作业二

一、

- 位图记录内存分配只需要每个内存单元占用一个比特，用0表示未分配，1表示已分配。而由题，共有$(128/n)M$ 个单元，每个单元占用1个比特，一个字节占8比特，故需要$(128/n)M/8$ 即$(16/n)M$个字节，即位图记录内存分配需要 $(16/n)MB$ 空间
- 链表记录的是每一块连续内存的信息。由题，$128MB$ 的空间数据和空闲交替排列，每段 $64KB$ ，那么一共有 $128MB/64KB = 1K$ 个段需要记录，也就是说链表有 $1K$ 个节点。每个节点需要 $32 + 16 + 16 = 64bit = 8B$ 的空间记录节点信息，那么链表一共要占用 $1K * 8B = 8KB$ 的空间。
- 从占用存储空间的角度看，位图法在 $n > 2048B$ ，即分配单元大于 $2KB$ 的情况下，其占用空间大小才会小于链表，否则位图比链表更占用空间。位图和链表各有优劣，位图法查询简单，链表与之相比需要遍历整个链表才可以得到空间的使用信息，但位图的容错能力不如链表，一个比特出错就会影响到内存分配状态。

二、

分别将题目中的内存空间编号

空闲区号	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
空闲区大小	10KB	4KB	20KB	18KB	7KB	9KB	12KB	15KB

- FirstFit从前到后寻找第一个能够满足需求的内存块。12KB请求会找到3号20KB空闲区，分配后3号变为8KB；10KB请求会找到1号10KB空闲区，分配后1号空闲区全部占用；9KB请求会找到4号18KB空闲区，分配后4号变为9KB。FirstFit会使位置靠前的空闲区优先被分配，产生较多碎片。
- BestFit寻找与请求最接近的空闲区进行分配。12KB请求会找到7号12KB空闲区，分配后7号空闲区全部被占用；10KB请求会找到1号10KB空闲区，分配后1号空闲区全部占用；9KB请求会找到6号9KB空闲区，分配后6号空闲区全部被占用。BestFit若能如上述情况找到恰好合适的空闲区就不产生碎片，否则容易产生较多碎片。
- WorstFit每次都寻找最大的分区来分配。12KB请求会找到3号20KB空闲区，分配后3号变为8KB；10KB请求会找到4号18KB空闲区，分配后4号变为8KB；9KB请求会找到8号15KB空闲区，分配后8号变为6KB。WorstFit不容易产生很小的碎片，但却会使大块内存被分割，当大请求的作业到来时可能没有足够的大段连续空间。
- NextFit每次从上一次分配空间的下一块空间开始查找，寻找第一个能够满足需求的空间。12KB请求会找到3号20KB空闲区，分配后3号变为8KB；10KB请求会找到4号18KB空闲区，分配后4号变为8KB；9KB请求会找到6号9KB空闲区，分配后6号空闲区全部被占用。NextFit不会像FirstFit那样让小碎片集中在内存前半部分，但也容易分割大块空闲区。

