版本号：v1.0 阶段：初稿

# 操作系统面试宝典

MRL Liu

（未经许可，不得传播）

2022年01月22日

本文以《现代操作系统（原书第3版）》》为参考，在具备基本逻辑的基础上只记录笔者认为对计算机网络领域面试相关的知识，深入研究可参考其他资料。

# 操作系统基础

操作系统的定义有很多种，但总体来说，我们可以认为它扮演着这样一个角色：

|  |
| --- |
| 操作系统是介于硬件资源和应用程序之间的一个系统软件，它可以管理硬件资源，同时也可以管理应用程序。 |

### 根据以上的叙述，我们也可以总结出操作系统的主要功能：

1. 硬件资源管理

计算机必要的硬件资源有CPU、内存、硬盘和I/O设备。这些硬件资源是有限的，所以必须进行合理地资源分配和资源回收。

1. 应用程序管理

操作系统为应用程序提供硬件资源的服务，同时也要管理应用程序，即控制进程的生命周期、资源分配等。

## 一、进程管理

### 1、进程（process）

### （1）程序和进程

**程序**在运行前是存储在硬盘里的静态文件，运行时才被**CPU**加载到内存。**进程本质上就是一个正在运行的程序**，是对运行时程序的封装，其包含了该程序运行的所有资源和操作。

我们通过以下比喻来理解CPU对进程的管理：

|  |
| --- |
| CPU的每个核是一个厨师，程序是保存在存储箱里的食谱，其用来制作菜谱前一直放在存储箱里，程序的各种输入数据就是做菜的各种原料，程序运行后可以实现的各种功能就是最终做好的菜品。  进程就是厨师阅读食谱、获取各种原料并做菜的一系列动作的总和，厨房就是该进程对应的地址空间，厨师烹饪每一道菜品都必须在对应的厨房按照对应的食谱进行，即使烹饪同一道菜，也必须在不同的厨房，所以计算机中的同一个程序运行了两遍，也是两个不同的进程。 |

### （2）CPU的并发

单核的CPU在任意时刻只能运行一个程序（一个厨师在任意时刻只能按照一个食谱做一道菜），但是现在的复杂操作系统基本都是多任务的，即CPU可以同时运行多个程序。我们继续上述的比喻来描述这种CPU同时运行多个程序的机制：

|  |
| --- |
| 假设现在需要一个单核CPU同时运行多个程序：A、B，即厨师要同时按照不同的食谱A、B做不同的菜品A、B。  厨师的做法是，在某段时间内，先去厨房A按照食谱A烹饪菜品A，然后做一小段时间后，记录下此时食谱A的进度（保存进程的当前状态）；然后跑去厨房B按照食谱B烹饪菜品B，同样做一小段时间后，记录下此时食谱B的进度，返回厨房A继续按照食谱A烹饪菜品A。  在上述过程中，一个厨师在任意时刻只能按照一个食谱进行烹饪，每工作一段时间就切换到另一个食谱。这样子的好处是，对每个菜品来讲，每次它被烹饪时都独占了厨师的精力，对顾客来讲，这些菜品看起来是同时在烹饪的。 |

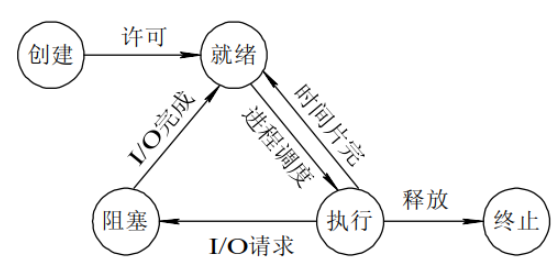
同样的原理，单核CPU在任意时刻只能运行一个程序，所以单核CPU在每个时间片（由程序计时器计数的一小段时间）内运行一个程序，时间片时结束就切换到另一个程序。由于每个时间片足够小，对每个程序来讲，每次运行都似乎独占了CPU的资源；对每个用户来讲，每个程序似乎都可以同时运行。我们将这种机制称为CPU的并发。注意以下CPU的并发和并行概念的区别：

|  |
| --- |
| CPU的**单个核心**在**每个时间片**内分别执⾏多个进程，称为**CPU的并发**；  CPU的**多个核心**在**同一个时间片**内同时执行多个进程，称为**CPU的并行**。 |

