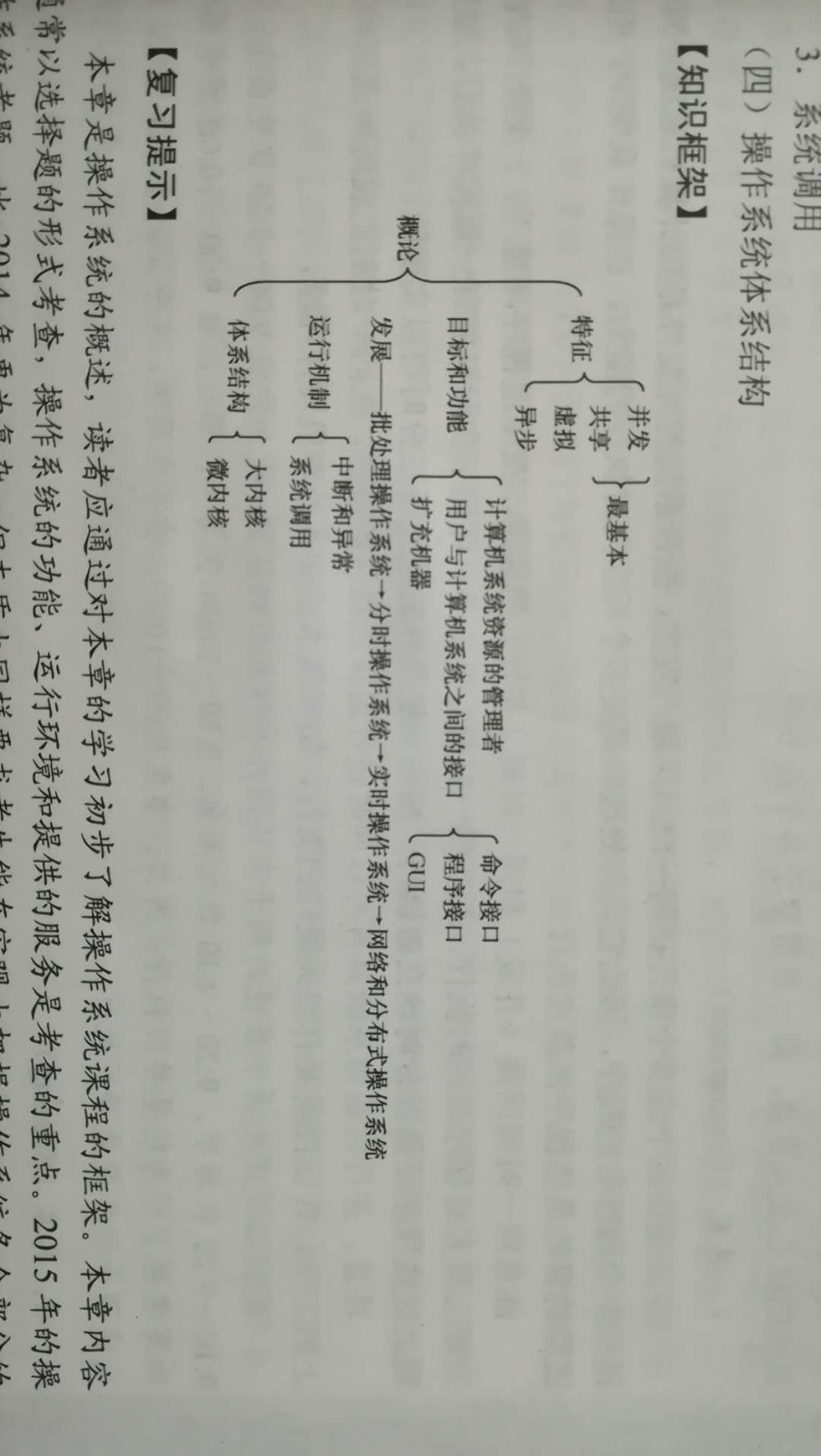
# 【操作系统概述】



操作系统：一种系统软件，管理计算机硬件和软件，组织工作和调度资源，以提供给用户和软件方便的接口。

基本特征：并发、共享、虚拟、异步

工作内容：进程管理、存储器管理、文件管理、设备管理等

## 【特征】

### 【并发 concurrence】同时运行多个程序

注意并发与并行的区别：

**并发**——同一时间段内间隔运行进程。每一时刻，单处理机环境下实际只能执行一道程序，操作系统通过分时实现并发。

**并行**——同一时刻完成多种工作，需要硬件支持，比如多处理机硬件环境。

### 【共享 sharing】资源共享

系统中资源可供多个并发执行的进程共同使用。

**互斥共享方式**——如打印机，虽然可以共享，但一段时间内只允许一个进程访问。进程A访问某资源时，必先提出请求，如果资源空闲，系统把资源分配给A进程。这时B进程再访问就必须等待。

**同时访问方式**——同时是宏观上的，微观上还是分时共享，进程交替访问资源。如磁盘设备，可以被多个进程“同时”访问。

### 【虚拟 virtual】把一个物理实体变成若干个逻辑对应物

虚拟处理器——时分复用

虚拟存储器——空分复用

### 【异步 asynchronism】

在多道程序环境下，允许多个程序并发执行，但是由于资源有限，进程的执行并不是一贯到底，而是走走停停，以不可预知的速度向前推进。（可能导致进程产生与时间有关的错误，就像对全局变量访问顺序不当会导致程序出错一样）

