操作系统

By: 诚夏 SincereXIA @ XD Univ.

第一章 操作系统概述

什么是操作系统

操作系统是控制和管理计算机和软件资源、合理地组织计算机工作流程，以及方便用户的程序与相关文档的集合。

形成标志：多道程序设计技术（并发）的出现

操作系统的特征

1. **并发**：两个或多个活动在同一给定的时间间隔中进行

   > 并发是指宏观上在一段时间内能同时运行多个程序，而并行则指同一时刻能运行多个指令。
   >
   > 并行需要硬件支持，如多流水线或者多处理器。
   >
   > 操作系统通过引入进程和线程，使得程序能够并发运行。

2. **共享**：计算机系统中的资源被多个任务所共用

   > 有两种共享方式：互斥共享和同时共享。
   >
   > 互斥共享的资源称为临界资源，例如打印机等，在同一时间只允许一个进程访问，需要用同步机制来实现对临界资源的访问。

3. **虚拟**：物理上的实体变为逻辑上的对应

   > 主要有两种虚拟技术：时分复用技术和空分复用技术。例如多个进程能在同一个处理器上并发执行使用了时分复用技术，让每个进程轮流占有处理器，每次只执行一小个时间片并快速切换。

4. **不确定性**：（异步性）每个程序什么时候执行，向前推进速度快慢，是由执行的现场所决定。但同一程序在相同的初始数据下，无论何时运行都应获得同样的结果。

操作系统的基本功能

1. 硬软资源管理

1. 进程管理

   进程控制、进程同步、进程通信、死锁处理、处理机调度等。

2. 内存管理

   内存分配、地址映射、内存保护与共享、内存扩充等。

3. 文件管理

   文件存储空间的管理、目录管理、文件读写管理和保护等。

4. 设备管理

   完成用户的 I/O 请求，方便用户使用各种设备，并提高设备的利用率。主要包括缓冲管理、设备分配、设备处理、虚拟设备等。

2. 用户接口

- **程序级接口：** 一组系统调用命令，在汇编语言程序中可以直接使用系统调用

- **作业控制级接口：** 命令方式、图形界面

操作系统的发展历史

1. **早期的人工操作方式**
2. **单道批处理系统**
    1. 一次加载若干个作业组装的磁盘,使用同一组相同的系统带
    2. 脱机输入/输出, 使用卫星机 (脱机批处理)
    3. 主要问题: cpu和i/o设备忙闲不均
3. **缓冲, 中断, 通道技术**
    1. 中断: cpu 收到外部中断信号之后, 停止原来工作, 处理中断事件, 之后回到断点继续工作
    2. 通道: 控制 i/o 设备与内存间的数据传输, 可独立于 cpu 运行, 实现计算与输入输出的并行
    3. **SPOOLING技术**: 并行的外围设备联机操作, **以磁盘为缓冲区解决低速的 i/o 设备与 cpu 之间的速度匹配问题**

       SPOOLing 系统的组成主要有三大部分：
        1. **输入井和输出井**：磁盘上开辟的两个大存储空间。
        2. **输入缓冲区和输出缓冲区**：在内存中开辟两个缓冲区，输入缓冲区暂存由输入设备送来 的数据，后送输入井；输出缓冲区暂存从输出井送来的数据，后送输出设备。
        3. **输入进程和输出进程：**利用两个进程模拟脱机 I/O 时的外围处理机。
4. **多道批处理系统**

   在内存中同时存放若干作业, 共享系统资源, 并同时运行, 为了解决系统资源的管理问题, 必须设计操作系统

   特点:
    1. 多道: 内存中同时存放多个作业
    2. 宏观并行: 所有程序同时运行
    3. 微观串行: 交替使用 cpu
    4. 调度性: 作业调度选择作业装入内存
    5. 无序性: 调度次序与时间无关

   优点:
    1. 提高 cpu 利用率
    2. 增加系统吞吐量

       不同作业在 资源忙碌/作业完成才切换, 系统开销小

   缺点:
    1. 平均周转时间长, 作业需要排队
    2. 无交互能力

现代操作系统分类

1. **分时系统:**

   把计算机的系统资源进行时间上的分割, 依次轮流使用时间片

   特点: 同时性、独立性、及时性、交互性（通用系统）
    1. 人机交互性好

2. 共享主机, 多用户
3. 用户独立性, 互不干扰

关键问题:

1. 及时接受输入
2. 及时响应
3. 减少兑换信息量

2. **实时系统:**

   实时性, 可靠性, 在限定的时间内, 对输入进行快速处理, 并做出响应

   特征: 响应时间短、系统高可靠性和安全性（专用系统）

   1. 实时时钟管理： 提供系统时间, 定时延时功能
   2. 过载保护：缓冲区排队，丢弃某些任务，动态调整任务周期；
   3. 高度可靠安全, 冗余备份

3. **嵌入式系统**

   面向特定应用, 先进技术, 高效率设计, 与应用结合, 软件固化, 不具备自举开发能力

4. **多处理机操作系统**

5. **网络操作系统**

6. **分布式操作系统**

	耦合程度	并行性	透明性	健壮性
分布式	紧密耦合, OS 同质	一个进程可分散在机器上并行执行	资源调度透明, 用户无法控制	要求更强的容错能力
网络式	允许异种 os 互联, 协议同质	进程独立	不透明, 可以明确指定	

7. **个人计算机操作系统（微机操作系统）**

针对个人使用优化的操作系统

操作系统的结构

1. 整体式系统：无结构操作系统
2. **模块化结构:**

   整个系统分成若干功能各异的模块，**模块预先定义有接口**，各**模块之间只能通过这些接口进行通信。**模块可以自由调用。典型的操作系统： UNIX(模块化的操作系统)。

3. **分层式结构**

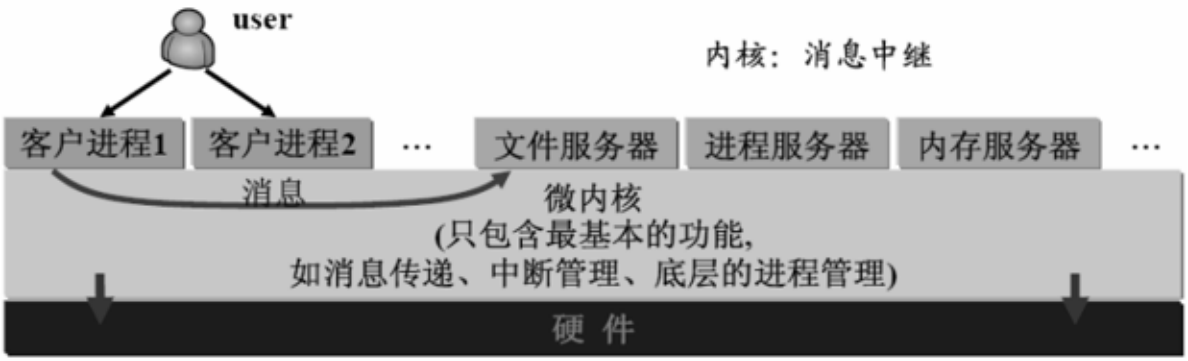
将系统按照层次结构**划分为若干“层”**。某一层次上代码**只能调用低层次上的代码**，使模块间的**调用有序化**。系统每加一层，就构成一个比原来功能更强的虚拟机。第一个按照这种思想构造的系统是THE系统(68年，Dijkstra等设计，简单的批处理系统)。

4. **内核体系结构**
    1. **微内核结构**

        内核仅留下基本功能，其他分离出去，用用户态下的进程实现。 **良好的扩充性, 模块相互隔离**, 调用关系明确，通过消息通信

        优点：内核可靠性高，可移植性好，扩展性好

        缺点：运行效率不高

    2. **单内核结构**

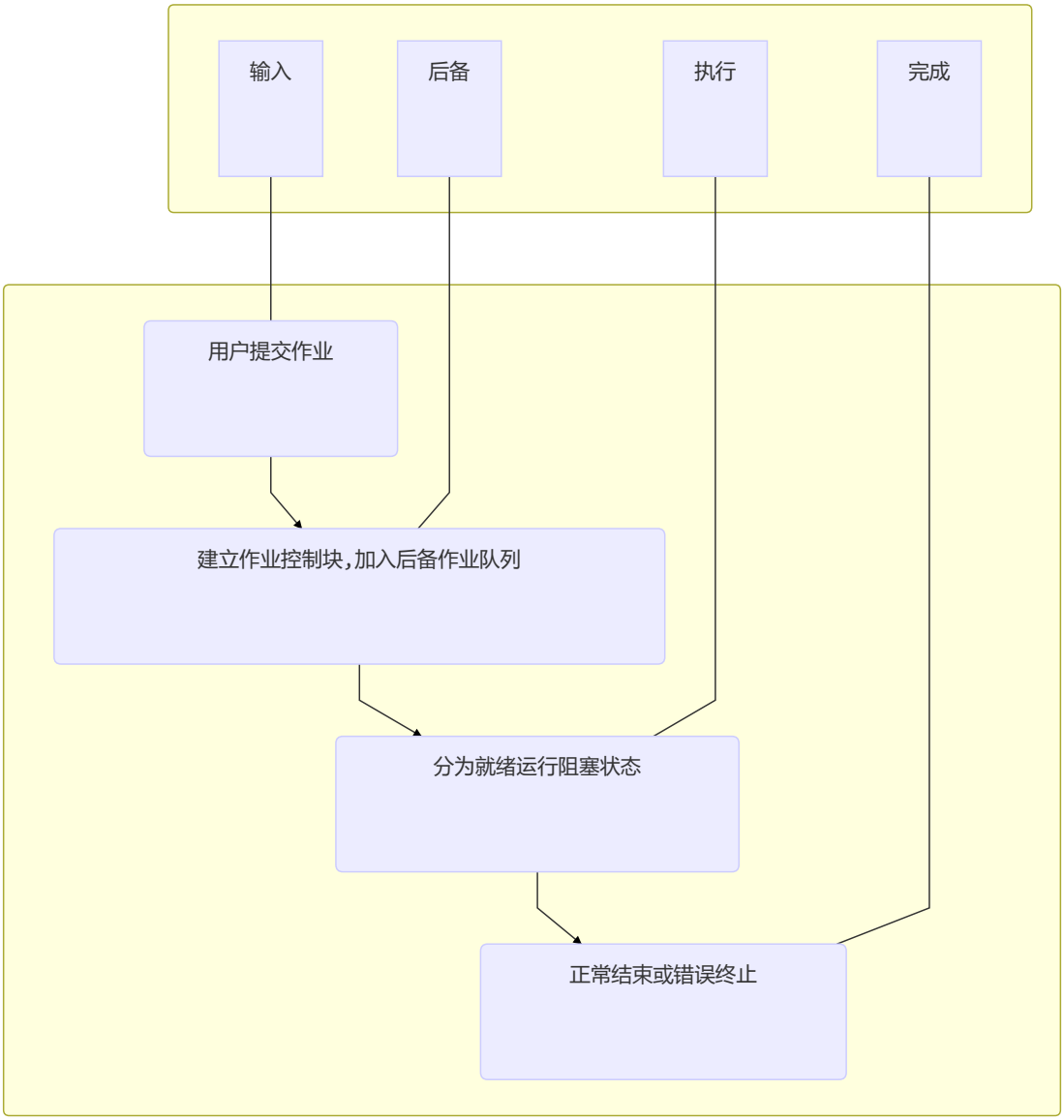
        单内核是将操作系统功能作为一个紧密结合的整体放到内核。通过函数通信

