

Operating System Principle, OS

2025年秋. 操作系统原理

## 第05章 死锁

课 程 组：邹德清, 李珍, 李志, 苏曙光

企业教师：华为认证专家(鸿蒙方向)

## ● 主要内容

- 资源的概念
- 死锁的概念
- 产生死锁的原因和必要条件
- 解决死锁问题的策略

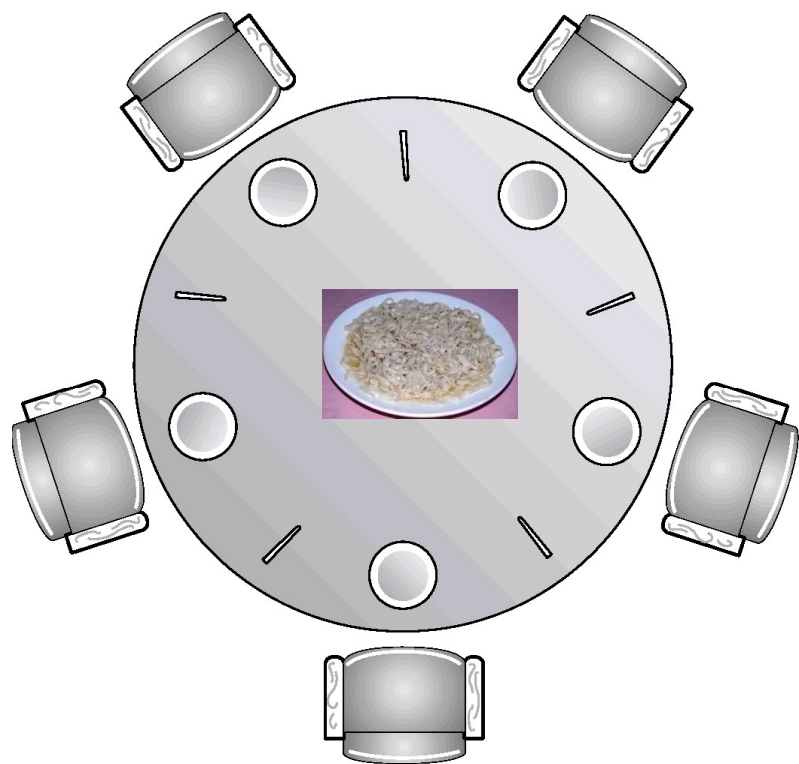
## ● 重点

- 死锁的概念
- 产生死锁的原因和必要条件

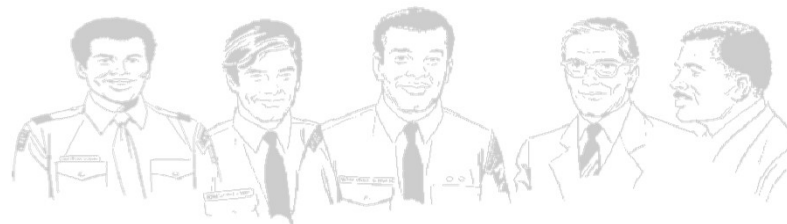
- 三个经典的同步问题
  - 生产者-消费者问题
    - ◆ 同步和互斥混合
  - 读者-写者问题
    - ◆ 互斥问题
  - 哲学家就餐问题

# 哲学家就餐问题

- 五个哲学家围坐圆桌边，桌上有1盘面和5支筷子。哲学家的生活：思考-休息-吃饭-思考-.....



```
UINT Philosopher (int i) // i 哲学家编号
{
    while (TRUE)
    {
        思考;
        休息;
        吃饭; //正用2支筷子
    }
}
```



# 用线程实现哲学家的生活 **Philosopher**

int **S**[5] = { 1, 1, 1, 1, 1 }; // **S**[i]信号量: i号筷子是否可用: 1可用, 0不可用  
// 每个哲学家左手边筷子与该哲学家编号相同。

UINT **Philosopher** (int i) // 线程函数, i是哲学家的编号

{

while (TRUE)

{

思考;

休息;

