

进程同步与信号量习题课

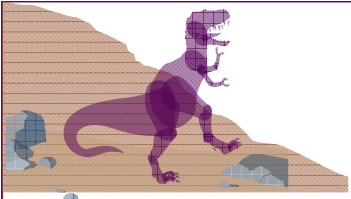
肖 卿 俊

办公室：计算机楼532室

电邮：csqjxiao@seu.edu.cn

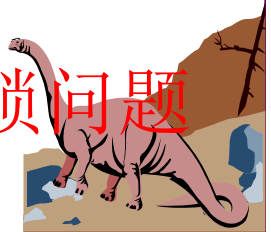
主页： <http://cse.seu.edu.cn/PersonalPage/csqjxiao>


电话：025-52091023



进程同步与互斥：习题1

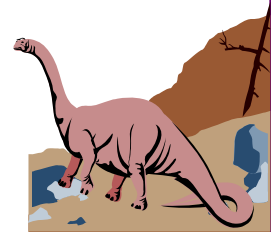
- 假设有5个哲学家，共享一张放有五把椅子的桌子，每人分得一把椅子。
- 桌子上总共只有5支筷子，在每人两边分开各放一支
- 哲学家们在肚子饥饿时才试图分两次从两边拾起筷子就餐。
- 条件：
 1. 只有拿到两支筷子时，哲学家才能吃饭。
 2. 如果筷子已在他人手上，则该哲学家必须等待到他人吃完之后才能拿到筷子。
 3. 任一哲学家在自己未拿到两支筷子吃饭之前，决不放下自己手中的筷子。
- 试用信号量解决该哲学家用餐问题，**要避免死锁问题**

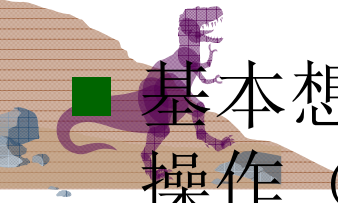




■ 基本想法1: 最多允许四个哲学家同时进餐,以保证至少有一个哲学家能够进餐,最终总会释放出他所使用过的两支筷子,从而可使更多的哲学家进餐。

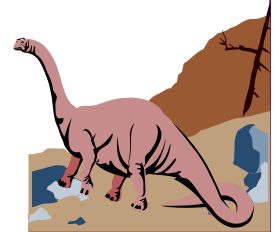
```
semaphore chopstick[5]={1, 1, 1, 1, 1};
semaphore room=4;
void philosopher(int i)
{
    do {
        think();
        wait(room); //请求进入房间进餐
        wait(chopstick[i]); //请求左手边的筷子
        wait(chopstick[(i+1)%5]); //请求右手边的筷子
        eat();
        signal(chopstick[(i+1)%5]); //释放右手边的筷子
        signal(chopstick[i]); //释放左手边的筷子
        signal(room); //退出房间释放信号量room
    } while(true);
}
```

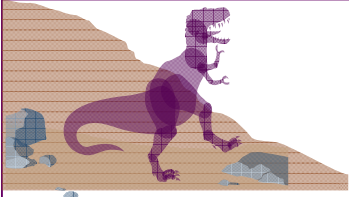




■ **基本想法2：** 将拿左筷子，与拿右筷子当做一个原子操作（即只有当左右筷子均可以拿到时，才拿筷子）

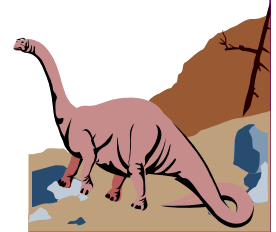
```
semaphore mutex = 1 ;  
semaphore chopstick[5]={1, 1, 1, 1, 1};  
void philosopher(int i)  
{  
    do {  
        think();  
        wait(mutex);  
        wait(chopstick[(i+1)]%5);  
        wait(chopstick[i]);  
        signal(mutex);  
        eat();  
        signal(chopstick[(i+1)]%5);  
        signal(chopstick[i]);  
    } while(true);  
}
```

