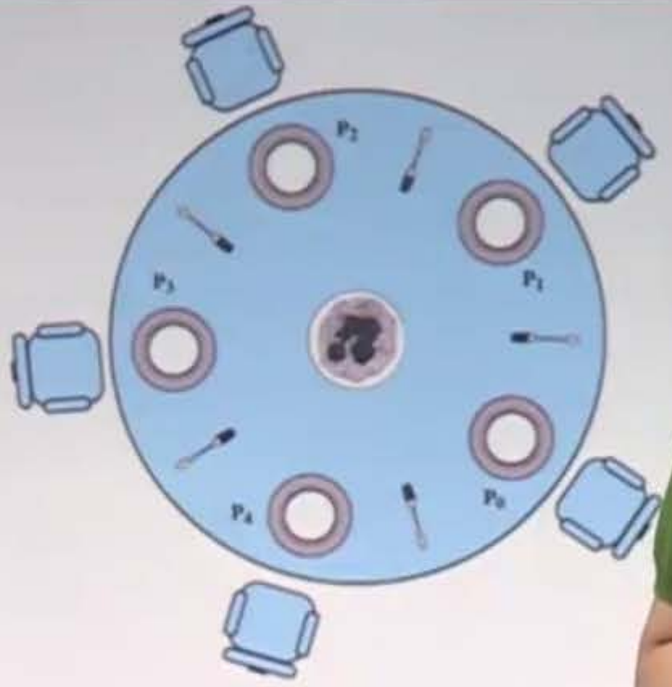
经典的哲学家就餐问题

问题描述：

- 有五个哲学家围坐在一圆桌旁，桌中央有一盘通心粉，每人面前有一只空盘子，每两人之间放一只筷子
- 每个哲学家的行为是思考，感到饥饿，然后吃通心粉
- 为了吃通心粉，每个哲学家必须拿到两只筷子，并且每个人只能直接从自己的左边或右边去取筷子（筷子的互斥使用、不能出现死锁现象）



问题模型：
应用程序中并发线程执行时，协调处理共享资源

哲学家就餐问题第一种解决方案

```
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
      while (true) {
            think();
            P (fork[i]);
            P (fork [(i+1) mod 5]);
            eat();
            V (fork [(i+1) mod 5]);
            V (fork[i]);
}
}
void main()
{
     parbegin (philosopher (0), philosopher (1),philosopher (2),
philosopher (3),philosopher (4));
}
```

产生死锁？

为防止死锁发生可采取的措施

- 最多允许4个哲学家同时坐在桌子周围
- 仅当一个哲学家左右两边的筷子都可用时，才允许他拿筷子
- 给所有哲学家编号，奇数号的哲学家必须首先拿左边的筷子，偶数号的哲学家则反之
- …… ……

当然还有一些手段和方法，大家可以去 考虑一下。

哲学家就餐问题第二种解决方案

```
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
        while (true) {
                think();
                P (room);
                P (fork[i]);
                P (fork [(i+1) mod 5]);
                eat();
                V (fork [(i+1) mod 5]);
                V (fork[i]);
                V (room);
}
}
void main()
{
        parbegin ( philosopher (0), philosopher (1),philosopher (2),
philosopher(3),philosopher (4) );
}
```

最多允许4个
哲学家同时坐
在桌子周围

哲学家就餐问题第三种解决方案

使用管程解决哲学家就餐问题

```
void philosopher[k=0 to 4]
/* the five philosopher clients */
{
while (true) {
      <think>;
      get_forks(k);          /* client requests two forks via monitor */
      <eat spaghetti>;
      release_forks(k);  /* client releases forks via the monitor */
}
}
```

