哲学家就餐问题 [编辑]

维基百科，自由的百科全书

哲学家就餐问题（英语：Dining philosophers problem）是在计算机科学中的一个经典问题，用来演示在并发计算中多线程同步（Synchronization）时产生的问题。

在1971年，著名的计算机科学家艾兹格·迪科斯彻提出了一个同步问题，即假设有五台计算机都试图访问五份共享的磁带驱动器。稍后，这个问题被托尼·霍尔重新表述为哲学家就餐问题。这个问题可以用来解释死锁和资源耗尽。[1][2][3]

问题描述 [编辑]

哲学家就餐问题的演示

哲学家就餐问题可以这样表述，假设有五位哲学家围坐在一张圆形餐桌旁，做以下两件事情之一：吃饭，或者思考。吃东西的时候，他们就停止思考，思考的时候也停止吃东西。餐桌上有五碗意大利面，每位哲学家之间各有一只餐叉。因为用一只餐叉很难吃到意大利面，所以假设哲学家必须用两只餐叉吃东西。他们只能使用自己左右手边的那两只餐叉。哲学家就餐问题有时也用米饭和五根筷子而不是意大利面和餐叉来描述，因为吃米饭必须用两根筷子。

这个问题不考虑意大利面有多少，也不考虑哲学家的胃有多大。假设两者都是无限大。

问题在于如何设计一套规则，使得在哲学家们在完全不交谈，也就是无法知道其他人可能在什么时候要吃饭或者思考的情况下，可以在这两种状态下永远交替下去。

进一步说明 [编辑]

这个问题旨在说明避免死锁的挑战，死锁是一种程序无法继续运行的状态。要更好理解这个问题，假设我们要求哲学家遵守以下规则：

- 哲学家在左边的叉子可用（没有其他人拿起）之前处于思考状态。如果左边的叉子可用，就拿起来。
- 哲学家等待右边的叉子可用。如果右边的叉子可用，就拿起来。
- 如果两个叉子都已经拿起来，开始吃意大利面，每次吃面都花费同样的时间。
- 吃完后先放下左边的叉子。
- 然后放下右边的叉子。
- 开始思考（进入一个循环）

这个解法是失败的，当每个哲学家都拿起左侧的叉子，等待右侧的叉子可用时，就会进入死锁状态，每个哲学家将永远都在等待（右边的）另一个哲学家放下叉子。[4]

如果特定的哲学家由于时间问题而无法同时获得两个资源，那么资源匮乏也可能独立于死锁而发生。例如，假设规定当哲学家等待另一只餐叉超过五分钟后就放下自己手里的那一只餐叉，并且再等五分钟后进行下一次尝试。这个策略消除了死锁（系统总会进入到下一个状态），但仍然有可能发生“活锁”。如果五位哲学家在完全相同的时刻进入餐厅，并同时拿起左边的餐叉，那么这些哲学家就会等待五分钟，同时放下手中的餐叉，再等五分钟，又同时拿起这些餐叉。

互斥是此问题的基本概念。在实际的计算机问题中，缺乏餐叉可以类比为缺乏共享资源。一种常用的计算机技术是资源加锁，用来保证在某个时刻，资源只能被一个程序或一段代码访问。当一个程序想要使用的资源已经被另一个程序锁定，它就等待资源解锁。当多个程序涉及到加锁的资源时，在某些情况下就有可能发生死锁。例如，某个程序需要访问两个文件，当两个这样的程序各锁了一个文件，那它们都在等待对方解锁另一个文件，而解锁永远不会发生。

复杂的系统（例如操作系统内核）使用成千上万的锁，如果要避免死锁，资源匮乏和数据损坏等问题，就需要遵守更严格的方法和同步协议。

解法 [编辑]

服务生解法 [编辑]

一个简单的解法是引入一个餐厅服务生，哲学家必须经过他的允许才能拿起餐叉。因为服务生知道哪只餐叉正在使用，所以他能够作出判断避免死锁。

为了演示这种解法，假设哲学家依次标号为A至E。如果A和C在吃东西，则有四只餐叉在使用中。B坐在A和C之间，所以两只餐叉都无法使用，而D和E之间有一只空余的餐叉。假设这时D想要吃东西。如果他拿起了第五只餐叉，就有可能发生死锁。相反，如果他征求服务生同意，服务生会让他等待。这样，我们就能保证下次当两把餐叉空余出来时，一定有一位哲学家可以成功的得到一对餐叉，从而避免了死锁。

