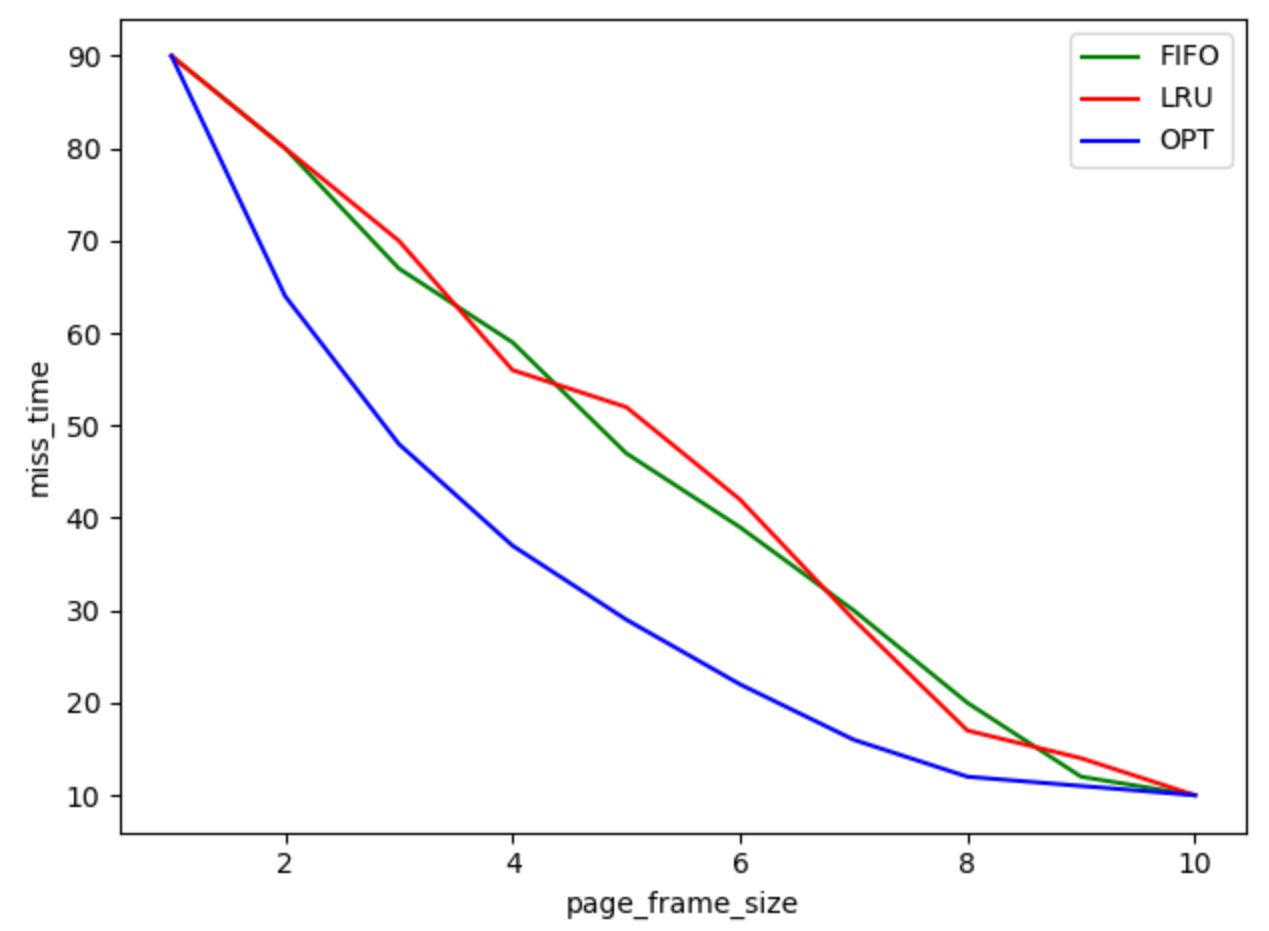
1. 叙述缺页中断的处理流程。

纯分页系统不会有缺页中断，只有虚拟存储系统才会产生缺页中断。当虚拟存储访问的页面不在内存中时，会引发缺页中断，其流程如下：

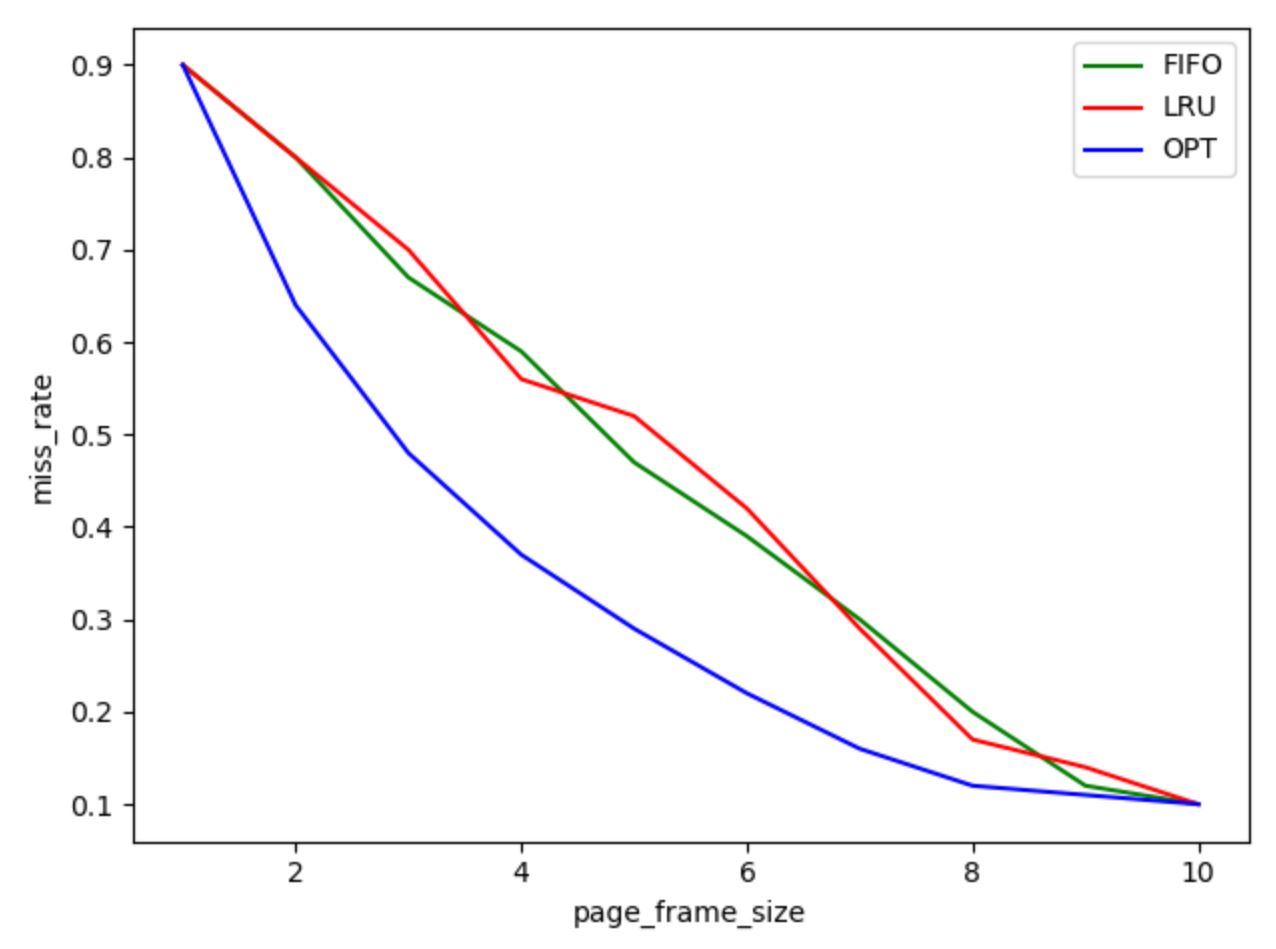
1. 陷入内核态，进行现场保护
2. 确定引发缺页中断的虚拟页面（以及进程id）
3. 检查权限，如果权限不够，则杀死进程
4. 如果权限足够，则查找一个空闲页框（或者通过页面置换算法换出一个页框）来得到一个可使用的页框
5. 如果找到的可使用页框有内容，其dirty位表示其被修改过，则要先保护住该页，并将其写回磁盘
6. 等待页面完全没问题时，将该虚拟地址对应的磁盘内容写入页框，引起磁盘调用
7. 等磁盘页面全部写完后，OS接收中断并更新页表，将虚拟页面映射到新写入的页框，并标记为正常状态
8. 根据（1）中的现场保护，恢复现场