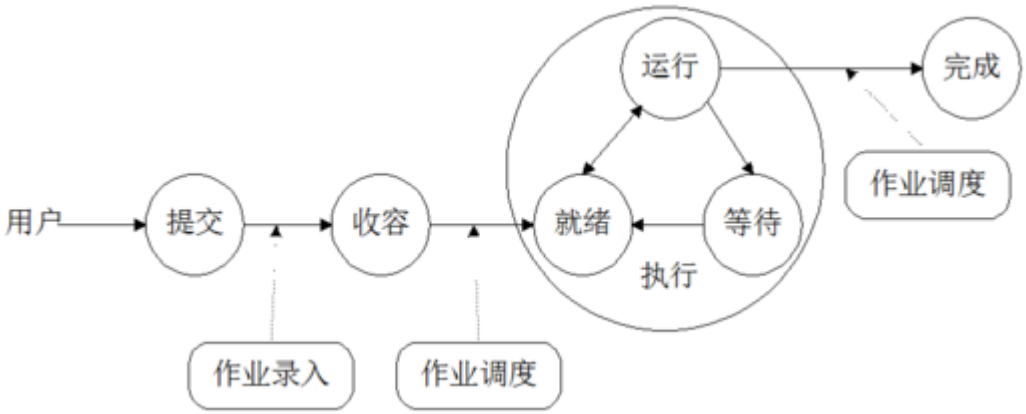
        由于各模块共享信息，因此有很高的性能。

第二章 用户接口与作业管理

作业

1. 基本概念 作业是**用户一次请求计算机系统为他完成任务所进行的工作总和**
2. 作业类型
    1. 脱机作业
       不直接交互，通常用于批处理系统
    2. 联机作业
       直接交互，用于分时系统和微机系统
3. 作业的组成 **包括： 程序、数据、作业控制信息（作业说明书） 三部分**
4. 作业的处理过程

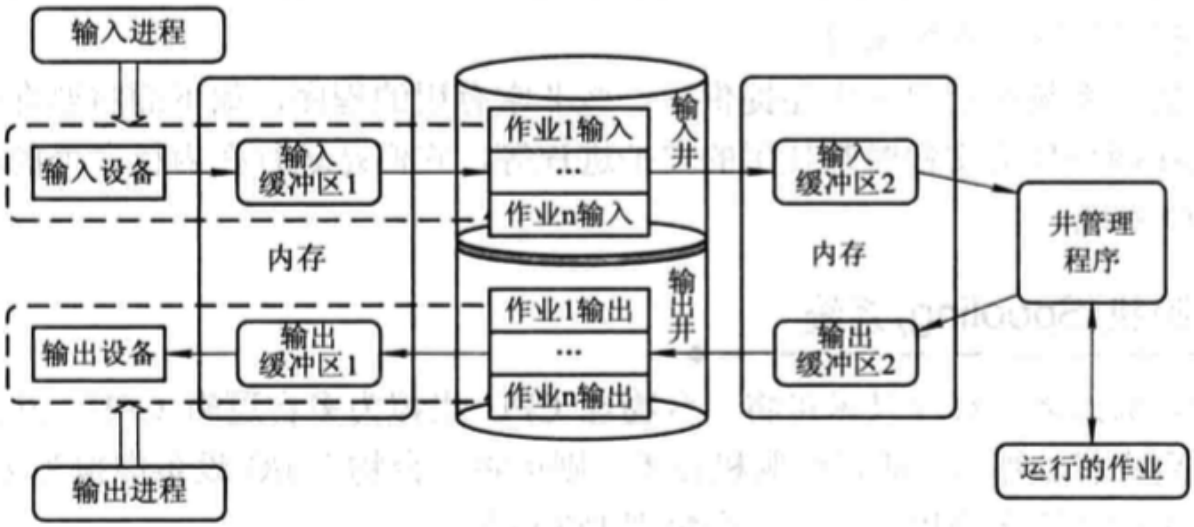

输入
后备
执行
完成
用户提交作业
建立作业控制块,加入后备作业队列
分为就绪运行阻塞状态
正常结束或错误终止


运行
完成
作业调度
用户
提交
收容
就绪
等待
执行
作业录入
作业调度

作业的输入/输出方式

1. 联机输入、输出 主机控制I/O
2. 脱机输入/输出 外围处理机控制， 人工干预
3. 脱机输入/输出
4. SPOOLing 系统 用磁盘来模拟独占设备的操作，**一台独占设备变成多台并行的虚拟设备（独占变共享）**，兼具脱机和联机方式的优点，可以实现联机方式下的主机和外围设备的同时工作，又称为假脱机

   SPOOLing 由专门负责 I/O 的常驻内存进程和**输入输出井**组成 **作用：** 独占变共享，实现了虚拟设备的功能



(a) SPOOLing的工作原理

作业控制块

作业控制块 JCB 是作业存在的**唯一标志**，当作业进入系统后，系统会为其创建作业控制块，用来存放管理和控制作业所必须的信息，只有作业退出系统后，JCB 才被撤销

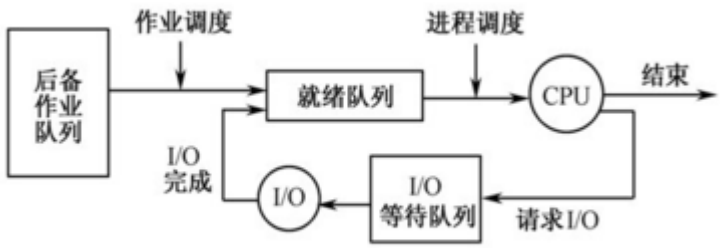
> JCB 包含该作业的标识信息、状态信息、调度参数、资源需求和其他控制信息

作业后备队列就是按照某种原则将后备作业的 JCB 排成的一个或多个序列，以便作业调度。

作业调度

1. 多级调度结构

> 计算机内的调度结构：



两级调度简化队列图

1. **高级调度：** **作业调度**，使后备作业进入主机

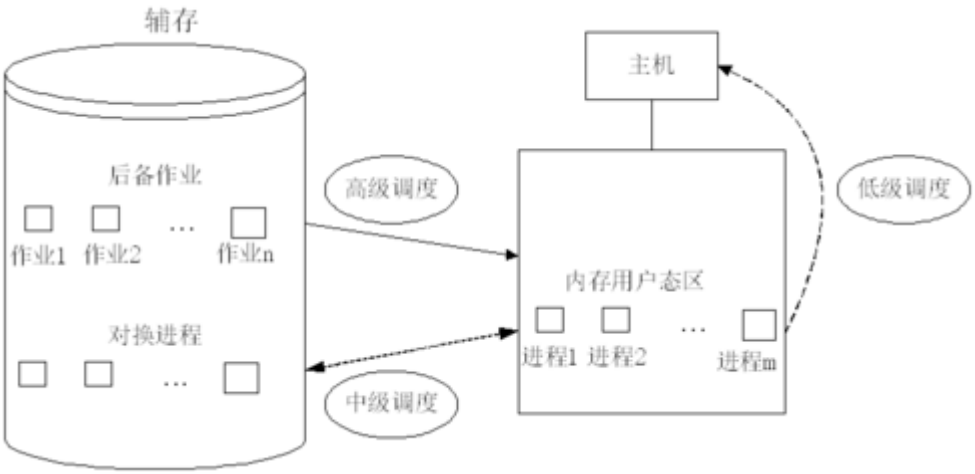
   作业就是用户程序及其所需的数据和命令的集合，作业管理就是对作业的执行情况进行系统管理的程序的集合。作业调度程序的主要功能是审查系统是否能满足用户作业的资源要求以及按照一定的算法来选取作业。

2. **中级调度：** 对换调度，主机中的作业在 ram（内存） 和 swap 区间（硬盘）的调换

   提高内存的利用率和系统吞吐量，使得暂时不运行的进程从内存对换到外存上。

3. **低级调度：** **进程调度**：决定 ram 中的哪个进程可以占用 cpu， 开始运行

   根据一定的算法将cpu分派给就绪队列中的一个进程。进程调度是操作系统中最基本的一种调度，其调度策略的优劣直接影响整个系统的性能。



2. 作业调度算法的评价指标

1. **CPU 利用率**

   CPU 忙碌时间占总时间的百分比

2. 吞吐量

   单位时间内 CPU 完成作业的数量

3. **周转时间 和 周转系数**

   若作业 J_i 的提交时间为 t_{si} , 执行时间为 t_{ri} 完成时间为 t_oi 作业 J_i 的周转时间：*从提交到完成的总时间*

   $$T_i = t_{oi} - t_{si} \qquad i = 1, 2, \ldots n$$

   周转系数：周转时间/执行时间 *总时间 / 有用的时间*

$$W_i = T_i / t_{ri}$$

n 个 作业的平均周转时间，平均周转系数

$$T = \frac{1}{n}\sum_{i=1}^{n} T_i$$

$$W = \frac{1}{n}\sum_{i=1}^{n} W_i$$

作业的平均周转时间越短，系统的吞吐量就越高

> 要想提高吞吐量，就应优先考虑运行短作业；若要提高 CPU 利用率，则应优先考虑长作业

常见的作业调度算法

1. 单道批处理系统作业调度算法
    1. **先来先服务(FCFS)**

        > 按作业到达先后进行调度，即启动等待时间最长的作业。这种算法忽视了吞吐量和平均周转时间

        有利于长作业，不利于短作业，不利于IO 繁忙的作业
    2. **短作业优先调度（SJF）**

        > 以要求运行时间长短进行调度，即启动要求运行时间最短的作业。

        这种算法可以有效降低作业的平均等待时间，提高系统的吞吐量， 对长作业不利，出现饥饿现象，未考虑作业的紧迫程度。
    3. **最高响应比优先算法 (HRP)**