资源分级解法 [编辑]

另一个简单的解法是为资源（这里是餐叉）分配一个偏序或者分级的关系，并约定所有资源都按照这种顺序获取，按相反顺序释放，而且保证不会有两个无关资源同时被同一项工作所需要。在哲学家就餐问题中，资源（餐叉）按照某种规则编号为1至5，每一个工作单元（哲学家）总是先拿起左右两边编号较低的餐叉，再拿编号较高的。用完餐叉后，他总是先放下编号较高的餐叉，再放下编号较低的。在这种情况下，当四位哲学家同时拿起他们手边编号较低的餐叉时，只有编号最高的餐叉留在桌上，从而第五位哲学家就不能使用任何一只餐叉了。而且，只有一位哲学家能使用最高编号的餐叉，所以他能使用两只餐叉用餐。当他吃完后，他会先放下编号最高的餐叉，再放下编号较低的餐叉，从而让另一位哲学家拿起后边的这只开始吃东西。

尽管资源分级能避免死锁，但这种策略并不总是实用的，特别是当所需资源的列表并不是事先知道的时候。例如，假设一个工作单元拿着资源3和5，并决定需要资源2，则必须先要释放5，之后释放3，才能得到2，之后必须重新按顺序获取3和5。对需要访问大量数据库记录的计算机程序来说，如果需要先释放高编号的记录才能访问新的记录，那么运行效率就不会高，因此这种方法在这里并不实用。

Chandy/Misra解法 [编辑]

1984年，曼尼·钱迪和贾亚达夫·米斯拉提出了哲学家就餐问题的另一个解法[5]，允许任意的用户（编号$P_1, \cdots, P_n$）争用任意数量的资源。与资源分级解法不同的是，这里编号可以是任意的。

- 对每一对竞争一个资源的哲学家，新拿一个餐叉，给编号较低的哲学家。每只餐叉都是“干净的”或者“脏的”。最初，所有的餐叉都是脏的。
- 当一位哲学家要使用资源（也就是要吃东西）时，他必须从与他竞争的邻居那里得到。对每只他当前没有的餐叉，他都发送一个请求。
- 当拥有餐叉的哲学家收到请求时，如果餐叉是干净的，那么他继续留着，否则就擦干净并交出餐叉。
- 当某个哲学家吃东西后，他的餐叉就变脏了。如果另一个哲学家之前请求过其中的餐叉，那他就擦干净并交出餐叉。

这个解法允许很大的并行性，适用于任意大的问题。

参考资料 [编辑]

1. ^ 戴克斯特拉, 艾兹赫尔. EWD–1000 (PDF). E·W·戴克斯特拉档案馆. 得克萨斯大学奥斯汀分校美国历史中心. (文字版本)
2. ^ J. Díaz; I. Ramos. Formalization of Programming Concepts: International Colloquium, Peniscola, Spain, April 19—25, 1981. Proceedings. Birkhäuser. 1981: 323, 326 [2020–12–16]. ISBN 978–3–540–10699–9. （原始内容存档于2016–12–19）.
3. ^ Hoare, C. A. R. Communicating Sequential Processes (PDF). usingcsp.com. 2004 [最初由普林帝斯霍尔于1985出版] [2020–12–16]. （原始内容存档 (PDF)于2016–01–27）.
4. ^ 戴克斯特拉, 艾兹赫尔. EWD–310 (PDF). E·W·戴克斯特拉档案馆. 得克萨斯大学奥斯汀分校美国历史中心. (文字版本)
5. ^ Chandy, K. M.; Misra, J. The Drinking Philosophers Problem. (PDF). ACM Transactions on Programming Languages and Systems. 1984 [2021–01–17]. （原始内容存档 (PDF)于2012–01–13）.

延伸阅读 [编辑]

- Silberschatz, Abraham; Peterson, James L. Operating Systems Concepts. Addison–Wesley. 1988. ISBN 0–201–18760–4.
- Dijkstra, E. W.（1971, June）. Hierarchical ordering of sequential processes（页面存档备份，存于互联网档案馆）. Acta Informatica 1 (2): 115–138.
- Lehmann, D. J., Rabin M. O,（1981）. On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. Principles Of Programming Languages 1981（POPL'81）, pages 133–138.

分类： 并发计算