所以他一次通过管程就拿到两只筷子，当然了用完了筷子就把它还回去，也是通过管程

哲学家就餐问题第三种解决方案

```
monitor dining_controller;
cond ForkReady[5];
boolean fork[5] = {true};
void get_forks(int pid)
{
int left = pid;
int right = (++pid) % 5;
/*grant the left fork*/
if (!fork(left)
    cwait(ForkReady[left]);
 /* queue on condition variable */
fork(left) = false;
/*grant the right fork*/
if (!fork(right)
    cwait(ForkReady(right);
/* queue on condition variable */
fork(right) = false:
}
```

```
void release_forks(int pid)
{
int left = pid;
int right = (++pid) % 5;
/*release the left fork*/
if (empty(ForkReady[left])
/*no one is waiting for this fork */
    fork(left) = true;
else /* awaken a process waiting on
fork */
    csignal(ForkReady[left]);
/*release the right fork*/
if (empty(ForkReady[right])
/*no one is waiting for this fork */
    fork(right) = true;
else /* awaken a process waiting on thi
fork */
    csignal(ForkReady[right]);
}
```

哲学家就餐问题第四种解决方案(1)

```
#define  N  5
#define  THINKING  0
#define  HUNGRY    1
#define  EATING    2
#typedef  int semaphore;
int state[N];
semaphore mutex=1;
semaphore s[N];
```

为了避免死锁，把哲学家分为三种状态，思考，饥饿，进食，并且一次拿到两只筷子，否则不拿

好，那么也是规 定要么一次拿到两只筷子，要么就不拿，我们来看一下它的 实现。

哲学家就餐问题第四种解决方案(2)

```
void  philosopher (int i)
{  while (true)
   {
    思考;
    P(&mutex);
    state[i] = HUNGRY;
    test(i);
    V(&mutex);
    P(&s[i]);
    拿左筷子;
    拿右筷子;
    进食;
    放左筷子;
    放右筷子;
    P(&mutex)
    state[ i ] = THINKING;
    test([i-1] % 5);
    test([i+1] % 5);
    V(&mutex);
   }
}
    state[ i ] = THINKING;
    s[ i ] = 0;
```

而测试的结果呢可以导致这个 P 操作的进行，顺利进行

哲学家就餐问题第四种解决方案(3)

```
void test(int i)
{
    if (state[ i ] == HUNGRY)
        && (state [(i-1) % 5] != EATING)
        && (state [(i+1) % 5] != EATING)
        {
            state[ i ] = EATING;
            V(&s[ i ]);
        }
}
```

那么这个时候就做了一个，大家可以看到做了一个 V 操作 而这个

哲学家就餐问题讨论

- 何时发生死锁？
- 怎样从死锁中恢复？
- 怎样避免死锁的发生？
- 如何预防死锁？

所以我们针对这样四个问题，给出了讨论

本讲重点

- 掌握死锁的基本概念
  - 理解产生死锁的四个必要条件
  - 理解死锁、活锁、“饥饿”的区别
- 掌握死锁的解决方案
  - 死锁预防
    资源的有序分配法
  - 死锁避免
    银行家算法、安全/不安全状态
  - 死锁检测与解除
- 理解资源分配图及在解决死锁问题上的应用
- 理解哲学家就餐问题，掌握解决哲学家就餐问题的各种方法

