操作系统第三章练习

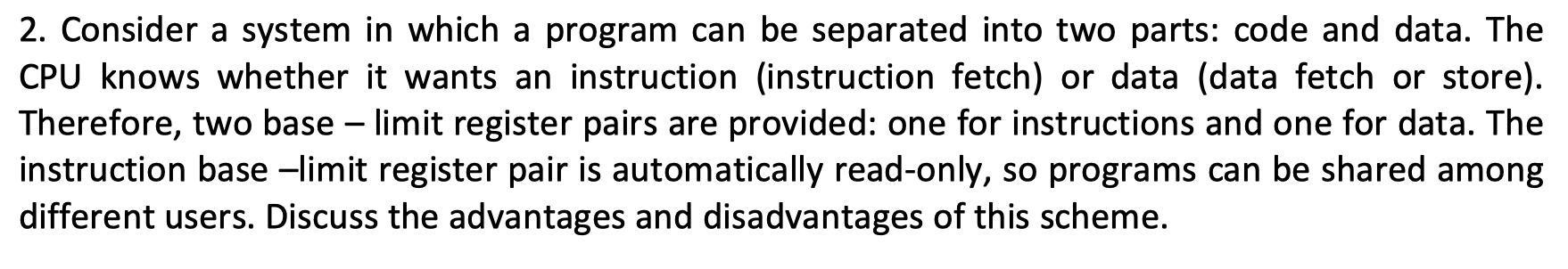
计科1602 郑宇飞 16281089



ANSWER：

逻辑地址所指向的并不是一个真实存在的地址，物理地址指向的是真实存在的地址。

逻辑地址由CPU产生，而物理地址由MMU产生。



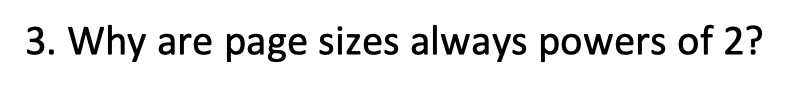
ANSWER：

优点：

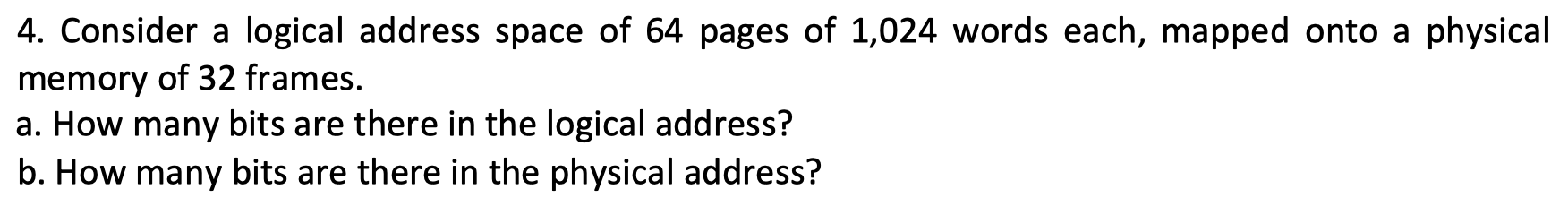
这种方式非常有利于数据和代码的共享，因为只需要保留一份在内存中，而且代码可以被其他的进程所共享。另外，这种方式有很高的安全性，由于保存的那一份代码是只读的，所以可以防止错误的修改。

缺点：

这种方法将原先联系密切，位置应该放置在一起的代码段和数据段给分隔开了。



ANSWER：可以提高硬件计算的效率，可以避免像传统算数那样很慢的计算，假如10011001，我们设置offset为4位，由于采用了2的整数次幂这种方式，那么直接提取1001就可以，得到page：offset = 1001 ：1001 ，可以非常快捷的计算。

．

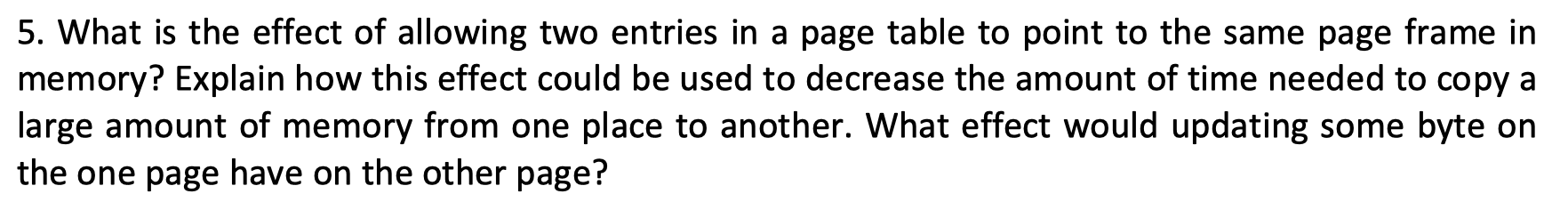
ANSWER:

a. 64=2^6,1024=2^10

所以共有2^16个逻辑地址，前6位表示页号，后10位表示页的偏移。

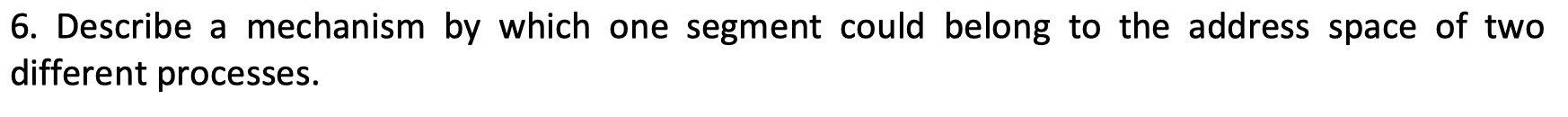
那么逻辑地址就有16位。

b. 32=2^5,所以共有2^(5+10)=2^15,其中前5位表示页块，后10位表示偏移，因此物理地址有15位。



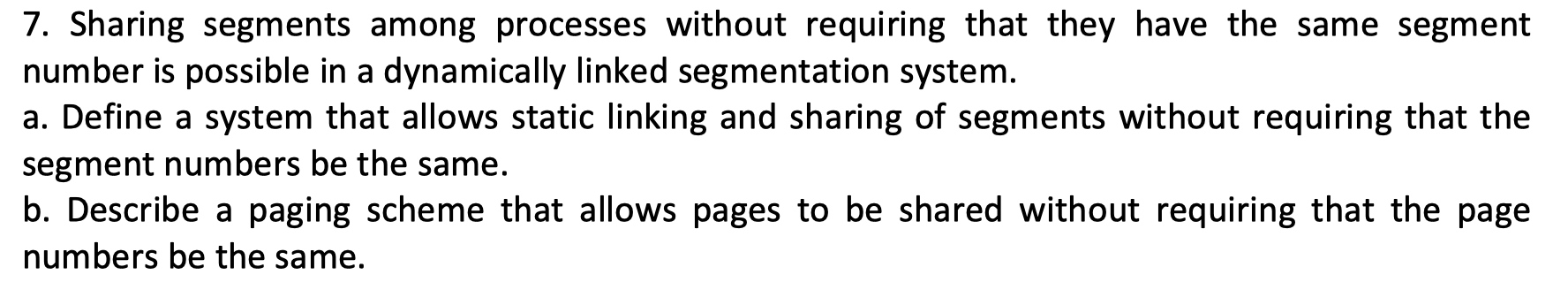
ANSWER:

通过让页表中的两项指向内存中的同一页框，可以使得用户共享代码数据尤其是在一些需要拷贝大量代码的场景中，这样做不仅节约了空间，同时也节约了时间。但是当用户更新的时候，会产生一些问题：一个用户对于其造成的修改，其他用户得到的信息也会受到修改，可能造成错误。



ANSWER:

这两个进程的段表必须有相同的边界限制指针，并且所共享的段的大小必须是一致的，也就是说这两个进程必须共享的是相同的一片物理区域。



ANSWER:

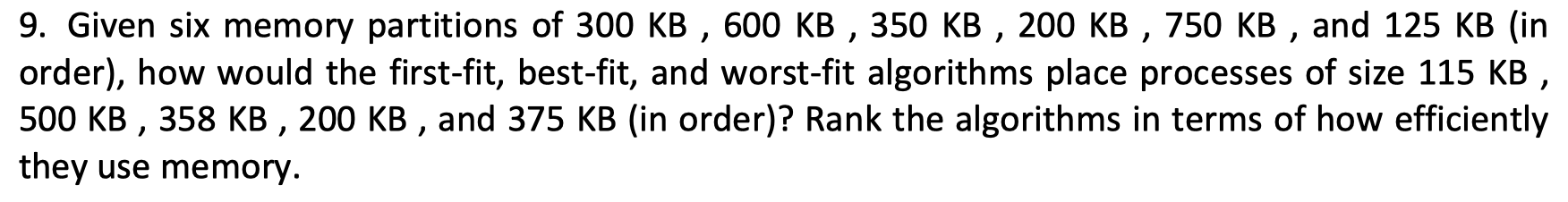
a与b都可以归于一类问题，如何在不需要段号和页号相同的情况下来达到共享的目的？MULTICS可以解决这样的问题。MULTICS通过联系4种寄存器，一个寄存器指示当前程序段，另一个指示段的边界，另一个指示堆栈段，最后一个指示段中的数据区域。通过这样的方法可以使得所有的参考通过一种间接的映射来映射到当前的段或者页号。所以只需要改变寄存器，就可以让不同的进程在不拥有相同页号或者段号的前提下执行相同代码。



ANSWER:

内部碎片是指明确属于某一个进程，但是没有被这个进程使用同时也无法被系统利用的内存空间。外部碎片指的是不属于某个进程，还没有分配出去，但是由于太小了分配的内存空间。

9



ANSWER:

.first-fit：会把300KB的内存空间分配给115KB，这样的话，会造成185KB的内部碎片；

Best-fit：会把200KB的内存空间分配给115KB，这样的话，会造成85KB的内部碎片；

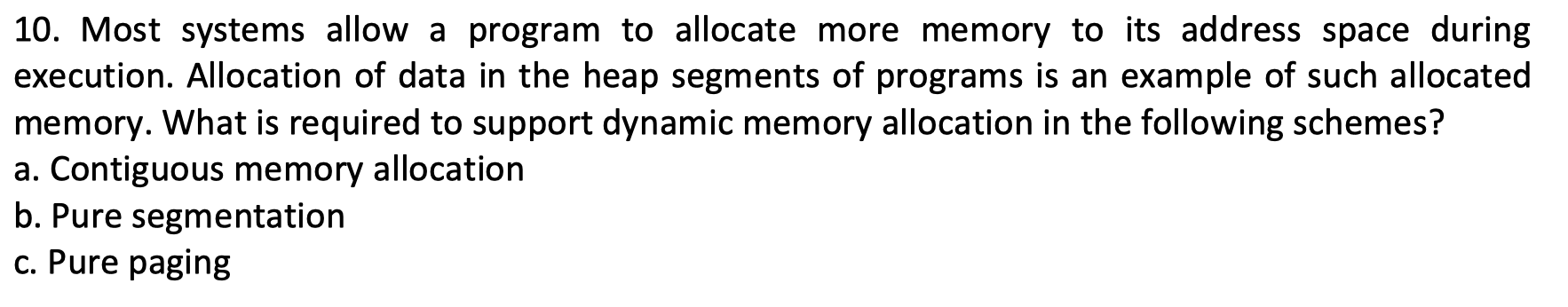
Worst-fit：会把750KB的内存空间分配给115KB，这样的话，会造成635KB的内部碎片。

如果在115KB，500KB，358KB，200KB，375KB中分配的话，

First-fit：会把115KB分配出去，不产生碎片；

Best-fit：会把115KB分配出去，不产生碎片；

Worst-fit：会把500KB分配出去，产生385KB的内部碎片。

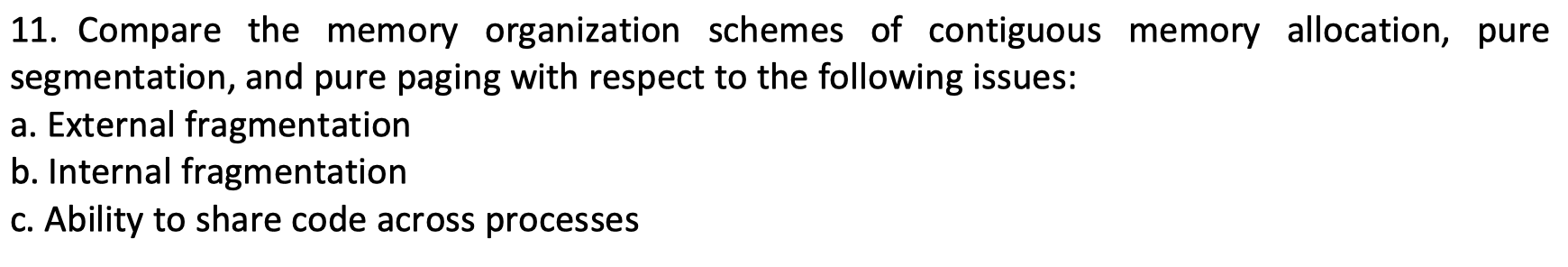


ANSWER:

a. contiguous memory allocation: 当程序在执行过程中发现无法分配更多的空间给这个程序了（因为题目要求要可以边执行边分配），那么可能需要重新决定一块内存空间，然后把整个进程重新分配到那个空间中去。

b. 在进程执行的过程中，如果分配给段的空间越来越多以至于空间不够时，也需 要类 似于a的策略，来重新分配段给这个程序。

c. 页的使用不同于上述两种，当发现不够分配时，可以直接增加新的页。

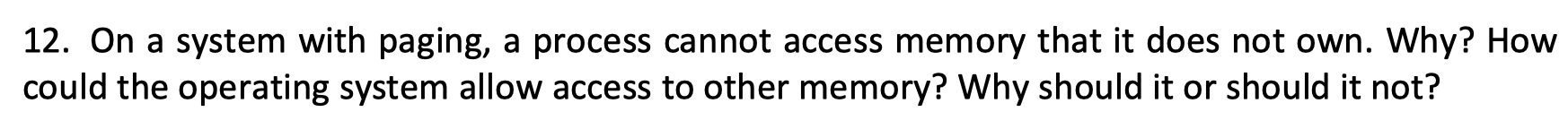


ANSWER:

连续内存分配会有外部碎片的出现，当空间连续分配给几个进程，而其中的一个进程完成时，会有外部碎片的产生；另外连续内存分配也不支持代码的共享。

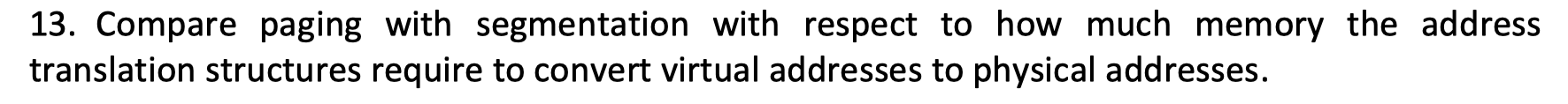
分段也会造成外部碎片的产生，当一个已经完成的进程的段被一个新的进程的段取代时，就会造成外部碎片。分段允许代码的共享，两个进程可以共享相同的代码段同时拥有自己的数据段。

分页不会造成外部碎片，但是会造成内部碎片，因为不可能保证每一个进程正好装满一个页。分页允许代码的共享。



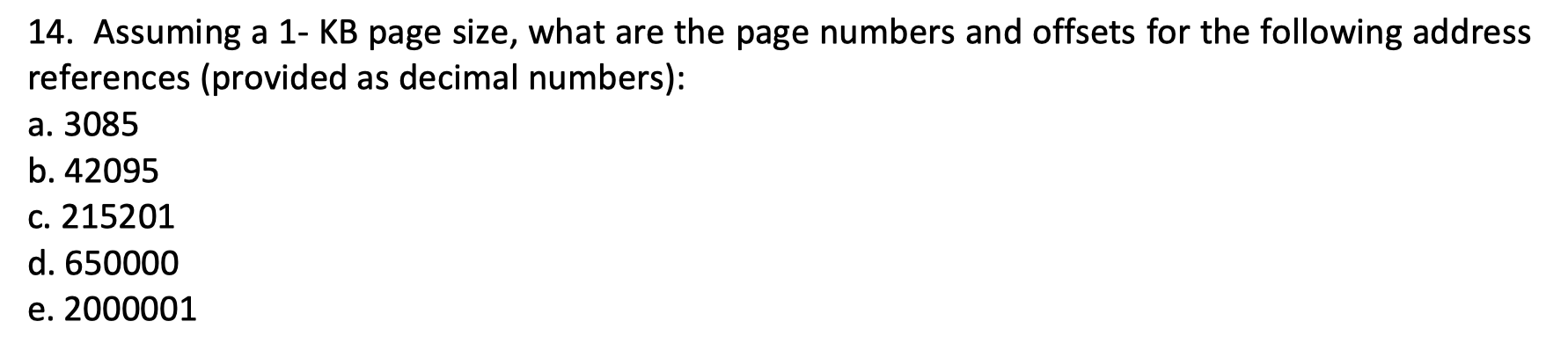
ANSWER:

进程对页的访问时通过页表来进行的，一个不属于这个进程的内存是不会写到这个进程的页表当中的，因此不可能访问到。如果要允许进程访问到其他的内存，可以将这个内存额外加入到进程的页表中，通过这样可以做到某一内存空间被多个进程所共享，实现进程之间通信。



ANSWER:

页式存储需要更多的内存来转化地址，而段式存储中每个段只需要两个寄存器，一个保存段的基地址，另一个保存长度；页式存储每一页都需要一个入口，这个入口提供了页所在的物理地址。



ANSWER:

1KB=2^10B，所以页内地址有10位。

3085转换成二进制为：110000001101，所以页号为11，偏移为0000001101，转换为十进制为：页号：3，页内偏移：13

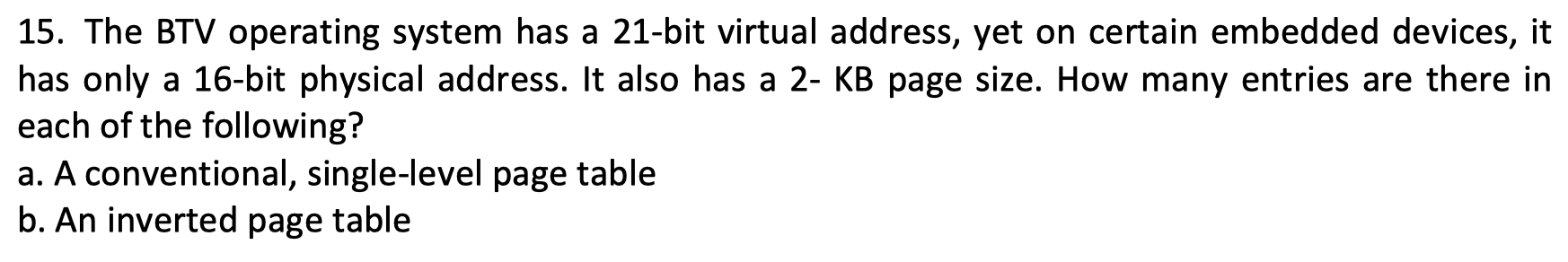
42095转换成二进制为：1010010001101111，所以页号为101001，偏移为0001101111，转换为十进制为：页号为41，偏移为111

215201转换为二进制为 110100100010100001 ，所以页号为11010010，偏移为0010100001，转换为十进制得：页号为210，偏移为161

650000转换为二进制为 10011110101100010000 ，所以页号为1001111010 页偏移为 1100010000 ，转换为十进制可得：页号634，页偏移：784