9. 继续执行那条引发缺页的指令。
10. 假设页面的访问存在一定的周期性循环，但周期之间会随机出现一些页面的访问。例如：  
    0,1,2...,511,431,0,1,2...511,332,0,1,2,...,511等。请思考：  
    (1) LRU、FIFO和Clock算法的效果如何？  
    如果页框数小于512，则不论采用何种算法，其效果都比较接近——因为循环访问将会导致无限缺页。中间的那次随机访问根本无法预测，每种算法对其性能应当是相同的。如果页框数大于512，那么这几种算法效果也都差不多，因为能容下512次访问的话，那就只有第一轮会疯狂缺页，后几轮访问都不会缺页了。  
    (2) 如果有500个页框，能否设计一个优于LRU、FIFO和Clock的算法？  
    针对这种循环访问的情况，我觉得应当设计一种先入后出的算法，比如0号最先进来，那应该在页表中停留的时间最长，以等待下一次循环访问到0号，获得一定的收益。
11. 假设有10个页面，n个页框。页面的访问顺序为0, 9, 8, 4, 4, 3, 6, 5, 1, 5, 0, 2, 1, 1, 1, 1, 8, 8, 5,  
    3, 9, 8, 9, 9, 6, 1, 8, 4, 6, 4, 3, 7, 1, 3, 2, 9, 8, 6, 2, 9, 2, 7, 2, 7, 8, 4, 2, 3, 0, 1, 9, 4,  
    7, 1, 5, 9, 1, 7, 3, 4, 3, 7, 1, 0, 3, 5, 9, 9, 4, 9, 6, 1, 7, 5, 9, 4, 9, 7, 3, 6, 7, 7, 4, 5, 3, 5, 3, 1, 5, 6, 1,  
    1, 9, 6, 6, 4, 0, 9, 4, 3。  
    当n在[1,10]中取值时，请编写程序实现OPT、LRU、FIFO页面置换算法，并根据页面访问顺序模拟执行，分别计算缺页数量，画出缺页数量随页框数n的变化曲线（3条线）