【用户与系统之间的接口】

命令接口

程序接口

（GUI）

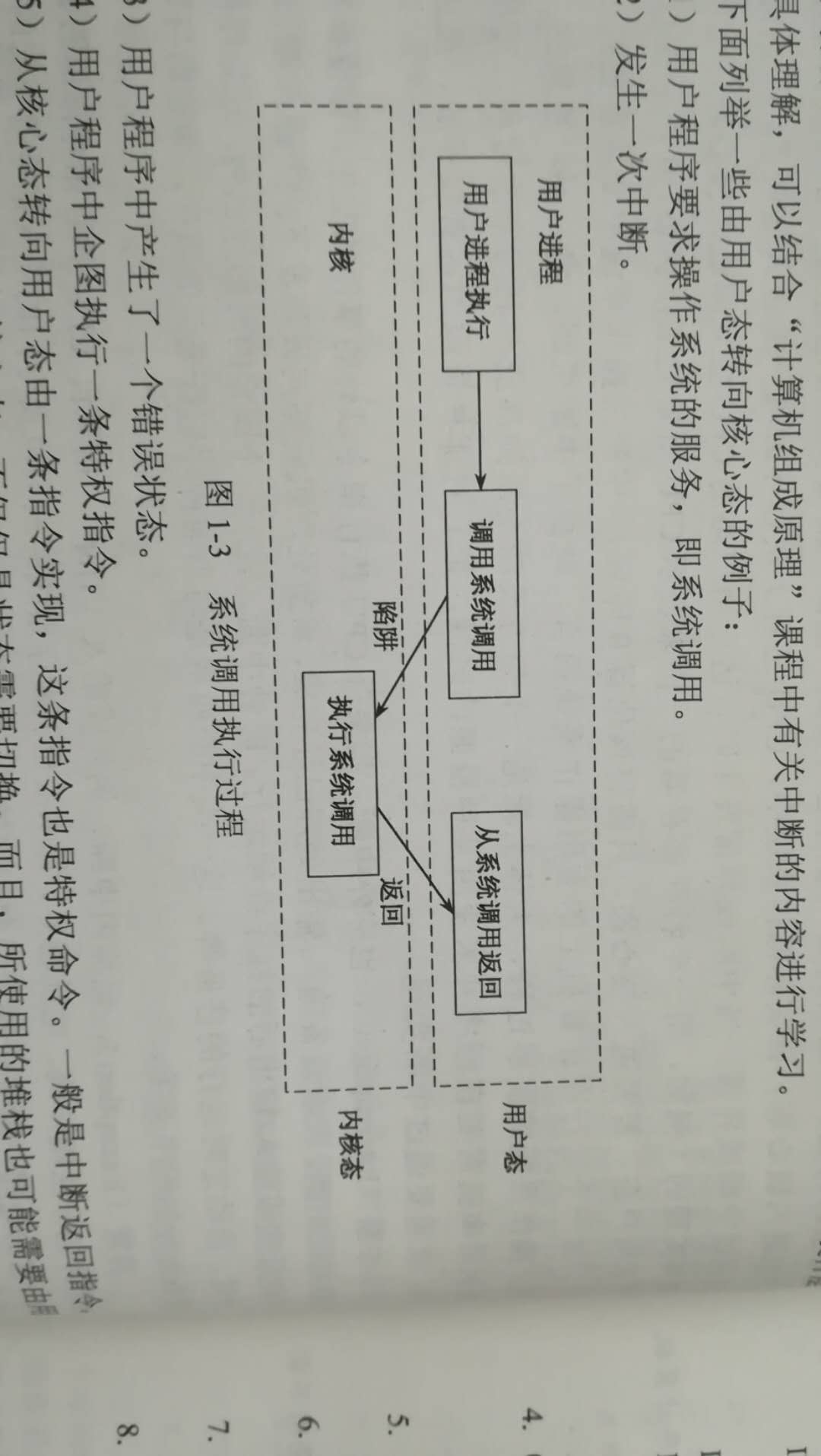
【分布式计算机系统】

系统中任意两台计算机可以通讯，系统中每一台计算机具有相同地位，没有从机和主机，每台计算机上的资源为所有用户所共享

【操作系统的运行机制】

CPU执行两种不同性质的程序：①操作系统内核程序，②用户自编程序或系统外层的应用程序。

【用户态、内核态】



【原语 atomic operation】

底层的可被调用的公用小程序，不可被分割，不可被中断。

## 【运行机制】

### 【中断 interruption】也称外中断

来自CUP执行指令以外事件，如：

