

实验 3 报告

黄伟健 P14206019

练习 0：填写已有实验

本实验依赖实验 1/2。请把你做的实验 1/2 的代码填入本实验中代码中有“LAB1”，“LAB2”的注释相应部分。

练习 1：给未被映射的地址映射上物理页（需要编程）

完成 do_pgfault (mm/vmmc) 函数，给未被映射的地址映射上物理页。设置访问权限的时候需要参考页面所在 VMA 的权限，同时需要注意映射物理页时需要操作内存控制结构所指定的页表，而不是内核的页表。

在 do_pgfault (mm/vmmc) 函数中，添加以下代码：

```
// try to find a pte, if pte's PT(Page Table) isn't existed, then create a PT
// (notice the 3th parameter '1')
```

```
if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
    cprintf("get_pte in do_pgfault failed\n");
    goto failed;
}
```

if (*ptep == 0) { // if the phy addr isn't exist, then alloc a page & map the phy addr with logical addr

```
    if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
        cprintf("pgdir_alloc_page in do_pgfault failed\n");
        goto failed;
    }
}
```

else { // if this pte is a swap entry, then load data from disk to a page with phy addr

// and call page_insert to map the phy addr with logical addr

```
if(swap_init_ok) {
    struct Page *page=NULL;
    if ((ret = swap_in(mm, addr, &page)) != 0) {
        cprintf("swap_in in do_pgfault failed\n");
        goto failed;
    }
}
```

```

    }
    page_insert(mm->pgdir, page, addr, perm);
    swap_map_swappable(mm, addr, page, 1);
}
else {
    cprintf("no swap_init_ok but ptep is %x, failed\n", *ptep);
    goto failed;
}
}
}

```

make qemu 之后的结果：

```

parallels@ubuntu: ~/Mac/ucore_lab/labcodes/lab3
kern/init/init.c:72: grade_backtrace+34
ebp:0xc011fff8  eip:0xc010007f  args:0x00000000 0x00000000 0x0000ffff 0x40cf9a00
memory management: default_pmm_manager
e820map:
  memory: 0009fc00, [00000000, 0009fbff], type = 1.
  memory: 00000400, [0009fc00, 0009ffff], type = 2.
  memory: 00010000, [000f0000, 000fffff], type = 2.
  memory: 07efe000, [00100000, 07ffdfbf], type = 1.
  memory: 00002000, [07ffe000, 07ffffff], type = 2.
  memory: 00040000, [fffc0000, ffffffff], type = 2.
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
PDE(0e0) c0000000-f8000000 38000000 urw
|-- PTE(38000) c0000000-f8000000 38000000 -rw
PDE(001) fac00000-fb000000 00400000 -rw
|-- PTE(000e0) fafb0000-fafec000 000e0000 urw
|-- PTE(00001) fafeb000-fafec000 00001000 -rw
----- END -----
check_vma_struct() succeeded!
page fault at 0x00000100: K/W [no page found].
page fault at 0x00000100: K/W [no page found].
page fault at 0x00000100: K/W [no page found].

```

由结果可见，通过 check_pgfault 函数的测试后，有 “check_pgfault() succeeded!” 的输出，表示练习 1 基本正确。

练习 1 比较简单 就按照注释要求 先 get_pte 获取二级页表中的入口项 ,get_pte 在 lab2 中已经完成了二级页表不存在时自动创建（第三个参数要设置为 1），所以不用担心不存在的问题。然后如果二级页表项为 0 则表明该页还未被分配，于是再通过 pgdir_alloc_page 分配一下即可。

```
make qemu
PDE(0e0) c0000000-f8000000 38000000 urw
[-- PTE(38000) c0000000-f8000000 38000000 -rw
PDE(001) fa000000-fb000000 00400000 -rw
[-- PTE(000e0) fa000000-fafe0000 000e0000 urw
[-- PTE(00001) fafeb000-fafec000 00001000 -rw
----- END -----
check_vma_struct() succeeded!
page fault at 0x00000100: K/W [no page found].
check_pgfault() succeeded!
check_vmm() succeeded.
lde 0: 10000(sectors), 'QEMU HARDDISK'.
lde 1: 262144(sectors), 'QEMU HARDDISK'.
SWAP: manager = fifo swap manager
BEGIN check_swap: count 1, total 31995
Setup Page Table for vaddr 0x1000, so alloc a page
Setup Page Table vaddr 0-4MB OVER!
set up init env for check_swap begin!
page fault at 0x00001000: K/W [no page found].
page fault at 0x00002000: K/W [no page found].
page fault at 0x00003000: K/W [no page found].
page fault at 0x00004000: K/W [no page found].
set up init env for check_swap over!
write Virt Page c in fifo_check_swap
write Virt Page a in fifo_check_swap
write Virt Page d in fifo_check_swap
write Virt Page b in fifo_check_swap
write Virt Page e in fifo_check_swap
page fault at 0x00005000: K/W [no page found].
page fault at 0x000000ae: K/R [no page found].
not valid addr ae, and can not find it in vma
trapframe at 0xc011fcf4
edi 0x00000001
esi 0x00000000
ebp 0xc011fd70
oesp 0xc011fd14
ebx 0x00007c7b
edx 0xc0303000
ecx 0x00005000
eax 0x00000092
ds 0x---0010
es 0x---0010
fs 0x---0023
gs 0x---0023
trap 0x00000000 Page Fault
err 0x00000000
eip 0xc010c46b
cs 0x---0008
flag 0x00000046 PF,ZF,IOPL=0
kernel panic at kern/trap/trap.c:185:
handle pgfault failed. invalid parameter

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>
```

