Project 4 Virtual Memory 设计文档

中国科学院大学

吴俊亮

2020 年 12 月 7 日

1. 内存管理设计

内核使用二级页表，每一页大小为 2MB；用户进程使用三级页表，每一页大小为 4KB。页表项的数据结构按硬件规定，如下图所示。其中软件位的低位（PTE 的第 8 位）定义为当前页是否在 SD 卡中，若该软件位为 1，则 PPN 中存放的是当前页在 SD 卡中的扇区号。

63 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

2. 缺页处理设计

有三种情况会触发缺页异常：①PTE 的 V 位为 0，且软件位为 0；②PTE 的 V 位为 0，且软件位为 1；③PTE 的 A 位或 D 位为 0 且发生了访问或者写入。第一种情况表示当前虚地址未被分配物理页框，我们需要分配一个物理页框并建立映射；第二种情况表示当前虚地址对应的物理页框在 SD 卡中，我们需要从 SD 中读取该页框写回内存中；第三种情况表示需要软件置 A 位和 D 位，我们需要根据 cause 的值设置 A 位或 D 位。

管理物理页框的数据结构如下所示，该数据结构管理 0x51000000~0x5e000000 的物理地址，该地址由用户进程使用，包括用户进程的代码段、用户栈、内核栈以及动态访问的虚地址。valid 表示该物理页框是否在使用，pin 表示该物理页框能否被替换，pgtable 存储管理该物理页框的最后一级页表项的地址。在缺页分配时，寻找一个 valid 为 0 的页框，但不是每次都从第一个物理页框开始搜索，而是从上一次分配的物理页框的后一个物理页框开始搜索以提高效率。

```
typedef struct user_page{
    uint8_t valid;
    uint8_t pin;
    PTE* pgtable;
```

```
}user_page_t;
```

3. C-Core 设计

用户进程使用的地址空间为 0x51000000~0x5e000000，附加 SD 中存储的 256 个物理页框。swap 操作直接在内核中阻塞完成，没有使用专门的进程。页替换算法使用 NRU，优先替换未被访问、未被修改的页。该页替换算法易实现且比较接近最优的 LRU 算法，但可能需要扫描所有物理页框的 A 位和 D 位，比较耗时。

测试用例如下所示，测试时设置用户进程能使用的地址空间只有 16 个物理页框，但数组 array 占用了 64 个物理页框，会触发 swap。先向 array 数组每个元素写入数据，再读取出来验证是否正确。

```c
#include <stdio.h>
#include <sys/syscall.h>

#define LENGTH (4096 * 8)
int main()
{
    int flag = 1;
    unsigned long *array = (unsigned long *)0x20000000;
    for(int i = 0; i < LENGTH; i++){
        *(array + i) = i;
    }
    for(int i = 0; i < LENGTH; i++){
        if(*(array + i) != i){
            flag = 0;
        }
    }

    if(flag){
        sys_move_cursor(1, 1);
        printf("Success!");
    }
    else{
        sys_move_cursor(1, 1);
        printf("Failed!");
    }
}
```

4. 关键函数功能

实现 NRU 算法的函数如下所示，flag 表示当前找到的最适合替换的页的情况，寻找过程中 flag 只能递增。如果找到了未被访问，未被修改的页，直接返回该页的物理地址；如果找到了比当前 flag 记录的情况更适合替换的页，则更新返回值 pa 和 flag。

```
// select a user page to write to disk (NRU), returns pa
ptr_t selete_page()
{
    int flag = 0;
    ptr_t pa = 0;
    /*
    * 1: A = 1, D = 1
    * 2: A = 0, D = 1
    * 3: A = 1, D = 0
    * 4: A = 0, D = 0
    */
    for(int i = 0; i < USER_PAGE_NUM; i++){
        if(user_page[i].pin == 0){
            PTE* pgdir = user_page[i].pgtable;
            if(!get_attribute(*pgdir, _PAGE_ACCESSED) && !get_attribute
(*pgdir, _PAGE_DIRTY)){
                pa = num2pa(i);
                flag = 4;
                break;
            }
            else if(get_attribute(*pgdir, _PAGE_ACCESSED) && !get_attri
bute(*pgdir, _PAGE_DIRTY)){
                if(flag < 3){
                    pa = num2pa(i);
                    flag = 3;
                }
            }
            else if(!get_attribute(*pgdir, _PAGE_ACCESSED) && get_attri
bute(*pgdir, _PAGE_DIRTY)){
                if(flag < 2){
                    pa = num2pa(i);
                    flag = 2;
                }
            }
            else if(get_attribute(*pgdir, _PAGE_ACCESSED) && get_attrib
ute(*pgdir, _PAGE_DIRTY)){
                if(flag < 1){
                    pa =num2pa(i);
```

```
                    flag = 1;
                }
            }
        }
    }
    assert(pa != 0);
    return pa;
}
```

参考文献

[1] https://osblog.stephenmarz.com/ch3.2.html