设备发出I/O结束中断，表示输入/输出处理已经完成，希望处理机能向设备发送下一个输入/输出请求。

时钟中断，表示一个固定时刻已到，让处理机执行其他任务。

这些中断与当前程序无关，来自于外部事件

### 【异常 exception】也称内中断

非法操作、地址越界、算术溢出

### 【系统调用】

操作系统提供给用户的一些子功能，可以在程序中调用。凡是与资源有关的操作，都必须通过系统调用的方式，由操作系统代为完成。

设备管理

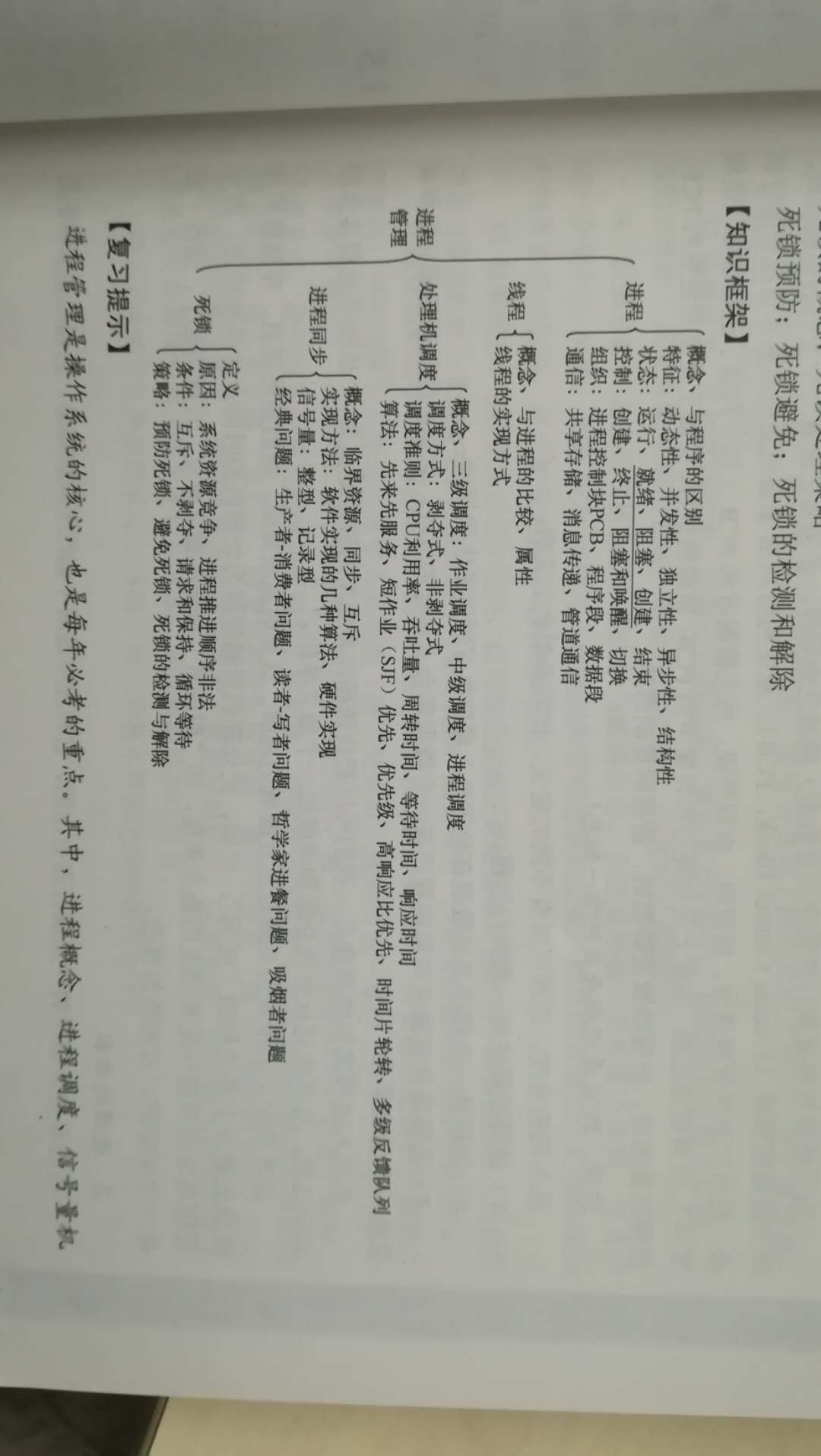
文件管理

进程控制

进程通讯

内存管理

# 【进程管理】



## 【进程 process】

进程是有多程序的并发执行而引出的，它和程序是截然不同的两个概念。

动态性——进程是程序的一次执行