完成以后，发现不再刷屏。

练习 2：补充完成基于 FIFO 的页面替换算法（需要编程）

继续完成 do_pgfault，当 ptep 不为 0 时，此时该页被换到硬盘上，ptep 存储的是硬盘上的对应位置，那么，首先检查内存交换系统是否初始化好了，如果发现初始化失败了显然此时程序就是已经出问题了，只有初始化完成了程序才能继续。

那么首先就调用 swap_in 将硬盘上得页换入到一个新分配的物理内存页中，接着用 page_insert 将这个物理内存页加入页表中，和需要访问的虚拟地址建立映射关系。注意由于注释中没有讲解 page_insert 的参数，故查看其注释已确定参数：

//page_insert - build the map of phy addr of an Page with the linear addr la

// paramemters:

// pgdir: the kernel virtual base address of PDT

// page: the Page which need to map

// la: the linear address need to map

// perm: the permission of this Page which is setted in related pte

```
// return value: always 0
```

```
//note: PT is changed, so the TLB need to be invalidate
```

接着，注释要我们调用 `swap_map_swappable`，按照相应文档，该函数是用来查询页的访问情况并间接调用相关函数，换出“不常用”的页到磁盘上。是专门为了 lab3 才在此处进行调用的，既然要调用，那么就先研究这个函数。

查看此函数会发现其作用就是将刚被访问的页添加到 FIFO 记录最近被访问页的链表中。那么首先补全这个函数，`list_add` 即可。但是查看这个函数的参数，会发现，前三个参数很明显，但是最后一个参数 `swap_in` 就有点不知道是做什么的，猜测的话可能是类似于那个 `get_pte` 中得 `create` 参数的作用，但是这里暂时没用到吧。既然暂时没用，于是，就按照 `pgdir_alloc_page` 中对其的调用，将最后一个参数填 0 了。

最后就还剩下 `_fifo_swap_out_victim` 函数了，通过 `list_prev(head)` 我们可以得到 FIFO 需要换出的页，然后将其删除，最后将 `ptr_page` 指向该页，这里涉及到又 `list_entry_t *` 找到其对应的 Page，于是可以通过 `to_struct` 来实现，但这里可以更简单地使用 `le2page`：

```
// convert list entry to page
```

```
#define le2page(le, member) \
```

```
to_struct((le), struct Page, member)
```