P(**S**[i]); //取左手边的筷子

P(**S**[(i+4) % 5]); //取右手边的筷子

吃饭; //正用**2**支筷子

V(**S**[(i+4) % 5]); //放下右手的筷子

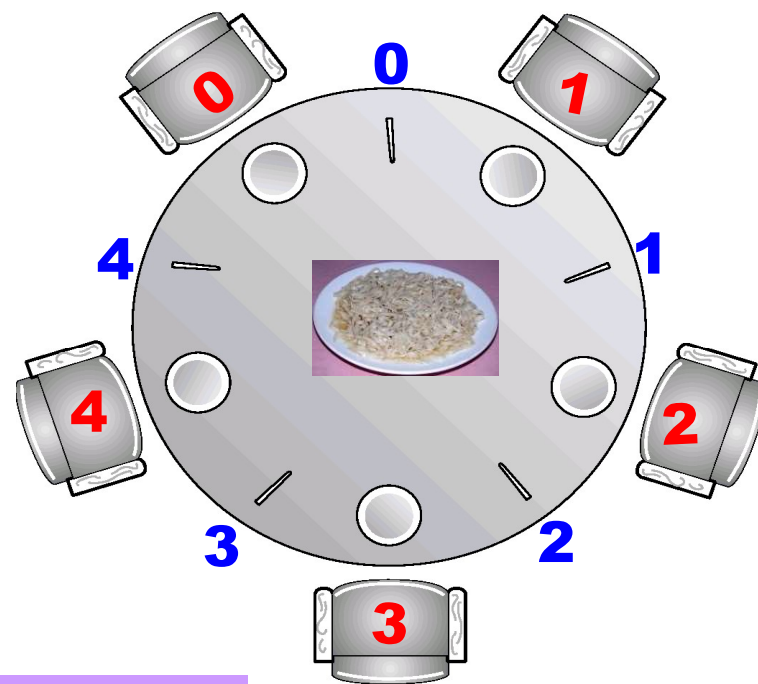
V(**S**[i]); //放下左手的筷子

要求: 用**1**双 | 每次取**1**支 | 身边 | 吃完才放下

思考: 若**5**个线程先后在P(**S**[i])处”就绪”, 结果怎样?

不允许相邻**2**位同时吃饭!

