIME 列显示了自进程启动以来已经消耗的 CPU 时间。它通常以分钟:秒钟的格式表示，显示了用户态和内核态的 CPU 时间。用户态时间是进程执行自身代码所消耗的 CPU 时间，而内核态时间是进程执行系统调用和内核操作所消耗的 CPU 时间。CMD 列显示了启动进程的完整命令行。这是用于启动进程的命令，包括执行的程序和任何参数。通过查看 CMD 列可以了解进程是如何被启动的以及正在做什么。

3. top 命令

top 命令和 ps 命令的基本作用是相同的，显示系统当前的进程和其他状况，但是 top 是一个动态显示过程，即可以通过用户按键来不断刷新进程的当前状态，如果在前台执行该命令，它将独占前台，直到用户终止该程序为止。其实 top 命令提供了实时的对系统处理器的状态监视。它将显示系统中 CPU 的任务列表。该命令可以按 CPU 使用、内存使用和执行时间对任务进行排序，而且该命令的很多特性都可以通过交互式命令或者在定制文件中进行设定。top 命令输出的实例如图 5-1 所示。对命令执行结果显示的各列的解释如下。

```
top - 19:04:08 up 9 days,  3:09,  8 users,  load average: 2.17, 2.08, 2.06
Tasks: 242 total,   8 running, 234 sleeping,   0 stopped,   0 zombie
Cpu(s):  8.3%us,  0.2%sy,  0.0%ni, 91.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     19983M total,    19777M used,      206M free,      567M buffers
Swap:     2053M total,       10M used,     2043M free,    12326M cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
32757 root      20   0 4462m 3.0g 5440 S   100 15.3   1542:40 qemu-kvm
32461 root      20   0 11580 1380 1120 R   100  0.0   1563:47 sh
31437 root      20   0 4626m 2.4g 5436 R     4 12.1  14:36.89 qemu-kvm
29553 root      20   0 17256 1392  932 R     0  0.0   0:00.02 top
31438 root      20   0     0    0    0 S     0  0.0   0:12.80 vhost-31437
32758 root      20   0     0    0    0 S     0  0.0   0:25.21 vhost-32757
    1 root      20   0 10540  796  748 S     0  0.0   0:04.59 init
    2 root      20   0     0    0    0 S     0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S     0  0.0   0:01.64 ksoftirqd/0
    6 root      RT   0     0    0    0 S     0  0.0   0:01.08 migration/0
    7 root      RT   0     0    0    0 S     0  0.0   0:01.66 watchdog/0
    8 root      RT   0     0    0    0 S     0  0.0   0:01.09 migration/1
    9 root      20   0     0    0    0 S     0  0.0   0:05.58 kworker/1:0
   10 root      20   0     0    0    0 S     0  0.0   0:01.31 ksoftirqd/1
   11 root      20   0     0    0    0 S     0  0.0   0:50.48 kworker/0:1
   12 root      RT   0     0    0    0 S     0  0.0   0:01.27 watchdog/1
   13 root      RT   0     0    0    0 S     0  0.0   0:01.64 migration/2
   14 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/2:0
   15 root      20   0     0    0    0 S     0  0.0   1:01.89 ksoftirqd/2
   16 root      RT   0     0    0    0 S     0  0.0   0:01.38 watchdog/2
   17 root      RT   0     0    0    0 R     0  0.0   0:01.12 migration/3
   18 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/3:0
   19 root      20   0     0    0    0 S     0  0.0   0:22.84 ksoftirqd/3
   20 root      RT   0     0    0    0 S     0  0.0   0:01.33 watchdog/3
   21 root      RT   0     0    0    0 S     0  0.0   0:01.56 migration/4
   22 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/4:0
   23 root      20   0     0    0    0 S     0  0.0   0:00.01 ksoftirqd/4
   24 root      RT   0     0    0    0 S     0  0.0   0:01.29 watchdog/4
```

图 5-1　top 显示

(1) PID 列显示进程的唯一标识符，用于区分不同的进程。

(2) USER 列显示进程的所有者(用户名)。

(3) PR(Priority)列显示进程的优先级。较小的值表示较高的优先级。

(4) NI(Nice value)列显示进程的 nice 值,用于指定进程的优先级。较大的值表示较低的优先级。

(5) VIRT(Virtual Memory)列显示进程使用的虚拟内存大小,包括进程映像、库、堆栈等。

(6) RES(Resident Memory)列显示进程实际使用的物理内存大小,即进程在物理内存中的部分。

(7) SHR(Shared Memory)列显示进程使用的共享内存大小。

(8) S 列显示进程的状态。常见状态包括 R(运行)、S(睡眠)、Z(僵尸)等。

(9) %CPU(CPU Usage)列显示进程使用的 CPU 时间占总 CPU 时间的百分比。

(10) %MEM(Memory Usage)列显示进程使用的物理内存占系统总内存的百分比。

(11) TIME+列显示进程已经使用的 CPU 时间。

(12) COMMAND(Command)列显示启动进程的完整命令行。

4. kill 命令

当需要中断一个前台进程的时候,通常使用 Ctrl+C 组合键,而对于后台进程不能用组合键来终止,这时就可以使用 kill 命令。该命令可以终止前台和后台进程。终止后台进程的原因包括:该进程占用 CPU 的时间过多、该进程已经死锁等。

kill 命令是通过向进程发送指定的信号来结束进程的。如果没有指定发送的信号,那么默认值为 TERM 信号。TERM 信号将终止所有不能捕获该信号的进程。至于那些可以捕获该信号的进程可能就需要使用 kill 信号(它的编号为 9),而该信号不能被捕捉。kill 命令的语法格式有以下两种。

```
kill [-s 信号 | -p] [-a] 进程号…
kill -l [信号]
```

进程号可以通过 ps 命令的输出得到。-s 选项是给程序发送指定的信号,详细的信号可以用“kill -l”命令查看;-p 选项只显示指定进程的 ID。杀死 pid 为 1409 的进程,在 root 权限下执行如下命令。

```
# kill -9 1409
```

显示所有的信号及其编号对应关系,示例如下。

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

5.3.2 调度启动进程

除了对普通进程管理外，系统管理员还承担着一个非常重要的任务，就是定期或不定期地启动某些对系统进行维护的任务。这些定期或不定期启动的任务是建立在操作系统提供的计时器组件的基础之上的。在 Linux 中，计时器是一种用于测量时间间隔、触发定时事件或延迟执行的机制。Linux 提供了多种类型的计时器，以下是其中几种常见的计时器。

实时时钟(RTC)：实时时钟是计算机硬件上的一个独立计时器，通常由电池供电，即使系统关闭也能保持时间的持久性。它用于存储系统的实时时间，并在系统启动时与系统时钟进行同步。

内核定时器(Kernel Timer)：内核定时器是 Linux 内核中的一个组件，用于在内核级别实现定时功能。它提供了一种在特定时间间隔内触发回调函数的机制，用于执行各种内核任务和操作。内核定时器通常由内核线程或内核模块使用，可以用于处理定时事件、任务调度、时间敏感的操作等。

POSIX 定时器(POSIX Timer)：POSIX 定时器是 Linux 中实现 POSIX 标准定义的定时器接口的机制。它允许应用程序以精确的方式进行时间跟踪和计时。通过使用 POSIX 定时器 API，应用程序可以创建定时器，设置定时器的触发时间、间隔和回调函数，并处理定时器事件。

定时器轮(Timer Wheel)：定时器轮是一种数据结构，用于管理和触发一组定时器。它将一系列定时器按照触发时间划分到不同的槽位中，并周期性地遍历这些槽位，触发到期的定时器。定时器轮常用于高效地管理大量的定时器，并提供快速的定时器触发和处理能力。

