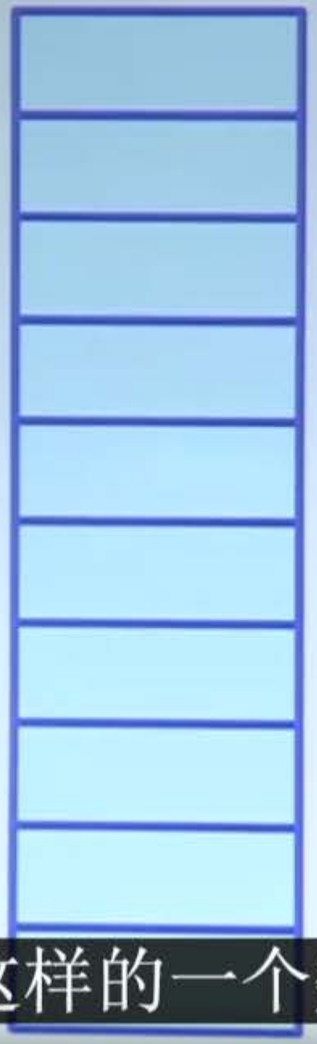
位图法、空闲区表、空闲区链表

物理内存管理

下面呢，我们来介绍物理内存管理 我们对物理内存呢，有不同的啊划分

空闲内存管理

- 数据结构
  - 位图
    - 每个分配单元对应于位图中的一位，0表示空闲，1表示占用（或者相反）
  - 空闲区表、已分配区表
    - 表中每一项记录了空闲区（或已分配区）的起始地址、长度、标志
  - 空闲块链表



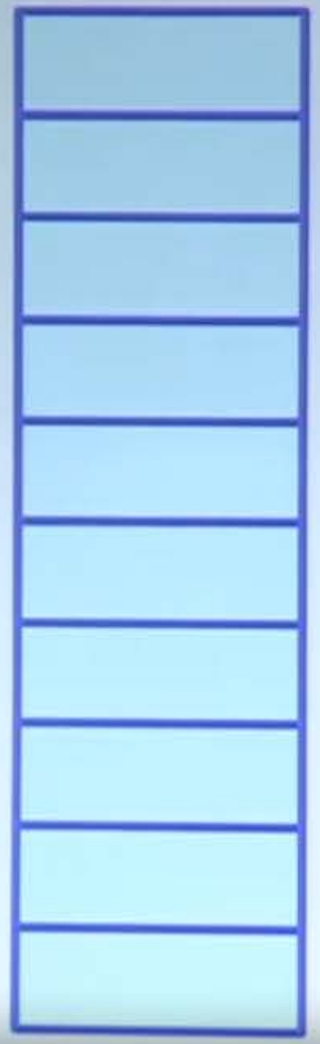
等长划分

bitmap 位图的方式 那么这样的一个数据结构，每一个分配单元

1:12 / 7:18

空闲内存管理

- 数据结构
  - 位图
    - 每个分配单元对应于位图中的一位，0表示空闲，1表示占用（或者相反）
  - 空闲区表、已分配区表
    - 表中每一项记录了空闲区（或已分配区）的起始地址、长度、标志
  - 空闲块链表



等长划分

内存分配算法

- 首次适配 first fit
  - 在空闲区表中找到第一个满足进程要求的空闲区
- 下次适配 next fit
  - 从上次找到的空闲区处接着查找
- 最佳适配 best fit
  - 查找整个空闲区表，找到能够满足进程要求的最小空闲区
- 最差适配 worst fit
  - 总是分配满足进程要求的最大空闲区

