计算机系统体系结构 Project8

517030910326 王孝诚

2019.12.1

实验环境

Windows 10 下使用 VMWare Workstation 15 Player 创建和运行虚拟机，虚拟机环境是 Linux 发行版 Ubuntu16.04.6 LTS。

1 虚拟内存管理

编写一个程序，将大小为 $2^{16} = 65536$ 字节的虚拟地址空间映射物理地址空间。读入的文件包含一些 32 位的地址，但我们只用低 16 位。要使用到 TLB 和页表，这里 TLB 有 16 个 entry，页表大小为 256 entry*256 byte，内存大小相同。

1.1 设计思路

1. **初始化页表和 TLB：**将页表和 TLB 的每个条目赋值-1，表示没有内容。

2. **从输入数字中提取页号和偏移：**页号：将读入的数字右移页表大小的位数，这里是 8 位，即 offset 的位数；再与 255 按位与，就获得了 8 位的页号。偏移：读入的数字直接用 255 来按位与，就获得了偏移。

3. **查询步骤：**获得页号与偏移后，首先根据页号在 TLB 中查找，若命中，则直接获得页框号，从而有物理地址在内存中查找；否则，根据页号访问页表相应页，若未发生页缺失，则同样获得页框号，得到物理地址；若发生了页缺失，就要从 backing store 文件中取得相应页并存到内存的某个页框中；最后根据 TLB 命中情况以及页缺失情况更新页表及 TLB。

4. **当物理地址空间小于虚拟地址空间：**此时查询的虚拟地址不能一一对应到物理地址，所以采用 FIFO 的页替换算法。当要更新页表而物理内存已满时，新页将把内存中最早进入的页框覆盖掉，同时页表和 TLB 中映射到被替换掉的页框的条目将被清除。这样的 FIFO 物理内存很好实现，维护一个地址，指向内存的下一处存储位置，每次内存被存入后，该地址 $addr := (addr + FRAME\_SIZE)\%MEM\_SIZE$, 即可循环使用内存空间。

1.2 核心代码解释

1. 一些常量和全局变量

```c
#define PAGE_SIZE 256
#define PAGE_ENTRIES 256
#define FRAME_SIZE 256
#define FRAME_ENTRIES 256
#define MEM_SIZE (FRAME_SIZE * FRAME_ENTRIES)
#define VIRTUAL_SIZE (PAGE_SIZE * PAGE_ENTRIES)
#define TLB_ENTRIES 16
#define OFFSET_BITS 8
#define OFFSET_MASK 255
```

```c

int page_table[PAGE_ENTRIES];
int tlb[TLB_ENTRIES][2];
char memory[MEM_SIZE];
int mem_ptr = 0;
int faultNum = 0;
int tlbHit = 0;
int tlb_ptr = 0;
```

2. 初始化

```c
for (int i=0; i<PAGE_ENTRIES; i++){
    page_table[i] = -1;
}

for (int i=0; i < TLB_ENTRIES; i++){
    tlb[i][0] = -1;
    tlb[i][1] = -1;
}

int addrNum = 0;
int physical;
int value;
char buf[10];

const char *store_file = "BACKING_STORE.bin";
const char *input_file = argv[1];
const char *out_file = "output.txt";

int store_id = open(store_file, O_RDONLY);
FILE *input_fp = fopen(input_file, "r");
FILE *output_fp = fopen(out_file, "a");
char *store_ptr = mmap(0, VIRTUAL_SIZE, PROT_READ, MAP_SHARED,
     store_id, 0);
```