死 锁



就绪

继续

# 死锁

- 主要内容

- 死锁的概念

- 死锁的起因、必要条件和解决策略

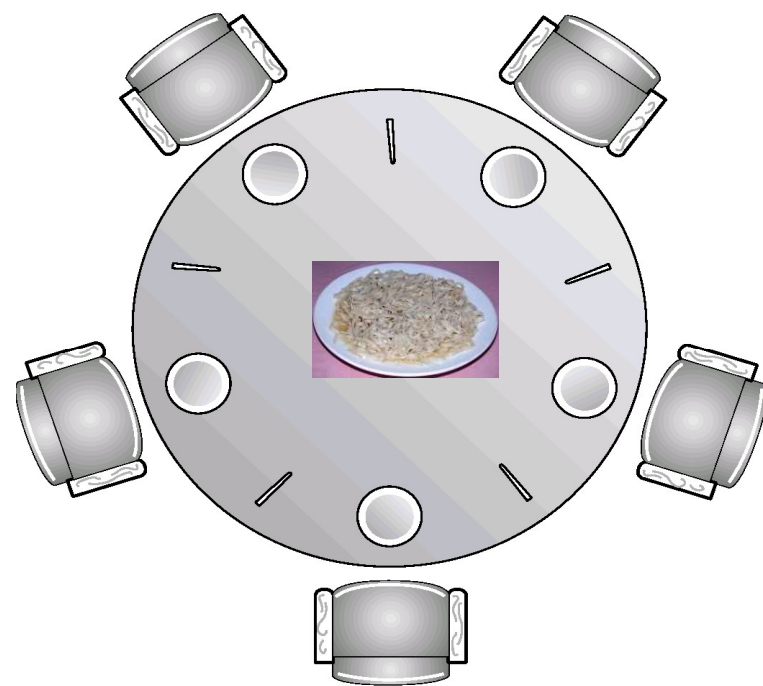


# 死锁

## ● 死锁的定义

- 两个或多个进程无限期地等待永远不会发生的条件的一种系统状态。【结果：每个进程都永远阻塞】

每个哲学家都无限期地等待邻座放下筷子！  
而邻座没有吃完之前不会放下筷子！  
而邻座缺一只筷子永远都无法吃完！



# 死锁

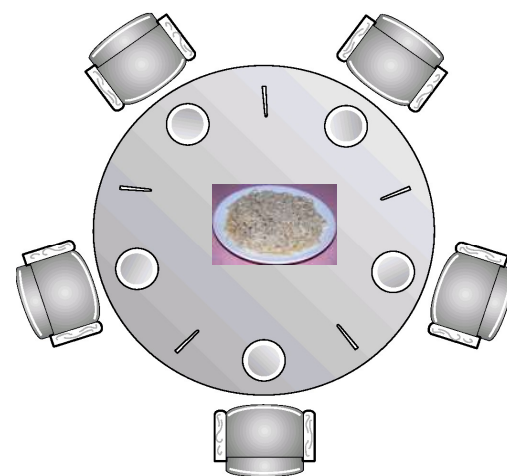
- 死锁的起因

- 系统资源有限

- ◆ 资源数目不足以满足所有进程的需要，引起进程对资源的竞争而产生死锁。

- 并发进程的推进顺序不当

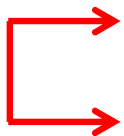
- ◆ 进程在运行过程中，请求和释放资源的顺序不当，导致进程产生死锁。





**int Data =0 ;**     /\* 信号量：缓冲区中的数据个数，初值**0** \*/  
**int Space = 5;**   /\* 信号量：缓冲区中的空位个数，初值**5** \*/  
**int mutex = 1;**   /\* 信号量：缓冲区互斥使用，初值**1**，可用 \*/

```
producer_i ( ) // i = 1 .. m
```

```
{  
    while( TRUE )  
    {  
        生产1个数据 ;  
 P(Space);  
P(mutex);  
        存1个数据到缓冲区;  
V(mutex);  
V(Data) ;  
    }  
}
```

实现：

- 1.不能向**满**缓冲区**存**
- 2.不能从**空**缓冲区**取**
- 3.生产者之间的**互斥**
- 4.消费者之间的**互斥**
- 5.生产者和消费者之间的**互斥**

```
consumer_j ( ) // j = 1 .. k
```

```
{  
    while( TRUE )  
    {  
        P(Data);  
P(mutex);  
        从缓冲区取1个数据;  
V(mutex);  
V(Space);  
        消费一个数据;  
    }  
}
```

**int Data =0 ;**     /\* 信号量：缓冲区中的数据个数，初值**0** \*/  
**int Space = 5;**   /\* 信号量：缓冲区中的空位个数，初值**5** \*/  
**int mutex = 1;**   /\* 信号量：缓冲区互斥使用，初值**1**，可用 \*/

```
producer_i ( ) // i = 1 .. m
```

```
{  
    while( TRUE )  
    {  
        生产1个数据 ;  
        P(mutex);  
        P(Space);  
        存1个数据到缓冲区;  
        V(mutex);  
        V(Data) ;  
    }  
}
```

实现:

- 1.不能向**满**缓冲区**存**
- 2.不能从**空**缓冲区**取**
- 3.生产者之间的**互斥**
- 4.消费者之间的**互斥**
- 5.生产者和消费者之间的**互斥**

```
consumer_j ( ) // j = 1 .. k
```

```
{  
    while( TRUE )  
    {  
        P(Data);  
        P(mutex);  
        从缓冲区取1个数据;  
        V(mutex);  
        V(Space);  
        消费一个数据;  
    }  
}
```