        > 优先调度响应比高的作业
        >
        > 响应比RP = 作业响应时间/作业估计运行时间
        >
        > = **（作业估计运行时间+作业等待时间）/作业估计运行时间**
        >
        > = **1 + 作业已等待时间/作业估计运行时间**

        响应比 RP = 作业响应时间/作业估计运行时间 = （作业估计运行时间+作业等待时间）/ 作业估计运行时间=1+作业等待时间/作业估计运行时间
2. 多道批处理系统作业调度算法
    1. **优先级调度算法** 用户指定优先级进行调度，优先级高的作业先启动
    2. 均衡调度算法

        > 这种算法的基本思想是根据系统的运行情况和作业本身的特性对作业进行分类。作业调度程序轮流地从这些不同类别的作业中挑选作业执行。这种算法力求均衡地使用系统的各种资源。

        根据作业特性进行分类挑选

总结：就平均周转时间和平均周转系数来说，最短作业优先算法最小 ，先来先服务算法最大， 响应比高优先算法居中

> 例：性能分析

设有四个作业，其提交时刻、执行时间如下表所示：

作业号	提交时刻	运行时间
1	8.00	2.00
2	8.50	0.50
3	9.00	0.10
4	9.50	0.20

1. 先来先服务调度算法：顺序为1 2 3 4，计算平均周转时间T和平均周转系数W，如下表所示。

作业	提交时间	执行时间	开始时间	完成时间	周转时间	周转系数
1	8.00	2.00	8.00	10.00	2.00	1.00
2	8.50	0.50	10.00	10.50	2.00	4.00
3	9.00	0.10	10.50	10.60	1.60	16.00
4	9.50	0.20	10.60	10.80	1.30	6.50
平均周转时间T＝1.725小时，平均周转系数W＝6.875					6.90	27.50

2. 最短作业优先调度算法：由于在8.00开始执行作业，当时仅有1，而作业2，3，4尚未到达，故作业1是最短作业。作业1执行完成后是10.00，此时作业2，3，4均已经到达，故选最短作业3，然后是4，2。所以顺序为1,3,4,2。平均周转时间和平均周转系数的计算结果如下表所示。

作业	提交时间	执行时间	开始时间	完成时间	周转时间	周转系数
1	8.00	2.00	8.00	10.00	2.00	1.00
2	8.50	0.50	10.30	10.80	2.30	4.60
3	9.00	0.10	10.00	10.10	1.10	11.00
4	9.50	0.20	10.10	10.30	0.80	4.00
平均周转时间T＝1.55小时，平均周转系数W＝5.15					6.20	20.60

3. 响应比高者优先算法：在作业1执行完成，计算作业2，3，4的响应比分别为：4，11，3.5，因此作业1执行完成后选中作业3完成。执行顺序为1，3，2，4。按此算法求得的平均周转时间和平均周转系数如下表所示。

作业	提交时间	执行时间	开始时间	完成时间	周转时间	周转系数
1	8.00	2.00	8.00	10.00	2.00	1.00
2	8.50	0.50	10.10	10.60	2.10	4.20
3	9.00	0.10	10.00	10.10	1.10	11.00
4	9.50	0.20	10.60	10.80	1.30	6.50
平均周转时间T＝1.625小时，平均周转系数W＝5.675					6.50	22.70

系统功能调用

1. **管态（内核态） 和 目态（用户态）**

> 现代CPU一般都有几种不同的指令执行级别；在高执行级别下，代码可以执行**特权指令，访问任意的物理地址**，这种CPU执行级别就对应着内核态；在相应的低级别执行状态下，代码的掌控范围 会受到限制。只能在对应级别允许的范围内活动

管态（系统态）：操作系统程序运行的状态

算态（目态）：用户程序运行的状态

特权指令：只能在管态下执行而不能在算态下执行的特殊指令

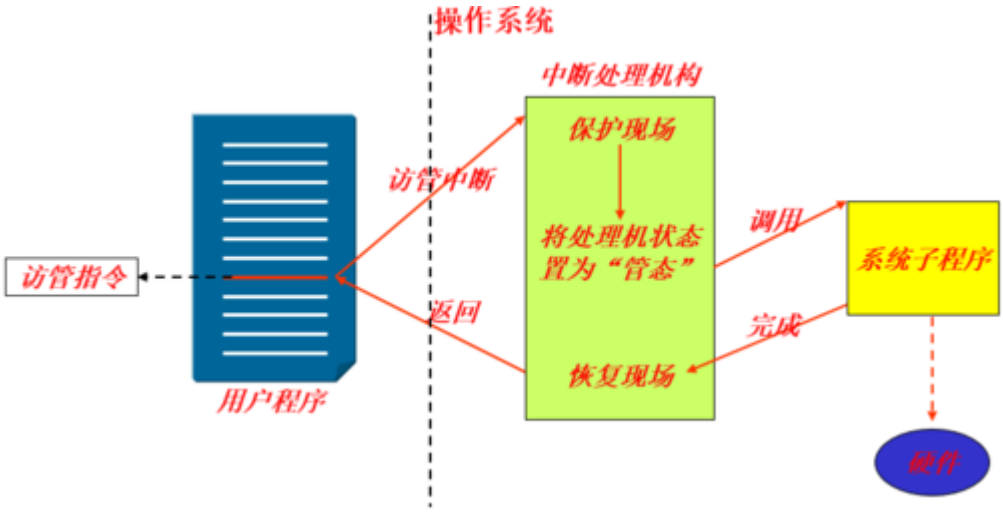
> 常见的特权指令有如下几类：
>
> 1. 传送程序状态字的指令
>
> > 程序状态字通常包括以下状态代码： CPU的工作状态码——指明管态还是目态，用来说明当前在CPU上执行的是操作系统还是一般用户，从而决定其是否可以使用特权指令或拥有其它的特殊权力 条件码——反映指令执行后的结果特征 中断屏蔽码——指出是否允许中断
>
> 2. 启动、测试和控制外设的指令
> 3. 存取特殊寄存器的指令

2. 系统功能调用

用户在程序中用 *访管指令* 调用由操作系统提供的子功能集合。

> 访管指令：本身不是特权指令，基本功能是“自愿进管”，能引起访管中断。

系统功能的调用方法：



系统功能调用与普通过程调用的区别：

1. 运行在不同的系统状态 （一次指令运行，跨越目态和管态）
2. 进入方式不同：需要先通过软中断机制，进入操作系统核心，经过核心处理，转向相应的命令处理程序
3. 执行完成后，将进行重新调度，让优先级高的进程先执行

第三章 进程管理

引言：进程的引入

现在的操作系统多为并发执行，为了提高资源的利用率

1. 顺序执行

> 顺序执行是单道批处理系统的执行方式
>
> 程序在执行时，各程序段必须按照先后次序逐个执行。程序各程序段的这种先后次序可用前趋图表示。

特点： 顺序、封闭（独占、资源状态无法改变）、可再现

2. 多道程序设计

> 把一个以上的程序放入内存中，并且同时处于运行状态，这些程序共享CPU和其它资源。

特点：

1. **多道**：内存中由多道程序，处于就绪、运行、阻塞三种状态
2. **宏观上并行**
3. **微观上串行**
4. 优点：**CPU、设备利用率高，系统吞吐量大**
5. 缺点：多道程序设计复杂，内存需求大，外围设备调度管理复杂

3. 并发执行

> 一段时间内有多道程序在同时运行称为并发性

特征：

1. 失去封闭性：共享资源，程序之间互相制约
2. 间断性：程序执行时间不连贯
3. 不可再现：程序执行的结果随速度，环境的不同而不同

并行是并发的特例，并发是并行的拓展

引入**进程**，反映程序执行的独立性、并发性和动态性

进程

1. 进程的定义

「进程是程序的一次执行，该程序可以和其他程序并发执行；它是一个动态实体，在传统的操作系统设计中，进程既是基本的分配单元，也是基本的执行单元」

2. 进程和程序的区别

进程最基本的属性是「动态性」 和 「并发性」

1. 程序是永存的，进程是暂时的
2. 程序是静态的观念，进程是动态的观念
3. 进程由三部分组成：程序+数据+进程控制块（描述进程活动情况的数据结构）
4. 进程和程序不是一一对应的
    - 一个程序可对应多个进程即多个进程可执行同一程序
    - 一个进程可以执行一个或几个程序

3. 进程的组成

由程序段、数据段和进程控制块(PCB)组成。

1. **程序和数据** 是进程的实体
2. **进程控制块** 是进程存在的唯一标志

> 进程控制块 (Process Control Block, PCB) 描述进程的基本信息和运行状态，所谓的创建进程和撤销进程，都是指对 PCB 的操作。

操作系统为进程 **创建进程控制块** 和 **分配地址空间** 的过程就是进程创建的过程

4. 进程控制块

1. 进程控制块的构成：

   操作系统记录进程状态和信息的基本数据结构， 包括

   1. **标识信息**
      唯一标示进程，有**进程标识**（创建进程时分配的唯一代码），**用户标识**（指明进程的所有者）和 **父进程标识**（指明创建该进程的进程标识）
   2. **现场信息**
      与 CPU 有关的现场信息，寄存器状态、堆栈指针，用户中断之后的恢复
   3. **控制信息**
      操作系统对其进行调度室用到的信息。有进程状态，调度信息、队列指针、资源占有使用信息
2. 进程控制块的组织方式
   1. 以队列形式存在
   2. **以表的形式存在——PCB表**，有 **链接方式** （队列指针直接保存 PCB 的地址） 和 **索引方式** （除执行指针外，其他状态进程的 PCB 信息，由对应的索引表保存）

3. 进程控制块的作用

   进程控制块是进程组成中最关键的部分。

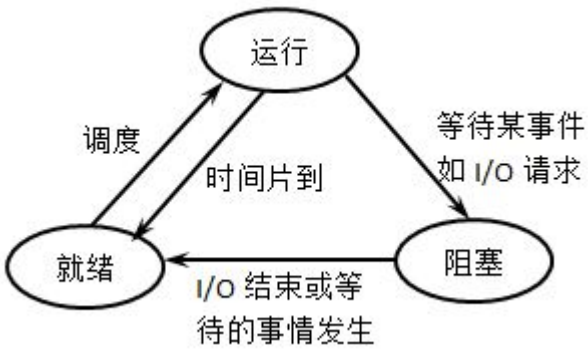
   - 每个进程有唯一的PCB。
   - 操作系统根据PCB对进程实施控制和管理。
   - 进程的动态、并发等特征是利用PCB表现出来的。
   - PCB是进程存在的唯一标志。