三、

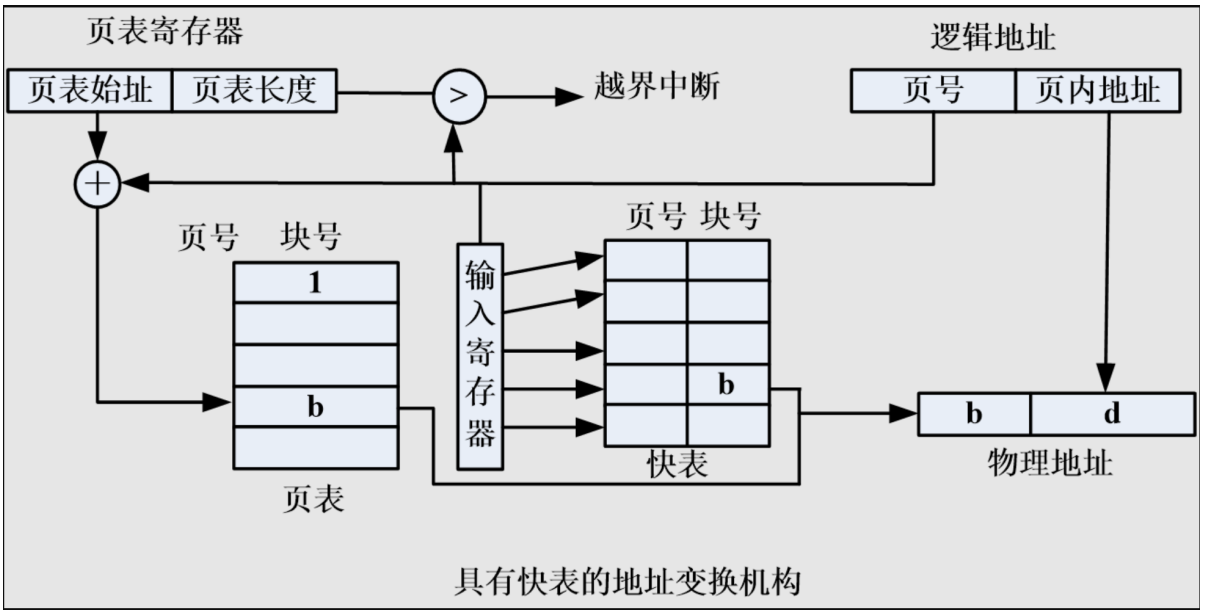
- **定义**：逻辑地址是操作系统的用户（程序）编写应用程序时所用的地址，物理地址是内存中实际的地址，地址映射指的是逻辑地址向物理地址转换的过程。
- 逻辑地址： 在编写程序时，可能会声明一个整型数组 `int arr[100];` 。在这里，数组的第一个元素 `arr[0]` 有一个逻辑地址，比如1000（这个值是随机的，由系统分配），而实际上它可能并不位于物理内存的第1000个位置。

- 物理地址： 如果有一个物理内存条，其中第一个存储单元的物理地址是 0x00，那么这个地址指向的是内存条上的实际物理位置。
- 地址映射： 如果程序中的逻辑地址1000被映射到物理地址0x1F4，则当程序尝试访问逻辑地址1000时，内存管理单元（MMU）会自动将此请求转换为访问物理地址0x1F4的请求。
- 对32位的MIPS体系结构而言，每一个进程都拥有独立的4GB逻辑地址空间，每一个区域的地址空间有其特定的用途，kuseg、kseg0、kseg1和kseg2都有各自的地址映射方式。MIPS体系结构广泛应用于不同硬件条件的嵌入式系统，具体机器的物理内存和物理地址不尽相同，但逻辑地址到物理地址的映射按区域基本确定，有通过MMU转换的区域，也有直接高位清零转换的区域。

四、

页式存储管理的目的是高效利用内存，并实现大逻辑空间映射小物理空间。要使用页式内存管理，就需要存储一张页表，用于记录逻辑页向物理页的映射。由于查页表会增加一次访问内存，影响访问性能，人们设计了快表，即页表的cache，在查找页面映射时先查快表，若命中则无需查页表，若没有命中再访问页表，并做快表替换。

具体工作流程入下图所示：



上图是拥有一级页表和快表的系统的地址映射机制。逻辑地址划分为页号和页内偏移两个部分，得到逻辑地址时先在快表中查找页号，如果命中则直接根据其对应的物理页号再加页内偏移找到物理地址，若没有命中则在内存中查页表，将查到的页表项替换进快表，然后根据物理页号加页内偏移访问物理地址。

五、

1. **中断触发**：当CPU尝试访问一个不在物理内存中的页面时，硬件自动触发一个缺页中断。
2. **内核态转换**：操作系统接管控制权，转入内核态，开始处理缺页中断。
3. **保存现场**：操作系统保存引起中断的进程的当前状态，包括程序计数器、寄存器内容等，以便中断处理完毕后能恢复到中断前的状态。
4. **查找页面**：操作系统检查该虚拟页是否有效，即它是否在进程的地址空间内。如果不在，则是一个非法访问，可能导致进程终止。如果页面有效，系统将进一步查找该页是否已经在磁盘上（例如，交换空间或者映射文件中）。
5. **选择牺牲页**：如果物理内存已满，操作系统需要从物理内存中选择一个页面来替换，通常通过某种页面替换算法（如LRU、FIFO等）进行选择。
6. **写回磁盘**：如果被替换的页面被修改过（脏页），则需要将其写回到磁盘上的适当位置。

