# 操作系统课程设计
# 实验报告


姓名：田晓宇

学号：201600301313

指导老师：杨兴强

时间：2019年1月10日

目录

## 实验目的

阅读Linux0.11源代码，了解各个部分的工作原理，清楚内存管理部分的内在逻辑，设计剧本，提取内存部分的数据，将整个内存管理部分可视化。

另外，我们在此基础上，另外完成了Linux 0.12中swap和linux 0.96中mmap部分。

## 实验环境

硬件：计算机

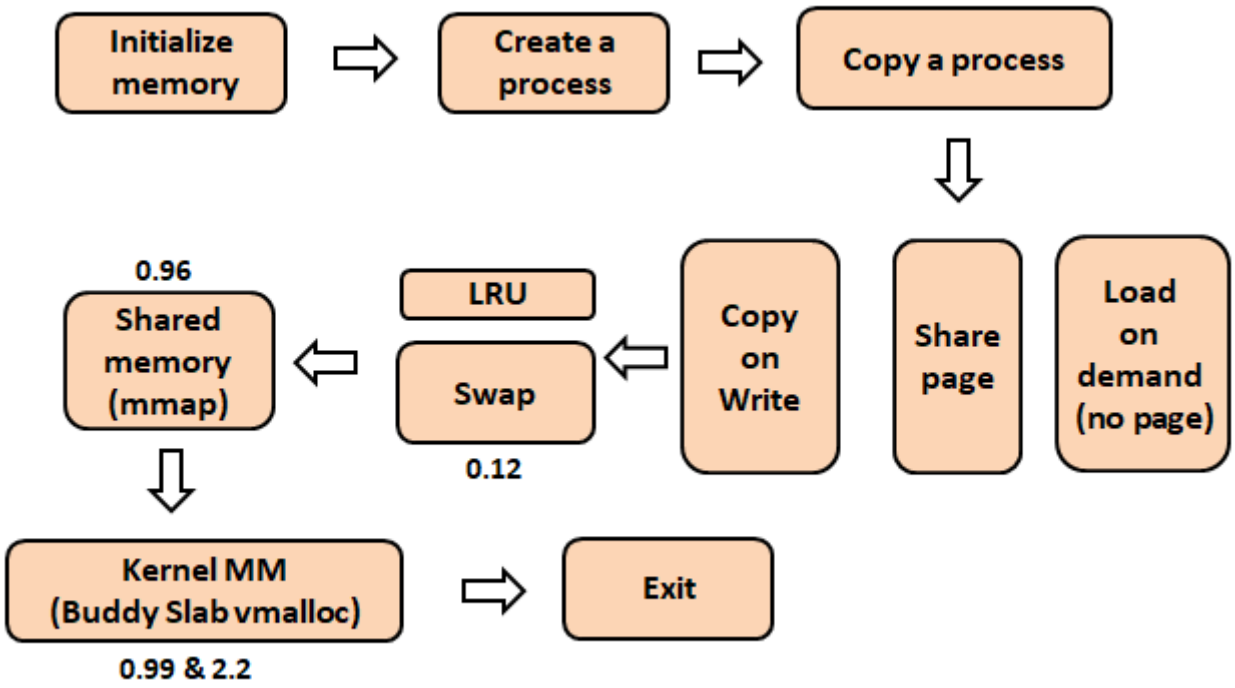
软件：bochs

cocos creator

Linux 0.11 Linux .12 Linux 0.96相关源码