本节介绍的 at 和 crontab 命令是用户级别的定时任务工具，它们依赖于操作系统提供的时间管理和任务调度机制。操作系统中的计时器是提供时间参考和基准的底层组件，为 at、cron 和其他时间相关的功能提供支持。at 和 cron 可以利用操作系统的计时器来确定任务的触发时间，以便调度和执行任务。

1. 定时运行一批程序(at)

at 命令是一种定时任务的调度工具，在执行时依赖于操作系统提供的 atd 守护进程来启动和执行任务。atd 守护进程(at daemon)是一个后台进程，负责管理和执行通过 at 命令提交的定时任务。它会在系统启动时自动启动，并一直在后台运行，等待执行计划任务。当用户使用 at 命令提交一个任务时，at 命令会将任务信息传递给 atd 守护进程。atd 守护进程会将任务信息存储在一个队列中，并在指定的时间触发执行。atd 守护进程利用系统的任务调度和执行机制，根据指定的时间来调度和执行任务。atd 守护进程依赖于操作系统提供的时间管理功能和任务调度机制来确保任务在指定的时间执行。它通常使用系统的时钟、计时器和定时中断等功能来确定当前时间，并与任务的执行时间进行比较，以触发任务的执行。

1) at 命令

用户使用 at 命令在指定时刻执行指定的命令序列。该命令至少需要指定一个命令和一个执行时间。at 命令可以只指定时间，也可以时间和日期一起指定。at 命令的语法格式如下。

```
at [ - V] [ - q 队列] [ - f 文件名] [ - mldbv] 时间
at - c 作业 [作业…]
```

2) 设置时间

at 允许使用一套相当复杂的时间指定方法,例如:

(1) 接受在当天的 hh:mm(小时:分钟)式的时间指定。如果该时间已经过去,那么就放存第二天执行。

(2) 使用 midnight(深夜)、noon(中午)、teatime(饮茶时间,一般是下午 4 时)等比较模糊的词语来指定时间。

(3) 采用 12 小时计时制,即在时间后面加上 AM(上午)或者 PM(下午)来说明是上午还是下午。

(4) 指定命令执行的具体日期,指定格式为 month day(月日)或者 mm/dd/yy(月/日/年)或者 dd.mm.yy(日.月.年)。指定的日期必须跟在指定时间的后面。

上面介绍的都是绝对计时法,其实还可以使用相对计时法,这对于安排不久就要执行的命令是很有好处的。指定格式为 now+count time-units,now 就是当前时间,time-units 是时间单位,这里可以是 minutes(分钟)、hours(小时)、days(天)、weeks(星期)。count 是时间的数量,究竟是几天,还是几小时等。还有一种计时方法就是直接使用 today(今天)、tomorrow(明天)来指定完成命令的时间。

下面通过一些例子来说明具体用法。例如,指定在今天下午 4:30 执行某个命令。假设现在时间是中午 12:30,2022 年 3 月 5 日,可用命令格式如下。

```
at 4:30pm
at 16:30
at 16:30 today
at now + 4 hours
at now + 240 minutes
at 16:30 5.3.22
at 16:30 3/5/22
at 16:30 Mar 5
```

以上这些命令表达的意义是完全一样的,所以在安排时间的时候完全可以根据个人喜好和具体情况自由选择。一般采用绝对时间的 24 小时计时法可以避免由于用户自己的疏忽造成计时错误,例如,上例可以写成 at 16:30 3/5/22。

3) 执行权限

对于 at 命令来说,需要定时执行的命令是从标准输入或者使用-f 选项指定的文件中读取并执行的。如果 at 命令是从一个使用 su 命令切换到用户 Shell 中执行的,那么当前用户被认为是执行用户,所有的错误和输出结果都会送给这个用户。但是如果有邮件送出的话,收到邮件的将是原来的用户,也就是登录时 Shell 的所有者。例如,在 3 月 5 日上午 10 点执行 slocate -u 命令。在 root 权限下执行命令如下。

```
# at 10:00 3/5/22
at> slocate - u
at>
[1] + Stopped at 10:00 3/5/22
```

上面的结果中,输入 at 命令之后,会出现提示符 at>,提示用户输入命令,在此输入了

slocate -u，然后按 Enter 键。还可以输入多条命令，当所有要执行的命令输入结束后，按 Ctrl+D 组合键结束 at 命令。

在任何情况下，管理员账户都可以使用这个命令。对于其他用户来说，是否可以使用就取决于/etc/at.allow 和/etc/at.deny 文件。

2. 周期性运行一批程序(cron)

前面介绍 at 命令都会在一定时间内完成一定任务，但是它只能执行一次。也就是说，当指定了运行命令后，系统在指定时间完成任务，以后就不再执行了。但是在很多情况下需要周期性重复执行一些命令，这时候就需要使用 cron 命令来完成任务。

crontab 命令是一个用于管理周期性定时任务的命令行工具，在执行时依赖于操作系统提供的 cron 守护进程。系统启动时，cron 守护进程会自动启动，并开始周期性地运行。cron 守护进程会检查每个用户的 crontab 文件，读取其中的定时任务和时间表。cron 守护进程会根据当前系统时钟和任务的时间表，判断哪些任务需要执行。如果有任务需要执行，cron 守护进程会调用相应的执行程序来执行任务。执行程序会按照任务定义的命令或脚本进行操作。执行完任务后，cron 守护进程会将任务的执行情况记录到系统日志中。cron 守护进程会等待一段时间后再次检查定时任务，继续执行下一轮的任务调度和执行。需要注意的是，cron 守护进程并不会直接修改 crontab 文件。要编辑 crontab 文件，需要使用 crontab 命令来进行操作。当用户使用 crontab 命令修改 crontab 文件后，cron 守护进程会重新加载相应的定时任务和时间表。

1) 运行机制

首先 cron 命令会搜索/var/spool/cron 目录，寻找以/etc/passwd 文件中的用户名命名的 crontab 文件，被找到的这种文件将装入内存。例如，一个用户名为 userexample 的用户，对应的 crontab 文件应该是/var/spool/cron/userexample，即以该用户命名的 crontab 文件存放在/var/spool/cron 目录下面。

cron 命令还将搜索/etc/crontab 文件，这个文件是用不同的格式写成的。cron 启动以后，它将首先检查是否有用户设置了 crontab 文件，如果没有就转入睡眠状态，释放系统资源。所以该后台进程占用资源极少，它每分钟被唤醒一次，查看当前是否有需要运行的命令。

命令执行结束后，任何输出都将作为邮件发送给 crontab 的所有者，或者是/etc/crontab 文件中 MAILTO 环境变量中指定的用户。这是 cron 的工作原理，但是 cron 命令的执行不需要用户干涉，用户只需要修改 crontab 中要执行的命令。

2) crontab 命令

crontab 命令用于安装、删除或者显示用于驱动 cron 后台进程的任务配置。用户把需要执行的命令序列放到 crontab 文件中以获得执行，而且每个用户都可以有自己的 crontab 文件。crontab 命令的常用方法如下。

(1) crontab-u：设置某个用户的 cron 服务，root 用户在执行 crontab 时需要此参数。

(2) crontab-l：列出某个用户 cron 服务的详细内容。

(3) crontab-r：删除某个用户的 cron 服务。

(4) crontab-e：编辑某个用户的 cron 服务。

例如，root 查看自己的 cron 设置，命令如下。

```
# crontab -u root -l
```

3）crontab 文件

在 crontab 文件中输入需要执行的命令和时间。该文件中每行都包括 6 个域，其中前 5 个域是指定命令被执行的时间，最后一个域是要被执行的命令。每个域之间使用空格或者制表符分隔。格式如下。

```
minute hour day-of-month month-of-year day-of-week commands
```