2000001转换为二进制为 111101000010010000001

页号：11110100001 页偏移： 0010000001，转换成十进制为：页号：1953，页偏移：129

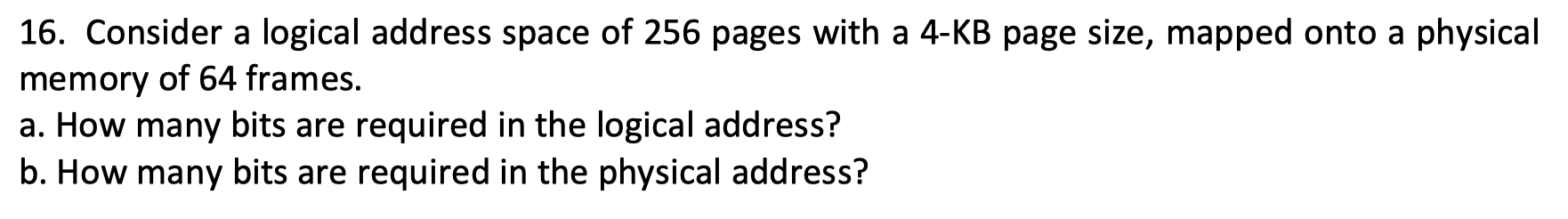


ANSWER:

传统页表使用的是页号，而倒置页表使用的是页框号：因为页的大小为2KB=2^11B，所以页的偏移为11位。

a. 由于逻辑地址21位，所以共有21-11=10位来为普通页表提供入口，故有：2^10个入口。

b. 由于物理地址16位，所以共有16-11=5位来为倒置页表提供入口，故有：2^5个入口。

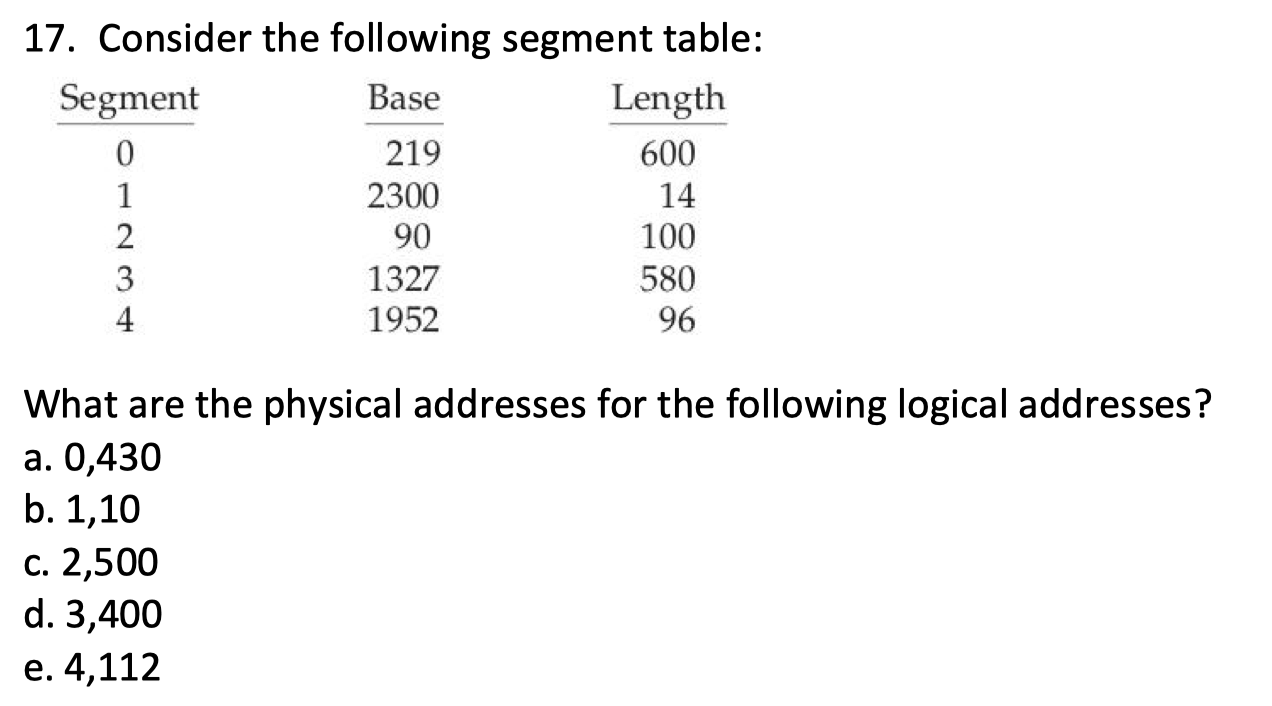


ANSWER:

256=2^8,64=2^6,4KB=2^12B

a. 8+12=20

b. 6+12=18



ANSWER:

0,430属于Segment0，物理地址为219+430=649 1,10属于Segment1，物理地址为2310；2,500不属于Segment2，因为length为100，500>100，所以越界了。

3,400属于Segment3，物理地址为 1327+400=1727

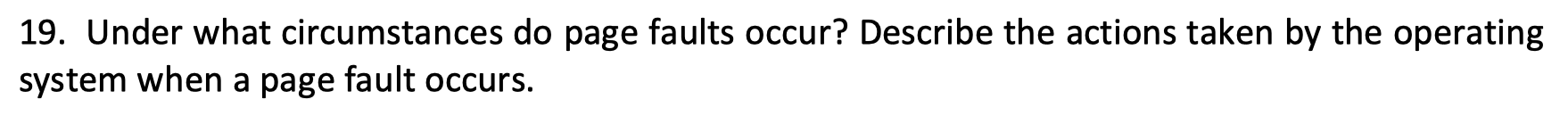
4,112不属于Segment4，因为112>96，越界了。



ANSWER:

通过这样的一种给页表分页的机制，页表可以变得更大，使内存的分配更简化（相比于分配可变大小的块，这样的固定页面大小的机制更能体现这种优势），而且也可以使页表把不常使用的一部分给交换出去。