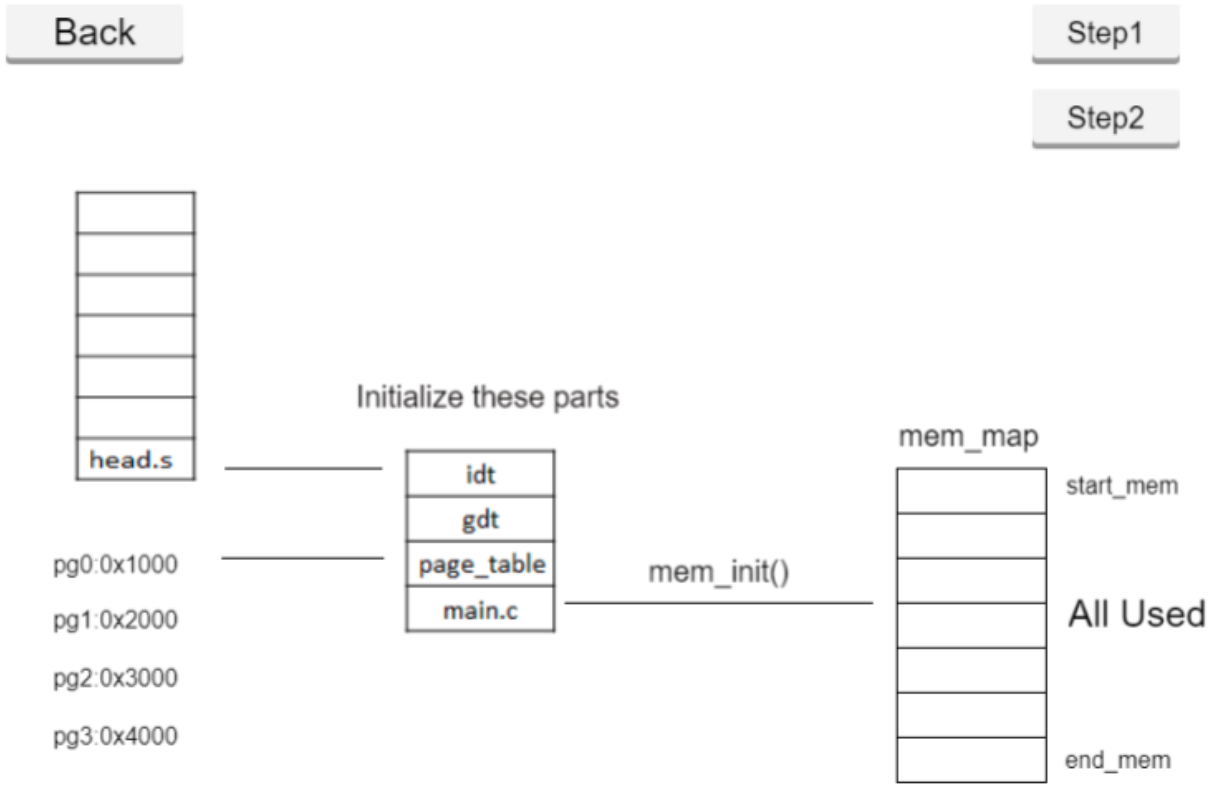
## 实验内容及结果

### 1.预想流程：

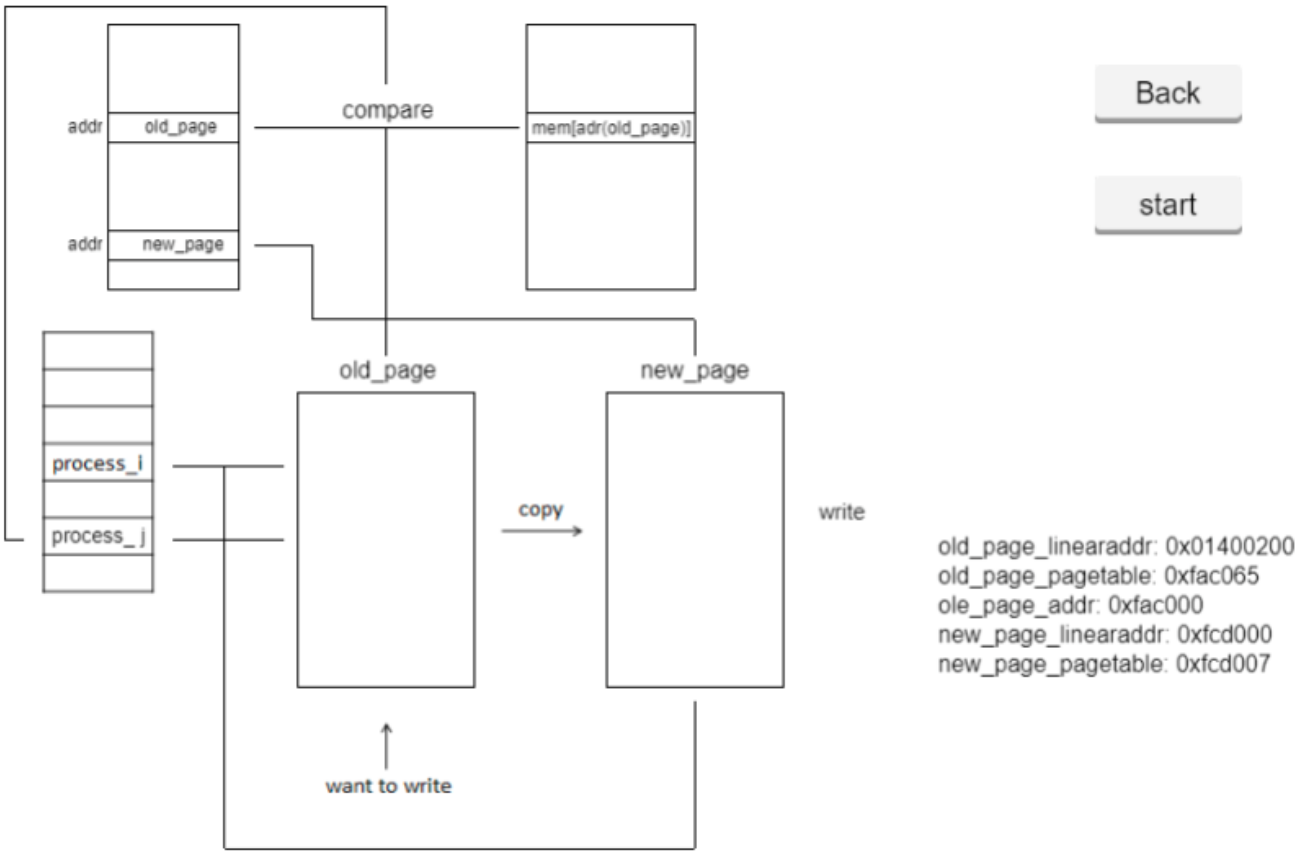


| | |
|---|---|
| Initialize memory | head.s<br>main.c<br>memory.c(mem_init()) |
| Create a process | sys.h<br>fork.c fork()<br>find_empty_process() |
| Copy a process | fork.c fork()<br>copy_process()<br>copy_mem()<br>memory.c<br>mopy_page_tables() |
| Copy on write | do_wp_page()<br>un_wp_page() |
| Share page | share_page() |
| Load on demand | execve() |
| Swap | swap.c |
| Shared memory | mmap.c: mmap()<br>munmap()<br>msync() |
| Kernel MM | kmalloc.c<br>slab.c<br>vmalloc.c |
| Exit | do_exit()<br>free_page_tables() |