并发性——引入进程是为了使程序与其他进程的程序并发执行

独立性——进程是独立运行、独立获得资源、独立接收调度的基本单位

异步性——进程按各自独立的、不可预知的速度向前推进

结构性——进程由程序段、数据段、进程控制段组成，每个进程都配一个PCB

【进程控制块 Process Control Block】PCB，一个专门的数据结构

创建进程——实质是创建进程映像中的PCB

撤销进程——实质是撤销进程映像中的PCB

【进程映像】由程序段、相关数据段、PCB组成

### 【进程的5个状态】

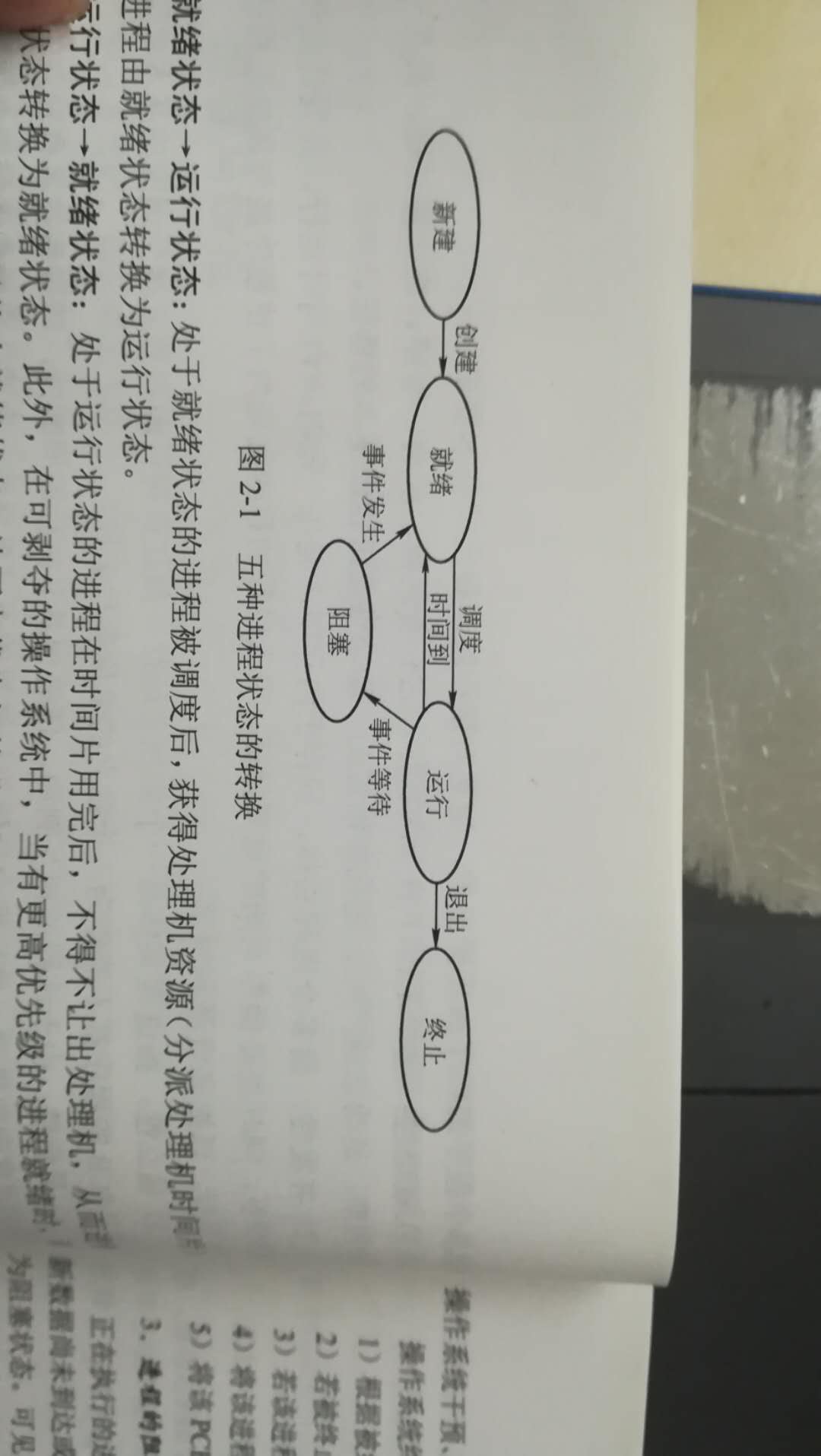
**运行**——进程正在处理机上运行。单处理机环境下，每一时刻只有一个进程在运行