5. 进程的基本状态

1. 三状态模型：

   进程的3种基本状态是：运行态、就绪态和阻塞态。

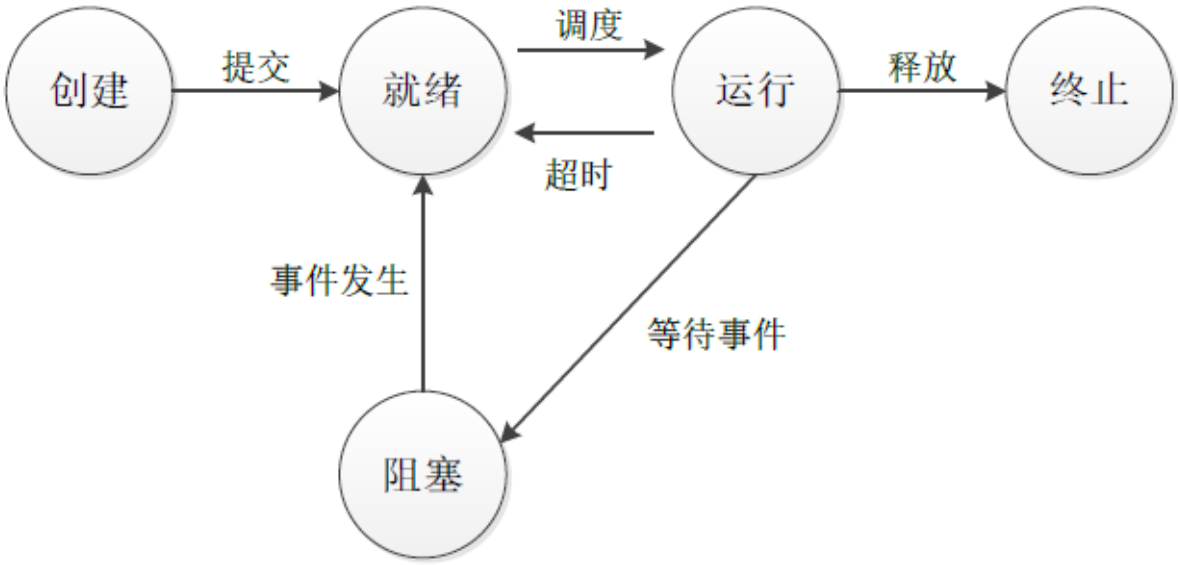
   - **运行态（Running）**
     占有CPU，并在CPU上运行
   - **就绪态（Ready）**
     已经具备运行条件，但由于没有空闲CPU，而暂时不能运行
   - **等待态（Waiting/Blocked）** （其他说法：阻塞态、封锁态、睡眠态）
     因等待某一事件而暂时不能运行



   应该注意以下内容：

   - 只有就绪态和运行态可以相互转换，其它的都是单向转换。就绪状态的进程通过调度算法从而获得 CPU 时间，转为运行状态；而运行状态的进程，在分配给它的 CPU 时间片用完之后就会转为就绪状态，等待下一次调度。
   - 阻塞状态是缺少需要的资源从而由运行状态转换而来，但是该资源不包括 CPU 时间，缺少 CPU 时间会从运行态转换为就绪态。

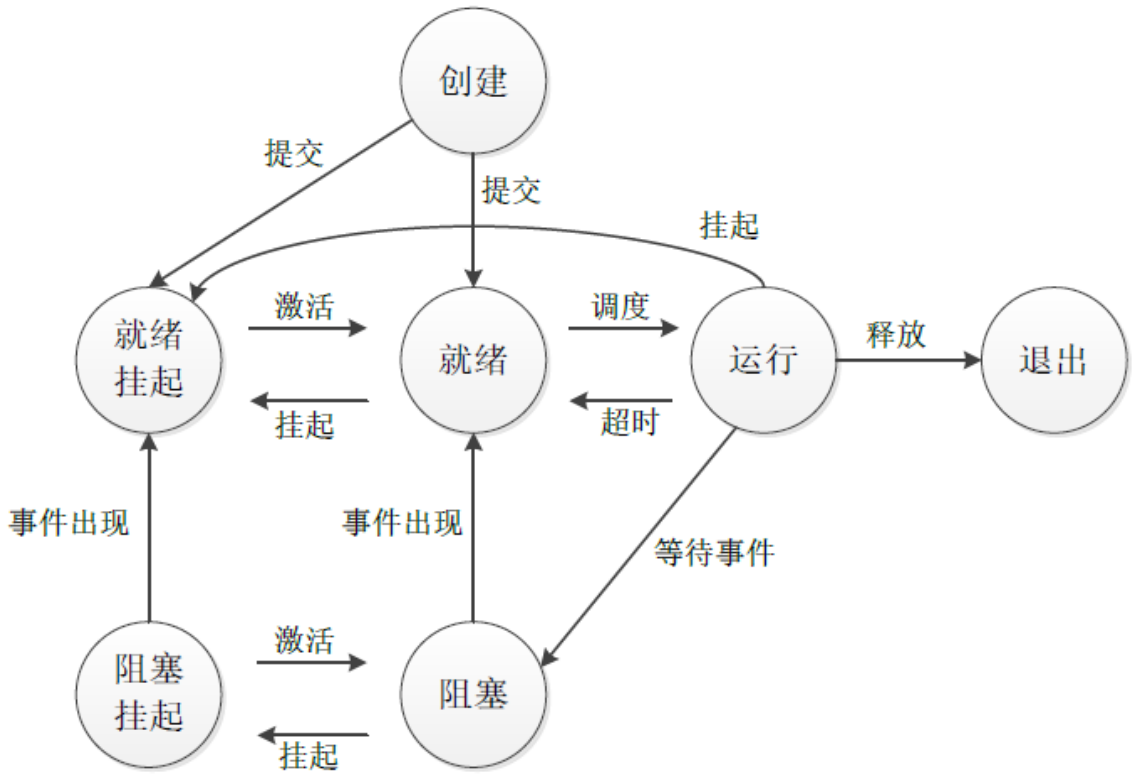
2. 五状态模型



新增 「新建态」和「终止态」：

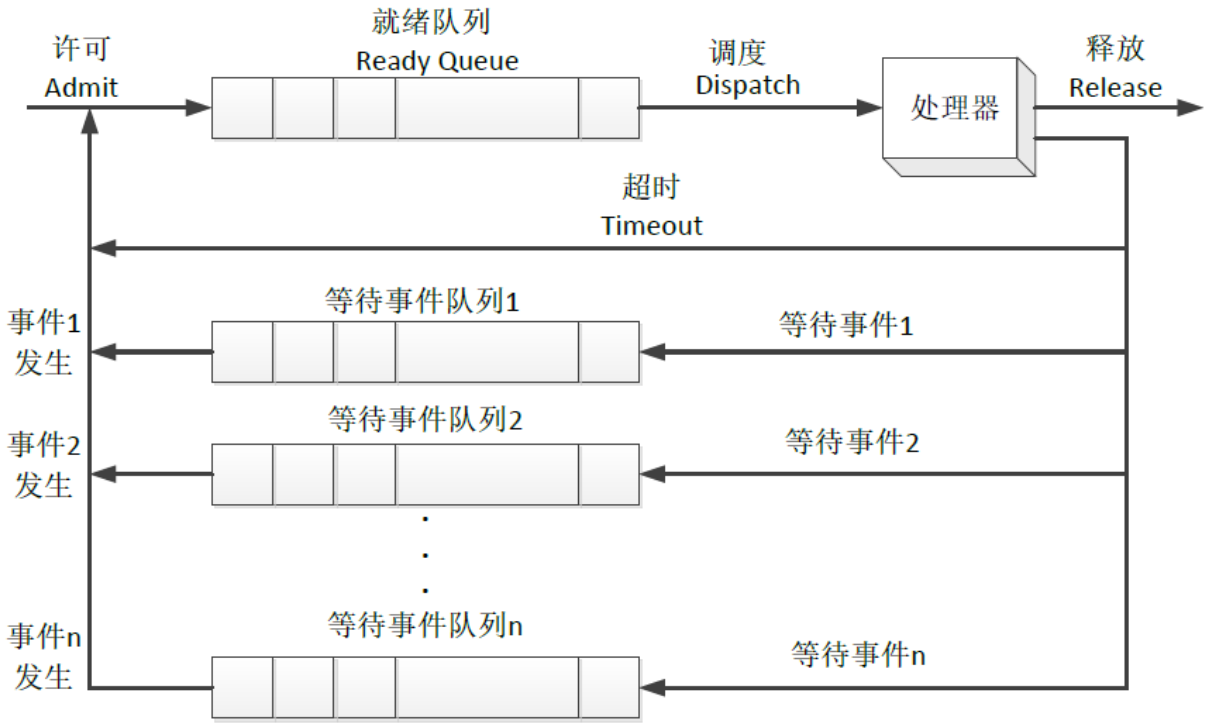
1. 新建态：
   已经完成创建进程的工作，但因为资源不足，进程未执行
2. 终止态：
   终止执行后，进行资源回收

3. 七状态模型



4. 进程队列

队列是常用的组织各进程 PCB 的一种方式，可以按照不同的状态和等待原因的不同，进一步细化为多个队列



6. 进程控制

通过原语(Primitive)实现

> 原语是指由机器指令构成的可完成特定功能的程序段。它是一个机器指令的集合，在执行时不能被中断。多采用屏蔽中断方法实现。

原语有：创建、撤销、阻塞、唤醒、挂起、激活 原语

7. 进程关系的树形结构

优点：

1. **资源分配严格**：
   只能分配到父进程资源
2. **进程控制灵活**
   可以给进程不同的控制权限
3. 进程结构清楚，关系明确

进程调度

> 进程调度是低级调度，从就绪队列中选择某个进程占用 cpu

1. 批处理系统中的调度

1.1 先来先服务

> first-come first-serverd（FCFS）

调度最先进入就绪队列的作业。

有利于长作业，但不利于短作业，因为短作业必须一直等待前面的长作业执行完毕才能执行，而长作业又需要执行很长时间，造成了短作业等待时间过长。

1.2 短作业优先

> shortest job first（SJF）

调度估计运行时间最短的作业。

长作业有可能会饿死，处于一直等待短作业执行完毕的状态。因为如果一直有短作业到来，那么长作业永远得不到调度。

1.3 最短剩余时间优先

> shortest remaining time next（SRTN）

2. 交互式系统中的调度

2.1 优先级调度

2.2 时间片轮转

2.3 多级反馈队列

进程间的相互作用

同步： 多个进程按一定顺序执行；

> **临界资源**： 一段时间内只允许一个进程访问的资源

互斥： 多个进程在同一时刻只有一个进程能进入临界区。

临界区（互斥区）： 进程中涉及到临界资源的程序段

信号量以及 PV 操作

信号量（Semaphore） 是表示资源的实体，是一个与队列有关的整形变量，其值只能由 P、V 操作改变

公用信号量：用于实现进程之间的互斥，初始值为 1 ，它联系的一组并行进程均可对它实施 P、V 操作

私用信号量： 实现进程之间的同步，初始值为 0 或 n

> **PV 操作是原语操作**

信号量的数据结构：

```
struct semaphore{
    int value;
    pointer_PCB queue;
}

P(s){
    s.value = s.value-1;
    if (s.value<0){
        该进程状态置为等待;
```

```
        该进程的 PCB 插入相应的等待队列末尾 s.queue;
    }
}

V(s){
    s.value = s.value + 1;
    if(s.value<0){
        唤醒相应等待队列中等待的一个进程;
        改变其状态为就绪态;
        将其插入就绪队列;
    }
}
```