对于每一项的说明如表 5-2 所示。

表 5-2　参数说明

参　数	描　述	参　数	描　述
minute	分钟(0～59)	month	一年的第几个月(1～12)
hour	小时(0～23)	day-of-week	一周的星期几(0～6)，0 代表星期日
day-of-month	一个月的第几天(1～31)	commands	需要执行的命令

这些项都不能为空，必须指定值。除了数字还有几个特殊的符号“*”“/”和“-”“,”。其中，* 代表所有的取值范围内的数字，/代表每的意思，“*/5”表示每 5 个单位，“-”代表从某个数字到某个数字，“,”代表分开几个离散的数字。对于要执行的命令，调用的时候需要写出命令的完整路径。例如，晚上 6 点到 10 点之间每两个小时，在/tmp/test.txt 文件中加入 sleepy 文本。在 crontab 文件中对应的行如下。

```
* 18-22/2 * * * echo "sleepy" >> /tmp/test.txt
```

每次编辑完某个用户的 cron 设置后，cron 自动在/var/spool/cron 下生成一个与此用户同名的文件。此用户的 cron 信息都记录在这个文件中，这个文件是不可以直接编辑的，只可以用 crontab -e 来编辑。用户也可以另外建立一个文件，使用“cron 文件名”命令导入 cron 设置。

假设有个用户名为 userexample，它需要为自己创建一个 crontab 文件。步骤如下。

步骤 1　可以使用任何文本编辑器建立一个新文件，并将向该文件加入需要运行的命令和要定期执行的时间，假设该文件为 ～/userexample.cron。

步骤 2　在 root 权限下使用 crontab 命令安装这个文件，使用 crontab 命令使之成为该用户的 crontab 文件。命令如下。

```
# crontab -u userexample ～/userexample.cron
```

这样 crontab 文件就建立好了，可以转到/var/spool/cron 目录下面查看，发现多了一个 userexample 文件。这个文件就是所需的 crontab 文件。cron 启动后，每过一分钟读一次 crontab 文件，检查是否要执行里面的命令。因此该文件被修改后不需要重新启动 cron 服务。

4）编辑配置文件

cron 服务每分钟不仅要读一次/var/spool/cron 内的所有文件，还需要读一次/etc/crontab，因此通过配置这个文件也能得到 cron 的服务。用 crontab 配置是针对某个用户的，而编辑/etc/crontab 是针对系统的任务。此文件的文件格式如下。

```
SHELL = /bin/bash
PATH = /sbin:/bin:/usr/sbin:/usr/bin
MAILTO = root //如果出现错误,或者有数据输出,数据作为邮件发给这个账号

# For details see man 4 crontabs # Example of job definition:
# . ---------------- minute (0 - 59)
# | . -------------- hour (0 - 23)
# | | . ------------ day of month (1 - 31)
# | | | . -------- month (1 - 12) OR jan,feb,mar,apr …
# | | | | . ---- day of week (0 - 6) (Sunday = 0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user - name command to be executed
```

5.3.3 挂起/恢复进程

在某些情况下,如对进程进行调试和追踪、用户要求、资源竞争、信号处理等,进程可能会被挂起。在 Linux 中,挂起进程(Suspend Process)指的是将正在运行的进程暂停执行,使其进入停止状态。当进程被挂起时,它会停止在 CPU 上的执行,不再消耗 CPU 资源。进程的状态被标记为“停止”(Stopped),进程的所有资源和上下文信息都被保存,但进程暂时无法执行任何操作。挂起进程可以用于暂时中止进程的执行,以便进行其他操作,如调试、暂停任务执行、重新分配系统资源等。挂起/恢复进程一节将介绍 FusionOS 中与之相关联的命令。

作业控制允许进程挂起并可以在需要时恢复进程的运行,被挂起的作业恢复后将从中止处开始继续运行。只要在键盘上按 Ctrl+Z 组合键,即可挂起当前的前台作业。在键盘上按 Ctrl+Z 组合键后,将挂起当前执行的命令 cat。使用 jobs 命令可以显示 Shell 的作业清单,包括具体的作业、作业号以及作业当前所处的状态。

恢复进程执行时,有两种选择:用 fg 命令将挂起的作业放回到前台执行;用 bg 命令将挂起的作业放到后台执行。灵活使用上述命令,将给自己带来很大的方便。

5.4 作业和任务调度

在类 UNIX 系统中,作业管理是指对当前 Shell 会话中正在运行或等待执行的作业进行管理和控制的过程。以下是一些常见的类 UNIX 系统中的作业管理概念和命令。

作业(Job)是由当前 Shell 会话创建的一个或多个进程组成的任务单元。一个作业可以包含一个或多个进程,并且可以在前台或后台执行。

前台作业(Foreground Job)是指在当前的 Shell 会话中直接执行的作业。当用户在命令行输入一个命令并按下 Enter 键时,该命令会被作为前台作业执行,直接占用当前 Shell 会话的控制权,用户不能执行其他命令,直到该作业执行完成或被暂停。

前台作业通常表现为命令行中正在执行的进程或任务,它们可以输出信息到控制台,并且可以接收来自用户的输入。在前台作业执行期间,用户可以通过键盘输入特定的组合键来控制作业的行为,例如,暂停作业、终止作业等。

在前台作业执行期间,用户可以看到作业的执行过程,并且可以实时地与作业进行交

互。由于前台作业占用了当前 Shell 会话的控制权，因此用户不能同时执行其他命令，直到前台作业执行完成或被暂停。

通常情况下，一旦前台作业执行完成或被暂停，控制权会返回给用户，用户可以继续输入其他命令或执行其他任务。

后台作业(Background Job)是指在当前的 Shell 会话中以非阻塞方式执行的作业。当用户在命令行输入一个命令并在命令末尾加上 & 符号后，该命令会被作为后台作业执行，不会占用当前 Shell 会话的控制权，用户可以继续输入其他命令或执行其他任务。

后台作业通常以守护进程的形式运行，不会将执行过程输出到控制台，因此用户不会直接看到其执行过程。后台作业执行完成后，通常会在控制台输出一个消息提示，表明作业执行完成。

作业控制(Job Control)是指在 UNIX 或类 UNIX 操作系统中，对正在运行或等待执行的作业进行管理和控制的一系列操作。这些操作包括将作业置于前台或后台、暂停或恢复作业的执行、终止作业等。作业控制使用户能够更灵活地管理多个任务的执行，并且可以在需要时对作业进行操作。

通过作业控制，用户可以方便地管理和控制当前 Shell 会话中的作业，包括将作业置于前台或后台、暂停或恢复作业的执行，以及终止作业等操作。这些操作使用户能够更有效地管理多个任务的执行，提高工作效率。

每个作业都有一个唯一的作业编号(Job ID)，用于标识和操作作业。以下是一些常用的类 UNIX 系统中的作业管理命令。

1. jobs 命令

jobs 命令用于列出当前 Shell 会话中正在运行或等待执行的作业列表。在 UNIX 或类 UNIX 系统中，用户可以使用 jobs 命令查看当前 Shell 会话中的作业，并且可以使用不同的选项来过滤和控制输出。

```
jobs [options]
```

