LEC 6

1. 概念

在死锁 (Deadlock) 问题中，每个哲学家都持有一只餐叉，并且都在请求其右手边的餐叉。当所有哲学家都同时持有左手边的餐叉并且请求右手边的餐叉时，就会发生死锁。

死锁 (Deadlock)

死锁是并发环境下的一种常见问题，它指的是多个进程因为竞争资源而陷入相互等待的状态，导致它们无法向前推进的现象。如果没有外部干涉，所有进程都将无法继续执行。

特征：

- 各进程相互等待对方持有的资源，导致进程阻塞，无法向前推进。
- 可以通过资源分配图来描述进程之间的依赖关系，如果存在环路则表示发生了死锁。

饥饿 (Starvation)

饥饿是指某个进程无法获取到所需的资源，导致无法向前推进的现象。饥饿通常发生在某些进程持续获取资源而其他进程无法获取到资源的情况下。

特征：

- 进程长时间等待某个特定资源，但始终无法获取到。
- 可能导致某些进程一直处于等待状态，无法被处理。

死循环 (Livelock)

死循环是指某个进程在执行过程中一直重复执行某个循环，但始终无法推出的现象。死循环通常是由于程序设计缺陷导致的，例如逻辑错误或算法设计错误。

特征：

- 进程在执行过程中一直重复某个循环，无法正常退出。
- 可能导致系统资源被占用，其他进程无法正常执行。

共同点与不同点

共同点：

- 都是进程无法向前推进的现象，导致系统资源无法被充分利用。

不同点：

1. 死锁是多个进程因竞争资源而相互等待的情况，至少有两个或多个进程同时发生死锁；而饥饿是某个进程无法获取到所需资源而无法向前推进，可能只有一个进程发生饥饿。
2. 死锁的进程处于阻塞状态；饥饿的进程处于等待状态，但不一定是阻塞状态。
3. 死锁是由于资源分配策略不当导致的；饥饿通常是由于资源分配不公平或优先级不当导致的；而死循环是由于程序设计缺陷导致的。

2. 死锁产生必要条件

1. 互斥条件 (Mutual Exclusion)：每个资源只能被一个进程使用，如果资源被一个进程占用，其他进程必须等待释放才能使用。争抢资源的方式会导致死锁。
2. 不可抢占条件 (Hold and Wait)：进程已经持有至少一个资源，并且在等待获取新的资源时不释放已经持有的资源。这意味着即使有资源可以满足某个进程的需求，也有可能无法满足，因为其他进程正在持有所需的资源。
3. 持有和等待条件 (Hold and Wait)：进程已经持有至少一个资源，但又对新请求的资源提出等待，而该资源被其他进程占用。在等待新的资源时，进程不释放已经持有的资源，导致资源无法被其他进程利用。
4. 循环等待条件 (Circular Wait)：存在一个进程资源的循环等待链，链中的每个进程都在等待下一个进程所持有的资源。这意味着存在一个资源循环链，每个进程都在等待下一个进程所持有的资源，最终导致死锁。

值得注意的是，死锁发生时一定存在循环等待链，但循环等待链并不一定会导致死锁。例如，在哲学家就餐问题中，如果每个哲学家都按照一定的规则拿起餐具，那么就不会发生死锁，即使存在循环等待链。因此，循环等待是死锁发生的必要条件，但不是充分条件。如果系统中每个资源都按照一定的规则分配给进程，那么循环等待就不会导致死锁。

3. 死锁发生

1. 竞争系统资源：各个进程争夺系统资源时，当其中一个进程占有资源而不释放时，其他进程无法继续执行，引发死锁。这种资源通常是不可抢占的，如打印机等。
2. 进程推进顺序不当：如果进程在申请和释放资源的顺序上存在问题，也可能导致死锁。例如，两个进程分别获取了资源A和资源B，然后又试图获取对方已经占有的资源，最终导致死锁。
3. 信号量的误用：在使用信号量进行进程同步时，如果不正确地使用信号量，也可能导致死锁。例如，在生产者-消费者问题中，如果生产者在执行 `mutex` 操作之前没有正确判断 `empty` 信号量的值，而直接进入临界区，则可能导致消费者无法访问临界区，从而发生死锁。

总之，死锁通常发生在多个进程之间争夺系统资源的过程中，如果资源的申请和释放顺序不当，或者在使用同步机制时出现错误，都可能导致死锁的发生。因此，在设计并发系统时，需要合理规划资源的分配和释放策略，并确保正确使用同步机制，以避免死锁的发生

