

memory manager

一段原码a.java是需要被compiler编译

生成reloc code（重新编码）也就是a.obj，然后与library LINK到一起，作为.EXE（absolute program）存储到secondary memory中

然后如果我们想要运行这个a.exe，memory manager就会告诉main memory需要多少空间。然后main memory 给他提供这么多的空间并用PC COUNTER记录第一行所在位置，然后load 这个.exe

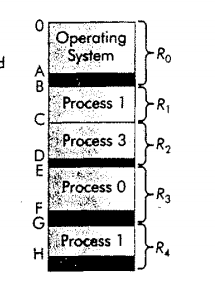
一行一行运行

figure 11.6

multiple programming：CPU不停的switch每个program，必须要read main meemory

首先我们要明确的是OS必须存在main memory中

1.fixed-partition memory



我们把main memory划分成许多小块，小于等于这个小块就可以放进去，大于最大小块，不会进行这个程序（如果是fixed情况）,

如果要多个程序（超过4个），只能wait

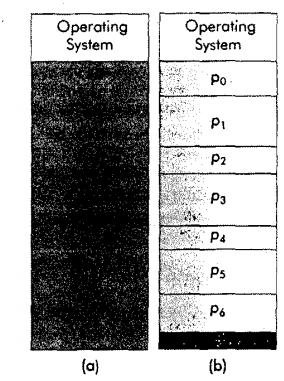
如果程序450，这时有个500的partition，我们把450程序放入500partition，那么多的50叫做internal memory fragment 。无法使用，造成浪费

internal fragmentation出现在fixed-sized partition中

注意，external fragmentation也会出现在fixed-sized partition中，只要你没分割完。

variable partition：

进多少我给多少

没有internal fragmentation，但最下面会有external fragmentation

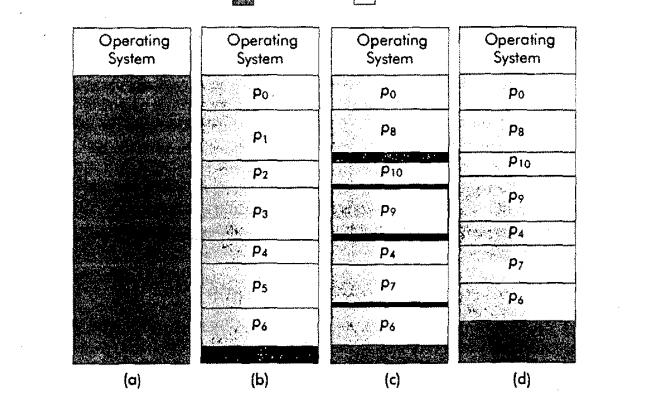
这时P1结束P8就进入

进入P8这里也有分类

best fit:80的空进去75的

worst fit，80的空进去5的

best fit不见得是best

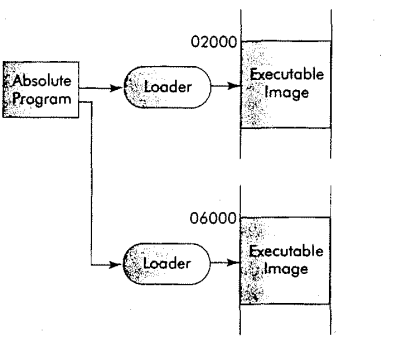


产生了新的fragmentation，注意，这个fragmentation是external fragmentation（小的也是）

然后我们通过dynamic allocation动态分配，（图d），把所有program上移，

因为是在main memory间互换，所以不需要过多时间

但是问题在于这样每个program的address改变了（我们main memory中exe文件是开头位置+长度）

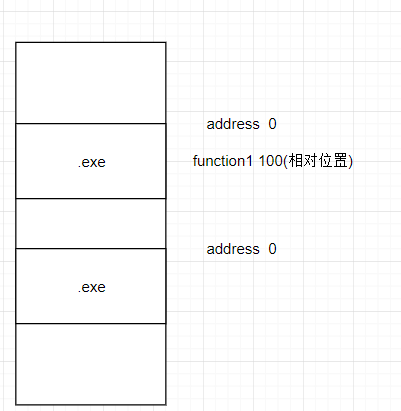


当一个可执行映像在main memory之间互换的时候。程序的机械指令必须被改变，因为CPU是利用PC counter来读取main memory第几行到几行 来运转这个程序的