3. 查询步骤

```c
while(fgets(buf, 10, input_fp)!=NULL){
    addrNum++;
    int logical_addr = atoi(buf);
    int offset = get_offset(logical_addr);
    int page_number = get_page_number(logical_addr);

    int frame_number = search_tlb(page_number);
    if(frame_number!=-1){ // tlb hit
        physical = frame_number + offset;
        value = memory[physical];
    }
    else{
        frame_number = search_page_table(page_number);
        if(frame_number!=-1){
```

```
                physical = frame_number+offset;
                update_tlb(page_number, frame_number);
                value = memory[physical];
            }
            else{  // page fault
                int page_address = page_number * PAGE_SIZE;
                memcpy(memory + mem_ptr, store_ptr + page_address, PAGE_SIZE);
                frame_number = mem_ptr;
                for(int i=0;i<PAGE_ENTRIES;i++){
                    if(page_table[i]==frame_number){
                        page_table[i] = -1;
                    }
                }
                for (int i=0; i < TLB_ENTRIES; i++){
                    if(tlb[i][1] == frame_number){
                        tlb[i][0] = -1;
                        tlb[i][1] = -1;
                    }
                }
                physical = frame_number + offset;
                value = memory[physical];
                page_table[page_number] = frame_number;
                update_tlb(page_number, frame_number);
                mem_ptr = (mem_ptr + FRAME_SIZE) % MEM_SIZE;
            }
        }

        fprintf(output_fp, "Virtual address: %d ", logical_addr);
        fprintf(output_fp, "Physical address: %d ", physical);
        fprintf(output_fp, "Value: %d\n", value);

}
```

4. 获得页号

```
int get_page_number(int virtual) {
    return (virtual >> OFFSET_BITS) & (PAGE_ENTRIES-1));
}
```

5. 获得偏移

```
int get_offset(int virtual) {
    return virtual & OFFSET_MASK;
}
```

6. 查询 TLB

```
int search_tlb(int page_number) {
    for (int i = 0; i < TLB_ENTRIES; i++) {
        if (tlb[i][0] == page_number) {
            tlbHit++;
```

```
            return tlb[i][1];
        }
    }
    return -1;
}
```

7. 查询页表

```
int search_page_table(int page_number) {
    if (page_table[page_number] == -1) {
        faultNum++;
        return -1;
    }
    return page_table[page_number];
}
```

8. 更新 TLB

```
void update_tlb(int page_number, int frame_number) {
    tlb[tlb_ptr][0] = page_number;
    tlb[tlb_ptr][1] = frame_number;
    tlb_ptr = (tlb_ptr + 1) % TLB_ENTRIES;
}
```

9. 实现 FIFO 页替换

```
memcpy(memory + mem_ptr, store_ptr + page_address, PAGE_SIZE);
mem_ptr = (mem_ptr + FRAME_SIZE) % MEM_SIZE;
```

1.3 实验结果

当物理地址空间和虚拟地址空间同样大时，由于最初内存里没有存任何页，所以会有页缺失；当所有被查询的页缺失一次后，以后将不再缺失；由于查询是随机的，所以页缺失数量略小于总的页数量；而也是因为随机访问，故 TLB 命中数不高。

```
Virtual address: 48065 Physical address: 25793 Value: 0
Virtual address: 6957 Physical address: 26413 Value: 0
Virtual address: 2301 Physical address: 35325 Value: 0
Virtual address: 7736 Physical address: 57912 Value: 0
Virtual address: 31260 Physical address: 23324 Value: 0
Virtual address: 17071 Physical address: 175 Value: -85
Virtual address: 8940 Physical address: 46572 Value: 0
Virtual address: 9929 Physical address: 44745 Value: 0
Virtual address: 45563 Physical address: 46075 Value: 126
Virtual address: 12107 Physical address: 2635 Value: -46
Number of Translated Addresses = 1000
Page Faults = 244
Page Fault Rate = 0.244
TLB Hits = 54
TLB Hit Rate = 0.054
```

Figure 1: 物理地址空间为 256*256

当物理地址空间只有逻辑地址空间一半时，由于简单的 FIFO 页替换算法导致很多的页缺失；每个虚拟地址对应的物理地址也不再与之前的相同，但值是相同的。

```
Virtual address: 48065 Physical address: 18113 Value: 0
Virtual address: 6957 Physical address: 27693 Value: 0
Virtual address: 2301 Physical address: 21245 Value: 0
Virtual address: 7736 Physical address: 13112 Value: 0
Virtual address: 31260 Physical address: 5148 Value: 0
Virtual address: 17071 Physical address: 5551 Value: -85
Virtual address: 8940 Physical address: 5868 Value: 0
Virtual address: 9929 Physical address: 6089 Value: 0
Virtual address: 45563 Physical address: 6395 Value: 126
Virtual address: 12107 Physical address: 6475 Value: -46
Number of Translated Addresses = 1000
Page Faults = 538
Page Fault Rate = 0.538
TLB Hits = 54
TLB Hit Rate = 0.054
```

Figure 2: 物理地址空间为 256*128