4. 死锁处理策略

死锁处理策略

4.1 预防死锁（Deadlock Prevention）静态

在此策略下，系统采取措施来破坏死锁发生的四个必要条件之一，从而防止死锁的发生。这包括：

1. 破坏互斥条件：即对于某些资源，使之不再需要进程互斥地访问。例如，将原本只能独占使用的资源改造成可以共享使用，从而避免了死锁的发生。
2. 破坏不可抢占条件：进程在占有资源时，不允许被其他进程抢占，只能由进程自主释放。这意味着当某个进程请求新资源而得不到满足时，需要主动释放已占有的资源，待以后再重新申请。
3. 破坏持有和等待条件：进程在请求新资源时不阻塞，而是在开始执行之前一次性获取所需的全部资源。这样，就避免了因持有部分资源而等待其他资源导致的死锁。
4. 破坏循环等待条件：通过规定资源的申请顺序，使得所有进程对资源的申请都按照同一个顺序进行，从而避免形成资源的循环等待链。

进程	现有资源	所申请的资源
a	1，3，5，7，12，14	一定大于等于15
b	2，4，6	一定大于等于7

4.2 避免死锁（Deadlock Avoidance）动态

在避免死锁策略中，系统通过安全序列（Safe Sequence）的判断来预防死锁的发生。安全序列指的是如果系统按照这个序列给进程分配资源，那么每个进程都能顺利完成，并且系统不会进入不安全状态。系统可能存在多个安全序列，资源分配后剩余资源仍然能够满足其他进程的需求。

如果系统能找到一个安全序列，那么系统就处于安全状态。但是，如果系统在资源分配后找不到任何一个安全序列，那么系统就进入了不安全状态，这意味着之后的所有进程都无法顺利执行下去。此时，如果某个进程提出资源请求，系统也可能重新回到安全状态。

在进行资源分配之前，系统应该考虑最坏情况，即如果系统处于安全状态，那么不会发生死锁。但是，如果系统进入不安全状态，那么有可能发生死锁。因此，在进行资源分配之前，需要预先评估这次分配是否会导致系统进入不安全状态，以此来决定是否允许资源分配的请求。

银行家算法是一种用于避免死锁的算法，其核心思想是在资源分配时保证系统处于安全状态。它根据系统中现有的资源数目和每个进程的最大资源需求量来进行资源的分配。

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0 ✓
P_3	0 1 1 ✓
P_4	4 3 1 ✓

（Request i ）

银行家算法的步骤如下：

1. 统计各种类型资源的总数以及系统当前可用资源数目。
2. 对于每个进程，统计其最大需求资源数以及已分配的资源数。
3. 找出一个安全序列，使得每个进程的资源需求都可以被满足，即使在其他进程释放资源之后也不会发生死锁。

（3，3，2）{p1}　（5，3，2）{p1，p3}　（7，4，3）.... {p1, p3, p0, p2 ,p4}

4. 根据安全序列进行资源分配，并更新系统的可用资源数目。

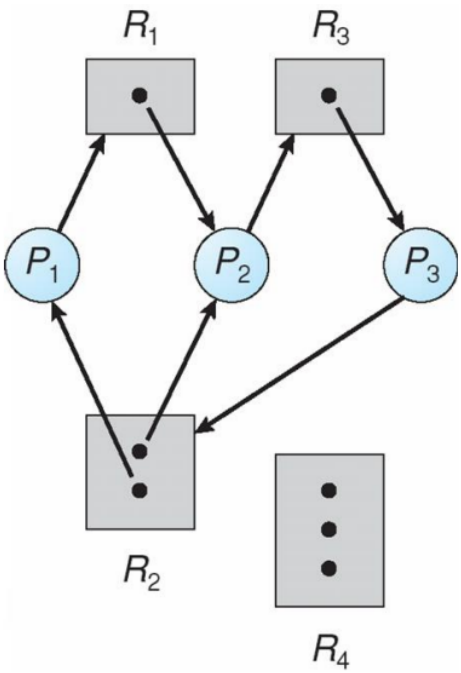
在银行家算法中，找到一个安全序列的方法是依次检查每个进程的资源需求是否可以被满足。如果某个进程的资源需求可以被满足，则将其添加到安全序列中，并更新系统的可用资源数目。通过这种方式，可以找到一个安全序列，并确保系统处于安全状态。