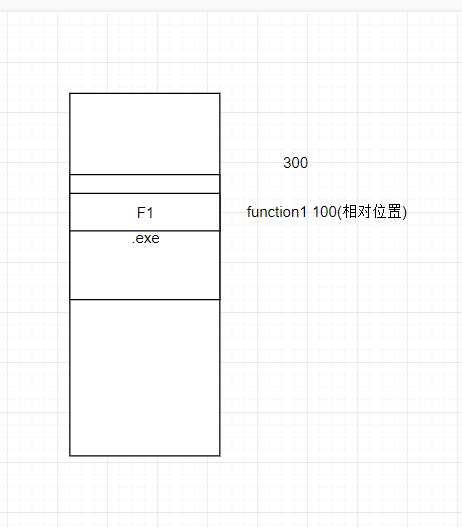
但是假设这个function写的时候是从第三百行到第四百行。而COPY的时候没有第300行到400行给你用

解决方案：virtual address

所有的program(.exe文件)都是从0开始，



在secondary memory的时候，function一开始都是相对位置



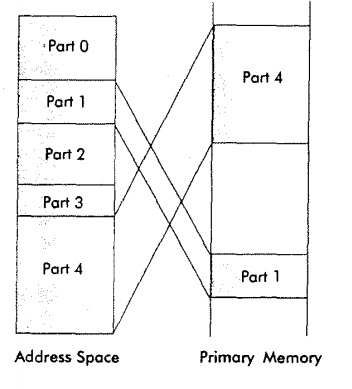
那么main memory中，当PC counter用这个function的时候，他的实际address等于SA(start address 300)+virtual address(100)=400

我们每次想在main memory移动他只需要改sa就行，process甚至不知道发生了什么，这就叫context switch

这就叫做dynamic address relocation

这样的系统叫做relocatable system

Virtual memory



virtual memory并不真实存在，而是一个思想

他利用virtual address把一个程序切割成了多个part，把需要运行的部分swap到main memory，大大节省了main memory的空间

当然,swap 会让整个Project更慢

换一种解释：程序的执行是建立在main memory的address上的,virtual memory可以把secondary memory的exe文件以main memory的方式来address，只要在需要用他的时候把他放到main memory上，实际上是建立在secondary memory上的

但是如果是上图所示情况，part4结束，我们开始Part2，又造成了fragment

解决方法，在part切割的时候，全部切割成same size

PAGE与FRAME

假设secondary memory有一个10000行的exe文件

我们把它break size为100，一共有100part，那么page就是100（part的学名）

frame：main memory中最多有几part，总容量1000，10\*100,那么f0,f1,f2,f3,…f9就是他的十个frame

没有pregnant

实际上还是有fregnant的，假设main memory去除OS，容量是1024，我们只能切走1000，但是这个24是O（1）,是constant loss，是我们可以忽略的external pregnant

这个方式也会产生internal pregnant

比如说一个程序长度是9901，那么注定有一个Page是1，被放入100的frame里，产生了99的internal pregnant. 24是external pregnant

worst case，9901并且那个1是一直被需要的page来维持运行。

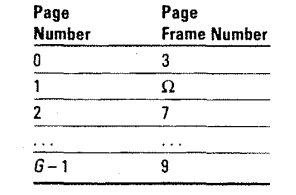
为什么这个idea好,因为worst case过分极端，要非常bad luck才能进行

Virtual address第一行是main method

假设我们需要VA的4215行

第一步，决定他在哪个page，4215/100取整，他在第42page，（Page从0开始算）

第二步，LOAD他，并查看page table，看是不是在main memory里，如果在，在哪个frame，如果不在，并且没有更多的frame给你用，我们就等一个page完成并swap



第三步，换进main memory以后，计算他的physical address，physical address=frame number\*100+4215mod 100(line number)

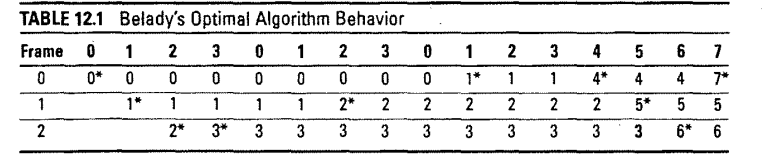
page fault:当程序需要某一行的data/code,但这一个page还没有被main memory所load