其中：

-l：显示作业号以及相关的进程号和作业状态的详细信息。

-p：仅显示作业的进程组 ID，不显示作业号。

-n：显示作业编号，而不是作业的全部信息。

-r：仅显示运行中的作业。

-s：仅显示已停止的作业。

示例：显示作业的详细信息，包括进程号。

```
jobs -l
```

列出所有作业：使用 jobs 命令不带任何参数，将列出当前 Shell 会话中的所有作业，包括正在运行的作业和已暂停的作业。

2. fg 命令

fg：将一个或多个作业置于前台执行，并将控制权交给该作业。

```
fg [作业号]
```

其中,作业号是要切换到前台的作业的作业号。如果未提供作业号,则默认将最近运行的作业切换到前台。

示例 1:将最近的后台作业切换到前台运行。

```
fg
```

示例 2:将作业号为 1 的后台作业切换到前台运行。作业号是运行 jobs 命令时所显示的作业号。

```
fg 1
```

3. bg 命令

bg:将一个或多个暂停的作业置于后台继续执行。

```
bg [作业号]
```

其中,作业号是要在后台继续运行的作业的作业号。如果未提供作业号,则默认将最近停止的作业在后台继续运行。

示例:将作业号为 1 的停止作业在后台继续运行。作业号是运行 jobs 命令时所显示的作业号。

```
bg 1
```

4. Ctrl+Z

Ctrl+Z:暂停当前正在前台执行的作业,并将其置于后台。

这些命令使用户能够方便地管理和控制当前 Shell 会话中的作业,包括查看作业列表、切换作业的前台和后台执行状态,以及终止作业的执行。通过作业管理,用户可以更有效地管理和控制多个任务的执行。

5.5 管理服务

5.4 节介绍了对操作系统进程进行管理的一些命令,在操作系统中,还存在着一类特殊的进程,称为后台服务。在 Linux 中,后台服务以守护进程的形式在后台运行。后台服务是在后台运行的长期运行进程,提供各种功能和服务,而无须用户交互。这些后台服务通常在系统启动时启动,并在整个系统运行期间保持活动状态。

systemd 是在 Linux 下,与 SysV 和 LSB 初始化脚本兼容的系统和服务管理器。systemd 使用 socket 和 D-Bus 来开启服务,提供基于守护进程的按需启动策略,支持快照和系统状态恢复,维护挂载和自挂载点,实现了各服务间基于从属关系的一个更为精细的逻辑控制,拥有更高的并行性能。

5.5.1 概念介绍

systemd 开启和监督整个系统是基于 unit 的概念。unit 是由一个与配置文件对应的名

字和类型组成的(例如,avahi. service unit 有一个具有相同名字的配置文件,是守护进程 Avahi 的一个封装单元)。unit 有多重类型,如表 5-3 所示。

表 5-3　unit 说明

unit 名称	扩　展　名	描　　述
Service unit	. service	系统服务
Target unit	. target	一组 systemd units
Automount unit	. automount	文件系统挂载点
Device unit	. device	内核识别的设备文件
Mount unit	. mount	文件系统挂载点
Path unit	. path	在一个文件系统中的文件或目录
Scope unit	. scope	外部创建的进程
Slice unit	. slice	一组用于管理系统进程分层组织的 units
Socket unit	. socket	一个进程间通信的 Socket
Swap unit	. swap	swap 设备或者 swap 文件
Timer unit	. timer	systemd 计时器

所有的可用 systemd unit 类型,可在如表 5-4 所示的路径下查看。

表 5-4　可用 systemd unit 类型

路　　径	描　　述
/usr/lib/systemd/system/	随安装的 RPM 产生的 systemd units
/run/systemd/system/	在运行时创建 systemd units
/etc/systemd/system/	由系统管理员创建和管理的 systemd units

5.5.2　特性说明

1. 更快的启动速度

systemd 提供了比 UpStart 更激进的并行启动能力,采用了 socket/D-Bus activation 等技术启动服务,带来了更快的启动速度。

为了减少系统启动时间,systemd 的目标是:

(1) 尽可能启动更少的进程。

(2) 尽可能将更多进程并行启动。

2. 提供按需启动能力

当 sysvinit 系统初始化的时候,它会将所有可能用到的后台服务进程全部启动运行。并且系统必须等待所有的服务都启动就绪之后,才允许用户登录。这种做法有两个缺点:首先是启动时间过长;其次是系统资源浪费。

某些服务很可能在很长一段时间内,甚至整个服务器运行期间都没有被使用过。如 CUPS,打印服务在多数服务器上很少被真正使用到。读者可能没有想到,在很多服务器上 SSHD 也是很少被真正访问到的。花费在启动这些服务上的时间是不必要的;同样,花费在这些服务上的系统资源也是一种浪费。

systemd 可以提供按需启动的能力,只有在某个服务被真正请求的时候才启动它。当

该服务结束时，systemd 可以关闭它，等待下次需要时再次启动它。

3. 采用 cgroup 特性跟踪和管理进程的生命周期

init 系统的一个重要职责就是负责跟踪和管理服务进程的生命周期。它不仅可以启动一个服务，也能够停止服务。这看上去没有什么特别的，然而在真正用代码实现的时候，或许会发现停止服务比一开始想得要困难。

服务进程一般都会作为守护进程(daemon)在后台运行，为此服务程序有时候会派生(fork)两次。在 UpStart 中，需要在配置文件中正确地配置 expect 小节。这样 UpStart 通过对 fork 系统调用进行计数，从而获知真正的精灵进程的 PID。

cgroup 已经出现了很久，它主要用来实现系统资源配额管理。cgroup 提供了类似文件系统的接口，使用方便。当进程创建子进程时，子进程会继承父进程的 cgroup。因此无论服务如何启动新的子进程，所有的这些相关进程都会属于同一个 cgroup，systemd 只需要简单地遍历指定的 cgroup 即可正确地找到所有的相关进程，将它们逐一停止即可。

4. 启动挂载点和自动挂载的管理

传统的 Linux 系统中，用户可以用/etc/fstab 文件来维护固定的文件系统挂载点。这些挂载点在系统启动过程中被自动挂载，一旦启动过程结束，这些挂载点就会确保存在。这些挂载点都是对系统运行至关重要的文件系统，如 HOME 目录。和 sysvinit 一样，systemd 管理这些挂载点，以便能够在系统启动时自动挂载它们。systemd 还兼容/etc/fstab 文件，可以继续使用该文件管理挂载点。

有时候用户还需要动态挂载点，如打算访问 DVD 内容时，才临时执行挂载以便访问其中的内容，而不访问光盘时该挂载点被取消(umount)，以便节约资源。传统地，人们依赖 autofs 服务来实现这种功能。

systemd 内建了自动挂载服务，无须另外安装 autofs 服务，可以直接使用 systemd 提供的自动挂载管理能力来实现 autofs 的功能。

5. 实现事务性依赖关系管理

系统启动过程是由很多的独立工作共同组成的，这些工作之间可能存在依赖关系，如挂载一个 NFS 文件系统必须依赖网络能够正常工作。systemd 虽然能够最大限度地并发执行很多有依赖关系的工作，但是类似“挂载 NFS”和“启动网络”这样的工作还是存在天生的先后依赖关系，无法并发执行。对于这些任务，systemd 维护一个“事务一致性”的概念，保证所有相关的服务都可以正常启动而不会出现互相依赖，以至于死锁的情况。

6. 与 SysV 初始化脚本兼容

和 UpStart 一样，systemd 引入了新的配置方式，对应用程序的开发也有一些新的要求。如果 systemd 想替代目前正在运行的初始化系统，就必须和现有程序兼容。任何一个 Linux 发行版都很难为了采用 systemd 而在短时间内将所有的服务代码都修改一遍。

systemd 提供了和 sysvinit 以及 LSB initscripts 兼容的特性。系统中已经存在的服务和进程无须修改。这降低了系统向 systemd 迁移的成本，使得 systemd 替换现有初始化系