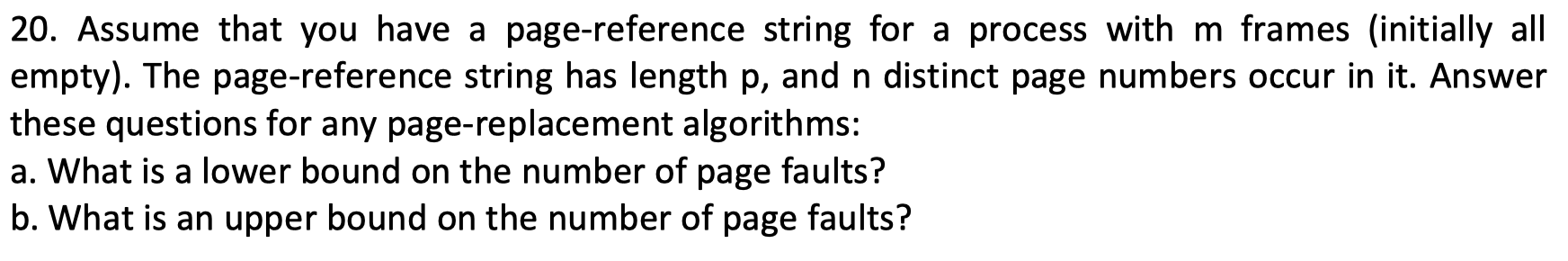
这样做的缺点是需要更多内存来保证地址的转换。



ANSWER:

缺页现象在当进程需要访问某一页面但是这个页面还未被调入内存时产生。

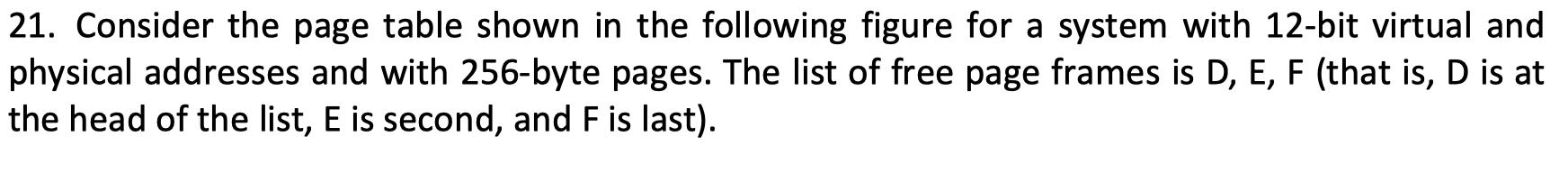
解决：当这个进程无效时，则直接抛弃这个进程， 不做其他的处理；当进程仍然是有效的时候，则先找到一个空闲的页框，然后发起I/O请求，找到指定的页面装入页框中，然后恢复进程的运行。

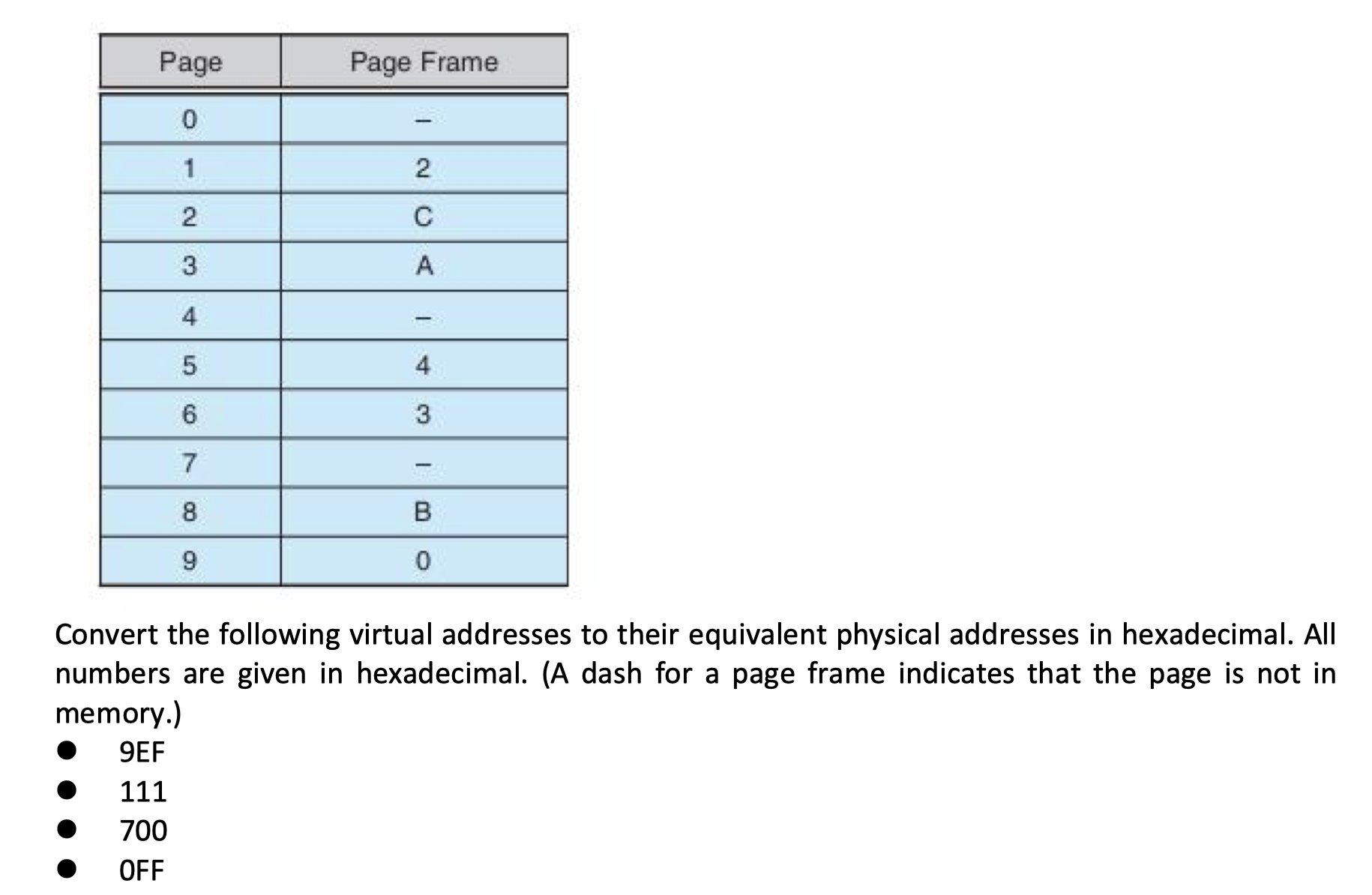


ANSWER:

a. n，至少在请求n个不同的页面时要发生缺页，所以最少为n次

b. p, 如果在字符串的每一位访问时都要不同的页，那么最多会发生p次





ANSWER:

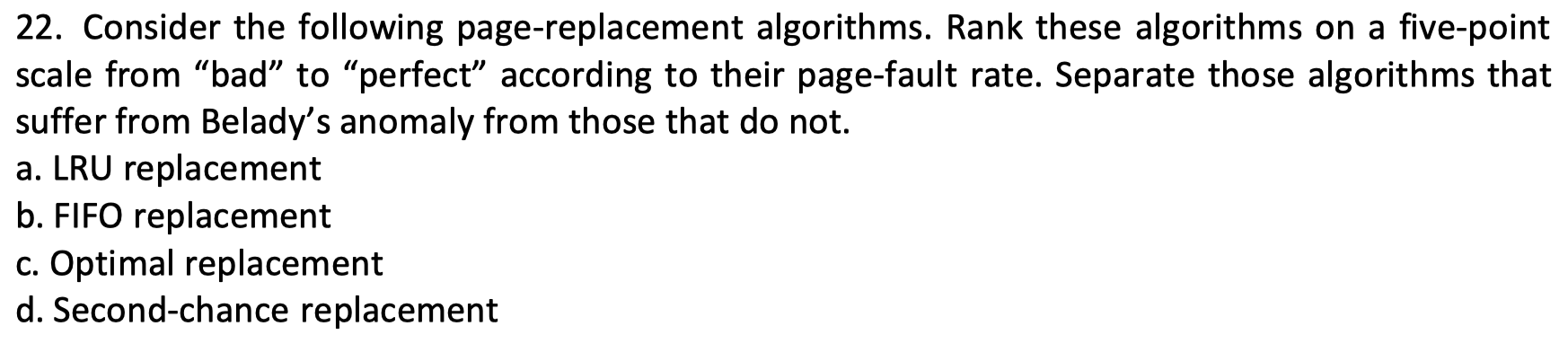
由于页的大小是256=2^8字节，所以需要8位来充当偏移地址，所以只需要看给定数字的首位即可。

9EF中9对应于表中的0，所以物理地址为0EF；

111中1对应于表中的2，所以物理地址为211；

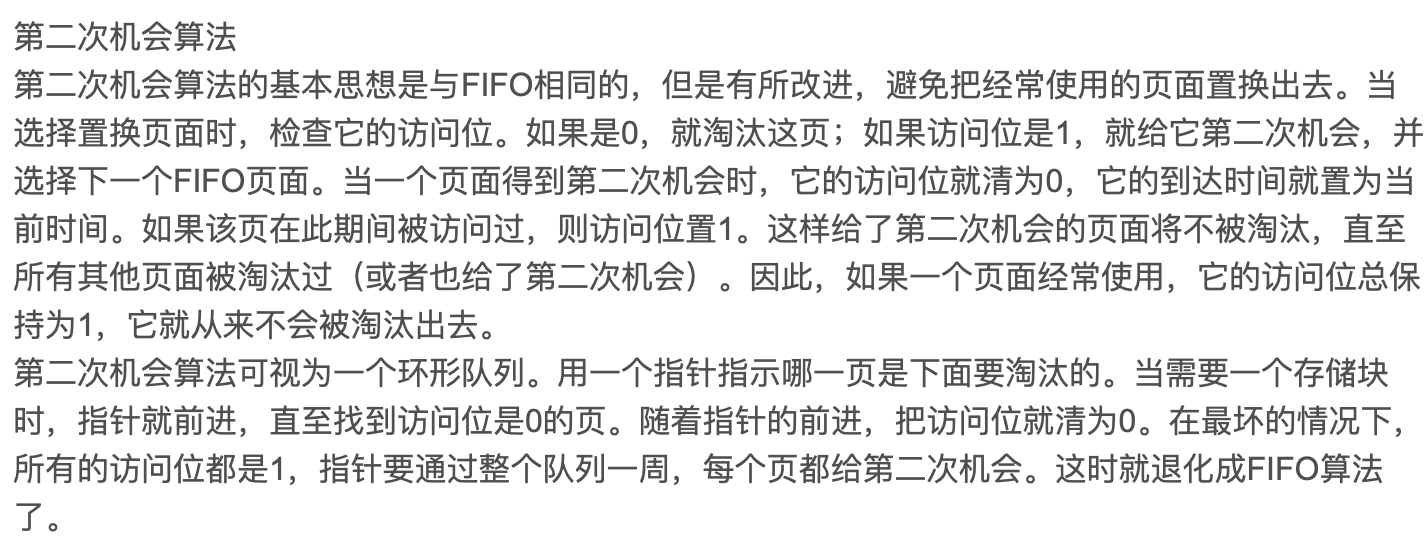
700中7对应于表中的-，没有找到，分配D，所以物理地址为D00；

0FF中0对应于表中的-，没有找到，分配E，所以物理地址为E00.



ANSWER:

Second-chance算法：

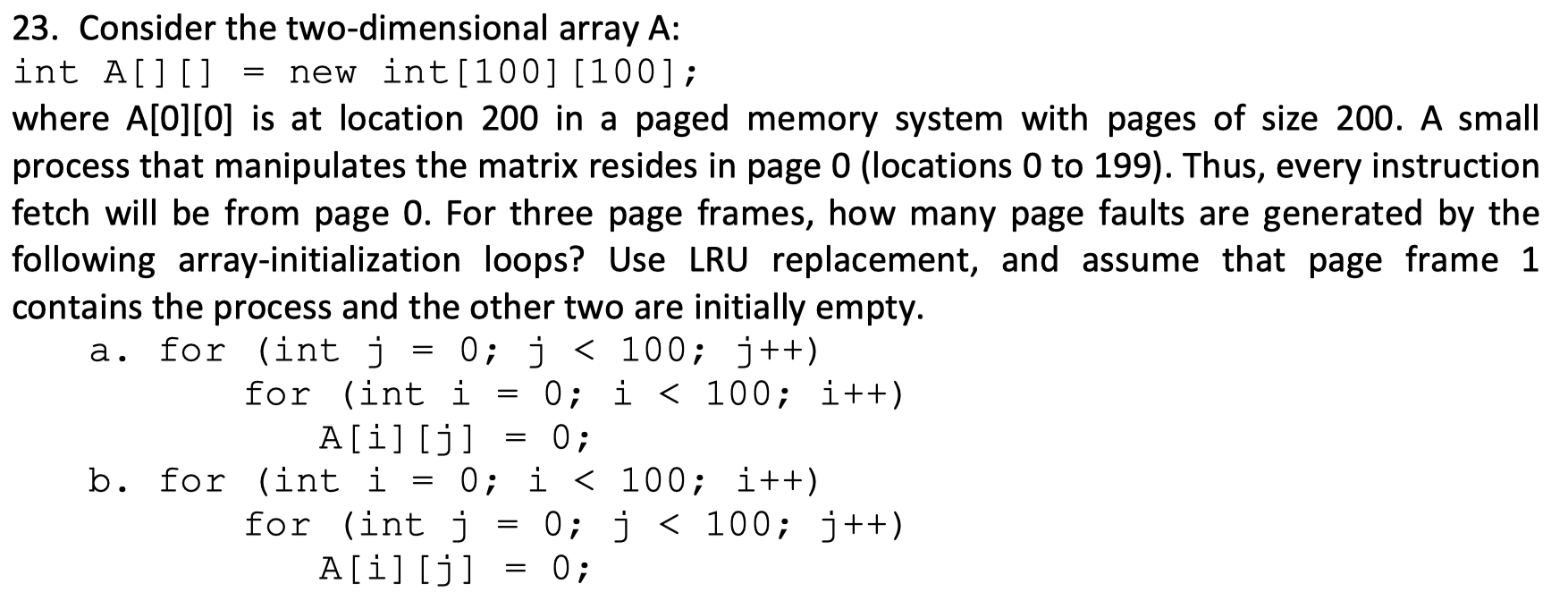


1: Optimal ，不会产生Belady’s anomaly；

2. LRU，不会产生 Belady’s anomaly;

3. Second-chance 会产生Belady’s anomaly；

4. FIFO，会产生Belady’s anomaly



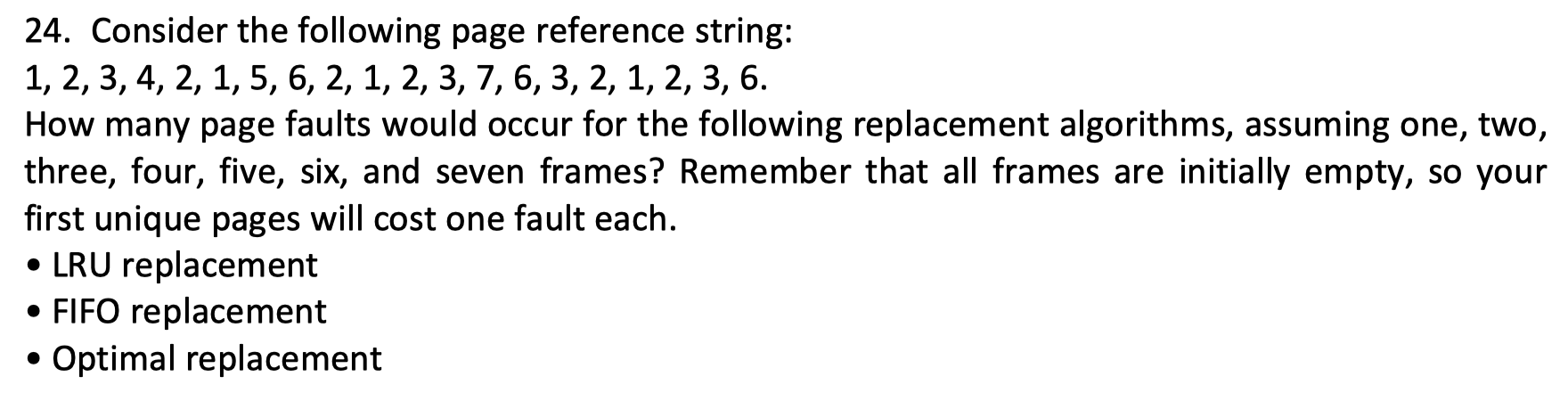
ANSWER:

a. 因为有三个页框，其中一个页框装的是指令，另两个页框装的是数组。根据LRU算法，如果发现数据缺页了替换的话一定会先把装指令的页框先替换了，另外由于两个页框装的是数据页，一共可以装400个数据，那么根据外层循环是A[i][j]中的i，每过4个指令就会替换一次。

因此替换总数为：100\*100/4+100\*100/4=5000

b. 这个相对于上题数据的替换频率变成了每400条指令替换一次，然后每次数据替换必然伴必然伴随着一次指令的页替换（因为LRU最近最久未使用的原则，在替换数据页的时候肯定会把装指令的那个页给替换掉）。

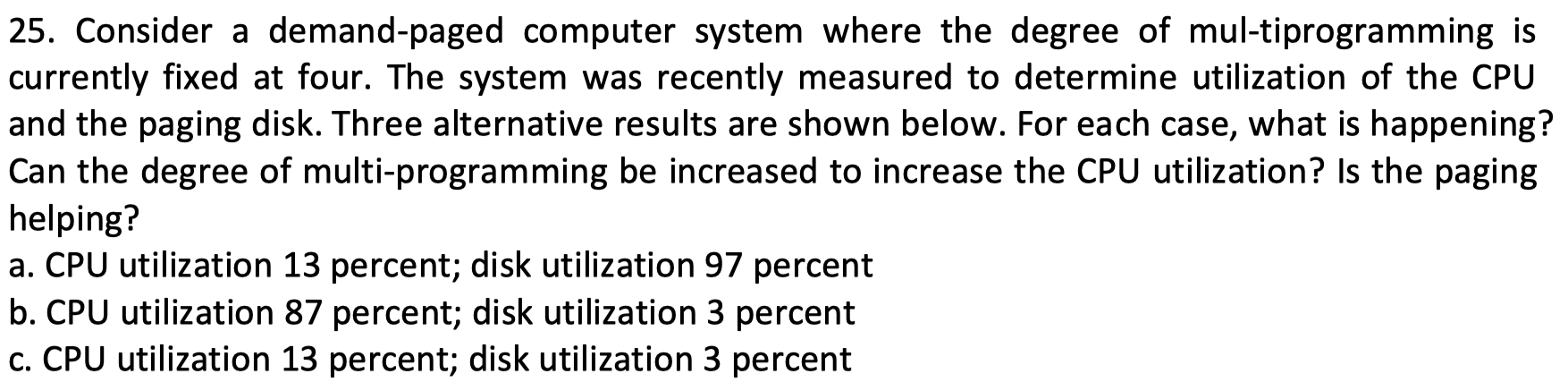
因此替换总数为：2\*100\*100/400=50



ANSWER:

结果如下所示：



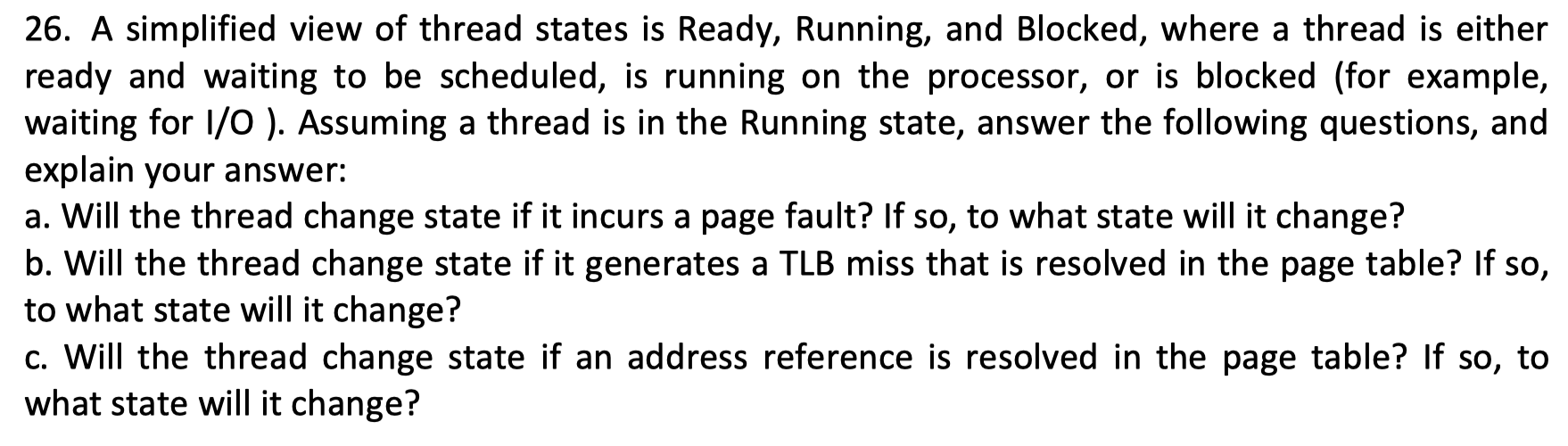


ANSWER:

a. CPU利用率较低，而磁盘利用率很高，可能出现了颠簸现象，不停的在置换页到内存中去，这种情况应该考虑减少多道程序数量。

b. 系统比较正常，应该适当增加多道程序数来提高磁盘利用率

c. 系统磁盘利用率较低，应该考虑提高多道程序数。

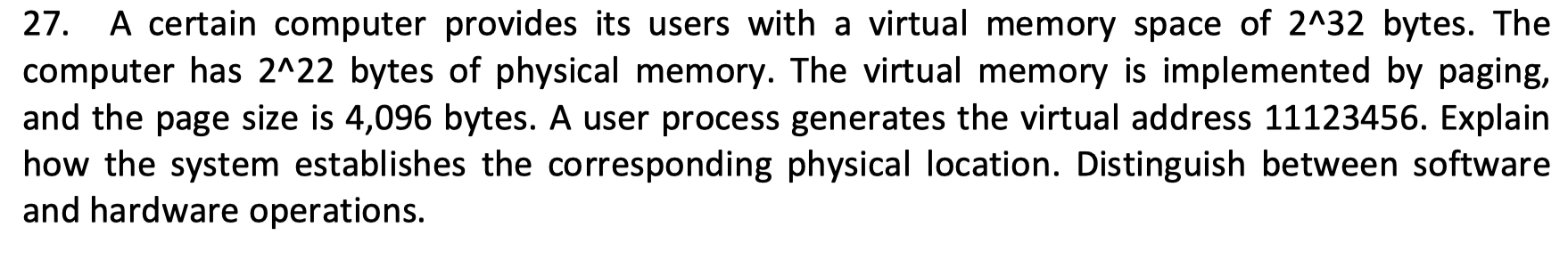


ANSWER:

a. 缺页会导致线程先进入blocked状态，等到所缺失的页重新装入了内存，才会重新回到running状态。

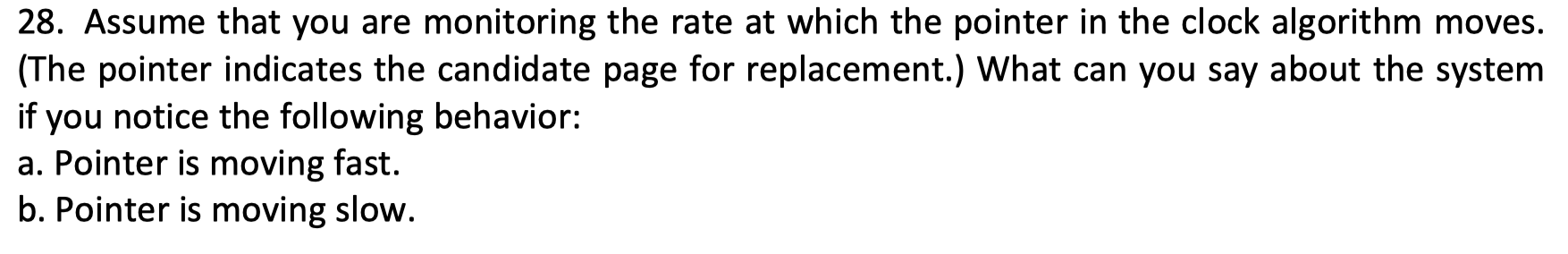
b. 不一定造成线程状态的改变，TLB只是相当于一种cache，如果没有找到会直接去页表里面寻找，不会造成状态的改变；但是如果内存里也没有找到就成为缺页的情况，就需要进入blocked状态了。

c. 不需要，因为如果地址引用在页表中被解决，就不需要多余的I/O操作来解决上述问题，因此不需要线程状态的改变。



ANSWER:

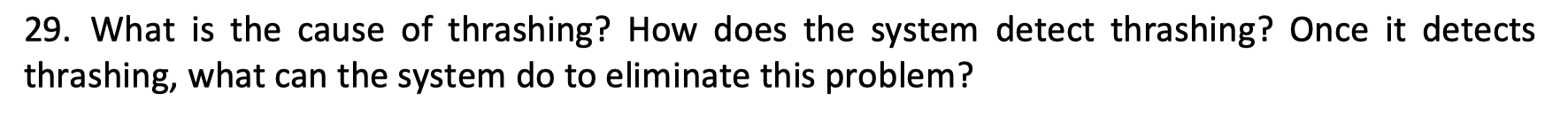
11123456转换为二进制为：101010011011 101100000000，由于页面大小是4096bytes，也就是2^12bytes，那么偏移地址就是12位，因此，页号：101010011011，页内偏移：101100000000，就可以通过页号对应的页框拼接上偏移地址得到物理地址，在这个过程中，硬件MMU参与了地址转换。



ANSWER:

a. 因为钟表的指针在不断的搜索访问位和修改位为0的值，如果指针移动的非常快，说明这些页框最近都被修改和访问过，也就说明系统正在频繁的调度页面。

b. 情况和上述恰恰相反，说明页面被调换的操作不频繁。



ANSWER:

当操作系统频繁替换调度一些比较频繁使用的页面，频繁的造成缺页现象，使得CPU的利用率非常的低，会造成颠簸现象。可以通过对比于多道程序的级别来评估CPU的利用率来实现对于颠簸的检测。 可以通过减少多道程序调度等级来消除颠簸。