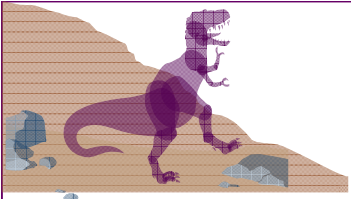




基本想法3:
规定奇数号的
哲学家先拿起
他左边的筷子
，然后再去拿
他右边的筷子;
而偶数号的哲
学家则相反

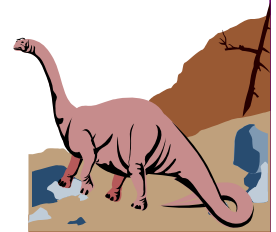
```
semaphore chopstick[5]={1, 1, 1, 1, 1};
void philosopher(int i)
{
    do {
        think();
        if(i%2 == 0) {    //偶数哲学家，先右后左。
            wait (chopstick[(i+1)%5]);
            wait (chopstick[i]);
            eat();
            signal (chopstick[(i+1)%5]);
            signal (chopstick[i]);
        } else {        //奇数哲学家，先左后右。
            wait (chopstick[i]);
            wait (chopstick[(i+1)%5]);
            eat();
            signal (chopstick[i]);
            signal (chopstick[(i+1)%5]);
        }
    } while(true);
}
```

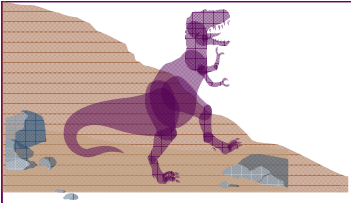




进程同步与互斥：习题2

- 假设有两个生产者进程A、B和一个销售者进程C，他们共享一个无限大的仓库。
 - ◆ 生产者每次循环生产一个产品，然后入库供销售。
 - ◆ 销售者每次循环从仓库中取出一个产品进行销售。
 - ◆ 不允许同时入库，也不允许边入库边出库。
 - ◆ 要求生产A产品和B产品的件数满足以下关系：
- $n \leq A \text{的件数} - B \text{的件数} \leq m$ ，其中n、m是正整数
 - ◆ 要求销售A产品和B产品的件数满足以下关系：
- $n \leq A \text{的件数} - B \text{的件数} \leq m$ ，其中n、m是正整数
- 试用信号量实现这一同步问题。

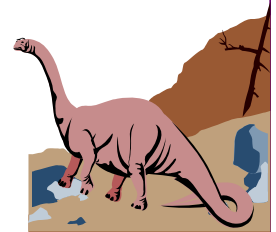


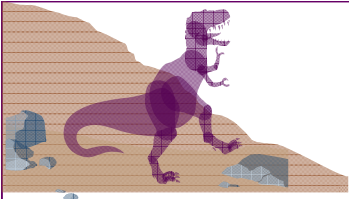


进程同步与互斥：习题2

■ 分析：本题中存在着以下的同步和互斥关系：

- ◆ 生产者A、B和消费者之间不能同时将产品入库和出库，故仓库是一个临界资源；
- ◆ 两个生产者之间必须进行同步，当生产的A、B产品的件数之差大于等于 m 时，生产者A必须等待；小于等于 $-n$ 时，生产者B必须等待；

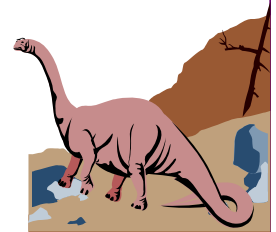


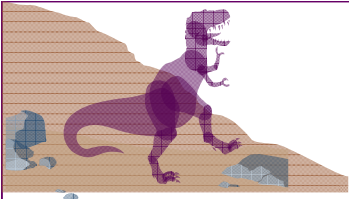


进程同步与互斥：习题2

■ 生产者和销售者之间也必须进行同步

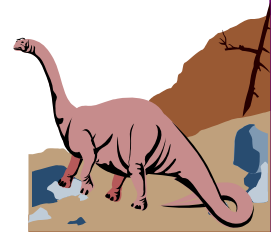
- ◆ 只有当生产者生产出产品并入库后，销售者才能进行销售；
- ◆ 由于销售的产品件数必须满足关系 $-n \leq A \text{的件数} - B \text{的件数} \leq m$ ，
因此当销售的A、B产品件数之差大于等于m而仓库中已无B产品时，或者销售的A、B产品的件数之差小于等于-n而仓库中已无A产品时，销售者C必须等待