**就绪**——进程获得了除处理机以外的一切所需资源，一旦得到处理机就可以运行

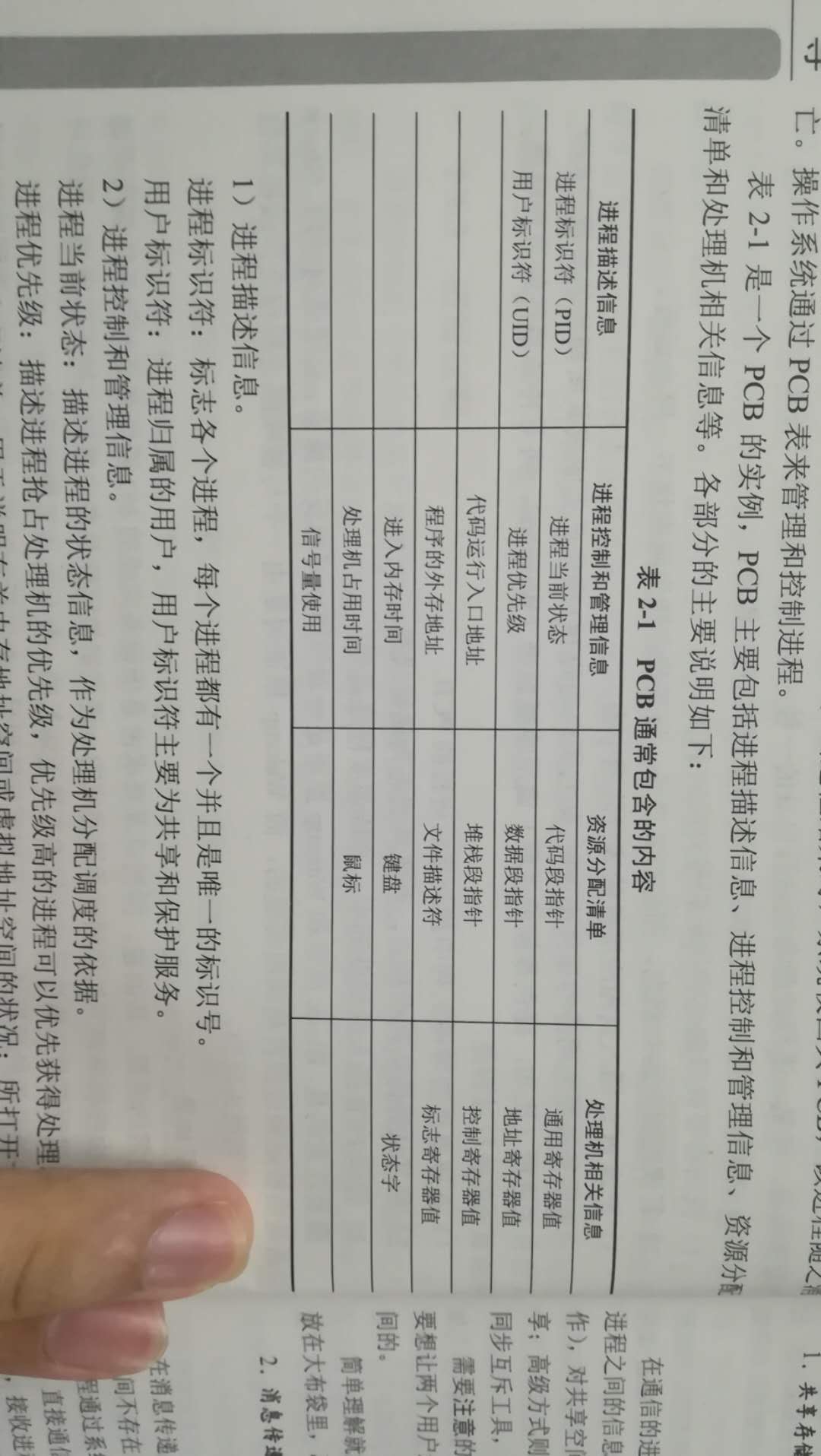
**阻塞**——又称等待状态，进程正在等待某一事件而暂停运行，即使处理机闲置也不能运行

**创建**——进程正在被创建，尚未进入就绪状态

**结束**——当进程需要结束时，系统将其置位结束状态，然后再回收资源



### 【PCB】



## 【线程 thread】

进程process——实现并发

线程thread——减小并发的开销

引入线程后，线程是一个基本的CPU执行单元，可以理解为“轻量级进程”。

进程是拥有资源的基本单位，线程没有资源，同一进程中的各线程共享进程的资源。同进程中的线程切换，不会引起进程切换；一个进程中的线程切换到另一个进程中的线程，会引起进程切换。

**在多CPU的计算机系统中，各个CPU同时为一个进程内的多个线程服务，可以缩短进程的处理时间。**

为什么线程的提出有利于提高系统的并发性？因为有了线程，线程切换时，有可能会发生进程切换，也有可能不发生进程切换，那么平均下来，每次切换所需的开销就小了，因而，能让更多的线程参与并发，也不会影响到响应时间等问题了。

## 【进程同步】

### 【临界资源】

一次只能被一个进程使用，必须互斥地访问。访问临界资源的代码段叫做**临界区**。

### 【同步】

直接制约关系。多个进行协同完成任务时，在某些位置上协调它们的工作次序而等待、传递信息所产生的制约关系。

### 【互斥】

间接制约关系。一个进程使用临界资源，另一个进程就必须等待。

### 【信号量】

信号量S只能被两个标准的原语wait(S)和signal(S)访问，也可以记为“P操作”和“V操作”。

**利用信号量实现同步**

定义S为process1和process2的公共信号量，process2中的语句y要在process1完成语句x之后才能执行，利用S解决同步问题：

semaphore S=0;

process1()

{

…

x;

V(S); //告诉process2，x语句已经完成

…

}

process2()

{

…

P(S); //检测x语句是否完成

y;

…

}

**利用信号量实现互斥**

semaphore S=1;

process1()

{

…

P(S); //加锁

临界区 //访问临界资源

V(S); //解锁

…

}

process2()

