

2018.10.23-linux0.11-内存管理可视化-展示内容文字版报告

1.内存管理方式

- 分段
- -->线性地址空间
- -->分页
- -->二级页表

由于分段机制，以及线性地址空间，由进程实际运行时直接给出，故若不涉及段保护，则与操作系统的内存管理关系不大，因而内存管理的可视化主要涉及分页机制。

而 linux0.11 的分页机制构成为二级页表（细节略去）。

2.涉及代码

- memory.c --实际管理内存
- page.s --中断处理
- swap.c --虚拟内存

page.s 的中断处理部分为汇编代码编写，而具体功能则是调用了 memory.c 中的 do_no_page() 函数，所以没有可视化展示的空间。

swap.c 的虚拟内存功能在 linux0.11 中并未实现，所以不优先考虑可视化展示该部分。

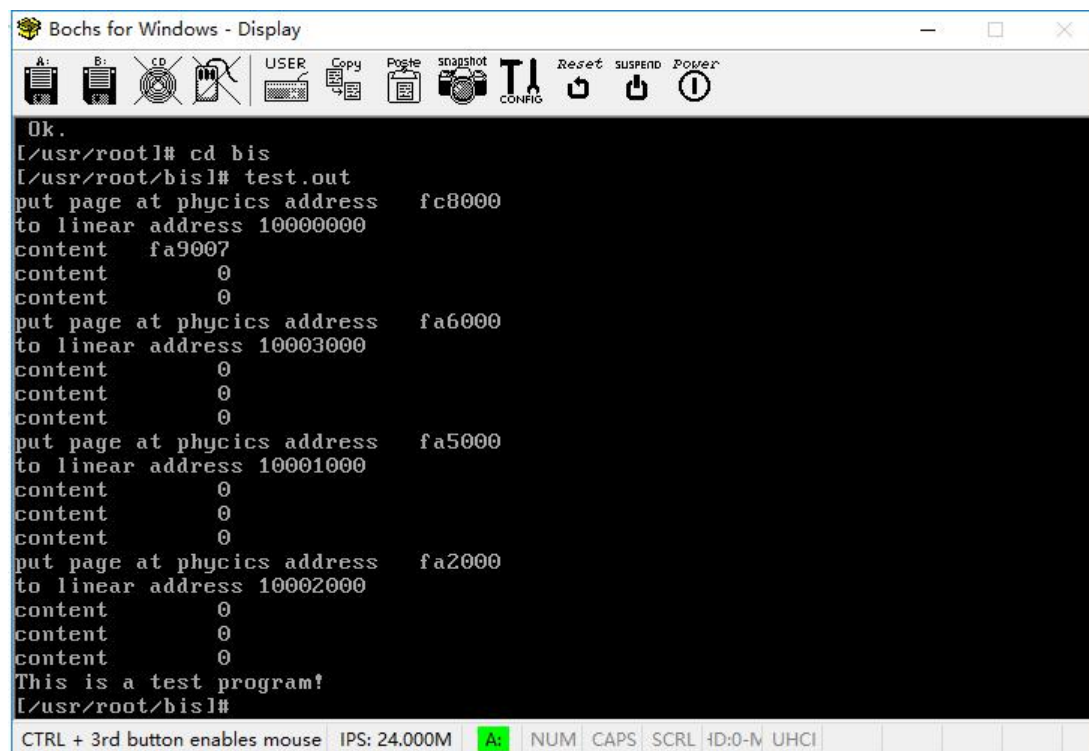
3.预期效果

- 通过执行一个最简单的程序，展示 memory.c 中函数的执行过程。
- 包括：
- 体现出函数之间的调用关系。
- 函数的逻辑功能和具体实现。

4.实际效果

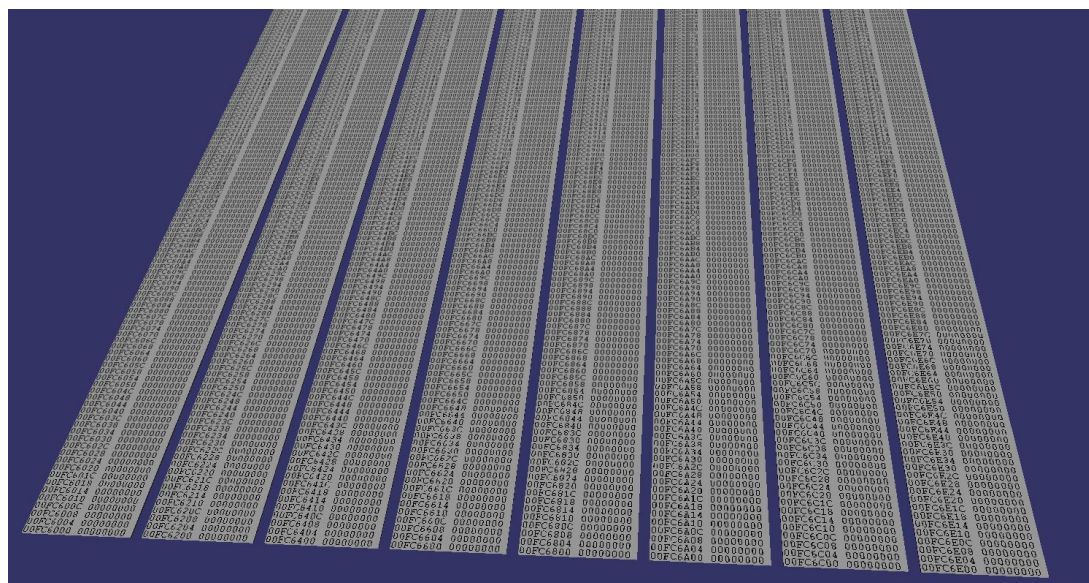
如下是一张调试时输出中间结果（在缺页处理函数中）的截图，其含义是：

在发出执行 test.out 程序的命令之后，新创建的进程的页表的物理地址是 fc8000，而此时进程欲访问的线性地址是 10000000。而在页表中的前三项的内容分别为 fa9007,0,0。而结合线性地址可以看出，此时进程真正访问到的物理地址就是 fa9000。以此类推。

