中山大学移动信息工程学院本科生实验报告

（2017 学年春季学期）

课程名称：Operating System　　任课教师：饶洋辉　　批改人(此处为 TA 填写)：

年级+班级	1501	专业（方向）	移动信息工程
学号	15352006	姓名	蔡丽芝
电话	13538489980	Email	314749816@qq.com
开始日期	2017/6/12	完成日期	2017/6/12

1. 实验目的

实现 3 种页面置换算法 FIFO,LRU,OPT

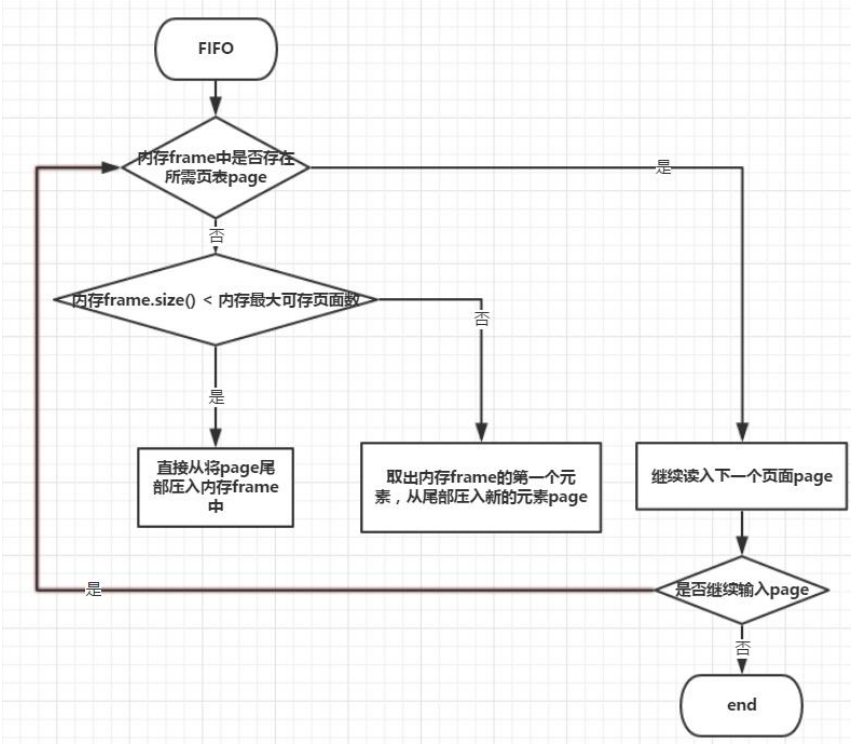
2. 实验过程

（一）算法简述

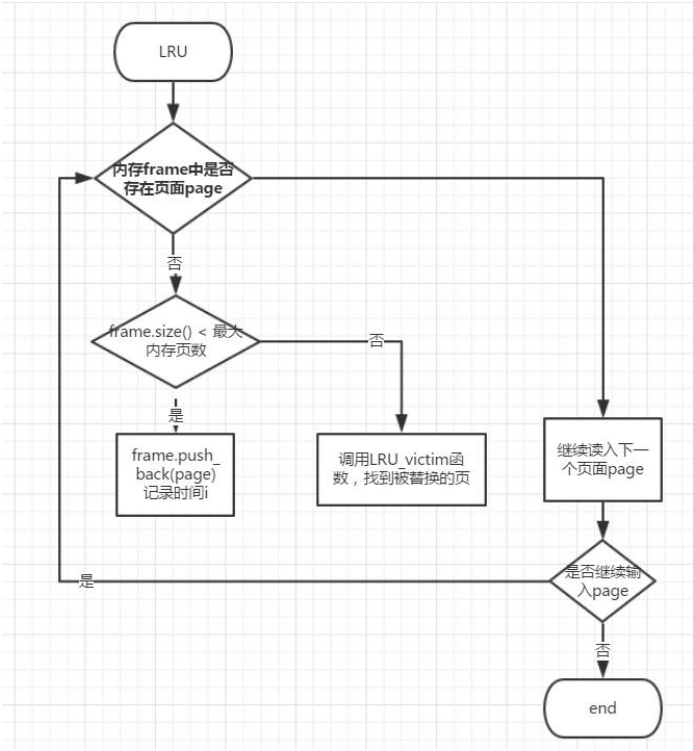
在地址映射的过程中，如果需要访问的页面不在内存中，就产生缺页中断。在缺页中断的过程中，如果操作系统中无空闲，则需要使用页面置换算法，在内存中选择一个页面移出，放入新的页面。第一种：FIFO，先进先出，淘汰最先进入主存中的一页。第二种：OPT,淘汰以后不再访问的页面，或者淘汰距离现在最长时间后再访问的页。第三种：LRU,淘汰在最近一段时间内较久未被访问的页面。