{

…

P(S); //加锁

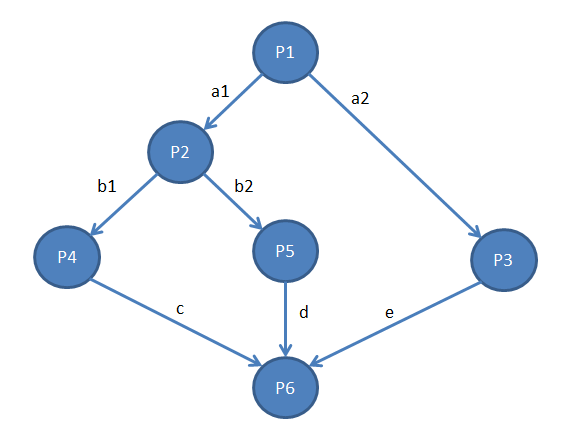
临界区 //访问临界资源

V(S); //解锁

…

}

**利用信号量实现前驱关系**



P1

{

…

V(a1);V(a2);

}

P2

{

P(a1)

…

V(b1;V(b2);)

}

P3

{

P(a2);

…

V(e);

}

P4

{

P(b1)

…

V(c);

}

P5

{

P(b2);

…

V(d);

}

P6

{

P(c);P(d);P(e);