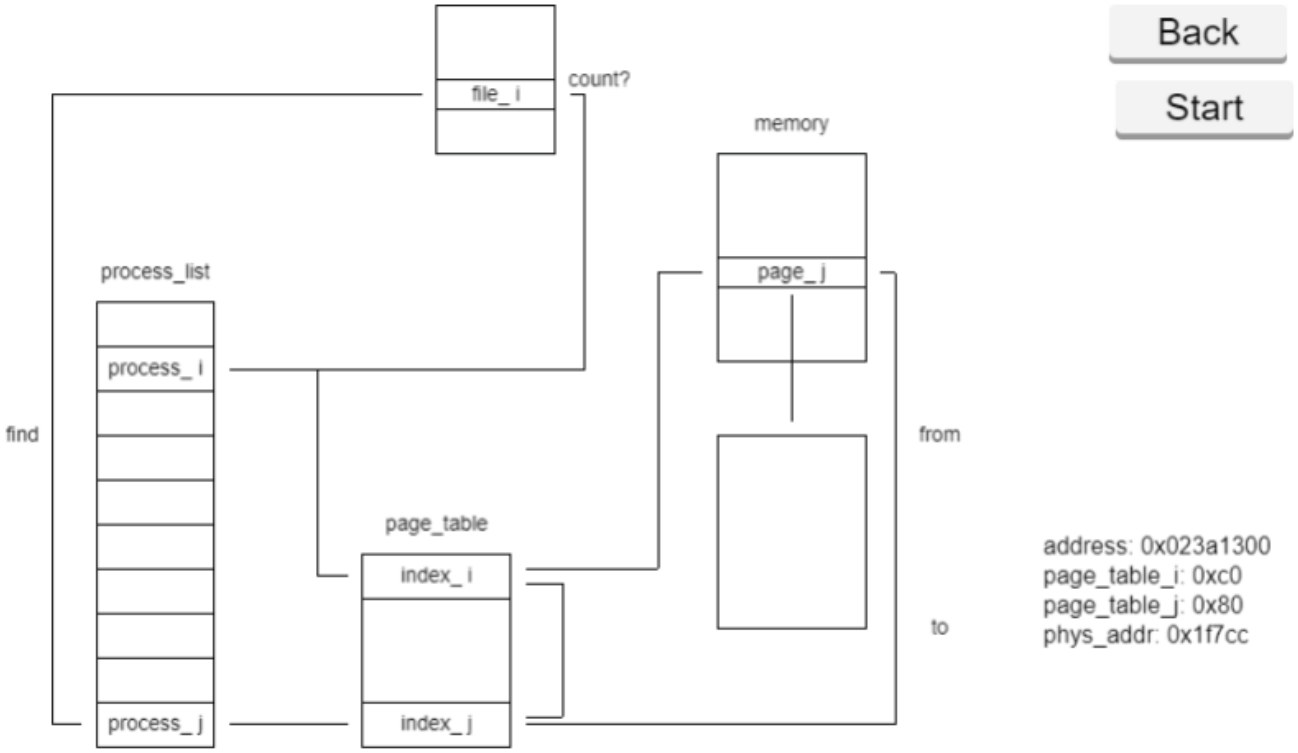
- **Initialization：**



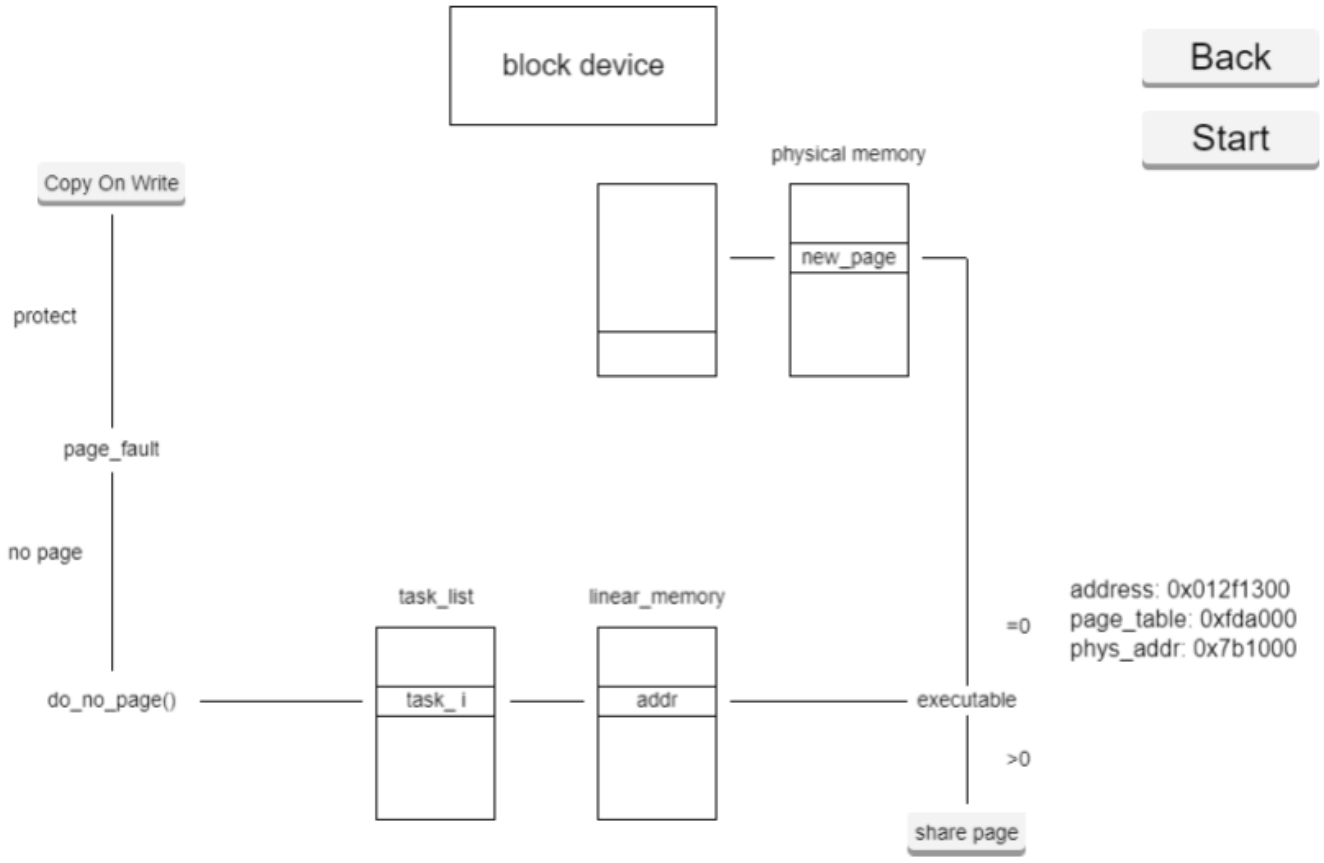
- **Copy On Write：**



- **Share Page：**

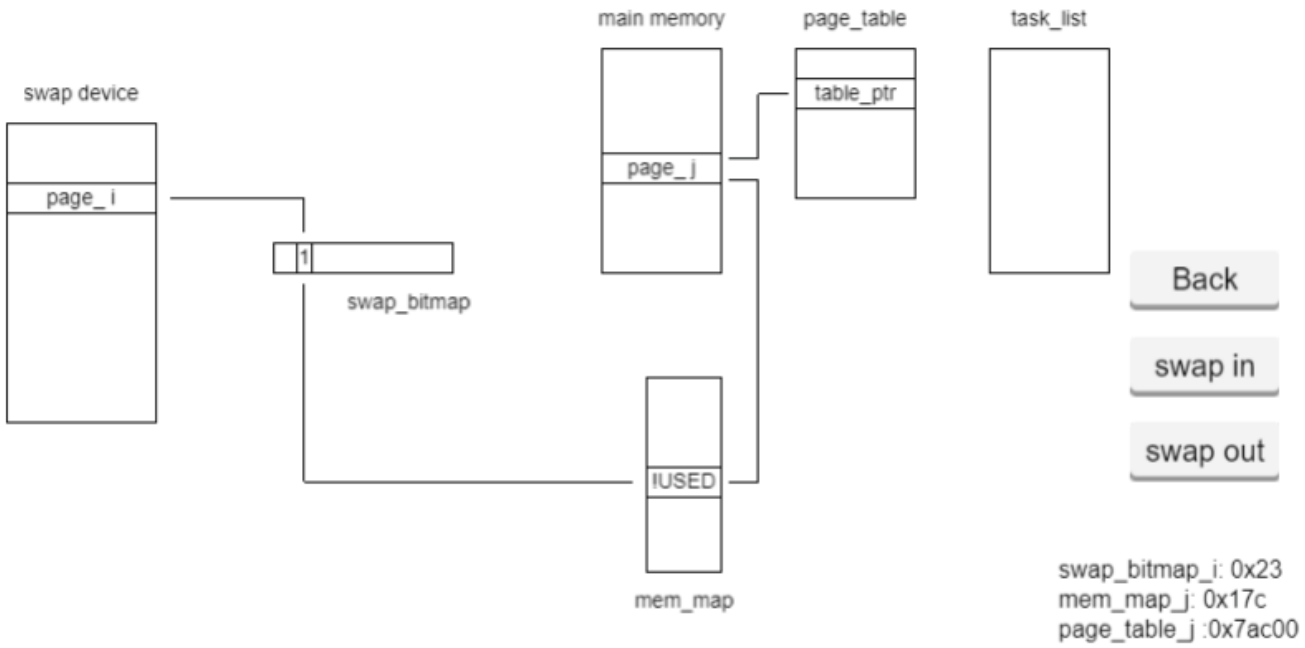

Back
Start
file_ i
count?
memory
page_ j
process_list
process_ i
find
from
page_table
index_ i
to
address: 0x023a1300
page_table_i: 0xc0
page_table_j: 0x80
phys_addr: 0x1f7cc
process_ j
index_ j