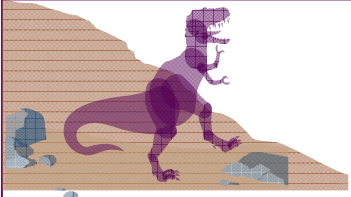




进程同步与互斥：习题2

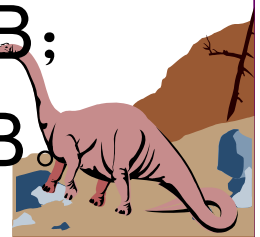
- 为了互斥地入库和出库，为仓库设置互斥信号量**mutex**，其初值为1。
- 由于仓库无限大，入库只要操作互斥就可以完成，无须考虑仓库是否有空位。
- 但出库要考虑有无产品，因此需要设置两个资源信号量，其中
 - ◆ **SA**对应于仓库中的**A**产品量，
 - ◆ **SB**对应于仓库中的**B**产品量。
- 它们的初值都为0.

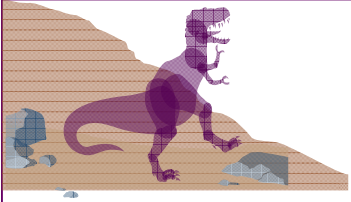




进程同步与互斥：习题2

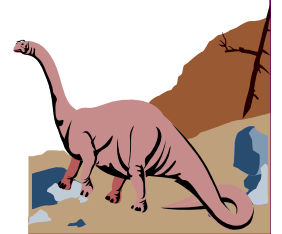
- 为了使生产的产品件数满足 $-n \leq A \text{的件数} - B \text{的件数} \leq m$ ，须设置两个同步的信号量，其中
 - ◆ S_{AB} 表示当前允许 A 生产的产品数量，初值为 m ，
 - ◆ S_{BA} 表示当前允许 B 生产的产品数量，初值为 n ；
- 为了让销售满足： $-n \leq A \text{的件数} - B \text{的件数} \leq m$ ，用 **difference** 变量表示所销售的 A 、 B 产品数量之差，即 $\text{difference} = A \text{的件数} - B \text{的件数}$ ；
 - ◆ 当 $\text{difference} == -n$ 时，不能取产品 B ，只能取 A ；
 - ◆ 当 $\text{difference} == m$ 时，不能取产品 A ，只能取 B ；
 - ◆ 当 $-n < \text{difference} < m$ 时，既可以取 A 也可以取 B 。

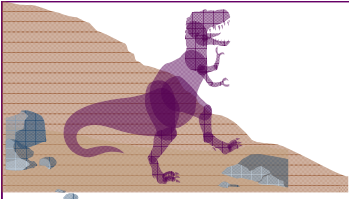




进程同步与互斥：习题2

■ 算法

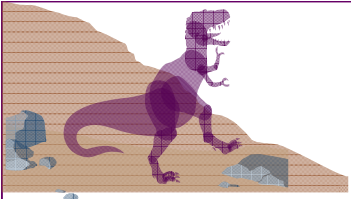




进程同步与互斥：习题3

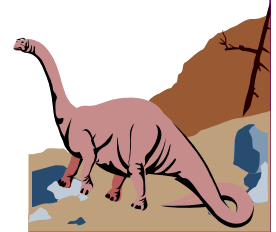
- 考虑三个吸烟者进程和一个经销商进程的系统
 - ◆ 每个吸烟者连续不断地做烟卷并抽他做好的烟卷，做一支烟卷需要烟草、纸和火柴三种原料。
 - ◆ 这三个吸烟者分别掌握有烟草、纸和火柴。
 - ◆ 经销商源源不断地提供上述三种原料，但他只随机的将其中的两种原料放在桌上，具有另一种原料的吸烟者就可以做烟卷并抽烟，且在做完后给经销商发信号，然后经销商再拿出两种原料放在桌上，如此反复。
- 基于信号量设计一个同步算法描述他们的活动

