- 内存管理需要实现的功能
- 虚拟 (逻辑) 地址空间和物理地址空间 (Address Space)
- 连续内存分配 (Contiguous Memory Allocation)
  - 动态空间分配算法：First-Fit、Best-Fit、Worst-Fit、Quick-Fit
- 内部碎片和外部碎片 (Fragmentation)
- 转换检测缓冲区/快表 (Translation Look-aside Buffers, TLB)
- 有效访问时间 (Effective Access Time)

一、内存管理需要实现的功能（简答）

内存管理：操作系统对内存的划分和动态分配

- 内存空间的分配和回收
- 地址转换：逻辑地址到物理地址
- 内存空间的扩充：利用虚拟存储或自动覆盖技术
- 内存共享：允许多个进程访问内存的同一部分
- 存储保护：各道作业在各自存储空间运行，互不干扰

二、虚拟地址和物理地址（简答）

进程运行和看到的都是逻辑地址

不同进程可以有相同逻辑地址，因为这些相同逻辑地址可以映射到主存不同位置

物理地址空间：内存中物理单元的集合，是地址转换的最终地址

地址重定位MMU，逻辑地址通过页表映射到物理内存

三、连续内存分配（T？）

内部碎片

程序小于固定分区也要占用一个完整的内存分区（造成空间浪费）

已经被分配出去，却不能被利用的内存空间

外部碎片

动态分区中内存中产生越来越多小的内存块（内存利用率下降）
还没有被分配出去（不属于任何进程），但由于太小无法分配给申请空间的新进程

单一内存分配
仅一道用户程序，无外部碎片，无需内存保护，但只能用于单用户单任务，有内部碎片

固定分区分配
有大小不一定相等的固定分区，有一个状态装配位
缺点：太大放不进去，太小内部碎片

动态分区分配
分区和大小不定，产生大量外部碎片，需要合并回收

T4：动态空间分配算法

First Fit:

- 从头找第一个大小满足的
- 最快，最好
- 缺点：低地址（开始的地方）有很多小空闲分区，每次查找都要经过一遍

Next Fit:

- 从上次结束的位置向下查找
- 尾部分裂成小碎片

Best Fit:

- 按容量递增形成空闲分区链，避免大材小用
- 性能差，每次最佳分配都产生很小难以利用的内存块，产生了最多的外部碎片

Worst Fit:

- 递减，就要大材小用
- 性能差，大内存块很快就没了

- 分段 (Segmentation)
  - 段表 (Segment Table) 的结构、分段中的地址转换
- 分页 (Paging)
  - 页表 (Page Table) 的结构、分页中的地址转换、页面共享
  - 页表的实现方式：多级页表、倒排页表
  - 不同页面大小对分页的影响
- 虚拟内存 (Virtual Memory)
  - 请求调页 (Demand Paging)
  - 缺页错误 (Page Fault) 的处理流程

四、

转换检测缓冲区&相联存储器&快表（为什么）

- 为什么：否则，先访问页表，得到物理地址，再取数据或指令，反而比不用页慢了一倍
- 利用了局部性原理：时间/空间

T5：有效访问时间EAT（计算）

- 无论找不找得到，搜索TLB都需要时间
- tlb找到了也要有访存时间！！
- 如果二级页表，后面那个式子2t变成3t
- 找不到时间访问时间就要双倍一下
- 还好找不到的概率比较低

- **Effective Access Time (EAT) = $(\varepsilon + t)\,\alpha + (\varepsilon + 2t)(1 - \alpha)$**
  - Hit Ratio (α): percentage of times that a page number is found in the TLB
  - Memory access time (t)
  - TLB search time (ε)
- Suppose the hit ratio is 80%, and it takes 20ns for TLB lookup and 100ns for memory access
  - EAT = 0.8 x 120 + 0.2 x 220 = 140ns (40% slowdown)
- If hit ratio is 99%
  - EAT = 0.99 x 120 + 0.01 x 220 = 121ns

五、分段

为什么：从用户和程序员角度考虑，方便编程、信息保护、共享、动态增长、动态链接
为什么地址空间二维：段长度不固定，不能像页氏管理一样给出一个整数就能确定物理地址，无法通过整数除法算出段号，无法求余算段内偏移，这两个一定要给出。而且还因为不定长，需要偏移量检测越界，不像页氏管理不会页内越界

