我们希望一个分配器的吞吐率最大化，吞吐率定义为每个单位时间里完成的请求数。例如，如果一个分配器在 1 秒内完成 500 个分配请求和 500 个释放请求，那么它的吞吐率就是每秒 1000 次操作。一般而言，我们可以通过使满足分配和释放请求的平均时间最小化来使吞吐率最大化。正如我们会看到的，开发一个具有合理性能的分配器并不困难，所谓合理性能是指一个分配请求的最糟运行时间与空闲块的数量成线性关系，而一个释放请求的运行时间是个常数。

- 目标 2：最大化内存利用率。天真的程序员经常不正确地假设虚拟内存是一个无限的资源。实际上，一个系统中被所有进程分配的虚拟内存的全部数量是受磁盘上交换空间的数量限制的。好的程序员知道虚拟内存是一个有限的空间，必须高效地使用。对于可能被要求分配和释放大块内存的动态内存分配器来说，尤其如此。

有很多方式来描述一个分配器使用堆的效率如何。在我们的经验中，最有用的标准是*峰值利用率*(peak utilization)。像以前一样，我们给定 n 个分配和释放请求的某种顺序

$$R_0, R_1, \cdots, R_k, \cdots, R_{n-1}$$

如果一个应用程序请求一个 p 字节的块，那么得到的已分配块的*有效载荷*(payload)是 p 字节。在请求 R_k 完成之后，*聚集有效载荷*(aggregate payload)表示为 P_k，为当前已分配的块的有效载荷之和，而 H_k 表示堆的当前的(单调非递减的)大小。

那么，前 $k+1$ 个请求的*峰值利用率*，表示为 U_k，可以通过下式得到：

$$U_k = \frac{\max_{i \leq k} P_i}{H_k}$$

那么，分配器的目标就是在整个序列中使峰值利用率 U_{n-1} 最大化。正如我们将要看到的，在最大化吞吐率和最大化利用率之间是互相牵制的。特别是，以堆利用率为代价，很容易编写出吞吐率最大化的分配器。分配器设计中一个有趣的挑战就是在两个目标之间找到一个适当的平衡。

旁注 放宽单调性假设

我们可以通过让 H_k 成为前 $k+1$ 个请求的最高峰，从而使得在我们对 U_k 的定义中放宽单调非递减的假设，并且允许堆增长和降低。

9.9.4 碎片

造成堆利用率很低的主要原因是一种称为*碎片*(fragmentation)的现象，当虽然有未使用的内存但不能用来满足分配请求时，就发生这种现象。有两种形式的碎片：*内部碎片*(internal fragmentation)和*外部碎片*(external fragmentation)。

*内部碎片*是在一个已分配块比有效载荷大时发生的。很多原因都可能造成这个问题。例如，一个分配器的实现可能对已分配块强加一个最小的大小值，而这个大小要比某个请求的有效载荷大。或者，就如我们在图 9-34b 中看到的，分配器可能增加块大小以满足对齐约束条件。

内部碎片的量化是简单明了的。它就是已分配块大小和它们的有效载荷大小之差的和。因此，在任意时刻，内部碎片的数量只取决于以前请求的模式和分配器的实现方式。

*外部碎片*是当空闲内存合计起来足够满足一个分配请求，但是没有一个单独的空闲块足够大可以来处理这个请求时发生的。例如，如果图 9-34e 中的请求要求 6 个字，而不是 2 个字，那么如果不向内核请求额外的虚拟内存就无法满足这个请求，即使在堆中仍然有