- **Page Fault:**

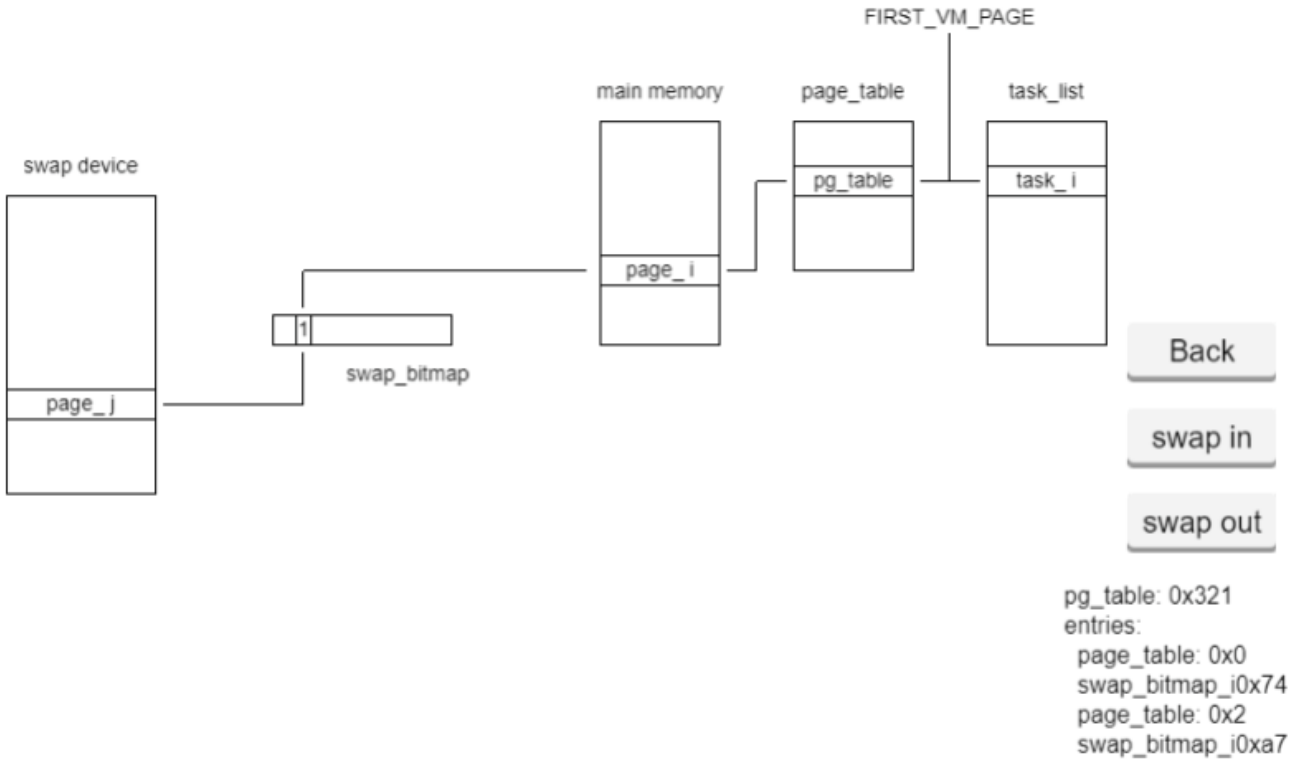

block device
Back
Start
physical memory
Copy On Write
new_page
protect
page_fault
no page
task_list
linear_memory
address: 0x012f1300
page_table: 0xfda000
phys_addr: 0x7b1000
=0
do_no_page()
task_i
addr
executable
>0
share page