段表的结构

- 每个段表项对应了进程中的一项
- 段表现记录该段在内存中的起始和长度：段号+段长+本段在主存的起始地址

T3：分段中的地址转换（计算？）

- 设置段表寄存器保存段表始址F+段表长度M
- 逻辑地址A=段号S+段内偏移量W（注意二进制还是十进制）
- S必须比M小，否则越界中断
- 段表项地址=F+S*段表项长度（不是M），查阅段表现，取出前几位得到段长C和后几位段始址b
- W必须比C小，否则越界中断
- 物理地址E=b+W

六、分页

页表结构

记录页面对应的物理块号，作用是实现从页号到物理块号的地址映射
1、页表项=页号+物理内存块号
（地址结构=页号+页内偏移）（物理地址=物理内存块号+页内偏移）
2、请求分页系统中的页表项：（增加4个字段）

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

- 状态位：是否已调入内存
- 访问字段：一段时间内被访问次数，或最近多久未被访问
- 修改位：调入内存后是否被修改过
- 外存地址：通常是物理块号

T3：分页中的地址转换（计算）

- 设置页表寄存器保存页表始址F+页表长度M（之前在PCB），页面大小L

- 页号P=A/L，业内偏移量W=A%L(简直不要太方便)（页4kb即十六进制逻辑地址后三位不看）
- P必须比M小，否则越界中断
- 查表得到b：页表项地址=F+P*叶表项长度（4B），取出页表项内容b（物理块号，即页框号）
- 页表长度M：总页数，页表项长度：页地址占用的空间
- 物理地址E=b*L+W

给地址，先换算成页号和偏移量

页面共享

为什么：不同用户运行同一程序，没必要有同一页面的多个副本

（但只有程序文本适合共享，数据页面不行，除非只读数据）

实现高性能消息传递系统，只需复制页面名字，不需要复制所有数据

多级页表

为什么&优势：避免把全部页表一直保存在内存中；超大连续空间难

顶级页表只能有一个

占用的地址位数是用多少个页表项算的：

- 地址位数=一级页表（页目录号）+二级页表（页号）
- 系统位数32=剩余逻辑地址空间（页目录号）+顶级页表页表项个数对应位数（页号）+页内偏移地址
- 页号即一页能包含多少页表项，页目录号用总的减去页号和页内偏移算

倒排页表

每个页框对应一个页表项，而不是虚拟页面

计算倒排页表页表项个数：不过是节省了页表项4B占的两位罢了（物理内存-页面大小）

- 优点：节省了大量空间；只需为所有进程维护一张表
- 缺点：仅包含调入内存的页面缺页时仍需多访问一次磁盘，地址转换效率低
- 但查找表项（n,p）必须搜索整个倒排列表，且每个访存操作都要执行一次搜索，不仅是缺页时
- TLB引用之后好很多，再引入散列表搜索，使散列表槽数和物理页面一样多

页面大小对分页影响（页面大小设计？？）

- 小页面内部碎片小，但需要大页表，浪费TLB
- 最佳页面大小算法se/p+p/2最小，即p^2=2se时取等

> 进程占用空间s，页表项大小e，页面大小p

T1：页表计算

段页式存储：虚拟地址位数-最大段位数=段号位数

段内地址=最大段位数，分出12位给偏移量

进程的虚拟地址空间是32-12

- 页大小+32位可算出总页数
- 页表最大包含多少页表项=总页数（不是每页有多少个页表项！不看物理内存位数！！）
- 页表空间：页表项大小*总页数
- 一页的页表项数=页大小\times页表项大小
- 页内偏移量要用页大小确定
- 虚页数=虚拟地址-页内偏移，页表级数=虚页数/每页页表项数
- 页号是向下取整的！
- 物理地址=物理内存块号（理论上可利用页表项的所有位数）+页内偏移

7、

假设一个支持分页的计算机系统有36位的虚拟地址，页面大小为8KB，每个页表项占用4 Bytes

a) 虚拟地址空间中共有多少个页面?

b) 该系统可访问的最大物理地址空间为多少?

c) 如果进程的平均大小为8GB，此时应选择一级、二级还是三级页表? 为什么? 在你选择的方案下，页表的平均大小是多少?

