Virtual Memory Manager

对于main()函数，

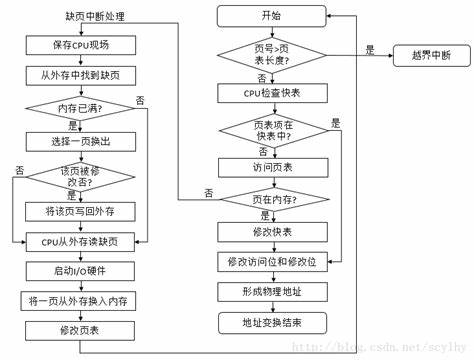
首先是从命令行读入地址文件名，定向输入输出文件等初始操作。并调用init()函数对TLB、页表、内存进行初始化操作，即将TLB置空、页表全部修改为无效、内存置空，以及将记录TLB命中次数TLB\_hit、缺页次数Page\_fault、总地址访问次数count全部置零，并定向后备存储文件BACKING\_STORE.bin，以方便后续调用。

然后就是通过while循环，从输入的地址文件中读取想要访问的逻辑地址并进行处理。首先采用位掩码和位移动的运算符直接从逻辑地址log\_add得到对应的页号page\_number和页内偏移offset，并简单判断两者是否越界；然后调用frame\_number\_access()函数根据页号获得对应的帧号frame\_number。然后还原出物理地址phy\_add。最后根据物理地址，调用memory\_access()从对应的内存位置调用存储的数据value。最后将逻辑地址、物理地址和数据值按格式输出到output文件result.txt中，方便后续验证正确性。

在循环结束后，根据过程中记录的TLB命中次数TLB\_hit、缺页次数Page\_fault、总地址访问次数count，计算缺页错误率和TLB命中率，并将其打印在控制台上。

Frame\_number\_access()函数：

依照流程图



来寻找对应的frame\_number。

首先遍历TLB，若TLB命中，则将返回命中处对应的frame\_number，并将命中位置的最近访问时间RU改成当前时间（用conut表示），以及TLB命中次数加一。

若为命中，则在页表中寻找对应的位置，若对应页表有效，则只需要返回对应的frame\_number即可，并且调用TLB\_replace()函数根据LRU的方法更新TLB。

若未命中，则需要调用page\_fault\_solve()函数对于缺页错误进行处理，缺页次数加一，待处理好之后，再次调用frame\_number\_access()函数，重新开始寻找帧号。

TLB\_replace()：遍历TLB列表，找到最近一次访问最早的列表位置（由于对TLB置空时，将其RU设置为-1，，则若是TLB存在空位置，其必是最近一次访问最早的，也即是优先使用空位置），将此时页表中对应的页号和帧号填入其中，并修改RU为当前时间count。

page\_fault\_solve()：首先根据LRU算法，与TLB列表替换方式相同，找到最近一次访问最早的帧的位置（由于对memory置空时，将其RU设置为-1，，则若是memory存在空位置，其必是最近一次访问最早的，也即是优先使用空位置），然后根据页号，从外存（backing\_store）中读取缺页，将其写回内存memory中，并将对应的帧的RU修改为当前时间count。过程中应用到了fseek和fread两种函数来读取数据。最后需修改页表，将当前页号对应的帧号修改为被缺页替换的帧号，并修改其为有效状态。此外，若还存在其他的页号对应于此帧号，需将其改为无效，表示该帧已经被修改。

对于程序的运行结果，由于给了验证集correct.txt，则使用代码为

if cmp -s "correct.txt" "result.txt"

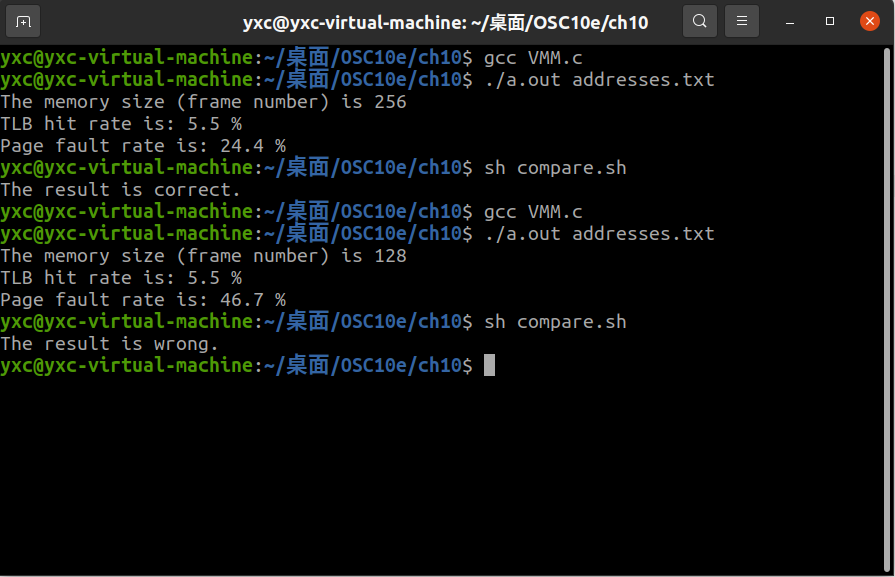
then

echo "The result is correct."

else

echo "The result is wrong."

fi

的compare.sh文件验证结果的正确性。实验结果如下。

可以看到，当采用256个页帧时结果正确。

若为了更贴近实际，将物理内存该为128个页帧，可以看到TLB的命中率几乎不变，而缺页错误率几乎翻一倍，这也与内存大小减小一半对应。