程序代码详见附录

缺页数量变化图：

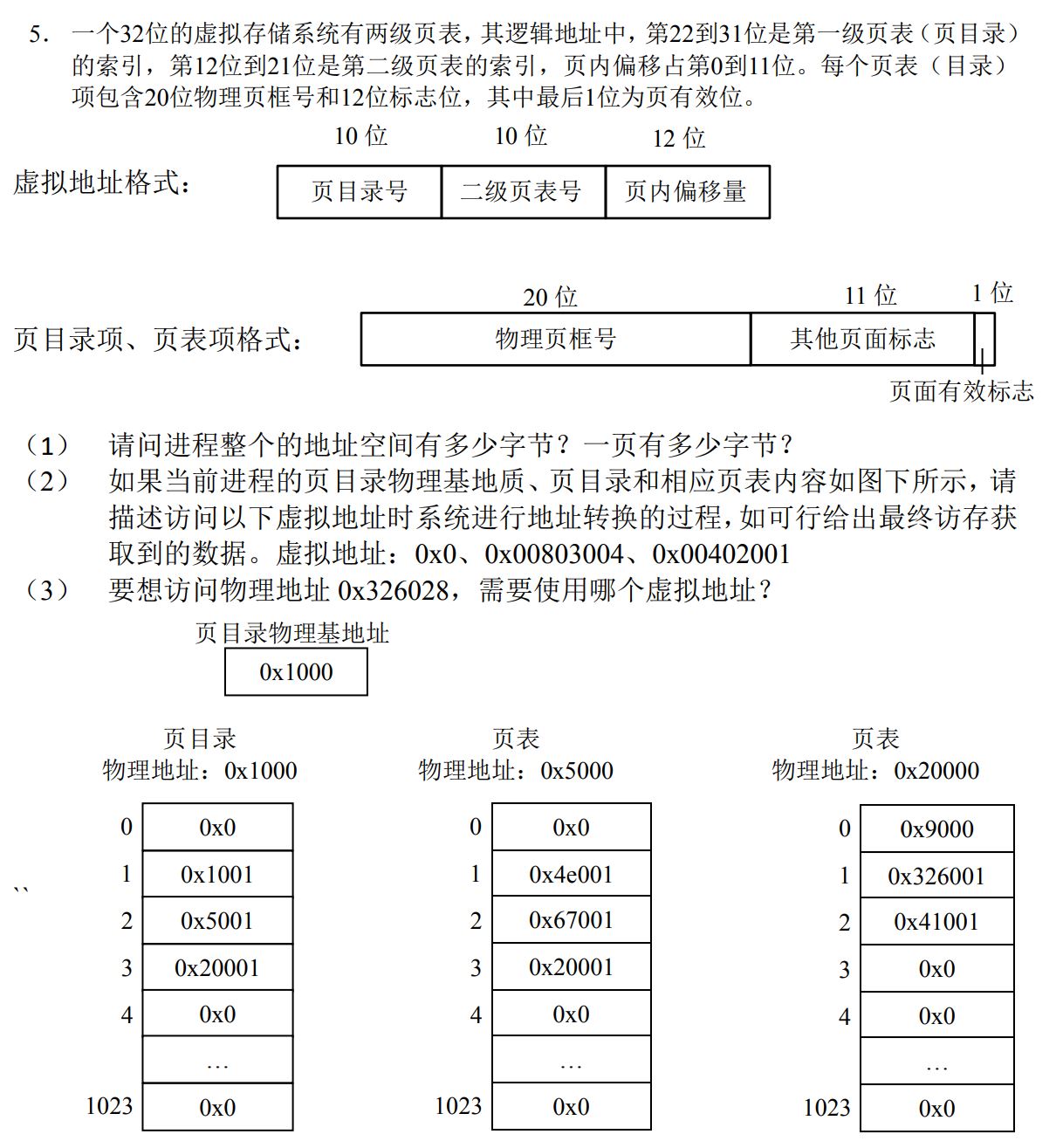


缺页率变化图：



4． 一个32位的虚拟存储系统有两级页表，其逻辑地址中，第22到31位是第一级页表，12位到21位是第二级页表，页内偏移占0到11位。一个进程的地址空间为4GB，如果从0x80000000开始映射4MB大小页表空间，请问第一级页表所占4KB空间的起始地址？并说明理由。（注意B代表字节，一个32位地址占4字节）

4GB的32位地址空间的地址为0x00000000-0xffffffff，0x80000000正处于这段地址空间的二分之一部分，因此对应二级页表的页表项也是在页表的中部。二级页表共4MB，范围为0x80000000-0x803fffff，因此对应二级页表自身的页表项（也就是一级页表的位置）就是0x80200000.从这个地址起始的4KB空间就是一级页表。



1. 进程的地址空间有4GB（2^32个字节），一页有2^12个字节=4KB。
2. 0x0对应的一级页表的页表项中，有效标志为0，因此不可被访问；  
   0x00803004对应的二进制码为0000 0000 1000 0000 0011 0000 0000 0100，取前10位为一级页表的页表项，即一级页表的页表项为第二个。找到对应的内容，发现二级页表的物理页框号为0x5（页框大小为0x1000，因此其地址为0x5000）且有效可访问；再寻找虚拟地址中中部的10位，为0000 0000 11，因此取0x5000起始的二级页表中，第3个页表项，取出的物理页框号是0x20，且有效位为1.最后，将物理页框号与虚拟地址的后12位——0000 0000 0100，进行拼接，得到物理地址为0x20004，取出的数据为0x0.  
   0x00402001的二进制码为0000 0000 0100 0000 0010 0000 0000 0001，首先取出前十位页目录项，为1，到第一个页目录项中查找，找到了二级页表位置为0x1，有效。再到起始地址为0x1000的页表中，查找虚拟地址中10位——0000000010——得到对应二级页表的第二项，物理页框号为0x5，有效。拼接上虚拟地址偏移，得到访问的位置为0x5001，访出数据为0x4e001.
3. 虚拟地址应为0b 0000 0000 1100 0000 0001 0000 0010 1000