(a)

8 KB =2^{13} 位，因此虚拟页号位数为 36 - 13 = 23 位，共有 2^{23} 个页面。

(b)

页大小对应了偏移量，13位

页表项占用 4 Bytes，即32 位，全部用来存放物理页号，再加上 13 位的页内偏移量，共45 位。

则最大的物理地址为 0x1FFF FFFF FFFF

(c)

进程的平均大小为2^{33} = 8GB，每个进程共 2^{20} 个页面，每页有2^{11}个页表项

只使用一级页表：页表占用 $2^{33-11} = 2^{22}$的空间，即 4 MB($2^{20} \times 4B$)

使用二级页表：把一级页表看成4MB新程序，新程序分成2^{22-13}页，即2^9个二级页表，页表占用 $2^9 \times 4B(\text{页表}) + 2^{13}(\text{页目录}) = 5 \times 2^{11}$，即 10 KB

使用三级页表：顶级页1页，总空间与二级页表基本相同

结论：选用二级页表

- 优于三级原因：空间大小类似，地址翻译速度快于三级页表。
- 优于一级原因：页表大小为 10KB，远小于一级页表，可以使用一页来存放页目录项，页表项存放在内存中的其他位置，不用保证页目录项和页表项连续。

页表平均大小：页表平均大小一定是8KB，但如果本题想问的是二级页表占用的空间大小，则为10KB

七、虚拟内存

请求分页，流程？？？

- 开始内存中没有页，页面在需要时才装入，而不是预先装入
- 建立在基本分页系统基础之上，增加请求调页和页面置换功能，需要硬件支持

- 需要维持一个有效位：是否在内存中

缺页错误处理流程

- 首先不要忘了保存CPU现场（陷入内核、保存PC...）
- 找到缺页后需要判断内存是否满了，不能直接CPU读缺页
- 内存满要用页面置换算法，被修改的页替换出去要写回外存(替换慢，所以挂起当前进程、上下文切换，页框标记为忙)
- 最终换页流程=CPU读缺页+启动IO+换入内存+修改页表
- 整个流程结束后，回到修改快表+修改访问位和修改位