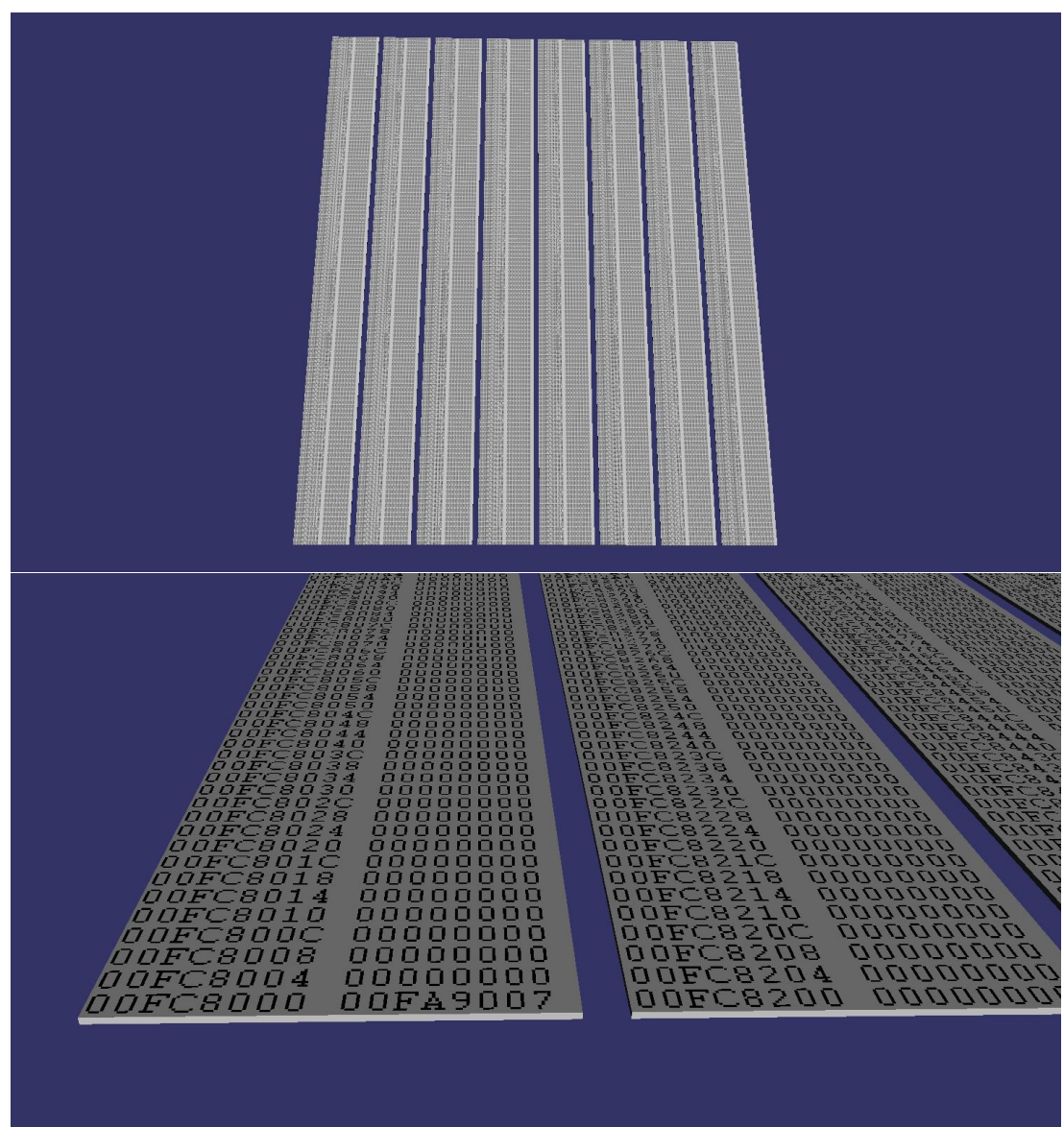


```
Ok.  
[/usr/root]# cd bis  
[/usr/root/bis]# test.out  
put page at phycics address    fc8000  
to linear address 10000000  
content    fa9007  
content    0  
content    0  
put page at phycics address    fa6000  
to linear address 10003000  
content    0  
content    0  
content    0  
put page at phycics address    fa5000  
to linear address 10001000  
content    0  
content    0  
content    0  
put page at phycics address    fa2000  
to linear address 10002000  
content    0  
content    0  
content    0  
This is a test program!  
[/usr/root/bis]#
```

以下图片是可视化的展示部分。



00	00FC2728	00000000	00FC2728	00000000	00FC2728
00	00FC271C	00000000	00FC271C	00000000	00FC271C
00	00FC2718	00000000	00FC2718	00000000	00FC2718
00	00FC2714	00000000	00FC2714	00000000	00FC2714
00	00FC2710	00000000	00FC2710	00000000	00FC2710
00	00FC270C	00000000	00FC270C	00000000	00FC270C
00	00FC2708	00000000	00FC2708	00000000	00FC2708
00	00FC2704	00000000	00FC2704	00000000	00FC2704
00	00FC2700	00000000	00FC2700	00000000	00FC2700
00	00FC26FC	00000000	00FC26FC	00000000	00FC26FC
00	00FC26F8	00000000	00FC26F8	00000000	00FC26F8
00	00FC26F4	00000000	00FC26F4	00000000	00FC26F4
00	00FC26F0	00000000	00FC26F0	00000000	00FC26F0
00	00FC26EC	00000000	00FC26EC	00000000	00FC26EC
00	00FC26E8	00000000	00FC26E8	00000000	00FC26E8
00	00FC26E4	00000000	00FC26E4	00000000	00FC26E4
00	00FC26E0	00000000	00FC26E0	00000000	00FC26E0



