内存管理实验

一、实验目的

通过模拟实现按需调页式存储管理的几种基本页面置换算法，了解虚拟存储技术的特点，掌握虚拟存储按需调页式存储管理中几种基本页面置换算法的基本思想和实现过程，并比较它们的效率。

二、实验内容及要求

2.1 生成内存访问串：首先用 srand()和 rand()函数定义和产生指令地址序列，然后将指令地址序列变换成相应的页地址流。

2.2 设计算法，计算访问缺页率并对算法的性能加以比较

a.最优置换算法(Optimal)

b.最近最少使用(Least Recently Used)

c.先进先出法(First In First Out)

2.3 要求：分析在同样的内存访问串上执行，分配的物理内存块数量和缺页率之间的关系；并在同样的情况下，对不同置换算法的缺页率比较

三、数据结构介绍

int order[ORDER_NUM]: 指令序列

int page[ORDER_NUM]: 指令序列对应的页地址流

int distance[block_num]: OPT 算法中存储内存块中相应页的距离

int used[PAGE_NUM]: 在 LRU 算法中，存储页的在物理内存块中的时间

int fre[PAGE_NUM]: FIFO 算法中，存储页在物理内存块中的时间

四、主要函数介绍

1.int is_exit(int page):检测页面是否在内存块中。若在，返回在内存块中的下标；否则，返回-1

2.int find_empty():寻找内存块中是否有空余。若有，返回空内存块下标；否则，返回-1

3.void init_ins():从地址 0 开始，通过随机数产生若干个地址，其中 70%的指令是顺序执行的，10%的指令是均匀分布在前地址部分，20%的指令是均匀分布在后地址部分。

具体操作：使用随机数 rate=rand()%10 产生 0~9 之间的随机数，rate≤6

时，顺序执行;rate=7 时，向前生成地址，tmp=rand()%(a+1);rate≥8 时，先后生成地址，tmp=rand()%(ORDER_NUM-a)+a。之后再转换为对应的页地址流

4.void OPT()伪码描述:

```
for i in range(ORDER_NUM):
   if 请求页号 page[i] is not in phy_memory:
      if phy_memory is not full
         page[i]进入内存
      else://进行替换
         求出内存块中页号的最大距离
         phy_memory[max_index]=page[i]
```

5.void LRU()伪码描述:

```
for i in range(ORDER_NUM):
   if page[i] is in phy_memory:
      used[]++;
      used[i]=0;//刚出现过，所以置 0
   else:
      if phy_memory is not full:
         page[i]进入内存,used[]++;
      else://进行替换，选择最近最少使用页面
         phy_memory[max_index]=page[i]
         used[max_index]=0,used[]++;
```

void FIFO()伪码描述:

```
for i in range(ORDER_NUM):
   if page[i] is in phy_memory:
      fre[]++
   else:
      if phy_memory is not full:
```

page[i]进入内存

fre[]++

else://进行替换，选择最先进来的页面

phy_memory[max_index]=page[i]

fre[]++

六、结果展示(生成 400 条指令地址)