于是乎 `le2page(victim, pra_page_link)` 就得到了对应的 Page 的指针。

然后这里的注释是：

```
 //(1) unlink the earliest arrival page in front of pra_list_head queue
```

```
 //(2) set the addr of this page to ptr_page
```

```
make qemu
[... PTE(38000) c8000000-f2000000 38000000 -rw
PDE(001) fac00000-fb000000 00400000 -rw
[... PTE(000e0) faf00000-faf00000 000e0000 urw
[... PTE(00001) fafeb000-fafec000 00001000 -rw
..... END
check_vma_struct() succeeded!
page fault at 0x00000100: K/M [no page found].
check_pgfault() succeeded!
check_vmm() succeeded!
lde 0: 10000(sectors), 'QEMU HARDDISK'.
lde 1: 262144(sectors), 'QEMU HARDDISK'.
SWAP: manager = fifo swap manager
BEGIN check_swap: count 1, total 31995
setup Page Table for vaddr 0x1000, so alloc a page
setup Page Table vaddr 0-4MB OVER!
set up init env for check_swap begin!
page fault at 0x00001000: K/M [no page found].
page fault at 0x00002000: K/M [no page found].
page fault at 0x00003000: K/M [no page found].
page fault at 0x00004000: K/M [no page found].
set up init env for check_swap over!
write Virt Page c in fifo_check_swap
write Virt Page a in fifo_check_swap
write Virt Page d in fifo_check_swap
write Virt Page b in fifo_check_swap
write Virt Page e in fifo_check_swap
page fault at 0x00005000: K/M [no page found].
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
write Virt Page b in fifo_check_swap
write Virt Page a in fifo_check_swap
page fault at 0x00001000: K/M [no page found].
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
write Virt Page b in fifo_check_swap
page fault at 0x00002000: K/M [no page found].
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 3 with swap_page in vadr 0x2000
write Virt Page c in fifo_check_swap
page fault at 0x00003000: K/M [no page found].
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in fifo_check_swap
page fault at 0x00004000: K/M [no page found].
swap_out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
count is 0, total is 7
check_swap() succeeded!
++ setup timer interrupts
100 ticks
kbd [000]
kbd [000]
kbd [000]
kbd [000]
```

既然要实现改进的时钟算法，那么我们首先将 FIFO 算法的文件复制一份，在其框架上进行修改即可。

实现时钟算法的时候，需要注意的是，其页的引用位和修改位是由 MMU 硬件来实现的，故我们需要做的只是适时将其置为 0 即可。

那么在 mmuh 中找出对应宏：

```
/* page table/directory entry flags */
#define PTE_P 0x001 // Present
#define PTE_W 0x002 // Writeable
#define PTE_U 0x004 // User
#define PTE_PWT 0x008 // Write-Through
#define PTE_PCD 0x010 // Cache-Disable
#define PTE_A 0x020 // Accessed
#define PTE_D 0x040 // Dirty
#define PTE_PS 0x080 // Page Size
#define PTE_MBZ 0x180 // Bits must be zero
```

```
#define PTE_AVAIL 0xE00 // Available for software use
// The PTE_AVAIL bits aren't used by the kernel or interpreted by the
// hardware, so user processes are allowed to set them arbitrarily
```

其中 PTE_A 应该就是我们所需要的引用位，PTE_D 就是我们所需要的修改位。

然后由于时钟算法每次是从上一次的位置继续开始扫描，而不是从头开始，那么我们就需要同时在 mm struct 中存储链表头和上一次的位置（因为链表扫描的时候需要判断链表头，所以我们必须对其进行存储），这样我们就需要修改 mm struct 的定义。

然而，我们在定义链表头的时候，可以也为该链表头定义一个 Page，将该 Page 的 ref 设置为一个正常的页不会达到的值，比如 0，这样我们就可以识别出链表头而不用在 mm 中存储链表头。然后，为了方便 check，我们在每次访问结束之后，将整个链表（含每个页的引用位和修改位）都输出，这样可以清楚的看到每一步是否都和我们预想的一样。但是在实现的时候有一个问题，就是由于 swap 中对应的 _xx_check_swap 函数是无参数的，没法获取页目录表，而想要输出每个页的引用位和修改位就必须要有页目录表，为了不改变外面的 check 函数，故最终采取在 swap_xxc 中定义一个全局变量存储，然后本来想在 init_mm 中对其进行初始化，后来发现此时 mm 结构中的 pgdir 还没有初始化，最后，选择在页换入，也就是 _xx_map_swappable 中对 pgdir 进行设置。

本来这样我们应该很轻松的就完成了整个算法，但是，在 check 的时候，却发现怎么也不对劲，观察输出的链表可以很明显的发现虚拟地址就不对。

```
outputList();
```

```
readPage(5);
```

```
assert(pgfault_num == 5);
```

```
readPage(1);
```

```
assert(pgfault_num == 6);
```

```
readPage(2);
```

就这样的操作却直接导致：

```
set up init env for check_swap over!
```

```
0x0 -> 0x1000 1 1 -> 0x2000 1 1 -> 0x3000 1 1 -> 0x4000 1 1
```

```
read Virt Page 0x5000 in ec_check_swap
```

```
page fault at 0x00005000: K/R [no page found]
```

```
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
```

```
0x0 -> 0x5000 1 0 -> 0x2000 0 1 -> 0x3000 0 1 -> 0x4000 0 1
```

```
read Virt Page 0x1000 in ec_check_swap
```

```
page fault at 0x00001000: K/R [no page found]
```

```
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
```

```
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
```

```
0x0 -> 0x5000 1 0 -> 0x2000 0 0 -> 0x3000 0 1 -> 0x4000 0 1
```

```
read Virt Page 0x2000 in ec_check_swap
```

```
page fault at 0x00002000: K/R [no page found]
```

```
kernel panic at kern/mm/swapc:103:
```

```
assertion failed: (*ptep & PTE_P) != 0
```

```
Welcome to the kernel debug monitor!!
```

```
Type 'help' for a list of commands
```

```
K>
```