…

}

### 【管程 monitor】一种同步互斥工具

### 【经典同步问题】

消费者-生产者问题

读者-写者问题

哲学家进餐问题

吸烟者问题

### 【死锁】

【产生原因】进程推进顺序非法

①请求和释放资源的顺序不当：并发进程P1、P2分别保持了资源R1、R2，而进程P1申请资源R2，进程P2申请资源R1，两者都会因为所需资源被占用而阻塞。

②信号量使用不当：进程A等待进程B法的消息，而进程B又在等待进程A发的消息，双方并非争夺资源，而是因等待对方资源而导致死锁。

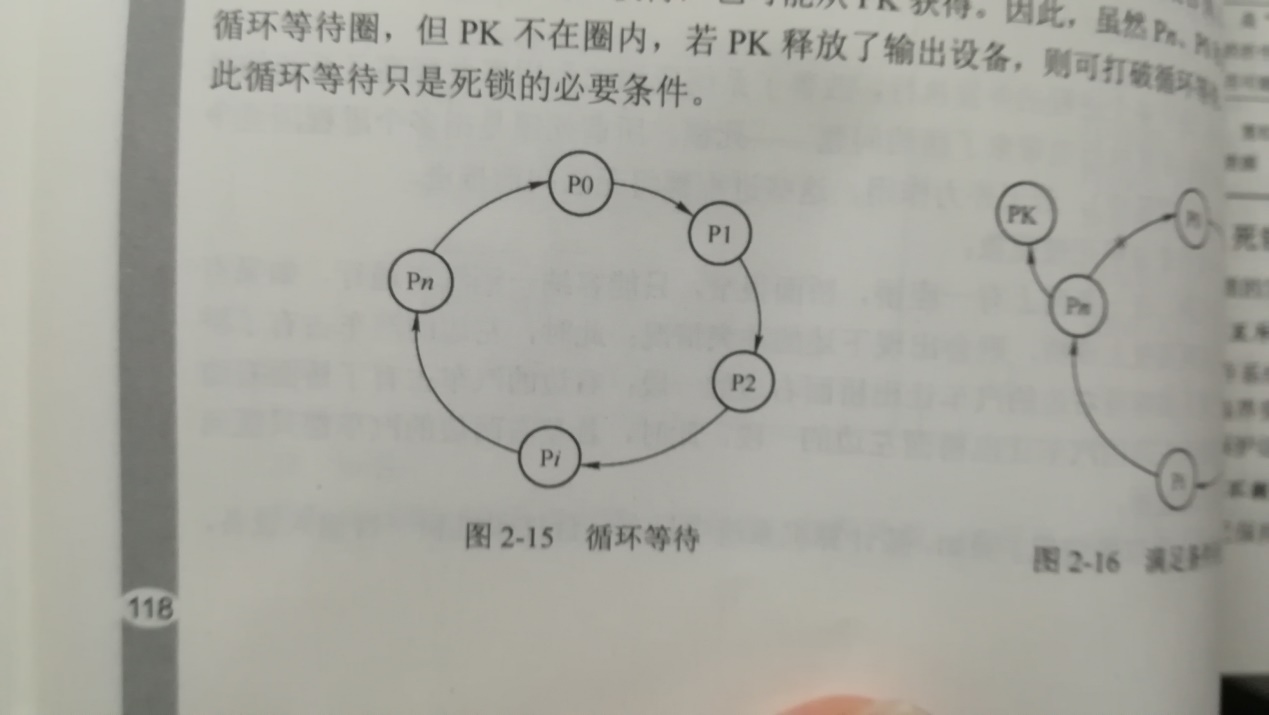
【必要条件】

互斥条件：某资源一段时间内仅为一个进程所占有

不剥夺条件：资源只能通过进程自己释放，不能被其他进程强行夺走

请求和保持条件：对新资源请求不到，而对自己的资源又保持不释放

循环等待条件：



【死锁的处理策略】

预防死锁——破坏四个必要条件中的一个或几个

避免死锁——系统安全状态、银行家算法

死锁检测及解除——

# 【内存管理】

## 【虚拟内存】

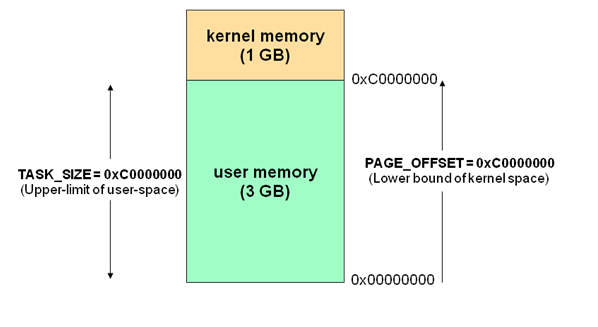
当物理内存（内存条）不够时，在硬盘上开一块空间来用，这块空间不是由内存提供的，座椅叫“虚拟内存”。

目前计算机的存储器系统一般分为三个层次，cache—主存—辅存。cache是为了解决cpu与主存处理速度的矛盾加入的，辅存便是为了扩大主存容量。降低成本。“虚拟”字面意思就是假的，不真实的，虚拟存储系统是忽略了主存与辅存的差异，将其都看作是主存。它基于程序的局部性原理，在程序运行时，将当前运行所需的程序和数据调于内存，其余部分依旧放在辅存，当要运行辅存部分的程序时便从辅存调入主存，如果主存已满，就按照一定的算法与主存的一部分进行替换。这操作对于应用程序员而言是透明的，他们察觉不到这些由操作系统和硬件共同完成的步骤，看起来就觉得辅存也属于主存的一部分。