本周要求

- 重点阅读教材

  第6章相关内容

- 重点概念

死锁 活锁 饥饿
资源分配图
死锁预防
死锁避免
死锁检测与解除

0:00 下面我们以哲学家就餐问题来总结归纳解决死锁问题的各种方法 哲学家就餐问题是一个经典的进程之间的同步互斥问题 它的问题描述是这样的 有五个哲学家围坐在一个圆桌旁 桌子中间呢有一盘通心粉，每个人面前呢有一只空盘子 而两个哲学家之间呢放了一只筷子 哲学家的日常行为呢，是思考问题 当他感到饥饿的时候呢就去吃通心粉 为了吃到通心粉呢，我们规定每个哲学家必须拿到两只筷子 而且只能够从他左边或右边来取筷子 在解决哲学家就餐问题的时候，我们考虑到筷子呢是互斥使用的 同时呢我们不能够出现死锁现象 那么哲学家就餐问题 是一个经典的同步互斥问题，它的问题模型是这样的 在应用程序当中，一些并发的线程执行的时候，它们 怎么去协调对资源的使用，特别是对共享资源的使用 我们先来看一下哲学家就餐问题的第一种解决方案 这种解决方案呢，是把筷子当做一个信号量来处理 因此，要去拿筷子的时候，就去 对信号量进行 P 操作，相当于申请一只筷子 我们来看一下，它申请右边这只筷子，再申请左边这只筷子 那么在什么情况下 会出现死锁呢？也就是说当 每个哲学家都拿到了右边这只筷子 都等在拿左边这只筷子的时候 就会出现死锁。 我们看一下这里头 当他拿到一只筷子之后，他被切换，下 CPU 了。 然后另外一个哲学家 拿到了他右边的筷子，那么如果这个非常的 巧，那么每个哲学家都刚好拿到了右边这只筷子 那么又等待左边这只筷子，那么这个系统，这个哲学家就无限等待下去了 这就是第一种问题的解法 那么为了防止死锁的发生，可能采取各种各样的措施 我们现在给出典型的几个方法 第一个方法呢是最多允许 4 个哲学家同时坐在桌子旁边 也就是当哲学家去思考问题的时候，他可以在四处溜达 当他感到饥饿的时候呢，他再坐在桌子旁边 所以这就是第一种方法 第二种方法呢，是说仅当一个哲学家左右两边的筷子都 可以使用的时候呢，才允许他拿筷子，这样他会一次拿到两只筷子 第三种情况呢，就是我们给哲学家编个号 然后我们规定奇数号的哲学家必须首先拿左边的筷子 偶数号的哲学家他要先拿右边的筷子 那么如果这样去拿筷子可能就不会出现死锁问题了 当然还有一些手段和方法，大家可以去 考虑一下。 那么现在我们来看看用 允许，最多允许 4 个哲学家坐在 桌子旁边为例，我们看看怎么样来解决死锁问题 那在这里头呢，我们来看一下 增加了一个新的信号量就是 room 这个 room 的最大值是 4，因此当哲学家饥饿了想要吃的时候呢，他要先坐到桌子旁边 那么如果我们用 P(room) 来表示他能不能坐到桌子旁边，那么这样的话呢 最多允许 4 个哲学家坐到桌子旁边，就不会出现死锁问题了 这个也就是我们应

该是初中学到的这种鸽巢原理 或者叫做抽屉法则。 因为在 4 个哲学家坐到桌子旁边 有 5 只筷子，所以肯定有一个哲学家会拿到两只筷子而去吃 这就是第二种解决方案。 我们还看一下管程 怎么样来用于解决哲学家就餐问题，因为我们刚才说了，说 允许一个哲学家当他两只筷子都能拿到的时候，才允许他去拿筷子 那我们实际上是可以通过管程来解决这个问题。 我们设定了 管程之后，那么每个哲学家去取筷子的时候呢，实际上就是 有这么一个取筷子的操作，而这个操作的实现呢是在管程里头 所以他一次通过管程就拿到两只筷子，当然了用完了筷子就把它还回去，也是通过管程 简单看一下这个实现，我们 定义了这样一个管程之后，我们来看一下这个取筷子的操作 取筷子操作就是说他先拿，比如说左边的筷子，拿到了左边的筷子之后呢，再拿右边的筷子- ，所以他 如果中间拿不到两只筷子，他就在管程里等待就可以了 所以呢，这样的话我们就说管程的问题呢实际上就是 当两个，两只筷子都可用的时候呢，才让他去拿，所以他一次拿到两只筷子 我们再看一种 哲学家就餐问题的一种解法，这种解法呢，实际上是为了避免死锁 把哲学家的行为就分成了三种状态 第一种状态呢是思考，第二种状态呢它是一个饥饿状态 第三种状态呢就是一个进食的状态。 好，那么也是规 定要么一次拿到两只筷子，要么就不拿，我们来看一下它的 实现。 我们来看一下这个 代码，每个哲学家呢他的行为呢首先是思考 然后他可能会感到饥饿，因此呢他需要一个状态来记录他现在的状态是饥饿状态 在他之前呢是个思考状态。 那么为了记录这个状态呢，我们需要做一个 P 操作，因为 这个状态可以被多个哲学家进行修改，那么我们是要看看就是说互斥，互斥 然后做完了这个状态的改变之后，他就要 判断是不是同时能拿到两只筷子，所以做一个测试 如果测试的结果通过了，他就可以同时拿到两只筷子，他就去做拿筷子的操作 这里头呢，能不能通过测试呢，是要通过这个 P 操作来决定 如果拿到了两只筷子 就开始去进食，然后再去把筷子放回去 然后再去改变状态，依然是注意要用一个 互斥保护这个临界区。 好，我们再来看一下这每个 哲学家都要去判断一下能不能拿到两只筷子，就是测试 而测试的结果呢可以导致这个 P 操作的进行，顺利进行 我们来看一下测试这段代码 测试呢传递进来的参数呢，实际上是某个 哲学家的编号，比如说第 2 号哲学家 要来测试一下，那么 2 号哲学家来测试首先看一下 他的状态首先是一个饥饿状态， HUNGRY 同时，他的右边 因为他的右边是 1 号哲学家，也就是 1 号哲学家 并没有在吃还是在吃，如果 他的状态是 EATING ，那么这个条件就不成立 那么如果他没有在吃，右边的 1 号哲学家没有 在吃，同时左边的 3 号 哲学家也没有在