统成为可能。

7. 能够对系统进行快照和恢复

systemd 支持按需启动，因此系统的运行状态是动态变化的，人们无法准确地知道系统当前运行了哪些服务。systemd 快照提供了一种将当前系统运行状态保存并恢复的能力。

例如，系统当前正运行服务 A 和 B，可以用 systemd 命令行对当前系统运行状况创建快照。然后将进程 A 停止，或者做其他的任意的对系统的改变，如启动新的进程 C。在这些改变之后，运行 systemd 的快照恢复命令，就可立即将系统恢复到快照时刻的状态，即只有服务 A 和 B 在运行。一个可能的应用场景是调试：如服务器出现一些异常，为了调试用户将当前状态保存为快照，然后可以进行任意的操作，如停止服务等。等调试结束，恢复快照即可。

5.5.3　管理系统服务

systemd 提供 systemctl 命令来运行、关闭、重启、显示、启用/禁用系统服务。

1. sysvinit 命令和 systemd 命令

systemd 提供 systemctl 命令与 sysvinit 命令的功能类似。当前版本中依然兼容 service 和 chkconfig 命令，相关说明见表 5-5，但建议用 systemctl 进行系统服务管理。

表 5-5　sysvinit 命令和 systemd 命令对照表

sysvinit 命令	systemd 命令	备　注
service network start	systemctl start network. service	用来启动一个服务（并不会重启现有的服务）
service network stop	systemctl stop network. service	用来停止一个服务（并不会重启现有的服务）
service network restart	systemctl restart network. service	用来停止并启动一个服务
service network reload	systemctl reload network. service	当支持时，重新装载配置文件而不中断等待操作
service network condrestart	systemctl condrestart network. service	如果服务正在运行那么重启它
service network status	systemctl status network. service	检查服务的运行状态
chkconfig network on	systemctl enable network. service	在下次启动时或满足其他触发条件时设置服务为启用
chkconfig network off	systemctl disable network. service	在下次启动时或满足其他触发条件时设置服务为禁用
chkconfig network	systemctl is-enabled network. service	用来检查一个服务在当前环境下被配置为启用还是禁用
chkconfig \-\-list	systemctl list-unit-files \-\-type=service	输出在各个运行级别下服务的启用和禁用情况
chkconfig network \-\-list	ls /etc/systemd/system/ *. wants/network. service	用来列出该服务在哪些运行级别下启用和禁用
chkconfig network \-\-add	systemctl daemon-reload	创建新服务文件或者变更设置时使用

2. 显示所有当前服务

如果需要显示当前正在运行的服务，使用如下命令。

```
systemctl list-units --type service
```

如果需要显示所有的服务(包括未运行的服务)，需要添加-all 参数，使用如下命令。

```
systemctl list-units --type service --all
```

例如，显示当前正在运行的服务，命令如下。

```
$ systemctl list-units --type service
UNIT                          LOAD   ACTIVE     SUB     JOB   DESCRIPTION
atd.service                   loaded active     running         Deferred execution scheduler
auditd.service                loaded active     running         Security Auditing Service
avahi-daemon.service          loaded active     running         Avahi mDNS/DNS-SD Stack
chronyd.service               loaded active     running         NTP client/server
crond.service                 loaded active     running         Command Scheduler
dbus.service                  loaded active     running         D-Bus System Message Bus
dracut-shutdown.service       loaded active     exited          Restore /run/initramfs on shutdown
firewalld.service             loaded active     running         firewalld - dynamic firewall daemon
getty@tty1.service            loaded active     running         Getty on tty1
gssproxy.service              loaded active     running         GSSAPI Proxy Daemon
irqbalance.service            loaded active     running         irqbalance daemon
iscsid.service                loaded activating start   start Open-iSCSI
```

3. 显示服务状态

如果需要显示某个服务的状态，可执行如下命令。

```
systemctl status name.service
```

相关状态显示参数说明如表 5-6 所示。

表 5-6 状态参数说明

参 数	描 述
Loaded	说明服务是否被加载，并显示服务对应的绝对路径以及是否启用
Active	说明服务是否正在运行，并显示时间节点
Main PID	相应的系统服务的 PID 值
CGroup	相关控制组(CGroup)的其他信息

如果需要鉴别某项服务是否运行，可执行如下命令。

```
systemctl is-enabled name.service
```

is-active 命令返回结果的解释如表 5-7 所示。

表 5-7 is-active 命令的返回结果解释

状 态	含 义
active	这个服务正在运行
inactive	这个服务没有运行

同样，如果需要判断某个服务是否被启用，可执行如下命令。

```
systemctl is-enabled name.service
```

is-enabled 命令返回结果的解释如表 5-8 所示。

表 5-8　is-enabled 命令的返回结果解释

状　　态	含　　义
"enabled"	已经通过/etc/systemd/system/目录下的 Alias=别名、.wants/或 .requires/软连接被永久启用
"enabled-runtime"	已经通过/run/systemd/system/目录下的 Alias=别名、.wants/或 .requires/软连接被临时启用
"linked"	虽然单元文件本身不在标准单元目录中，但是指向此单元文件的一个或多个软连接已经存在于/etc/systemd/system/永久目录中
"linked-runtime"	虽然单元文件本身不在标准单元目录中，但是指向此单元文件的一个或多个软连接已经存在于/run/systemd/system/临时目录中
"masked"	已经被/etc/systemd/system/目录永久屏蔽（软连接指向/dev/null 文件），因此 start 操作会失败
"masked-runtime"	已经被/run/systemd/systemd/目录临时屏蔽（软连接指向/dev/null 文件），因此 start 操作会失败
"static"	尚未被启用，并且单元文件中的 "[Install]" 小节中没有可用于 enable 命令的选项
"indirect"	尚未被启用，但是单元文件的 "[Install]" 小节中 Also=选项的值列表非空（也就是列表中的某些单元可能已被启用），或者它拥有一个不在 Also=列表中的其他名称的别名软连接。对于模板单元来说，表示已经启用了一个不同于 DefaultInstance=的实例
"disabled"	尚未被启用，但是单元文件的 "[Install]" 小节中存在可用于 enable 命令的选项
"generated"	单元文件是被单元生成器动态生成的。被生成的单元文件可能并未被直接启用，而是被单元生成器隐含地启用了
"transient"	单元文件是被运行时 API 动态临时生成的。该临时单元可能并未被启用
"bad"	单元文件不正确或者出现其他错误。is-enabled 不会返回此状态，而是会显示一条出错信息。list-unit-files 命令有可能会显示此单元

例如，查看 gdm.service 服务状态，命令如下。

```
# systemctl status gdm.service
gdm.service - GNOME Display Manager Loaded: loaded (/usr/lib/systemd/system/gdm.service;
enabled) Active: active (running) since Thu 2013-10-17 17:31:23 CEST; 5min ago
Main PID: 1029 (gdm)
   CGroup: /system.slice/gdm.service
           ├─1029 /usr/sbin/gdm
           ├─1037 /usr/libexec/gdm-simple-slave --display-id /org/gno
…

 └─1047 /usr/bin/Xorg :0 -background none -verbose -auth /r… Oct 17 17:31:23 localhost
systemd[1]: Started GNOME Display Manager.
```