信号量值的含义：

1. `s.value >= 0` 时， 其值表示还有可用的资源数
2. `s.value < 0` 时，其绝对值表示有多少个进程处于阻塞态

实例：

1. **生产者——消费者问题**

> **问题描述：**使用一个缓冲区来保存物品，只有缓冲区没有满，生产者才可以放入物品；只有缓冲区不为空，消费者才可以拿走物品。

因为缓冲区属于临界资源，因此需要使用一个互斥量 $mutex$ 来控制对缓冲区的互斥访问。

生产者进程：

```
while(true){
    生产一件产品;
    P(S缓);                 /*申请一个空缓冲区*/
    P(mutex);               /*申请缓冲区的使用权*/
    放入一件产品;
    V(mutex);               /*释放缓冲区的使用权*/
    V(S产);                 /*释放产品增加满缓冲区*/
}
```

消费者进程：

```
while(true){
    P(S产);                 /*申请一个满缓冲区*/
    P(mutex);               /*申请缓冲区的使用权*/
    拿出一件产品;
    V(mutex);               /*释放缓冲区的使用权*/
    V(S缓);                 /*增加空缓冲区*/
    消费产品;
}
```

注意，不能先对缓冲区进行加锁，再测试信号量。也就是说，不能先执行 P(mutex) 再执行 P(S缓)。如果这么做了，那么可能会出现这种情况：生产者对缓冲区加锁后，执行 P(S缓) 操作，发现 S缓 = 0，此时生产者睡眠。消费者不能进入临界区，因为生产者对缓冲区加锁了，也就无法执行 V(S缓) 操作，S缓 永远都为 0，那么生产者和消费者就会一直等待下去，造成死锁。

2. **读者——写者模型**

> 允许多个进程同时对数据进行读操作，但是不允许读和写以及写和写操作同时发生。

一个整型变量 count 记录在对数据进行读操作的进程数量，一个互斥量 count_mutex 用于对 count 加锁，一个互斥量 data_mutex 用于对读写的数据加锁。

读者进程：

```
P(mutex);
readnum=readnum+1;
if (readnum==1) P(write);
V(mutex);
read file;
P(mutex);
readnum=readnum - 1;
if (readnum==0) V(write);
V(mutex);
```

写者进程：

```
P(write);
write file;
V(write);
```

使用注意：

1. **互斥操作时，P、V在同一进程中，同步操作时，处于不同的进程**
2. **对互斥信号量的 P 操作在后**

第四章 死锁

死锁的原因

1. 进程推进顺序不当
2. 对互斥资源的分配不当

> **系统资源不足并不是产生死锁的原因**，进程资源如果不足则进程就 不会被创建，只有在资源部分分配以后，剩余的资源不能满足某些个进程的请求，造成进程集无法推进的现象才是死锁。

产生死锁的四个必要条件

1. **互斥条件：** 任一时刻只允许一个进程使用资源。
2. **非剥夺控制：**只能由占用资源的进程自己释放资源

3. **零散请求（保持和等待）**：进程占有部分资源，申请更多的资源
4. **循环等待：**请求资源的进程形成了循环

死锁的处理

忽略死锁（鸵鸟算法）

死锁的检测与恢复

- 资源分配图
  - 静态资源分配

    删除所有未阻塞进程，释放其占有资源
  - 临时资源分配图

    删除未阻塞进程的请求边，使其请求的资源数减一
- 死锁解除
  - 重新启动系统
  - 撤销进程
  - 剥夺资源
  - 进程回退

死锁的避免

用动态的方法判断资源的使用情况和系统的状态，分配资源之前，判断是否会发生死锁，如果会，资源就不分配

1. 安全状态

   某一时刻，系统能按某种顺序为每个进程分配其所需资源，使每个进程都能顺利地完 成，则称此时系统处于安全状态。反之，称之为不安全状态。

   不安全状态不一定发生死锁，死锁一定属于不安全状态

   寻找一个安全序列
2. 银行家算法
   - 单个资源的银行家算法

   一个小城镇的银行家，他向一群客户分别承诺了一定的贷款额度，算法要做的是判断对请求的满足是否会进入不安全状态，如果是，就拒绝请求；否则予以分配。

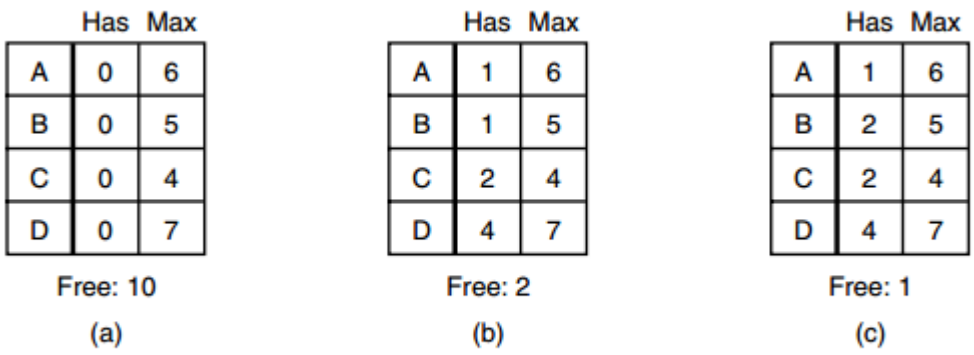


Figure 6-11. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

上图 c 为不安全状态，因此算法会拒绝之前的请求，从而避免进入图 c 中的状态。

- 多个资源的银行家算法

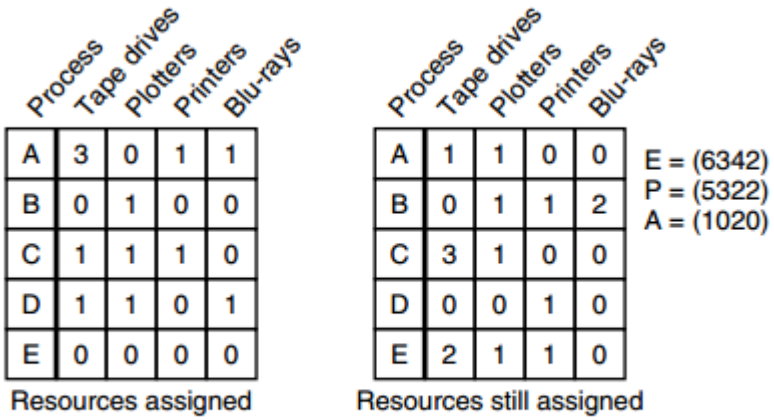


Figure 6-12. The banker's algorithm with multiple resources.

上图中有五个进程，四个资源。左边的图表示已经分配的资源，右边的图表示还需要分配的资源。最右边的 E、P 以及 A 分别表示：总资源、已分配资源以及可用资源，注意这三个为向量，而不是具体数值，例如 A=(1020)，表示 4 个资源分别还剩下 1/0/2/0。

检查一个状态是否安全的算法如下：

- 查找右边的矩阵是否存在一行小于等于向量 A。如果不存在这样的行，那么系统将会发生死锁，状态是不安全的。
- 假若找到这样一行，将该进程标记为终止，并将其已分配资源加到 A 中。
- 重复以上两步，直到所有进程都标记为终止，则状态时安全的。

如果一个状态不是安全的，需要拒绝进入这个状态。

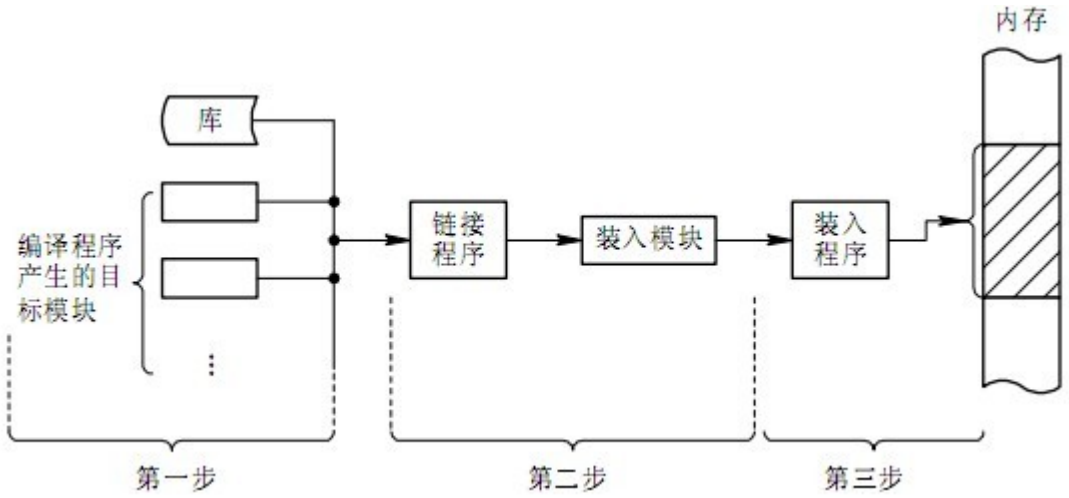
死锁的预防

死锁的预防策略是以破坏死锁产生的必要条件为目的，对资源的申请加以限制的

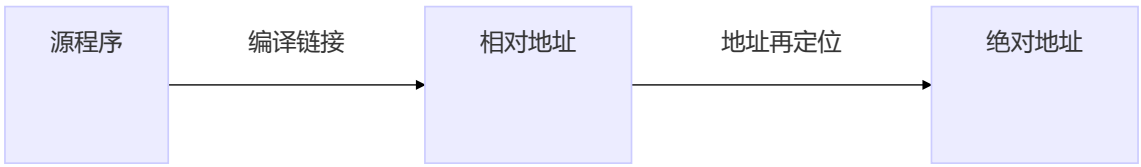
破坏互斥条件：某些设备可以通过 SPOOLING 系统将独享设备改造成为共享设备，以此 可以解决互斥问题，例如打印机。 **破坏非剥夺条件**：资源暂时释放策略，申请新的资源得不到满足则暂时释放已有的资 源。 **破坏占用并请求条件**：一次性申请全部资源。 **破坏循环等待条件**：资源有序申请，给资源编号，使用时按升序进行