观察输出会发现，前两次的 `outputList()` 的结果都是对的，但是第三次 `outputList()` 的时候（`readPage(1)`之后），这个时候结果就错了，正确的结果应该是：

```
0x0 -> 0x5000 1 0 -> 0x1000 1 0 -> 0x3000 0 1 -> 0x4000 0 1
```

也就是说，新换入的页的虚拟地址错了。本来应该是 `0x1000`，却变成了 `0x2000`，这样，在下次 `readPage(2)` 的时候，会检测出需要换出的页正好为 `0x2000` 那一项，而此时，`0x2000` 并不存在，是已经被换出了的，无法再换出，故而发生了错误。最后我们会发现，`ucore` 的实现中，`pmmc` 中的号称会进行物理地址与虚拟地址对应的函数 `page_insert (//page_insert - build the map of phy addr of an Page with the linear addr la)` 中，并没有将 `page` 的 `praddr` 设置为线性地址 `la`。而在 `fifo` 算法 `check` 时，由于检测样例的特殊，并不会出现我们这里出现的将一个已经换出页再次换出这种事，故虽然程序错了，也没有被检测出来。

最后实现 `check`：

```
outputList();
```

```
readPage(5);
```

```
assert(pgfault_num == 5);
```

```
readPage(1);
```

```
assert(pgfault_num == 6);
```

```
readPage(3);
```

```
assert(pgfault_num == 6);
```

```
readPage(2);
```

```
assert(pgfault_num == 7);
```

```
readPage(3);
```

```
assert(pgfault_num == 7);
```

```
writePage(1);
```

```
assert(pgfault_num == 7);
```

```
readPage(4);
```

```
assert(pgfault_num == 8);
```

```
writePage(3);
```


assert(pgfault_num == 9);

writePage(4);

assert(pgfault_num == 9);

运行如下（和理论相同）：

```
set up init env for check_swap over!
0x0 -> 0x1000 1 1 -> 0x2000 1 1 -> 0x3000 1 1 -> 0x4000 1 1
read Virt Page 0x5000 in ec_check_swap
page fault at 0x00005000: K/R [no page found].
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
0x0 -> 0x5000 1 0 -> 0x2000 0 1 -> 0x3000 0 1 -> 0x4000 0 1
read Virt Page 0x1000 in ec_check_swap
page fault at 0x00001000: K/R [no page found].
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
0x0 -> 0x5000 1 0 -> 0x1000 1 0 -> 0x3000 0 1 -> 0x4000 0 1
read Virt Page 0x3000 in ec_check_swap
0x0 -> 0x5000 1 0 -> 0x1000 1 0 -> 0x3000 1 1 -> 0x4000 0 1
read Virt Page 0x2000 in ec_check_swap
page fault at 0x00002000: K/R [no page found].
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 3 with swap_page in vadr 0x2000
0x0 -> 0x5000 1 0 -> 0x1000 1 0 -> 0x3000 0 1 -> 0x2000 1 0
read Virt Page 0x3000 in ec_check_swap
0x0 -> 0x5000 1 0 -> 0x1000 1 0 -> 0x3000 0 1 -> 0x2000 1 0
write Virt Page 0x1000 in ec_check_swap
0x0 -> 0x5000 1 0 -> 0x1000 1 1 -> 0x3000 0 1 -> 0x2000 1 0
read Virt Page 0x4000 in ec_check_swap
page fault at 0x00004000: K/R [no page found].
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
0x0 -> 0x5000 0 0 -> 0x1000 0 1 -> 0x4000 1 0 -> 0x2000 1 0
write Virt Page 0x3000 in ec_check_swap
page fault at 0x00003000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
0x0 -> 0x3000 1 1 -> 0x1000 0 1 -> 0x4000 1 0 -> 0x2000 1 0
write Virt Page 0x4000 in ec_check_swap
0x0 -> 0x3000 1 1 -> 0x1000 0 1 -> 0x4000 1 1 -> 0x2000 1 0
count is 0, total is 7
check_swap() succeeded!
++ setup timer interrupts
100 ticks
```