4. 运行服务

如果需要运行某项服务，请在 root 权限下执行如下命令。

```
systemctl start name.service
```

例如，运行 httpd 服务，命令如下。

```
# systemctl start httpd.service
```

5. 关闭服务

如果需要关闭某个服务，请在 root 权限下执行如下命令。

```
systemctl stop name.service
```

例如，关闭蓝牙服务，命令如下。

```
# systemctl stop bluetooth.service
```

6. 重启服务

如果需要重启某项服务，请在 root 权限下执行如下命令。

```
systemctl restart name.service
```

执行命令后，当前服务会被关闭，但马上重新启动。如果指定的服务当前处于关闭状态，执行命令后，服务也会被启动。例如，重启蓝牙服务，命令如下。

```
# systemctl restart bluetooth.service
```

7. 启用服务

如果需要在开机时启用某个服务，请在 root 权限下执行如下命令。

```
systemctl enable name.service
```

例如，设置 httpd 服务开机时启动，命令如下。

```
# systemctl enable httpd.service
ln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-user.target.
wants/httpd.service'
```

8. 禁用服务

如果需要在开机时禁用某个服务，请在 root 权限下执行如下命令。

```
systemctl disable name.service
```

例如，在开机时禁用蓝牙服务启动，命令如下。

```
# systemctl disable bluetooth.service
Removed /etc/systemd/system/bluetooth.target.wants/bluetooth.service.
Removed /etc/systemd/system/dbus-org.bluez.service.
```

5.5.4 改变运行级别

systemd 用目标(target)替代了运行级别的概念，提供了更大的灵活性，如可以继承一个已有的目标，并添加其他服务，来创建自己的目标。表 5-9 列举了 systemd 下的目标和常见 runlevel 的对应关系。

表 5-9　运行级别和 systemd 目标

运行级别	systemd 目标	描　　述
0	runlevel0. target，poweroff. target	关闭系统
1，s，single	runlevel1. target，rescue. target	单用户模式
2，4	runlevel2. target，runlevel4. target，multi-user. target	用户定义/域特定运行级别。默认等同于 3
3	runlevel3. target，multi-user. target	多用户，非图形化。用户可以通过多个控制台或网络登录
5	runlevel5. target，graphical. target	多用户，图形化。通常为所有运行级别 3 的服务外加图形化登录
6	runlevel6. target，reboot. target	重启系统
emergency	emergency. target	紧急 Shell

查看当前系统默认的启动目标，命令如下。

```
systemctl get - default
```

查看当前系统所有的启动目标，命令如下。

```
systemctl list - units  -- type = target
```

改变系统默认的目标，在 root 权限下执行如下命令。

```
systemctl set - default name. target
```

改变当前系统的目标，在 root 权限下执行如下命令。

```
systemctl isolate name. target
```

改变当前系统为救援模式，在 root 权限下执行如下命令。

```
systemctl rescue
```

改变当前系统为紧急模式，在 root 权限下执行如下命令。

```
systemctl emergency
```

这条命令和“systemctl isolate emergency. target”类似。命令执行后会在串口有如下打印信息。

```
You are in emergency mode. After logging in, type "journalctl - xb" to viewsystem logs,
"systemctl reboot" to reboot, "systemctl default" or "exit" to boot into default mode.
Give root password for maintenance
(or press Control - D to continue):
```

5.5.5　关闭、暂停和休眠系统

1. systemctl 命令

systemd 通过 systemctl 命令可以对系统进行关机、重启、休眠等一系列操作。当前仍兼容部分 Linux 常用管理命令，对应关系见表 5-10。建议用户使用 systemctl 命令进行操作。

表 5-10　命令对应关系

Linux 常用管理命令	systemctl 命令	描　　述
halt	systemctl halt	关闭系统
poweroff	systemctl poweroff	关闭电源
reboot	systemctl reboot	重启

2. 关闭系统

关闭系统并下电，在 root 权限下执行如下命令。

```
systemctl poweroff
```

关闭系统但不下电机器，在 root 权限下执行如下命令。

```
systemctl halt
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，可以添加“--no-wall”参数。具体命令如下。

```
systemctl --no-wall reboot
```

3. 重启系统

重启系统，在 root 权限下执行如下命令。

```
systemctl reboot
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，可以添加“--no-wall”参数。具体命令如下。

```
systemctl --no-wall reboot
```

4. 使系统待机

使系统待机，在 root 权限下执行如下命令。

```
systemctl suspend
```

5. 使系统休眠

使系统休眠，在 root 权限下执行如下命令。

```
systemctl hibernate
```

使系统待机且处于休眠状态，在 root 权限下执行如下命令。

```
systemctl hybrid-sleep
```

5.6　管理软件包

5.3 节和 5.5 节分别介绍了一般进程和守护进程的管理命令。本节介绍操作系统中与进程密切相关的“程序”的管理。程序是一组指令的集合，它们以特定的顺序和方式编写，用

于执行特定的任务。程序通常存储在硬盘或其他存储介质上，并且需要加载到内存中才能执行。程序是静态的代码集合，而进程是程序在计算机上运行的实例，具有自己的执行环境和状态。程序是进程的来源，而进程是程序的执行状态。

FusionOS 采用 DNF(Dandified YUM)作为软件包管理工具。DNF 用于在基于 RPM 的 Linux 发行版中进行软件包的安装、升级和卸载。它是作为 YUM(Yellowdog Updater Modified)的后继者而开发的。DNF 软件包管理的原理如下。

仓库配置：DNF 通过配置仓库来获取软件包信息。仓库可以是本地的软件包文件夹或远程的软件包源。DNF 会根据配置文件中的仓库信息获取软件包列表、版本信息和依赖关系等。

依赖解析：在安装或升级软件包时，DNF 会解析软件包的依赖关系。它会检查软件包所需的其他软件包，以确保这些依赖关系已经满足或可以解决。如果存在未满足的依赖关系，DNF 将尝试自动解决它们，或者报告错误。

解析事务：DNF 会将用户的操作(如安装、升级或卸载软件包)转换为一个事务。事务包含一系列要执行的操作步骤。DNF 会考虑依赖关系、版本冲突等因素，确保事务执行的正确性。

事务执行：DNF 会执行生成的事务。它会按照顺序执行每个操作步骤，包括下载软件包、安装或卸载软件包、更新软件包数据库等。DNF 还会记录事务执行的结果，以便用户可以查看操作的状态和详细信息。

事务验证：在事务执行完成后，DNF 会进行验证以确保操作的正确性。它会检查软件包的完整性、依赖关系和文件冲突等。如果验证失败，DNF 会回滚事务并报告错误。

DNF 的优势在于它对软件包依赖关系的处理能力更强大，具有更好的解析算法和依赖解决方案。它还支持模块化软件包管理和扁平命名空间等高级功能。通过 DNF，用户可以方便地管理和维护系统上的软件包，并轻松解决依赖关系问题。

需要说明的是，DNF 与 yum 完全兼容，提供了 yum 兼容的命令行以及为扩展和插件提供的 API。使用 DNF 需要管理员权限，本节所有命令需要在管理员权限下执行。

5.6.1　配置 DNF

1. DNF 配置文件

DNF 的主要配置文件是/etc/dnf/dnf.conf，该文件包含以下两部分。

(1) main 部分保存着 DNF 的全局设置。

(2) repository 部分保存着软件源的设置，可以有一个或多个“repository”。

此外，在/etc/yum.repos.d 目录中还保存着一个或多个 repo 源相关文件，它们也可以定义不同的 repository。所以 FusionOS 软件源的配置一般有两种方式，一种是直接配置/etc/dnf/dnf.conf 文件中的 repository 部分，另外一种是在/etc/yum.repos.d 目录下增加.repo 文件。