第五章 存储管理

> 存储管理的主要管理对象是 **内存**
>
> ### 程序如何运行
>
> 在多道程序环境下，要使程序运行，必须先为之创建进程。而创建进程的第一件事，便是将程序和数据装入内存。如何将一个用户源程序变为一个可在内存中执行的程序，通常都要经过以下几个步骤：
>
> 首先是要编译，由编译程序(Compiler)将用户源代码编译成cpu可执行的目标代码，产生了若干个目标模块(Object Module)（即若干程序段），
>
> 其次是链接，由链接程序(Linker)将编译后形成的一组目标模块（程序段），以及它们所需要的库函数链接在一起，形成一个完整的装入模块(Load Module)；

> 最后是装入，由装入程序(Loader)将装入模块装入内存。图示出了这样的三步过程。
>
> 
>

地址重定位 （程序的装入）



地址重定位的方式：

1. **绝对装入**：在可执行文件中记录物理内存地址

   > 地址重定位在编译链接时确定

2. **可重定位装入**：可执行文件中列出 **需要重定位的地址单元** 以及 **相对地址**
   - **静态再定位**：程序执行（装入内存时）完成重定位
   - **动态再定位**：程序执行期间，在存储访问之前进行，装入内存的仍然是程序的逻辑地址值
3. **动态运行期装入**

链接

1. 静态链接：再生成可执行文件时进行
2. 动态链接：在装入或运行可执行文件时执行

存储管理方案

需要一次性全部装入内存的方案：

1. **分区存储：**

- 单一连续分区：作业运行时占用整个内存的地址空间
- 固定分区：在系统初始化时将内存空间分为若干个不等大的分区 **有内外碎片**
- 可变分区：装入或执行过程中，动态创建分区 **没有内碎片，有外碎片**
  - 采用空闲存储区表进行管理
  - 分区分配算法
    - 最先适应
    - 最佳适应
    - 最差适应
    - 循环最先适应

2. **段式存储：**

   段之间可以不连续

   - 可以实现共享、分段保护、动态链接、段可以动态增长

3. **页式存储：**

   作业的虚拟地址空间划分为若干个 **长度相等的页（虚页）**，主存划分为与虚页长度相等的 **页框（实业、块）**

   - 系统在内存中开辟 **页面变换表（进程页表）** 页表中的内容有 **页号、块号、页内偏移量**
   - 记录内存中哪些页面是空闲的数据结构是 **内存页表（物理页面表、空闲内存页表）**
   - 描述系统内进程页表的位置和大小：**请求表**

   硬件支持：

   - 页表基地址寄存器
   - 页表长度寄存器
   - **快表（联想寄存器）** 加速地址变换过程

   页式存储优点：

   - 没有外碎片
   - 程序不必连续存放

   缺点：

   - 程序全部装入内存
   - 不利于共享和动态链接

> 段式和页式的比较：
>
> | | 目的 | 大小 | 逻辑地址 | 表的长度 | 内存共享 |
> |---|---|---|---|---|---|
> | **段式** | 用户应用的需要 | 大小不固定 | 二维，每段一个逻辑地址空间 | 段表短 | 能 |
> | **页式** | 系统管理的需要 | 大小固定 | 一维，同一个地址空间 | 页表长 | 不能 |

4. **段页式：**

   用户程序按段式划分，物理内存按页式划分

   - 系统需要同时配置段表和页表
   - 获取数据需要三次访问内存

不需要一次性装入：

- 交换技术和覆盖技术
- **虚拟存储**
  - 虚拟存储管理的基础是 **程序的局部性原理**，程序在执行过程中的一个较短时期内，所执行的指令地址和指令的操作数地址分别局限于一定的区域

    表现为： **时间局部性、空间局部性**
  - 分类：**请求分页、请求分段、请求段页式存储管理**
  - 如果要访问的 页（段） 不在内存，**引发 缺页（段）中断**
  - 页面调度的策略：
    - 页面调度策略
    - 置页策略
    - 页面置换策略
  - **页面淘汰算法：**
    - 最佳算法
    - **先进先出 FIFO**：会产生 **Belady 现象**，原因是算法的置换特征与进程访问内存的动态特征相矛盾
    - 第二次机会
    - 页面缓冲算法
    - **最近最少使用 LRU**
      - 硬件实现
      - 软件实现
        - 最不经常使用 NFU
        - 最近未使用 NRU

第六章 文件管理

文件

定义： 文件是是**记录在外存上的**，具有**符号名**的，在**逻辑上具有完整意义**的一组**相关信息项的集合**

> **文件的组成部分：**
>
> 1. **文件体：** 文件真实的内容
> 2. **文件说明：** 操作系统为了管理文件所用到的信息

文件系统：

定义： 是操作系统中实现文件统一管理的一组软件和相关数据的集合，是专门负责管理和存取文件信息的软件机构

> 文件系统的功能（了解）：
>
> 按名存取、统一的用户接口、并发访问和控制文件、安全性控制、优化性能、差错恢复

文件的结构

- **逻辑结构：**是从用户观点出发，所观察到的文件组织形式，是用户可以直接处理的数据及其结构，它独立于物理特性，又称为文件组织 (File Organization)。
  - **有结构的记录式文件**：定长记录、变长记录，两种记录在处理前，每个记录的长度是可知的
  - **无结构的流式文件：** 文件体为字节流，采用顺序访问方式
- **物理结构（四种）：**是指文件的内部组织形式，即文件在物理存储设备上的存放方法
  - **连续结构（顺序结构）**
    - 优点：
      1. 存储方式简单。
      2. 对文件记录进行批量存取时，其存取效率较高。
      3. 支持定长记录的直接存取，可以通过计算获得存储位置。
    - 缺点：
      1. 不支持随机查找。如果要随机地查找或修改单个记录，此时系统需要逐个地查找诸记录，性能较差，尤其是当文件较大时情况将更为严重。
      2. 存在外部碎片。
      3. 不便于记录的增加或删除操作。
  - **链接结构：**将逻辑上连续的文件信息存放在不连续的物理块上，每个物理块设有一个指针指向下一个物理块。
    - 优点：
      1. 提高了磁盘空间利用率，不存在外部碎片问题。
      2. 有利于文件插入和删除，及其动态扩充。
    - 缺点：
      1. 仍然不支持随机查找。
      2. 由于存储空间可能不连续，带来更多的寻道次数和寻道时间。
      3. 需要牺牲一些空间存放链接指针，同时需要维护这些指针，增加了系统开销。
      4. 可靠性问题，如指针出错。
  - **索引结构**：将逻辑上连续的文件信息(记录)存放在不连续的物理块中，系统为每个文件建立一个专用数据结构——索引表，索引表中存放文件的逻辑块号和物理块号的对应关系。
    - 优点：
      1. 即能顺序存取,又能直接存取。
      2. 满足了文件动态增长、插入删除的要求。
      3. 没有外碎片，外存空间利用率较高。
    - 缺点：
      1. 较多的寻道次数和寻道时间。
      2. 索引表本身需要存储空间，同时对索引表的维护会增加系统开销。

    UNIX 文件系统的索引结构：

四种寻址方式：直接、一级间接、二级间接、三级间接

- **Hash 文件**：采用计算寻址结构，它由主文件和溢出文件组成

  在每个记录中需要有一个关键字字段，检索时给出记录键值，通过哈希函数计算出该记录在文件中的相对位置。这就是通常所说的Hash方法（散列法或杂凑法），利用这种方法所建立的文件称为Hash文件。

文件的存取方式

- **顺序存取**：指对文件中的信息按顺序依次读写的方式在提供记录式文件结构的系统中，顺序存取法就是严格按物理记录排列的顺序依次读取。
- **随机存取**：
  - **直接存取法**：允许用户随意存取文件中任意一个物理记录。
  - **按键存取法**： 根据文件中各记录的某个数据项内容来存取记录的，这种数据项称之为"键"。 （Hash）

文件目录

文件控制块（FCB）的有序集合称为文件目录，文件目录是由文件控制块组成的，专门用于文件的检索，实现「**按名存取**」

> **文件控制块的主要内容及作用：**
>
> - 基本信息类：文件名、文件的物理地址
> - 存取控制信息类：文件的存取权限
> - 使用信息类：文件建立日期、修改日期、访问日期、当前使用的信息

文件目录提供的功能：

- **实现"按名存取"**。用户只须提供文件名，即可对文件进行存取。这是文件系统向用户提供的最基本的服务。
- **提高对目录的检索速度**。合理地组织目录结构，加快对目录的检索速度，从而加快对文件的存取速度。这是在设计一个大、中型文件系统时，所追求的主要目标。
- **实现文件共享**。在多用户系统中，应允许多个用户共享一个文件，以节省大量的存储空间并方便用户。
- **解决文件重名问题**。系统应允许不同用户对不同文件采用相同的名字，以便于用户按照自己的习惯命名和使用文件。

文件目录结构：

- **一级目录结构：** 优点：简单；缺点：1. 查找速度慢 2. 不允许重名 3.不便于实现文件共享
- **二级目录结构：** 为每一个用户建立一个单独的用户文目录UFD
- **多级目录**：或称为树状目录。在文件数目较多时，便于系统和用户将文件分散管理。适用于较大的文件系统管理。

  优点:

  - 提高了检索目录的速度
  - 较好地解决了重名问题

  缺点：

  - 这种结构不便于用户共享文件

记录的成组与分解

把若干个逻辑记录合成一组存放在一个物理块的过程。进行成组操作时必须使用主存缓冲区，缓冲区的长度等于逻辑记录长度乘以成组的块因子。

记录成组的优点是提高了存储空间的利用率；减少了启动外设的次数，提高系统的工作效率。主要缺点是需要软件增加成组和分解的额外操作，以及容纳最大块长的I/O缓冲区。

外存空间管理

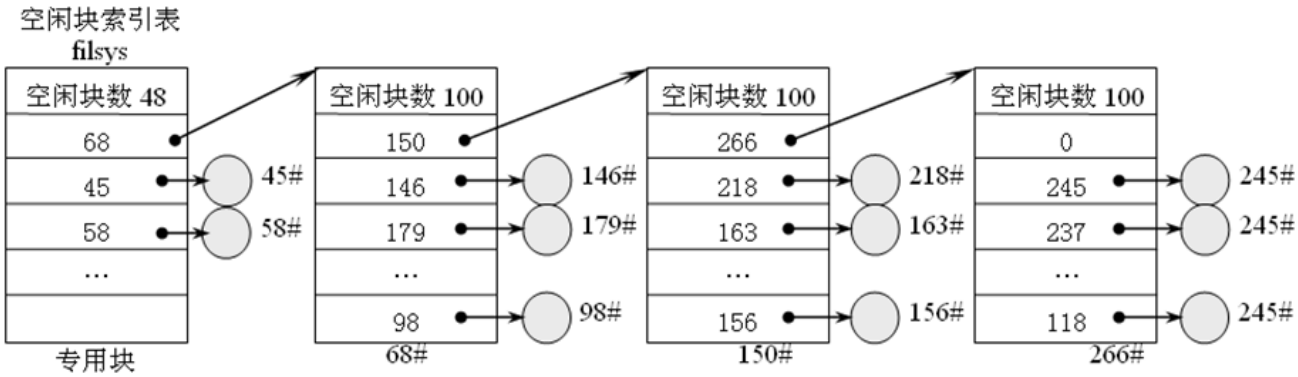
外存空闲空间管理的数据结构通常称为 **磁盘分配表**

常用的空闲空间的管理方法有：**空闲区表、位示图和空闲块链**三种。

Unix 系统的成组链接法

将空闲块分成若干组，每100个空闲块为一组。每组的第一个空闲块登记了下一组空闲块的物理盘块号和本组空闲块总数 。

理解掌握分配和回收空闲盘块的算法



文件空间的分配和管理

常用的外存分配方法

- 连续分配
- 链接分配
- 索引分配

内存中所需的表目

> 为了提高系统的工作效率，在内存设置文件管理机构称为打开文件机构

1. **内存文件控制块（内存 I 节点）**

   > 当打开某一个文件时，如果找不到其相应的内存I节点，就在内存I节点表中分配一个空闲表项，并将该文件的外存I节点中的主要部分复制进去，填入外存I节点号。

   每一个打开的文件，只能有一个内存文件控制块

2. **系统打开文件表**

   子进程共享父进程的全部打开文件，系统打开文件表共用，修改 `f_count` 计数+1

3. **用户（进程）打开文件表**

   每一个进程都有一张用户打开文件表，子进程在父进程的打开文件表的基础上增加表项

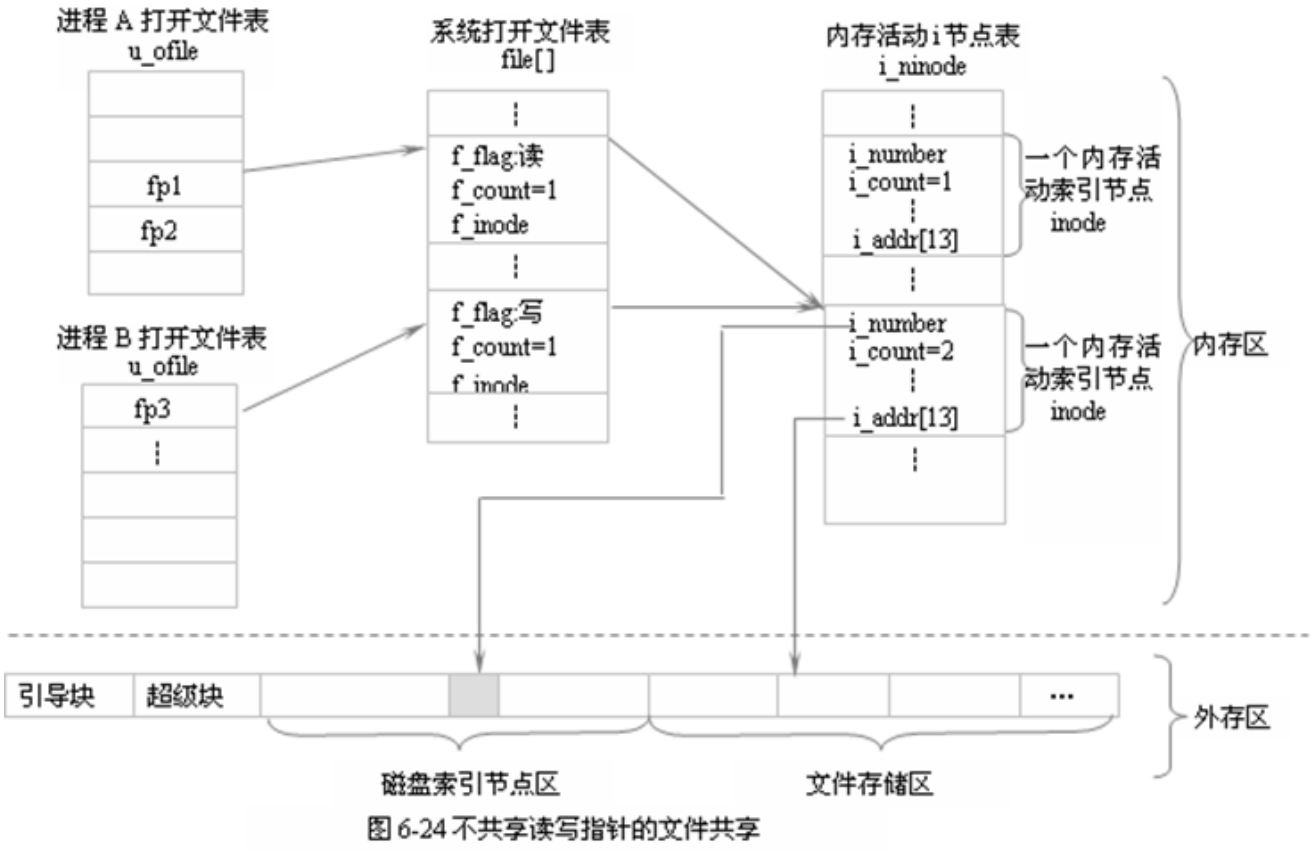


图 6-24 不共享读写指针的文件共享

文件共享

- **基于索引节点的共享**:
  - **静态共享**(硬链接)

    一个文件同时属于多个文件目录项(例如被多个用户共享)，并且这种关系不管文件此时是否在被使用，都存在。

    Unix 通过**索引节点**（inode）来实现文件共享链接的，并且只允许链接到文件，不允许链接到目录
  - **动态共享**

    出现在进程共享文件时，伴随着进程的生成而存在，进程的终止而消失。
    - **不共享读写指针的文件共享：** 不共用系统打开文件表项，共用内存活动 I 节点
    - **共享读写指针的文件共享：** 共用系统打开文件表项
- **利用符号链接共享**
  - 软连接：也称符号链接，**生成符号链接文件**时，将符号链接文件登记在该用户共享目录项中，文件内容为被链接文件的路径名。

可靠性与安全性

转储和恢复

常用的转储方法：**静态转储和动态转储、海量转储和增量转储**

存取控制

- 存取控制矩阵

- 存取控制表
- 用户权限表

磁盘调度

先进行移臂调度、然后进行旋转调度

磁盘移臂调度算法

- **先来先服务：** 根据请求的先后顺序调度，公平，简单。平均寻道距离大
- **最短寻道时间有先 SSTF：**访问的磁道与当前磁头所在的磁道距离最近，不能保证平均寻道时间最短，可能导致「饥饿」现象
- **扫描算法 SCAN ：** 又称「电梯调度」电梯总是保持一个方向运行，直到该方向没有请求为止，然后改变运行方向。电梯算法（扫描算法）和电梯的运行过程类似，总是按一个方向来进行磁盘调度，直到该方向上没有未完成的磁盘请求，然后改变方向。

  因为考虑了移动方向，因此所有的磁盘请求都会被满足，解决了 SSTF 的饥饿问题。
- **循环扫描调度 CSCAN：**磁头只做单向运动

磁盘旋转调度算法

同一柱面，扇区号从小往大依次访问，遇到相同扇区号，选择一个访问，另一个下一周访问

第七章 设备管理

计算机中负责管理I/O的机构称为I/O系统（硬件和软件的组合）

IO 设备的 I/O 控制方式

- **程序控制 I/O**

  在一个设备的操作没有完成时，控制程序一直检测设备的状态，直到该操作完成，才能进行下一个操作。
- **中断驱动 I/O**

  I/O 操作由程序发起，在操作完成时，由外设向CPU发出中断，通知该程序，数据的每次读写通过 CPU。
  - **优点**：在外设进行数据处理 时，CPU不必等待，可以继 续执行该程序或其他程序。 提高了CPU的利用率。中断技术使得CPU和外设之间的并行工作成为可能。
  - **缺点**：**数据仍然需要通过CPU进行传输**，由于CPU每次处理的数据量少，因此这种方式**只适于数据传输率较低的设备。**
- **直接存储访问 I/O**

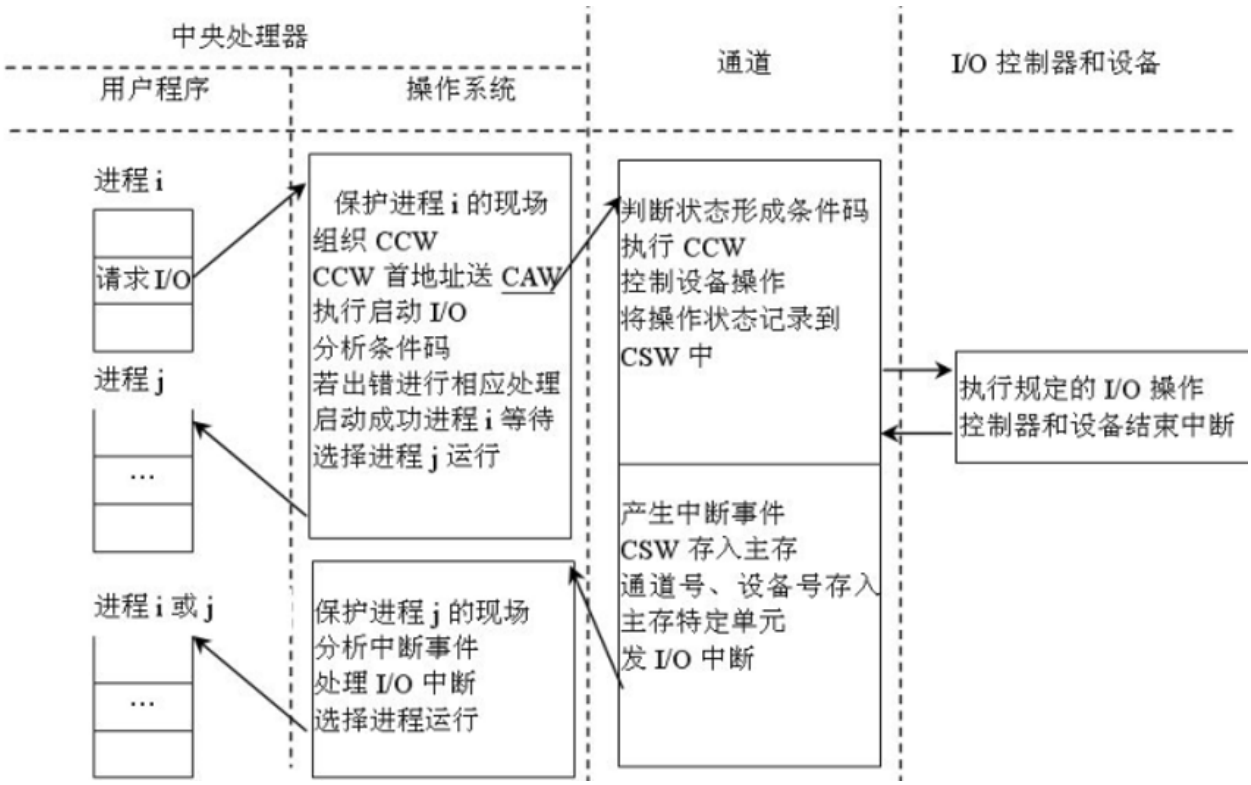
  对I/O设备的控制由DMA控制器完成，在DMA控制器的作用下，设备和主存之间可以成批地进行数据交换，而不用CPU的干涉。
  - **优点：** CPU 只需干预 I/O 的开始和结束，适于高速设备
- **通道控制方式 I/O**

