

Project8 实验报告

余北辰 519030910245

1 实验概述

1.1 实验名称

Designing a Virtual Memory Manager

1.2 实验内容

1. 实现虚拟内存管理器的模拟
2. 给定page number = 256, page size = 256B, TLB entries = 16, frame size = 256B, frame number = 256, 完成对TLB miss和page fault的处理。
3. 统计Page fault rate和TLB hit rate。

2 实验环境

- Ubuntu 18.04.5 LTS
- Linux version 5.4.0-72-generic
- VirtualBox 6.1.18

3 实验过程与结果展示

3.1 变量定义与初始化

首先使用宏定义，定义页大小、页条目数目等实验要求的几个常量：

```
1 #define PAGE_SIZE 256
2 #define FRAME_SIZE 256
3 #define PAGE_ENTRIES_NUM 256
4 #define FRAME_ENTRIES_NUM 256
5 #define TLB_ENTRIES_NUM 16
```

分别定义页、页框和TLB的结构并声明。

由于要使用LRU算法做页面替换，因此引入 `latest_used` 用于记录：

```
1 typedef struct
2 {
3     int latest_used;
4     int page_number;
5     int frame_number;
6 } TLB;
7
8 typedef struct
9 {
10    int valid;
11    int frame_number;
12 } PAGE;
13
14 typedef struct
15 {
16    int latest_used;
17    char data[PAGE_SIZE];
18 } FRAME;
19
20 TLB tlb[TLB_ENTRIES_NUM];
21 PAGE pagetable[PAGE_ENTRIES_NUM];
22 FRAME memory[FRAME_ENTRIES_NUM];
```

引入 `Clk` 变量作为时间轴：

```
1 int clk;
```

使用 `init()` 对各结构进行初始化：

```
1 void init()
2 {
3     for (int i = 0; i < TLB_ENTRIES_NUM; ++i)
4     {
5         tlb[i].frame_number = tlb[i].page_number = tlb[i].latest_used = -1;
6     }
7     for (int i = 0; i < PAGE_ENTRIES_NUM; ++i)
```

```

8     {
9         pagetable[i].frame_number = -1;
10        pagetable[i].valid = 0;
11    }
12    for (int i = 0; i < FRAME_ENTRIES_NUM; ++i)
13    {
14        memory[i].latest_used = -1;
15    }
16 }

```

在 `main()` 函数中，定义三个文件的文件指针，以及一些其他所需的变量：

```

1    strcpy(address_filename, argv[1]);
2
3    FILE *addresses = fopen(address_filename, "r");
4    FILE *backing_store = fopen("BACKING_STORE.bin", "rb");
5    FILE *out = fopen("out.txt", "w");
6
7    int logical_address = 0;
8    int pagefault, tlb_miss;
9    double pagefault_number = 0, tlb_hit_number = 0;
10   int page_offset, page_number, frame_number;
11   int physical_address = 0;
12   int value;
13
14   init();
15   clk = 0;

```

3.2 读取逻辑地址

循环从addresses文件中，读取逻辑地址的值；

每读取一个逻辑地址的值，时钟变量clk的值加1；同时从逻辑地址中获得页偏移和页号：

```

1    fscanf(addresses, "%d", &logical_address);
2    while (!feof(addresses))
3    {
4        clk++;
5        pagefault = tlb_miss = 1;
6        page_offset = logical_address & 0xff;
7        page_number = (logical_address >> 8) & 0xff;

```

3.3 TLB hit

如果TLB中有该页号对应的条目，则直接读TLB而获得页框号，同时更新TLB和memory中 **latest_used** 的值：

```
1   for (int i = 0; i < TLB_ENTRIES_NUM; ++i)
2   {
3       if (tlb[i].page_number == page_number)
4       {
5           tlb_hit_number++;
6           tlb_miss = pagefault = 0;
7           frame_number = tlb[i].frame_number;
8           tlb[i].latest_used = clk;
9           memory[frame_number].latest_used = clk;
10          break;
11      }
12  }
```

3.4 TLB miss & Page Table hit

如果TLB miss，但是页表中能够找到该条目，则直接读页表而获得页框号，并调用 **tlb_update()** 更新TLB的条目，同时更新memory和TLB中相应条目的 **latest_used** 的值：

```
1   if (tlb_miss && pagetable[page_number].valid == 1)
2   {
3       pagefault = 0;
4       frame_number = pagetable[page_number].frame_number;
5       tlb_update(page_number, frame_number);
6       memory[frame_number].latest_used = clk;
7   }
```

tlb_update() 使用LRU算法，寻找latest_used值最小的条目做替换，实现代码如下：

```
1   void tlb_update(int page_number, int frame_number)
2   {
3       int min_time = __INT_MAX__;
4       int min_tlb_number = -1;
5       for (int i = 0; i < TLB_ENTRIES_NUM; ++i)
6       {
7           if (tlb[i].latest_used < min_time)
8           {
9               min_time = tlb[i].latest_used;
10              min_tlb_number = i;
11          }
12      }
13      tlb[min_tlb_number].frame_number = frame_number;
```

```
14     tlb[min_tlb_number].page_number = page_number;
15     tlb[min_tlb_number].latest_used = clk;
16 }
```