1) 配置 main 部分

/etc/dnf/dnf.conf 文件包含的 main 部分，配置示例如下。

```
[main]
gpgcheck = 1
installonly_limit = 3
clean_requirements_on_remove = True
best = True
skip_if_unavailable = False
```

配置参数说明如表 5-11 所示。

表 5-11 main 配置参数说明

参 数	说 明
clean_requirements_on_remove	删除在 dnf remove 期间不再使用的依赖项，如果软件包是通过 DNF 安装的，而不是通过显式用户请求安装的，则只能通过 clean_requirements_on_remove 删除软件包，即它是作为依赖项引入的。默认值为 True
best	升级包时，总是尝试安装其最高版本，如果最高版本无法安装，则提示无法安装的原因并停止安装。默认值为 True
gpgcheck	可选值 1 和 0，设置是否进行 gpg 校验。默认值为 1，表示需要进行校验
installonly_limit	设置可以同时安装 installonlypkgs"指令列出包的数量。默认值为 3，不建议降低此值
skip_if_unavailable	可选值 True 和 False，用来控制元数据仓库不可用时的行为。默认值为 False，当元数据仓库不可用时会停止存储并报错

2）配置 repository 部分

repository 部分允许用户定义定制化的 FusionOS 软件源仓库，各个仓库的名称不能相同，否则会引起冲突。配置 repository 部分有两种方式，一种是直接配置/etc/dnf/dnf.conf 文件中的 repository 部分，另一种是配置/etc/yum.repos.d 目录下的.repo 文件。

（1）直接配置/etc/dnf/dnf.conf 文件中的 repository 部分。

下面是[repository]部分的一个最小配置示例。

```
[repository]
name = repository_name
baseurl = repository_url
```

参数说明如表 5-12 所示。

表 5-12 repository 参数说明

参 数	说 明
name＝repository_name	软件仓库(repository)描述的字符串
baseurl＝repository_url	软件仓库(repository)的地址。 • 使用 HTTP 的网络位置，如 http://path/to/repo。 • 使用 FTP 的网络位置，如 ftp://path/to/repo。 • 本地位置，如 file://path/to/local/repo

（2）配置/etc/yum.repos.d 目录下的.repo 文件。

使用管理员权限添加 FusionOS repo 源，示例如下，请按实际修改参数。

```
# vi /etc/yum.repos.d/FusionOS.repo
[OS]
name = repository_name
baseurl = repository_url
```

```
enabled = 1
gpgcheck = 1
gpgkey = gpgkey_url
```

其中，enabled 为是否启用该软件源仓库，可选值为 1 和 0。默认值为 1，表示启用该软件源仓库。gpgkey 为验证签名用的公钥。

3）显示当前配置

（1）要显示当前的配置信息。

```
dnf config-manager --dump
```

（2）要显示相应软件源的配置，首先查询 repo id。

```
dnf repolist
```

然后执行如下命令，显示对应 id 的软件源配置，其中，repository 为查询得到的 repo id。

```
dnf config-manager --dump repository
```

（3）也可以使用一个全局正则表达式，来显示所有匹配部分的配置。

```
dnf config-manager --dump glob_expression
```

2. 创建本地软件源仓库

要建立一个本地软件源仓库，请按照下列步骤操作。

步骤 1　安装 createrepo 软件包。在 root 权限下执行如下命令。

```
dnf install createrepo
```

步骤 2　将需要的软件包复制到一个目录下，如/mnt/local_repo/。

步骤 3　创建软件源，执行以下命令。

```
createrepo --database /mnt/local_repo
```

3. 添加、启用和禁用软件源

本节将介绍如何通过 dnf config-manager 命令添加、启用和禁用软件源仓库。

1）添加软件源

要定义一个新的软件源仓库，可以在/etc/dnf/dnf.conf 文件中添加 repository 部分，或者在/etc/yum.repos.d/目录下添加.repo 文件进行说明。建议通过添加.repo 的方式，每个软件源都有自己对应的.repo 文件，下面介绍该方式的操作方法。

要在系统中添加一个这样的源，请在 root 权限下执行如下命令，执行完成之后会在/etc/yum.repos.d/目录下生成对应的 repo 文件。其中，repository_url 为 repo 源地址。

```
dnf config-manager --add-repo repository_url
```

2）启用软件源

要启用软件源，请在 root 权限下执行如下命令，其中，repository 为新增.repo 文件中的 repo id(可通过 dnf repolist 查询)。

```
dnf config-manager --set-enable repository
```

也可以使用一个全局正则表达式，来启用所有匹配的软件源。其中，glob_expression为对应的正则表达式，用于同时匹配多个 repo id。

```
dnf config-manager --set-enable glob_expression
```

3）禁用软件源

要禁用软件源，请在 root 权限下执行如下命令。

```
dnf config-manager --set-disable repository
```

同样地，也可以使用一个全局正则表达式来禁用所有匹配的软件源。

```
dnf config-manager --set-disable glob_expression
```

5.6.2 管理软件包

使用 DNF 能够让用户方便地进行查询、安装、删除软件包等操作。

1. 搜索软件包

可以使用 rpm 包名称、缩写或者描述搜索需要的 rpm 包，使用命令如下。

```
dnf search term
```

示例如下。

```
$ dnf search httpd
Last metadata expiration check: 0:35:14 ago on Wed 06 Apr 2022 09:08:27 PM CST.
========================= Name Exactly Matched: httpd ============================
httpd.aarch64 : Apache HTTP Server
=================== Name & Summary Matched: httpd ======================== ====
libmicrohttpd-help.noarch : This help package for libmicrohttpd
======================== Name Matched: httpd ================================
httpd-filesystem.noarch : The basic directory for HTTP Server
httpd-help.noarch : Documents and man pages for HTTP Server
httpd-tools.aarch64 : Related tools for use HTTP Server
libmicrohttpd.aarch64 : Lightweight library for embedding a webserver in applications
```

2. 列出软件包清单

要列出系统中所有已安装的以及可用的 rpm 包信息，使用命令如下。

```
dnf list all
```

要列出系统中特定的 rpm 包信息，使用命令如下。

```
dnf list glob_expression…
```

示例如下。

```
$ dnf list httpd
Last metadata expiration check: 0:36:00 ago on Wed 06 Apr 2022 09:08:27 PM CST.
Available Packages
httpd.aarch64                2.4.43-12.u5.fos22            base
```

3. 显示 rpm 包信息

要显示一个或者多个 rpm 包信息，使用命令如下。

```
dnf info package_name…
```

例如搜索，命令如下。

```
 $ dnf info httpd
Last metadata expiration check: 0:36:46 ago on Wed 06 Apr 2022 09:08:27 PM CST.
Available Packages
Name         : httpd
Version      : 2.4.43
Release      : 12.u5.fos22
Architecture : aarch64
Size         : 1.2 M
Source       : httpd-2.4.43-12.u5.fos22.src.rpm
Repository   : base
Summary      : Apache HTTP Server
URL          : https://httpd.apache.org/
License      : ASL 2.0
Description  : Apache HTTP Server is a powerful and flexible HTTP/1.1 compliant web server.
```

4. 安装 rpm 包

要安装一个软件包及其所有未安装的依赖，请在 root 权限下执行如下命令。

```
dnf install package_name
```

也可以通过添加软件包名字同时安装多个软件包。配置文件/etc/dnf/dnf.conf 添加参数 strict=False，运行 dnf 命令参数添加--setopt=strict=0。请在 root 权限下执行如下命令。

```
dnf install package_name package_name…  -- setopt = strict = 0
```