（二）伪代码或流程图

FIFO



LRU



LRU_victim 函数伪代码：

```
int LRU_victim(deque<page> &frame, int time)
{
    int min = 99999;
    for pos <- frame[0] to fram[max]
        do: if(pos.time < min)
            then min = pos.time, victim = pos;
        end
    if(min != INF)
      then victim.value = page值
           victim.time = time
           return 1;
    else
           return 0;
}
```

OPT 伪代码：

```
int OPT(string s, int a, int b)
{
    // num为需要的页面数组，frame为内存，大小为numf
    for i: 0 to a-1
      do  if(frame中存在页面num[i])
            page.time = i;
          else                              // 不存在所要需的页面
          {
            if(frame仍然有空闲页)
                直接把num[i]压入内存中即可
            else                            // 不存在空闲页
            {
                for pos: frame的一个元素 to frame的最后一个元素
                  do:  遍历数组num，找到每一个pos在未来第一次被使用的时间
                if(frame中的某些page在数组num以后找不到了)
                    调用LRU_victim函数，找到以前最早使用的页面并替换。
                else
                {
                    把pos.future_used_time最大的一项替换成新的num[i]
                }
            }

          }
}
```

（三）具体实现

声明一个结构体 page，用于记录页最近被使用的时间，将来被使用的时间点，和值 value

```
struct page
{
  int value;
  int latest_v_time;
  int future_used_t;
  page(int c, int t){value = c; latest_v_time = t; future_used_t = -1;}
};
```

Latest_v_time：表示最近被使用到的时间点

Future_used_time：表示 page 将来第一个被使用到的时间点

```
bool isExist(deque<page> &frame, int v, int time)
{
   deque<page>::iterator pos;
   for(pos = frame.begin(); pos != frame.end(); pos++)
   {
     if((*pos).value == v)                          // 在内存frame中找到一页内容值和所需的页相同
     {
        (*pos).latest_v_time = time;                // 把该请求页的最近使用使用更改为当前的时间
        return true;
     }
   }
   return false;
}
```

注释如图

```
void FIF0(string s, int a, int b)
{
  deque<page> frame;
  for(int i = 0; i <= a-1; i++)
  {
    if(!isExist(frame, num[i], i))          // 内存中不存在请求项
    {
      pageFault++;                          // 明显不存在所需的页，错误页要进行加一
      if(frame.size() < numf)               // 内存中仍然存在空闲帧
        frame.push_back(page(num[i], i));   // 直接把num[i]压入内存frame中
      else                                  // 其余情况是，内存中不存在空闲帧，这时需找出其中一页，进行替换
      {
        frame.pop_front();                  // 按照先进先出的原则，替换掉frame的第一个元素
        frame.push_back(page(num[i], i));   // 直接把num[i]压入内存frame中
      }
    }
  }
```