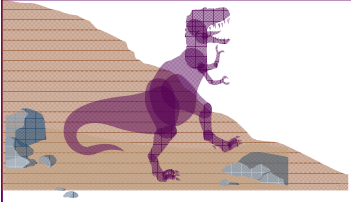




进程同步与互斥：习题3

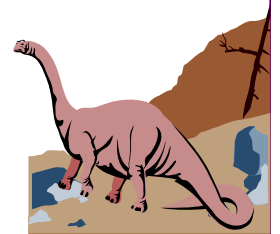
- 可以考虑：设置三个信号量SA、SB和SC，分别代表三种原料组合，初值均为0，即
 - ◆ SA表示烟草和纸，
 - ◆ SB表示纸和火柴，
 - ◆ SC表示烟草和火柴，
- 桌面上一次只能放一种组合，可以看作是放一个产品的缓冲区，设置信号量SD初值为1，控制经销商往桌子上放原料；

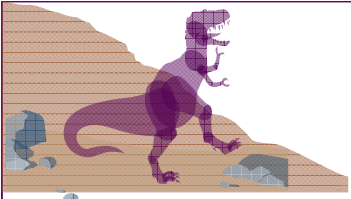




进程同步与互斥：习题3

■ 算法

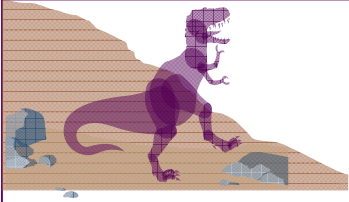




进程同步与互斥：习题4

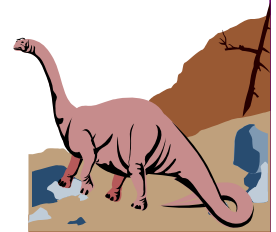
- 现有4个进程R1，R2，W1，W2，它们共享可以存放一个数的缓冲器B。
 - ◆ 进程R1每次把从键盘上输入的一个数存放到缓冲器B中，供进程W1打印输出；
 - ◆ 进程R2每次从磁盘上读一个数放到缓冲器B中，供进程W2打印输出。
 - ◆ 当一个进程把数据存放到缓冲器B后，在该数还没有被打印输出之前不准任何进程再向缓冲器中存数
 - ◆ 在缓冲器B中还没有存入一个新的数之前不允许任何进程加快从缓冲区中取出打印。
- 怎样才能使这四个进程并发执行时协调工作？

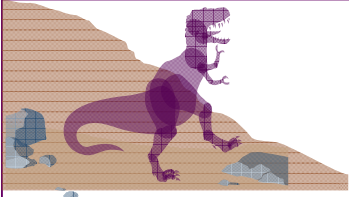




进程同步与互斥：习题4

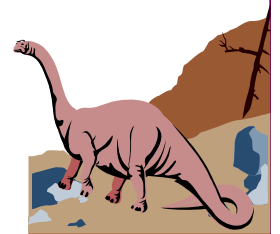
- 这四个进程实际上是
 - ◆ 两个生产者 R1, R2, 和
 - ◆ 两个消费者 W1, W2,
- 各自生成不同的产品给各自的消费对象。
- 他们共享一个的缓冲器。由于缓冲器只能存放一个数，所以，R1和R2在存放数时必须互斥。而R1和W1、R2和W2之间存在同步。

