10 虚拟内存

背景：内存相对于硬盘来说较小，一些存放在硬盘上的程序很难全部放在内存执行，但是每次执行的代码只是整个程序的一部分，而且存在一部分的代码很少被执行到。

后备区 (virtual memory, backing store)：硬盘上的一个区域（交换分区，一般速度较快），交由内存管理当作内存使用。

- 后备区充当物理内存的缓冲区；
- 后备区也被划分成若干页进行管理；
- 后备区和普通硬盘间的数据可以进行重映射进行动态调整，不需要实际的数据迁移。

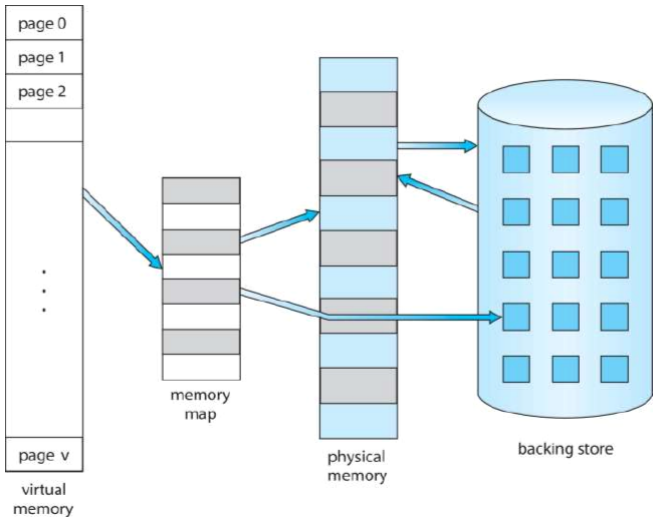
虚拟内存：将用户的逻辑地址和物理内存隔离开的技术，即“逻辑地址”对应的“虚拟”的内存空间。

- 虚拟内存的大小可以进行调整；
- 虚拟内存可以使应用程序运行的更快；
- 逻辑地址空间可以远大于物理地址空间；
- 能够允许更多的数据共享；
- 能够允许更多程序并发执行；
- 虚拟内存被划分成页进行管理。

虚拟地址空间 (Virtual address space) 即逻辑地址空间。

- 稀疏地址空间 (Sparse address spaces)；
- 栈从顶向下增长，堆从底向上增长。

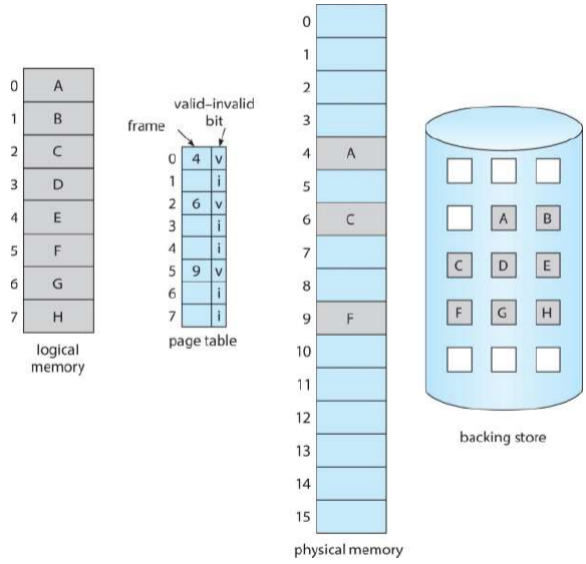
按需分页 (Demand Paging)：当一个进程被调入物理内存时，他的所有页并不是全部被调入，而是根据需要调入物理内存。运用 **懒交换** (Lazy Swapper) 技术：直到进程需要用到某一页的数据时，才将该页从虚拟内存调入物理内存（虚拟内存中的数据仍然存在）。



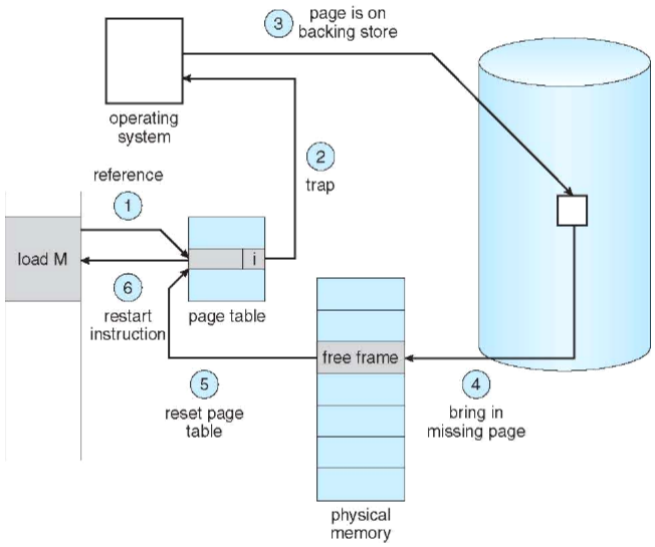
需要实现的功能

- 如果页已经存在在物理内存中（内存驻留页, memory resident），直接映射到物理内存；
- 如果页不在物理内存中，映射到虚拟内存，并将其调入物理内存（这一个过程中，程序不需要做任何的改动）。

Valid-Invalid Bit：页表里的合法位。Valid 表示该页在内存中，在页表中可以找到对应关系；Invalid 表示该页在后备区中，在页表中不能找到对应关系，需要去后备区调用，产生 **页面失效** (Page Fault)。



解决页面失效



- 通过 TLB+页表 查询该页是否在物理内存中；
- 如果不在物理内存中，启动陷阱程序，进入操作系统内部；
- 查询页在后备区的位置；
- 在物理内存中申请空闲页框（可能需要放回一些其他页），将该页调入物理内存中；
- 更新页表；
- 重新执行指令。

按需调页的两个阶段

- 第一阶段，纯导入页面 (Pure demand paging)（初始物理内存为空）；
- 第二阶段，将物理内存和后备区的页进行调换。可能需要分析进程的数据访问特征 (access patterns)、局部性 (locality of reference)。
  - temporal locality：时间局部性；
  - spatial locality：空间局部性；
  - access frequency：访问频率。

空闲页表 (Free frame list)：存储空闲的页表。

按序调页的有效访问时间

$$EAT = (1-p)\cdot \text{memory access} + p\cdot(\text{page fault overhead} + \text{swap page in} + \text{swap page out})$$

Read Copy Update, RCU 是解决页面共享问题的一种锁的实现（进程管理层面）。

写复制 (Copy-on-Write, CoW) 内存管理机制，允许父进程和子进程共享一些内存页面。当一个进程需要写一个页面时，复制一份页面，并在新页面上进行处理（初始直接页共享，有修改再复制）。

页面替换 (Page Replacement) 当出现页面失效时，用页面替换将需要的页面从后备区调到内存中，替换掉一个内存中的页面。

- 当数据发生修改，有两种处理方式：
  - **写回 (write back)** 只写内存中的数据，效率高但是同步差：当内存中该页被修改时，记录**脏位 (dirty bit)**，当该页被替换后，再将内存中的数据写入硬盘进行保存；
  - **写穿 (write through)** 同时写内存的数据和硬盘的数据，效率低但同步好，适用对数据同步性要求高的应用。
- 页面替换算法：找到一个**牺牲页 (victim frame)**进行页面替换。