lru算法：

该算法在内存中页面填满的情况下，选择替换的是最近最久未使用算的页面。可以将所有的页面情况先大体分为两类。一类是内存中已经存在了当前的页面，另一类是内存中还没有存在当前的页面。对于前一种情况，我们需要做的就是将内存中已经存在的当前页面进行位置上的转换来完成。这里使用将该页面置于队尾的方法来实现对于队列中顺序的更新，保证队列的队首是最近最久未使用的页面，这样在选择的情况下，永远只需要替换队首就可以方便地完成页面的置换。对于后一种情况，我们需要再判断此时页面是否已经达到上限，没有空缺可以用来填写新的页面。如果页面数没有达到上限，那么只需要将新的页面直接添加在队尾即可。如果页面的数目已经达到了上限，那么就需要要进行替换。由于在之前的工作过程中已经保证了队首的页面即为我们要选择替换的页面，所以我们可以直接将原本队首的后一个页面置为队首，然后将新的页面添加在页尾即可。

nru算法：

该算法也被称为简单的Clock置换算法。主要思想是对于页面给予两次机会，通过访问位来进行最终置换页面的判断。第一种情况，先对于页面内是否存在当前的页面进行判断，如果已经存在，那么只需要将这一页的访问位设置为1，不需要对顺序做出改变。第二种情况，当页面内不存在当前页面，这种情况下，如果内存中的页数还未达到上限，那么只需要将当前的页面接在整个队列中。具体过程可以分为先将队首和队尾断链，随后将当前的页面和断开的队首和队尾连接起来，然后更新队首和队尾。如果页面数目已经达到了上限，那么就需要对整个队列中的页面进行筛选，通过不断的循环，找出队列中第一个满足访问位为0的页面，然后将其取出并更换。过程中既需要将取出的部分重新连接（还需要判断是否为队首），还需要将新加入的部分接在队尾，并更新队首和队尾，然后将整个队列重新连接起来。

生成页面部分的思想和算法：

由于真正的程序具有局部性，所以在生成数据的过程中，我们需要对这个情况进行模拟。这里使用的方式是选择4个页面作为经常重复的页面，另外12个作为可能执行到的页面来重复。整个过程大概可以被概括为一个重复的大循环中内部有四个if和else的判断，分别会执行不同的内容。四个判断即为我们选择的四个经常重复的页面，而进入判断条件的内容则在选择的12个可能语句中。这样生成的序列更加可能符合真实情况下的程序，所以在执行的过程中，当页面数达到一定数量而还没有达到最大的情况下，其命中率已经基本和页面为32的情况下的命中率一致。

感想：

在书写这个程序的过程中，我认为真正困难的并不在于算法的实现。算法的选择部分的实现基本都已经可以根据逻辑来进行书写，整体内容的实现都可以选择使用链表的形式来进行实现。可能对于有些情况比如fifo这样的算法，链表并不能表现出优势，但是当涉及到nru这样的算法时，就能够真正感受到使用链表的便捷，和更强的实现能力。

程序真正的精华在于如何生成序列。序列的内容需要尽可能符合程序的真正情况。比如在实际的程序运行中，程序具有局部性，这样的性质使得程序可以在运行的过程中先将一部分的内容调入内存然后即可运行，随后在之后的运行中，再将需要的页面调入内存或者与内存中已有的页面进行对换。而和哪个页面进行对换就取决于我们的算法，不同的算法会选择不同的页面作为替换的对象，算法所选出的页面不同也会影响整体的效率。Opt算法是理论上最好的算法，但是在实际实现的过程中并不可能达到这样的效率（即页面的替换次数最少，因为无法预知之后的情况）。平均情况下，效率最差的应该是FIFO因为该算法基本没有对页面的之后使用做任何的参考，所以页面的替换很可能会变得冗余而无法达到较高的效率。

整个实验的过程中实际已经在尽可能的对于实际生活中操作系统进行模拟，但是仍然有着很多的东西仍然没有被完善的考虑。比如调入虚存中的并不一定是一个进程的所有页面，在页面调入虚存的过程中可能还存在着选择，关于这些问题在程序中暂时还没有进行实现，不过可以先对于这些问题进行思考，构思更加接近真实的一种模拟情况。