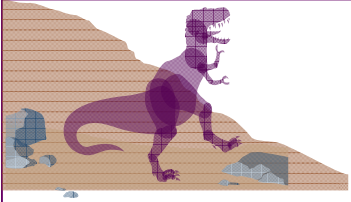




进程同步与互斥：习题4

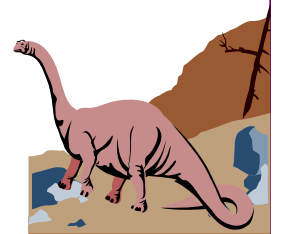
- 为了协调它们的工作可定义三个信号量：
- **S**：表示能否把数存入缓冲器**B**，初始值为1.
- **S1**：表示**R1**是否已向缓冲器存入从键盘上读入的一个数，初始值为0.
- **S2**：表示**R2**是否已向缓冲器存入从磁盘上读入的一个数，初始值为0.





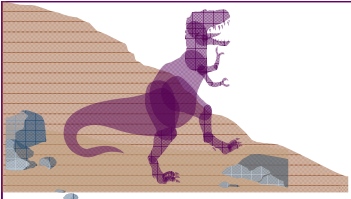
进程同步与互斥：习题4

■ 算法



随堂练习

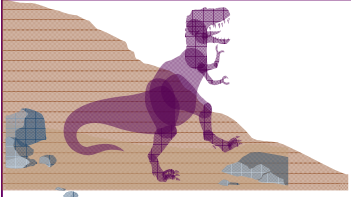
- 如果有三个进程R、W1、W2共享一个缓冲器B，而B中每次只能存放一个数。
- 当缓冲器中无数时，进程R可以将从输入设备上读入的数存放在缓冲器中。只有等缓冲区中的数被取出后，进程R才能再存放下一个数
 - ◆ 若存放在缓冲器中的是奇数，则允许进程W1将其取出打印；
 - ◆ 若存放在缓冲器中的是偶数，则允许进程W2将其取出打印。
 - ◆ 进程W1或W2对每次存入缓冲器的数只能打印一次；W1和W2都不能从空缓冲中取数。
- 写出这三个并发进程能正确工作的程序。



进程同步与互斥：习题5

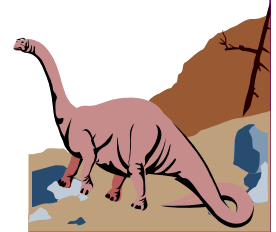
- **a, b**两点之间是一段东西向的单行车道，现要设计一个自动管理系统，管理规则如下：
 - ◆ 当**ab**之间有车辆在行驶时，同方向的车可以同时驶入**ab**段，但另一方向的车必须在**ab**段外等待；
 - ◆ 当**ab**之间无车辆在行驶时，到达**a**点(或**b**点)的车辆可以进入**ab**段，但不能从**a**点和**b**点同时驶入，当某方向在**ab**段行驶的车辆驶出了**ab**段且暂无车辆进入**ab**段时，应让另一方向等待的车辆进入**ab**段行驶。
- 请用信号量机制为工具，对**ab**段实现正确管理以保证行驶安全。

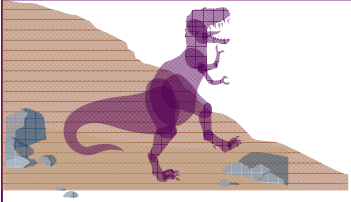




进程同步与互斥：习题5

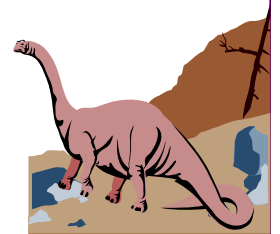
- 这是读者—写者问题的一种变形。
- 我们设置两个共享变量和三个互斥信号量
 - ◆ 变量**ab**记录当前**ab**段上由**a**点进入的车辆的数量，
 - ◆ **S1**用于从**a**点进入的车互斥访问共享变量**ab**，
 - ◆ 变量**ba**记录当前**ab**段上由**b**点进入的车辆的数量，
 - ◆ **S2**用于从**b**点进入的车互斥访问共享变量**ba**，
 - ◆ **Sab** 用于**a**、**b**点的车辆互斥进入**ab**段。
- 两个共享变量**ab**和**ba**的初值分别为0、0。
- 三个信号量的初值分别为1、1和1。

