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。

2． 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?

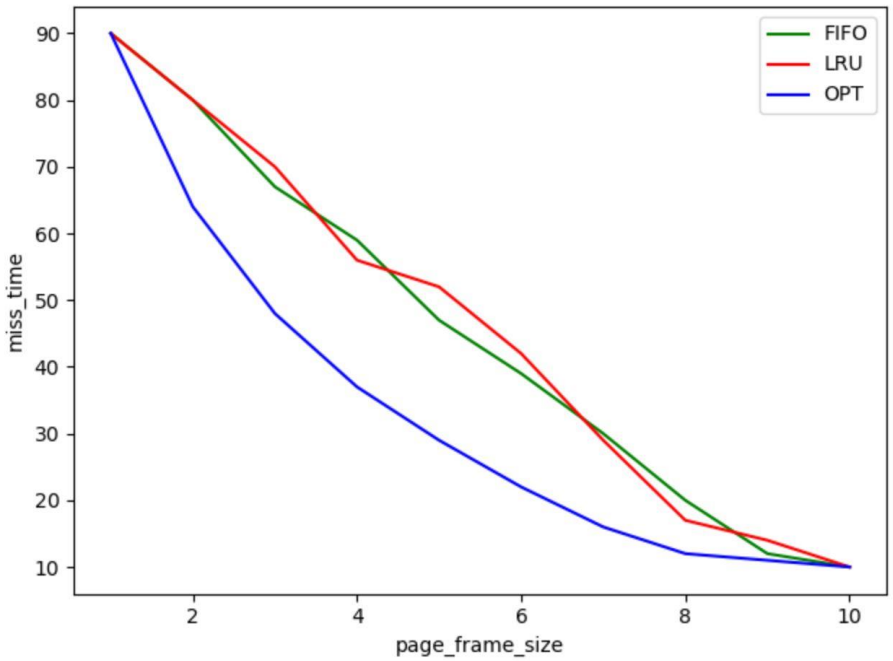
针对这种循环访问的情况,我觉得应当设计一种先入后出的算法,比如 0 号最先进来,那应该在页表中停留的时间最长，以等待下一次循环访问到 0 号，获得一定的收益。

3．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。

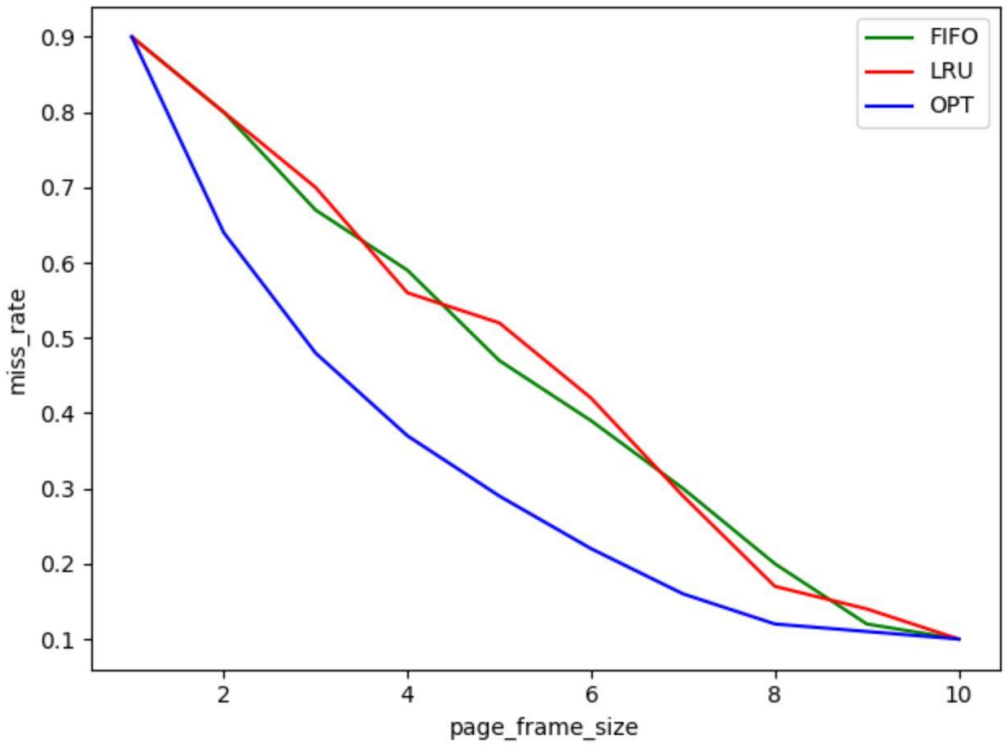
当 n 在[1,10]中取值时，请编写程序实现 OPT、LRU、FIFO 页面置换算法，并根据页面访问顺序模拟执行，分别计算缺页数量，画出缺页数量随页框数 n 的变化曲线（3 条线）

程序代码详见附录

缺页数量变化图:



缺页率变化图:

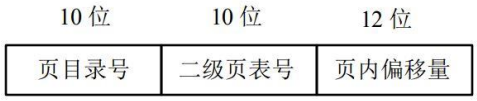


4．一个 32 位的虚拟存储系统有两级页表，其逻辑地址中，第 22 到 31 位是第一级页表，12 位到 21 位是第二级页表，页内偏移占 0 到 11 位。一个进程的地址空间为 4GB，如果从 0x80000000 开始映射 4MB 大小页表空间，请问第一级页表所占 4KB 空间的起始地址？并说明理由。（注意 B 代表字节，一个 32 位地址占 4 字节）

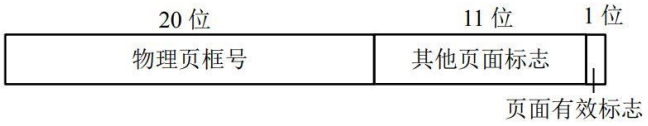
4GB 的 32 位地址空间的地址为 0x00000000-0xffffffff, 0x80000000 正处于这段地址空间的二分之一部分，因此对应二级页表的页表项也是在页表的中部。二级页表共 4MB，范围为 0x80000000-0x803fffff，因此对应二级页表自身的页表项（也就是一级页表的位置）就是 0x80200000.从这个地址起始的 4KB 空间就是一级页表。

5. 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（页目录）的索引，第12位到21位是第二级页表的索引，页内偏移占第0到11位。每个页表（目录）项包含20位物理页框号和12位标志位，其中最后1位为页有效位。

虚拟地址格式：



页目录项、页表项格式：



（1） 请问进程整个的地址空间有多少字节？一页有多少字节？

（2） 如果当前进程的页目录物理基地质、页目录和相应页表内容如图下所示，请描述访问以下虚拟地址时系统进行地址转换的过程，如可行给出最终访存获取到的数据。虚拟地址：0x0、0x00803004、0x00402001

（3） 要想访问物理地址 0x326028，需要使用哪个虚拟地址？

页目录物理基地址

0x1000

页目录
物理地址：0x1000

0	0x0
1	0x1001
2	0x5001
3	0x20001
4	0x0
	…
1023	0x0

页表
物理地址：0x5000

0	0x0
1	0x4e001
2	0x67001
3	0x20001
4	0x0
	…
1023	0x0

页表
物理地址：0x20000

0	0x9000
1	0x326001
2	0x41001
3	0x0
4	0x0
	…
1023	0x0

``

(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")
```