## 【用户空间与内核空间】

现在操作系统都是采用虚拟存储器，那么对32位操作系统而言，它的寻址空间（虚拟存储空间）为4G（2的32次方）。

为了保证用户进程不能直接操作内核，保证内核的安全，操心系统将虚拟空间划分为两部分，一部分为内核空间，一部分为用户空间。针对linux操作系统而言，将最高的1G字节（从虚拟地址0xC0000000到0xFFFFFFFF），供内核使用，称为内核空间，而将较低的3G字节（从虚拟地址0x00000000到0xBFFFFFFF），供各个进程使用，称为用户空间。每个进程可以通过系统调用进入内核，因此，Linux内核由系统内的所有进程共享。于是，从具体进程的角度来看，每个进程可以拥有4G字节的虚拟空间。



有了用户空间和内核空间，整个linux内部结构可以分为三部分，从最底层到最上层依次是：硬件-->内核空间-->用户空间。

**内核态与用户态：**

（1）**当一个任务（进程）执行系统调用而陷入内核代码中执行时，称进程处于内核运行态（内核态）。**此时处理器处于特权级最高的（0级）内核代码中执行。当进程处于内核态时，执行的内核代码会使用当前进程的内核栈。每个进程都有自己的内核栈。

（2）**当进程在执行用户自己的代码时，则称其处于用户运行态（用户态）**。此时处理器在特权级最低的（3级）用户代码中运行。当正在执行用户程序而突然被中断程序中断时，此时用户程序也可以象征性地称为处于进程的内核态。因为中断处理程序将使用当前进程的内核栈。

# 【文件管理】

# 【IO管理】