**思考：是否会死锁？  
描述如何进入死锁？**

**int Data =0 ;**    /\* 信号量：缓冲区中的数据个数，初值**0** \*/  
**int Space = 5;**    /\* 信号量：缓冲区中的空位个数，初值**5** \*/  
**int mutex = 1;**    /\* 信号量：缓冲区互斥使用，初值**1**，可用 \*/

**producer\_i ( )** // i = 1 .. m

{ // Data = 5, Space = 0

while( TRUE )

{

生产1个数据；

P(mutex); //mutex=1→0

P(Space);

存1个数据到缓冲区；

V(mutex);

V(Data) ;

}

}

实现：

- 1.不能向**满**缓冲区**存**
- 2.不能从**空**缓冲区**取**
- 3.生产者之间的**互斥**
- 4.消费者之间的**互斥**
- 5.生产者和消费者之间的**互斥**

**死锁!**

**consumer\_j ( )** // j = 1 .. k

{ // Data = 5, Space = 0

while( TRUE )

{

P(Data); // Data = 5→4

P(mutex);

从缓冲区取1个数据；

V(mutex);

V(Space);

消费一个数据；

}

}

# 死锁

- 关于死锁的一些结论

- 陷入死锁的进程至少是2个 【反证：若仅有1个进程死锁...】

- ◆ 两个或以上进程才会出现死锁

- 参与死锁的进程至少有2个已经占有资源【反证：若仅1个或0个占有资源...】

- 参与死锁的所有进程都在等待资源

- 参与死锁的进程是当前系统中所有进程的子集

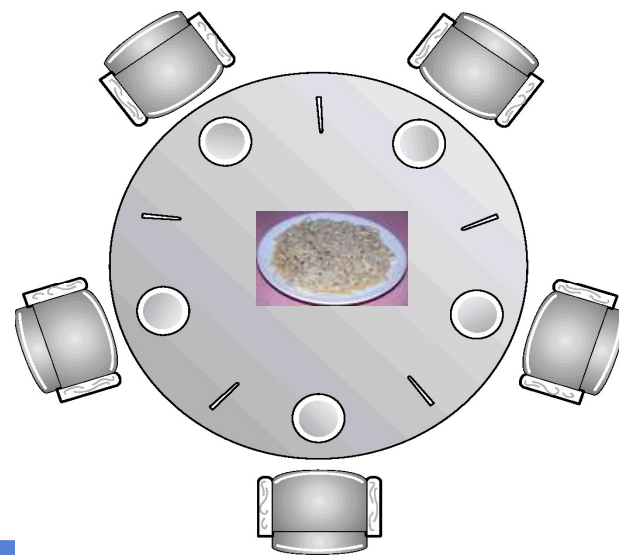
- 死锁会浪费大量系统资源，甚至导致系统崩溃

# 死锁

## ● 死锁的另一个定义

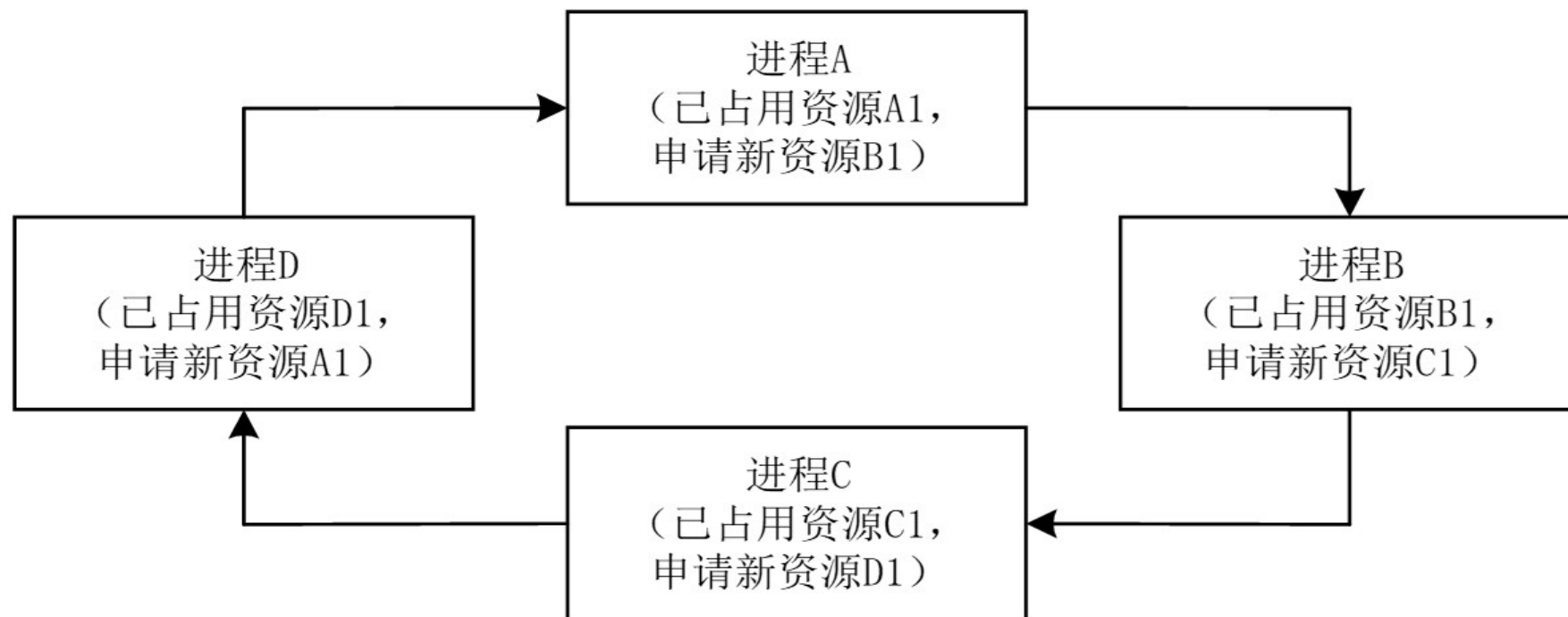
■ 在两个或多个进程中，**每个**进程都**已持有**某种资源，但又**继续申请**其它进程**已持有的某种资源**。

◆ 每个进程都拥有其运行所需的**部分资源**，但又**不足够**运行，从而每个进程都不能向前推进，陷于**阻塞**状态。这种状态称**死锁**。



# 死锁

## ● 死锁的必要条件



## ■ 环路条件

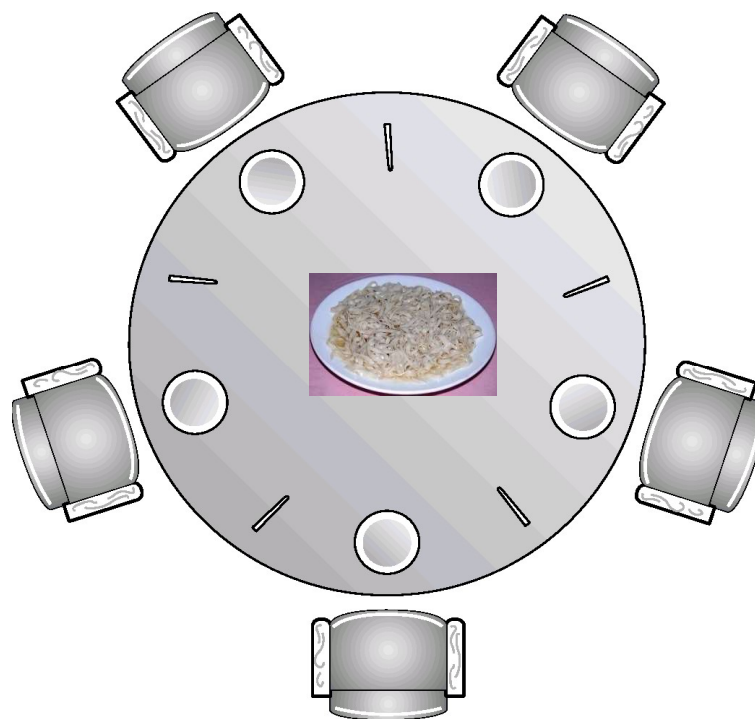
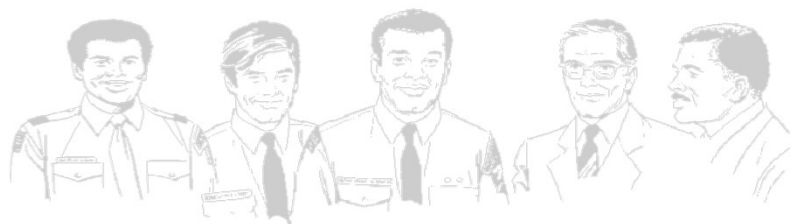
◆ 多个进程构成环路：环中每个进程已占用的资源被前一进程申请，而自己所需新资源又被环中后一进程所占用。