- **Swap**

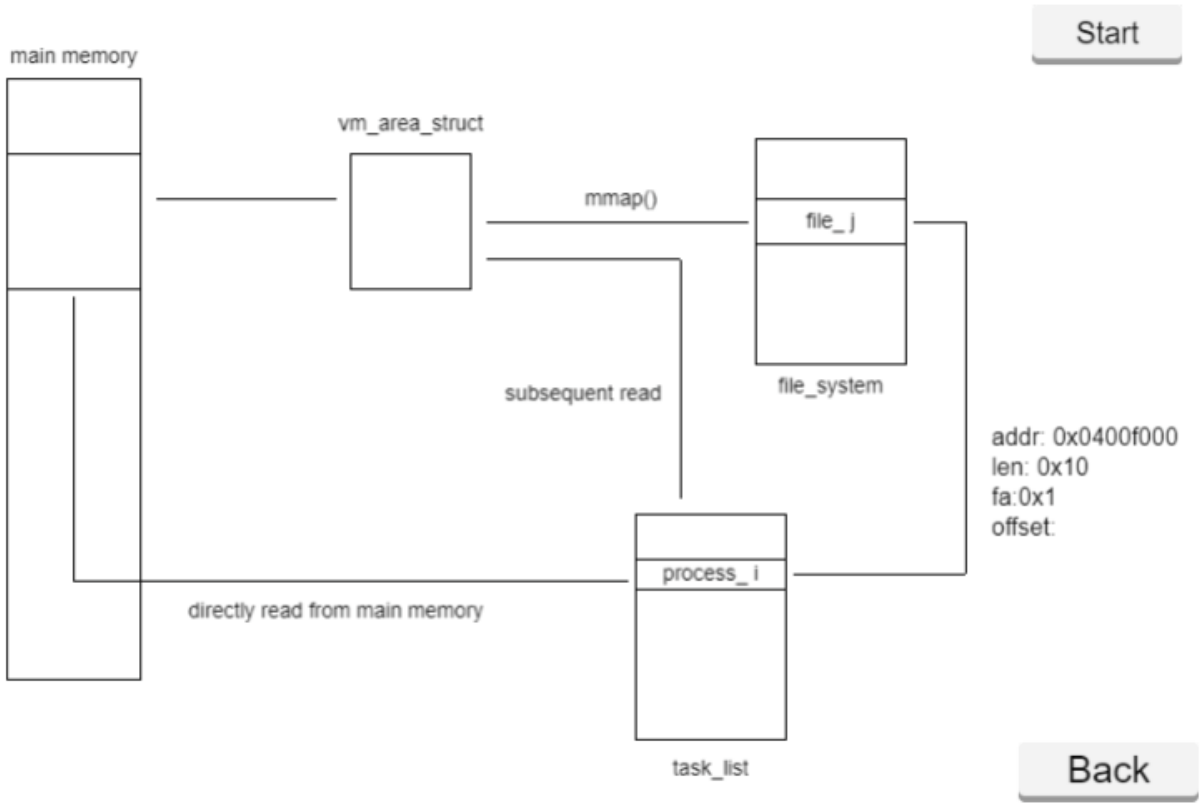
- **Swap in：**




- **Swap out：**



- **mmap：**


Start
main memory
vm_area_struct
mmap()
file_ j
file_system
subsequent read
addr: 0x0400f000
len: 0x10
fa:0x1
offset:
process_ i
directly read from main memory
task_list
Back

• 部分数据：

```
  old_page_linearaddr:'0x01400200',
  old_page_pagetable:'0xfac065',
  ole_page_addr:'0xfac000',
  new_page_linearaddr:'0xfcd000',
  new_page_pagetable:'0xfcd007',
};
export const SharePage={
  address:'0x023a1300',
  page_table_i:'0xc0',
  page_table_j:'0x80',
  phys_addr:'0x1f7cc',
};
export const Swap={
  swapIn:{
    swap_bitmap_i:'0x23',
    mem_map_j:'0x17c',
    page_table_j:'0x7ac00',
  },
  swapOut:{
    pg_table: '0x321',
    entries: [
      {
        page_table: '0x0',
        swap_bitmap_i: '0x74',
      },
      {
        page_table: '0x2',
        swap_bitmap_i: '0xa7',
      },
    ],
  },
};
```

## 工作总结及建议

**总结:**

1.我的工作是完成了Initialize、Copy on write、share page、page fault、swap、mmap六大部分的可视化工作。

