Project 8: Designing a Virtual Memory Manager

（一）问题分析

该项目需要模拟一个虚拟内存管理器，实现以下功能：

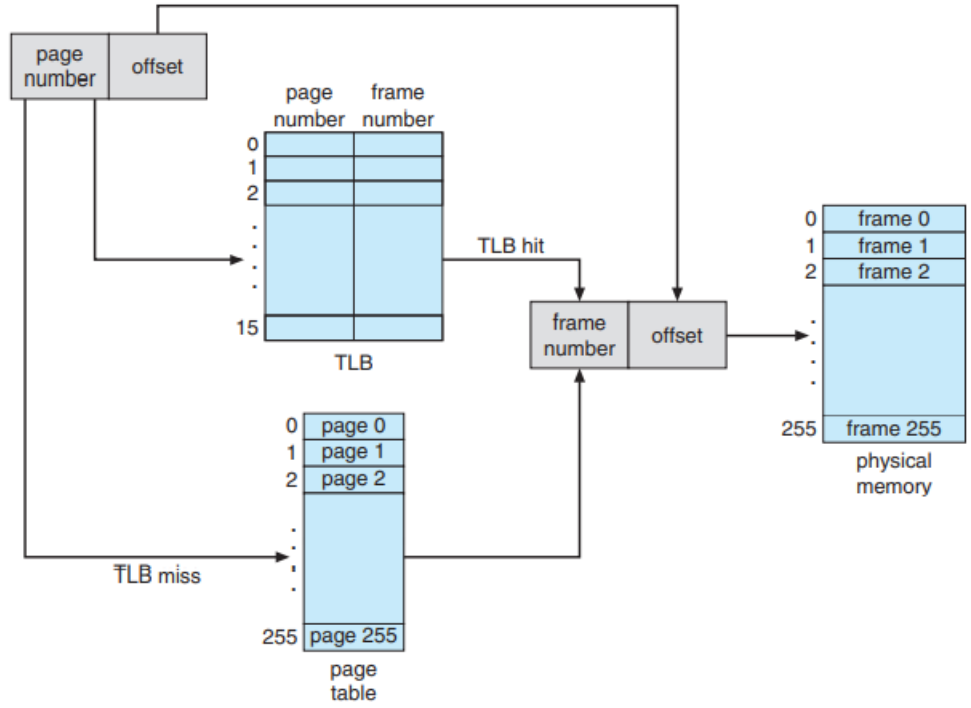
1、维护一个基于分页的虚拟内存空间。

2、通过 TLB 或页表将逻辑地址转换为物理地址。

3、请求调页（发生缺页错误时进行磁盘与物理内存间的页面置换）。

4、通过物理地址访问物理内存。

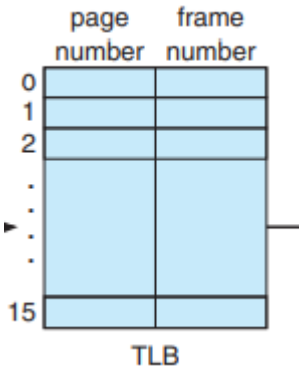
5、计算缺页错误率和 TLB 命中率。



（二）实现细节

一、数据结构

1、TLB 条目



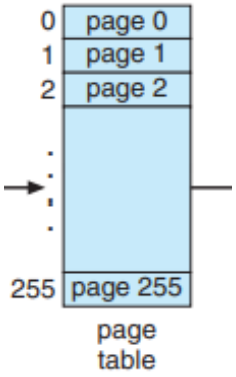
```c
typedef struct TLB_ENTRY
{
        int page_num;
        int frame_num;
        int last_used_time;
} TLB_ENTRY;
```

（1）page_num：对应页表中的页码。

（2）frame_num：对应物理内存中的帧码。

（3）last_used_time：该条目最后一次被访问的时间。

2、页表条目

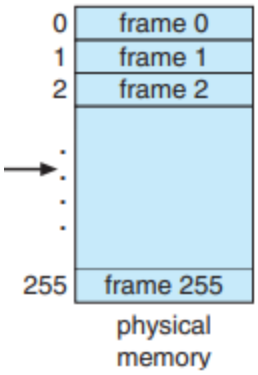



```c
typedef struct PAGE_TABLE_ENTRY
{
        int frame_num;
        int valid;              // 1 - valid; 0 - invalid
} PAGE_TABLE_ENTRY;
```