3.5 Page fault

如果发生了缺页错误，那么就需要进行页面替换。

首先将当前的页表中 `page_number` 对应的条目的有效位置为1；

然后根据LRU算法寻找合适的页面进行替换，调用 `fseek()` 和 `fread()` 函数，从 `BACKING_STORE.bin` 中读取相应地址所对应的值，写入物理内存中。

同时，更新memory中相应条目的 `latest_used` 值。

同时还需调用 `tlb_update()` 函数更新TLB。

```
1  else if (pagefault)
2      {
3          pagefault_number++;
4          pagetable[page_number].valid = 1;
5          int min_time = __INT_MAX__;
6          int min_frame_number = -1;
7          for (int i = 0; i < FRAME_ENTRIES_NUM; ++i)
8              {
9                  if (memory[i].latest_used < min_time)
10                     {
11                         min_time = memory[i].latest_used;
12                         min_frame_number = i;
13                     }
14              }
15          for (int i = 0; i < PAGE_ENTRIES_NUM; ++i)
16              {
17                  if (pagetable[i].frame_number == min_frame_number &&
18                      pagetable[i].valid == 1)
19                      {
20                          pagetable[i].valid = 0;
21                          break;
22                      }
23              }
24          memory[min_frame_number].latest_used = clk;
25          fseek(backing_store, page_number * PAGE_SIZE, SEEK_SET);
26          fread(memory[min_frame_number].data, sizeof(char), FRAME_SIZE,
27                backing_store);
28
29          frame_number = pagetable[page_number].frame_number =
30              min_frame_number;
```

```
28 |         tlb_update(page_number, frame_number);
29 |     }
```

3.6 其他部分

读取到页框号后，根据页框大小和页偏移计算出物理地址的值，并通过访问物理内存获得地址对应的值。

通过 `fprintf()` 函数将访问的结果写入 `out.txt` 文件中。

重复读取新的逻辑地址，继续循环。

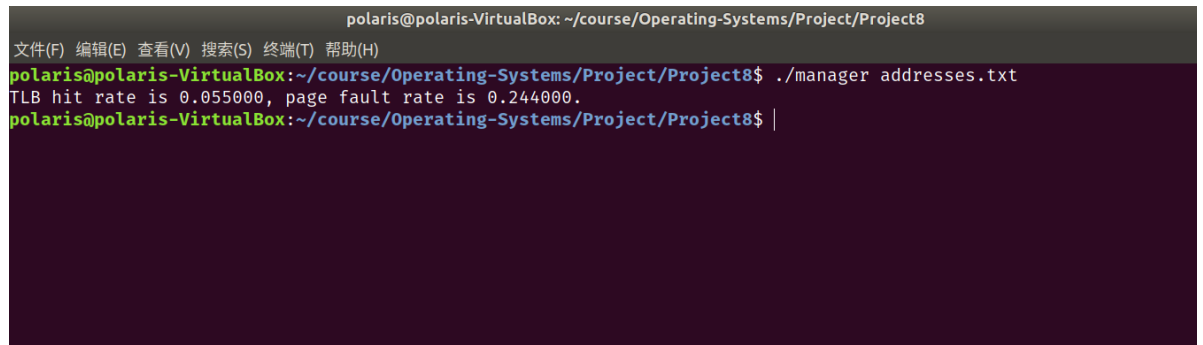
```
1 |     physical_address = frame_number * FRAME_SIZE + page_offset;
2 |     value = memory[frame_number].data[page_offset];
3 |     fprintf(out, "Virtual address: %d Physical address: %d Value: %d\n",
4 |         logical_address, physical_address, value);
5 |     fscanf(addresses, "%d", &logical_address);
```

读到EOF后跳出循环，关闭各文件，并打印TLB命中率和缺页错误率。

```
1 |     fclose(out);
2 |     fclose(addresses);
3 |     fclose(backing_store);
4 |
5 |     printf("TLB hit rate is %f, page fault rate is %f.\n", tlb_hit_number/clk,
6 |         pagefault_number/clk);
```