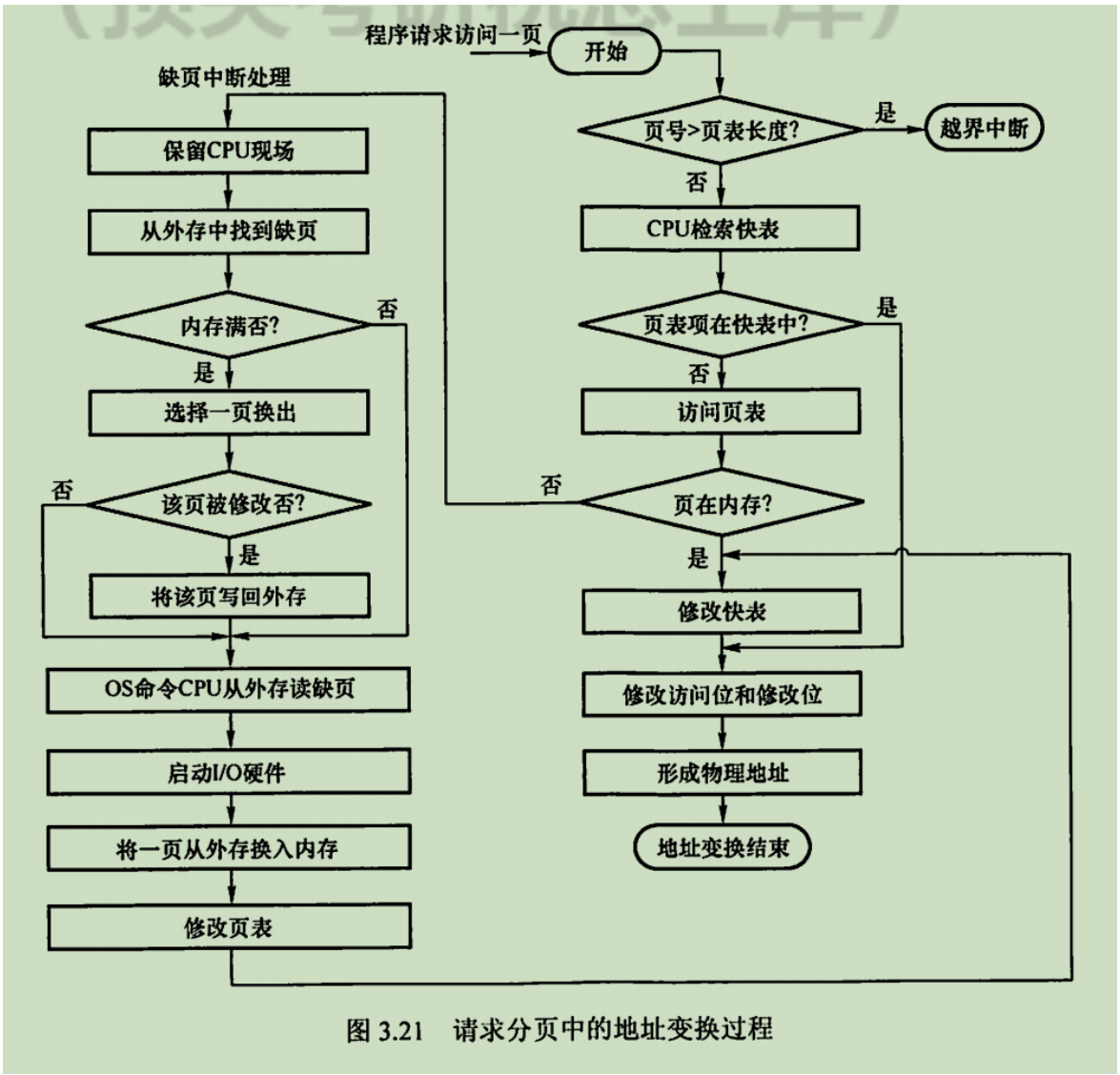

图 3.21　请求分页中的地址变换过程

- 页面置换算法 (Page Replacement)
  - FIFO、Optimal、Least Recently Used (LRU)、Not Recently Used (NRU)、Not Frequently Used (NFU)
  - 缺页中断率 (Page Fault Frequency) 和页框分配
- 工作集 (Working Set)
  - 抖动 (Thrashing) 和访问局部性 (Locality of Reference)
  - 工作集页面置换算法

八、页面置换算法

缺页中断率？？

页框分配？？？

OPT事后诸葛亮

- 需要置换页时，淘汰掉以后不再访问或距现在最长时间才访问的页
- 只能用于检测其他算法

FIFO

- Belady异常：分配的物理块增多，性能反而可能降低

LRU

- 淘汰访问字段最大的（上次访问到现在的时间）
- 堆栈类算法不出现Belady异常
- 难以实现！：需要维持一个特殊链表，开销巨大

NFU（aging）

- 为每个页面给一个软件计数器，每次时钟中断（频率影响性能）加上R位，记录被访问频繁程度
- 为了提高性能，每次加时将计数器右移一位，R位加到最高位上
- 置换掉计数器值最小的那个
- 最接近LRU的算法，怎么再接近？：增加寄存器位数，存储更多历史信息；减少时间间隔

第二次机会&CLOCK

- 设置一个检索指针，页面被替换时，指针指向下一位置

- 淘汰访问位为0中最老的，并在查找0过程中把指针遇到的1置0，如果都是1就全部置0

NRU=LFU=改进CLOCK

找到目标后随机置换，所以可能多个解！

没有M位就只看R位

除了页面使用情况还考虑修改位（置换代价）(修改位被置0一定是被写回磁盘)

好处：减少磁盘IO次数，但扫描次数比较多

- 扫描查找A=0，M=0（不改变访问位A）
- 如失败，扫描查找A=0，M=1（所有扫描过的访问位置0）
- 如失败，再来一遍必成功

九、工作集

抖动

刚换入的页面又要换出，刚换出的页面又要换入，这频繁的页面调度行为

原因：同时运行的进程太多，分配的物理块太少

工作集

某段时间内进程要访问的页面集合（基于局部性原理，要最近访问过的页面）

- 有时间t和工作集窗口大小Δ般比较大）
- 局部性好的，工作集大小比窗口小很多

工作原理：为进程分配大于其工作集的物理块，若还有空闲物理块就再调一个进程，若超了就暂停一个进程防止抖动

如何防止抖动？：内存中工作集总尺寸不能超过物理内存大小

T2：页面置换算法

对于一个有 4 个页框的机器，其每一页对应的载入时间、最近一次访问时间、以及每个页面的 Reference 和 Modify 位如下表所示。此时，FIFO、第二次机会、NRU 和 LRU 算法分别会选择哪个页面进行置换？

页面	载入时间	最近一次访问时间	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

FIFO（先进先出）：置换最早进入的页面3

第二次机会：页面2是R位为0中最早进入的，置换

NRU（最近未使用）：页面 2 的R和M位均为0，置换

LRU（最近最少使用）：页面1最近一次最晚，不用管R和M位，置换