（1）frame_num：对应物理内存中的帧码。

（2）valid：有效-无效位。记录该页是否在物理内存空间内。1 表示存在，0 表示不存在或已被换出。

3、物理内存帧




```c
typedef struct FRAME
{
        char data[SIZE_OF_PAGE]; //size of frame = size of page; 1 char = 1 Byte
        int last_used_time;
} FRAME;
```

（1）data[]：物理内存的每帧存储 256 个 1 Byte 的数据，在 c 语言中 char 类型刚为 1 字节。页面大小为 256，帧大小等于页面大小。
（2）last_used_time：该帧最后一次被访问的时间。

二、全局变量
1、TLB[]：转换表缓冲区。
2、page_table[]：页表。
3、physical_memory[]：物理内存。

4、pc：记录内存引用次数。
5、page_fault：记录缺页错误次数。
6、TLB_hit：记录 TLB 命中次数。
7、time：记录当前时间。

三、函数
1、init()：初始化所有全局变量。
2、page_number_in()：通过移位操作，从 16 位逻辑地址中取出前 8 位，前 8 位为页表中的页码。
3、offset_in()：通过并运算，从 16 位逻辑地址中取出后 8 位，后 8 位为 offset。

4、TLB_replacement()：采用 LRU 算法。
（1）遍历 TLB，查找最久没有被访问的条目，即最后一次被访问时间最早的条目。因为初始化时空条目的最后访问时间被初始化为-1，所以存在空条目时会选中空条目。
（2）用本次被访问的页码和帧码更新该条目。

5、page_replacement()：采用 LRU 算法。
（1）遍历物理内存，查找最久没有被访问的帧，即最后一次被访问时间最早的帧。因为初始化时空闲帧的最后访问时间被初始化为-1，所以存在空闲帧时会选中空闲帧。
（2）将磁盘（“BACKING_STORE.bin”）中与请求调入内存的页面对应的存储区域的数据写入该帧（利用 fseek()和 fread()）。因为主函数对内存数据只做读操作，所以换入物理内存的数据不会被修改，在页面置换时不需要将物理内存中的数据换出。
（3）更新页表。页表中帧码为（1）中选中帧的条目失效，因为该帧已换入新页面。
（4）更新 TLB。TLB 中帧码为（1）中选中帧的条目被移除，即设置为空条目。
（5）返回选中帧的帧码。

6、get_frame_num()
（1）首先遍历 TLB，查找是否存在目标页码对应的条目，若存在，TLB 命中次数增加 1，更新该条目的最后访问时间，返回该条目的帧码。
（2）如果遍历结束，没有退出函数，说明 TLB 未命中。接着在页表中查找。
（3）如果目标页码对应的条目失效，缺页错误次数增加 1，进行页面置换。用 page_replacement()的返回值，即置换后访问的物理内存区域的帧码更新该条目的帧码，将有效-无效位设为有效。
（4）用目标页码和该条目的帧码更新 TLB。
（5）返回该条目的帧码。

7、access_memory()

（1）更新目标帧码对应的物理内存帧的最后访问时间。

（2）返回该帧的 offset 位置存储的数据。

四、主函数

1、执行内存引用

（1）初始化。

（2）循环读入 addresses.txt 中的逻辑地址。

（3）通过并运算取出有用的位数，即后 16 位。

（4）调用 page_number_in()函数得到页码。

（5）调用 offset_in()函数得到 offset。

（6）调用 get_frame_num()函数得到帧码。

（7）调用 access_memory()函数访问物理内存得到读取的数据。

（8）每次循环都更新内存引用次数和当前时间。

（9）每次访问内存都输出逻辑地址、物理地址和读到的数据到 output.txt 文件。

（10）循环结束后，在控制台输出物理内存帧的数量。

（11）计算缺页错误率和 TLB 命中率，并在控制台输出。

2、检查输出结果

（1）循环读入 correct.txt 和 output.txt 中的数据，检查是否一致。

（2）如果不一致，输出错误出现的行数。

（3）循环结束后，输出检查结果。

（三）运行结果

```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$ gcc -o vmm
vmm.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$ ./vmm addre
sses.txt
[256 frames]
Page-fault rate: 24.400000 %
TLB hit rate: 5.500000 %
Accept
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$ gedit vmm.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$ gcc -o vmm
vmm.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$ ./vmm addre
sses.txt
[128 frames]
Page-fault rate: 53.900000 %
TLB hit rate: 5.500000 %
Accept
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch10$
```

可以看出，当物理内存变大时，缺页错误率会变低。