注释如图

```
void LRU(string s, int a, int b)
{
  deque<page> frame;
  deque<page>::iterator pos;
  for(int i = 0; i <= a-1; i++)
  {
    if(!isExist(frame, num[i], i))          // 内存中不存在请求项
    {
      pageFault++;                          // 明显不存在所需的页，错误页要进行加一
      if(frame.size() < numf)               // 内存中仍然存在空闲帧
          frame.push_back(page(num[i], i));   // 直接把num[i]压入内存frame中
      else                                  // 其余情况是，内存中不存在空闲帧，这时需找出其中一页，进行替换
      {
         LRU_victim(frame, i);              // 调用LRU_victim函数，找到frame中最近一段时间最久没被访问的那一页。
      }

    }
  }
```

解释如图

```
int LRU_victim(deque<page> &frame, int time)
{
   deque<page>::iterator pos;
   deque<page>::iterator victim = frame.begin();
   int min = INF;
   /*整个循环遍历结束后，victim表示的就是最近一段时间最久没被使用的页*/
   for(pos = frame.begin(); pos != frame.end(); pos++)
   {
    if((*pos).future_used_t == -1 &&(*pos).latest_v_time < min)     // 如果当前页的时间是小于min
    {
      min = (*pos).latest_v_time;                                   // 那么就把当前页的时间赋值给min
      victim = pos;                                                 // 同时把当前页当成别替换的victim
    }
   }
```

OPT

```
if(!isExist(frame, num[i], i))
{
  pageFault++;
  if(frame.size() < numf)
    frame.push_back(page(num[i], i));
  else
  {
```

首先判断请求项是否存在于内存中，如果存在，直接只改变页的最近使用时间即可，如果不存在，将 pageFault++，如果仍然存在空闲页，直接压入。否则，找到将来最晚使用的页进行代替。

```
/*遍历frame中的所有页内容，记录每一页在该时刻以后最早被使用的时间点.*/
for(pos = frame.begin(); pos != frame.end(); pos++)
{
    (*pos).future_used_t = -1;                  // 用来判断当前页是否在以后会用到，如果是-1，则表示以后不会被用到
    for(int j = i+1; j < input.length(); j++)   // 遍历该时刻后的num数组
    {
       if((*pos).value == num[j])               // 找到被使用的时间
       {
           cout << "Sd" ;
         (*pos).future_used_t = j;              // 把当前页的future_used_t记为j
         break;
       }
    }
}
```

```
/*遍历frame的整个内容，找到能被替代的页*/
for(pos = frame.begin(); pos != frame.end(); pos++)
{
   if((*pos).future_used_t == -1)                // 表明此时存在某一项在该位置的请求后方找不到，此时调用LRU_victim函数
   {
     LRU_victim(frame, i);                      // 选择最远被使用的那一个
     break;
   }
   else                                         // 帧表所有项在该位置的请求后方都能找到，
   {
      if((*pos).future_used_t > max)            // 选择这些项中，最晚被使用到的(第一次出现的位置最靠后的那一个)
      {
         max = (*pos).future_used_t;
         victim = pos;
      }
   }
}
```

解释如注释

3. 实验结果

	smie15352006	1000	C++	Accepted	0.08sec	308 KB	5303 Bytes	2017-06-12 17:26:29

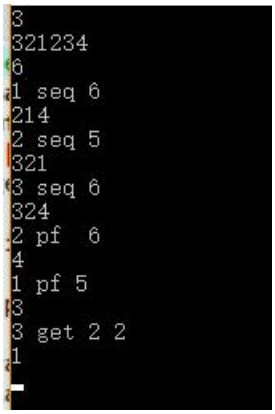
样例：　　理论输出

3　　3 2 1 2 3 4　　3 2 1 2 3 4　　3 2 1 2 3 4

321234　　FIFO: 3 2 1 1 1 4　LRU: 3 2 1 1 1 4　OPT: 3 2 1 1 1 4

6		3 2 2 2 1	3 2 2 2 2	3 2 2 2 2
1 seq 6	214	3 3 3 2	3 3 3 3	3 3 3 3
2 seq 5	321	F F F T T F	F F F T T F	F F F T T F
3 seq 6	324			
2 pf 6	4			
1 pf 5	3			
3 get 2 2	1			