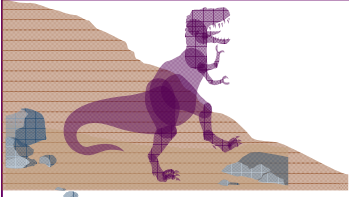




进程同步与互斥：习题5

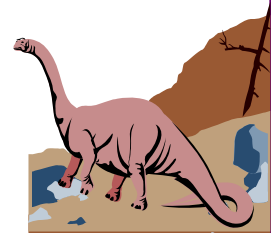
■ 算法

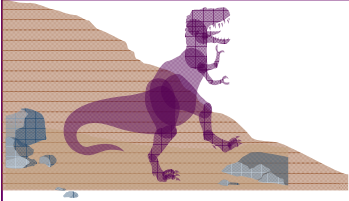




进程同步与互斥：习题6

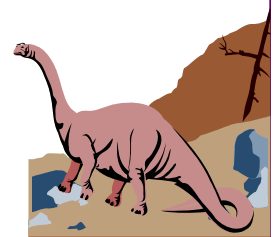
- 和尚挑水问题：寺庙里有多多个小、老和尚，一水缸，水缸容积**10**桶水。
- 小和尚取水放入水缸，老和尚从水缸取水饮用
- 水取自同一水井，水井每次只容一个桶取水。
- 桶总数**3**个，每次倒水入水缸水、或者从水缸中取水都仅为一桶。
- 试用信号量描述和尚取水、饮水的互斥与同步过程。

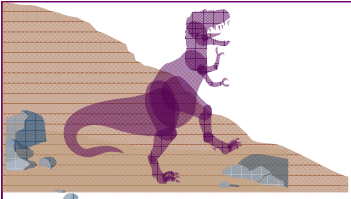




进程同步与互斥：习题6

- Semaphore mutex1=mutex2=1;
// 分别代表水井和水缸
- Semaphore empty=10; // 水缸的入水量
- Semaphore full=0; // 水缸的取水量
- Semaphore count=3; // 水桶个数





老和尚取水:

begin

wait(full)

wait(count)

wait(mutex2)

从水缸取水;

signal(mutex2)

signal(count)

signal(empty)

end

小和尚打水:

begin

wait(empty)

wait(count)

wait(mutex1)

从水井打水;

signal(mutex1)

wait(mutex2)

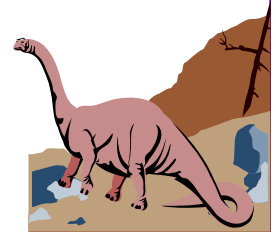
往缸中放水;

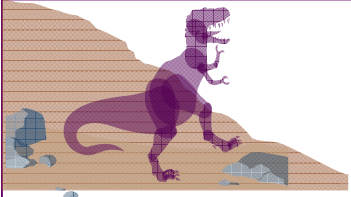
signal(mutex2)

signal(full)

signal(count)

end



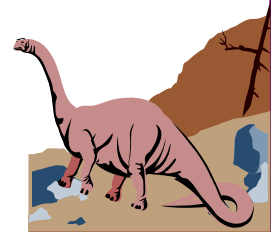


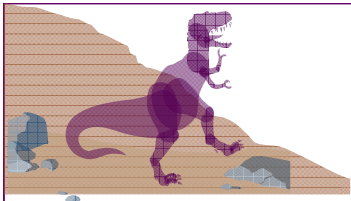
进程同步与互斥：习题7

■ 嗜睡的理发师问题：

- ◆ 一个理发店由一个有 N 张沙发的等候室和一个放有一张理发椅的理发室组成。没有顾客要理发时，理发师便去睡觉。
- ◆ 当一个顾客走进理发店时，如果所有的沙发都已经占用，他便离开理发店；否则，如果理发师正在为其他顾客理发，则该顾客就找一张空沙发坐下等待；如果理发师因无顾客正在睡觉，则由新到的顾客唤醒理发师为其理发。
- ◆ 在理发完成后，顾客必须付费，直到理发师收费后才能离开理发店。

