XV6 LAZY PAGE ALLOCATION 作业报告

软工2班 3017218092 侯雨茜

一、Part One: 从sbrk()消除分配

您的第一个任务是从 `sbrk(n)` 系统调用实现中删除页面分配，该实现是 `sysproc.c` 中的 `sys_sbrk()` 函数。 `sbrk(n)` 系统调用将进程的内存大小增加n个字节，然后返回新分配的区域（即之前的大小）的开始。新 `sbrk(n)` 应该只将进程的大小 `(myproc()-> sz)` 增加n并返回之前的大小，不应该分配内存。因此应该删除对 `growproc()` 的调用（仍然需要增加进程的大小）。

修改如下：

```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  myproc()->sz += n;
//  if(growproc(n) < 0)
//    return -1;
  return addr;
}
```

返回的地址是新分配的地址空间的开头，在此处就是原来地址空间的末尾，此处都是虚地址。

启动xv6，然后在shell中键入 `echo hi` ，运行结果如下：

```
init: starting sh
$ echo hi
pid 3 sh: trap 14 err 6 on cpu 0 eip 0x112c addr 0x4004--kill proc
```

运行结果正确。

二、Part Two: Lazy allocation

修改 `trap.c` 中的代码，以通过在故障地址处映射新分配的物理内存页来响应用户空间中的页面错误，然后返回到用户空间以使进程继续执行。应该在产生" pid 3 sh：trap 14"消息的 `cprintf` 调用之前添加代码。代码无需涵盖所有极端情况和错误情况；它只需可以让sh运行echo和ls之类的简单命令。

从vm.c中的allocuvm（）窃取代码，代码修改如下：

1. 由于需要在 `trap.c` 中调用 `vm.c` 中的 `int mappages(pde_t pgdir, voidva, uint size, uint pa, int perm)` 函数，所以要在 `vm.c` 中去除 `mappages()` 原本的 `static` 关键字。

```
// Create PTEs for virtual addresses starting at va that refer to
// physical addresses starting at pa. va and size might not
// be page-aligned.
int
mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
{
  char *a, *last;
  pte_t *pte;

  a = (char*)PGROUNDDOWN((uint)va);
  last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
  for(;;){
    if((pte = walkpgdir(pgdir, a, 1)) == 0)
      return -1;
    if(*pte & PTE_P)
      panic("remap");
    *pte = pa | perm | PTE_P;
    if(a == last)
      break;
    a += PGSIZE;
    pa += PGSIZE;
  }
  return 0;
}
```

2. 在 `trap.c` 中在调用之前使用 `extern` 关键字声明 `int mappages(pde_t pgdir, voidva, uint size, uint pa, int perm)` 函数。

```
extern int mappages(pde_t *pgdir, void *va, uint size, uint pa,
int perm);
```

3. 在 `trap.c` 中的 `void trap(struct trapframe *tf)` 的defaut部分添加代码，要放在原本就存在的 `if` 模块后。

```
//PAGEBREAK: 13
```

```
  default:
    if(myproc() == 0 || (tf->cs&3) == 0){
      // In kernel, it must be our mistake.
      cprintf("unexpected trap %d from cpu %d eip %x
(cr2=0x%x)\n",
              tf->trapno, cpuid(), tf->eip, rcr2());
      panic("trap");
    }
    // In user space, assume process misbehaved.
      char *mem;
      uint a;
      a = PGROUNDDOWN(rcr2());
      uint newsz = myproc()->sz;
      for (; a < newsz; a += PGSIZE) {
          mem = kalloc();
          memset(mem, 0, PGSIZE);
          mappages(myproc()->pgdir, (char*)a, PGSIZE, V2P(mem),
PTE_W|PTE_U);
      }
      return;
```

运行结果如下：

```
init: starting sh
$ echo hi
main-loop: WARNING: I/O thread spun for 1000 iterations
hi
```

延迟分配代码导致了 `echo hi` 起作用，运行结果正确。