我们注意到，对于CPU的并发来讲，CPU从一个进程切换到另一个进程需要保存进程的状态信息。

### （3）进程的状态

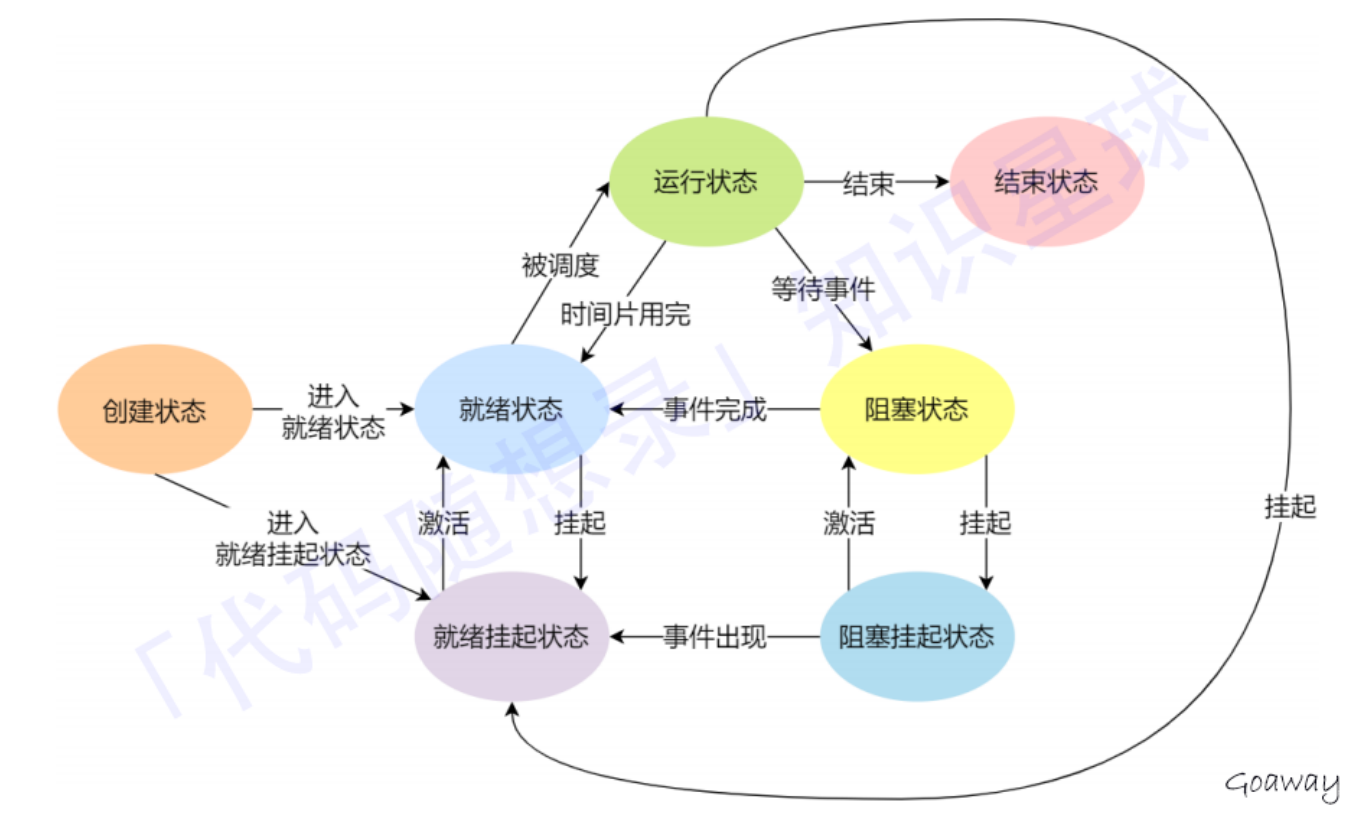
在前文描述中，我们已经知道，同一个进程存在多个状态，具体来讲，进程从创建到终止大致分为如下5种状态，即进程的五态模型：