程序输出



4. 回答问题

- 1. 阅读 PPT 中基本页面置换过程部分, 有哪些方法可以减少页面置换过程中的开销?
  答：可以通过脏位来降低额外的开销。给每页添加一个脏位。每当页内的任何内容被修改的时候，脏位就会被设置为 1 来表示该页被修改。如果该页的脏位被设置为 1，此时，该页如果被选择为替换页，就必须把该页写到磁盘上。然而，如果脏位没有被修改，仍为 0，就说明自从磁盘读入后该页并没有发生修改，此时磁盘上页的副本就没有必要重写，就避免了将内存页写回磁盘上的额外开销。

- 2. 对比 3 种页面置换算法。
  答：①FIFO 算法：最容易实现的页面置换算法，主要思想就是总是选择在内存停留时间最长的一页进行替换，淘汰最先调入主存的那一页，因为一般最先调入内存的页面，该页面再次被访问的可能性会比较小。该算法适用于按照线性顺序访问，其他情况下，错页率会很高，调出的页面可能是经常访问的，效率不高。
  ② LRU 算法：根据局部性的原理，一些刚刚被使用过的页面，可能会马上就用到，然而较长时间未使用的页面，肯能不会马上用到。因此，该算法的思想是，淘汰掉最近一段时间里教久未被访问的那一页。在实际中，经常会采用 LRU 算法，被认为是相当好的一种算法。
  ③ OPT 算法：淘汰是以后不再访问的那一页或者是距离现在最长时间后再访问的页。由于无法提前预知，所以该算法并不是一种可行的算法，但是可以作为衡量各种具体算法好坏的标准，具有理论意义。

- 3. 查阅并介绍其他常用的页面置换算法，并说出它们的适用情况
  答：①最不常用算法(Least Frequently Used, LFU): 当缺页的时候，置换访问次数最少的页面,由于需要利用一个数据结构来保存每一页被访问的次数,所以使用于一些访问的次数平均下来不是很多的情况。
  ②时钟置换算法：每一帧都会附带一个附加位，又称为使用位，记为 R。当发生缺页中断的时

候，如果 R 位是 0，就淘汰该页面，并把新的页面插入到这个位置，然后标准前移一个位置(逆时针方向)，如果 R 位是 1,将 R 置为 0，然后表针继续前移一个位置(逆时针方向移动),重复上述过程直到找到一个 R 为 0 的页面为止。 这种算法，开销不大，应用广。

③ 第二次机会算法(SCR)：思想基本和 FIFO 算法相同,但是该算法避免了把经常使用的页面置换出去。当缺页中断的时候，选择置换页面，检查访问位，如果是 0，就淘汰此页，若为 1，则给此页第二次机会，并将访问位变为 0，然后选择下一个 FIFO，直到发现某一位的访问位为 0，将此页置换出内存。

5. 实验感想

本次实验完成页面置换算法，我觉得对我来说，有些难度，首先是条件的判断，当出现什么字符的时候，要输出相应的结果。我的判断过程很繁琐，花了很多行代码。其次就是算法的实现了，FIFO 算法实现简单，直接经过压入换出就可以了，但是 LRU 和 OPT 的实现确实有些繁琐，一开始没有用结构体记录时间，导致整个判断过程很繁琐。后来，在写代码的过程中，老是觉得很多代码都是重复的，占据了很多行代码，所以用了函数的方式。最后，就是坑在了 opt 中，当出现一个表项在后面是无法找到的时候，应该调用 LRU 的方式，我一开始理解错了，以为是要满足内存中的所有页都无法在后面找到，才能调用 LRU。