在实际操作中，为了更快地找到安全序列，可以先计算出每个进程还需要的资源数量，然后比较系统当前可用资源数和进程所需资源数的差值，如果差值能够满足某个进程的需求，那么该进程就可以被满足，并且系统的可用资源数目会相应减少。通过依次执行这个过程，可以找到一个安全序列，并确保系统不会发生死锁。

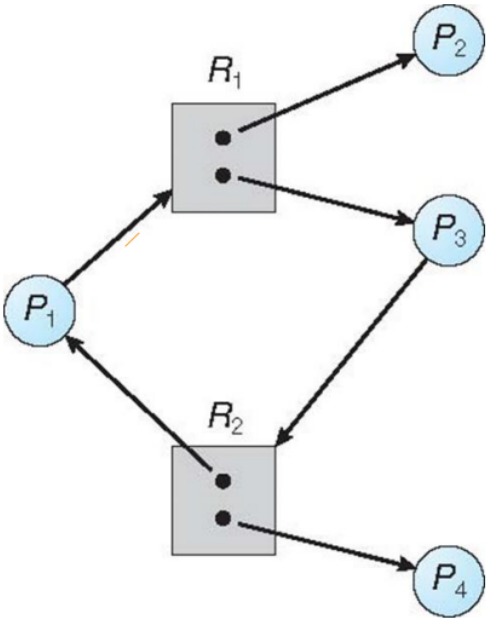
4.3 死锁检测与恢复

死锁检测与恢复是针对系统中已经发生死锁的情况的处理方法。以下是关于死锁检测和恢复的一些概念和算法：

Resource Allocation Graph With a Deadlock



Graph With a Cycle But No Deadlock



1. 死锁检测：
   - 死锁检测是用于确定系统是否已经发生死锁的过程。
   - 为了进行死锁检测，系统需要维护某种数据结构来存储资源分配信息，以及进程的请求信息。
   - 可以使用图的数据结构来表示资源分配情况和进程间的关系，其中节点代表进程或资源，边代表资源请求或分配的关系。
   - 通过分析这种图的结构，可以检测出是否存在死锁情况。
2. 死锁恢复：
   - 一旦系统检测到了死锁的发生，就需要采取措施解除死锁。
   - 死锁恢复算法旨在将系统从死锁状态中解脱出来，以恢复正常的运行状态。
   - 典型的死锁恢复算法包括资源剥夺、进程终止和资源回退等策略。
   - 资源剥夺策略是指系统主动剥夺某些进程的资源以解除死锁，进程终止策略是指系统终止部分或全部死锁进程以释放资源，而资源回退策略是指系统回退某些已经分配出去的资源以解除死锁。

死锁恢复是在系统检测到死锁发生时采取的解决措施，其目的是解除死锁并尽可能地恢复系统的正常运行状态。以下是几种常见的死锁恢复方法：

1. 资源剥夺法：
    - 该方法将死锁进程暂时挂起，并将其在内存中的状态保存起来，以便在后续执行中恢复。然后，系统抢占死锁进程所占用的资源，并将这些资源分配给其他进程。
    - 资源剥夺法的优点是实现简单，但可能会导致被挂起的进程长时间得不到资源而饥饿。
2. 进程终止法：
    - 也称为撤销进程法，该方法强制终止部分或全部死锁进程，并夺取这些进程所占用的资源。
    - 这种方法的优点是实现简单，但是被终止的进程可能已经运行了很长时间，因此可能会造成较大的损失。
3. 进程回退法：
    - 该方法是让死锁进程回退到足以避免死锁的状态，即回退到这些进程拥有某些资源之前的状态。
    - 系统需要记录进程的历史信息，以便能够回退到适当的状态。

在决定采取哪种死锁恢复措施时，可以考虑以下因素：

- 进程的优先级，通常选择优先级较低的进程先考虑。
- 进程已经执行的时间，执行时间较短的进程优先考虑。
- 进程完成所需时间，尽量避免选择即将完成的进程。
- 进程已经拥有的资源数，资源数较少的进程优先考虑。
- 进程的类型，例如交互式进程可能优先考虑以提高用户体验。

死锁检测算法则是依次消除与死锁进程相连的边，以解除死锁。通常是选择不影响其他进程的资源释放方式，以最小化对系统的影响。