将该空闲区分为两部分，一部分供进程使用，另一部分形成新的空闲区

示例

空闲区表

始址	长度	标志
15K	23K	未分配
48K	20K	未分配
80K	30K	未分配

已分配区表

始址	长度	标志
0K	15K	P_1
38K	10K	P_2
68K	12K	P_3
110K	10K	
		空项

30K，啊从 80K 开始的 30K 的空闲空间

5:08 / 7:18

示例

空闲区表

始址	长度	标志
15K	23K	未分配
48K	20K	未分配
98K	12K	未分配
		空项
		空项

已分配区表

始址	长度	标志
0K	15K	P_1
38K	10K	P_2
68K	12K	P_3
110K	10K	P_4
80K	5K	P_5
85K	13K	P_6

6:03 / 7:18

回收问题

- 内存回收算法
  - 当某一块归还后，前后空闲空间合并，修改内存空闲区表
  - 四种情况
    上相邻、下相邻、上下都相邻、上下都不相邻

一个孤立的啊，空闲区，那么它就在这个空闲区表中单独占一个行 所以这就是回收的问题

0:00 下面呢，我们来介绍物理内存管理 我们对物理内存呢，有不同的啊划分 一种呢是等长的划分，也就是把一片空闲的物理内存 划分成大小相等的啊区域 那么，每个区域我们就称之为一个分配单元 实际上就是说，当某一个进程需要进入内存的时候呢，那么 需要可能若干个分配单元啊，满足这个进程的对内存的需求 第二种的划分呢，实际上就是不等长的划分。 也就是 空闲啊区，在内存中的空闲区 会有若干个，每个呢大小都是不等的，那么我们也要用 一种数据结构把这种不等长的空闲区把它管理起来 所以呢，我们下面来看看，有哪些数据结构可供我们使用？ 对于这种等长划分的啊 空闲区域，那么我们实际上就可以用 bitmap 位图的方式 那么这样的一个数据结构，每一个分配单元 呢，实际上就是对应了这个某一位，啊一个 bit 0 就表示这个分配单元现在是空闲的 1 就表示这个分配单元已经分给了某个进程 所以，当一个新的进程想要进入内存，那么 它就要去在位图当中，搜索一连串的 0 那么这个 0 呢，我们可以是连续的若干个 0 那么这是牵扯到了一些字符串的匹配的算法 那么，对于这种不等长划分的区域呢，我们可以用下面两种数据结构。 一种呢叫做空闲区表 那么当然啦，对应的就是已分配区表，分出去的就叫已分配区表 那么，空闲的区域呢，叫空闲区表 那么，这个两张表其实啊，这个结构是相同的 只是，每一个表项，可能表示的是空闲区，也可能是表示的是 已分配给某个进程的这么一块区域。 通常呢，我们要记录这个区域的起始地址 还有这个区域的长度，以及我们要配上一个标志 这个标志呢，可能指出这个空闲区是空闲的 也可能会指的这个区域呢，是分配给某个进程了 也可能是指的，是这个表项本身呢，就是空表项 就是没有用的一个表项，都可以。 所以，我们用这么一个数据结构呢，可以来表示这种不等长的区域 当然啦，如果我们用空闲区链表呢，我们就是把每一个表项呢用链把它串起来 这就是空闲内存的一个管理的不同的数据结构 下面我们以空闲区表和这个已分配区表为例呢，来谈一下内存分配的算法 第一个算法呢，叫首次适配 它就是在空闲区表当中，找到第一个能够满足进程要求的空闲区 就可以了，啊，只要顺着这个表往下找，找到第一个 那么，第二种算法呢，叫下次适配 所谓下次适配呢，实际上是对 首次适配的这么一个，算法的一个稍微的改进 它就是从上次找到的空闲区这个地方开始，继续接着查找 因为，首次适配是每次都从表头开始往下找 那么，下次适配呢是上次找到什么位置 下次啊，接着啊往下找，找到第一个啊满足进程要求的空闲区就可以了 第三种算法呢，叫最佳适配。 所谓最佳适配呢 就是查找整个空闲区表，找到一个能 够满足进程要求的那个最小的空闲区 当然，还有一种算法叫做最差适配 所

谓最差适配就是说，在这个空闲区表中总是 分配能够满足进程要求的那个最大的空闲区 那么，不同的啊，这个算法 当找到了一个满足要求的空闲区之后 实际上呢，是要把这个空闲区，把它分割成两部分 一部分呢是分配给进程，另一部分呢 是作为一个新的空闲区，记录在这个相应的数据结构里头 这就是内存分配算法。 好，下面我们来看一下 有一个新的进程 啊，要进入内存，那么它需要 5K 的空间 那我们选中了这样一个空闲的空间 现在有 30K ，啊从 80K 开始的 30K 的空闲空间 那么经过了这样一轮分配之后呢，那么 从 85K 到还剩下 25K 的空闲空间 那么已分配区表里头呢，我们就增加了一个新的进程啊，选项 然后，又有一个进程要进入啊内存 所以呢，我们接着把这 25K 又分解出去 那么就变成了还剩下 12K 那么，在已分配区表呢，我们就得到了这样一个结果啊，这样一个结果 那么，这样就是一个某个分配算法啊，每次在 使用的过程当中，那么得到的这么一个数据结构的改变 那么这个分配算法，大家可以看出来，是一个最差 适配啊，每次都是从最大的啊，那个空闲区开始分 空闲区分配的问题介绍完了 下面我们来看一下，进程执行结束后 还回空间以后，系统如何将它们回收的 内存的回收算法，其实主要考虑的一个问题就是合并 那么，当某一块啊空闲区还回来以后 那么它前后有没有空闲区，然后合并成一个更大的空闲区 来修改相应的空闲区表呢？通常情况下，我们分成四种情况 一种呢就是上相邻，也就是这个空闲区的 上面有一个空闲区，它们两个可以合并成一个更大的空闲区 下相邻，或者是上下都相邻 也就是有三块空闲区，最后合并成一个更大的空闲区 最后一种情况，就是上下都不相邻，也就是这个还回的空闲区就是 一个孤立的啊，空闲区，那么它就在这个空闲区表中单独占一个行 所以这就是回收的问题