解决page fault的算法

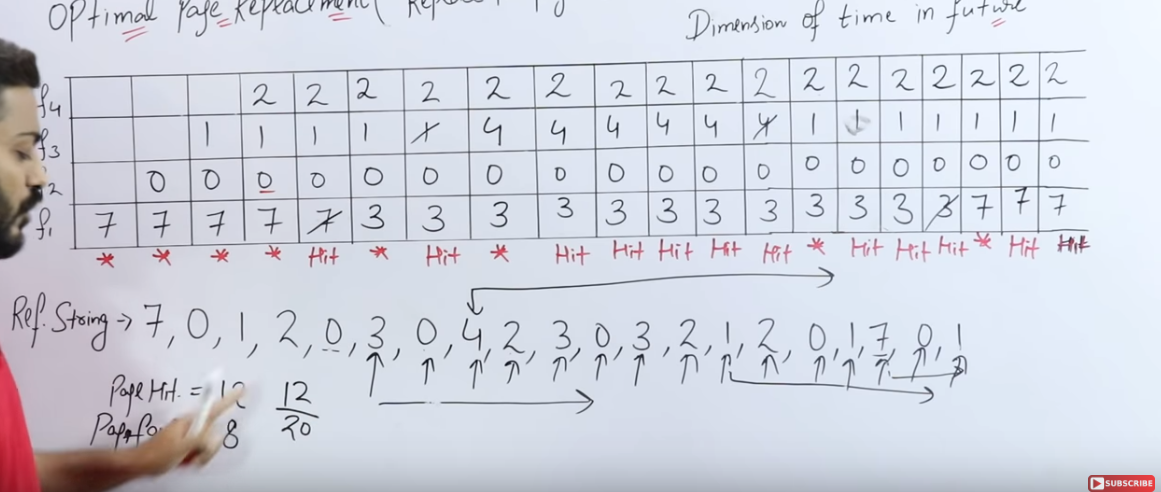
1.belady optimal algorithm behavior

这个algorithm是建立在我们知道reference stream的情况下的



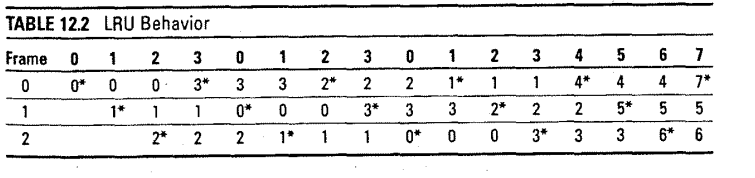
第一步查看有几个frame，我们有三个frame，

换掉最远的第一次出现的那个，比如3这里插入，我们的选项是换012，最远的是2，换2，567这样的情况会换左边的最远距离



我们没有改变frame的时候就叫一个Hit（算法生效）,hit ratio等于hit/total

替代算法，least recently used





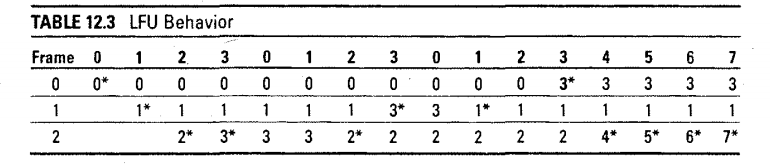
我们认为最老的那个是不再被需要的，



比如说这里2需要替换，可选项是701，理他最远的是7,3这里要替换，可选项是201，理他最远的是1（哪个第一个出现的最远）

基于假设，老的就不再重要

least frequently used with random tie balance



每一个出现的数都会有个counter，换取counter最小的，如果counter一样，随机

complexity是O（n），每次都要check所有counter

什么时候最好,在locality中

什么时候最坏,从locality跳出来以后

什么时候给予更多的frame

当page fault数量超过minimum，会有更多frame也就是更多memory，一旦你到达了minimum,就不会给你更多memory

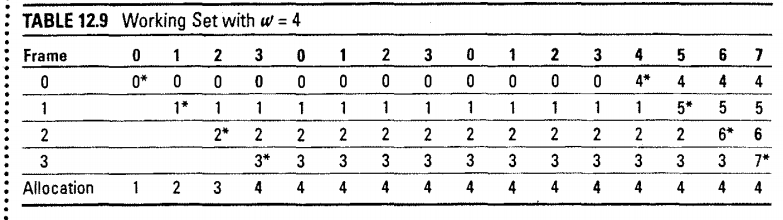
换句话说，你能替代的frame是有限制的

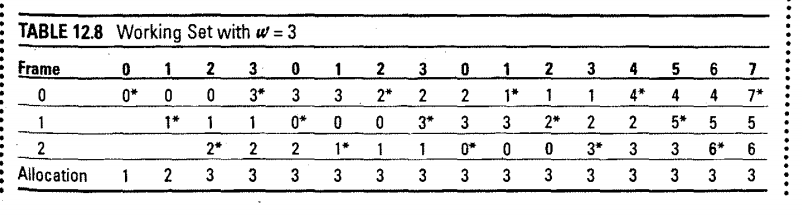
fix number replacement

比如告诉你，你能用的frame是F2，F12，F22,那么即使整个memory是空的，你也只能使用指定memory

working set algorithm

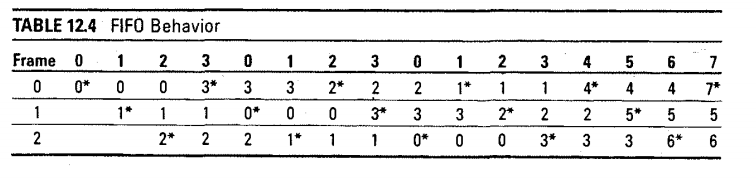
不给particular process指定particular memory，一直给你memory只要有空闲你就给你，但是你只能用指定的frame，如果超过一段时间（w）不用，我收回，这样给别人省下frame