示例如下。

```
# dnf install httpd
```

安装 rpm 包的过程中，若出现安装失败，请参见本书公众号中关于“安装时出现软件包冲突、文件冲突或缺少软件包导致安装失败”的相关内容。

5. 下载软件包

使用 DNF 下载软件包，请在 root 权限下输入如下命令。

```
dnf download package_name
```

如果需要同时下载未安装的依赖，则加上--resolve，使用命令如下。

```
dnf download -- resolve package_name
```

示例如下。

```
# dnf download -- resolve httpd
```

6. 删除软件包

要卸载软件包以及相关的依赖软件包，请在 root 权限下执行如下命令。

```
dnf remove package_name…
```

示例如下。

```
# dnf remove totem
```

5.6.3　管理软件包组

软件包集合是服务于一个共同目的的一组软件包，如系统工具集等。使用 DNF 可以对软件包组进行安装/删除等操作，使相关操作更高效。

1. 列出软件包组清单

使用 summary 参数，可以列出系统中所有已安装软件包组、可用的组，可用的环境组的数量，命令如下。

```
dnf groups summary
```

使用示例如下。

```
# dnf groups summary
Last metadata expiration check: 0:11:56 ago on Wed 06 Apr 2022 07:45:14 PM CST.
Available Groups: 8
```

要列出所有软件包组和它们的组 ID，命令如下。

```
dnf group list
```

使用示例如下。

```
# dnf group list
Last metadata expiration check: 0:37:26 ago on Wed 06 Apr 2022 09:08:27 PM CST.
Available Environment Groups:
  Server
  Virtualization Host
Installed Environment Groups:
  Minimal Install
Available Groups:
  Container Management
  Development Tools
  Headless Management
  Legacy UNIX Compatibility
  Network Servers
  Scientific Support
  Security Tools
  System Tools
  Smart Card Support
```

2. 显示软件包组信息

要列出包含在一个软件包组中必须安装的包和可选包，使用命令如下。

```
dnf group info glob_expression…
```

例如，显示 Development Tools 信息，示例如下。

```
# dnf group info "Development Tools"
Last metadata expiration check: 0:38:07 ago on Wed 06 Apr 2022 09:08:27 PM CST.
Group: Development Tools
Description: A basic development environment.
Mandatory Packages:
  FusionOS-rpm-config
  autoconf
  automake
  binutils
  bison
  flex
  gcc
  gcc-c++
  gdb
  gettext
  glibc-devel
  libtool
  make
  patch
  pkgconf
  rpm
  rpm-build
Default Packages:
  asciidoc
  byacc
  ctags
  diffstat
  elfutils
  gcc-gfortran
  git
  intltool
  ltrace
  patchutils
  perl-Fedora-VSP
  perl-generators
  pesign
  source-highlight
  subversion
  systemtap
  valgrind
  valgrind-devel
Optional Packages:
  babel
  chrpath
  cmake
  expect
  gcc-objc
  gcc-objc++
  mercurial
  rpmdevtools
  rpmlint
  systemtap-sdt-devel
  systemtap-server
```

3. 安装软件包组

每一个软件包组都有自己的名称以及相应的 ID(groupid),可以使用软件包组名称或它的 ID 进行安装。要安装一个软件包组,请在 root 权限下执行如下命令。

```
dnf group install group_name
dnf group install groupid
```

例如,安装 Development Tools 相应的软件包组,命令如下。

```
# dnf group install "Development Tools"
# dnf group install development
```

4. 删除软件包组

要卸载软件包组,可以使用软件包组名称或它的 ID,在 root 权限下执行如下命令。

```
dnf group remove group_name
dnf group remove groupid
```

例如,删除 Development Tools 相应的软件包组,命令如下。

```
# dnf group remove "Development Tools"
# dnf group remove development
```

5.6.4　检查并更新

dnf 可以检查系统中是否有软件包需要更新。可以通过 dnf 列出需要更新的软件包,并可以选择一次性全部更新或者只对指定包进行更新。

1. 检查更新

如果需要显示当前系统可用的更新,使用命令如下。

```
dnf check - update
```

使用示例如下。

```
# dnf check - update
Last metadata expiration check: 0:02:10 ago on Wed 06 Apr 2022 11:28:07 PM CST.
packagename1        version1              updates
packagename2        version2              updates
packagename3        version3              updates
…
```

2. 升级

如果需要升级单个软件包,在 root 权限下执行如下命令。

```
dnf update package_name
```

例如,升级 rpm 包,示例如下。

```
# dnf update packagename
Last metadata expiration check: 0:02:10 ago on Wed 06 Apr 2022 11:30:27 PM CST.
```

```
Dependencies Resolved
 ================================================================================
Package                    Arch       Version       Repository       Size
 ================================================================================
Updating:
packagename1               aarch64       version1       updates       Size1
packagename2               aarch64       version2       updates       Size2
packagename3               aarch64       version3       updates       Size3
Transaction Summary
 ================================================================================
Upgrade 3 Package

Total download size: Total Size
Is this ok [y/N]:
```

类似地，如果需要升级软件包组，在 root 权限下执行如下命令。

```
dnf group update group_name
```

3. 更新所有的包和它们的依赖

要更新所有的包和它们的依赖，在 root 权限下执行如下命令。

```
dnf update
```

小结

本章介绍了进程管理的知识，包括进程的基本概念、查看进程信息的命令、定时任务调度，以及进程的挂起和恢复操作。然后介绍了在 Linux 操作系统中，管理服务和后台进程的重要概念和工具，以及 systemd 的特性和使用方法。最后介绍了 FusionOS 中的软件包管理和操作系统中与进程相关的程序管理，包括仓库配置、依赖解析、事务处理等；本地软件源仓库的创建、软件源的添加、启用和禁用；软件包的搜索、列出清单、显示信息、安装、下载和删除操作。

习题

1. 请解释进程的概念。它的主要作用是什么？

2. 什么是进程控制块？它包含哪些信息？

3. Linux 中的进程查看命令有哪些？请分别介绍 who、ps 和 top 命令的作用和常用选项。top 命令相对于 ps 命令有什么优势？

4. 在 Linux 中，如何使用 kill 命令终止一个进程？

5. 什么是 Linux 中的计时器？它们的作用是什么？什么是实时时钟？它在系统中的作用是什么？什么是内核定时器？它用于哪些任务？什么是 POSIX 定时器？它的作用是什么？什么是定时器轮？它在哪些情况下使用？

6. 什么是 at 和 crontab 命令？它们分别用于什么样的定时任务？atd 守护进程是什

么？它的作用是什么？

7. 如何使用 at 命令来定时运行一批程序？它的语法是怎样的？

8. crontab 命令用于什么目的？它的常用方法有哪些？如何编辑 crontab 文件以配置周期性运行的任务？它的文件格式是怎样的？

9. 什么是后台服务？什么是 systemd？什么是 systemd unit？

10. systemd 提供了哪些特性？

11. 如何使用 systemctl 命令运行、关闭、重启、显示、启用/禁用系统服务？

12. 如何查看系统中的运行服务和状态？如何改变系统的默认目标？

13. 什么是程序和进程之间的关系？

14. FusionOS 使用什么作为软件包管理工具？它的管理原理是什么？

15. DNF 主要配置文件是什么？它包含哪两部分内容？

16. 如何在 DNF 中配置主要部分的参数？举例说明几个参数的含义。

17. 如何配置 DNF 的软件源仓库？有哪两种方式？

18. 如何显示当前的 DNF 配置信息？如何显示特定软件源的配置？

19. 如何创建本地软件源仓库？请列出步骤。

20. 如何添加、启用和禁用软件源仓库？

21. 如何搜索软件包、列出软件包清单、显示软件包信息、安装软件包、下载软件包和删除软件包？