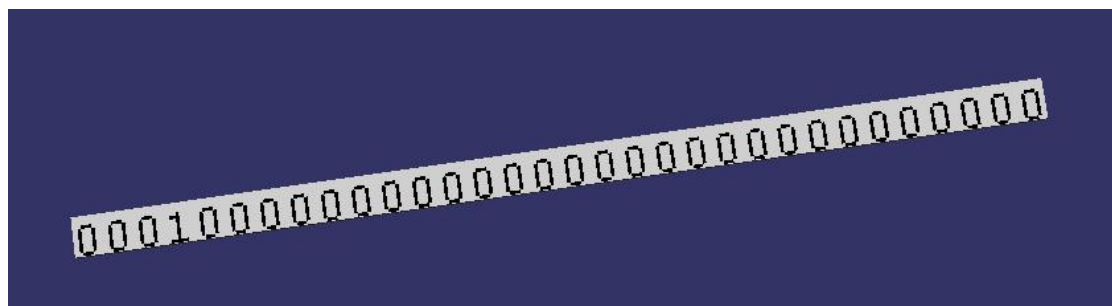
上面四幅图片，展示了内存中的四个页表（4KB）的数据（16 进制），而具体表现就是场景中浮空的板子，而上面写明的就是内存的地址（左）和内存中对于的内容（右），以下将会更加具体地展示出一次线性地址经过二级页表的转化而变成物理地址的过程。