# 死锁

## ● 死锁的必要条件

- 若限定最多4个人同时吃饭，  
则就是破坏环路条件



## ■ 环路条件

- ◆ 多个进程构成环路：环中每个进程已占用的资源被前一进程申请，而自己所需新资源又被环中后一进程所占用。

## 5.3 解决死锁的策略

- 预防死锁
- 避免死锁
- 检测死锁
- 恢复死锁

## 5.3 解决死锁的策略

### ● 预防死锁

■ 通过设置某些限制条件，破坏死锁四个必要条件中的一个或多个，来防止死锁。

◆ 破坏互斥条件 ..... (难)

◆ 破坏不剥夺条件 ..... (代价大)

◆ 破坏部分分配条件 ..... (预先静态分配)

◆ 破坏环路条件 ..... (有序资源分配)

□ 较易实现，(早期)广泛使用。

□ 缺点：由于限制太严格，导致资源利用率和吞吐量降低。



## 5.3 解决死锁的策略

- 避免死锁

- 在资源的分配过程中，用某种方法分析该次分配是否可能导致死锁？若会则不分配；若不会就分配。

- 银行家算法 【不做要求】

- 只需要较弱的限制条件，可获得较高的资源利用率和系统吞吐量。缺点：实现较难。

## 5.3 解决死锁的策略

### ● 检测和恢复死锁

- 允许死锁发生，但可通过**检测机制**及时检测出死锁状态，并精确确定与死锁有关的进程和资源，然后采取适当措施，将系统中已发生的死锁**清除**，将进程从死锁状态解脱出来。

- ◆ 检测方法

- 复杂

- ◆ 恢复方法

- 撤消或挂起一些进程，以回收一些资源。

- 缺点

- ◆ 实现难度大

# 预先静态分配法【MOOC学习】

- 目的

- 破坏部分分配条件

- 策略

- 全部分配法：进程运行前将所需全部资源**一次性**分配给它。因此进程在运行过程中不再提出资源请求，从而避免出现阻塞或者死锁。





# 预先静态分配法【MOOC学习】

## ● 特点/缺点

- **特点：**进程仅当其所需全部资源可用时才开始运行。
- 应用设计和执行开销增大：进程运行前估算资源需求。
- 执行可能被延迟：进程所需资源不能全部满足时。
- 资源利用率低：资源被占而不用。

## ● 改进

- 资源分配的单位由进程改为程序步。

# 有序资源分配法【MOOC学习】

- 目的：破坏环路条件，使得环路无法构成。

- 策略

- 系统中的每个资源分配有一个唯一序号；

- 进程每次申请资源时只能申请序号更大的资源！

- ◆ 如果进程已占有资源的序号最大为M，则下次只能申请序号大于M的资源，而不能再申请序号小于或等于M的资源。

- [如何证明？] 按此规则分配资源系统不会死锁。

- 思考：按此规则，某进程申请资源时，是否一定能马上得到？

- 资源分配策略

- 分配资源时检查资源序号是否符合递增规定

- 若不符合则拒绝（并撤销该进程）

- ▲ 若符合且资源可用则予以分配

- 若符合但资源不可用则不分配，陷于阻塞。

# 死锁是小概率事件

## ● Windows, Linux的死锁解决方案

### ■ 鸵鸟策略

