外部碎片解决办法： 允许物理空间分配以非连续的方式进行（分段、分页）

* Page table is kept in main memory（页表是放在主存中的）
* **Page-table base register (PTBR)**
  + points to the page table （指向页表的基地址）
* **Page-table length register (PRLR)**
  + indicates size of the page table（记录页表长度）
* 在PTBR+PRLR的这种实现模式下，每次对指令和数据的访问都会涉及到几次内存访问？

**2次。**

一次访问页表，一次访问指令和数据所在的内存地址

指令重启的详细过程！！！！！

证明面包店算法满足临界区的3大条件（1。互斥2。空闲让进3。有限等待）

do{

choosing[i]=true;

number[i]=max(number[0],…,number[n-1])+1;

choosing[i]=false;

for(j=0;j<n;j++){

while(choosing[j]);

while((number[j]!=0) &&(number[j],j)<(number[i],i));

}

临界区

number[i]=0;

剩余区

}while(1);

解答：

* Number[i]=0 表示进程Pi还没打算争取进入临界区，所以不参与竞争
* 所以，考虑竞争的时候从Number[i]!=0的进程来考虑
* 有且仅有(number[i], i)最小的进程可以获得进入临界区的权限
* 每个进程i在进入竞争状态的时候，其前面最多有n-1个进程处于其优先地位。而且这些进程在进入并退出临界区之后，其Number就会被置0，或者也有可能随后被赋予一个比Number[i]更大的值，也就是说它们退出再进来，就必须排在Pi的后面了。
  + 所以进程Pi最多等待n-1次，就可以进入临界区

文件系统实现中的分配方法：

* 连续分配的主要优点如下：

(1) 支持顺序存取和随机存取

(2) 顺序访问速度快

(3)所需的磁盘寻道次数和寻道时间最少

* 连续分配的主要缺点如下：

(1) 要求有连续的存储空间

(2) 必须事先知道文件的长度，以后不能动态增长（预留空间、重新分配和移动）

(3)不利于文件插入和删除

(4)外部碎片问题

思考：**程序文件、数据库文件是否适合连续分配？**

**连续文件不宜用来存放用户文件、数据库文件等经常被修改的文件。**

* 该分配方案可能会导致磁盘碎片，严重降低外存空间的利用率。
* 解决方法之一，系统定期或不定期采用紧凑技术，将小分区合并为大的、连续分区，将文件占用空间合并在一起。

链接结构：

**优点：**

* + 不存在外部碎片问题，提高了磁盘空间利用率
  + 有利于文件插入和删除
  + 有利于文件动态扩充

**缺点：**

* + 存取速度慢，不适于随机存取
  + 可靠性问题，如指针出错
  + 更多的寻道次数和寻道时间
  + 链接指针占用一定的空间
* 链接结构使得文件的存储空间失去连续性，不便于文件的顺序访问。
* 为此，系统可以周期性的把每个文件的若干离散数据块进行整理，并修改文件分配表中文件的起始地址。

索引结构：

* **优点:**
  + 既可以满足文件动态增长的要求，又可以较为方便和迅速地实现随机存取。
    - 因为有关逻辑块号和物理块号的信息全部放在一个集中的索引表中，而不是像串联文件结构那样分散在各个物理块中。
* 缺点:
  + 由于使用了索引表而增加了存储空间的开销。
  + 在存取文件时需要至少访问存储器二次以上。其中，一次是访问索引表，另一次是根据索引表提供的物理块号访问文件信息。由于文件在存储设备的访问速度较慢，因此，如果把索引表放在存储设备上，势必大大降低文件的存取速度。
* 改进方法：对某个文件进行操作之前，系统预先把索引表放入内存。

**其中存在的问题?**

文件很大，文件索引表也就较大。

* + 如果索引表的大小超过了一个物理块，那么我们必须象处理其他文件的存放那样决定索引表的物理存放方式，但这不利于索引表的动态增加；
  + 索引表也可按串联方式存放，但这却增加了存放索引表的时间开销。