1. 进程产生线性地址 10000000



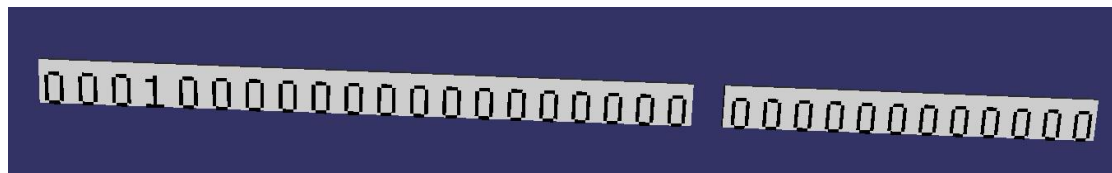
10000000

转换为 2 进制



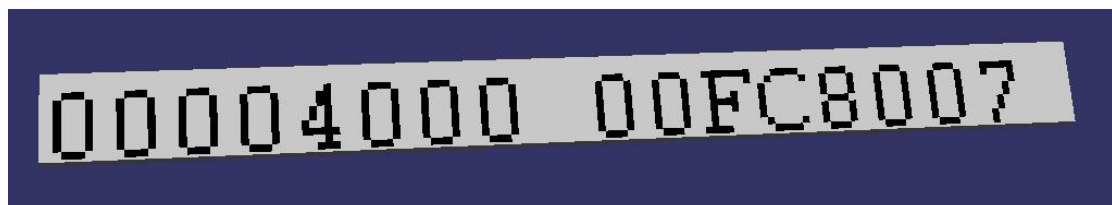
00010000000000000000000000000000

2. 取出高 20 位



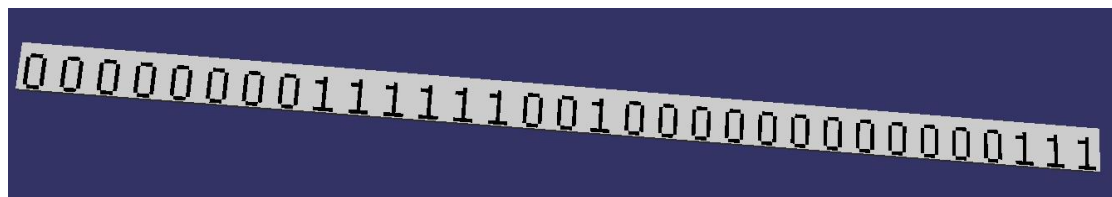
00010000000000000000

查询一级页表（linux0.11 中，一级页表的物理地址固定在 0）



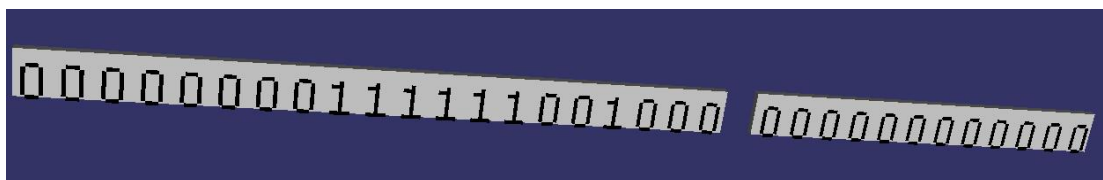
00004000 00FC8007

转换为 2 进制（页表中内容，代表了二级页表物理地址（去除标志位后））

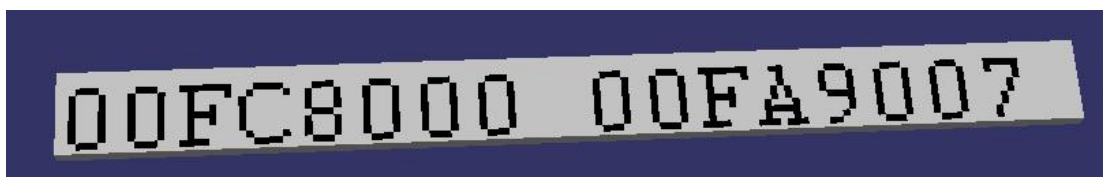


00000000111111001000000000000000

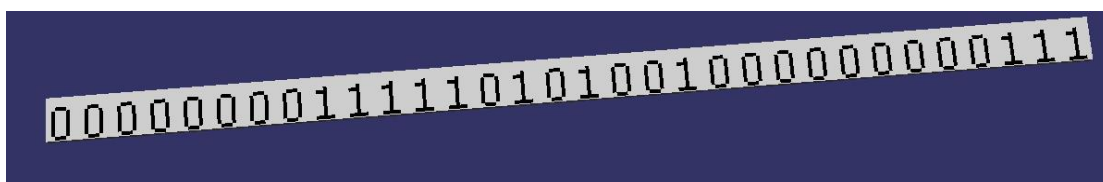
3. 再次取出高 20 位



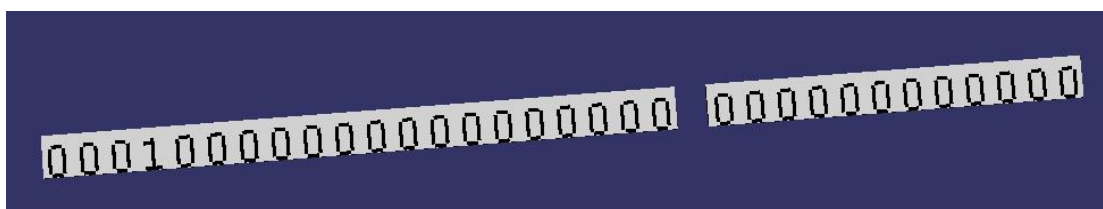
查询二级页表（此处与上面展示的 4KB 中内存相同）



转换为 2 进制



4. 取高 20 位，和线性地址的低 12 位



结合形成最终物理内存的地址（FA9000）



5. 实际访问内存



以上，即完成一次对于物理内存访问的过程（仅展示中间结果，未包含数据位的移动过程）。