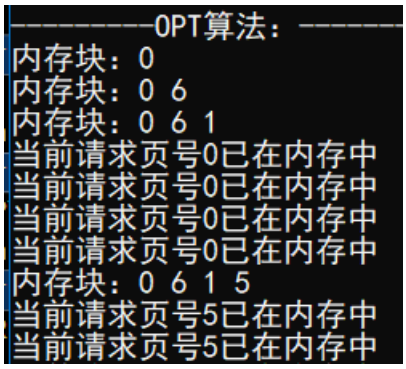
首先生成地址序列，并转换为页地址流

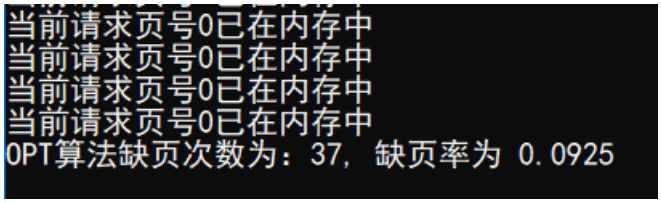
```
指令序列为:
  0 133  29  11   8   0   1 109 110 111 112 113 114 115 388 389 397 398 399 315
 65  13  11   5   1   0 284 364 365 366 390 395 398 399 384  18  12   6   4   3
  2   0 246 247 248 249 295 296 297 298 299 189 148  98  96  26   8   6   3   2
  1   0   1   2   3   4   5   6 369 370 371 372 373 382 243  89  44   7   1   0
  1   2   3   4   5   0   1   2   3   4   1   0   1   0   1 346 384 385 386 398
399  63  56  25  13   3   1   0 255 256 257 258 259 260 263 264 326 327 234  65
 44  37  16  11   6   5   4   1   0   1   2 359 360 370 371 372 391 392 393 395
396 397 398 398 356 193  52  43   1   0   1 142 143 144  87  38  26   3   1   0
397 398 399  60  19   6   1   0   1   2   3   4   5   6   7   8   9  10  11  12
  3   2   0   1   2   3 180 181 182 183 164 143  87  50   5   2   1   0   1  28
 29  30 241 242 243  38   3   2   1   0 176 177 178 179 180 394 395 399 175 140
109  24  21   2   1   0   1   2 197 198 199 366  88   2   1   0   1   2 298 275
 95  24  20   7   5   0   1   2   3   4 284 285 286 287 341 342 343 339 286  95
 86  48  11   9   2   0   1   2 162 163 219 220 280 332 333 334 335 336 337 338
192  72  51  29   2   1   0   1   2   3   4   5   6 381 382 383 384 385 386 387
388 398 399 109  83  57  37   7   4   0   1   2   1   0   1   0  24  25 328 329
332 391 392 393 377  87  85  76  50  30  20  18   4   3   0   1   0   1 368 369
398 229  85   0   1   2   3   4  85 279 279 280 281 282 324 345 346 397 398 399
 35  32  13   6   0   1   2 261 354 355 356 357 358  65  35   4   3   0   1   2
  3   4   5   6 233 306 307 308 309 310  16  14  10   6   5   0   1   2   3   4
请输入每页指令数： (1-ORDER_NUM)
20
页面地址流为:
Page  0:   0   6   1   0   0   0   0   5   5   5   5   5   5   5  19  19  19  19  19  15
Page  1:   3   0   0   0   0   0   0  14  18  18  18  19  19  19  19  19   0   0   0   0   0
Page  2:   0   0  12  12  12  12  14  14  14  14  14   9   7   4   4   1   0   0   0   0
Page  3:   0   0   0   0   0   0   0   0  18  18  18  18  18  19  12   4   2   0   0   0
Page  4:   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  17  19  19  19  19
Page  5:  19   3   2   1   0   0   0   0  12  12  12  12  12  13  13  13  16  16  11   3
```

```
页面地址流为:
Page  0:   0   6   1   0   0   0   0   5   5   5   5   5   5   5  19  19  19  19  19  15
Page  1:   3   0   0   0   0   0  14  18  18  18  19  19  19  19  19   0   0   0   0   0
Page  2:   0   0  12  12  12  12  14  14  14  14  14   9   7   4   4   1   0   0   0   0
Page  3:   0   0   0   0   0   0   0   0   0  18  18  18  18  18  19  12   4   2   0   0   0
Page  4:   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  17  19  19  19  19
Page  5:  19   3   2   1   0   0   0   0  12  12  12  12  12  13  13  13  16  16  11   3
Page  6:   2   1   0   0   0   0   0   0   0   0   0  17  18  18  18  18  19  19  19  19
Page  7:  19  19  19  19  17   9   2   2   0   0   0   7   7   7   4   1   1   0   0   0
Page  8:  19  19  19   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Page  9:   0   0   0   0   0   0   9   9   9   9   8   7   4   2   0   0   0   0   0   1
Page 10:   1   1  12  12  12   1   0   0   0   0   8   8   8   8   9  19  19  19   8   7
Page 11:   5   1   1   0   0   0   0   0   9   9   9  18   4   0   0   0   0   0  14  13
Page 12:   4   1   1   0   0   0   0   0   0   0  14  14  14  14  17  17  17  16  14   4
Page 13:   4   2   0   0   0   0   0   0   8   8  10  11  14  16  16  16  16  16  16  16
Page 14:   9   3   2   1   0   0   0   0   0   0   0   0   0  19  19  19  19  19  19  19
Page 15:  19  19  19   5   4   2   1   0   0   0   0   0   0   0   0   0   1   1  16  16
Page 16:  16  19  19  19  18   4   4   3   2   1   1   0   0   0   0   0   0   0  18  18
Page 17:  19  11   4   0   0   0   0   0   4  13  13  14  14  14  16  17  17  19  19  19
Page 18:   1   1   0   0   0   0   0  13  17  17  17  17  17   3   1   0   0   0   0   0
Page 19:   0   0   0   0  11  15  15  15  15  15   0   0   0   0   0   0   0   0   0   0
请输入内存块数量(1-page_num)： 9
```