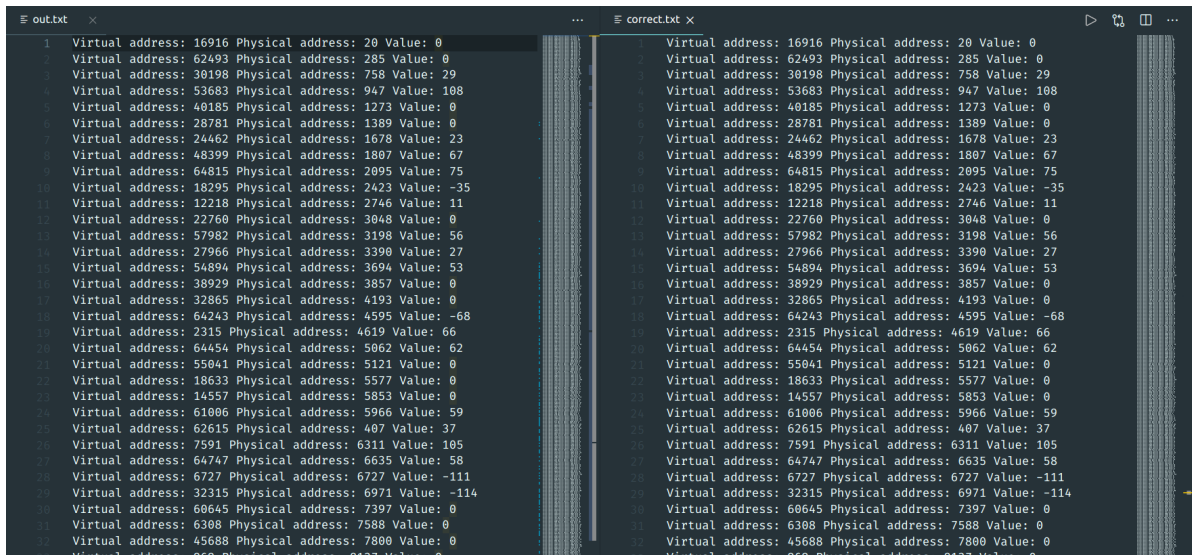
3.7 测试结果

TLB命中率和缺页错误率：



```
polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project8
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project8$ ./manager addresses.txt
TLB hit rate is 0.055000, page fault rate is 0.244000.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project8$ |
```

out.txt文件与correct.txt的对比（部分）：



```
1 Virtual address: 16916 Physical address: 20 Value: 0
2 Virtual address: 62493 Physical address: 285 Value: 0
3 Virtual address: 30108 Physical address: 758 Value: 29
4 Virtual address: 53683 Physical address: 947 Value: 108
5 Virtual address: 40185 Physical address: 1273 Value: 0
6 Virtual address: 28781 Physical address: 1389 Value: 0
7 Virtual address: 24462 Physical address: 1678 Value: 23
8 Virtual address: 48399 Physical address: 1807 Value: 67
9 Virtual address: 64815 Physical address: 2095 Value: 75
10 Virtual address: 18395 Physical address: 2423 Value: -35
11 Virtual address: 12218 Physical address: 2746 Value: 11
12 Virtual address: 22760 Physical address: 3048 Value: 0
13 Virtual address: 57982 Physical address: 3198 Value: 56
14 Virtual address: 27966 Physical address: 3390 Value: 27
15 Virtual address: 54894 Physical address: 3694 Value: 53
16 Virtual address: 38929 Physical address: 3857 Value: 0
17 Virtual address: 32865 Physical address: 4193 Value: 0
18 Virtual address: 64243 Physical address: 4595 Value: -68
19 Virtual address: 2315 Physical address: 4619 Value: 66
20 Virtual address: 64454 Physical address: 5062 Value: 62
21 Virtual address: 55041 Physical address: 5121 Value: 0
22 Virtual address: 18633 Physical address: 5577 Value: 0
23 Virtual address: 14557 Physical address: 5853 Value: 0
24 Virtual address: 61006 Physical address: 5966 Value: 59
25 Virtual address: 62615 Physical address: 407 Value: 37
26 Virtual address: 7591 Physical address: 6311 Value: 105
27 Virtual address: 64747 Physical address: 6635 Value: 58
28 Virtual address: 6727 Physical address: 6727 Value: -111
29 Virtual address: 32315 Physical address: 6971 Value: -114
30 Virtual address: 60645 Physical address: 7397 Value: 0
31 Virtual address: 6308 Physical address: 7588 Value: 0
32 Virtual address: 45688 Physical address: 7800 Value: 0
33 Virtual address: 850 Physical address: 8123 Value: 0
```

二者完全相同。

4 实验总结

1. 注意开始循环之前，要将各结构进行初始化。

5 实验参考资料

- 实验参考书籍：Operating System Concept，10th edition
- 实验源代码网址：<https://github.com/greggagne/osc10e>