  可进一步减少CPU的干预，即把对一个数据块的读(写)为单位的干预，减少为对一组数据块的读(写)及有关的控制和管理为单位的干预。 **一个通道控制多台设备。**CPU仅在I/O操作的开始和结束时花费少量时间处理与I/O有关的工作。**实现CPU、通道和I/O设备三者的并行操作**，从而更有效地提高整个系统的资源利用率。

通道的类型：

- **字节多路通道**：使用时间片轮转，轮流为各个子通道服务
- **数组选择通道**：高速传输数据，不支持并发
- **数组多路通道**：高速传输，支持并发

通道的工作原理：



IO 设备的分类

- **按数据组织分类**
  - **块设备：** 指以**数据块为单位**来组织和传送数据信息的设备。它属于**有结构**设备。块设备的基本特征是 **传输速率较高**，通常每秒钟为几兆位；它是**可寻址**的，即可随机地读/写任意一块；磁盘设备的I/O采用DMA方式。
  - **字符设备：** 指**以单个字符为单位**来传送数据信息的设备。这类设备**一般用于数据的输入和输出**(交互式终端、 打印机)。它属于无结构设备。
- **按数据传输率分类**

  低速设备、中速设备、高速设备
- **资源分配角度分类**
  - **独占设备**：在一段时间内只允许一个用户（进程）访问的设备。（终端、打印机）
  - **共享设备**：在一段时间内允许多个进程同时访问的设备。（磁盘）
  - **虚拟设备**：通过虚拟技术将一台独占设备变换为若干台供多个 用户（进程）共享的逻辑设备。（SPOOLing）

IO 设备管理软件

- **中断处理程序**

  解决高速处理设备和低速输入输出设备之间的矛盾，提高系统工作效率。

- **设备驱动程序**

  接受来自与设备无关的上层软件的抽象请求；进行与 设备相关的处理。

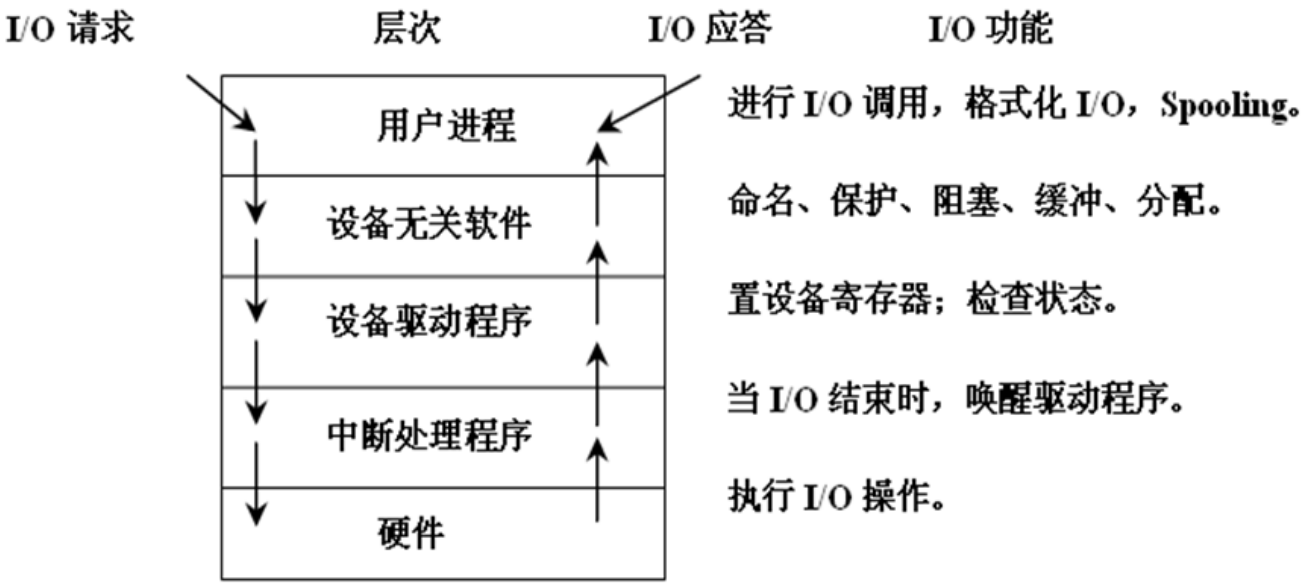
- **与设备无关的系统软件**

  是建立在设备驱动程序之上的，与具体设备无关的I/O功能的集合

- **用户空间的 I/O 软件**

  Spooling系统：构成虚拟设备

  - 提高了I/O的速度，缓和了CPU与低速I/O设备速度不匹配的矛盾
  - 利用高速共享设备，将独占设备改造为共享设备。
  - 实现了虚拟设备功能：用户都感到独占了一台设备。



缓冲

引入缓冲的原因：

- 缓和CPU与I/O设备间速度不匹配的矛盾
- 减少对CPU的中断频率，放宽对中断响应时间的限制
- 提高CPU和I/O设备之间的并行性

缓冲技术的分类

- 单缓冲、双缓冲、环形缓冲、缓冲池

UNIX 的缓冲区管理

为方便对缓冲区进行管理，UNIX系统设置了三种队列。

- **自由 buf 队列**

  在Unix系统中，一个可被分配作它用的缓冲存储区，其相应的 buf 位于自由buf队列中。 自由buf队列采用 FIFO 管理算法。一个缓存被释放时，其相应的 buf 被送入自由buf队列的队尾；当要求分配一个缓存时，从队首取出一个buf，它所管理的缓存就可被“移作它用”。

- **设备 buf 队列**

每类设备都有一个buf队列。 一个缓存被分配用于读、写某类块设备上的某一字符块时，其相应的buf就进入该类设备的buf队列，**除非被“移作它用”，否则一直保留在该队列中。**

- **NODEV 设备队列**

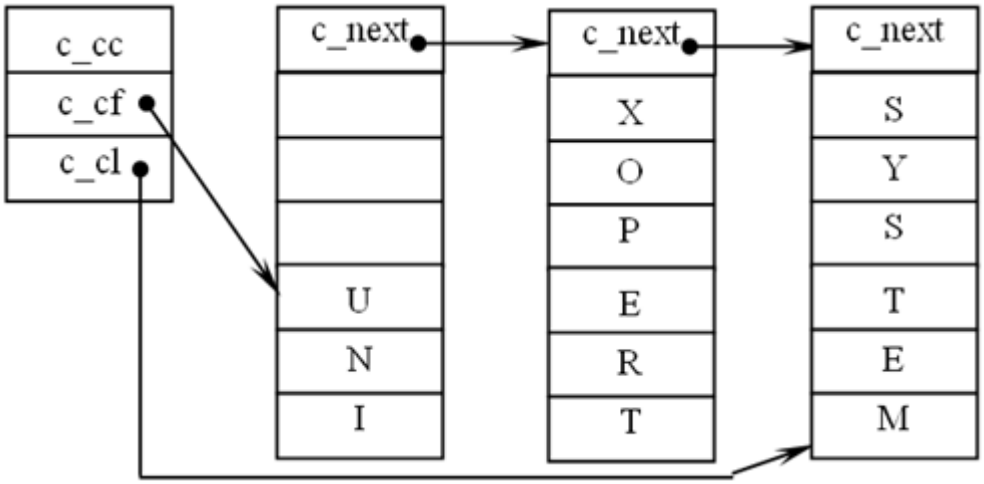
  当系统需要使用缓存，但不与特定的设备字符块相连时，则将分配到的缓存控制块buf送入NODEV设备队列。用于传参以及存放文件系统的资源管理块

一个可以移作他用的缓存buf，同时处于原设备buf队列中和自由buf队列中。

1. 缓存buf内容不变，如若还要用，直接从自由buf队列中抽出。避免了重复、费时的I/O操作过程，从而大大提高了系统的效率。
2. 如果要将一个缓存重新分配，只需要将它从自由buf队列和设备 buf队列中同时抽出，送入新的设备buf队列。实现了进程对有限缓存的共享。

字符设备缓存

- 自由字符缓存队列：由空闲的字符缓存构成自由队列。
- I/O字符缓存队列：字符设备通过字符缓存进行输入或输出。各个正被使用的字符缓 存按照它们的不同用途形成多个I/O队列，每个队列设置一个控制块



第九章 分布式计算机系统

分布式计算机系统(Distributed Computer Systems)是由**多个分散的计算机经网络连接而形成的统一的计算机系统。**其中各个资源单元(物理的或逻辑的)**既相互协同又高度自治**，能在**全系统范围内实现资源管理**，动态地进行任务分配或功能分配，并能**并行地**运行**分布式程序。**

分布式系统的特征

分布性、自治性、透明性、共享性、协同性

分布式系统的基本功能

通信、资源共享、协同工作

通信方式

- **实现消息传递的通信原语**
- **远程过程调用 RPC**：允许程序调用位于其它节点机器上的过程。调用过程如下：当节点A 上的进程想调用节点B上的一个过程时， A上的调用进程被挂起，调 用信息以参数的形式从节点A传送到节点B，在B 上执行被调用过程， 然后将执行的结果返回节点A。对程序员来说，他看不到消息传递过 程和I/O处理过程。

死锁

- 资源型死锁：可剥夺、非剥夺。在一组进程序竞争非剥夺性资源时，是会因进程的推进顺序不当，进入了不安全区，从而导致发生进程死锁
- 消息型死锁：在不同结点中的进程，为发送和接收分组而竞争缓冲区，以致发生了既不能发送消息，也不能接收消息的僵持状态。