■ 试用信号量实现这一同步问题。



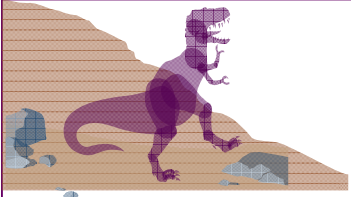


进程同步与互斥：习题7

■ 分析：本题中，顾客进程和理发师进程之间存在着多种同步关系：

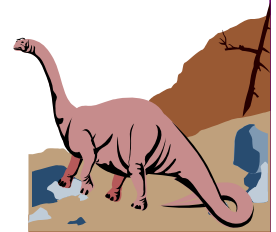
1. 只有在理发椅空闲时，顾客才能做到理发椅上等待理发师理发，否则顾客便必须等待；只有当理发椅上有顾客时，理发师才可以开始理发，否则他也必须等待；这种同步关系类似于单缓冲的生产者-消费者问题中的同步关系，故可通过信号量**empty**和**full**来控制；

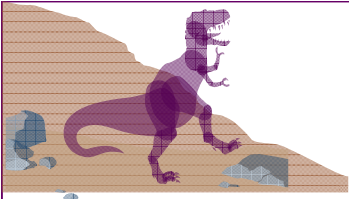




进程同步与互斥：习题7

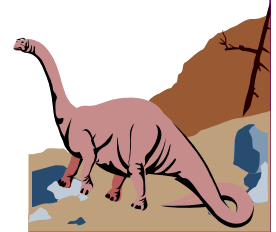
2. 理发师为顾客理发时，顾客必须等待理发的完成，并在理发完成后理发师唤醒他，这可单独使用一个信号量**cut**来控制；
3. 顾客理完发后必须向理发师付费，并等理发师收费后顾客才能离开；而理发师则需等待顾客付费，并在收费后唤醒顾客以允许他离开，这可分别通过两个信号量**payment**和**receipt**来控制。





进程同步与互斥：习题7

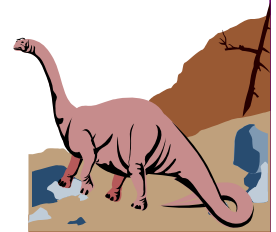
4. 等候室中的N张沙发是顾客竞争的资源，故还需为它们设置了一个资源信号量**sofa**
5. 为了控制顾客的人数，使顾客能够在所有沙发都被占用时离开理发店，还必须设置一个整型变量**count**来对理发店等待的顾客进行计数。该变量将被多个顾客进程互斥地访问并修改，这可通过一个互斥信号量**mutex**来实现。

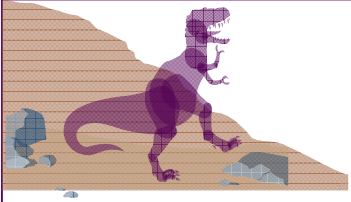




进程同步与互斥：习题7

- 为解决上述问题，需设置一个整型变量**count**用来对理发店重的顾客进行计数，并需设置7个信号量，其中：
 - ◆ **mutex**用来实现顾客进程对**count**变量的互斥访问，其初值为1；
 - ◆ **sofa**对应于等候室中的N张沙发，其初值为N；
 - ◆ **empty**表示是否有空闲的理发椅，其初值为1；
 - ◆ **full**表示理发椅上是否有等待理发的顾客，初值为0；
 - ◆ **cut**用来等待理发的完成，其初值为0；
 - ◆ **payment**表示用来等待付费，其初值为0；
 - ◆ **receipt**用来等待收费，其初值为0。





进程同步与互斥：习题7

■ 算法：