7. **从磁盘读取页面**：操作系统将缺失的页面从磁盘读入到刚才选择的物理内存页框中。
8. **更新页表**：操作系统更新当前进程的页表，将新载入的页面地址映射到相应的虚拟地址上，并调整相关页面的访问权限和状态位。
9. **恢复现场**：操作系统恢复在缺页中断发生时保存的进程状态。
10. **重试指令**：操作系统让CPU重新执行引发缺页中断的那条指令。由于现在缺失的页面已经被载入到物理内存中，这次访问应该能成功进行。

整个处理流程是由操作系统的内存管理子系统完成的，旨在透明地为进程提供一个看似连续和巨大的地址空间，即使物理内存的实际大小远小于进程的虚拟地址空间。

六、

1.

一级页表的缺陷在于，当逻辑地址空间很大的时候，页表本身会占用很多内存，查页表的过程也很耗费时间。因此人们设计多级页表机制，即为页表再设置页表，然后实行动态页表调入，即只将当前要用的页表项调入内存，其余的需要用时再调入。多级页表的优势在于节省了内存中存储页表的空间。

2.

页面大小为16KB，需要用14位来表示，即页内偏移占到地址的14位，对虚拟地址而言剩余24位。设剩余的24位中假设有n位用于查第一级页表，(24-n)位用于查第二级页表。n位页表域可以表示 2^n 个虚拟页，每个虚拟页占用4个字节的页表项，则页表一共占用$2^n * 4B = 2^{n+2}B$空间，这些由页表占用的空间一共排在 $2^{n+2}B/16KB = 2^{n-12}$ 个页面上，而这些页面需要由(n-12)位来表示，也就对应的是页目录的(24-n)位。此时有n-12 = 24-n，容易得n=18。

因此虚拟地址中由6位表示第一级页表，18位表示第二级页表，14位表示页内偏移。

七、

1.

由题可知，一共有512个可能得页面需要装载。

当物理页框数大于512时，那么不管哪种算法都不会产生缺页中断，LUR、FIFO和Clock算法得效果是相同的，他们全未被使用过。

当物理页框数不足512个时，那么三种算法的效果也是几乎等同的。

- FIFO：在首先装满所有物理页框之后，对于FIFO而言，在需要淘汰物理页的时候会从首先装入，即周期性循环中最靠前的物理页开始淘汰，不及该循环中的物理页再次出现，它一定会被淘汰，从而FIFO在应对这种情况的循环时，除了那个随机出现的页面之外，所有页面都会缺页。
- Clock：在首先装满所有物理页框之后，对于Clock算法而言情况类似，Clock在FIFO的原则之上给页面“第二次机会”，即如果一个页面装入后被再次访问过，就“再给它一次生存的机会”，但在上述情况中等待被淘汰的物理页大概率均不会有装入内存后的第二次访问机会，随机出现的页面可能会有再次访问从而享受到Clock的“庇护”，但大部分页面大概率都会缺页。
- LRU：对于LRU而言，它的效果也和前两者几乎等同，这是因为LRU淘汰最久没有用到的页面，在题目所述的循环中最久不用的页面和最先到来的页面几乎是等价的，因此LRU也会面临几乎所有页面都缺页的情况。

故面临题目所述的周期性页面访问时，LRU、FIFO和Clock的效果几乎一致，在页框足够时不会产生任何缺页，但在页框不足时几乎每次访问都会缺页，缺页率接近100%。

2.

可以使用“FILO 先进后出”（或后进先出）算法，其性能会大大提升。在最初的500个物理页被占满之后，到来的第500号虚拟页会替换掉499号，第501号又会替换500，等等，在此第500-511号虚拟页会全部缺页，但在新的循环到来时，前499个虚拟页会全部命中，夹在两次循环之间的随机页也会有500/512的概率命中，缺页率降至约为12/512=2.34%，远远优于LRU、FIFO和Clock。

八、

一个内存字32位，即4字节。总内存字数量：$128MB/4B = 2^{25}$个内存字

读写一个内存字需要10纳秒，总共$T = 2^{25} * 10 * 10^{-9} \approx 0.336$秒