=0xc01028

附录：

实现算法部分代码：import queue  
  
  
page\_frame = 1  
page\_list = []  
fifo\_pointer = 0  
lru\_record = {}  
now\_time = 0  
  
query\_list = [0, 9, 8, 4, 4, 3, 6, 5, 1, 5, 0, 2, 1, 1, 1, 1, 8, 8, 5,  
 3, 9, 8, 9, 9, 6, 1, 8, 4, 6, 4, 3, 7, 1, 3, 2, 9, 8, 6, 2, 9, 2, 7, 2, 7, 8, 4, 2, 3, 0, 1, 9, 4,  
 7, 1, 5, 9, 1, 7, 3, 4, 3, 7, 1, 0, 3, 5, 9, 9, 4, 9, 6, 1, 7, 5, 9, 4, 9, 7, 3, 6, 7, 7, 4, 5, 3, 5, 3,  
 1, 5, 6, 1,  
 1, 9, 6, 6, 4, 0, 9, 4, 3]  
  
  
def create\_page\_list():  
 for i in range(page\_frame):  
 page\_list.append(-1)  
  
  
def clear\_page\_list():  
 for i in range(page\_frame):  
 page\_list[i] = -1  
  
  
def opt\_replace(i):  
 future\_list = query\_list[(i + 1):]  
 future\_times = []  
 for item in page\_list:  
 q = 0  
 while q < len(future\_list):  
 if future\_list[q] == item:  
 break  
 q += 1  
 future\_times.append(q) # find the one latest used  
 max\_time = max(future\_times)  
 for q in range(len(future\_times)):  
 if future\_times[q] == max\_time:  
 break # find the max subscript  
 page\_list[q] = query\_list[i]  
 return  
  
  
def fifo\_replace(i):  
 global fifo\_pointer  
 page\_list[fifo\_pointer] = query\_list[i]  
 fifo\_pointer += 1  
 fifo\_pointer %= page\_frame  
  
  
def lru\_replace(i):  
 global now\_time  
 index = min(lru\_record, key=lambda x: lru\_record[x])  
 lru\_record.pop(index)  
 for j in range(len(page\_list)):  
 if page\_list[j] == index:  
 break  
 page\_list[j] = query\_list[i]  
 lru\_record[query\_list[i]] = now\_time  
 # print(lru\_record)  
  
  
def go\_through(func):  
 miss\_time = 0  
 total\_time = 0  
 clear\_page\_list()  
 global fifo\_pointer  
 global now\_time  
 for i in range(len(query\_list)):  
 flag = 0  
 total\_time += 1  
 now\_time += 1  
 for j in page\_list:  
 if j == query\_list[i]:  
 flag = 1 # page list hits  
 break  
 if flag == 0: # page list misses  
 miss\_time += 1  
 # print("miss! at the {} query, for page {}".format(i, query\_list[i]))  
 # print(page\_list)  
 for j in range(page\_frame):  
 if page\_list[j] == -1:  
 flag = 1  
 lru\_record[query\_list[i]] = now\_time  
 break # there's still some empty pages  
 if flag == 0:  
 func(i) # do replacement  
 else:  
 page\_list[j] = query\_list[i]  
 fifo\_pointer = (j+1) % page\_frame  
 else:  
 lru\_record[query\_list[i]] = now\_time  
 # print(lru\_record)  
 return miss\_time / total\_time  
  
  
def main():  
 create\_page\_list()  
 global page\_frame  
 global page\_list  
 miss\_rate\_list = []  
 for page\_frame in range(1, 11):  
 page\_list = []  
 create\_page\_list()  
 clear\_page\_list()  
 miss\_rate = go\_through(opt\_replace) # choose replace algorithm  
 miss\_rate\_list.append(miss\_rate)  
 print("fifo:", miss\_rate\_list)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

绘图部分代码：import numpy as np  
import matplotlib.pyplot as plt  
  
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])  
y1 = np.array([0.9, 0.8, 0.67, 0.59, 0.47, 0.39, 0.3, 0.2, 0.12, 0.1])  
y2 = np.array([0.9, 0.8, 0.7, 0.56, 0.52, 0.42, 0.29, 0.17, 0.14, 0.1])  
y3 = np.array([0.9, 0.64, 0.48, 0.37, 0.29, 0.22, 0.16, 0.12, 0.11, 0.1])  
  
plt.xlabel("page\_frame\_size")  
plt.ylabel("miss\_rate")  
plt.plot(x, y1, color="green")  
plt.plot(x, y2, color="red")  
plt.plot(x, y3, color="blue")  
plt.legend(["FIFO", "LRU", "OPT"])  
plt.show()  
plt.imsave("img.png")