2.由于时间及实现难度方面问题，我们未能完成预想中 Kernel MM该项，鉴于上一级fork & exit已经实现的比较完善，故未再做重复性工作。

3. 主要可视化代码约为2000行左右。不足之处是未想到cocos creator并不像官方手册所说的对paper.js、raphael.js等第三方绘图库支持的那么好，故只用了自带的绘图系统。

**建议：**

如果下一级同学或者其他人员想做内存管理这一部分的话，有以下建议：

1.大体包含内容可以参考我在上一部分列出的预想流程，如果你只想做课内的内容，可以把我标注出非0.11部分去掉即可；如果你想另外做些扩展，可以参考列出的linux后续版本对内存管理做出的改进和新增功能。当然，希望有同学能把我们未能完成的Kernel MM实现。

2.对于可视化工具的选择，我们选择了本认为会更好的cocos2d，而非采用上一级的processing。当然，随着工作的进行，发现这个选择极其错误。内存管理部分可视化工作根本不需要多么强大的工具或者效果，processing足够应付，使用cocos2d可能会起到意想不到的相反效果。如果你坚持使用cocos2d系列，那么你需要注意一下方面：cocos creator对第三方绘图库的支持实际上有很多坑，尽量提前踩；cocos creator的animation和graphics系统对于事件的组织可能不像最初你想象的那样。当然，如果你用cocos creator开发过相关应用或者你本身就是大神，那么请忽略我所说的建议。

3.最后一条建议，如果下一级同学刚上手时一头雾水，不知道该怎么上手，那么可稍微看一下这一条：首先，反复阅读你想要做的部分的源代码，理解每一个函数，每一条语句它想干什么，然后结合着网上相关部分的内容讲解看，看完之后再反复看代码，看的次数足够多了，感觉整个架构和思路都会比较清晰的呈现在你脑子里了，可视化工作也就顺理成章的知道怎么做了；其次，可视化应该怎么展现？展现些什么内容？所谓可视化（在内存管理这一部分），就是把抽象的代码实现的工作流程转化为容易理解的图形化的展示，无非就是把在之前说的呈现在你脑子里的架构流程给别人展示出来，当你阅读思考源代码足够多的时候自然就知道怎么做了，可能你的展现形式还要比我们上两级要更好