吃 因此我们可以看到 2 号哲学家想进食，同时 1 号和 3 号哲学家都没有在进食状态 那么 2 号哲学家就可以拿到两只筷子，就可以去进食，因此 就把 2 号哲学家的状态改变成为 EATIING 状态 那么这个时候就做了一个，大家可以看到做了一个 V 操作 而这个 V 操作呢，实际上是我们刚才说 当 2 号哲学家调用了 test 这个操作之后 那么返回以后接着往下执行，到了这儿能不能通过这个检查 这样的话如果能通过检查，就是刚才你做了 V 操作，那么这个 执行的 P 操作就可以通过了。 如果 如果，我们看一下， 2 号哲学家 做检查的时候，假设 1 号哲学家正在就餐 那么也就说这个条件不成立了，所以就没有做这个 V 操作 那么他回来走到这个地方的时候，他就会被阻塞在这个 P 操作的位置，就不能够继续去拿筷子进食 那么这是对于一个哲学家他要想就餐的时候，他要做这样一个测试 我们再来看一下，假设 一个哲学家他还回了筷子之后 他的又变成了重新思考的状态 那么他还要做两件事情，因为他拿到筷子的时候 可能有左右两边的科学家都来想拿到这只筷子 但是由于他 正在进食，所以左右两边的某个哲学家可能 拿不到筷子之后呢，就处于等待状态，等待在他 那个哲学家所对应的这个信号量上。 而 当这个哲学家结束了这个进食 重新变成思考之后呢，那么他还要有一个 测试，看一看是不是要把左右两边的哲学家来 释放，或者是归还的这个筷子可以分配给他们 因此在这又做了两个测试，这是对，假定以 2 号哲学家为例，这是对 1 号哲学家做测试 同时呢，以对 3 号哲学家做测试 所以当 2 号哲学家吃完了，变成 思考状态的时候呢，对 1 号和 3 号哲学家做测试，我们假设以 3 号哲学家为例 假设 3 号哲学家刚才，当然这个 i 就是 3 了，对吧，刚才 3 号哲学家呢是 饥饿状态，但是 2 号哲学家他正在 吃，所以这个条件不满足，因此 3 号哲学家就 结束了测试之后呢就等在 这个位置，就是 3 号哲学家这个位置 那么 2 号哲学家去做了一个测试之后 实际上呢，通过了这样一段代码 我们知道 3 号哲学家的满足条件了就可以 执行这个 V 操作了。 一旦执行这个 V 操作，就把刚才等在 那的 3 号哲学家就把他释放了，把他重新编程就绪，让他可以上 CPU 了 那么这就是哲学家就餐问题的这样一种解决方案 大家回去呢仔细地去看一下这个代码，去理解这样一个关系 下面呢我们对哲学家就餐问题呢进行一些讨论 我们看一看能不能回答这样几个问题 第一个问题，什么时候发生死锁？ 我们根据刚才的分析，我们看到 当每个哲学家都拿到了右边这只筷子 那么这个时候都在等左边这只筷子，那么这就是说发生了死锁 第二个问题，发生了死锁之后 怎么从死锁中恢复或者是怎