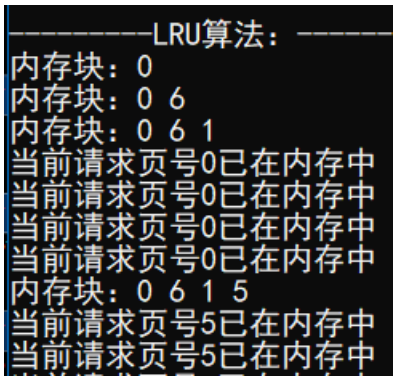
OPT 算法展示：会输出请求页号以及内存块中的信息，并在最后输出缺页率

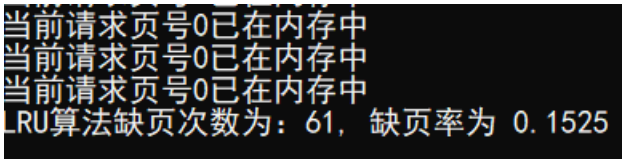
中间输出信息较多，只展示开头和结尾



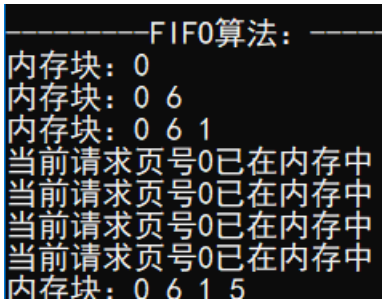


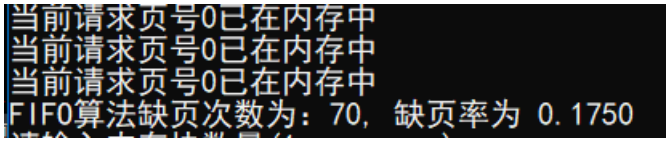
LRU 算法结果展示：





FIFO 算法结果展示：

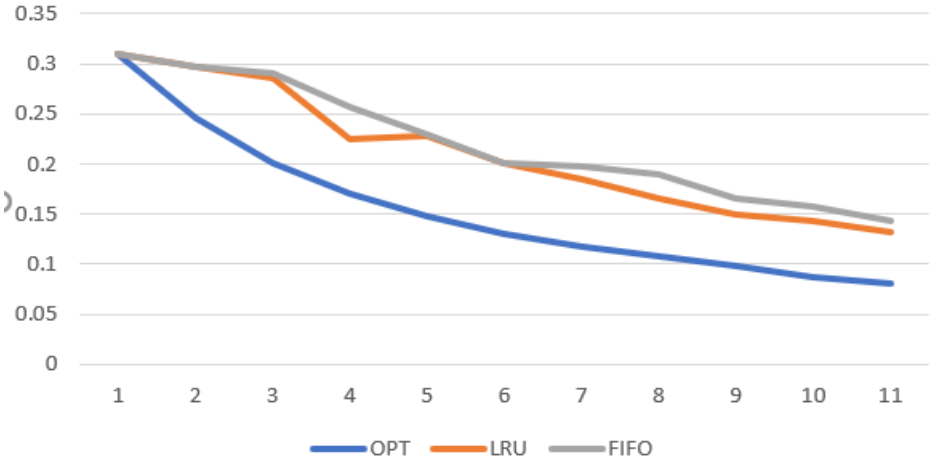




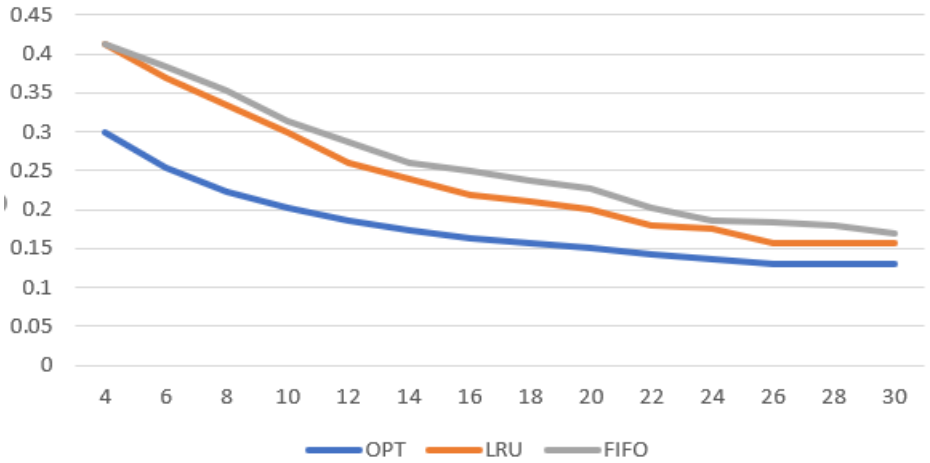
七、性能比较：调整主函数结构

```
while (1)
{
    printf("请输入内存块数量(1-page_num): ");
    cin >> n;
    initMemory(n);
    printf("\n---------OPT算法: ---------\n");
    OPT();
    /*initMemory(n);
    printf("\n---------LRU算法: ---------\n");
    LRU();
    initMemory(n);
    printf("\n---------FIFO算法: ---------\n");
    FIFO();*/
}
```

400 条指令地址，每页 20 条



300 条数据，每页 6 条



物理内存块数量与缺页率之间的关系：

a. 随着内存块数的增加，缺页率大体上逐渐下降

b. 当内存块数量增大到一定值时，缺页率变化趋缓直至不再变化

不同算法比较：

a. 由于 OPT 算法提前判断了未来情况，所以缺页率低也是意料之内的。

b. 总体而言，LRU 算法优于 FIFO 算法，但不排除个别情况下 FIFO 优于 LRU

c. 当内存块接近页数时，三种算法的缺页率均逐渐稳定在某个的值

八、实验总结：

本次实验通过编程实现 OPT、LRU、FIFO 三种经典的算法模拟了内存管理的过程，对于算法的运行过程以、原理以及算法之间的性能有了更深入的理解和更直观的感受。