|  |  |  |
| --- | --- | --- |
| **状态类型** | **状态特点** | **状态切换条件** |
| 创建态 | 创建进程，调度程序为其分配内存空间等资源 | 如果成功获取除处理机以外的其他资源则自动进入就绪态。 |
| 就绪态 | 因为其他进程正在运行而暂时停止。 | 等待分配处理机资源，得到后可立即运行 |
| 执行态  （运行态） | 该进程正在运行，处于独占用一个CPU核心的时间片 | 时间片用尽后自动进入就绪状态 |
| 阻塞态 | 该进程正在等待某一事件发生而暂时停止运行。 | 比如等待客户端连接或用户输入。 |
| 终止态 | 进程结束，资源被回收。 | 可能是任务结束或者遇到已知问题而主动结束也可能是遇到严重未知错误或被其他进程杀死而被动结束 |

为了满足系统和用户观察、分析和及时修改进程的需要，还引入了一个对进程的重要操作——挂起，增加挂起操作后，进程的就绪状态分为活动就绪态（激活）和静止就绪态（挂起），阻塞状态分为活动阻塞态（激活）和静止阻塞态（挂起）。

进程的七态模型：



注意，在所有状态的切换中，只有就绪态和运行态是双向转换的，其他都是单向转换。

### （4）进程的实现

为了实现上述描述的进程模型，操作系统维护着一张进程表（本质是一个结构数组或结构链表），每一个进程占用其中的一个进程表项，即进程控制块（PCB），每个PCB包含了一个进程的重要状态信息，例如程序计时器、堆栈指针、内存分配状况、所打开文件的状态、账号的调度信息等。

### （6）守护进程

启动操作系统时，通常会创建若干个进程，其中有些需要同用户交互并替用户完成某些工作，这时这些进程是前台进程，如果进程不需要与用户进行交互但是还有执行一些必要的功能，那么这些进程就是后台进程。

守护进程（daemon，也叫精灵进程）是一种特殊的后台进程，其通常完全独立于控制终端，并且周期性地执行某种任务或等待处理某些事件，例如收发电子邮件的网络进程，在大部分事件下都处于休眠状态，但是当接收到电子邮件时就被唤醒。操作系统中有很多守护进程。

### 线程（）

在传统的操作系统中，每个进程都有一个对应的地址空间（address space）和一个控制线程。线程就是轻量级进程

多线程的独特优势？（为什么需要）有了进程为什么还需要多线程？/为什么需要多进程？

|  |
| --- |
| 很多应用程序的功能实现需要同时有多种活动同时进行，而且有些活动可能需要阻塞，比如服务器等待客户端连接，服务器在等待客户端连接的同时，可能需要同时处理用户的输入，显然这种需求单进程/单线程是完成不了的。  多线程是同一个地址空间内的共享同一个地址空间和所有可用数据的并行实体，线程比进程更加轻量级，更容易创建和撤销，在许多系统中创建线程比创建进程要快10~100倍。 |

多进程具有不同的地址空间，

进程可以对其进行读写，从逻辑上看，进程是一个可以容纳运行一个程序所需要所有信息的容器。

进程可以被挂起和重启

一个进程可以创建多个子进程，所以进程不断创建下去就会得到一个进程树。

### 3、进程间通信（process）

## **存储系统**

## **文件系统**

## **输入输出**

## **互斥同步**

# **Linux操作系统**

父子进程

孤儿进程和僵尸进程

# **Windows操作系统**

# 迭代日志表

|  |  |  |
| --- | --- | --- |
| **迭代版本** | **迭代工作** | **迭代日期** |
| V1.0 | 建立初稿文档，完成第1章和第2章的整理。 | 2022-01.15-01.20 |
| V1.1 |  |  |