么去解除死锁？ 根据我们刚才所讨论的各种解除死锁的方法 当中，我们选择一种就是某个哲学家放下一只筷子 那他拿到了一只，把那只放下，那这样的话呢就 解除了死锁。 第三个是 如果我们想说采取了死锁避免这种方法来解决死锁问题 在哲学家就餐问题里头怎么样来避免死锁的发生呢？ 我们来琢磨一下，什么情况下 会发生死锁呢？比如说第一个哲学家来了之后，他拿到了一只筷子 然后他被切换下 CPU。 第二个哲学家也拿到了一只筷子，正好也被切换下去。 那么到了第 5 个哲学家来的时候，系统中还剩下最后一只筷子 而如果第 5 个哲学家拿到这只筷子，系统就出现死锁了 所以我们知道根据银行家算法 那么最后这只筷子只能够分配给 手里拿到一只筷子的某个哲学家 那么这就是死锁避免这个 解决的方法呢，在哲学家问题上的一个应用。 就是我们的银行家算法 用在这呢，实际上就是说如果系统中还剩下最后一只筷子，那么这只筷子一定 只能分配给手里拿到一只筷子 的哲学家，而不能你手里头什么都没拿到，你要这只筷子给你 不行，这就是一个避免的思想 那么如何预防死锁发生呢？ 也就是在资源分配算法的时候我们怎么设计然后让死锁不会发生。 前面我们已经介绍了说 你只能同时拿到两只筷子，你才让你去拿，这就是一种一次性分配的思想 那么我们也可以采用刚才提出的一种方案 就是说我们采用资源的有序分配法 我们给每个哲学家编个号，我们给每个筷子编号，我们就要求哲学家 在申请筷子的时候，首先要申请编号小的那只筷子，再申请编号大的那只筷子 那也就是说大部分哲学家都是先拿 右边的再拿左边的，只有最后一个哲学家，比如说 5 号哲学家 他得先拿左边的筷子，再拿右边的筷子，因为他右边筷子 的编号比如说是 5，左边的筷子是编号是 1，所以他是要 按反的方向，先拿左再拿右，所有其他的先拿右再拿左，这样的话就能满足 资源的有序分配法，而采用资源有序分配法，那么系统中不会出现死锁 所以我们针对这样四个问题，给出了讨论 我们本讲的内容呢，主要是要求大家掌握死锁的基本概念 理解产生死锁的四个必要条件 要理解死锁、 活锁还有"饥饿"这样三个概念的区别 重点我们要掌握死锁预防 死锁避免，死锁检测与解除这样三种解决死锁的方案 其中死锁预防当中的重点实际上是资源的有序分配法 而死锁避免呢我们的关键点是安全状态、 安全 序列这样的概念以及银行家算法 我们还介绍了资源分配图，那么大家 理解一下资源分配图在解决死锁问题上是怎么样去应用的 比如说，死锁定理怎么样运用到死锁检测的过程中 最后我们是以哲学家就餐问题 为例来讨论各种解决死锁的方案在这个问题上的应用 死锁问题

呢是 教材中的第 6 章，那么重点的一些概念呢，我们 在这里列出来了，希望大家能够回去好好地复习

本讲的内容呢就介绍到这里，谢谢大家！