working set的长度就是从占据开始数那么多格，就只能这么多，然后t+1没有就收回，有就继续

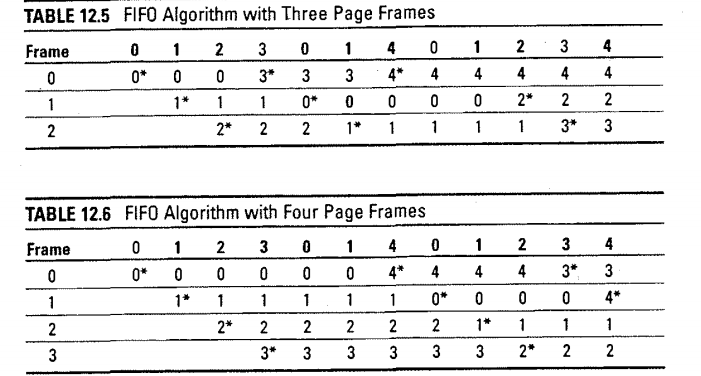
FIFO, terrible algorithm，因为他不是stack algorithm



替换掉持续时间最长的page

比如3这里插入，最长的0,0插入最长的1，

然而随着frame越多，fault反而越多



9到10，这种情况叫做belady’s anomaly

stack algorithm:

**stack algorithm** is one in which the pages kept in memory for a frame set of size N will always be a subset of the pages kept for a frame size of N + 1. And it states that **stack** based **algorithm** does not suffer from Belady's anomaly.

增长frame不会导致fault增加

唯一good algorithm，random replacement algorithm， 因为complexity是O（1），随便取一个，替换

但是我们做的一切都是为了避免page fault，其实影响好小

CPU进行一条指令所用的时间其实就等于进入main memory的时间

eat(effective access time)=ma(memory access time)

effective access time= (1-p)\*access time+p\*pagefault time

p是pagefault的几率

只要pagefault time够小，那么造成的影响就小

SWAP SPACE

计算机有足够的main memory存储，但是有时候我们需要更多的main memory，所以我们需要占用secondary memory， swap space是在secondary memory上的空间，他是一个虚拟内存，存储process image进程映像。swap space帮助OS假装有更多的内存

process image：当memory allocate一个Process（给一个process在main memory中分配位置），并在试图运行这个Process之前，他必须创造一个image（映像，也就是复制品）存储在main memory中（换句话说，实际分配的是复制品），因为在multitasking中 ，kernel核心会不停的re -enter这个process然后又left off， 如果process在这过程中变化了，就很难搞。

swap space是main memory中满了以后存储在secondary 中的process image

virtual memory 是swap space与main memory的结合。

virtual memory 同样是一种思想。就是用

他们的关系是：

page实现两者互换。

page size的大小选择

如果page size太小，

坏处：page table大，一页内容少，page fault频繁

好处：可以运行更多软件

如果page size太大

坏处：运行软件少

好处：page table小，一页内容多，page fault少

总而言之太大太小都不好

假设进入cache是2ns，进入main memory是40ns,

分以下几种情况

1.table足够小，完全在cache(主板上的memory)里

2+40(找table，找对应page)

2.如果cache太小，只能存40％table

理论时间是0.4(2+40)+0.6(40+40)

为什么不是0.6(2+40+40)即找cache没找到，再找mm?

因为他把指定的part放在cache里

例如0到40的page放在cache，41到100放在mm，如果我们要的是51，直接找Mm

3.如果page table太大，mm不够用，50％second memory,30％main memory， 20％cache

0.2\*(2+40)+0.3(40+40)+0.5(40+200+40+40)

40+200，现请求main memory发现不在，然后200解决page fault问题，把这个page table换到mm里，然后再重复mm动作

这三种情况都建立在所求页已经在mm里

200=getting pagetable+page table swapping+updating the table

如果不在main memory里的情况

需要用200来换page

0.2\*(2+200//不在main memory，换到main memory并Update table+2//查找新table+40)

+0.3(40+200+40+40)

+0.5(40找main memory table没找到+200//从secondary 找到这段table+40//重新找main memory table，+200//发现不在main memory又换main memory+40//check主table+40//找到)

segment,如果对很重要的page size不一样的process，直接用segment。